N° d'ordre : 2013 EMSE 0717

THÈSE

présentée par

Mohamed Reda YAICH

pour obtenir le grade de
Docteur de l'École Nationale Supérieure des Mines de Saint-Étienne

Spécialité : Informatique

Adaptation et Conformité Sociale dans la Gestion de la Confiance – Une approche multi-agent

soutenue à Saint-Étienne, le 29 Octobre 2013

Membres du jury:

Président :	Pierre MARET	-	LaHC - Université Jean Monnet de Saint-Etienne
Rapporteurs :	Timothy NORMAN	-	King's College - University of Aberdeen
	Laurent VERCOUTER	-	LITIS - INSA de Rouen
	Mohand-Said HACID	-	LIRIS - Université Claude Bernard Lyon 1
Examinateurs :	Norra CUPPENS-BOULAHIA	-	LABSTICC - Telecom Bretagne
Directeurs de thèse :	Olivier BOISSIER	-	ENS des Mines de St-Etienne
	Gauthier PICARD	-	ENS des Mines de St-Etienne
	Philippe JAILLON	-	ENS des Mines de St-Etienne

Spécialités doctorales :	**Responsables :**
SCIENCES ET GENIE DES MATERIAUX	K. Wolski Directeur de recherche
MECANIQUE ET INGENIERIE	S. Drapier, professeur
GENIE DES PROCEDES	F. Gruy, Maître de recherche
SCIENCES DE LA TERRE	B. Guy, Directeur de recherche
SCIENCES ET GENIE DE L'ENVIRONNEMENT	D. Graillot, Directeur de recherche
MATHEMATIQUES APPLIQUEES	O. Roustant, Maître-assistant
INFORMATIQUE	O. Boissier, Professeur
IMAGE, VISION, SIGNAL	JC. Pinoli, Professeur
GENIE INDUSTRIEL	A. Dolgui, Professeur
MICROELECTRONIQUE	

EMSE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'État ou d'une HDR)

AVRIL	Stéphane	PR2	Mécanique et ingénierie	CIS
BATTON-HUBERT	Mireille	PR2	Sciences et génie de l'environnement	FAYOL
BENABEN	Patrick	PR1	Sciences et génie des matériaux	CMP
BERNACHE-ASSOLLANT	Didier	PR0	Génie des Procédés	CIS
BIGOT	Jean Pierre	MR(DR2)	Génie des Procédés	SPIN
BILAL	Essaid	DR	Sciences de la Terre	SPIN
BOISSIER	Olivier	PR1	Informatique	FAYOL
BORBELY	Andras	MR(DR2)	Sciences et génie de l'environnement	SMS
BOUCHER	Xavier	PR2	Génie Industriel	FAYOL
BRODHAG	Christian	DR	Sciences et génie de l'environnement	FAYOL
BURLAT	Patrick	PR2	Génie Industriel	FAYOL
COLLOT	Philippe	PR0	Microélectronique	CMP
COURNIL	Michel	PR0	Génie des Procédés	DIR
DARRIEULAT	Michel	IGM	Sciences et génie des matériaux	SMS
DAUZERE-PERES	Stéphane	PR1	Génie Industriel	CMP
DEBAYLE	Johan	CR	Image Vision Signal	CIS
DELAFOSSE	David	PR1	Sciences et génie des matériaux	SMS
DESRAYAUD	Christophe	PR2	Mécanique et ingénierie	SMS
DOLGUI	Alexandre	PR0	Génie Industriel	FAYOL
DRAPIER	Sylvain	PR1	Mécanique et ingénierie	SMS
FEILLET	Dominique	PR2	Génie Industriel	CMP
FOREST	Bernard	PR1	Sciences et génie des matériaux	CIS
FORMISYN	Pascal	PR0	Sciences et génie de l'environnement	DIR
FRACZKIEWICZ	Anna	DR	Sciences et génie des matériaux	SMS
GARCIA	Daniel	MR(DR2)	Génie des Procédés	SPIN
GERINGER	Jean	MA(MDC)	Sciences et génie des matériaux	CIS
GIRARDOT	Jean-jacques	MR(DR2)	Informatique	FAYOL
GOEURIOT	Dominique	DR	Sciences et génie des matériaux	SMS
GRAILLOT	Didier	DR	Sciences et génie de l'environnement	SPIN
GROSSEAU	Philippe	DR	Génie des Procédés	SPIN
GRUY	Frédéric	PR1	Génie des Procédés	SPIN
GUY	Bernard	DR	Sciences de la Terre	SPIN
GUYONNET	René	DR	Génie des Procédés	SPIN
HAN	Woo-Suck	CR	Mécanique et ingénierie	SMS
HERRI	Jean Michel	PR1	Génie des Procédés	SPIN
INAL	Karim	PR2	Microélectronique	CMP
KLOCKER	Helmut	DR	Sciences et génie des matériaux	SMS
LAFOREST	Valérie	MR(DR2)	Sciences et génie de l'environnement	FAYOL
LERICHE	Rodolphe	CR	Mécanique et ingénierie	FAYOL
LI	Jean Michel		Microélectronique	CMP
MALLIARAS	Georges	PR1	Microélectronique	CMP
MOLIMARD	Jérôme	PR2	Mécanique et ingénierie	CIS
MONTHEILLET	Franck	DR	Sciences et génie des matériaux	SMS
PERIER-CAMBY	Laurent	PR2	Génie des Procédés	DFG
PIJOLAT	Christophe	PR0	Génie des Procédés	SPIN
PIJOLAT	Michèle	PR1	Génie des Procédés	SPIN
PINOLI	Jean Charles	PR0	Image Vision Signal	CIS
POURCHEZ	Jérémy	CR	Génie des Procédés	CIS
ROUSTANT	Olivier	MA(MDC)		FAYOL
STOLARZ	Jacques	CR	Sciences et génie des matériaux	SMS
SZAFNICKI	Konrad	MR(DR2)	Sciences et génie de l'environnement	CMP
TRIA	Assia		Microélectronique	CMP
VALDIVIESO	François	MA(MDC)	Sciences et génie des matériaux	SMS
VIRICELLE	Jean Paul	MR(DR2)	Génie des Procédés	SPIN
WOLSKI	Krzystof	DR	Sciences et génie des matériaux	SMS
XIE	Xiaolan	PR1	Informatique	CIS

ENISE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'État ou d'une HDR)

FORTUNIER	Roland	PR	Sciences et Génie des matériaux	ENISE
BERGHEAU	Jean-Michel	PU	Mécanique et Ingénierie	ENISE
DUBUJET	Philippe	PU	Mécanique et Ingénierie	ENISE
LYONNET	Patrick	PU	Mécanique et Ingénierie	ENISE
SMUROV	Igor	PU	Mécanique et Ingénierie	ENISE
ZAHOUANI	Hassan	PU	Mécanique et Ingénierie	ENISE
BERTRAND	Philippe	MCF	Génie des procédés	ENISE
HAMDI	Hédi	MCF	Mécanique et Ingénierie	ENISE
KERMOUCHE	Guillaume	MCF	Mécanique et Ingénierie	ENISE
RECH	Joël	MCF	Mécanique et Ingénierie	ENISE
TOSCANO	Rosario	MCF	Mécanique et Ingénierie	ENISE
GUSSAROV Andrey	Andrey	Enseignant contractuel	Génie des procédés	ENISE

PR 0	Professeur classe exceptionnelle
PR 1	Professeur 1ère classe
PR 2	Professeur 2ème classe
PU	Professeur des Universités
MA (MDC)	Maître assistant
DR	Directeur de recherche

Ing.	Ingénieur
MCF	Maître de conférences
MR (DR2)	Maître de recherche
CR	Chargé de recherche
EC	Enseignant-chercheur
IGM	Ingénieur général des mines

SMS	Sciences des Matériaux et des Structures
SPIN	Sciences des Processus Industriels et Naturels
FAYOL	Institut Henri Fayol
CMP	Centre de Microélectronique de Provence
CIS	Centre Ingénierie et Santé

Mise à jour : 07/01/2013

To my children ...

Acknowledgments

If I have seen further it is by standing on the shoulders of Giants.
Isaac Newton

I would like to acknowledge the contributions of many people without whom I would not have been able to complete this work.

The first and foremost of these is, of course, my wife Wassila Amel. Thank you for your endless support and understanding. Thank you for everything you brought to my life. Thank you for giving me three wonderful children Sami Ibrahim, Rami Ismaël and Hiba Ikram. I'm lucky to have met you and I'm looking forward to spend the rest of my life with you.

I also owe my deepest gratitude and my warmest thanks to my supervisors, Olivier Boissier, Gauthier Picard and Philippe Jaillon. You committed guidance, understanding, encouraging and overly enthusiasm. You've made a deep impression on me and have provided a very important basis for this manuscript. Above all, thank you for all the nights and week-ends you've dedicated to read and correct my manuscript. Many thanks to your wives for their concern and understanding.

I'm also very thankful for the guidance of the reviewers and the jury members: Pr. Tim Norman, Pr. Laurent Vercouter, Pr. Mohand-Said Hacid, Pr. Pierre Maret and Pr. Nora Cuppens. It was an honor that you've accepted to evaluate my modest contribution. Thank you for spending your time reading my thesis and participating in my defense.

Many thanks to my family members. Above all, my mother which is my first fan and supporter. My father for being my motivation to succeed in life. My sister Maroua for her love and care. My sister Abbassia for her support. I want also to thank my brothers Samir and Kadiro for taking care of our parents during my absence. I'm very grateful to my grandmother "mima" for her blessings, to my uncles and aunts for their help. Also, I want to thank my step-family for their understanding and encouragement. Thank you for trusting me all these years.

When I joined the Ecole des Mines of Saint-Etienne, I was lucky to meet outstanding researchers and great persons who rapidly became my nearest friends. From all these people, I especially would like to thank Sonia Rouibi for her welcome, help and support in the two first years of my PhD. I want also to thank Malik Eddine Chalal (aka mami) and Abdelhamid Moutaouakil for their encouragements and advices in the last year. I'm also particularly grateful to Oussama Ben Amar, Lounes Bentaha, Nadjim Ouaissa, Adnane Lazrak, Yann Krupa, Rosine Kitio and Camille Persson. Thank you for the nice time we spent together, I have learned a lot from you.

I am grateful to the inhabitants of the Fayol Institute and beyond who have supported and entertained me during my thesis. Thanks to the members of the ISCOD team, thanks for the members of the DEMO team with whom I shared unforgettable moments during lunch and coffee breaks. Thanks to Gabi, Nilou, JF, Alain and marie-line. Thank you for the enjoyable moment we spent together at the “Espace Fauriel”.

I wrote this thesis while I was working as a research assistant at the Hubert Curien CNRS Laboratory of the University Jean Monnet of Saint-Etienne. I’m naturally grateful to the colleagues for their support and encouragement. I want to thank particularly Philippe Ezequel, François Jacquenet Fabrice Muhlenbach and Elisa Fromont.

I had the chance to supervise an Erasmus Master student during my PhD. So I wanted to thank Loredana Fau for her involvement and valuable contribution to my work.

I am thankful to my Algerian friends, who always supported me in my career. I especially would like to thank Ayoub, Azzeddine, Fouad, Hicham, Houari, Kada, Mohamed, Nouré, Raouf, Riad, Samir, Sid Ahmed and Zouaoui.

I would also like to acknowledge all my former teachers and professors, who hopefully still recognize me after all these years.

I would like to thank everybody who I have forgotten to thank. If I have forgotten to acknowledge you, I am in remiss. I want you to know that I am terribly sorry.

Last but not least, I want to thank you, you the reader of my manuscript, for your courage and interest. Thank you for making the research legend “ nobody ever reads a thesis” false !

دَعْوَىٰهُمْ فِيهَا سُبْحَٰنَكَ ٱللَّهُمَّ وَتَحِيَّتُهُمْ فِيهَا سَلَٰمٌ ۚ
وَءَاخِرُ دَعْوَىٰهُمْ أَنِ ٱلْحَمْدُ لِلَّهِ رَبِّ ٱلْعَٰلَمِينَ

Mohamed Reda Yaich
Saint-Etienne, October 2013

Contents

Part I

Introduction

Chapter 1

Introduction

The work presented in this dissertation addresses the problem of trust management in *open* and *decentralised* virtual communities (VCs). We address this problem by proposing an Adaptive and Socially-Compliant Trust Management System (ASC-TMS) adopting a multi-agent based approach. Our proposal relies on the joint use of adaptive trust policies and adaptation meta-policies, which allow VCs members to make adaptive and socially-compliant trust decisions. This chapter introduces the context, challenges, research motivations and contributions of this thesis.

1.1 Context

The astonishing recent advances in computer systems and networks boosted the use of the Internet, especially via mobile devices. This new pervasive nature of the Internet enabled billions[1] of worldwide distributed people to intensify their usage by creating, sharing, collaborating and socialising in many different ways among open and decentralised social structures called Virtual Communities (VCs).

The concept of virtual community was first proposed by Howard Rheingold as the title of his book [Rheingold, 1993]. In this book, Rheingold reported his experience on WELL[2] to advocate the potential benefits of belonging to a virtual community. Thenceforth, and particularly in the last decade, this concept has found a wide recognition in business and research.

Preece defined virtual communities as a "group of *people* with a common *purpose* whose interactions are mediated and supported by *computer platforms*, and governed by formal and informal *policies*" [Preece, 2001, Preece, 2004]. This definition characterises virtual communities throughout four primary concepts that are *people*, *purpose*, *platforms* and *policies* [Preece, 2001, Preece, 2004].

[1]List of virtual communities with more than 100 million active users http://en.wikipedia.org/wiki/List_of_virtual_communities_with_more_than_100_million_active_users

[2]Whole Earth 'Lectronic Link is one of the oldest virtual communities which relies on the use of Internet forums.

1.2 Motivations and Challenges

Generally, individuals get involved in virtual communities to collaborate for the purpose of achieving a particular goal. More that any other form of interaction, collaboration relies on resources, information and knowledge sharing, making security a critical issue for the success of virtual communities [Renzl, 2008, Casimir et al., 2012].

In security, several access control (AC) models have been successfully used to prevent unauthorised access to private or sensitive resources. However, most of these models operate under a known and finite universe of discourse; the system administrator needs to know beforehand what are the resources to be protected and what are the criteria based on which users should be granted to these resources. Due to the open nature of virtual communities, these solutions are clearly not applicable. Additionally, virtual communities are decentralised systems in which the existence of a unique central authority is neither possible nor desirable and each participant is the security administrator of its own resources.

The advent of the Internet and the evolution of communication from standalone systems to *open* and *decentralised* ones has driven research to think beyond security. In such context, the concept *trust management system* [Blaze et al., 1996, Blaze et al., 1999b] has been used in reference to systems that make use of *credentials* and *policies* to express delegation of rights based on existing *ad hoc* trust relationships. In this approach, it is the resources' owners, and not the security administrators, that are responsible for specifying who and when an access can be granted. So the role of a trust management system consists of evaluating whether the specified policies and credentials allow the establishment of a trust relationship (materialised by the *delegation* chain) between the resources *owner* and the resource *requester*. Using delegation mechanism, rights and privileges are dynamically created, evaluated and revoked. These systems do not need any-more to know beforehand the resources to be protected, or the users that are allowed to access to these resources.

While most of the trust management systems that have been proposed so far get focused on addressing *decentralisation* and *distribution* properties, the *social* and *dynamic* nature of virtual communities are still challenging issues. In what follows, we delineate the characteristics of trust management in *virtual communities* to derive the objectives of our thesis.

1.2.1 Social Aspects (C1): Social-Awareness

By using the concept of trust, research on *trust management* has made explicit the parallel between human societies and distributed computer systems. So many of what we can learn about human communities may be applicable in virtual ones (cf. Chapter 2). However, although there have been several contributions that draw inspiration from social sciences, little work has been done with respect to *what mechanisms humans use to deal with situations in which the trust decision they make may impact other members of the community.* In the light of this, the first challenge addressed in this thesis is to think about *trust* ***management mechanisms*** *that are not only safe for the* ***individual*** *using them but also* ***collectively*** *(or socially) harmful.*

1.2.2 Dynamic Aspects (C2): Context-Awareness

Open and decentralised virtual communities are by definition *dynamic* environments; *resources* are unpredictably created, updated and destroyed and the population turnover is known to be very high. However, the best practices in use by existing trust management system to handle such dynamics are essentially manual and *ad hoc*; the human user analyses changes in the environment and updates the used policies as required. Such practices lead, in large systems, to high operational costs, broken closed loop automation, and reduced reactivity. In the light of this, the second main challenge addressed in this thesis is to *investigate trust management mechanisms that are able to* ***automatically adapt*** *policies in reaction to environment changes.*

1.3 Objective and Requirements

The purpose of this research is to investigate issues of automated trust assessment, and to provide accordingly a system that assists members of *open* and *decentralised virtual communities* in their trust decisions.

1.3.1 Requirements

The realisation of this objective, given the challenges identified in the previous section, involves three important requirements that we describe below.

1.3.1.1 Communicability

In human-based virtual communities, the conditions on which members decide who to *collaborate* with are generally made implicit. However, given our objective (design a system that assists VC user in their trust decisions), members must be able to *communicate* and *exchange* these conditions, and thus make them explicit. *Communicability* in trust is important for several reasons [Koster, 2012]. First, to assist humans in their trust decisions, the system must have an explicit representation of the conditions on which it will make its trust evaluations. Second, in *open environments*, members may interact with individuals that they have no prior experience with. In such a situation, the ability to communicate about their trust conditions allows these members to gather valuable information (either from the partner, or from the other members of the community) that they can use in their trust evaluation. Finally, collaboration within VCs involves shared resources and knowledge, the conditions used to made decisions about these resources need to be explicitly discussed and shared within the community.

1.3.1.2 Expressiveness

In the previous section, we motivated the need for a specification language for virtual communities members to communicate about trust conditions among them selves and with their trust management system. However, the majority of existing systems are built upon a controlled

–and limited– set of information based on which they base trust evaluations [Marsh, 1994]. Such limitation represents a lock for subjectivity and constrains the expressiveness of these languages, and by the way the accuracy of the trust management system using them. This language must be sufficiently *expressive* and *rich* to not constrain virtual community members in the expression of their conditions, characterising the subjective nature of trust.

1.3.1.3 Flexibility

VCs are open environments that members can spontaneously join and leave, and where resources are unpredictably created, updated and destroyed. In such systems, future interactions are impossible to predict, making the specification of trust conditions hazardous and risky. In such settings, the conditions based on which trust is evaluated cannot be specified *ad vitam æternam* (specified once, used forever). So the general question that must guide the specification of these conditions should not be "how efficient these conditions are", but rather "how adaptive, in response to an ever-evolving environment they could be". However, in order to be adapted to environmental changes, the formalism used to express trust conditions must be *flexible* enough in order to be *automatically* adapted and evolved.

1.3.2 Objectives

Here are the aims and objectives addressed in thesis towards endowing *trust management systems* with *social-awareness* and *context-awareness* capabilities.

O1* – *Social trust management. According to Pearlman, "a key problem associated with the formation and operation of distributed virtual communities is that of how to specify and enforce community policies" [Pearlman et al., 2002]. *Collective policies* are core elements of a virtual community as suggested by the aforementioned definition. They are used by the members of the community to govern the access to communities and the resources shared among these communities. With respect to that, our first objective is to design a trust management system that addresses this objective. This objective is further decomposed into two sub-objectives:

- **O1.1 – Collective policy specification management.** We address the issue of *how to specify collective policies in a decentralised way.* The objective is to allow virtual communities members to build and agree on a policy to use for trust decision-making within the community.
- **O1.2 – Collective policy enforcement management.** Once collective policies have been specified, we address the issues of *how to ensure that community members are aware of these policies* and *how to enforce them.*

O2* – *Adaptive trust management. Policies constitute the bedrock of trust management systems. But a policy that is not adequate for its context is doomed, sooner or later, to become either too *restrictive* or too *permissive*. Neither a too restrictive policy nor a too permissive one are desirable, as both can be harmful for their user [3]. So given the highly dynamic environment in which virtual communities member evolve, our second objective is to design a trust management system that is sensitive to changes. This objective involves two sub-objectives:

O2.1 – **Individual policy adaptation management.** The objective is to provide virtual community members with mechanisms throughout which they can make their individual policies react to changes in their environment.

O2.2 – **Collective policy adaptation management.** Virtual community members need mechanisms throughout of which they can adapt the collective policies to which they are subject.

This thesis contributes to the field of *trust management* by achieving the objectives listed above. In the next section, we describe the main approach advocated in this thesis and present the contributions we have made towards the achievement of these objectives.

1.4 Approach and Contributions

In order to design a system that assists the members of open and decentralised VCs in their trust decisions, we draw inspiration from *Automated Trust Negotiation Systems (ATN)* which make the achievement of our objective easier due to the explicit use of policies. Then to achieve the social trust management:

- First, we draw inspiration from existing literature in *Decentralised Trust Management Systems (DTM)* in which *algorithms* and *mechanisms* have been proposed to make distributed systems make collective trust decisions. These mechanisms have been adapted in our thesis to *specify*, *use* and *adapt* collective policies.

- Second, we have made explicit the parallel between virtual communities and human communities. To that aim, we used models from *sociology* that explain in which conditions *collective policies* emerge (i.e. they are specified), are enforced and/or adapted. These mechanisms are drawn from the well-known *Social Influence Theory* (SIT) and have been used in our thesis to formalise the concept of *socially compliant trust management*.

Finally, we opted for multi-agent techniques (i.e. MAS) to make our trust management system work in a *distributed* and *decentralised* way. Concretely, we used *normative MAS* to make virtual community members comply with collective policies (i.e. top-down social compliance). We are also inspired by research in *self-organised MAS* to implement the *specification* and *adaptation* of collective policies. The choice of MAS hits in with our objectives as we use

[3]Too restrictive policies are detrimental for the user activity, while too permissive policies are dangerous of its security.

the *reactive* feature of agents to make our system react to environment changes, enabling the *adaptive trust management*.

In order to expound this approach throughout this thesis, we will present the following contributions:

- **A multi-agent trust management system** in which trust is evaluated in an open and decentralised way (cf. Chapters 5, 6 and 7). The system assists virtual community members in making trust decisions that are in compliance with their requirements (i.e. individual policies) and those of their community (i.e. collective policies).
- **A flexible trust policy language** that makes possible the expression of individual and collective policies. The language is inspired from *weighted-logics* as each condition is associated a weight that expresses its importance with respect to the evaluated trust level. The policy is also built on the top of an ontology that makes the policies intelligible by both humans and artificial agents (cf. Chapter 5).
- **A policy algebra** throughout which the system can manipulate trust policies. This algebra is defined in terms of operators that make virtual community members' *specify*, *enforce* and *adapt* their individual and collective policies (cf. Chapter 5). This algebra also defines operations based on which the system can *relax*, *restrict* and *update* any of the conditions stated by the policies it is manipulating. This is made possible thanks to the use of an ontology that enables the system to reason about the policies it is uses.
- **Social-awareness meta-policies** that allow virtual community members to specify *when* and *how collective policies* are *specified* and *enforced*. These meta-policies implement our formalisation of Social Influence Theory (cf. Chapters 4 and 7). They are expressed as event-condition-action (ECA) rules that users can personalise.
- **Context-awareness meta-policies** by means of which users specify to the system *when* and *how* their individual and collective policies can be adapted. These meta-policies are also expressed under the form of ECA rules (cf. Chapter 7)).

1.5 Thesis Outline

This thesis is structured into fives parts and organised as follows:

Part I contains this introductory chapter and motivates our research. We described objectives of this thesis that we used to delineate the implied requirements. Finally, we presented the approach we used to tackle these objectives and summarised our contributions.

Part II surveys the literature on trust, trust management, social influence theory and multi-agent systems. The main disciplines that investigated this concept are presented and their prominent characteristic defined. We identify *types* and *properties* of trust that we used in Chapter 6 to frame our trust model. Then we reviewed models from the two main approaches for trust management in computer science, namely *distributed artificial intelligence* and *security*. Finally, we discuss the limits of each approach with respect to our objectives. Then we explain why we decided to address the trust management in virtual communities from the *security* perspective. Chapter 3 details the background concepts of trust management systems. We identify important properties a TMS must exhibit and compare existing systems with respect to these properties and our requirements as well.

Part III presents our contribution. In Chapter 5 we present the multi-agent framework that we used to build *open* and *decentralised* virtual communities. It is on the top of this framework that we deployed our trust management system. Chapter 6 presents our contribution towards the management of trust. This contribution is composed of: (a) a policy specification language that allows virtual community members to specify the policies they require in their partners to consider them trustworthy, and (b) a trust management system that makes use of these policies to compute trust evaluation and eventually derive trust decisions. In Chapter 7, the mechanisms we used to bring *social-compliance* and *adaptiveness* to our trust management system. We provide a mechanism to specify (cf. Section 7.5.1), *enforce* (cf. Section 7.5.2), and *adapt* (cf. Section 7.5.3) collective policies. We provide also mechanisms to make *individual policies* fit the context of the interaction (cf. Section 7.4.1 and Section 7.4.2).

Part IV presents the evaluation of our contributions. In Chapter 8 we present the implementation of ASC-TMS. In Chapter 9, we apply our approach to an example of virtual community (open innovation virtual community). In Chapter 10 we evaluate the performance of ASC-TMS and discuss its benefits with respect to *open innovation communities*.

Part V summarises the main contributions proposed in this thesis. We present our conclusion and discuss some future line of research.

1.6 Related Publications

Part of this thesis has been previously peer-reviewed and published. These publications are listed below:

- Yaich, R., Boissier, O., Picard, G., and Jaillon, P. (2013). Adaptiveness and social-compliance in trust management within virtual communities. Web Intelligence and Agent Systems (WIAS), 11(4):315-338.

- Yaich, R., Boissier, O., Picard, G, and Jaillon, P. (2012b). An agent based trust management system for multi-agent based virtual communities. In Demazeau, Y., Müller, J. P., Rodríguez, J. M. C., and Pérez, J. B., editors, Advances on Practical Applications of Agents and Multiagent Systems, Proc. of the 10th International Conference on Practical

Applications of Agents and Multi-Agent Systems (PAAMS 12), volume 155 ofAdvances in Soft Computing Series, pages 217-223. Springer.

- Yaich, R., Boissier, O., Jaillon, P., and Picard, G. (2012a). An adaptive and socially-compliant trust management system for virtual communities. InThe 27th ACM Symposium On Applied Computing (SAC 2012), pages 2022-2028. ACM Press.
- Yaich, R., Boissier, O., Picard, G., and Jaillon, P. (2011b). Social-compliance in trust management within virtual communities. In European Workshop on Multi-agent Systems (EUMAS'11).
- Yaich, R., Boissier, O., Jaillon, P., and Picard, G. (2011a). Social-compliance in trust management within virtual communities. In 3rd International Workshop on Web Intelligence and Communities (WI&C'11) at the International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT 2011), pages 322-325. IEEE Computer Society.
- Yaich, R., Jaillon, P., Boissier, O., and Picard, G. (2011). Gestion de la confiance et intǵration des exigences sociales au sein de communautés virtuelles. In 19es Journées francophones des systèmes multi-agents (JFSMA'11), pages 213-222. Cépaduès.
- Yaich, R., Jaillon, P., Picard, G., and Boissier, O. (2010). Toward an adaptive trust policy model for open and decentralized virtual communities. InWorkshop on Trust and Reputation. Interdisciplines.

1.7 French Summary

Dans cette dissertation, nous nous intéressons à la problématique de la gestion de la confiance au sein de communautés virtuelles *ouvertes* et *décentralisée*. Pour cela, nous proposons un système de gestion de la confiance adaptatif et conforme aux exigences sociales (ASC-TMS). Notre proposition repose sur l'utilisation conjointe de politiques adaptatives et de méta-politiques d'adaptation. Ces méta-politiques permettent aux membres des communautés virtuelles de prendre des décisions qui sont à la fois en accord avec l'environnement mais également avec la communauté dans laquelle la décision a été prise. Dans ce chapitre, nous introduisons les contextes, les challenges, les motivations ainsi que les contributions apportées dans cette thèse.

1.7.1 Contexte

L'évolution technologique fulgurante dont nous avons été témoins lors de cette dernière décennie a dopé l'utilisation d'Internet et plus particulièrement via des dispositifs mobiles comme les smartphones et les tablettes. Ce nouveau mode de communication ubiquitaire a amené les utilisateurs à intensifier leurs activités de création, de partage, de collaboration et de sociabilisation de différentes manières et plus particulièrement au sein de nouvelles structure sociales, virtuelles, ouvertes et décentralisées. Ces structures sont communément appelées communautés virtuelles.

1.7.2 Motivations et Challenges

Généralement, les utilisateurs s'activent au sein de communautés virtuelles dans la perspective d'atteindre un objectif commun. Par exemple, les personnes qui constituent la communauté virtuelle des collaborateurs de Wikipedia ont pour objectif commun de construire la plus grande encyclopédie ouverte en ligne. Plus que toute autre forme d'interaction, la collaboration repose sur le partage de ressources, d'information et de connaissances. Cela place donc les problématiques de sécurité et plus particulièrement ceux liées à la confiance au centre des préoccupations de chaque membre.

En sécurité, plusieurs modèles de contrôle d'accès on était utilisé avec succès pour empêcher des manipulations non autorisés de ressources dites sensibles, privés ou confidentielles. Cependant, la plupart de ces modèles opèrent dans un univers connu et fini. La personne en charge des droits d'accès doit connaître à l'avance quels sont les utilisateurs du système, quelles sont les ressources sensibles et quels sont les droits de chaque utilisateur. Or, compte tenu de la nature ouverte des communautés virtuelles, ces mécanismes ne peuvent fonctionner. De plus, les communautés virtuelles sont décentralisés et l'existence d'une autorité centrale qui décide des droits de chacun n'est ni possible ni souhaitable.

Ainsi, l'avènement de l'internet a poussé les chercheurs dans le domaine de la sécurité à envisager de nouvelles pistes et à proposer des solutions qui vont au-delà de la sécurité. À partir des années 2000, une nouvelle génération de systèmes a vu le jour sous la dénomination des systèmes de gestion de la confiance. Alors que la plupart de ces systèmes s'est attelé à soulever les défis liés à la distribution et la décentralisation des systèmes modernes, la nature **sociale** et **dynamique** des communautés virtuelles demeure un défi pour la discipline. Dans cette thèse nous nous intéresserons à ses deux propriétés qu'on considère comme étant les principaux challenges pour la gestion de la confiance au sein de communautés virtuelles.

1.7.3 Objectives et Besoins

Dans ce contexte, l'objectif principal de cette thèse est de proposer un outil afin d'automatiser l'évaluation de la confiance dans le but d'assister les membres de communautés virtuelles *ouvertes* et *décentralisées* dans leur prise de décision.

1.7.3.1 Besoins

La satisfaction de cet objectif implique trois besoins primordiaux que nous identifiions comme suit:

- **Communication:** durant leurs activités au sein des communautés virtuelles, les humains reposent sur un ensemble de facteurs pour prendre leurs décisions. Ces facteurs sont implicites car l'utilisateur à tout intérêt à les maintenir secrets au risque de voir certains profiter d'éventuelles brèches. Or, dans notre objectif, nous souhaiterions que ces utilisateurs soient assistés dans leur prise de décision. Ainsi, le premier besoin que nous identifions est la nécessité d'avoir un moyen afin de permettent à chaque utilisateur d'exprimer de manière personnelle et subjective les conditions à partir desquelles il prend ses décisions.

- **Expressivité:** le besoin de communication fait appel directement à un autre besoin essentiel dans le cadre de la réalisation de l'objectif de la thèse. Ce besoin n'est autre que l'expressivité. En effet, le fait d'imposer aux utilisateurs l'expression de leurs exigences en matière de décision de confiance nous impose une grande richesse en matière d'expressivité. Or, à ce jour, les mécanismes utilisés reposent sur un ensemble connu et fini de conditions à partir desquels les décisions sont prises. En plus d'être un obstacle pour l'expressivité, ces solutions peuvent affecter la précision de l'évaluation fournie par le système.

- **Flexibilité:** compte tenu du caractère ouvert et dynamique des communautés virtuelles, les interactions futures sont presque impossibles à prédire rendant la spécification d'exigences risquée et hasardeuse. N'ayant aucun moyen de prédire l'imprédictible ni de le contrôler, il nous semble qu'un besoin essentiel pour la satisfaction de notre objectif réside dans la capacité de notre système à réagir aux éventuelles évolutions du contexte (à la fois sociale et environnementale). Ainsi, les exigences exprimées par les utilisateurs des communautés virtuelles doivent être assez flexibles pour qu'elles puissent s'adapter et évoluer avec le système.

1.7.3.2 Objectifs

Compte tenu des besoins listés plus haut ainsi que l'objectif de la thèse, nous pouvons lister ci-dessous les sous-objectifs suivants:

- *Gestion de la confiance sociale (O1)*: D'après Pearlman, un des problèmes majeurs des communautés virtuelles réside dans la spécification et la mise en œuvre de politiques collectives [Pearlman et al., 2002]. Ainsi, un des objectifs de la thèse consistera à trouver un moyen pour permettre cette gestion de la confiance sociale en offrant aux membres la possibilité de : (O1.1) *spécifier* et (O1.2) *mettre en œuvre* des politiques collectives.

- *Gestion de la confiance adaptative (O2)*: les politiques de confiance sont la pierre angulaire de la gestion de la confiance. Cependant, une politique qui n'est pas en adéquation avec son contexte est, tôt ou tard, condamné à devenir soit trop restrictive soit trop permissive. Ainsi, le second objectif de la thèse consiste à proposer des mécanismes afin de permettre aux politiques de s'adapter au contexte. Cet objectif se décompose en deux sous-objectifs: (O2.1) *adaptation des politiques individuelles* et (O2.2) *adaptation des politiques collectives*.

Cette thèse contribue au domaine de la gestion de la confiance en atteignant les objectifs listés dessus.

1.7.4 Approche

Afin de concevoir un système qui assiste les membres de communautés virtuelles *ouvertes* et *décentralisées* nous avons puisé notre inspiration dans le domaine des *systèmes de négociation automatique de la confiance*. Ces systèmes ont l'avantage d'utiliser de manière explicite des politiques de confiance. Cela qui facilite grandement l'implémentation de l'aspect adaptatif du système de gestion de la confiance. Ensuite, pour ce qui concerne l'aspect social :

- Premièrement, nous avons tiré profit des mécanismes utilisés dans les systèmes de gestion de la confiance décentralisées pour *spécifier*, *utiliser* et *adapter* les politiques collectives.

- Ensuite, nous avons utilisé des modèles sociologiques afin d'expliquer *quand*, *pourquoi* et *comment* ces politiques collectives *émergent*, *son utilisés* et *adaptées*.

Enfin, nous avons opté pour une approche multi-agent (i.e. Système Multi-Agent – SMA–) pour permettre à notre système de fonctionner de manière distribuée et décentralisée. Concrètement, nous avons utilisé un SMA normatif afin de faire en sorte que les membres des communautés se conforment aux politiques collectives (i.e., conformité descendante). Et on s'est inspiré des mécanismes proposés dans les systèmes auto-organisés afin de mettre en œuvre la spécification et l'adaptation des politiques collectives.

Part II

State of the Art

Chapter 2

On the Nature of Trust

In the real world as well as in the virtual ones, trust constitutes a fundamental concept for humans, without which they can neither act nor interact. This is probably why Abraham Maslow placed trust in the second layer of his hierarchy of humans' needs [Maslow, 1943], just after the physical requirements for human survival.

So unsurprisingly, trust received in the last decades much attention from several disciplines such as philosophy, psychology, sociology, economics and, more recently, computer science, each one from a different perspective. Given this large and ever-increasing literature on trust, in addition to the importance of this concept towards the contributions made in this thesis, it seems essential for the sake of clarification, to dedicate a whole chapter to this concept and review the state of knowledge to date.

This chapter aims at defining the concept of Trust in Computer Science and how it will be understood in this document. To that goal, we first examine how Trust is considered in Human societies by analyzing this concept through the looking glass of Philosophy, Psychology, Sociology and Economics. From that first understanding we focus on Trust in Computer science.

2.1 A "brief" Retrospective Study About Trust

The objective of this section is not to review, exhaustively, all contributions on the subject, the thesis format does not permit that. However, it seems fundamental for us to see how flagship disciplines that are philosophy, psychology, economics, sociology, and finally computer science addressed trust in order to be able to define (cf. Section 2.2), characterise (cf. Section 2.3.3) and classify (cf. Section 2.3) this concept.

2.1.1 The Philosophical Perspective

Research on trust in philosophy has a long tradition dating back to the ancient Greece. During that period, already, philosophers (e.g. Socrates, Plato and Aristotle) led a deep reflection on what motivates a person to rely on trust. Their conclusions suggested that only reason and rigorous examination of facts allows access to the truth [Plato, 1994]. So the philosophical reflection, at that time, seems to be against the idea of trust even if they recognise its importance, particularly in the organisation of the society. This conclusion was further reinforced with the advent of modern rationalism in the seventeenth century. At that period, trust was considered as an illusive belief that should be denied in favour of rationality. For instance, when Descartes addressed uncertainty, he suggested to reject any knowledge that is uncertain or based on testimonials [Taylor, 1941].

Nevertheless, few years later, some critics of rationalism came back on the controversial relationship between trust and reason. For instance, Kant believes that human reason alone cannot provide full access to knowledge [Kant, 1788]. He argues that beliefs such as trust do not necessarily oppose rationality [Lovejoy, 1968, Harris, 1977].

Thenceforth, modern philosophers seem to have accepted the existence of trust in interpersonal relationships. Also, there seems to be a form of consensus among philosophers that trust should be conceptualised as a belief that is founded on knowledge. However, even if they agree on the importance of knowledge, they still do not agree on the nature of this knowledge. For instance, for the movement initiated by Kant trust appears to involve beliefs that are not accepted on the basis of evidence (*rational trust*), while earlier philosophers (e.g. Aristotle and Descartes) seem to ignore all what is not rationally proved (*rational trust*).

2.1.2 The Psychological Perspective

In psychology, trust typically involves two cognitive processes: the feeling of vulnerability, and expectations towards the behaviour of the partner. These processes explain why there have been two main approaches in research about trust in psychology: *dispositional* and *interpersonal* trust.

Dispositional trust emerged from the works of earliest psychologists such as Deutsch [Deutsch, 2011, Deutsch and Gerard, 1955] and Rotter [Rotter, 1967]. With respect to these psychologists, trust can be conceptualised as an attitude that expresses the degree to which an individual is willing to rely on, cooperate with, or help others. This view of trust was also called *person-centred* as it highlights the likelihood of an individual to be vulnerable to others behaviour [Deutsch, 2011, Deutsch and Gerard, 1955].

The *Interpersonal trust* aspect of trust emerged in psychology in the mid 80s [Rempel et al., 1985]. This form of trust is conceptualised as a psychological state of an individual (the *trustor*) towards a particular interlocutor (the *trustee*) with respect to a specific purpose [Simpson, 2007].

So research on both *dispositional* and *interpersonal* considers trust as a cognitive process, a mental state or a feeling that aims at reducing or at least accepting uncertainty. They explained the process by the consent of an individual to shift from the confidence in their ability to predict partner's behaviour to the confidence on the partner's values, motivations, intentions and goals [Wieselquist et al., 1999].

2.1.3 The Sociological Perspective

Sociology has been one of the most prolific disciplines in research on trust (Misztal [Misztal, 1996] provided a comprehensive study about trust in sociology). For many sociologists (e.g. Luhmann [Luhmann, 1990] and Giddens [Khodyakov, 2007]), modern society is too complex for humans' decision making mechanisms. In consequence, they assume that humans naturally tend to rely on trust to simplify the complexity of the interactions they are involved in [Luhmann, 1990, Mahoney et al., 1994].

Sociologists described trust as a relationship and distinguished two types: *trust in individuals* and *trust in systems*. Further, Zucker [Zucker, 1986] explained that trust in individuals is produced based on the experience that an individual acquires after its repeated interactions. Zucker explained also that *trust in systems* is related to the structure to which the individual belongs and that guarantees specific attributes for individuals [Zucker, 1986]. For Luhmann, technology as systems provides a certain level of guarantees that are required for the trust establishment. Indeed, formal role descriptors (e.g. lawyer, doctor, professor) provide a partial foundation of trust. Provided that these professional did not receive their credentials by fraud, one may expect them to be trustworthy for the purpose of the role they committed to [Shapiro, 1998].

Trust in individuals corresponds to the *interpersonal trust* we presented in the philosophical perspective. The *trust in systems* was later used by economists in the context of *conventions*. These links between disciplines can be explained by the fact that philosophy was used as a basis for the works in sociology and that, in turn, concepts introduced in sociology was reused in economics.

2.1.4 The Economical Perspective

For neoclassical economists (e.g. Williamson [Williamson, 1993]), individuals (*homo economonicus* as they call them) are purely rational; they seek their personal interest through a permanent optimisation of utility [Mcallister, 1997, Mcallister, 1995]. So, *a priori*, economists refuse to rely on trust to explain humans decisions. For instance, for Williamson the existence of mutual interests is sufficient to enable cooperation and only the risk/opportunity ratio motivates decisions in economics. Thus in neoclassical economics, trust is merely a rational calculation of reciprocal interest.

However, in uncertain situations the individual calculus is made impossible due to the insufficiency of information [Orléan, 2000, Mcknight and Chervany, 1996]. So in response, some works conducted in economics consider that interactions between individuals requires a common framework called *convention* [Salais, 1989, Eymard-Duvernay et al., 2003]. The convention has been defined by Salais as "a system of reciprocal expectations about other's competences and behaviours" [Salais, 1989]. So a convention is a kind of trust that aims at increasing the quality of information that an individual has about others and thus mechanically, to lower the uncertainty with respect to their future behaviour [Hosmer, 1995]. With respect to that, Arrow defined trust as an “invisible institution” (i.e. *institutional trust*) having a real economical value, which is distinct from the objects of the interaction [Arrow, 1984, Arrow, 1974].

Thenceforth, contemporary economists devoted significant attention to trust which they consider as a lubricant for decision making within economical and social systems. This conceptualisation of trust has the merit of shedding the need to apprehend the phenomenon not only from an individual (rational) perspective but also from a social perspective.

2.1.5 The Computer Science Perspective

In the previous sections, we evidenced the importance of trust (from different perspectives) when humans are interacting with each other. The necessity of trust appeared with the advent of distributed computing (e.g. Internet) and the increasing use of electronic interpersonal interactions rather than face-to-face ones, especially when interacting partners are software agents [Grandison and Sloman, 2000]. In such settings, the inability to obtain complete information, that is inherent to any distributed system, subjects the use of software programs to great uncertainty.

Trust in computer science has been mainly investigated in *distributed artificial intelligence* (DAI) and *security* disciplines wherein uncertainty constitute the most challenging issue. In these disciplines, and more generally in computer science, it is not sufficient to just define it (what other disciplines mainly focused on). For automation, the concept of trust must be *formally* conceptualised and represented. The word *formal* here means that this concept should be captured, represented and manipulated by software programs. As a consequence, the main contribution of computer science to the research about trust constitute in formalisation, implementation and evaluation of theories introduced by scholars from the disciplines aforementioned. In Section 2.4, we further detail the *computer science* perspective by presenting a literature review of some models from both disciplines.

Discussion

After the brief retrospective study on trust, three important remarks may be made:

- Trust manifest itself in societies. Trust is relevant in situations involving two interacting entities. These entities are generally humans, but researchers in computer science also used trust in interactions between software agents too.

- Given the diversity of the disciplines in which trust has been investigated, this concept has been defined in different ways: *belief* (in philosophy), *cognitive process* (in psychology), *decision* (in economics), *relationship* (in sociology and computer science), and *action/decision* (in computer science).

- A comprehensive classification of trust can be made base on the *properties*, *nature* and *origin* of trust. These features are detailed in the following sections.

Finally, beyond their diversity, the different disciplines in which trust has been addressed agree about the centrality of trust in humans interactions, and virtual ones as well. However, and despite this agreement, up to date, there is no consensual definition about what is trust. Indeed, one can easily imagine the difficulty of making the diverse trust perspectives studied in previous sections fit into the same definition. For example, let us consider an e-commerce transaction in which the buyer trusts the seller to keep his payment records private and that the goods he bought will meet his expectation in terms of quality; in return, the seller trusts the buyer that the credit card (or any other payment means) he used during the transaction are not stolen or forged. In this example, trust is inherent to decisions made by the buyer and the seller as well. However, their answers to the question "how would you define trust?" would probably be different as each one uses trust for different reasons.

2.2 An Attempt to Define Trust

In this section, we will tackle the issue of defining trust from a computer science perspective. To that aim, we first discuss in Section 2.2.1 related concepts that are misleadingly used as synonyms of trust. Then we review in Section 2.2.2 some existing definitions and discuss their relevance with respect to our objective. Finally, we conclude this section by providing our definition of trust and discuss it.

2.2.1 What Trust Is Not?

In order to understand what trust really is, we must first clarify what trust is not. In this section, we are particularly interested in few concepts that have been confused with trust in recent works.

- Trust is not **trustworthiness**. Although trust and trustworthiness are two distinct concepts, there is a strong link between them. Indeed, an individual A trusts an individual B because B succeeds exhibiting trustworthiness characteristics that are relevant to A requirements.
- Trust is not **reliability**. Reliability refers to the ability of an individual to perform as expected even in unexpected circumstances. For instance, an individual A can rely on a individual B (e.g. policeman, doctor, mechanical) without that A trusts B.
- Trust is not **credibility**. Credibility is generally used in the context of an information provided by an individual. An individual says that another individual B is credible in reference to its ability to provide consistent and believable information.
- Trust is not **competence**. Competence has often been used as a particular form of trust (cf. [Deutsch, 2011, Deutsch and Gerard, 1955]). However, trust goes beyond the belief that a trusted person is competent or not. Consequently, competence can be considered as

a characteristic to consider a person trustworthy. But we can never say that all competent persons are trusted persons.

- Trust is not **reputation**. Even if these concepts are closely linked and often used interchangeably. Therefore, it is paramount to discuss their differences, which can be easily stressed by the following statements:

  – “A trusts B because of B’s good reputation.”

  – “A trusts B despite B’s bad reputation.”

  Based on the above statements, Jøsang[Josang and Presti, 2004, Jøsang, 2007] argued that trust is ultimately a personal and subjective concept while reputation is an objective measure of an individual’s trustworthiness based on others’ opinions and observations. In absence of personal experience, reputation is often used which explains why many scholars used these concepts interchangeably.

2.2.2 What Is Trust?

In order to define what we mean by “trust”. We take the definition provided by Gambetta [Gambetta, 2000]. The author defined trust (or, symmetrically, distrust) as a “particular level of the subjective probability with which an individual, A, expects that another individual, B, performs a given action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action”. First, we notice here that Gambetta, which is a sociologist, conceptualised trust as a mathematical concept, making its definition more concrete. Also, the part “a particular level” of the definition means that for Gambetta, trust can be somehow quantifiable. For Gambetta, 0 means complete distrust and 1 full trust. Further, this definition makes explicit the idea that trust is subjective and introduces the specificity of trust: trust is made with respect to a specific action to be performed. This definition takes into account uncertainty induced by the behaviour of the interacting partner, without which there would be no need for trust. Further, Gambetta states that “if we were blessed with an unlimited computational ability to map out all possible contingencies in enforceable contracts, trust would not be a problem”. With this statement, Gambetta highlights the fact that trust involves decision in complex situations that are hard to grasp for human minds (cf. bounded computational agents).

The complex nature of the situations involving trust is further reinforced by the definition provided by *Niklas Luhmann* [Luhmann, 1990]. For Luhmann “the complexity of the future world is reduced by the act of trust”. Luhmann approaches trust from a sociological background as he relates the use of trust to interactions among societies and questions the existence of society without trust [Lamsal, 2001, Marsh, 1994]. He considers trust as one of the most important “internal mechanisms for the reduction of complexity”. However, the authors defined complexity in very abstract terms even if generally he used it in reference to *uncertain* situations.

In computer science, the definitions proposed by McKnight [Mcknight and Chervany, 1996], Grandison [Grandison, 2003] and Castelfranchi [Castelfranchi and Falcone, 2000b, Falcone and Castelfranchi, 2001] are frequently used. McKnight [Mcknight and Chervany, 1996] defined trust as “the extent to which one party is willing to depend on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible”. For Grandison and Sloman, trust is “the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context”. Finally, Castelfranchi adopted a more psychological approach to trust [Reh, 2008]. According to the author, trust is “a mental state, a complex attitude of a agent X towards another agent Y about the behaviour/action relevant for the result (goal) g”. Common to the three definitions, trust is considered as a situation of vulnerability and dependability towards the behaviour of another entity. Further, all definitions agree on the subjective and context-dependant nature of trust. Finally, for Castelfranchi, and Chaveny as well, trust is used as “the mental counterpart of delegation” [Falcone and Castelfranchi, 2001]. So the authors consider delegation as an action taking this mental state as an input. While Grandison adopts a broader approach as it consider trust as a belief but he reduced this belief to the knowledge about the competence of the trustee.

Each of the above definitions advanced our understanding of trust and provides important building bricks for the definition of this concept. However, none of them matches perfectly our conceptualisation of trust. In the light of this, we define trust as:

Definition 1 (Trust) *Trust is the deliberate decision of an individual A (called* trustor*) to be in a situation of vulnerability towards the behaviour of an individual B (called* trustee*) with respect to an issue X (i.e.* trust issue*) and within a context C.*

This definition highlights the most important concepts that are inherent to any trust decision. The uncertainty of the situation that makes the *trustor* vulnerable towards the behaviour of the *trustee*. This decision is based on an *evaluation* and leads to a trusting *action*. It is this trusting action that brings the *trustor* to the situation of vulnerability.

Discussion

In this section, we presented some of the well-known definitions of trust. Then we used important elements of each definition to propose a generic one that best matches the meaning of trust in the context of virtual environments. What our definition does not say however, is how the trusting decision can be made. So if we consider trust as a physical concept, the question we did not answer up to now is what is the recipe of a trust decision. For instance, taking the bread recipe analogy, one can easily agree on the fact that almost everybody knows what the bread is and what are the various contexts it is used in. However, if we ask people to describe its form or the list the ingredients that are used to make it, they will probably not agree. This analogy describes well the dilemma of defining trust. To that aim, we considered trust as a decision which is fabricated based on several and various ingredients. Some of them have already been mentioned (e.g. reputation, competence, reliability) and others will be progressively introduce throughout throughout the thesis.

2.3 Trust Features

In Section 2.1 we reviewed different disciplines to see how each of them handled trust. Based on that review, we selected and studied the most relevant definitions. In this section, we adopt an extensional approach to have a more concrete analysis of its features. To that aim, we will get focus the *natures* in which it has been conceptualised, the *sources* from which trust is acquired, and *properties* that has been associated to it. We will rely on these features in Section 2.4 to analyse the trust models we reviewed.

2.3.1 Trust Nature

In this section, we identify four forms in which trust has been conceptualised in different disciplines.

2.3.1.1 Trust as a “Belief”

As highlighted in our study, philosophers consider trust as a belief that a person places on the statements of others [Deutsch and Gerard, 1955, Rotter, 1967, Deutsch, 2011]. In their conceptualisation they distinguished rational beliefs (those that can be proved) and non-rational ones. However, philosophers do not agree on which (rational and non-rational) beliefs should be considered to characterise trustworthiness.

2.3.1.2 Trust as a “Mental State”

Principally used by psychologists, trust is represented as a mental state resulting from a cognitive process [Castelfranchi and Falcone, 1998, Castelfranchi and Falcone, 2000a, Falcone and Castelfranchi, 2001]. Also known as *competence trust* [Kaur, 2011], it addresses trust form the perspective of perceived *capabilities*, *expertises* and *skills* of the concerned trustee. Consequently, in this conceptualisation only cognitive entities are able to trust; so software that are not endowed with beliefs and goals cannot exhibit trust. This vision is also shared by some sociologists [Castelfranchi and Falcone, 2000b]. Nevertheless, the mental state nature of trust is based on reasoning (knowledge-driven). Thus this form of trust joins somehow the previous one in recognising the importance of beliefs (e.g. competences) based on which trust is built.

2.3.1.3 Trust as a “Measure”

Also known as *calculative trust* [Shapiro, 1998, Lewicki et al., 2006], its has been a dominant type research in economics. Here the trust is considered as a threshold which is compared to a computed value (measure of trust). Research on trust gave rise to numerous different ways in which trust measures can be represented. For instance, some works (e.g. [Udhayakumar et al., 2011, Artz and Gil, 2010, Kim et al., 2009]) make use of 0 and 1 binary values to represent, respectively, trust and distrust. Even if such representation seems to be simple and easy to understand and implement, it suffers from a lack of expressiveness. Other approaches used a multi-valued evaluation (e.g. *very untrustworthy*, *untrustworthy*, *trustworthy* and *very trustworthy*). It is also possible to represent trust as a continuous variable (e.g. in $[0, 1]$ or $[-1, 1]$) where trust is generally treated as a probability or a weighted evaluation of evidence.

2.3.1.4 Trust as a "Relationship"

This type of trust constitutes the dominant point of view of sociologists towards trust. In this conceptualisation, trust is built over acquaintances and social bonds [Golbeck and Hendler, 2006, Singh and Liu, 2003, Grandison, 2003, Falcone and Castelfranchi, 2001]. In many situations this form of trust is not cognitive and cannot be explained or justified only by available beliefs. At most, it can be merely justified with past experiences which does not explain how cold-start and bootstrapping trust are addressed (i.e. no one knows how first trust decisions are made) [Burnett et al., 2010, Artz and Gil, 2010].

2.3.1.5 Trust as a "Decision"

The decision nature of trust has been essentially stressed in disciplines such as economics, politics and law [Ramchurn, 2004, Ramchurn et al., 2004a]. This kind of trust relies on the comparison of perceived risk against the potential gains for making the trust decision. However, a strict decisional interpretation of trust can be misleading in certain situations wherein trust is not the unique factor influencing the decision (e.g. social pressures, business considerations, etc.).

In our definition, we adopted this point of view by considering trust as a decision. However, what we wanted to exploit in the decision nature of trust is its pragmatism. This a decision stresses the final stage with make an individual go for a choice rather another one [Burnett et al., 2011, Venanzi et al., 2011].

2.3.2 Trust Sources

In the previous section, we evidenced the importance of beliefs with respect to trust, whatever the form in which it has been conceptualised. In the light of this, we present in this section another classification of trust based on the nature of the beliefs it relies on. Among the diverse beliefs involved in the trust issue, we distinguish three ways which an individual can use to acquire them resulting in three types of trust.

2.3.2.1 Internal Trust

Also known as *dispositional trust* as discussed in Section 2.1.2, *internal trust* relates to the internal beliefs/mechanisms that an individual relies on to trust others [Deutsch and Gerard, 1955, Rotter, 1967, Falcone and Castelfranchi, 2001]. These beliefs are personal and reflect the extent to which an individual is willing to trust others independently from the trust characteristics they exhibit. The disposition to trust also reflects the ability of an individual to rely on others. This kind of trust can somehow be related to the *belief trust* and *mental state trust* (cf. Sections 2.3.2.1 and 2.3.1.2).

2.3.2.2 Direct Trust

Direct trust is the first type of trust that individuals develop in their lives. This kind of trust is further divided into *strong direct trust* and *weak direct trust* [Khodyakov, 2007]. *Strong direct trust* develops among individuals that have social ties (e.g. family members, relatives, friends, colleagues, etc.), while *weak social trust* develops beyond boundaries of individuals' acquaintances. *Weak direct trust* develops based on previous positive interactions, while the basis of *strong direct trust* is familiarity and similarity [Mangematin, 1998]. Also known as *interpersonal* trust [Lewicki et al., 2006, Wieselquist et al., 1999, Weinstock, 1999], *direct trust* has been principally highlighted in sociology and social psychology fields in which trust was conceptualised as a relationship. Consequently, *direct trust* is like *dispositional trust* maintained locally by the trustor and represents the trustor's personal trust regarding the trustee [Saadi et al., 2011].

2.3.2.3 Indirect Trust

In contrast to *direct trust*, *indirect trust* characterises the situations in which the trustor has few or no prior interactions/information with respect to the trustee. In such situation, the trustor does not have other choices but relying on third party trust. This can be achieve in two different ways: *vertical* and *horizontal* trust [Lee and Yu, 2009].

Vertical Trust

Vertical trust is based on mechanisms provided by systems and institutions to facilitate the trust establishment between the trustor and the trustee. This kind of trust aims at generating trust when trustors and trustees are lacking interactions and reliable information about each other. Thus *institutional trust* is particularly interesting in situations of cold-start or for bootstrapping trust (i.e. unknown or little known entities) [Kaur, 2011].

Horizontal Trust

In *horizontal trust* the trustor uses the opinions of other individuals to assess the trust it can put in the trustee. *Horizontal trust* can be split into two sub-categories: *recommendation trust* and *reputation trust*. In *recommendation trust*, the trustee indirectly assess the trustworthiness of the trustor based on the recommendation of a group of several recommending individuals. Thus the trust that the trustor put in these recommending individuals is transferred to the trustee using the aggregations of their recommendations. In *reputation trust*, the trust is computed (i.e. reputation) for each individual and propagated among the users. Depending on the architecture of the mechanisms used to make reputation available to others, existing approaches can be split into *centralised reputation* (e.g. eBay, Amazon and Yahoo Reputation System) or *decentralised reputation* (e.g. [Zhou et al., 2008, Zhou and Hwang, 2007]).

2.3.3 Trust Properties

In this section, we analyse trust in terms of properties. Surprisingly, agreement about trust can easily be found when the discussion becomes more concrete and tackles trust properties. Of course, each property does not necessary apply to each definition [Gray, 2006].

2.3.3.1 Multidimensional

Trust is a concept that involves a *trustor*, the individual that is according his trust, and a *trustee* which represents the individual that is trusted [Grandison and Sloman, 2000, Grandison, 2003, Baier, 1986] (i.e. two dimensions). However, trust is in general never accorded independently from any purpose. Therefore, it is more likely admitted to represent it as a three-part concept wherein the *trustor* trusts the *trustee* for a particular *issue* X [Farrell, 2009, Hardin, 1982] (i.e. three dimensions). The advantage of this vision is that it shows the limited character of the trust that usually involves a particular activity, a particular role or a particular domain (although someone can be trusted for multiple activities, roles or domains). For instance, an individual Alice trusts another individual Bob with respect to a particular issue X (e.g. driving her at home). Finally, recent works (e.g. [Yew, 2011]) advocate the fact that trust is better captured when considering four dimensions: an individual A trusts an individual B with respect to an issue X in a context C (cf. Definition 1) (i.e. four dimensions). For instance, if Alice may trust more a taxi-driver in the day to driver her to the airport as he is more likely to driver her in time, however, she only trusts her friends to drive her by night. So here, the context affects negatively (the taxi-driver) or positively (for the friends) the trust that Alice grants to others.

Multi-factor

Trust factors refer to the beliefs, information, percepts and stimuli that affects positively or negatively trust. For instance, Jøsang considers trust as subjective phenomenon that is based on various factors such reputation [Jøsang et al., 2007, Bhuiyan and Jøsang, 2010, Huynh et al., 2006]. Here again, almost every researcher working on trust has their own idea about what trust is made of. However, even if they do not agree on what are the particular factors of trust, they still agree on the fact that these factors are numerous and different.

Measurable

In human societies, the trust that one puts in another is not explicitly measured (i.e. without explicit analysing of information, risk and context). However, when it comes to software programs, trust has to be represented, captured and thus measured in one way or another. For instance, Jennings colleagues defined trust as a "measurable level of the subjective probability [...]" [Huynh et al., 2006]. Marsh argued that "trust has no measurable units, but its value, its worthwhileness, can be measured. It is, thus, a commodity, like information and knowledge" [Marsh, 1994]. However, many scholars disagree with the fact that trust can be measured and thus do not make explicit the way trust is measured. Instead, they only consider the outcome of the measure based on which a trust decision is made.

Comparable

We mentioned in the previous section that for some scholars trust can/hase to be measured. This property suggests two important properties: (a) there should be a certain threshold, above which, an individual can say that something or someone is *trustworthy* [Grandison, 2003, Liu, 2011, Herzig and Lorini, 2010, Marsh, 1994] (e.g. to say that B is trustworthy), (b) an individual is able to compare two partners based on their trust values [Grandison, 2003, Herzig et al., 2008] (e.g. to say that B is more trustworthy than C), (c) one can compare the trust values of the same individual for two distinct issues or at two different moments (e.g. B is more trustworthy to view a Wikipedia page than to modify it).

Dynamic

Almost all researchers on trust agree on the fact that the trust that a trustor puts on a trustee during two different interactions may be different (cf. Section 2.3.3.1). Thus, one can logically imagine that if the trustee violated the trust he was granted in the first interaction, this must affect (negatively) the trust put in him for the second interaction. Trust may also change because the condition in which trust was initially accorded do not hold anymore. Thus trust is intrinsically dynamic. It decreases with negative experiences an increases with positive ones [Marsh, 1994, Grandison, 2003].

Subjective

The subjective nature of trust has been evidenced in almost all the definitions we presented in Section 2.2. Gambetta used for instance the expression "the subjective probability" in its definition. Jonker and Treur [Jonker and Treur, 2001] provided a comprehensive study about how different interpretations of observed evidence lead to different assessments, characterising the subjective nature of trust. Their work is somehow comparable to the one proposed by Axelrod in 1984 [Axelrod and Hamilton, 1981, Axelrod and Hamilton, 1984], and extended by Marsh in 1994 [Marsh, 1994]. In Axelrod's and Marsh's models, artificial agents interact with each other in a Prisoner Dilemma tournament according to subjective trusting strategies (optimist, pessimist, and realist for Marsh) and (14 strategies for Axelrod). The results of these research initiatives evidenced the important of the subjectivity in trust evaluation.

Trust is subjective for three reasons: (a) individuals making trust often use incomplete information which leads to subjective evaluations of trust, (b) trust is a based on various factors and evidence which vary from one individual to another, (c) even when too individuals use the same set of factors, some of them carry more weight than others. Typically, personal experience carries more weight than reputation, recommendations or others experiences. These reasons put together characterise the subjective nature of trust in almost all situations.

Transitive

In previous section, we discussed the dynamic character of trust which is affected by positive and negative experiences. However, in situations of previously unknown or little known partners such experiences may be lacking which makes trust evaluation difficult. In such situations, transitivity plays an important role to allow individuals building trust based on others' experience. The idea behind transitivity in trust is illustrated in Figure 2.1.

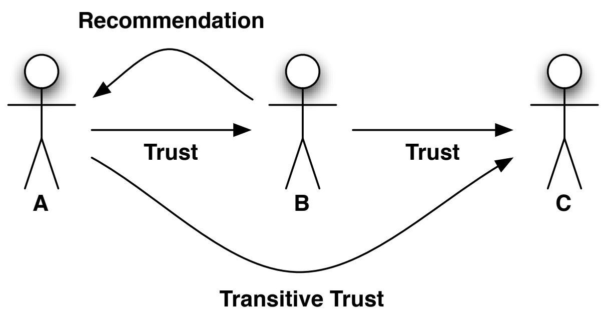


Figure 2.1 – Illustration of trust transitivity

In Figure 2.1, A trusts B, and B trusts C. So based on the recommendation of B to A about C, A finished by trusting C in combination with his trust in B. Many researchers however admit that trust is only conditionally transitive [Christianson and Harbison, 1997, Bhuiyan and Jøsang, 2010]. For instance, one can trust her doctor to make good diagnosis and her doctor trusts his mechanic to repair his car but one cannot trust her doctor's mechanic to make good diagnosis. Here, the trust one has for her doctor cannot be transferred to his mechanic which confirms that trust is conditionally transferable and that trust is context-aware.

Symmetry

Unlike transitivity, *symmetry* in trust is not always verified [Grandison, 2003]. Indeed, if all trust relationships were symmetric, then this would imply that trust relationships are always mutual which is not the case in almost all situations involving trust. Because individuals have different experiences, psychological backgrounds, and histories, it is understandable why two people may trust each other with different amounts [Golbeck, 2005]. For instance, an individual A may trust another individual B to drive him, but B is not obliged to trust A knowing that A trusts him. In this thesis, we prefer to talk about *unidirectional* trust and *bidirectional* trust. In fact, the *symmetry* property has an implied and misleading understanding which suggests that trust is *symmetric* or not for the same issue.

2.4 Trust Models Analysis

Trust has been extensively investigated in the last fifteen years which gave rise to numerous trust models. Most of these models have been developed in *distributed artificial intelligence* (DAI) or *security*. The objective of this section is not to review the complete literature on this subject, there exist several surveys that provided more comprehensive studies (c.f. [Jøsang et al., 2007, Grandison and Sloman, 2000, Ruohomaa and Kutvonen, 2005, Suryanarayana and Taylor, 2004, Artz and Gil, 2010, Krukow et al., 2008, Grandison, 2003]). Instead, our main concern is to classify representative models from each discipline and discuss the *nature* (cf. Section 2.3.1) of trust they model, based on which *sources of evidence* (cf. Section 2.3.2) and with what kind of *properties* (cf. Section 2.3.3). Following this analysis, we compare existing models with respect to the requirements we discussed in Section 1.3, namely *communicability*, *expressiveness* and *flexibility*.

2.4.1 Trust Models in Distributed Artificial Intelligence

Trust models in DAI fall into three categories; *probabilistic models*, *reputation models* and *socio-cognitive* models.

2.4.1.1 Probabilistic Models

Marsh's work [Marsh, 1994] on trust represents the first comprehensive and formal model of trust using a probabilistic approach. Marsh experimented in his doctoral thesis the use of trust as a basis for cooperation among autonomous agents. Starting from the assumption that "neither full trust or distrust are actually possible" in such systems, he proposed a complex calculation algorithm that computes a continuous value of trust based on several factors. This value represents the probability that A will behave "as if" he trusts B.

Manchala developed the first trust model that explicitly uses the notion of risk [Manchala, 1998]. Manchala specified a decision matrix in which he used the cost of the transaction, the expected outcome, and the history of the transactions. These factors are then used together in a probabilistic function to determine whether a transaction with a partner should be conducted or not. Manchala's risk-trust matrices are intuitive and simple to apply. The higher the value at stake, the more positive experiences are required to decide to trust [Jøsang, 2007].

Burnett and colleagues [Burnett, 2011], used recently the concept of stereotypical trust. In their approach, an individual uses machine learning methods to build stereotypes based on the agent past experience and the partners attributes. A stereotype represent a form of reputation that associates to a class of agents (characterised by their attributes) to the behaviour they tend to exhibit. These stereotypes are then used alongside with probabilistic model to make trust decisions. Previous to this approach, in 2009, Liu and colleagues [Liu et al., 2009] already used stereotypes in trust decision making. However, in their approach, they used the stereotypes to predict whether the undergoing interaction of an individual will be successful or not. More recently, Fang and colleagues proposed in [Fang et al., 2012] a new model in which they used fuzzy semantic decision tree (FSDT) learning methods to improve the ability of learning trust stereotypes based on only limited data about the partner.

Discussion

Probabilistic trust models consider trust as a measure that agents use to make trust decisions. This measure is computed based on direct experience of the individual making the trust evaluation. Probabilistic trust models consider trust as a concept involving a *trustor*, a *trustee* and an *issue*. Besides trust, these models take into account the importance of the interaction, the risk associated to the situation and other information such as the attributes of the interlocutor in [Burnett, 2011]. Thus trust is considered to be multi-factor. It is measured and can be compared. It also evolves over the time based on the experience of the individual. Trust is also subjective as it depends on the individual's one experience but is neither transitive nor asymmetric [Manchala, 1998]. Theses models are very *expressive* and partially flexible as they are able to adapt their evaluation using learning techniques. However, non of these models exhibits *communicability* features. In [Burnett, 2011], *argumentation* could be applied to allow agent to communicate, argument and agree on common stereotypes but to the best of our knowledge this feature is not supported [Burnett et al., 2013a].

2.4.1.2 Reputation Models

Reputation is the social evaluation of a group, a community or a society of agents towards the trustworthiness of an individual [Sabater and Sierra, 2001]. In DAI, and more particularly in multi-agent systems, reputation has been considered as a substantial dimension of trust [Jøsang and Ismail, 2002]. In the following, we review some predominant reputation models.

ReGreT [Sabater and Sierra, 2001] is a well known decentralised trust and reputation model for e-commerce. Proposed by Sabater and Sierra in 2001, the main objective of *ReGreT* was to make more accurate trust evaluations. To that aim, the authors used three factors based on which trust was computed: *the direct experience*, *the global reputation* and an ontological *fine-grained reputation* which defines reputation values for each trait of the individual using the ontology. In ReGreT, the network to which the agent belongs is used to assess the credibility of the information provided by each agent. Social relationship are presented in the form of fuzzy rules which are later used to determine whether the witness information provided by an agent should be considered or not.

Jøsang [Jøsang and Ismail, 2002] proposed a reputation model (called the Beta Reputation System) for the decision making in the context of e-commerce transactions. The authors used the concept of reliability along with the probability of success to determine the trustworthiness of a partner. The reliability of an individual is assessed in a direct and indirect way. The direct reliability is computed based on previous knowledge about the partner, while the indirect one is given by recommendation from other trust third party. The indirect reliability is then computed by making the average of all recommendation weighted by the recommender trust degree. Then this value is combined with the direct reliability in order to derive a trust degree. Once this trust degree obtained, it forms a belief that is described as set of fuzzy propositions such as "A believes that B is very trustworthy".

FIRE [Huynh et al., 2006] is another important model which has been designed by Huynh and colleagues for multi-agent systems. The authors compute trust based on past experiences, the role of the agent, its reputation and a kind of certified reputation. Roles are used to determine to which degree an agent that have a certain position in the society could be trusted. The main idea is that trust depends on the fulfilment of the role ascribed to the agent. Also, the authors make a distinction between witness reputation and certified reputation. Certified reputation is a reputation that comes from certified presumably trusted witness, while normal reputation comes from every agent of the society.

Vercouter and Muller proposed the LIAR model to process recommendations among multi-agent systems [Vercouter and Muller, 2010a, Vercouter and Muller, 2010b]. The authors consider trust as a multi-factor concept; they distinguished between the trust in a agent as a partner and the trust in an agent as a recommender. This latter form of trust is used to evaluate the witness information. The aggregation of witness information is performed using a weighted average, in which the trustworthiness degree of the recommender is used as the weight. Thus, the importance of witness information is proportional to the trustworthiness degree of the recommending agent.

Discussion

In reputation models, trust is considered as a measure which is directly or indirectly (horizontally) acquired. This measure reflects the evaluation of the society towards the trustworthiness of one individual.

Reputation models generally use the concepts of *reputation* and *trust* interchangeably. Thus, reputation is considered as an image of trust at the level of the society. In these models, trust is dynamic (evolves based on past experiences) and generic (the reputation of an individual is general and not purpose-specific). These models also rely on the *transitive* property of trust as they make use of direct and indirect experiences. reputation models assume an *unidirectional trust* but *bidirectional trust* is also supported as the trustee can evaluate the trustor before initiating the interaction.

Further, reputation models rely on the exchange of subjective evaluations to build reputation. However, these models do not communicate about *how* these evaluation are computed which makes *social compliance* hard to achieve. More recently, Koster [Koster, 2012], proposed a formalism to align reputation evaluations but this alignment is reduced to *translation* of *evaluation* without communicating about what information is used to make these evaluations. Moreover, reputation models are not very flexible as the scheme used to compute the evaluation is hard coded in the agents which makes it difficult to change.

Finally, the aggregation function used in these models is merely expressive; it relies massively on the past behaviour of the evaluated agent and ignores its intrinsic attributes (e.g. competence).

2.4.1.3 Socio-Cognitive Trust Models

The model proposed by Castelfranchi and Falcone is the prime trust model that explicitly stressed the importance of the socio-cognitive dimension of trust. They defined trust as a mental state based on which artificial agents can make delegation decisions within multi-agent systems. The authors consider trust as a combination of beliefs and goals that constitute this mental state. In a nutshell, an agent i trusts another agent j for doing the action α to achieve an objective φ iff [Krupa, 2012]:

- i has the objective φ,
- i believes that j is capable of doing α,
- i believes that j has the power to achieve φ by doing α,
- i believes that j intends to do α.

Recently, this model has been extended and formalised in the context of the *ForTrust* project [Herzig and Lorini, 2010, Hübner et al., 2009b]. This new model has been used in the context of detecting malicious (e.g. vandalism) contribution to Wikipedia [Krupa et al., 2009].

2.4.1.4 Discussion

In socio-cognitive models, trust is considered as a *mental state* which originates from an *internal* cognitive process. This process makes use of several types of beliefs (e.g. competence, dependence or willingness) to assess whether a trustor can trust a trustee about a particular issue (e.g. doing action α to achieve goal φ). In these models trust is *not explicitly measured* and thus is *not comparable*. Trust is *dynamic* but does not evolve based on past experience. Trust is also *unidirectionally* established as the process is performed by the agent delegating the action α and does not trigger the same process in the agent to which the action has been delegated (i.e. agent j).

Furthermore, socio-cognitive models are *not very expressive too* as the set of beliefs based on which trust is built are limited and known (i.e. competence, willingness, disposition, etc.).

For instance, the agent intrinsic attributes and its past experience are not considered at all. In addition, socio-cognitive models do not support *communicability* as the cognitive process used to derive trust is embodied in the agent. Consequently, this process is very hard to change which makes these models *not very flexible*.

2.4.2 Trust Models in Security

With the advance of Internet, the objective of researchers on security was to propose *decentralised* access control mechanisms to leverage the distributed nature of these systems. The general idea was to allow resource owners to state *who* they trust and for *which* issue. Based on this idea, several systems (called Trust Management Systems) have been proposed. We classify these models into two categories: *Decentralised Trust Management (DTM)* and *Automated Trust Negotiation (ATN)*. In the following, we present some works from both approaches. This review is deliberately made succinct as most of the works presented hereafter will be presented in more details in the next chapter.

2.4.2.1 Decentralised Trust Management Models (DTM)

Blaze and Lacy were the first to use the word *trust management* in the security domain [Blaze et al., 1999b, Blaze et al., 1996]. The authors justified the use of trust in reference to the delegation mechanisms they used to address distribution in modern systems (e.g. Internet). Credentials (digitally signed documents) are used to allow an individual to express the trust relationship it has with another individual with respect to a particular issue (e.g. accessing a resource). For instance an individual A may issue a credential stating that he trusts another individual B to access a resource R he owns. Subsequently, B can express the trust he puts in another individual C with respect to the same issue. Now if C requests to A the access to R, he will provide the credential by B and the objective of the model is to evaluate whether there is a valid trust (delegation) chain from A to C about accessing the resource R (cf. Chapter 3).

Grandison and Sloman proposed the SULTAN (Simple Universal Logic-Oriented Trust Analysis Notation) trust model [Grandison and Sloman, 2003]. This model was developed with the objective to support secure interactions in Internet applications. Typically, decisions derived using SULTAN are about the evaluation trust (and distrust) relationship, which has been used as a basis for developing distributed access control schemes (e.g. allowing users to access sensitive resources). SULTAN makes use of recommendation and past experience which are filtered based on rules (i.e. policies) stated by the user. The trust decision is then made based on this evaluation along with a risk approximation.

Kagal and colleagues [Kagal et al., 2003] proposed a trust model for distributed security in the context of multi-agent systems. They proposed a flexible representation of trust relationships that they express in the form of *permissions* and *delegations*. Later, they extended their model to integrate three new forms of trust relationships called *obligations*, *entitlements* and *prohibitions*. Concretely, the notion of trust used by the authors is purely *interpersonal* as it express the degree of privilege that one individual is willing to accord to its interlocutor. Kagal and colleagues described a scheme for representing delegation and restricting re-delegation among communities.

Discussion

In DTM, trust is represented as a *binary* evaluation. This evaluation is based on the the existence or not of a *trust relationship* between the trustor and the trustee about a particular issue (i.e. the action to be granted). Trust is *multi-factor* but only one type of information is used; a *key* or a *delegation credential*. Trust can be *measured* but its evaluation is binary making the comparison very hard to achieve. Trust is *not dynamic* as the policies used are not affected by the behaviour of the trustee. However, it is possible to make it dynamic by revoking credentials or by updating policies. Trust is *subjective* as existing trust relationships are established based on personal criteria. Further, *transitivity* is a cornerstone property for these models which is materialised by delegation and credentials concepts. Finally, trust is *unidirectionally* established from the trustor to the trustee. So the trustee is assumed to trust the trustor. However, in many situations the credentials disclosed during trust establishment are sensitive.

These models are *not very expressive* as the condition stated in the policies are generally limited to identity and delegation credentials. However, they are relatively *flexible* as the policies can be (manually) updated without stopping the system. Finally, thanks to the concept of policies, these models fully address the *communicability* requirement. In addition, the formal and even standard (some policies are specified in XML (cf. Section 3.3.2) makes these models easy to understand for both human and artificial agents.

2.4.2.2 Automated Trust Negotiation Models (ATN)

Credentials which are implemented in DTM as identity or delegation certificates are used in trust negotiation systems to convey the identity and the attributes of the holder. So releasing a credentials imply the disclosure of such sensitive information. To that aim, Winslett and colleagues [Yu et al., 2003] introduced the concept of *trust negotiation*. Trust is established through the gradual, iterative, and mutual disclosure of credentials and access control policies. This model has been proposed to leverage *privacy* issues that may arise when the disclosed credentials are sensitive. The authors build on exiting *decentralised access control models* to allow bilateral establishment of trust.

Trust-$\mathcal{X}$ is a trust management system that was designed for trust negotiation in peer-to-peer systems [Bertino et al., 2003, Bertino et al., 2004]. The Trust-$\mathcal{X}$ engine provides a mechanism for negotiation management. The main strategy used in Trust-$\mathcal{X}$ consists in releasing policies to minimise the disclosure of credentials. So only credentials that are necessary for the success of a negotiation are effectively disclosed [Squicciarini et al., 2007]. Trust-$\mathcal{X}$ provides also a mechanism to protect sensitive policies. Another novel aspect of this system consists in the use of trust tickets. Trust tickets are issued upon successful completion of a negotiation. These tickets can later be used in subsequent negotiations to speed up the process in case the negotiation concerns the same resource.

Bonatti and colleagues [De Coi et al., 2008] proposed a flexible and expressive negotiation model called *PROTUNE* (*PRovisional TrUst NEgotiation*). This framework is a system that provides distributed trust management and negotiation [Bonatti et al., 2010, Bonatti and Samarati, 2002] features to web services.*PROTUNE* consists in a policy language (based on PSPL [Bonatti and Samarati, 2000]) and an inference engine based on (PeerTrust [Nejdl et al., 2004]). The most innovative aspect of PROTUNE relies in its high degree of expressiveness. One of the main advances made by PROTUNE lies in the use of *declaration* along with credentials during the policy evaluation process. *Declarations* are the unsigned equivalent of credentials. They can also be considered as statements that are not signed by a certification authority. However, the most novel part of the project remains the policy specification language with combines access control and provisional-style business rules.

Discussion

ATN systems constitute the last generation of trust management systems. Therefore they inherited most of their advantages and some of their disadvantages. Like their ancestors, these systems consider trust as a decision which is made based on a *binary evaluation*. However, unlike DTM systems, trust in these systems is *built incrementally* and *bilaterally*. Also, trust is a *multi-factor* concept as all types of information can be used (including reputation and recommendation [De Coi et al., 2008]), thanks to the fine-grained policy language proposed in these systems.

ATN systems assume a *dynamic* trust which is gained through negotiation. It is also *transitive* as they rely on credentials and recommendation. Trust is difficult to compare as the outcome of the evaluation is binary. It is *subjective* as each individual makes use of its personal policy to make trust decisions. As these systems inherit from DTN systems they inherits their *communicability* and their *flexibility*. But ATN systems are more *expressive* as the policies used in these systems have been designed to enhance existing policies expressiveness [Seamons et al., 2002].

2.5 Discussion

Distributed artificial intelligence and *security* tackled trust in a distinct but complementary way. This distinction is explained by the difference among these disciplines in terms of objectives and tools.

DAI exploited the agent's ability to reason and communicate about the properties of their partners [Sabater and Sierra, 2005, Ramchurn et al., 2004a, Ramchurn et al., 2004b, Yew, 2011]. Based on this ability, researchers tried to reproduce/implement/extend models provided by flagship disciplines working on trust (i.e. philosophy, psychology, sociology and economics) [Yew, 2011]. These models specify *how* to compute the amount of trust an agent can place in its interlocutor, and *what* are the information this agent should gather to make such computation [Vercouter and Muller, 2010b]. Information are obtained from the agent past experience and observation (e.g. probabilistic models) or indirectly acquired based on other agents testimonials (e.g. reputation models). This information is then used with respect to a predefined scheme (e.g. aggregation function,) to derive a trust measure. This scheme is hard-coded in the agents and used by all agents of the system [Ramchurn et al., 2004a].

In contrast, the main concern of the models that have been proposed in security was the design of mechanisms to allow secure transmission of knowledge based on which the trust decision (e.g. authorisation) can be made. The transmitted information represent encrypted statement (i.e. credentials) that prove the identity, the rights and/or the attributes of their holder. The particularity of this approach is the use of *Trust Third Parties* (TTP) which are the entities that issues, validates and revokes that statements. As their names indicates, these applications are presumably trusted and aims at bootstrapping trust in the system. Finally, each individual makes use of an explicit and individual scheme (i.e. policies) that determines which information the *trustee* must provide to be trusted.

The remainder of this section does not further discuss the difference between the models proposed in each discipline, we assume that the discussions that followed each category of model suffice to stress these models properties. The result of these discussions is summarised in Table 2.1. Instead, we will focus on how the models proposed in each discipline addressed the requirements we identified in Section 1.3.

Communicability

The use of *policies* makes systems using *security models* able to communicate about the information they require to make trust decisions. These features are particularly interesting to enable *bidirectional trust*. These features also eases the process of collective decision making as the members of the same community are able to communicate about the policies they use.

In contrast, systems using *DAI models* are embodied with the scheme which is implemented as a function (e.g. aggregation, learning, stereotyping, etc.). Therefore, we consider these systems as *non communicative* as agents cannot communicate about the scheme they used, and thus will not be able to agree about a unique scheme they should use to make collective decisions.

Expressiveness

We measure the expressibility of a model based on two indicators: (a) the range of information based on which trust is computed, and (b) the way these information are aggregated in the evaluation process. With respect to the first indicator, *DAI models* and *security models* have similar expressiveness degree. While *DAI models* rely massively on *direct* and *indirect* experience, *security model* rely on the trustee's attributes. However, recent models in security advocates a *hybrid* approach in which all kinds of information (including those used in *DAI models*) are considered during the trust evaluation (e.g. PROTUNE).

Besides, with respect to the second indicator (i.e. aggregation mode), *security models* are clearly not very expressive [Sabater and Sierra, 2005]. Policies evaluation results in a binary measure and the condition stated by the policy are considered to be of equivalent importance. In contrast, *DAI models* offers a better expressiveness as the trust evaluation is nuanced using weights and weighted sums [Marsh, 1994, Saadi et al., 2011].

Flexibility

The ad-hoc functions used in *DAI models* constitute a strength for these models in terms of expressiveness. However, these functions constitute also a limit to *flexibility* of these models. Therefore, we consider these systems as partially flexible as they are not able to easily adapt to unpredictable changes. For instance, if one of the information used when computing the reputation becomes obsolete, the whole system should be stopped and each agent using the reputation model should be re-factored in consequence. Such an approach is obviously inadequate for large and dynamic systems such as virtual communities [Ruohomaa and Kutvonen, 2005].

At the opposite, *security models* make use of explicit policies which states what information should be used to derive trust. This approach is more *flexible* as the policies can be separately updated.

2.6 Conclusion

In this chapter we studied and analysed the trust issue from five different disciplines, namely philosophy, psychology, economics, sociology and computer science. We have deliberately started with such a broad area in order to demonstrate the multi-facet nature of this concept [Ramchurn et al., 2004b]. Then we reviewed and selected interesting definitions based on which we proposed a new one that best serves our objective. Further, we used our literature review to extract trust properties and types based on which we compared some of the models that have been proposed in the computer science community [Wehmeyer and Riemer, 2007, Ruohomaa and Kutvonen, 2005, Grandison, 2003]. Finally, in the last section we discussed the benefit and limits of the reviewed models (grouped into categories) with respect to the requirements we identified in the introduction of this manuscript (cf. Section 1.3).

In the light of this study, we decided to tackle the trust issue from a *security perspective*, and more particularly using a *automated trust negotiation model (ATN)*. This choice is made with respect to *communicability* and *flexibility* which are better supported in these models compared to the models proposed in *DAI*. *Communicability* and *flexibility* are core requirements without which none of the objectives identified in Section 1.3 would be satisfied. However, as evidenced in the previous section, much work has to be done to reduce the gap between the expressiveness degree of *security models* and the one of *DAI models*. This observation will be further detail in the next chapter where we analyse the *trust management literature* with more details.

Models	Classification		Properties	Requirements		
	Nature	Source		CMC	EXP	FLX
Probabilistic	Measure	Direct	MD(4), MF, MS, CP, SJ, DY	-	+	±
Reputation	Measure	Direct, Horizontal	MD(2), MS, CP, SJ, DY, TR	-	+	±
Socio-cognitive	Mental-State	Internal	MD(3), DY, TR	-	-	-
Decentralised TM	Relationship	Vertical	MD(3), SJ, TR	+	-	
Automated TN	Decision	Vertical, Horizontal	MD(3), MF, SJ, DY, TR, SY	+	±	+
Desired Features	Measure	Direct, Indirect	MD(4), MF, MS, CP, SJ, DY, TR, AS	+	+	+

Table 2.1 – Analysis of the review models

- MD: multidimensional in ter
  - MD(2): a trustor and a trustee
  - MD(3): MD(2) + an issue
  - MD(4): MD(3) + a context
- MF: multi-factor
- MS: measurable
- CP: comparable
- SJ: subjective
- DY: dynamic
- TR: transitive
- SY: symmetric
- CMC: communicability
- EXP: expressiveness
- FLX: flexibility

2.7 French Summary

Dans le monde réel comme dans le monde virtuel, la *confiance* est un concept fondamental pour les humains sans lequel ils ne peuvent ni agir ni interagir. Ainsi, c'est sans grande surprise que ce concept a été l'objet d'intenses travaux de recherche depuis des décennies, voire des siècles. Le chapitre 2 de ce manuscrit avait donc pour objectif de clarifier ce concept et de présenter comment est-ce qu'il doit être compris tout au long de ce manuscrit. Pour cela, nous avons dans un premier temps examiné comment la confiance a été étudiée et interprétée dans les *sciences humaines et sociales.* Ensuite, nous avons circonscrit notre étude aux travaux menés dans le domaine informatique.

2.7.1 Étude rétrospective sur la confiance

L'objectif de cette section n'est évidemment pas de proposer une revue exhaustive de l'ensemble des travaux réalisés sur la *confiance.* Cependant, il nous paraît essentiel de présenter comment les disciplines phares des *sciences humaines et sociales* telles que la *philosophie*, la *psychologie*, *la sociologie* et l'économie se sont appropriés ce concept dans le but de pouvoir, à notre tour, le définir, le caractériser et le classifier.

En **philosophie**, la recherche sur la confiance à une longue tradition qui remonte à la Grèce antique. Durant cette période, les philosophes grègues (e.g. Socrates, Plato and Aristotle) on menait une profonde réflexion sur ce qui motivait les individus à recourir à la confiance. Leurs conclusions suggéraient que la confiance est une forme de croyance qu'il fallait exclure du mécanisme de décision en faveur de la rationalité. Ce n'est qu'au 20ème siècle que des philosophes tel que *Kant* [Kant, 1788] commencent à admettre le rôle de la confiance en complément de la rationalité.

En **psychologie** la confiance est expliquée au travers de deux processus cognitifs: (a) le sentiment de vulnérabilité, et (b) les attentes vis-à-vis dû comportent du partenaire. Cela explique pourquoi la confiance a été abordée de deux manières au sein de cette discipline. D'une part la confiance *disproportionnelle* (cf. [Deutsch, 2011, Deutsch and Gerard, 1955, Rotter, 1967]) qui permet de capturer la prédisposition des individus à faire confiance, et d'autre part la confiance *interpersonnelle* qui caractérise l'état mental qui mène un individu à faire confiance à un autre individu à un moment donné et pour un objectif donné [Rempel et al., 1985, Simpson, 2007].

La confiance est clairement un phénomène social, c-à-d que la confiance n'intervient que lorsque deux individus ont besoin d'interagir. De ce fait, la **sociologie** est incontestablement la discipline la plus prolifique sur ce sujet. Pour de nombreux sociologues, la vie moderne est trop complexe pour les mécanismes de décision de l'humain et notamment la rationalité. Cela explique en grande partie le rôle et l'intérêt de la confiance en tant que catalyseur et simplificateur des relations interpersonnelles. Les sociologues (tels que Zucker [Zucker, 1986]) distinguent la confiance dans les individus et la confiance dans le système. La *confiance dans les individus* provient de l'expérience qu'on acquiert après nos interactions avec les autres alors que la confiance dans le système est assurée par la structure à laquelle l'individu appartient.

Par exemple, la confiance que j'accorde à mon médecin provient essentiellement de la confiance que j'accorde à l'état et l'université pour former des personnes compétentes.

En **économie**, les chercheurs ont longtemps refusé d'utiliser la confiance pour expliquer les décisions humaines. En effet, pour les économistes l'humain (*l'homo-economonicus*) est purement rationnel et l'existence d'intérêts réciproques ainsi que la ratio risque/opportunité suffisent à expliquer les décisions de coopération. Néanmoins, beaucoup n'hésitent pas à évoquer la confiance notamment lorsque le calcul rationnel est rendu impossible par le manque d'information. Ces chercheurs parlent de lubrifiants des prises de décisions ou d'institution invisible mais le concept est le même et l'idée de la considération de la confiance a pris son chemin dans la discipline.

Enfin, en **informatique**, la nécessité de la confiance apparaît avec l'avènement des systèmes distribués tels que l'Internet. Dans de tels systèmes, la difficulté à obtenir des informations complètes rend les interactions au sein de ces environnements incertaines et risquées ce qui motive le recours à la confiance.

La confiance numérique, ou la confiance virtuelle en référence aux systèmes dans lesquelles elle est mise en valeur, a été énormément étudiée dans le domaine de la *sécurité* et celui de l'*intelligence artificielle* où l'incertitude et le risque représente un enjeu majeur. Dans ces disciplines et en informatique de manière générale, définir ce que la confiance veut dire (ce que les autres disciplines ont essayé de faire) n'est clairement pas suffisant car l'objectif final est de permettre l'automatisation de ce mécanisme. En effet, pour être utilisée par des programmes, la confiance doit être conceptualisée et représentée de manière formelle et non ambiguë. Par formelle, nous entendons que la confiance doit être capturée, représentée et manipulée par des programmes informatiques. Par conséquent, l'essentiel des travaux sur la confiance dans le domaine informatique visait à formaliser, implémenter et évaluer des modèles et des théories proposées par des chercheurs des disciplines abordées plus haut.

En conclusion, malgré leur diversité, ces disciplines sont d'accord sur la centralité de la confiance dans les interaction réelles et virtuelles. Cependant, aucune définition consensuelle n'a été proposé. En effet, on pourrait aisément imaginer la difficulté à concilier les avis divergents en une seule définition. Afin d'illustrer cette difficulté, prenons l'exemple des sites de commerce électronique (e-commerce). Dans chaque transaction, l'acheteur doit faire confiance au vendeur pour qu'il garde les informations qu'il a utilisé pour le paiement sécrétés et que les produits achetés satisfont ses exigences en terme de qualité. Et en contre-partie, le vendeur doit faire confiance en l'acheteur pour que celui ci n'utilise pas des moyens de paiement volés et que celui ci ne va pas l'évaluer négativement si tout se passe bien. Dans ce scénario, la confiance est inhérentes à toute interaction, tandis que que si l'on demande à l'acheteur et au vendeur leur définition de la confiance, la définition avancée sera probablement différentes car les raisons pour lesquelles chacun fait confiance sont différentes.

2.7.2 Sur les traces de la confiance

Dans cette section nous allons essayer de définir le concept de confiance du point de vue informatique. Pour cela, nous allons tout d'abord débattre et clarifier certains concepts qui sont souvent utilisés, à tort, en tant que synonymes de la confiance. Puis nous présenterons quelques définitions et discuterons leur concordance avec notre interprétation de la confiance. Enfin, nous conclurons cette section en proposant notre définition.

Ce que la confiance n'est pas: afin de comprendre ce que la confiance est réellement, il faut avant tout commencer par écarter ce qu'elle n'est pas. Ainsi, dans la littérature on trouve souvent des concepts qui sont utilisés en tant que synonymes de la confiance alors qu'ils ne le sont pas. Nous citons ici à titre d'exemple quelques-uns de ces concepts. **Être digne de confiance** n'est pas la confiance elle-même. En général, pour que la confiance s'établisse le partenaire se doit de se montrer digne de confiance. Ainsi, être digne de confiance est un prérequis de la confiance mais il ne peut être considéré comme un synonyme. **Fiabilité** et confiance sont également souvent confondus. La fiabilité est la capacité d'un individu à se comporter tel qu'il est supposé le faire. Par exemple, les policiers sont censés être fiable et pourtant je peux avoir recours à un policier sans lui faire confiance. Ainsi, la fiabilité n'est qu'un ingrédient de la confiance qui n'est souvent pas obligatoire. D'autres concepts tels que la **crédibilité**, la **compétence** ou la **réputation** ont été confondu et donc utilisés à tort comme synonyme de la confiance.

Alors c'est quoi la confiance au juste ? pour Gambetta [Gambetta, 2000] la définition de la confiance repose sur deux concepts qui sont la probabilité subjective et les conséquences de la décision. Pour Gambetta, la confiance est "un niveau particulier de la probabilité subjective avec laquelle un agent accomplira une action spécifique, à la fois avant que nous ne puissions suivre chaque action (ou indépendamment de sa capacité de même pouvoir la tracer) et aussi dans un contexte dans lequel cela affecte notre propre action.". De son coté, Luhmann [Luhmann, 1990] met l'accent sur la complexité du monde dans lequel nous vivons. Ainsi, pour l'auteur "la complexité du monde future est réduite par l'action de la confiance".

Par ailleurs, dans le domaine informatique, les définitions proposées par McKnight [Mcknight and Chervany, 1996], Grandison [Grandison, 2003] et Castelfranchi [Castelfranchi and Falcone, 2000b, Falcone and Castelfranchi, 2001] sont souvent citées. Chacune de ces définitions à fait avancer notre analyse et compréhension du concept de confiance. Cependant, aucun ne reflète fidèlement notre interprétation de la confiance. C'est pour cela que nous définissons la confiance comme étant :

Definition 2 (Confiance) *La décision délibérée d'un individu A (le confiant appelé également trustor) dans une situation de vulnérabilité vis-à-vis d'un individu B (dépositaire appelé trustee) par rapport à un objectif X et dans un contexte C.*

2.7.3 Analyse des modèles de confiance

Dans cette section, nous allons présenter, analyser et comparer quelques modèles de confiance. Pour cela, nous nous sommes intéressés aux travaux proposés dans les disciplines phares qui sont la *sécurité* et *l'intelligence artificielle*.

Les modèles de confiance étudiés dans l'intelligence artificielle peuvent être classés en trois catégories: les modèles **probabilistes**, les modèles **de réputation** et les modèles **socio-cognitives**. Dans ce qui suit, nous décrivons succinctement chaque catégorie.

- **Modèles probabilistes:** initiés par Marsh [Marsh, 1994], ces modèles assument qu'une confiance absolue ne peut exister. Pour cela, ils proposent de calculer une valeur de confiance en utilisant un algorithme qui agrège plusieurs facteurs de confiance. Cette valeur représente la probabilité que le confiant A accorde sa confiance au dépositaire B. Plusieurs modèles se sont inspirés de cette approche, on note par exemple l'approche Manchala [Manchala, 1998] dans laquelle la notion de risque est explicitement considérée comme étant un facteur de la probabilité de confiance. Plus récemment, Burnett et ses collègues [Burnett, 2011] proposent l'utilisation de stéréotypes qui sont en réalité des probabilités de comportement appris tout au long des interactions.

- **Modèles de réputation:** la réputation est l'évaluation de la fiabilité (dans le sens digne de confiance) d'un individu par un groupe, une communauté ou une société d'individus. En intelligence artificielle, et plus particulièrement dans la communauté multiagent, la réputation a été considéré comme une dimension primordiale de la confiance [Jøsang and Ismail, 2002]. À ce titre, de nombreux modèles ont été proposés dans la dernière décennie dont l'objectif était d'améliorer l'évaluation, la propagation et la mise à jour des mesures de réputation. Par exemple, Regret est l'un des premiers modèles à base de réputation dans lequel les auteurs associaient un degré de confiance non seulement à l'individu évalué mais également aux évaluateurs ce qui permet de pondérer leurs évaluations en fonction de leur degré de confiance. Dans le Beta Réputation System [Jøsang and Ismail, 2002], Jøsang utilise un modèle en logique floue pour calculer la réputation en se basant sur les recommandations directes et indirectes. Dans FIRE, Huynh et ses collègues [Huynh et al., 2006] introduisent les rôles afin de filtrer au mieux les évaluations et augmenter la précision de la réputation. Enfin, dans LIAR, Vercouter et Muller [Vercouter and Muller, 2010a] distinguent entre la confiance d'un individu en tant que partenaire est sa confiance en tant que fournisseur de recommandation. Cette dernière mesure est utilisée ensuite pour filtrer les évaluations et pondérer la confiance.

- **Modèles socio-cognitifs:** Le modèle proposé par Castelfranchi et Falcone [Castelfranchi and Falcone, 2000b, Falcone and Castelfranchi, 2001] est l'un des premiers modèles computationnels à considérer la dimension cognitive de la confiance. Comme vu précédemment, les auteurs considèrent la confiance comme étant un état mental que l'individu construit à partir de faits, croyances et d'objectifs. Pour cela, ils ont construit un modèle formel dans lequel ces éléments sont utilisés pour déduire ou non une relation de confiance. En résumé, un individu i est susceptible de faire confiance en un autre individu j pour faire l'action α afin de réaliser l'objectif φ si et seulement si:

- i a comme objectif φ,
- i croit que j est capable de réaliser α,
- i croit que j a la capacité d'atteindre φ en réalisant α,
- i croit que j à l'intention de faire α.

Ce modèle a été récemment formalisé dans le contexte du projet ForTrust [Herzig and Lorini, 2010, Hübner et al., 2009b].

En sécurité, l'usage de la confiance a commencé au début du 20ème siècle avec l'avènement de l'Internet. L'objectif des chercheurs de la communauté était de trouver mes mécanismes permettant une gestion distribuée et décentralisée du contrôle d'accès. Pour cela, il fallait permettre aux propriétaires de ressources de spécifier en *qui* il avait confiance pour manipuler leurs ressources et dans *quelles circonstances*. Dans cette perspective, plusieurs systèmes (appelés Systèmes de gestion de la confiance) ont été proposé. Nous classons ces systèmes en deux catégories: *Systèmes de gestion de la confiance Décentralisée (GCD)* et *Systèmes de négociation automatique de la confiance (NAC)*.

- **Systèmes de gestion de la confiance Décentralisée:** Avec leur système PolicyMaker, Blaze et Lacy constituent les pionniers de ce type de systèmes [Blaze et al., 1999b, Blaze et al., 1996]. Les auteurs utilisent le concept pour justifier le recours aux mécanismes de délégation qu'ils ont mis en place afin de permettre un contrôle d'accès distribué et décentralisé. Les certificats (credentials) sont utilisés pour exprimer la relation de confiance qu'a un individu avec un autre individu. Le système repose sur ce graphe de délégation pour déduire les droits qu'un individu possède dans un système donné. Dans SULTAN, Grandison [Grandison and Sloman, 2003] intègrent l'utilisation des recommandations en plus des certificats. Des politiques sont exprimées sous forme de règles afin de filtrer les recommandations en fonction des besoins de l'utilisateur. À l'image de SULTAN, de nombreux systèmes emboîtent le pas à PolicyMaker et suivent le même mécanisme de fonctionnement. Commun à ces systèmes, l'évaluation de la confiance est toujours binaire car elle est basée sur l'existence ou non d'une relation de confiance prouvée.

- **Systèmes de négociation automatique de la confiance:** les certificats utilisés dans les systèmes de gestion de la confiance Décentralisée sont utilisés pour certifier les propriétés de l'individu demandant l'accès à la ressource. Or, ces certificats contiennent des informations sensibles et les donner à un interlocuteur implique un risque. C'est pour cela que Winslett et ses collègues [Yu et al., 2003] ont proposés une nouvelle génération de systèmes de gestion de la confiance dans lesquelles une négociation automatique permettait de rationaliser la diffusion de ces certificats. L'objectif était d'empêcher la diffusion de certains certificats durant une négociation si cela n'était pas nécessaire ou si l'interaction était vouée à l'échec. Avec l'engouement que connaît la société de nos jours pour le respect de la vie privée, ces systèmes ont de plus en plus de succès ce qui explique l'explosion de leur nombre. À titre d'exemple nous citerons *TrusBuilder* [Yu et al., 2003], $Trust - mathcalX$ [Bertino et al., 2003, Bertino et al., 2004] et plus récemment *PROTUNE* (*Provisional Trust Negotiation*) [De Coi et al., 2008].

Chapter 3

Trust Management Systems

The concept of trust has been recognised in Chapter 2 as an interaction enabler in situation of risk and uncertainty. Based on this concept, several models have been proposed in distributed artificial intelligence and in security. This chapter aims at analysing and identifying the basic elements participating to the management of Trust in Trust Management Systems. The choices that will be made in the light of the requirements and objective stated in the introduction (cf. Section 1.3).

In order to better position the different works realized in Trust Management, this chapters propose an historical view on the evolution of such systems from Access Control to a Trust Management approach with respect the consideration of openness and decentralization. From that first characterization, we provide the reader with the fundamental background concepts supporting this approach. Section 3.3 includes a survey in which the most relevant trust management systems are presented providing an overview of the existing systems. The systems are presented chronologically in order to stress the contribution provided by each system with respect to its predecessors. The systems are also structured by grouping similar systems into three categories: *authorisation-based*, *role-based* and *negotiation-based*. This section is followed in Section 3.4 by a discussion in which we compare the reviewed systems.

3.1 From Access Control to Trust Management

Trust management has its roots in security and more particularly in *access control*. *Access control* (AC) is the traditional mechanism by means of which software applications (originally operating systems) answer the question (i.e. request) “is the entity identified as being S can manipulate the object O via the action A?”. Here the verb “can” should be understood in term of rights and not in terms of capabilities. Further, as one may notice, this question can be easily contextualised with respect to the trust issue into “Can I trust S enough to allow him performing the action A on the object O?”. In this section, we present the different models that have been proposed to answer such questions. Before we proceed, however, we introduce basic terminology we will rely upon in our descriptions.

3.1.1 Access Control Model

Access control is the mechanism used by applications to determine who can be allowed to manipulate local resources. The abstract model of an access control mechanism is depicted in Figure 3.1.

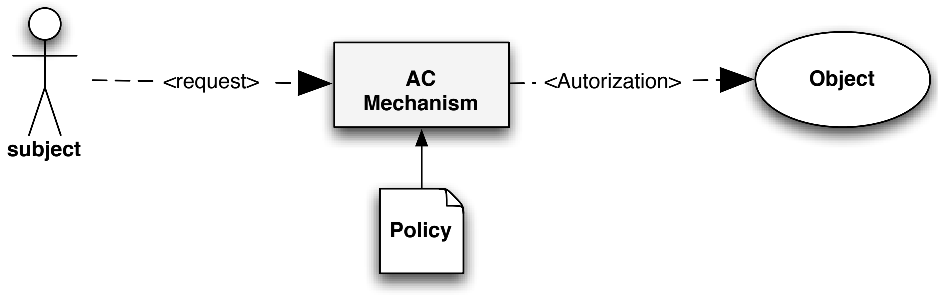


Figure 3.1 – A basic access control model (adapted from [Genovese, 2012])

- The *request* represents the type of interaction for which an authorisation is requested (e.g. read, use or login),

- The *subject*, often called *principal*, is the abstract entity (a human, a program or an artificial agent) requiring authorisation,

- The *object*[1] is the abstract artefact representing the resource the requester wants to interact with (e.g. a file, a service),

- The *mechanism* is the scheme that determines if the requesters is authorised to perform the requested interaction.

Thus, the access control mechanism plays the role of guard for the manipulation of sensitive local resources. It determines if a requester should or should not be authorised to manipulate the resource he requested. The specification of the information based on which a requester can be authorised represent permissions and is usually encoded in an *access control policy*. Based on the nature the policies used to specify permissions, a wide range of mechanisms have been proposed over the past decades to address the access control issue. These mechanisms can however be grouped into five categories: *identity-based*, *lattice-based*, *role-based*, *organisation-based* and *attribute-based*. The next sections provides a insight on how each mechanism addressed the access control issue.

[1]In the remainder of this document, we will use interchangeably the terms subject and principal. We will do so for the concepts of resource and object too.

3.1.2 Identity-Based Access Control

The general access control problem is often split into two main sub-problems: *authentication* and *authorisation*. *Authentication* aims at proving that the identity claimed by a principal is authentic, while *authorisation* tries to find whether there is a permission that allows this identity to manipulate the requested resource, and how this manipulation can be done. Identity-based Access Control (IBAC) has made implicit use of a closed world model, in which users and resources are *a priori* known. Under such assumptions, the problem of authorisation reduces to the one of authentication. Consequently, permissions to access a resource are directly associated with the principal's identifier (e.g. user name, login, public key) as illustrated in Figure 3.2. Access to the resource (an object in Figure 3.1) is only possible when such an association exists.

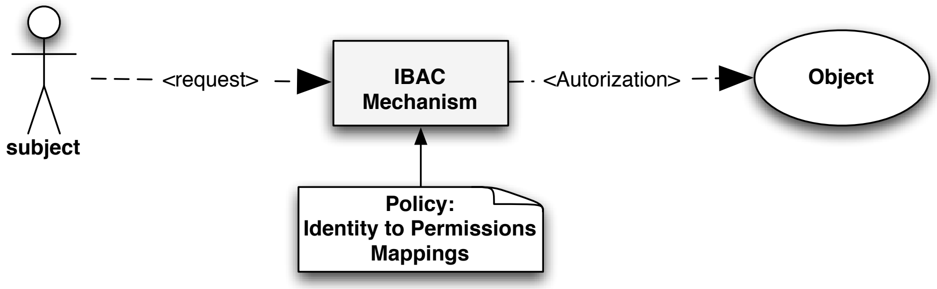


Figure 3.2 – An abstract IBAC Model

Here, the information used in the policy is the *identity* of the principal requesting access to the resource. Therefore, these models are called *identity-based*. A concrete example is the implementation using access control lists (ACL). ACL are the oldest and most basic form of access control commonly found in operating systems such as UNIX. ACL are lists of principals and the actions that each principal is permitted to perform on the resource. Here the *access control policies* represent rules that determine if association between the principal and a permission exists.

In the most general form, a permission is a triple (s, o, a), stating that a user s is permitted to perform an action a on an object o. Let S be the set of all users of the system, O the set of all objects and A the set of all possible actions. The *access control policies* represent a function $f : S \times O \to A$. Consequently, $f(s, o)$ determines the list of actions that the subject s is permitted to perform over the object o. Table 3.1 illustrates (as a matrix $A = |S| \times |O|$) the access control list of a system where $S = \{s_1, s_2, s_3\}, O = \{o_1, o_2, o_3\}$ and $A = \{a_1, a_2, a_3\}$ [El Houri, 2010].

IBAC models are very easy to implement and use. However, such approaches are unable to scale when the number of users increases [Yuan and Tong, 2005]. Moreover, the access control decisions are not related to any characteristic of the resource, the subject of the business application, making such approaches very vulnerable to attacks (ACL lists are easy to corrupt)

subject	o_1	o_2	o_3
s_1	$a_1, a_2, -$	a_2	a_2
s_2	$a_2,\ a_2$	$-$	$-$
s_3	$a_1,\ a_1, a_2$	a_1, a_2	a_1, a_2

Table 3.1 – Example of an access control list

and identity forgery (identity usurpation) [Chen, 2011].

3.1.3 Lattice-Based Access Control

Unlike IBAC models, lattice-based access control (LBAC) models (also known as mandatory access control models) are deployed when the access to an object depends on its characteristics and those of the subject, and not the wills of the object owner (i.e. ACL) [Sandhu, 1993]. Subjects' and objects' characteristics are represented by security labels (or levels) that are assigned to users and resources of the system. Objects' labels reflect the extent to which a resource is sensitive while a subject's label reflects the category of objects he is permitted to access. The systems in which LBAC models are implemented are often called *multi-level security systems* as the labels used in these systems represent a partial order (e.g. Top Secret, Secret, Confidential, Unclassified) which is assumed to form a *lattice*.

In LBAC, the process of access control is reduced to a control of data flow. For example, a *read* access to a resource is represented as a flow of data form the object to the subject, while a *write* access represent a flow of data from the subject to the object. In the light of this, the LBAC model's objective is to guarantee that data coming from a higher level never flows to lower level subjects, and that data coming from lowers level subject never flow up to objects of higher level. In sum, the label of a subject must be at least as high as the label of the object he wants to read, and to write on an object, the label must be at least as high as the subject's one [Chen, 2011]. These two security principles are respectively called "no-read-up" and "no-write-down" as illustrated in Figure 3.3 hereafter.

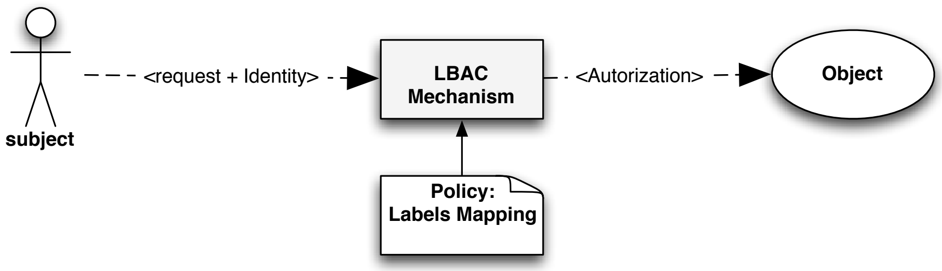


Figure 3.3 – Abstract lattice-based access control model

The Bell-LaPadula is the most famous model implementing LBAC. The Bell-LaPadula model has been used in both military applications and commercial ones. Bell-LaPadula was also used to implement the security mechanisms in the Multics operating systems.

The main limit of LBAC models is their lack of flexibility and scalability. Indeed, LBAC models are quite efficient and remain relatively manageable in systems with a small number of labels.

3.1.4 Role-Based Access Control

Both IBAC and LBAC have considerable deficiencies: LBAC models are clearly too rigid while IBAC are very hard to maintain and administrate [Becker, 2005]. This ascertainment has led several researchers in the early 1990s to investigate for alternative models such as *role-based access control* (RBAC) [Sandhu et al., 1996]. The development of RBAC was motivated by the fact that in most of the cases, sensitive resources were generally not owned by users but by the institution wherein users act in the capacity of a role of a job function [Becker, 2005, Yao, 2004]. Therefore, the key components in RBAC are *subjects*, *roles* and *permissions* as illustrated in Figure 3.4.

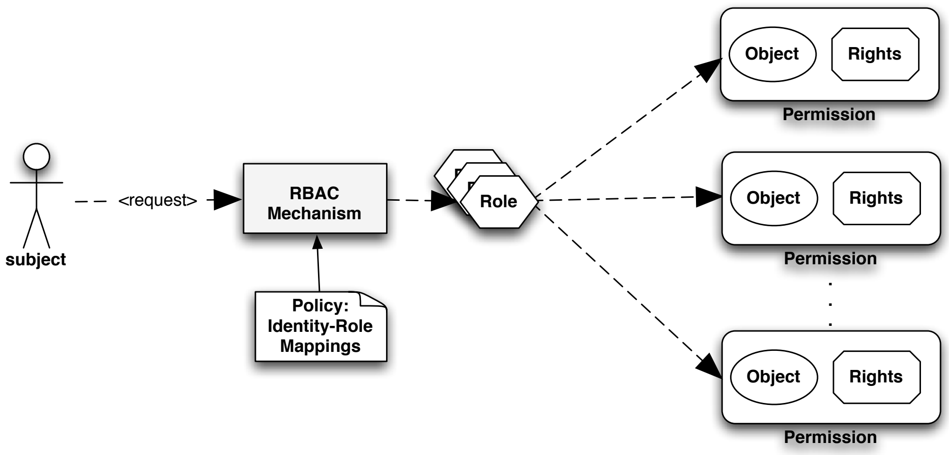


Figure 3.4 – Basic role-based access control model (adapted from [Genovese, 2012])

The policy represents here an assignment relation that associates users to the roles they hold, and roles to the permissions they are granted. Thus, roles represent an intermediary layer between subjects and permissions which makes RBAC a scalable access control mechanism, and reduces considerably the complexity of access control policies specification and administration when the subjects turnover is very high. When a subject joins or leaves the system, only the link between the user identifier and the role has to be updated. Subjects are assigned roles based on their duties, qualifications or competencies in the institution, while permissions are associated to roles based on the institution activities and goals.

RBAC received in the last twenty years considerable attention that conducted to the proposition of a whole family of models (e.g. [Ferraiolo and Kuhn, 2009, Sandhu et al., 1996, Nyanchama and Osborn, 1999, Sandhu et al., 2000, Dimmock et al., 2004, Ferraiolo et al., 2001, Li et al., 2002, Boella and van der Torre, 2005, Wang and Varadharajan, 2007, Finin et al., 2008]). However, most of the researchers would agree on the fact that $RBAC_0$ is the core model [Chen, 2011, Becker, 2005]. $RBAC_0$ is the main and the simplest model. $RBAC_1$ extends $RBAC_0$ with the capability to specify hierarchies of roles, introducing permissions' inheritance between roles. $RBAC_2$ extends it with constraints to enforce separation of duties, while $RBAC_3$ is a combination of $RBAC_1$ and $RBAC_2$.

3.1.5 Organisation-Based Access Control

Organisation-based access control (OrBAC) and RBAC have many similar aspects; e.g. in both approaches the concept of role is central. Therefore, most of the scholars describe OrBAC as an extension of RBAC [Kalam et al., 2003]. OrBAC aims at detailing the permissions that are used in abstract terms within RBAC. Indeed, RBAC policies define the mapping between identities and roles. Then based on these policies the access control mechanism grants permissions to subjects with respect to the role it belongs to. So the interpretation of permissions remains the responsibility of the administrator and may be very complex to perform; e.g. grouping similar rights, preventing inadequate permission and managing conflicting roles are some of the main issues to be addressed [Kalam and Deswarte, 2006].

3.1.6 Attribute-Based Access Control

The main idea of *attribute-based access control* (ABAC) models is to use policies that rely on the characteristics of authorised individuals instead of their identities, roles or clearances for issuing authorisations [Yuan and Tong, 2005, Lee, 2008]. These policies are then satisfied through the disclosure of credentials issued by third party attribute certifiers (e.g. organisations, companies, institutions, etc.). Consequently, subjects can gain access to resources without being priorly known by the system administrator (or the resource owner) as illustrated in Figure 3.5.

Unlike IBAC, LBAC and RBAC, ABAC policies can define permissions based on any relevant characteristic. Characteristics fall into three categories [Yuan and Tong, 2005]:

Subject attributes. Subjects are the entities requesting access to objects. Each subject can be characterised via a set of attributes without explicitly referring to its identity. In the ABAC literature, almost all information that can be associated to a subject is considered as an attribute. Such attributes may include subject's name, role, affiliation, address, age, and so on. Of course, the subject identity can also be considered as an attribute.

Object attributes. Objects are resources that the subject wants to manipulate. Like subjects,

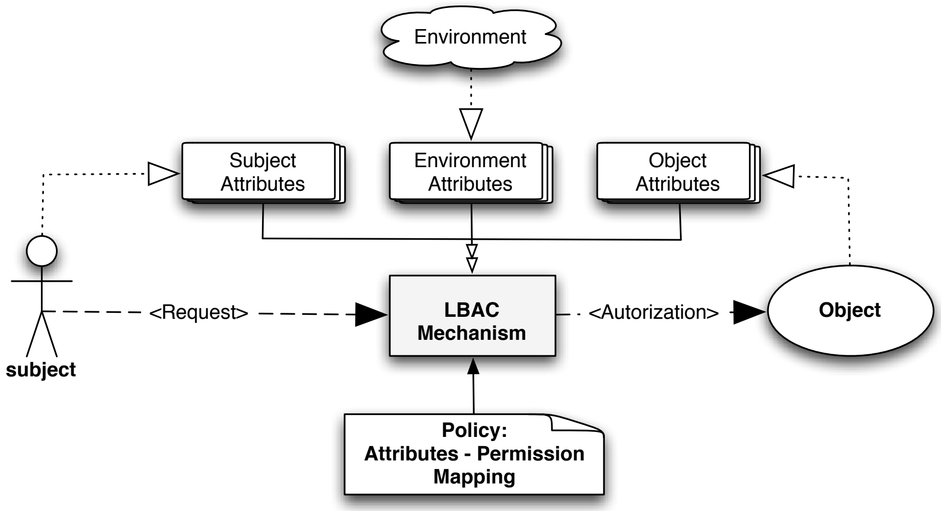


Figure 3.5 – Abstract attribute-based access control model

resources have properties that are also called attributes in the ABAC model. Resource attributes are also important in access control decision as they can affect the type of the permission accorded (e.g. a read on a text file does not have the same consequence as an execute on a program). Resource attributes may include the name of the resource, its type (text file, image, serve, etc.), the owner, and so on. These information is generally made public and can be extracted automatically from metadata.

Context attributes. The environment or more generally the context in which the interaction is undertaken has been ignored for a long time by the security community. In previous approaches, permissions are attached to individuals (IBAC), roles (RBAC) or labels (LBAC) and derive the same authorisation as long as the artefact to which they are attached remains valid (or when they are revoked by the system administrator). Thus, the context of the interaction never affects the access control decision, whereas environment attributes such as time, date, threats are relevant in applying the access control policy.

Discussion

We presented in this section an insight on predominant access control models. Though effective in many specific situations, the above classical approaches failed addressing the inherent properties (e.g. openness, distribution, decentralisation) of modern systems such as the Internet. The main limitation of access control model lies in the fact that they heavily rely on identity for policies evaluation. For instance, in IBAC, identity is the solely information based on which access control decisions are granted. LBAC, RBAC and OrBAC do not perform better; they only extend the IBAC approach to ease its application (e.g. permissions factorisation in most of the case) using specific concepts such as *labels*, *roles*, and *views*. This issue limits these models' scalability and rises several other problems such as privacy and confidentiality (e.g. people do not to reveal their password as they tend to use the same for many several applications).

In addition, in IBAC, LBAC, RBAC and OrBAC, the subject must be known to the system administrator (so that this latter adds a permission) before he would be able to request access to the resource. So these systems operate under closed and finite settings in which both subjects and objects must be known beforehand. This last issue, in addition to their limited scalability, make these models clearly not relevant for application in the context of Internet.

In contrast, ABAC does not rely explicitly on subjects' identities. Consequently, ABAC can manage requests originating from unknown subjects, as long as they are able to prove that they have the required attributes (i.e. specified in the ABAC policy). However, the approach advocated in ABAC can neither be distributed nor decentralised; attributes are certified by a central authority and the users themselves can not certify each other attributes. This representd a real limit to the application of ABAC in large, open and distributed systems.

Finally, and common to all models we reviewed in previous sections, access control decisions are the solely form of decision handled. However, distributed systems require more sophisticated decisions that combines access control and delegation. It is this last form of decision that motivated the advent of research on *trust management*.

3.2 Trust Management

We introduce in this section the *trust management* approach which encompasses *distributed trust management models (DTM)* and *automated trust negotiation models* that have been introduced in Section 2.4 of the previous chapter. Both models originate from the work of Blaze and colleagues [Blaze et al., 1996, Blaze et al., 1999a] which tried to address the limits of the above traditional access control models with respect to distribution and decentralisation limitations.

In the remainder of this section, we will first define the concept of *trust management*, then we will present in Section 3.2.3 primary concepts this approach relies upon.

3.2.1 Definition

Trust management has been defined by Blaze and colleagues as "a unified approach to specifying and interpreting security policies, credentials, relationships which allow direct authorisation of security-critical actions." [Blaze et al., 1996, Blaze et al., 1999a]. The main novelty of the approach introduced by Blaze *et al.* is that they unified the concepts of *security policy*, *credentials* and *authorisation* under the concept of *trust management*. However, their definition is too abstract and not very intuitive to explain what trust management really is.

For Jøsang, trust management is "the activity of collecting, codifying, analysing and presenting security relevant evidence with the purpose of making assessments and decisions regarding e-commerce transactions"[Jøsang et al., 2007, Jøsang, 2007]. Although, broader and more intuitive, this definition was criticised by Grandison in his doctoral thesis to be too domain-specific (i.e. e-commerce) [Grandison, 2003]. Nonetheless, Grandison reused it to define trust management as"the activity of collecting, encoding, analysing and presenting evidence relating to competence, honesty, security or dependability with the purpose of making assessments and decisions regarding trust relationships for Internet applications" [Grandison and Sloman, 2003, Grandison, 2003].

The main drawback in Grandison's definition is that the author restricted the nature of the evidences based on which the trust relationship can be established. Further, Grandison used the verb "collecting" for evidence while some of them can not collected but should be requested (e.g. credentials). Therefore, we prefer to adapt the above definitions to provide one that best matches our understanding of trust management.

Definition 3 (Trust Management) *The automated activity of collecting, requesting, providing and analysing information with the purpose of making trust decisions (e.g. access control, delegation, collaboration) based on policies.*

The main aspect we stress in this definition is the automated nature of trust management process. It is the automation requirement that makes the trust management issue complex and necessitates so much investigations (cf. Section 3.3). Further, we used the term information instead of evidence in order to comply with the analysis we have made in Section 2.5. Finally, we generalise the purpose of trust management to trust decisions rather than focusing on trust relationships. From our perspective, a relationship implies some kind of continuation in time, while a trust decision better reflects the dynamic nature of trust.

3.2.2 Trust Management System

Trust management systems (TMS) were originally designed to solve the problem of deciding whether a request to perform a potentially harmful action on a sensitive resource comply with the access control policy [Blaze et al., 1996, Blaze et al., 1999a]. Nevertheless, these systems are currently used in a broader way to evaluate whether a trust decision complies with the policy or not.

Definition 4 (Trust Management System) *An abstract system that processes a symbolic representation of trust relationship in the perspective of trust decision automation.*

In the above definition, the "symbolic representation of trust" refers to the concepts of credentials and policies by means of which the issuer states that he trusts the entity to which the statement is applicable. Of course, this trust is not generic, thus is most of the cases the statement concerns a specific issue (e.g. read a document). The symbolic representation of trust relationships can be best illustrated through the everyday ticket experience [Wikipedia, 2013].

The ticket (let us say a tram ticket) can be considered as a symbol of trust between the tram company and the ticker holder. The ticket acts as a proof that the holder paid for the journey and consequently that he is entitled to get on the tram. Once bought, the ticket can be later given to someone else, thus transferring the trust relationship. In the tram, only the ticket will be verified and not the identity of the holder. Concretely, in the above example, the tram ticket illustrates the importance for credentials while the tram inspector enforces the policy (which is quite simple here).

Thus, a trust management system aims at linking the requester and the requested via a trust relationship based on which a trust decision can be made. To that aim, trust management systems provide a language for the specification of **policies** and **credentials**, and a **trust management engine** (trust engine for short) that evaluates whether the provided credentials satisfy the specified policy.

3.2.3 Foundations

As illustrated in the previous section, trust management systems are made possible thanks to the introduction of *credentials*, *policies* and *trust engine* [Galinović, 2010, Nejdl et al., 2004, Lee et al., 2009, Ryutov et al., 2005, Winsborough and Li, 2006]. These three components are presented in the following sections.

3.2.3.1 Credentials

Credentials (or digital credentials) represent the counterpart of the paper credential we use in the real world (e.g. passport, driving licence, student card). They represent digital documents or messages that are certified (i.e. signed) by *certification authorities*. They allow user authentication but can also provide additional information such as the user's attributes (cf. Section 3.1.6), memberships or rights. Blaze, in his trust management jargon, defined credential as “a signed message that allows a principal to delegate part of its own authority to perform actions to other principals”. It is this definition that we used as a basis in our thesis. For instance, a public key "certificate" is an example of a credential. Public key infrastructures (PKI) have been systemically used by trust management systems to create, distribute, validate, store and revoke credentials. In what follows, we review two prominent approaches to PKI that are *certification authorities (CA)* and *cross-certification (CC)* [Linn, 2000]. Certification authorities approach relies on the trust that exists between an individual and the organisation/institution representing the certification authority, while cross-certification drew trust from the experience of others. We illustrate both approaches with two concrete implementations of these approaches that are, respectively, *X.509* and *PGP*.

Certification Authorities: X.509

X.509 is a widely used standard for credentials management. Typically, an X.509 certificate contains (but is not limited to) the following information: the *issuer name*, the *subject name*, *signature algorithm identifier* (e.g. RSA, DSA, etc.), the *validity period*, and optional information which can be an attribute-value pair [Samarati and Vimercati, 2001].

In X.509, the certification management process involves several entities: *Certification authorities* (CA) that are the entities which issue and revoke certificates, *Registration authorities* (RA) that vouch for the biding between public keys and, *repositories* that store and make available certificates and certificate revocation lists (CRLs), *certificate holders*, and *clients*.

The certification process starts when the certificate holder requests a certificate from the registration authority. The RA verifies the future holder identity and/or the attributes for which he wants to be certified (e.g. in the case of driving capabilities, RA checks whether the entity holds a driving licence in the real life). Once this step has performed, the RA forwards the request to the CA which signs the certificate after verifying that the RA really approved the request. Then the CA sends a copy of the certificate to the repositories so that clients who want to communicate with the requester can get the public key. Analogously, when a certificate needs to be revoked (e.g. a key has been compromised), the CA updates the repositories and adds an entry to the revocation list. Certification authorities are organised in a hierarchical way; the root CA certifies other CAs which in turn certify other CA or simple requester. This issues represent the most controversial aspect of X.509 as root CAs are self-certified entities [Yu, 2003, Conrad et al., 2012].

Cross-Certification: PGP

PGP (Pretty Good Privacy) is a public key infrastructure that has been initially designed to guarantee authenticity, integrity and non-repudiation of exchanged data. Although it can encrypt any data type, PGP has been most commonly used for emails exchange. Unlike X.509, in PGP each user can generate a pair (public key, private key) which is associated to his unique identity. However, the keys are used independently from the identity of the user, thus PGP guarantees authenticity, integrity and non-repudiation while preserving the confidentiality of the individual identity (i.e. privacy). A PGP certificate includes (but is not limited to) the following information: the *certificate holder*, *holder's information* (e.g. his name, his ID, his email, photo, etc.), *digital signature of the holder*, *validity period* and the *encryption algorithm*. Thanks to the holder's information, a unique PGP certificate can contain several information which can be certified via the same key [Yu, 2003].

In PGP, anyone can freely issue and certify its own certificates, thus everybody can act as a certification authority. Thus, everyone can certify others' public keys (PGP call this mechanism introduction), making the trust issues central to PGP. Therefore, PGP relies on a "web of trust" which is the network created by individuals introducing each other's keys. To that aim, PGP uses two distinct metrics: one quantitative (the key validity) and another qualitative (the trust level of a key).

In PGP, a user always trusts his own key making the level of this key *ultimate*. The other levels used in *PGP* are *complete*, *marginal*, *unknown* and *untrusted* (in decreasing order of trust level). Similarly, the validity of a key can be either *valid*, *marginally valid* or *invalid*. The amount of individuals signing the public key determines its validity, while the trust level of a key is determined via recommendation. The more a key is trusted the less validity it requires to be accepted [Gerck, 2000, Prohic, 2005].

3.2.3.2 Policies

Policies have been extensively used in the computer science literature (e.g. information systems, security, multi-agent systems). Initially, policies have been introduced in computer science to automate tasks and decision makings (e.g. batch instructions). But nowadays, the main motivation for using policies is to make systems support dynamic and adaptive behaviour. Policies allows a system to change its behaviour without being stopped.

Despite their extensive use in the literature, the concept of policies is still hard to define and the provided definitions are either too generic or domain specific. For instance, Sloman defined policies as "rules governing the choices in behaviour of a system" [Sloman, 1994]. While this definition captures the meaning of a general policy, it failed addressing its role which is to specify the circumstances under which the choices are made (in reaction to which conditions). Further, this definition reduces the form of a policy to a set of rules and thus excludes many of the approaches which do not rely on rules (cf. Section 3.3). Recently, De Coi and Olmedilla stated that "policies specify who is allowed to perform which action on which object depending on properties of the requester and of the object as well as parameters of the action and environmental factors" [De Coi and Olmedilla, 2008]. This definition makes explicit the ABAC approach, and thus covers *de facto* IBAC, LBAC, RBAC and OrBAC policies. However, this definition restricts the scope of a policy to situations in which an access request has to be evaluated. Consequently, this definition could not be used to describe situations in which a decision is not merely an access control decision (e.g. delegation). To avoid misunderstandings, we clarify the meaning of trust policies and define it as follows.

Definition 5 (Policy) *A policy is a statement that specifies under which conditions an entity (human or artificial) can be trusted for a specific issue (e.g. resource action, task delegation)*

With respect to Definition 1, a policy represents the expression of the conditions under which the individual A deliberately takes the decision to trust B. Thus from our perspective, the role of a policy is twofold: (i) it serves a means for A to express the policy that its trust management system will rely on, and (ii) it is used as a common language that A and B will use to exchange their respective trust conditions. In the light of that, the role of the policy specification language is paramount. The language provides the basic syntax to express conditions which represent the building blocks of a policy. Specification languages can be more or less verbal and can have solid or weak formal basis. Thus, depending on the nature of the policy specification language, policies fall into three categories: *informal*, *semi-formal*, and *formal*. In this thesis, we limit our attention to *formal* policies that can be understood by both artificial and human agents. In Section 3.3 we present the most important policies languages that have been proposed in the last fifteen years.

3.2.3.3 Trust Engine

The objective of the trust engine is to assess if the credentials provided by the requester are valid and whether they satisfy the specified policy. Importantly, trust management systems are not responsible for making trust decisions. It is always the human or the application using the TMS that decides whether to effectively trust the requester or not[2]. So the main advantage in using a TMS is to offload applications of complex and tedious tasks that are *credentials verification* and *policies evaluation*. Figure 3.6 illustrates this principle and shows the basic functioning of a trust management system.

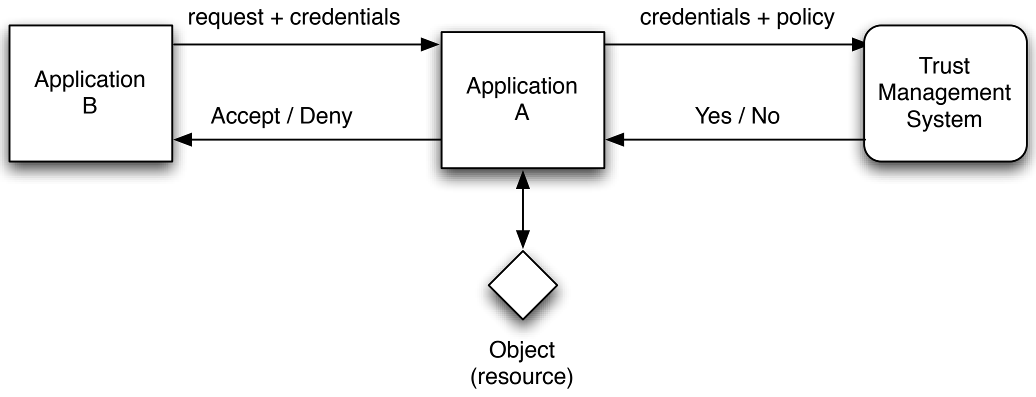


Figure 3.6 – Illustration of the functioning of a trust management system

In Figure 3.6, the application A invokes the trust management system to determine whether application B can be allowed to perform an operation on a resource. To that aim, the application A provides the TMS with its local policy for the resource concerned by the request and the credentials provided by application B. The TMS produces an answer based on which the application A decides to allow or deny the request of B.

Depending on their degree of sophistication, trust management systems can provide more or less functionalities. In our thesis, we are particularly interested in the TMS that output detailed answers. Figure 3.7 illustrates the degree of sophistication a TMS can achieve by providing more or less elements in its answers.

Based on the type of information provided by the TMS, Seamons and colleagues identified two functioning modes of trust management systems [Seamons et al., 2002]. In our work, we distinguish four modes that we summarise as follows:

- **Mode 1**: In this mode, the TMS produces a boolean answer (trust/no trust) that states whether the credentials provided satisfy the policy.
- **Mode 2**: In addition to the boolean answer, in this mode the TMS provides a *justification*, when the request is denied, that states which conditions in the policy the provided

[2]The study of how trust decisions are made is out of the scope of this thesis

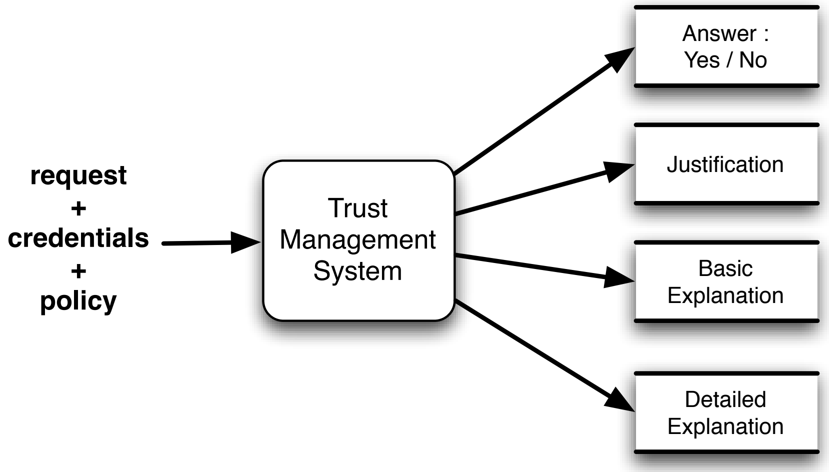


Figure 3.7 – Functioning modes of a trust management system

credentials were unable to satisfy.

- **Mode 3**: In this mode, the TMS provides an answer, a justification and an *explanation* when the policy is satisfied. The explanation contains all credentials that satisfy the policy.
- **Mode 4**: This last mode extends the third mode as it provides a *detailed explanation*. The detailed explanation is obtained by providing all subsets of credentials that satisfy the policy.

Modes 1 and 2 are often used by the resource owner to verify whether the credentials its interlocutor provided satisfy its policy, while modes 3 and 4 are used by the requester to determine whether the credentials it possesses (and which subset of credentials) satisfy the policy stated by the owner of the resource. These latter modes were particularly introduced in the context of trust negotiation that we will tackle in the next section.

3.2.4 Automated Trust Negotiation

Automated Trust Negotiation (ATN) (cf. Section 2.4.2.2) is an approach to trust management in which trust is established through the gradual, iterative, and mutual disclosure of credentials and access control policies [Ryutov et al., 2005, Yu, 2003]. Unlike the traditional trust management systems that have have been presented in the previous section, automated trust negotiation approach consider credentials as first class resources that should be protected through *release policies* dedicated to them. Consequently, ATN systems provide users with better fine-grained and flexible control over the disclosure of the potentially sensitive data conveyed by their credentials [Lee, 2008, Ryutov et al., 2005, Yagüe, 2006].

However, negotiation generally refers to the process by which agents can reach an agreement on matters of common interest [Parsons and Wooldridge, 2002]. In our thesis, we adapt this well-established definition to the *negotiation-based adaptation* of trust policies.

Definition 6 (Automated Trust Negotiation) *Automated trust negotiation is an iterative process in which two interacting agents reach an agreement on the credentials they are willing to release to gain each other's trust.*

The introduction of trust negotiation has several benefits. First, it better reflects the asymmetric nature of trust as suggested in the previous chapter (cf. Section 2.3.3). It allows also the establishment of bilateral trust as both participants in an interaction can request credentials from each other. Finally, it allows a more flexible trust management as trust is established gradually and incrementally. Research on trust negotiation has been principally focusing on *how to make trust management systems achieve trust negotiation?* and *how to make trust negotiation successful?*. The first question represents the requirements for trust negotiation, while the latter represents *trust negotiation strategies*. The requirements are further divided into *requirements for trust management systems* and *requirements for policy specification languages*.

Trust negotiation is inherently a strategy-driven process as the choice made by the individual affects the amount of credentials it releases and the time it makes in this task [Lee, 2008]. Therefore, recent research in trust negotiation area has been primarily focusing on proposing efficient and optimised negotiation strategies and protocols. Generally speaking, however, the implemented negotiation strategies fall into three categories: *eager*, *parsimonious* and *prudent* strategies [Grandison, 2003].

Eager Strategy: In the eager strategy, participants in the negotiation adopt a naive position in which they disclose almost all credentials they possess. The negotiation is considered to be successful when each participant received enough credentials to be assured about the interaction he is engaged in. The major advantage of this strategy is that it does not require the use of *release policies* and it minimises the time of the negotiation [Ardagna et al., 2007]. However, this strategy increases the amount of disclosed credentials and thus the sensitive date they convey.

Parsimonious strategy: With this strategy, participant exchange only credentials requests (no credential is released) and tries to find a possible sequence of credentials disclosure that can lead to a successful negotiation [Grandison, 2003]. Also, in parsimonious strategies, only credentials that are explicitly requested are released. Unlike the eager strategy, the parsimonious one minimises the credentials exposure but increases considerably the time of the negotiation without any guarantee of success.

Prudent strategy: is a mix of the previous strategies. An eager strategy is applied to credentials that are not sensitive and a parsimonious strategy is used for the sensitive ones. This strategy has been proved to over-perform the other strategies in situations where the negotiation involves the disclosure of sensitive and non sensitive credentials [Yu et al., 2000].

3.3 Trust Management Systems Analysis

Given the multiplicity of trust management systems[3] one may argue that a comparison among them is neither possible nor desirable. However, we think that from a higher perspective and with a good abstraction level, a comparison of all these systems is not only possible but even worth.

In this section, we review a selection of trust management systems. As discussed in Section 2.4.2 (cf. Chapter 2), these systems are split into *decentralised trust management systems (DTM)* and *automated trust negotiation systems*. DTM systems are further divided into *authorisation-based TMS (ABTMS)* and *Role-Based TMS*. These systems are presented in the chronological order in which they have been published. We think that respecting this chronological causality helps the reader to understand key elements that motivated the contribution of each system.

Each system is described with respect to the trust management key concepts that we presented above, namely *credentials*, *policies* and *trust engines*. For *credentials* we are interested in the nature of information used by each system to derive trust and the formalism used to express it. In *policies*, we will focus on the type, semantic, expressiveness and flexibility of the formalism used to represent policies. Finally, for *trust engines* we will stress the limits of these systems with respect to our *objectives* (cf. Section 1.2). We will particularly evaluate to which extent these systems are able to support the *social* (cf. Section 1.2.1) and *dynamic* (cf. Section 1.2.2) aspect of virtual communities.

3.3.1 Authorisation-Based TMSs

The *authorisation-based TMSs* category relates to systems that pioneered the trust management approach. Whilst most of these systems are considered nowadays as obsolete, the mechanisms they proposed remain valid and can be found at the basis of most of modern trust management systems. They inherit from IBAC and ABAC models. They build a trust chain in order to map a credential holder to the authorisation it can be trusted for.

PolicyMaker [Blaze et al., 1996]

PolicyMaker is the first application stamped as a trust management system. This system introduces the concept of programmable credentials and policies by means of an assertion

[3]judging from the list of surveys devoted to this subject; e.g. [Firdhous et al., 2012, Yao, 2004, Grandison, 2003, Krukow et al., 2008, Bandara et al., 2007, Yu, 2003, Braghin, 2011, Gray, 2006, Fernandez-Gago et al., 2007, Jøsang, 2007, Artz and Gil, 2010, Ruohomaa and Kutvonen, 2005, Jøsang et al., 2007, Saadi et al., 2011, Yagüe, 2006, Liu, 2011]

language. The syntax of an assertion is:

`Source ASSERTS Subject WHERE Filter` (3.1)

The above syntax can be used to specify both credentials and policies. Here, the statement represent a credential by means of which a *source* authorises a *subject* to perform actions that are accepted by the *filter* (i.e. interpreted program). The main difference between a credential and a policy is that in policies the keyword *policy* is always used as the source of the assertion. For instance, we can use the assertion given below to authorise the entity holding the public key “`rsa:123`” to access all resources shared among the community.

`policy ASSERTS “rsa:123” WHERE filter to access all shared resources` (3.2)

In PolicyMaker, an assertion (whether credential or policy) is used to state that the source (or the local application in the case of a *policy*) trusts the key holder to perform the actions accepted by the filter. However, the formalism used to encrypt keys is left to the application using PolicyMaker, thus PolicyMaker is generic with respect to the keys encryption scheme. The semantic of the operations and the enforcement of the trust evaluation decisions are also left to the application.

Typically, the application provides to the PolicyMaker trust engine a set of requested actions, a set of credentials and a policy. The objective of PolicyMaker is to check whether the credentials form a delegation chain by means of which the action can be linked to the key of the keys. PolicyMaker then replies with an answer (“`Trust`” or “`False`”) and it is up to the application to interpret the answer and take the appropriate decision. The functioning of the PolicyMaker engine is similar to the architecture we described in Figure 3.6 (cf. Section 3.2.2). PolicyMaker does not support *negotiation*. Policies are evaluated in a static and context-independent way. Finally, PolicyMaker can not be used to manage resources that are owner by more that one individual.

In PolicyMaker, the choice of the policy specification language is left open which makes policy evaluation undecidable in most of the cases [Grandison, 2003]. KeyNote [Blaze et al., 1999b], its successor, overcomes this drawback and imposes that the policies must be written in a specific language. KeyNote makes also the cryptographic verification which was left to the application in PolicyMaker.

REFEREE [Chu et al., 1997]

REFEREE (Rule-controlled Environment For Evaluation of Rules and Everything Else) is a W3C and AT&T joint trust management system used for web document access control. The system was developed based on the PolicyMaker architecture, but the functioning of the system is somehow different. Here, it is the resource providers (i.e. authors of web content) that are trying to gain the trust of the resource consumer (i.e. the site visitor). The system was essentially used to prevent minors from accessing illegal content. The system uses PICS

(Platform for Internet Content Selection) labels as credentials that the REFEREE trust engine (a browser plug-in) evaluates with respect to a local policy.

Profiles-0.92 is a rule-based policy language that was designed for REFEREE. As illustrated in Listing 3.1, each policy is an "s-expression" that is evaluated in a top-down manner.

```
(((invoke "load-label" STATEMENT-LIST URL "http://www.emse.fr/")
(false-if-unknown
(match
(("load-label" *)
(service "http://www.emse.fr/CA.html") *
(ratings (RESTRICT > trust 2)))))
STATEMENT-LIST))
```

Listing 3.1 – Example of a policy specified in Profiles-0.92 (adapted from [Chu et al., 1997]).

The above policy states that any document having a label (certified by *emse*) with a trust rating greater than 2 can be viewed by the user. The matching between labels and the conditions specified in the policy is purely syntactic. Thus it remains to the application and the labelling institution to define its semantic.

REFEREE trust engine evaluates the above policy in two steps. First, tries to find and download labels provided by the server which URL has been specified in the policy. Then a pattern-matcher is run to find a label with a trust rating. If the rating is greater than 2 the result of the evaluation would be `true`, if not the result would be `false` and if no label was found the result would be `unknown`. Thus, REFEREE trust engine implements a three-valued logic, specially for the management of the meaning of `unknown` [Chu et al., 1997, Grandison, 2003].

Binder [DeTreville, 2002]

Binder is the first trust management system which uses a logic-based policy language [DeTreville, 2002]. The particularity of Binder lies in its explicit specification of right delegation though the extension of Datalog with the `says` construct [Ceri et al., 1989]. In Binder, credentials represent keys which holder use to sign delegation assertions. Then policies are used to filter these assertions and map them to their authors. The specification language proposed in Binder allows the expression of two type of declarations: *beliefs* and *policies*. For instance, the following declaration is used by a individual A to state that another individual B can be trusted for joining his community and for reading his personal files.

```
can(B, read, MyFile).
can(B, join, MyCommunity).

can(X, join, MyCommunity) :- Y says trust(Y,X), can(Y,join,MyCommunity)
```

Listing 3.2 – Examples of Binder declarations (beliefs and policies).

The above example illustrated also the declaration of policies in Binder. In this example, the policy states that if A trusts an individual Y to join his community and that Y trusts another individual X, this latter can also be trusted to join the community. Worth noting in this policy is the use of the `says` construct.

The role of the Binder trust engine is to evaluate policies with respect to local assertions (i.e. beliefs) and assertions made by others (i.e. others' beliefs). The main novelty of the system lies in the addition of the `says` construct each time an assertion is received from others. Indeed, each time an individual sends an assertion, the Binder trust engine transforms this assertion into a certificate which is signed with the private key of the issuer. Then these assertions are sent to other Binder engines in order to make trust decisions. When an assertion is received, Binder verifies the validity of the certificate and automatically quotes the assertion with the `says` construct to distinguish them from local assertions.

SULTAN [Grandison, 2003]

SULTAN (Simple Universal Logic-based Trust Analysis) is a TMS that has been proposed for the specification and analysis of trust and recommendation relationships [Grandison, 2003]. In SULTAN, credentials represent certified statements about identity, qualification, risk assessment, experience or recommendations. The system is provided with a policy specification language in which policies are used to specify two types of policies: *trust/distrust* policies and *positive/negative* recommendation policies. In fact, trust policies corresponds to classical meaning of policies used in this thesis, while recommendation policies are statements by means of which individuals make recommendations to each others. In the following, we provide the syntax used to specify both types of policies.

$$PolicyName : \mathbf{trust}(Tr, Te, As, L) \leftarrow Cs; \tag{3.3}$$

The above policy represent a trust policy by means of which a trustor (i.e. Tr) trusts (or does not trust) to some extent (L corresponds to the level of trust) a trustee (i.e. Te) with respect to an action set (i.e. As) and if the conditions hold (i.e. Cs). Similarly, the recommendation policy defined hereafter specifies that the recommender (i.e. Rr) recommends at a recommendation level (i.e. L) the recommended agent (i.e. Re) to perform the action (i.e. As) if the conditions (i.e. Cs) hold.

$$PolicyName : \ \mathbf{recommend}(Rr, Re, As, L) \leftarrow Cs; \tag{3.4}$$

The SULTAN trust engine is responsible of collecting the information required for the policy evaluation, making trust relationship decisions, and monitoring the environment in the perspective of re-evaluating existing trust relationships.

Ponder [Damianou et al., 2001]

Ponder is merely a policy language for which there was no associated trust management system [Damianou et al., 2001]. Ponder is the first object-oriented policy language that adopts a role-based approach. Nevertheless, many of the features proposed by this language inspired other systems which explains our motivation to review it. Ponder is a declarative language which can be used for the specification of four types of policies, namely *authorisation*, *obligation*, *refrain* and *delegation*. Ponder pioneered the use of a deontic approach which was reused later by other languages such as Rei [Kagal et al., 2003] and KAos [Uszok et al., 2004, Uszok et al., 2003]. Furthermore, the main novel aspect of Ponder lies in the constructs it provides for updating, organising and handling policies on runtime according to the environment context.

For instance, the following example is an instantiation of a positive authorisation policy type called `rights`. The policy specifies that `members` (the subjects of the policy) can *modify* the target objects of type `file` that are stored in the common space of the community `com`.

```
type auth+ rights(member, target <file> com) {action modify(); }        (3.5)
```

Another interesting aspect of Ponder policies lies in the fact that subjects and objects to which a policy applies can be grouped. For instance, in the above example, the policy concerns all members of the community and all files, making factorisation of rights implicit and more efficient that what can be expressed in RBAC models or any other role-based trust management systems. Further, the authors assumed that their language is flexible, scalable and extensible; flexible as it allows the reuse of policies since many instance of the same policy can be created for many different conditions; scalable as it allows the definition of composite policies; and extensible as it accepts the definition of new types of policies that can be considered as subclasses of existing policies, thanks to the object-oriented approach. However, due to the absence of implementation, none of these properties have been proved to be valid.

Recently, Ponder has been redesigned, as *Ponder2*, to increase the functionality of authorisation policies (e.g. operators for all managed objects have been added). In contrast to the previous version, which was designed for general network and systems management, Ponder2 has been designed as an entirely extensible framework that can be used at different scales: from small embedded devices to complex services and virtual organisations.

3.3.1.1 Role-Based TMSs

In *authorisation-based TMSs*, the delegation of authority is used in a very restrictive way. For instance, in PolicyMaker, a community member cannot simply specify that “all members of my community can access my resources”. In these systems, the solution would be to delegate the right to the members I know and authorise these members to further delegate the right to each member they know. This approach makes trust decision complex and difficult to manage and control (e.g. a member can delegate the right to non members).

In response, a new generation of *role-based* trust management systems that take advantage of the strength of RBAC and trust management approaches have been used. From RBAC, role-based TMS borrow the concept of role and from trust management, they borrow delegation and distributed credentials certification. The combined use of roles and distributed credentials management makes these systems convenient for large scale systems such as virtual communities.

IBM Trust Establishment [Herzberg et al., 2000]

IBM Trust Establishment (IBM-TE) is a role-based trust management system developed by IBM for e-commerce applications. The main objective of IBM-TE is to map credential holders to groups. Credentials are specified in a generic language but the system provides transcoding mechanisms to handle X.509. Credentials are used to authenticate users, while policies are used to express restrictions on how a credential holder could belong to a group. Groups are used in the sense of roles which are mapped to authorisations (cf. Section 3.1.4). IBM-TE comes with a dedicated XML-based policy specification language called *Trust Policy Language* (TPL). An example that illustrates the TPL syntax is given in Listing 3.3.

```
<GROUP NAME="Community">
<RULE>
<INCLUSION ID="reco"
       TYPE="Recommendation"
       FROM="members"
       REPEAT="2">
</INCLUSION>
<FUNCTION>
       <GT>
       <FIELD ID="reco" NAME="Level"></FIELD>
       <CONST>1</CONST>
       </GT>
       </FUNCTION>
</RULE>
</GROUP>
```

Listing 3.3 – A fragment of a TPL Policy specifying a membership rule.

The above policy is used to add a new member to the group `Community` (role). The policy states that a new member can be admitted if he can provide two recommendations of existing members. For that, two XML tags are used: *inclusion* and *function*. `Inclusion` tag defines the credentials that the requester must provide (e.g. two recommendation credentials), while the `function` tag allows the definition of additional conditions over requested credentials (e.g. recommendations must be greater than, `GT`, 1). The trust engine processes credentials in a traditional way. Along with his request, the requester provides the set of credentials he possesses. These credentials are then evaluated by the engine to determine to which group the requester can be mapped [Herzberg et al., 2000].

Role-Based Trust Management Framework [Li et al., 2002]

The *Role-Based Trust Management Framework* (*RT*) was initially designed to support trust decision making in collaborative environments [Li et al., 2002].

In *RT*, roles are represented as attributes, so an individual is said to belong to a role if it possesses a credential in which the role identifier is specified. Unlike IBM-TE, *RT* uses an extension of Datalog [Ceri et al., 1989] to represent both credentials and policies. In *RT*, the terms credential and policy are used interchangeably. However, *RT* uses the term *certificate* in reference to the meaning of credential we use in our thesis. The formalism used to specify certificates have not been defined, but some approaches proved the compatibility of *RT* with both X.509, PGP and any certification mechanisms using PKI. The syntax used to specify policies is defined as follows:

$$Com.evaluator(?X) \leftarrow Com.recommender(?X) \wedge Com.Member \tag{3.6}$$

The above policy states that any member of the community that recommended another member is allowed to evaluate it. The *RT* policy specification language comes in five flavours RT_0, RT_1, RT_2, RT^T and RT^D [Li et al., 2002]. RT_0 is the basic language that supports roles hierarchies, delegation of authorities over roles, roles intersection and attribute-based delegation of authority. RT_1 adds to RT_0 the possibility to add parameters to roles (e.g. in the previous policy, the evaluator and the recommender roles are endowed with parameters), RT_2 adds to RT_1 logical objects. They represent a way to group (logically) resources of access modes in order to ease the specification of policies. RT^T adds thresholds to roles, while RT^D supports delegation of roles activation (i.e. roles that are active only within a specific context).

Cassandra [Becker and Sewell, 2004]

Cassandra is a TMS that aims at enabling individuals involved in potentially large scale systems (e.g. P2P systems) to share their respective resources under the restriction of local policies [Becker and Sewell, 2004]. The system has been principally deployed in the context of electronic health records access control. Cassandra is role-based and supports X.509 credentials. Policies in Cassandra are expressed in a language based on Datalog with constraints [Ceri et al., 1989]. For instance, a policy can state that an individual A can delegate the role of *moderator* as long as he commits to the role *admin*. This policy is specified using the constructor `canActivate()` as follows:

$$\texttt{canActivate}(B, moderator) \leftarrow \texttt{hasActivated}(A, admin) \tag{3.7}$$

Cassandra provides flexible delegation mechanisms which allow the explicit specification of delegation lengths [Becker and Sewell, 2004, De Coi and Olmedilla, 2008]. In addition, Cassandra proposes a mechanism for roles revocation, including cascade roles revocation (e.g. if an individual is revoked from its role, all individuals to which he delegated the role are revoked too). Finally, the Cassandra policy language allows the specification of roles hierarchies.

Cassandra trust engine is only available as a proof of concept implementation in OCaml. The main feature of the engine is the implementation of the semantic of Cassandra policies for their evaluation.

3.3.2 Automated Trust Negotiation Systems

In trust management, a trust decision comes after a complex interaction process, where parties exchange policies and credentials. Traditionally, in early TMS, trust is established in an unidirectional way: the resource owner is assumed to be trusted by the requester. Consequently, before manipulating the resource, the requester must provide its credentials to know whether its request is accepted or not. So if the request was not accepted, the requester would have uselessly released its credentials (which contains sensitive data such as its id, its age or its address). Therefore, due to privacy considerations, such an approach is not acceptable. To that aim, several *negotiation-based TMS* have been proposed.

TrustBuilder [Yu et al., 2003]

TrustBuilder was the first TMS to introduce the concept of trust negotiation. TrustBuilder uses X.509 certificates and TPL policies (cf. Section 3.3.1.1). The authors reused also the IBM-TE engine for the evaluation of policies. So TrustBuilder can be considered as an extension of IBM-TE to include negotiation features. The main novelty of this system lies in its rational management of credentials disclosure. To that aim, the trust engine is endowed with a negotiation module in which strategies are used to determine safe credentials disclosure for both parties involved in the interaction.

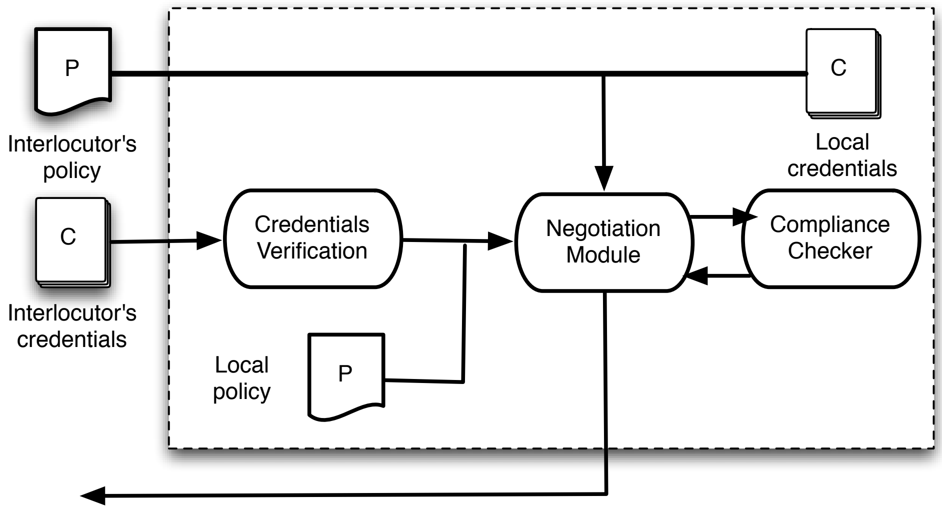


Figure 3.8 – Architecture of the TrustBuilder TMS

As illustrated in Figure3.8, TrustBuilder engine is split into three sub-modules: *credentials verification module*, *negotiation module* and *compliance checker module*. The core element of this architecture is the negotiation module which is responsible of enforcing negotiation strategies. The objective of a negotiation strategy is to minimise credentials disclosure. To that aim, TrustBuilder evaluates iteratively the policy of the interlocutor and the set of local credentials to compute the minimal set of credentials that satisfy the policy (as depicted in Figure 3.8).

Recently, Lee and colleagues proposed an extension of TrustBuilder that they called TrustBuilder2 [Lee et al., 2009]. This extension aims at endowing the TMS with four main functionalities: support of arbitrary policy languages, support of arbitrary credentials format, integration of interchangeable negotiation strategies and flexible policies and credentials management.

Fidelis [Yao, 2004]

Fidelis is a TMS that originates from the OASIS (Open Architecture for Secure, Interworking Services) distributed authorisation architecture project. Fidelis makes use of keys, X.509 and PGP as credential, and policies and credentials are systematically specified by distinct entities. Fidelis distinguishes two types of entities: *simple* and *composite* principals. Simple principals are in fact public keys while composite principals are groups of public keys (e.g. groups or communities).

The *Fidelis Policy Language* (FPL) is the core element of the system. The language is able to express recommendations and trust statements. The syntax of the language is presented in the following example.

```
any-statement: ind -> statement
asserts any-statement: self -> statement
where ind == 0x14ba9b925 || ind == 0x5918b01a || ...
```

Listing 3.4 – A blind delegation policy in Fidelis.

This policy represents a special type of delegation policies, called *blind delegation*. It is used to make an individual "blindly" trust and assert all assertions made by other individuals. In this example, the group of trusted individuals is constrained by the variable `ind`.

On the top of the FPL, the authors developed a trust negotiation framework in which meta-policies are used to specify negotiation strategies. Meta-policies are designed to express four types of conditions about when a credential could be disclosed: designated principal disclosure, context-specific disclosure, trust-directed disclosure, and mutual exclusion [Yao, 2004]. For instance, the following meta-policy is used to disclose the trust statement `T2(a, b): self->p` (which is used as a credential here) when negotiating with the `0xb258d29f` key holder.

```
negotiator(): self -> 0xb258d29f
grants disclose(T2(a, b): self->p)
```

Listing 3.5 – A credential disclosure meta-policy in Fidelis.

Fidelis does not support standard negotiation strategies. Thus termination property is not guaranteed making the evaluation of a policy not decidable in many situations. Finally, Fidelis distinguishes between *static* policies and *live* policies. Static policies do not depend on environment variables (e.g. date, time) to be evaluated, while live polices must be queried dynamically and tailored to each request. Nevertheless, live policies were only used in the context of negotiation as presented above and no adaptation mechanisms have been proposed as the description may suggest.

Trust-$\mathcal{X}$ [Bertino et al., 2003]

Trust-$\mathcal{X}$ is a TMS that was designed for trust negotiation in peer-to-peer systems [Bertino et al., 2003, Bertino et al., 2004]. Trust-$\mathcal{X}$ is built upon two bricks: $\mathcal{X}$-profiles and $\mathcal{X}$-TNL. $\mathcal{X}$-profiles are data structures used to store user's credentials along with uncertified declarations containing information about them (e.g. age, mail, address). $\mathcal{X}$-TNL stands for *XML-based Trust Negotiation Language*. $\mathcal{X}$-TNL has been developed for the specification of Trust-$\mathcal{X}$ certificates and disclosure policies.

```
<policySpec>
<properties>
       <certificate targetCertType= Corrier_employee>
       <certCond>
       //employee number[@code=Rental Car.requestCode]
       </certCond>
       <certCond> /.../[position=driver]
       </certCond>
       </certificate>
</properties>
       <resource target="Rental_Car"/>
       <type value="SERVICE"/>
</policySpec>
```

Listing 3.6 – Example of $\mathcal{X}$-TNL policy specification.

The code in Listing 3.6 shows an example of an $\mathcal{X}$-TNL policy defined by a rental car agency. The agency allows drivers of the Corrier society, which is part of the agency to rent cars without paying. This policy can be satisfied by providing a credential which is specified using the $\mathcal{X}$-TNL too. The syntax of a credential is described in the example provided in Listing 3.7.

```
<Corrier Employee credID='12ab', SENS= 'NORMAL' >
```

```
<Issuer HREF='http://www.Corrier.com' Title=Corrier Employees Repository/>
<name>
<Fname> Olivia </Fname>
<lname > White </lname>
</name>
<address> Grange Wood 69 Dublin </address>
<employee number code=34ABN/>
<position> Driver </position>
</Corrier Employee>
```

Listing 3.7 – Example of $\mathcal{X}$-TNL profile

The trust-$\mathcal{X}$ engine provides a mechanism for negotiation management. The main strategy used in Trust-$\mathcal{X}$ consists in releasing policies to minimise the disclosure of credentials. So, only credentials that are necessary for the success of a negotiation are effectively disclosed [Squicciarini et al., 2007]. Thus Trust-$\mathcal{X}$ makes use of a prudent strategy (cf. Section 3.2.4).

The primary novel aspect proposed in Trust-$\mathcal{X}$ consists in the use of *trust tickets*. Trust tickets are issued upon successful completion of a negotiation. These tickets can later be used in subsequent negotiations to speed up the process in case the negotiation concerns the same resource. Additionally, Trust-$\mathcal{X}$ provides also a mechanism to protect sensitive policies. This is achieved using *policy-precondition*; policies are sorted logically so that the satisfaction of a policy is the precondition of the disclosure of the subsequent policies.

Recently, Braghin [Braghin, 2011] proposed an interesting extension in which the framework is used to handle negotiations between groups of individuals instead of only between two individuals.

XACML

The *XML Access Control Markup Language* (XACML) [Humenn, 2003, Cover, 2007] is a generic trust management architecture. Even if in most of the works X.509 certificates are used with XACML, this framework accepts any credential format and thus is also generic with respect to credentials. XACML is also a standardised and interoperable policy specification language for expressing and exchanging policies using XML. The syntax of an XACML rule is defined as follows:

```
<Policy PolicyId="owner">
  <Rule RuleId="owner-r" Effect="Permit">
    <Condition>
      <Apply FunctionId=
          "urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <SubjectAttributeDesignator DataType=
            "http://www.w3.org/2001/XMLSchema#string">
          urn:oasis:names:tc:xacml:1.0:subject:subject-id
        </SubjectAttributeDesignator>
        <ResourceAttributeDesignator DataType=
            "http://www.w3.org/2001/XMLSchema#string">
          urn:emc:edn:samples:xacml:resource:resource-owner
        </SubjectAttributeDesignator>
      </Apply>
    </Condition>
  </Rule>
</Policy>
```

Listing 3.8 – A simple "owner" policy in XACML.

The above policy states that anybody can do anything with their own records. So in XACML the policies building bricks are rules. Each rule is composed of three main components: a *target*, a *condition* and an *effect* (cf. Listing 3.8). The target defines the rule scope (i.e. subject, actions and resources that the rule applies to); the condition specifies restrictions on the attributes of the subject and defines the applicability of the rule; the effect is either `permit` (the rule is called then a permit rule), or `deny` (the rule is called then a deny rule). When a request meets the rule target and satisfies the rule condition, the rule is applied and the decision specified in the effect is built, otherwise, the rule is not applicable and the request yields the decision `NotApplicable` [Li et al., 2009]. Policies group rules that relate to the same target and policies-sets group policies that relate to the same target.

The trust engine architecture is composed of several points, each representing a separate module: the Policy Decision Point (PDP) is the core of the architecture where policies are evaluated and decisions are made; the Policy Enforcement Point (PEP) is the point in which decisions are applied (e.g. issuing an authorisation); the Policy Retrieval Point (PRP) is the point in which policies are selected and retrieved from repositories; The Policy Information Point (PIP) is the point in which information is collected in the perspective of policies evaluation; and the Policy Administration Point (PAP) is the point in which policies are administrated (e.g, specified, updated, activated, deactivated, etc.).

Typically, the functioning of the XACML engine can be summarised as follows. The requester sends a request to the PEP (typically the application using the TMS), this request is then sent to the PDP which extracts the applicable policies with the help of the PRP. The PIP is then requested to collect and retrieve the credentials required for the evaluation of the policy. For that, the PIP interacts with the requester, the resources and the environment to extract each entity's attributes. Based on these information, the PDP evaluates the policy and provides an answer along with an optional obligation. This result is transmitted to the PEP that applies the obligation and grants access to the requester if the evaluation was positive.

Recently, Abi Haidar and colleagues [Abi Haidar et al., 2008] used an extended RBAC profile of XACML in the context of XeNA (The XACML negotiation of access), an access negotiation framework. XeNA brings together negotiation for trust establishment and access control management within the same architecture. XeNA trust engine is based on the TrustBuilder2 extended to support XACML access control and negotiation policies.

ATNAC [Ryutov et al., 2005]

Adaptive Trust Negotiation and Access Control (ATNAC) is an integrated TMS that combines two existing systems: GAA-API and TrustBuilder (already presented in Section 3.3.2). GAA-API is a generic authorisation and access control system that captures dynamically changing system security requirements. The system uses X.509 credentials to convey information between negotiating partners. ATNAC uses TPL to express trust policies that, in addition to the partner's properties, explicitly refer to the context suspicion level (SL). In ATNAC, several policies are specified and used to protect the same resource. So the main novelty of the system lies in the trust engine that monitors the environment suspicion level (thanks to GAAI-API) and adapts the selected policy based on the suspicion level. This mechanisms is used in ATNAC to provide adaptive trust negotiation in order to counter malicious attacks.

	Suspicion Level		
	low	medium	high
R_1	freely	freely	freely
R_2	freely	freely	C_1
R_3	freely	C_1	C_1 and C_2

Table 3.2 – Example of policies used in ATNAC (adapted from [Ryutov et al., 2005])

The above table illustrates the adaptive approach advocated by ATNAC. In this example, the resource R_1 is non sensitive and thus it is disclosed independently from the suspicion level. In contrast, R_3 can be freely disclosed in the context of *low* LS, requires a credential C_1 when SL is *medium* while both C_1 and C_2 are required when SL is *high*.

PROTUNE [Bonatti et al., 2008]

The *PRovisional TrUst NEgotiation framework* (PROTUNE) is a system that provides distributed trust management and negotiation [Bonatti and Olmedilla, 2005, Bonatti et al., 2008] features to web services. The PROTUNE framework provides: (a) a trust management language, (b) a declarative meta-language for driving decisions about information disclosure, (c) a negotiation mechanism that enforces the semantics of the meta-language, (d) a general ontology-based approach to support the policy language extension, and (e) an advanced policy explanation mechanism that is able to output *why*, *why not* and *how to* answers that are important during a negotiation process. One of the main advances made by PROTUNE lies in the use of *declarations* along with credentials during the policy evaluation process. Declarations are the unsigned equivalent of credentials. They can also be considered as statements that are not signed by a certification authority. However, the most novel part of the project remains the policy specification language which combines access control and provisional-style business rules.

In PROTUNE, policies are specified using the rule language defined in [Bonatti and Olmedilla, 2005]. The language is based on logic program rules extended with an object-oriented syntax[4].

```
allow(X,access(Resource)) :-
  goodReputation(X), validID(C).
validID(C) :-
      credential(identity, C[subject:X]).
goodReputation(X) :-
      declaration(Y,X,reputation(R)), Y!= X, R > 50.
```

Listing 3.9 – Example of a PROTUNE access policy.

The above PROTUNE policy states that an individual must provide a valid credential proving its identity and that he must have a good reputation to be allowed to access a resource. This kind of policy is called access policy. Similarly PROTUNE implements negotiation strategies through *release* policies which states under which conditions a credential can be released. An example of such resource is provided hereafter.

```
allow(release(credential(C[type:identity]))) :-
    credential(ta, Cred[issuer:'Trusted Authories']).
```

Listing 3.10 – Example of a PROTUNE release policy.

This policy states that identity credentials can be released only to individuals providing a credential that proves that they belong to the *'Trusted Authorities'* group.

3.4 Discussion

In the previous section, we have reviewed and analysed several trust management systems that we classified along basic components of trust management systems (cf. Section 3.2.3), namely *credentials*, *policies* and *trust engines*. The result of this comparison is summarised in Table 3.3.

3.4.1 Credentials

All reviewed systems rely on credentials in their trust management process. Credentials are used between the *trustors* and *trustees* to exchange *information* based on which trust can be established. With respect to that, credentials are essentially used as a means for bootstrapping

[4] $A.at : v$ means that the individual A has the attribute at and that this attribute has the value v. This expression is in fact an abbreviation of the logic predicate $at(A, v)$

trust between individuals that know little about each other. As presented in Section 3.2.3.1, there are two predominant approaches for credentials management, namely X.509 and PGP. So unsurprisingly, these two approaches have been used in most of the systems we reviewed in this chapter. However, due to its early availability X.509 was more supported than PGP. Nevertheless, even if they do not support PGP, many systems (e.g. IBM-TE) can easily integrate. This due to the fact that TMSs have a separate module that is responsible of credentials verification. Thus the consideration of a new formalism to represent credentials should be supported by all TMSs.

In terms of information, systems are split into three categories. In the first category, systems use credentials to support authentication. These systems inherit from IBAC models and use credentials to convey the identity of the holder (generally a key). Other systems such as PolicyMaker use credentials to represent *delegation of rights*. Finally, the last generation of TMS (e.g. TrustBuilder) makes use of fine-grained credentials in which all attributes of the holder can be represented (e.g. age, address, rights, roles). These systems inherit from ABAC and raised the *credentials privacy* issue which motivated *negotiation trust management systems*.

3.4.2 Policies

Policy specification languages represent a formal interface between humans and software agents (e.g. trust management systems). In order to be intelligible by both, these languages must satisfy some important criteria. De Coi and Olmedilla, on the basis of the work of Seamons and colleagues [Seamons et al., 2002], provided in [De Coi and Olmedilla, 2008] an exhaustive and comprehensive list of properties and requirements a policy specification language should satisfy in the context of trust negotiation. Some properties they reported are still valuable in the context of our work but most of them are either out of date or non relevant. In this section, we are interested in the formalism used to encode the policy, how expressive is this formalism, to which extent it is flexible and whether its management was purely syntactic or semantic.

3.4.2.1 Policies Formalism

Policies have been specified using four different formalisms. *Lisp* was primarily used in early systems such as PolicyMaker, KeyNote and REFEREE as their authors were interested in using the notion of *programmable policies*. For the same reasons, Ponder used an object-oriented approach for the specification of Ponder policies. However, these languages were rapidly abandoned in favour of more rigorous and application independent languages. For instance *logic programming* is the underlying formalism used in SULTAN, Fidelis and PROTUNE, while Binder, Cassandra and RT uses Datalog which is a subset of logic programming. Moreover, most of the commercial trust management systems (e.g. IBM-TE, XACML) used XML to represent policies. The main motivation in using XML was to make these policies interoperable and make policy specification more flexible [Yagüe, 2006].

3.4.2.2 Policies Expressiveness

The main objective of a policy is to express conditions. Thus, expressiveness is the most important property to be analysed in existing systems. Policy expressiveness have been interpreted in two different ways: based on the richness of the information used to state the policy, and based on the nature of the conditions stated by the policy.

The more a policy can express conditions on a wide range of information, the more it is considered to be expressive. With respect to that, TMS that make use of attribute-based approaches are the richest. For instance, languages such as PROTUNE, XACML, REFEREE, RT and ATNAC are known to be rich as they allow the specification of conditions over the attributes of the requester, the resources and the environment. In contrast, first general languages such as PolicyMaker or TrustBuilder have limited expressiveness as they rely on a unique type of information, namely identities or delegation credentials.

In terms of condition expressiveness, all reviewed policy languages proceed in the same way. Conditions are stated to set minimal values that are accepted for each information. Then policies are expressed by a conjunction or a disjunction of these conditions. This approach has a limited expressiveness as all conditions stated by a policy are considered to be equally important.

Moreover, many policy languages we reviewed adopted a binary evaluation in which either all conditions are satisfied or no condition is considered to be satisfied (e.g. early TMS). With respect to that, trust negotiation systems (e.g. TrustBuilder, Trust-$\mathcal{X}$, PROTUNE, etc.) are more flexible as the evaluation of a policy is progressively achieved. Nevertheless, no system considered partial policy evaluation in which the policy satisfaction level is computed even if all conditions are not met.

3.4.2.3 Policies Flexibility

In the literature, flexibility refers to the ability of a policy to be changed without stopping the system. With respect to that, almost all systems we reviewed are flexible as by definition policies are flexible. However, in our thesis we use a narrow definition of flexibility: here, flexibility refers to the ability of a policy to be adapted. Adaptation can be performed by adding, removing and/or updating any of the conditions stated by the policy.

Reviewed policy languages have not been designed in that direction. Therefore, none of these language can be easily adapted based on an automated process. However, some language such as XACML, Cassandra, $\mathcal{X}$-TNL, and Ponder have been designed with a particular care for the extensibility of their policies. If automated, these extensibility mechanisms can be used to produce a new policy which will substitute to the former one.

3.4.2.4 Policies Semantic

Semantic aspects was clearly not a concern for early TMS (e.g. PolicyMaker, REFEREE, IBM-TE). The objective at that time was to enable decentralised trust management in which all parties are assumed to agree on the semantics of the credentials and the policies they exchange. And even if semantic was recognised as an important feature of negotiation TMS, the interpretation of the semantic policies were limited to formal aspects. With respect to that, logic-based policy languages (e.g. Binder, SULTAN *RT* and PROTUNE) are considered to have a better and more formal semantic than XML-based ones (e.g. TPL, XACML and $\mathcal{X}$-TNL) or object-oriented ones (e.g. Ponder). Nevertheless, no policy language those that we reviewed makes an explicit use of ontologies such it has been advocated in recent works (e.g. [Sensoy et al., 2010] and [Finin et al., 2008]). In the light of this, all policy languages used in trust management have limited semantic capabilities.

3.4.3 Trust Engine

As stated before, we are interested in evaluating the extent to which the TMSs we analysed tackles the *challenges* we raised by virtual communities. To that aim, we compared these systems with respect the objective we set up in our thesis (cf. Section 1.3).

3.4.3.1 Social-awareness (Objective 1)

While we evidenced in the previous chapter how trust is intrinsically a social concept, surprisingly most of the reviewed trust management systems have neglected this dimension of trust. Thus all system we review are simply not able make the user of the system *specify* (Objective O1.1) and *enforce* (Objective O1.2) collective policies. In these system, a resource belongs to one and only one individual. If not, there should be a system administrator that specifies the policy to be used when granting access to common/shared resources. Such a *centralised* vision is simple not application as we stressed in the introduction of this thesis.

As showed in Table 3.3, inly XACML provides some mechanisms that could be used to implement *social-awareness*. These mechanisms allows independent entities, each having its own policy, the made trust decisions about a shared resource. These mechanisms are quite limited with respect to our objective but they represent interesting paths that we will explore in the next chapter. For *Objective O.1.2* which consist in enforcing collective policies, this issue has not been addressed in existing systems too.

3.4.3.2 Context-awareness (Objective 2)

A system is said to be *context-aware* if the trust evaluations it makes are influenced by the context in which these decisions are made. In other words, a trust management system is said to be context-aware if the policies based on which it performs its trust evaluations can be adapted in response to context changes. This capability is desired for both *individual* (Objective O.2.1) and *collective* (O.2.2) policies.

Surprisingly, a wide range of the works we reviewed could not be considered to be *context-aware*. Although, the policies used by some system consider the context in their trust evaluation, this context is never used to change the conditions stated by the policy. However, we can consider all trust management could be made (i.e. could be achieved) *context-aware* (cf. Table 3.3) as they make use of declarative policies and thus these policies could be adapted at runtime.

Worth noting, however, *RT* provides also a simple mechanism based on which running policies are activated and deactivated. However, the conditions stated by these policies are never updated based on the running context. *Fidelis* provides an interesting idea with the concept of live policies. live policies were only used in the context of negotiation as presented above and no adaptation mechanisms have been proposed as the description may suggest. Finally, *ATNAC* is the only system that supports the automatic change of running policies. These changes are based on the suspicion level but are limited to activation and deactivation of the policies used by the system. In fact, the user specifies several policies, each for a particular suspicion level, then the appropriate policy is used based on the current suspicion level. If we have to use this system for virtual communities, that means that a resource owner has to specify a policy not only for each resource, but also for each state this resource could reach. Of course, performing such a complex task is likely to not be feasible.

3.5 Conclusion

In this chapter, we introduced basic concepts of trust management and analysed the most interesting trust management systems.

We evidenced from the previous discussion that the surrounding context has been tackled in a very limited way. The conditions stated in the policies make a clear reference to this context, but when the context changes these policies are not updated. Importantly, existing trust management systems seem to neglect or have not been designed for the social dimension that may constrain/influence the trust decision. Although not explicitly mentioned, social context can arguably be considered as included in all interactions involving more that on individual. Thus any interaction is necessary influence or constrained by a social context. With respect to that, almost all the system we presented in this chapter reveal to be inaccurate to handle this social context.

In the light of these conclusions and limitations, we will explore in the next chapter existing approaches with respect to the consideration of this social context. We will particularly focus on how *collective policies* are specified, enforced and evolved in situations involving a social structures such as virtual communities.

	Credentials		Policy Language				Trust Engine	
	Formalism	*Information*	*Formalism*	*Exp*	*Flex*	*Sem*	*Saw*	*Caw*
PolicyMaker [Blaze et al., 1996]	NA	delegation	LISP	-	±	-	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
REFEREE [Chu et al., 1997]	PICS	attributes	LISP	±	-	-	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
Binder [DeTreville, 2002]	X.509	delegation	Datalog	±	±	-	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
SULTAN [Grandison, 2003]	Prolog	delegation	Logic Programming				$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
Ponder [Damianou et al., 2001]	NA	identity	Object Oriented	+	-	-	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
IBM TE [Herzberg et al., 2000]	X.509/PGP	identity	XML	-	±	+	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
RT [Li et al., 2002]	X.509	attributes	Datalog	±	-	-	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
Cassandra [Becker and Sewell, 2004]	Datalog	attributes	Datalog	±	±	-	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
TrustBuilder [Yu et al., 2003]	X.509/PGP	attributes	XML	+	±	-	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
Fidelis [Yao, 2004]	Prolog	identity	Logic Programming	±	±	-	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
Trust-$\mathcal{X}$ [Bertino et al., 2003]	$\mathcal{X}$-TNL	attributes	XML	+	±	±	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
XACML [Cover, 2007]	X.509	attributes	XML	+	±	±	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
ATNAC [Ryutov et al., 2005]	X.509	attributes and context	XML	+	±	-	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$
PROTUNE [Bonatti et al., 2008]	NA	attributes and context	Logic Programming	+	±	+	$O^{1.1}$ $O^{1.2}$	$O^{2.1}$ $O^{2.}$

Table 3.3 – Summary of the comparison of trust management systems.

- $O^{X.X}$: Achieved
- $O^{X.X}$: Achieved but manually
- $O^{X.X}$: Could be achieved
- $O^{X.X}$: Not achieved
- Exp: Expressiveness
- Flex: Flexibility
- Sem : Semantic
- Saw : Social Awareness
- Caw : Context Awareness

3.6 French Summary

Dans le chapitre 2, nous avons montré comment la confiance était devenue un concept incontournable dans les systèmes combinant risque et incertitude. Dans ce chapitre, nous nous positionnant dans le domaine de la sécurité et retraçons les évolutions qu'a connues ce domaine pour arriver aux systèmes de gestion de la confiance telle que nous les connaissons d'aujourd'hui.

3.6.1 Du contrôle d'Accès à la Gestion de la Confiance

Les systèmes de gestion de la confiance puisent leurs racines dans la sécurité et plus particulièrement des mécanismes de contrôle d'accès. Le contrôle d'accès est le mécanisme qu'utilisent les applications (à l'origine les systèmes d'exploitation) pour répondre à la question (i.e., requête) "est ce que l'entité identifiée en tant S (i.e., sujet) peut manipuler l'objet O en performant l'action A ". Dans cette section, nous allons retracer l'évolution des différents modèles qui ont été proposés pour répondre à cette question. Nous distinguons cinq types de modèles de contrôle d'accès que nous décrivons brièvement ci-dessous.

- **Contrôle d'accès à base d'identité (IBAC):** le contrôle d'accès est généralement subdivisé en deux étapes qui sont l'authentification et l'autorisation. L'authentification vis a assurer que l'identité du demandeur est authentique alors que l'autorisation détermine s'il existe une permission qui autorise le possesseur de cette identité de manipuler la ressource et sous quelles conditions. Les contrôles d'accès à base d'identité confondent les deux étapes et associent donc l'identité des individus avec leurs permissions.

- **Contrôle d'accès à treillis (LBAC):** Dans certaines situations, le contrôle d'accès dépend plus des propriétés des objets et des caractéristiques du demandeur que sur son identité. Dans ce type de mécanismes, les étiquettes ou labels sont associés aux objets et au sujet du système. Ensuite, les mêmes réglés sont appliquées pour déterminer si un individu ayant un label donné à droit d'accéder à une ressource. Ainsi, le processus de contrôle d'accès se réduit en un contrôle du flot de données au sein du système. Par exemple, une lecture dans un fichier est considérée comme un flot de données du fichier vers le sujet. Afin d'assurer la protection des données confidentielle, deux principes sont mis en œuvre: "pas de lecture vers le haut " et "pas d'écriture vers le bas ". Le premier assure que les individus avec un label inférieur ne peuvent pas avoir accès à des données d'ordre supérieur. Le second assume que les individus d'un niveau supérieur n'ont pas autorisé à écrire sur des ressources de niveau plus bas.

- **Contrôle d'accès à base de rôle (RBAC):** les modèles à base d'identité comme les modèles à base de treillis ont des limites considérables. Les premiers sont difficiles à concevoir (il faut connaître tous les acteurs et les ressources à l'avance) et à maintenir, alors que le second est trop rigide et ne donne aucune liberté au créateur de la ressource. C'est en voulant proposer une alternative plus flexible que les chercheurs ont eu l'idée d'utiliser les rôles pour gérer les droits d'accès au sein d'organisations.

- **Contrôle d'accès à base d'organisations (OrBac):**Le contrôle d'accès à base d'organisations et les contrôles d'accès à base de rôles ont beaucoup de points en commun. Par exemple, dans les deux modèles le concept de rôle est central. C'est pourquoi, la plupart des chercheurs du domaine présentent OrBAC comme une extension de RBAC. En réalité, OrBAC a permis d'affiner la définition de permission, tâche qui était laissée à la charge de l'administrateur sur RBAC.

- **Contrôle d'accès à base d'attributs (ABAC):** la principale idée du modèle ABAC consiste à utiliser les attributs des politiques font référence aux attributs du demandeur au lieu de son identité, de label ou de son rôle comme ce fut le cas dans les modèles vus précédemment (IBAC, LBAC, RBAC et OrBAC). La satisfaction de ces politiques repose ensuite sur la production de certificats (credentials) établies par des autorités de certification jouant le rôle de tiers de confiance dans l'interaction. Par exemple, un politique pourrait statuer que seules les personnes capables de conduire une voiture peuvent en louer une. Ainsi, les agents de locations vont devoir exiger de chaque client de produire une preuve (i.e. un certificat) pour cet attribut. Le rôle de certificat est rempli ici par le permis qui est établi par la préfecture de police. Cette institution joue le rôle d'autorité centrale et permet l'établissement, de facto, de la relation de

  À l'instar d'IBAC, LBAC, RBAC et OrBAC, dans ABAC les politiques peuvent utiliser toutes sortes d'attributs pour l'établissement d'autorisations. Nous classons ces attributs en trois catégories: (i) attributs relatifs au sujet de l'interaction, (ii) Attributs relatifs à l'objet de l'interaction, (iii) attributs relatifs au contexte de l'interaction.

3.6.2 Gestion de la confiance

La principale limite des modèles de contrôle d'accès présentés dans la section précédente repose sur leur dépendance vis-à-vis de l'identité. À l'aube de l'Internet, l'identité constituait un défi technique pour les chercheurs (i.e., gestion décentralisée), mais également éthique (i.e., respect de la vie privée). L'évolution des modèles de confiance vers du ABAC a permis certes d'améliorer le second point, mais ABAC était incapable de permettre un contrôle d'accès décentralisé en raison de sa dépendance vis-à-vis des autorités de certifications et la nécessite d'avoir recours à un point d'évaluation centrale.

C'est dans ce contexte qu'au début des années deux mille, Matt Blaze et ses collègues [Blaze et al., 1996, Blaze et al., 1999a] a introduit une nouvelle discipline de recherche dédiée au mécanisme de contrôle d'accès distribué et décentralisé. Blaze et ses collègues ont introduit alors le terme “gestion de la confiance " dans le dictionnaire de la sécurité et marqués de ce fait la naissance d'une nouvelle branche dans le domaine de la sécurité informatique.

Jøsang [Jøsang et al., 2007], a défini la gestion de la confiance comme étant “l'activité de collecte, codification, analyse et de représentation des preuves dans la perspective de faire des évaluations et prendre des décisions dans le cadre de transaction électronique ". Cette définition a été reprise par Grandison [Grandison, 2003] afin d'y ajouter la nature des preuves récoltées (compétence, honnêteté, sécurité) ainsi que l'objectif de cette évaluation (prendre des décisions à propos de relations de confiance dans le cadre d"application internet).

Dans notre manuscrit, nous nous sommes basé sur ces deux définitions afin de proposer la nôtre.

Definition 7 (Gestion de la confiance) *L'activité automatique de collecter, réclamer, fournir et analyser les informations dans la perspective de prendre des décisions de confiance (e.g., contrôle d'accès, délégation, collaboration) à base de politiques.*

3.6.3 Systèmes de gestion de la confiance

Les systèmes de gestion de la confiance permettent à une application de savoir si une requête de réaliser une action, potentiellement dangereuse, sur une ressource sensible respect la politique de contrôle d'accès [Blaze et al., 1996, Blaze et al., 1999a]. Néanmoins, ce terme est actuellement utilisé dans un sens plus large pour décrire des systèmes permettant d'évaluer si une décision se respecte une décision de confiance.

Concrètement, un système de gestion de la confiance permet d'établir une relation de confiance entre le demandeur et le demandé. Cette relation permettra à au demandé d'appréhender le risque et de prendre une décision de confiance. Pour cela, les systèmes de gestion de la confiance fournissent un langage pour la spécification de *politiques de confiance* ainsi que de *certificats* dits credentials. Ces systèmes proposent également un moteur de confiance qui va permettre de vérifier si les credentials fournis satisfont la politique qui a été spécifiée. Ces trois éléments constituent les briques de base de tout système de gestion de la confiance. Dans ce qui suit, nous présentons brièvement chacun des trois éléments :

- *Credentials :* Ce sont le contrepartie des lettres de créance papier utilisée dans le monde réel (e.g. passeport, permis de conduit, carte de crédit). Ils constituent des documents numériques certifiées (i.e., signés) par des autorités de certification. Les carentiels permettent l'authentification des utilisateurs mais peuvent être également utilisée pour affirmer des informations sur des attributs de l'utilisateur (e.g. son âge, son adresse, ses origines). Il existe deux approches de définition de credentials: par *autorités de certification* et par *certification croisée*. La première repose sur la confiance que portes l'individu aux institutions (e.g. préfecture, mairie) alors que la seconde fait référence à la confiance issue de l'expérience des autres (réputation, bouche à oreille). Dans le manuscrit, nous avons illustré chacune des deux approches avec des implémentations concrètes, respectivement, *X.509* [Samarati and Vimercati, 2001] et *PGP* [Gerck, 2000, Prohic, 2005].

- *Politiques de confiance :* Les politiques ont été largement utilisé en informatique. Initialement introduites pour automatiser des tâches (e.g. exécution en mode batch). Néanmoins, les politiques sont de nos jours essentiellement pour permettre à des systèmes de modifier leur comportement de manière dynamique. Ainsi, le système est en mesure de changer de comportement dans avoir a être arrêté. Cependant, malgré leur succès, leur définition demeure assez floue et les définitions avancées par certains auteurs sont soit trop vagues ou au contraire trop spécifiques. Dans cette thèse, nous proposons de définir les politiques de confiance en tant que expression des conditions qu'une entité (humain ou agent) doit satisfaire pour qu'il soit considéré digne de confiance dans un contexte donné (e.g. contrôle d'accès, délégation). Ainsi, de notre point de vu, les politiques ont un rôle double: (i) ils permettent au propriétaire de ressources de spécifier les politiques que sont systèmes de gestion de la confiance va utiliser pour faire ses évaluations, (ii) ils permettent a deux interlocuteur d'échanger les conditions respectives ainsi que les credentials permettant la satisfaction de celles-ci.

- *Moteur de confiance :* L'objectif d'un moteur de confiance est d'évaluer si les crendentials, ou plus largement les informations, fournis par le demandeur sont valides et s'ils permettent de satisfaire la politique spécifiée par celui-ci. Il faut souligner que les moteurs de confiance ne sont nullement responsables de la prise de décision de confiance. Leur tâche se limiter à évaluer si le demandeur parvient à satisfaire la politique ou pas. Ainsi, le principal intérêt d'utiliser un système de gestion de la confiance est de décharger l'application y faisant appel des tâches fastidieuses de vérification des credentials ainsi que l'évaluation des politiques comme évoqués précédemment.

3.6.4 Analyse des Systèmes de gestion de la confiance

Dans cette section, nous passons en revue une sélection des systèmes de gestion de la confiance. Comme précisé dans le chapitre précédent, ces travaux sont scindés en deux catégories : les *Systèmes de négociation de confiance automatique* et les *Systèmes de gestion de la confiance décentralisée*. Cette dernière catégorie est ici divisée en deux sous-catégories: les *Systèmes à base d'autorisation* et les *Systèmes à base de rôles*.

3.6.4.1 Systèmes à base d'autorisation

Cette catégorie comporte l'ensemble des travaux qui sont considérés comme étant les pionniers de la discipline. Parmi ces travaux, on peut citer PolicyMaker dont les auteurs ont introduit le terme *gestion de la confiance* [Blaze et al., 1996]. On peut citer égalent REFEREE [Chu et al., 1997], Binder [DeTreville, 2002], SULTAN [Grandison, 2003] et Ponder [Damianou et al., 2001]. La plupart de ces systèmes peuvent être considérés comme étant obsolètes de nos jours. Néanmoins, les mécanismes qu'ils ont introduits ont inspiré bon nombre de systèmes utilisé de nos jours. Ces systèmes reposent sur le mécanisme de délégation. Chaque utilisateur peut émettre un certificat qui va lui permettre de certifier la relation de confiance qu'il a avec le bénéficiaire du certificat. De son coté, le propriétaire de la ressource définit une politique qui délimitait la porté des personnes en qui il a confiance via ce mécanisme de délégation. Finalement, le rôle du moteur de confiance consiste à trouver une chaîne de délégation valide et qui satisfait la politique du propriétaire et qui permet de relier le demandeur à la ressource demandé.

3.6.4.2 Systèmes à base de rôles

Dans le système présenté dans la section précédente, la délégation des droits était utilisée d'une manière assez restrictive. Par exemple, dans PolicyMaker, il est impossible à un individu A de spécifier que tous les membres appartenant à son groupe ont droit d'accéder aux ressources que A possède. Pour cela, il aurait fallu recenser l'ensemble des membres du groupe et spécifier des autorisations individuelles. Sinon, donner le droit à un membre et donner le droit à celui-ci de déléguer ce droit aux autres. Ces deux solutions demeurent faisable mais complexifient grandement la gestion des autorisations. Afin de pallier à ce genre de situations, une nouvelle génération de systèmes de gestion de la confiance à vu le jour. Ces systèmes se sont inspirés ont essayé de tirer profit de la force de RBAC dans tout ce qui concerne la rationalisation des autorisations. Ainsi, l'utilisation combinée des rôles et la gestion distribuée des crendials à permis à cette nouvelle génération de systèmes de prendre rapidement le dessus sur les systèmes classiques reposant sur les autorisations individuelles. Parmi ces travaux, on peut citer IBM Trust Establishment [Herzberg et al., 2000], Role-Based Trust Management (RT) [Li et al., 2002] et Cassandra [Becker and Sewell, 2004].

3.6.4.3 Systèmes à base de négociation

Traditionnellement, les systèmes de gestion de la confiance permettent l'établissement de relations de confiance de manière unilatérale. Donc, le propriétaire de la ressource (demandé) est supposé être digne de confiance pour le demandeur. Donc le demandeur doit fournir tous les credentials requis par le propriétaire afin de savoir si sa demande est accepté ou non. Or, si elle ne l'est pas, il aura divulgué des informations potentiellement sensibles (e.g. son âge, son adresse et ses coordonnées) pour rien. Ainsi, suite à la sensibilisation croissante des utilisateurs de l'Internet pour ce qui concerne la diffusion de leurs informations privées, les approches classiques de gestion de la confiance ne sont plus tolérables. Pour ces raisons, une nouvelle génération de systèmes de gestion de la confiance, appelé systèmes de négociation de confiance, à vue le jour dans la perspective de permettre l'établissement de la relation de confiance de manière incrémentale et sûre pour les deux interlocuteurs. Plusieurs systèmes tels que Trust-Builder, Fidelis, Trust-$\mathcal{X}$ [Bertino et al., 2003], XACML [Humenn, 2003, Cover, 2007], ATNAC [Ryutov et al., 2005] et PROTUNE [Bonatti et al., 2008]. Le principal facteur de différenciation de ces systèmes réside sur l'expressivité du langage de politiques qu'ils proposent ainsi que de l'efficacité de la stratégie de négociation qu'ils proposent.

Chapter 4

The Social Dimension of Trust Management

In the previous chapter, we observed that the objective of existing trust management systems was to make trust decisions safe for the individual. However, in this chapter we are interested in mechanisms that make such decisions collectively harmful too.

Indeed, based on the assumption that individuals tend to perform better when they act together than they can do alone, the concept of community has become very common to enable collaboration in nowadays human societies and virtual ones as well. In such context, it is a very common situation that individuals come to table with different backgrounds, skills, and even policies. If backgrounds and skills heterogeneity represents the richness of these systems, policies heterogeneity may represent a burden to collaboration if these policies conflict and no mechanisms exists to resolve such conflicts.

In what follows, of this chapter, we analyse mechanisms that could be used to address such issue. We are particularly concerned by answering the following questions:

1. how virtual community members make trust decisions which consequences may affect the other members,
2. whether and how being in a community affects the way a member makes his trust decisions,
3. whether and how a member may affect the way other members make their trust decisions.

In the light of this, we provide in Section 4.1 a detailed insight on how researchers in trust management addressed the above issues. Drawing on this, we looked for other disciplines in which similar phenomena have been studied and understood. To that aim, we reviewed existing theories and models that originate from flagship disciplines involving both individual and collective dimensions, namely sociology and multi-agent systems [Coleman, 1990, Yao, 2004].

4.1 Social dimension of Trust Management

In trust management and access control, policies are used to make decisions that emanate from and express the requirements of a single entity. However, when it comes to virtual communities, such assumption does not hold. The consequence of a trust decision is borne by all the members, making the consideration of the policy of each individual paramount for the community cohesion.

In what follows, present two main approaches, namely *combination* and *integration*, that were proposed to address the above issue. The objective of these works were to allow different partners with personal access control policies to make decisions that satisfy the policies of all partners. Then we present the limitations of both approaches and discuss how trust management systems failed to address the *social dimension* of trust decisions.

4.1.1 Policy Combination

Policy combination has been introduced in the context of XACML. As mentioned in Section 3.3.2, XACML is a well established policy language which defines three principal elements : *rules*, *policies* and *policies-sets*. The effect (i.e. decision) of the policy is obtained by combining the effects of the sub-policies or rules according to some *combination* strategy [Li et al., 2008, Rao et al., 2009].

Before we delve in policy combination mechanisms, let us recall the basic elements XACML is built upon. XACML rules are composed of three main components: *target*, *condition* and *effect*. The target defines the rule scope (i.e. subject, actions and resources that the rule applies to); the condition specifies restrictions on the attributes of the subject and defines the applicability of the rule; the effect is either `permit(P)` (the rule is called a permit rule), `deny(D)` (the rule is called then a deny rule). The decision output can be either `Permit` (`P`) or `Deny` (`D`) or `NotApplicable` (`NA`).

Policies group rules that relate to the same target. Thus, policies are composed of a set of rules, a target (which is identical to the one contained by the rule), a *rule-combination* algorithm (RCA), and *obligations*. Similarly, policies-sets group policies that relate to the same target. Consequently, policies-sets are composed of a set of policies (or policies-sets), a target, a *policy-combination* algorithm (PCA) and obligations. Obligations represent actions to be triggered when enforcing the (combined) decision. Finally, the (rule or policy) *combination-algorithm* specifies how the decision of each policy (or rule) are combined to derive a unique decision.

As sketched in Figure 4.1, for each request, several rules or policies may apply. So in order to solve eventual conflicts (when rules/policies do not compute the same decision), XACML provides six standard policy (resp. rule) combination algorithms. These algorithms applies to both rules (RCA) and policies (PCA) except for the last algorithm that applies only to policies. These algorithms are described as follows:

- **Deny Overrides**: This algorithm prefers “`Deny`” (D) to “`Permit`” (P) to '“NotApplicable” (NA). In other words, if the evaluation of any policy or rule is `D`, the result of the whole will be `D`. Otherwise, if no policy is evaluated to `D` and some policies are evaluated to `P` (and thus the rest to `NA`), then the result will be `P`. Otherwise, the

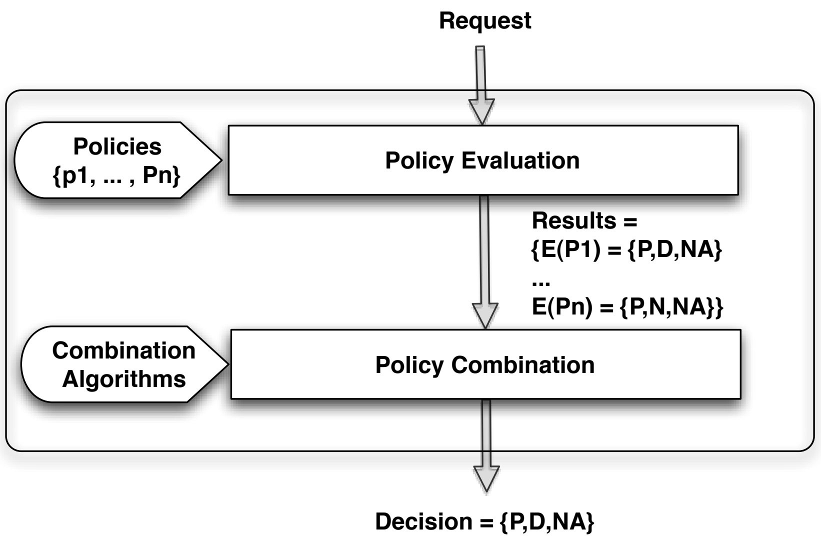


Figure 4.1 – A sample policy combination process.

result will be by default `NA`.

- **Ordered Deny Overrides**: This algorithm is similar to the Deny-Overrides except that policies are evaluated in the order in which they appear.
- **Permit Overrides**: In this algorithm, "`Permit`" is preferred to "`Deny`" and "`Deny`" is preferred to "`NotApplicable`".
- **Ordered Permit Overrides**: This algorithm is similar to the Permit-Overrides except that policies are here evaluated in the order they appear.
- **First Applicable**: This algorithm returns the result of the first policy (or rule) that can be applied to the request.
- **Only One Applicable**: This algorithm only applies to policies and not rules. It returns the result of the unique policy that applies to the request. If such policy does not exist, it returns `NA`.

Conceptually, combination algorithm has as inputs the set of policies, and a set of credentials provided by the application that wants to access the resource. The algorithm first selects the policies that apply to the request based on their target field, then evaluates each policy with respect to the set of credentials. Once all policies have been evaluated, the algorithm combines all the decisions to compute a final one.

The policy combination approach assumes that all parties concerned by a decision are available when a decision has to be made. In open and distributed systems, such condition would be hard to satisfy. Therefore, we present in the next section two alternative approaches.

4.1.2 Policy Integration and Composition

Policy integration and policy composition refer to the same concept: building a new (integrated or composed) policy based on policies that have been specified by multiple and distinct parties. Thus, in policy integration (we keep this term henceforth) decisions that concern multiple individuals are made based on a process under which individual policies are integrated to build a collective one.

In what follows, we present two approaches in which an algebra for policy integration has been proposed.

4.1.2.1 Algebra for Composing Policies

Bonatti and colleagues [Bonatti and Samarati, 2002] were the first to propose an algebra for policy integration. The authors integrate access control policies using logic programming and partial evaluation techniques for algebraic expressions [Rao et al., 2009].

Bonatti and colleague used an abstract form of policies that are defined based on three sets, S, O and A, denoting respectively subjects, objects and actions. So the authors assume a generic approach as they do not make any assumption on the subject, the objects and the actions. Likewise, the formalism in which policies are expressed is too generic to be considered as a policy language. This explains why we did not present it in details neither here nor in Section 3.3.

Depending on the particular application, subjects can be users, roles or groups, objects can be files, relations or any artefact, while actions are operations that subjects can perform on objects (e.g. read, write, update). Policies are defined as sets of ground (which do not contain any free variables) authorisation terms. These policies specify constraints that are formulated as first-order predicates (e.g. $isOwner(s, o)$ that is evaluated as true if the subject s owns the object o).

In order to understand and illustrate the composition operators we present hereafter a concrete example. In the example we assume the existence of two policies (i.e. P_1, P_2) which first authorises $Alice$ to read a $file_1$, and Bob to access $service_2$. And the second policy states that $Alice$ is authorised to write on $file_1$. These two policies are specified as follows:

- $P_1 = \{(Alice, file_1, read), (Bob, service_2, access)\}$
- $P_2 = \{(Alice, file_1, write)\}$

Based on the policies framework, Bonatti and colleagues proposed a set of six composing operators. Among the six operators, three worth to be explained as the others do not integrate policies.

Addition (+) integrates two policies by returning their union. For instance, with respect to the above example policies P_1 and P_2, the integrated policy would be defined as follows:

$$P_1 + P_2 = \{(Alice, file_1, read), (Bob, service_2, access), (Alice, file_1, write)\}$$

This integrated policy would authorise *Alice* to read and write on $file_1$ in addition to authorising *Bob* to access $service_2$ (cf. [Bonatti and Samarati, 2002] for a detailed description).

Conjunction (&) integrates two policies and returns their intersection. The integrated policy would be defined as follows:

$$P_1 \& P_2 = \{(Alice, file_1, read)\} \tag{4.1}$$

The resulting policy will only authorise *Alice* to read $file_1$ as both policies agreed on this privilege. So while addition enforces maximum privilege, conjunction enforces minimum privilege.

Subtraction (−) integrates two policies by eliminating all authorisations that are not permitted by the second policy. The integrated policy would be defined as follows:

$$P1 - P2 = \{(Alice, file_1, write)\} \tag{4.2}$$

The resulting policy will only authorise *Alice* to only write on $file_1$ as the second policy does not authorise to access $service_1$.

The three remaining operators are not described either because they only restrict one policy (e.g. closure ($*$) and scoping restriction (ˆ)), or because they can be obtained using the three operations aforementioned. In Figure 4.2, the authors summarised the operators of the algebra we presented above and their semantics and illustrated two possible graphical representations of algebraic operations.

The above algebra for policy composition is relatively intuitive despite the complexity of its foundations. However, the authors did not provide any concrete implementation for their framework and they did not specify how each operator could be implemented. Further, some scholars criticised the fact this approach adopts a binary evaluation (e.g. `Accept` or `Deny`) and does not handle three-valued outputs such the one presented in Section 4.1.1 (i.e. `Accept`, `Deny`, `NotApplicable`).

4.1.3 Algebra for Fine-Grained Integration of Policies

Mazzoleni et al. developed a policy integration process on the top of the XACML policy language [Mazzoleni et al., 2006]. This work represents an extension of combination mechanism (cf. Section 4.1.1) in which integration algorithms are used in complement to combination algorithms. These algorithms specify how individuals want their policies to be integrated in case there is a need to integrate them with other policies (i.e. specified by other individuals). Consequently, each individual involved in the system has to provide the policy he is using and how this policy should be integrated with other policies. The decision regarding policy integration is therefore automatically computed by taking into account the requirement of each party involved in the system [Rao et al., 2011].

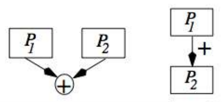
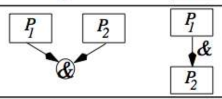
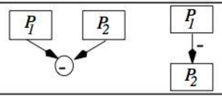
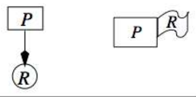
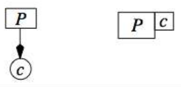
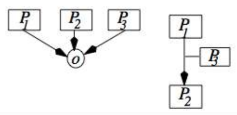
Operator	Semantics $[\![\ ]\!]_e$	Graphical representation
$P_1 + P_2$	$[\![P_1]\!]_e \cup [\![P_2]\!]_e$	
$P_1 \& P_2$	$[\![P_1]\!]_e \cap [\![P_2]\!]_e$	
$P_1 - P_2$	$[\![P_1]\!]_e \setminus [\![P_2]\!]_e$	
$P * R$	$T_{R \cup [\![P]\!]_e \cup B} \uparrow \omega$	
$P\hat{\ }c$	$\{t \in [\![P]\!]_e \mid t \text{ satisfy c}\}$	
$o(P_1,P_2,P_3)$	$[\![(P_1 - P_3) + (P_2 \& P_3)]\!]_e$	

Figure 4.2 – Operators of the algebra for composing policies and their graphical representation [Bonatti and Samarati, 2002].

Even if they claim an algebraic approach, the main novelty of their work lies in the possibility to define similarity-based combination algorithms. So the integrating algorithms rely heavily on the mechanisms they defined for analysing the similarity of two policies which we consider as the most important part of the work to be described.

Policies Similarity

Policies similarity aims at analysing the behaviour of the policies to be integrated based on their effects (i.e. `Permit`, `Deny` or `NotApplicable`). Concretely, the process tries to detect if there is any correlation between the effect of two (or more) policies. For instance, the integration process may be confronted with situations in which requests that are permitted by the first policy (P_1) are denied by the second one (P_2). The authors refer to such a situation by the "*Restricts*" notion (i.e. P_2 Restricts P_2).

In the light of this, the authors proposed a three-phase process for integrating policies: (i) policies are decomposed into rules and similarity between rules is computed, (ii) the similarities of the groups are grouped and simplified based on their effects, (iii) the policy similarity is extracted using the combination algorithms specified in the policies.

In *rule similarity computing*, the objective is to detect any similarity between each of the rules that compose the policies to be integrated. Two rules are said to have a similarity if they define conditions on the same attributes (e.g. age). Once the similarity is identified, the process identifies which rule uses the most restrictive condition for these attributes.

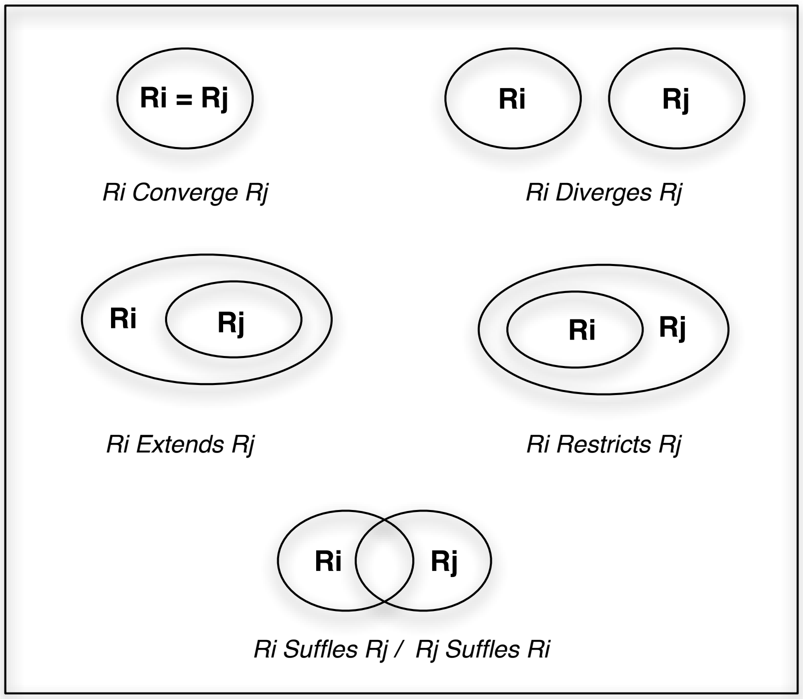


Figure 4.3 – Types of similarities between rules.

As illustrated in Figure 4.3, the authors distinguish four forms of rule similarities that we briefly summarise hereafter.

Converge: two rules converge if they are satisfied by the same set of credentials.

Diverge: two rules diverge if they are satisfied by two distinct sets of credentials.

Restrict and Extend: a rule restricts (resp. extends) another rule if it can be satisfied by a subset (resp. a superset) of the credentials that satisfy the rule it restricts (resp. extends).

Shuffle: two rules shuffle if they are satisfied by two intersecting sets of credentials.

Once the similarity has been computed, rules are grouped based on their similarities and their effects (i.e. `Permit`, `Deny` or `NotApplicable`). Then transformations are applied to these rules to reduce the number of similarities to be considered. The objective is to obtain two sets of rules; those having the effect `Permit` and those having the effect `Deny` (cf.

[Mazzoleni et al., 2006] for more details). As a final step, the authors complete the policy similarity process by considering rule combination algorithms specified by the issuer of the policies that are integrated.

The process of computing policies similarities given by the authors is very important as it gives valuable information about which policy is the most restrictive. However, the complete integration process goes further as it uses this information alongside with the integration preferences of all entities involved. To that aim, the authors introduced four preference types with respect to policy integration. These preferences are defined as follows:

Converge Override: does not consider policies other than the ones it specified.

Restrict Override: does not consider policies that authorise requests it denies.

Extend Override: does not authorise policies that denies requests it accepts.

Deny Override: considers only policies which effect is `Permit`.

When two policies are integrated, the preferences specified by both policies must be satisfied. To achieve such a goal, the two preferences are combined considering the policy similarity computed previously.

Finally, [Rao et al., 2011] relied recently on this work to propose a language-independent policy integration framework. The framework supports all combination and integration algorithms presented above and adds new forms of integration such as policies jumps. To that aim, the authors used Multi-Terminal Binary Decision Diagrams (MTBDD) to represent the policies and generate the integrated policy [Rao et al., 2011]. The objective of this work was not to propose new mechanisms for policy integration. Instead, the authors improved the performance of the combination which was the main flaw of this work.

Discussion

Trust management in virtual communities raises several issues in which the decision made by a member can affect the community as a whole. For instance, decisions by means of which members admit a new member or grant access to a shared resource may be prejudicial to the other members and motivates the need for mechanisms by means of which an individual can make a decision that reflects the community position.

To that aim, we explored in this section two mechanisms, namely policies *combination* and *integration*, by means of which similar issues have been addressed. Combination aims at building a unique decision based on the evaluation of several policies, while integration tries to build a unique policy from several policies and then to derive a decision based on this unique policy.

Using combination means that for each decision made by a member, each member of the community has to evaluate the same request and compute a decision. Then based on these individual decisions the agent receiving the request computes a common decision using a combination algorithm. This approach is clearly not applicable in the context of virtual communities for two reasons. First, policy combination algorithms assume that all parties are available to evaluate any request, and second there always exists an administrator who can solve conflicts. However, in virtual communities this is not the case as members can unpredictably be available in the system and any member will refuse to give up its autonomy to a unique administrator. Thus, conflicts that may appear can never be solved and no decision can be made, penalising the community.

In contrast, integration does not make such assumptions. However, integration is still a recent research field in which most of the contributions are either purely theoretical, or XACML-specific and thus inherit all the flaws that make this language inappropriate for the trust management in virtual communities as discussed in Section 3.4.2. Moreover, works on integration are not realistic as they do not make explicit the use of collective policy of the community as it is done in human societies (cf. Section 4.2.1.1). Thus, each agent is assumed to be aware of the policy used by the other members which poses two problems. First, the integration has proven to fail when too many policies have to be integrated (besides complexity and time consumption), and second, such approach could not be used when the policies used by the members are sensitive.

4.2 The Social Influence Theory

According to Allport [Allport, 1985], most sociologists regard their discipline as “an attempt to understand and explain how the thought, feeling, and behaviour of individuals are influenced by the actual, imagined, or implied presence of others”. This phenomenon is called *social influence*. Social influence can be assessed in all situations involving two entities; two people, two groups, or a person and a group. In our thesis, we have limited our investigation to the influence occurring between a person and the group that represents his community. The influence is a directed relationship, thus systematically one entity plays the role of *source* of influence while the other is the *target*. In the light of this, scholars such as Sherif, Asch, Milgram, Moscovici or more recently Latané identified two forms of influence: influence *from the group to the individual* and influence *from the individual to the group*. This two forms of influence are illustrated in Figure 4.4.

The influence represents a special form of interaction which can be either verbal (i.e. the medium of the interaction are opinions) or behavioural (i.e. the medium of the interaction is a behaviour). In what follows, we review the principal theories that have been proposed to analyse, understand and explain the social influence phenomena.

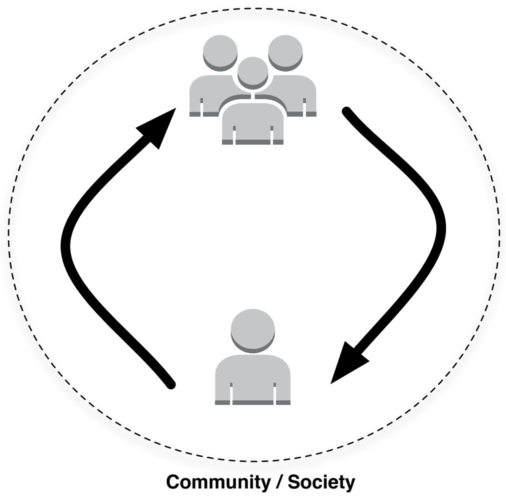


Figure 4.4 – Illustration of the social influence process.

4.2.1 Majority Influence

Majority influence has been defined as the attempt by the majority of people in a group (or a figure representing them) to impose its common position on an individual or a minority of dissenters. In this section, we present theories that conducted the sociologists to establish the existence of the majority influence.

4.2.1.1 The Autokinetic Effect

Muzafer Sherif was the first to investigate the influence that individuals among a group apply to each others in the context of norms emergence [Sherif, 1936, Sherif, 1937]. Sherif recruited a group of individuals that were thinking that their visual perception is being tested. The idea was to ask person sat in a dark room in which a beam of light was displayed to estimate the distance that the light had moved. In fact, Sherif was exploiting the *autokinetic* effect of the eye. The *autokinetic* effect is when very small movements of the eyes make a spot of light in a darkened room appear to move because the eyes lack a stable frame of reference. Sherif repeated the experience several times in which the light never moved. At first, participants were alone and they were asked to report individually their own perception (ranging from 1 to 10 inches). Then, they were grouped together. Each time the beam was displayed the participants were asked to estimate the distance one by one.

As illustrated by Figure 4.5, after few iterations, individuals among the same group tend to converge on a common distance, but each group agrees on a different distance. This early study conducted by Muzafer Sherif evidenced the existence of influence phenomena within small group of individuals.

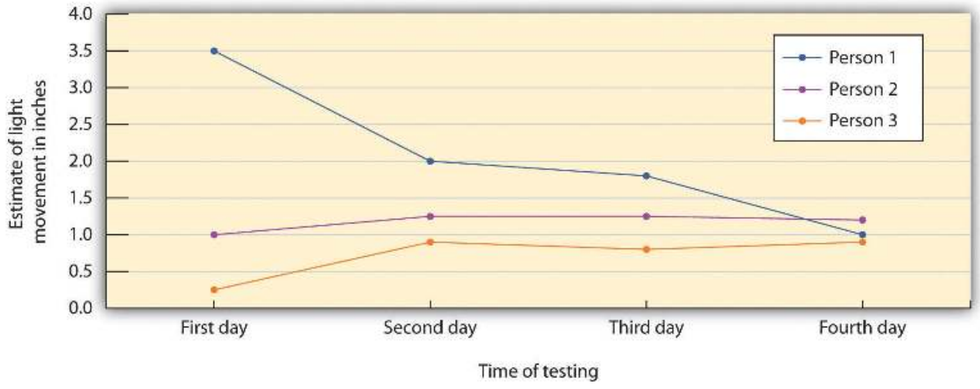


Figure 4.5 – The results of autokinetic effect experience [Sherif, 1936, Sherif, 1937]

4.2.1.2 The Asch Effect

Shlomon Asch [Asch, 1955] is considered by modern sociologists to be the father of the social influence theory. He conducted several experiments in the early 50s to investigate the extent to which social influence from a majority group could affect a person to conform [Asch, 1955]. The participants were 123 male college students (from 17 to 25 years old) who participated in vision test depicted in Figure 4.6. The participants were asked to look at a standard line (the one at the left) and to compare it with three comparison lines (the ones at the right).

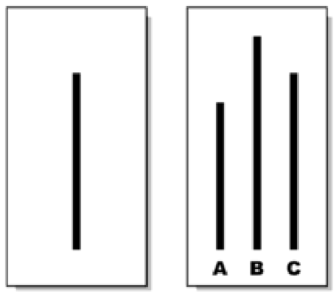


Figure 4.6 – Line-judgement experience in Asch's Study: which line - A, B, or C - has the same length as the standard line? [Asch, 1955]

Asch formed several groups of participants and in every group there was only one individual that was the actual participant. Others were "confederates" who had previously agreed on a wrong answer. Of course, none of the participants were aware of this agreement. The experience was to ask each participant from the group to state aloud which comparison matched the standard line. The participants disposition was such that the real participant would give his opinion after hearing the confederates' answers. Surprisingly, while the decision was an obvious choice, Asch demonstrated in this study that most of the participants (i.e. 79%) did conform to the group opinion in most of the time. This phenomenon is called the *Asch effect*.

Though its relatively simple experiments, Asch evidenced the power of social influence and introduced the concept of conformity in groups.

Definition 8 (Conformity) *Conformity refers to the tendency of individuals to "publicly" conform to the group norms while privately rejecting them.*

Since Asch, many researchers have carried out hundreds of social influence and conformity studies and confirmed that individuals tend to conform to the incorrect opinions of majorities [Jongh, 2013].

4.2.1.3 The Milgram Experiment

Another famous series of social influence experiments were conducted by Milgram in the 1960s. Milgram was particularly motivated to measure the willingness of an individual to conform (or obey) instructions given by another individual representing an authority. Even if this situation involves two people, the sociological construction implies that the authority represents a figure representing the group that placed this individual at this position. The motivation of Milgram's experiment was to study the degree to which one individual is able to reach to follow order even if these orders conflict with his personal motivations. The participants were invited (via newspaper and direct mail) to participate in a study at the Yale University. All participants were male persons (between 20 and 50) with various educational background.

Participants were instructed to inflict painful and potentially fatal electric shocks on another person. This person was a team member but of course the shocks generator was a fake one, but the participants were unaware of that. The generator had switches ranging from "slight shock" to "Danger: severe shock". The participants had to play the role of the teacher and the "confederate" always played the role of learner. The teacher was asked to deliver a shock to the learner each time this latter gives an incorrect answer. While the participants thought that they were delivering real electrical shocks, "confederates" was asked to pretend to be shocked and plead loudly to be released. At the final stage of the experiment, and after several wrong answers and shocks, the "confederate" became silent and refused to answer any question. The examiner (representing the authority in this experiment) instructed the participant to consider silence as a wrong answer and to continue delivering shocks.

The results and conclusions made by Milgram following these experiments were disturbing. None of the participants disobeyed before 300 volts, and a majority (65%) continued to obey to the end of the experiments, and thus administrating the maximum voltage. After the experiments, all participants confirmed that the order was again their personal conscience, but they admit that they were unable to resist to authority. Milgram used the term *obedience* in reference to this kind of social influence.

4.2.2 Minority Influence

As illustrated in previous sections, during the 1950s and 1960s, the study of the social influence was principally focused on the phenomena majority influence, principally through the notions of *conformity* and *obedience*. In majority influence, social researchers wanted to understand the reasons why an individual changes its judgement, behaviour or attitudes in the direction of the position held by the majority. However, in parallel to the majority influence, the minority is able to influence too. Relying on the concept of *social conflict*, minorities have shown that it is possible to bring change in the majority opinions, attitudes and behaviours.

4.2.2.1 Moscovici Experiment

Since the end of the 1960s, Moscovici started to investigate the reciprocal process of majority influence [Moscovici, 1969]. His objective was to understand the social mechanisms by means of which an individual (or a minority of individuals) exerts an influence that make the group change its position in favour of the minority [Bowser, 2013]. Moscovici and colleagues researched minority influence in a conformity situation in which 192 participants that were randomly affected to either a *consistent*, *inconsistent* or *control* group. Each group was composed of 4 participants and 2 confederates representing the minority (except the control group). The experience consists in asking the group members to state the colour of 36 slides, all were blue but with varied brightness.

In the consistent group, the 2 confederates described all slides as green. In the inconstant group, the confederates stated that 12 slides were blue and described the others as green. In the control group, there were no confederates.

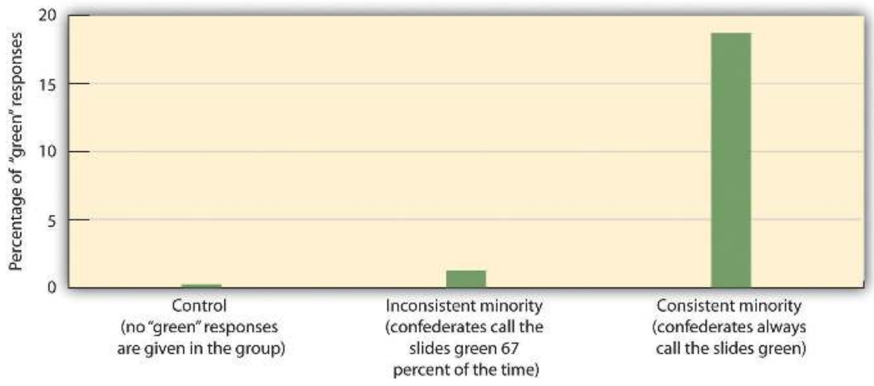


Figure 4.7 – The results of Moscovici experience [Moscovici, 1969].

The results of the study (cf. Figure 4.7) showed that a non neglectful portion of the group (32%) gets converted to the incorrect statement made by the minority (i.e. the confederate person). Thus, in these experiments, Moscovici evidenced the fundamental role of the minority consistence in bringing changes to the majority position. For the author, being consistent is more likely to influence the majority than if the minority often changes its position.

Definition 9 (Conversion) *Conversion is the ability of a minority to influence and convince a majority that the point of view of the minority is correct.*

The results of Moscovici were later confirmed by many other researchers proving that a minority could, surprisingly, influence the decision made by a majority without relying on power or authority.

4.2.2.2 Nemth Experience

Nemeth [Nemeth, 1986] explained that when majorities are faced with a consistent attitude (the one of the minority) they are confused and try to understand why the minority is so determined to express controversial opinions [Nemeth, 1986]. The majority influence is built upon this questioning that disconcerts the majority and leads in some situation to changes.

Later, Nemeth questioned whether consistency alone was sufficient for a minority to exert an influence over a majority. To explore this phenomenon, Nemeth used a mock jury in which groups of three participants and a confederate had to decide the amount of compensation to be accorded to a victim of an accident. Nemeth evidenced the importance of *flexibility* and *compromises* to make the majority admit the minority opinion.

In fact, Nemeth provided valuable results about how the majority interprets the minority attitude. The consistency argued by Moscovici is perceived as an inflexible, dogmatic and rigid attitude, so the majority will unlikely change its view. In contrast, if the minority position is perceived to be cooperative, reasonable and less extreme the influence of the minority over the majority will have better chances to succeed.

4.2.3 The Social Impact Theory

In the previous sections, we presented several works that demonstrate the extent to which people (majorities or minorities) can be influenced by others. In this section, we present works that try to identify situational factors that makes people *conform* or not. Based on the findings and results of previous works, Latané [Latané, 1981] formalised the concept of *social impact* in which he tried to characterise the social situation that determines the extent to which an individual is willing to confirm or to convert. To that aim, Latané assimilated the social influence (called impact) on an individual to the way light-bulbs illuminate a surface. Having this analogy in mind, Latané stated that the social influence felt by an individual is function of the strength, immediacy and number of the source and the target of the influence [Rehm and Endrass, 2009].

$$SocialImpact = \frac{(Strength \times Immediacy \times Number)_{source}}{(Strength \times Immediacy \times Number)_{target}} \tag{4.3}$$

where:

- **Strength** states how important and valuable the influencing individuals are to the influenced person,
- **Immediacy** represents the spatial or social closeness of the influencing individuals,
- **Number** expresses how many persons are exerting the influence on the influence person.

Discussion

Whether they represent a majority or a minority, individuals tend to change their opinions, attitudes and behaviour under the pressure of the social structure to which they belong (groups, communities or societies). What the reader should keep in mind after reading this is that (1) individuals tend to rapidly converge towards a common opinion which represent the position of the group with respect to a particular issue (the autokinetic effect investigated by Sherif in Section 4.2.1.1), (2) when faced to the position of the group, people tend to conform to it even if they still disagree privately. This kind of conformity is called *normative conformity* and has been evidenced by Asch (cf. Section 4.2.1.2). Further, people tend to conform if their position is conflicting with the one of the group. If there is no conflict, people always prefer to keep their own position. Finally, (3) when the group does not meet the *strength*, *immediacy* and *number* conditions or when individuals exhibit *consistence* and/or *flexibility* they can lead the group to change its position in response to their pressure (cf. Section 4.2.2). This last phenomenon represents an *informational conformity* in which a group is influenced by a minority and gets converted to their position.

4.3 Multi-Agent Systems

Multi-Agent Systems (MAS) is a well-established software development paradigm in distributed artificial intelligence. MAS research has taken its roots from the problems encountered in the implementation of robust and scalable software systems. The implementation of such systems led researchers to propose solutions that are composed of distributed computation units that engage in flexible, high-level interaction with one another and with their environment as well [Conte et al., 1998]. These computation units are called agents (initially intelligent agents) and are defined as follows [Jennings and Wooldridge, 1998].

Definition 10 (Agent) *An agent is a computer system situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objective [Jennings and Wooldridge, 1998].*

In the above definition, *autonomy* refers to the capability of an agent to perform actions in order to achieve its objective without the direct intervention of humans [Wooldridge and Jennings, 1995]. However, what the above definition does not say is how an agent can achieve its objectives. Wooldridge and Jennings [Jennings and Wooldridge, 1998] identified three characteristics that agents require to satisfy their design goals, namely *reactivity*, *proactivity* and *sociability*. Reactivity means that the agent should perceive its environment and react to changes that occur in it. This is important for the agent to achieve simple goals (e.g. report temperature). Proactivity refers to the ability of an agent to reason about its environment's changes and exhibit goal-directed behaviour and/or take initiatives. This feature is required when goals are complex and the agent needs to adapt to its environment. Finally, sociability means that the agent is able to interact with other agents (or humans). This last feature is paramount as agents have limited capabilities that prevent them from achieving ambitious complex objective. To that aim, they need to interact/collaborate with others in order to achieve their objectives [Jennings and Wooldridge, 1998].

Agents are often involved in collective structures in which they engage together to achieve a common objective. These collective structures are called *multi-agent systems* (MAS).

Definition 11 (Multi-Agent System) *A multi-agent system is a collection of multiple, eventually independent agents working together in a common environment to solve problems that are beyond the capabilities or knowledge of individual agents [Jennings and Wooldridge, 1998, Dignum, 2004].*

MAS are often used to implement open systems in which a mix between autonomy and cooperation is required (e.g. coalitions, communities, virtual organisations). However, in this kind of systems, agents are developed by different persons, and thus cannot be assumed to behave as expected (inadvertently or deliberately). Castelfranchi [Castelfranchi, 2000] defined this issue as the *social order* problem [Castelfranchi, 2000]. The author described the social order as "any form of systemic phenomenon or structure which is sufficiently stable". It is the existence of such a problem, and its analogy with the issues presented in this chapter introduction, that motivated our choice in addressing the trust management problem from a multi-agent perspective.

The *social order* problem stimulated MAS researchers for more then ten years during which they proposed several *control mechanisms* that address this problem from two different perspectives: *top-down* and *bottom-up*. Top-down approaches are said to be *system-centred*. The mechanisms by means of which agents are controlled are made explicit and are *orchestrated* by the system (e.g. authorities or particular agents). In contrast, bottom-up approaches are *agent-centred* and rely on *implicit control* that is performed by the agents themselves [Castelfranchi, 2000, Grizard et al., 2007, Rakotonirainy et al., 2009, Andrighetto et al., 2010, Villatoro, 2013, Bowser, 2013]

4.3.1 Top-Down Perspective: Orchestrated Control

In this approach, the problem of controlling the behaviour of the members of a group is solved in a prescriptive and explicit way. Depending on the nature of these prescriptions and the way they are made explicit to the agents, this approach is split into two sub-approaches: by assigning *organisations* to systems, or by imposing *norms* to agents.

4.3.1.1 Organisational Approach

The concept of organisation is not new. It has been introduced from the begging of MAS (cf. Corkill's doctoral dissertation [Corkill, 1983]). However, organisations as first class abstraction for programming MAS were made famous in the team-oriented programming (TOP) model [Tambe, 1997, Tambe, 1998, Pynadath et al., 2000].

Definition 12 (Organisation) *Purposive supra-agent pattern of emergent or (pre)defined agents cooperation, that could be defined by the designer or by the agents themselves [Boissier et al., 2010].*

Concretely, multi-agent organisations are defined in terms of roles, relationships and authirity structures that govern agents behaviour. They are used by the system designer to impose an expecteed behaviour to the agent that compose the system [Horling and Lesser, 2004]. The expected behaviour is represented as a set of constraints to which each agent is subject once he joins the system. These constraints are generally associated to roles that agents commit to in the context of a particular objective [López y López and Luck, 2004]. Thus, roles represent the cornerstone of an organisation by means of which the organisations prescribes the behaviour of agents playing this role [Tinnemeier et al., 2009].

In order to implement such systems, the specification of the organisation has to be made explicit so that the agents that belongs to this system can reason on them [Hubner et al., 2007]. To the best of our knowledge, there exist four predominant approaches for implementing organisations in multi-agent systems: ISLANDER [Esteva et al., 2002], STEAM [Tambe and Zhang, 2000], AGRE [Ferber et al., 2005], $\mathcal{M}OISE^+$ [Hubner et al., 2002] and OPERA [Dignum, 2004].

The specification of an organisation is usually defined with respect to two distinct dimensions: *functional* and *structural* dimensions. The functioning of an organisation defines the global objective of the systems and how this objective can be further defined into sub-objectives (e.g. STEAM and $\mathcal{M}OISE^+$); while the structure of an organisation determines the roles, the groups and how groups and roles are organised (e.g. STEAM, AGR and $\mathcal{M}OISE^+$).

Among these models and systems, the $\mathcal{M}OISE^+$ model is a good candidate since it combines functional and structural dimensions while keeping them independent and extensible. The glue which is used to bind both dimensions is a normative dimension that makes it possible to express obligations and permissions with respect to the achievement of collective goals while playing roles participating to the structural dimension. *Norms* and *normative multi-agent systems* are presented in the next section.

4.3.1.2 Normative Approach

In the previous section, we presented a *top-down* approach for controlling agents with organisation concepts. However, in many situations the orchestration of organisations can be complex to be achieved, especially in presence of a large population and a high agent turnover

[Haynes et al., 2013]. In that context, *norms* represent a flexible way by means of which agents behaviour is constrained.

Normative multi-agent systems (NMAS) combine theories, mechanisms and frameworks from normative systems (NS) with the ones from multi-agent systems (MAS). Boella and colleagues defined a *normative multi-agent system* as a "MAS composed of mechanisms to represent, communicate, distribute, detect, create, modify, and enforce norms, and mechanisms to deliberate about norms and detect norm violation and fulfilment" [Boella et al., 2008]. Meyer and Wieringa [Meyer and Wieringa, 1993] defined normative multi-agent systems as "systems in the behaviour of which norms play a role and which need normative concepts in order to be described or specified" [Meyer and Wieringa, 1993, Meyer et al., 1998].

Definition 13 (Norm) *A norm is a formal specification of deontic statements that aims at controlling the behaviour of software agents and interactions among them [Villatoro, 2013, Tinnemeier et al., 2009, Governatori and Rotolo, 2009].*

Norms are used to specify constraints on the behaviour of agents. In our thesis, we distinguish between *legal norms* and *social norms*. Legal norms are created, maintained and enforced by entities that are outside the system (e.g. system designer or an authority). In contrast, social norms originate from inside the system and are enforced by the agents themselves. This section addresses legal norms, while social norms will be presented in Section 4.3.2.

Both legal and social norms are stated in terms of *obligations*, *permissions* and *prohibitions*. In what follows, we present how legal norms are implemented in multi-agent systems by *regimentation* or by *regulation* [Campenni et al., 2009].

- In *regimentation*, norms are hard coded in the agents, thus each agent is by default norm compliant. Regimentation can be achieved in two ways: *mediation* and *hard-wiring*. In mediation, agents' interactions are mediated by a reliable entity that controls their current behaviour and prevents deviation from the expect one [Criado, 2013].

- In *regulation* makes the norms intelligible to agents and states what these agents are obliged, permitted or prohibited to do. *Regulation* is achieved by a third party (e.g. system designer, regulation authority, particular agent) that observes the agents' behaviour and sanctions those that do not comply with the norm.

In the light of this, and compared to regimentation, regulation represents a *flexible*, *adaptive* and *robust* control mechanisms as agents can deliberately violate these norms when faced to unexpected circumstances [Haynes et al., 2013]. Finally, the norms of an organisation define the obligations, permissions and prohibitions that an agent is subject to within the organisation (e.g. $\mathcal{MOISE}^+$ [Hubner et al., 2002] and OPERA [Dignum, 2004]). For instance, drivers in France drive on the right side of the road, while the norm in United Kingdom is to drive on the left side of the road.

4.3.2 Bottom-Up Perspective: Emergent Control

In the previous section, we presented two *top-down* mechanisms (i.e. organisations and legal norms) by means of which a group of agents copes with the autonomy of its members. Both mechanisms rely on a third party entity that is responsible for enforcing these mechanisms and sanctioning the deviant agents. However, in many situations having such an entity is not feasible (e.g. self-organising multi-agent systems).

In this section, we present the *bottom-up approach*, called *emergent control*, which has been proposed as an alternative solution for the social order problem. In this approach, all agents in the system are collectively involved in creating, modifying and enforcing the constraints to which they are subject [Burgemeestre et al., 2010]. These constraints represent social norms that result from the agreement of agents about what represents an appropriate behaviour [Conte et al., 2009].

Definition 14 (Social Norm) *A social norm is a generally accepted way of behaving that most agents in a group agree on and endorse as right and proper [Smith and Makckie, 2000].*

It is relatively simple to understand how legal norms are created, updated and enforced (cf. Section 4.3.1.2). However, is it much less obvious how the same process applies to social norms [Campenni et al., 2009]. Therefore, most of works on social norms draw inspiration from works on sociology and social-psychology. As presented in Section 4.2, sociology provided several theories that directly address the concept of social norm. For instance, the *autokinetic effect* demonstrated by Sherif in Section 4.2.1.1 studies the mechanisms by means of which norms emerge.

Consequently, research on social norms has been particularly interested in interpreting and reusing these models and theories to implement *emergence of social norms* and their *enforcement*. *Norms emergence* studies the mechanisms used by agents to create and modify social norms in the system, while *social control* focuses on how social norms are enforced by agents [Criado, 2013].

4.3.2.1 Norms Emergence

In the context of social norms, the main objective of the MAS research community is to understand: (a) how norms appear in the mind of individuals, and (b) how these norms are disseminated within the society until their adoption. Social norm creation is explained as a manifestation of *social influence* which has two parts: (1) norms are developed through pressures towards uniformity (cf. Section 4.2.1.1), and (2) they are propagated within the system using *conformity* (cf. Section 4.2.1.2). As these mechanisms have already been presented in previous sections, we focus the remainder of this section in the techniques used to implement them.

Social norms emergence is generally implemented in multi-agent systems using the agents' *self-organising* capabilities. Farley and Clark defined a self-organising system as "a system which changes its basic structure as a function of its experience and environment"

[Farley and Clark, 1954]. In this definition, the authors highlighted the two main features of a self-organising system that are *adaptation* and *learning* [Ashby, 1947, Banzhaf, 2009]. In self-organising systems, local interactions between individuals give rise to *observable* global features such as *robustness*, *scalability* or *social order*. This phenomenon is commonly called *emergence* [Alechina et al., 2013]. Conte and colleagues defined emergence as a macro-level effect of interactions among agents, which are carried out at the micro-level.

The principal idea of self-organisation is to endow local agents with mechanisms that, when these agents are in interaction, leads to the emergence of some desirable properties in the existing system. Some of these properties have been identified and grouped as follows:

Self-configuration: refers to the ability of the system to reconfigure itself in order to achieve its goals.

Self-optimisation: is used in reference to the ability of the system to adapt its behaviour to improve its utility in achieving its goals.

Self-healing: refers to the ability of the system to overcome the states that prevent it from achieving it goal. Concretely, it tries to bring the system back to the normal working conditions.

Self-protection: aims at allowing the system to detect attacks that prevent it from achieving the goal, and if possible detect threats and avoid them.

All these properties are generally called self-* properties as what they have in common is that they are achieved autonomously and collectively by the agents that compose the system.

To that aim, the agent must be able to retrieve its internal variables and the ones of the entities that compose its environment (*monitor*). It should also be able to interpret and process such data (*analyse*), deliberate and take decision on how to react given this data (*plan*), and finally trigger actions in order to apply the decision he took (*execute*). The overall process is executed in a loop, called the MAPE-K loop (Monitor, Analyse, Plan, Execute, Knowledge), that is supported by an internal *knowledge base* in which the agent stores the data it processed during the cycle [Dubois, 2011, Koehler and Giblin, 2003].

4.3.2.2 Social Control

In self-organising systems, and in absence of a trusted authority, the agents themselves are responsible for enforcing the social norms. The enforcement of social norms by the agent population is called *social control*. Social control provides multi-agent societies with a decentralised control mechanism. It is generally achieved with three punishment mechanisms: *sanctions*, *reputation* and *exclusion*.

Sanctions

Sanctions and *punishment* are well known mechanisms to exert a social control on wrongdoers. In utilitarian systems, a sanction results in a loss of utility. Several works in multi-agent systems have considered the use of sanction mechanisms to ensure social order. Pasquier and colleagues provided in [Pasquier et al., 2006] an ontology of social control tools in which they classified different types of sanctions that can be applied in MAS. The authors distinguish *negative* sanctions from *positive* sanctions. Negative sanctions are used to discourage norms violators, while positive sanctions represent rewards that encourage compliant behaviours.

Allowing peer sanctioning ensures a decentralised norms enforcement. However, in absence of a central authority, sanctioning non compliant implies bilateral cost; the sanction has a cost for both the sanctioning and the sanctioned agents. Consequently, in a such situation, agents tend to refrain from sanctioning as the sanction implies a loss of utility [Burgemeestre et al., 2010, López y López and Luck, 2004, Boella and van der Torre, 2005, Bocchiaro and Zamperini, 2012]. To that aim, recent works tried to propose mechanisms in which sanctions are applied collectively [Villatoro, 2013].

Reputation

In non-utilitarian systems, sanction mechanisms we presented in the previous section are not efficient or are impossible to apply. For instance, in social networks, the concept of utility is hard to implement. In such context, social sanctions are used to alter the image of the individual among the society.

Reputation is the most used mechanism to sanction non compliant agents [Grizard et al., 2007]. As presented in Chapter 2, reputation reflects the image of an individual as seen by the member of the society. So when an agent does not comply with a norm, its reputation is negatively updated by all the agents that observed its deviance [Eisenegger, 2009]. The new reputation value is propagated and thus all agents (even those that did not observe the non-compliant behaviour) update their value. Nevertheless, reputation has two main limits: how the system is initiated (cold start, bootstrap) and how to deal with anonymity.

Exclusion

Sanctions and reputation are generally used as a warning to discourage non compliant agents from violating the norms that govern the society. However, in many situations agents are not sensitive to such warnings and pursue violating norms. Such behaviour does not only threaten the system's stability, but may also influence other agents to deviate if no effective action is taken. For that, *exclusion* seems to be the ultimate solution for controlling non-compliant agent.

Exclusion is also considered as a consequence of bad reputation. Agents tend to refuse to interact with agents having a bad reputation. This situation is assimilated to an implicit exclusion even if the agent still belongs to the society. Nevertheless, in a fully decentralised system combined with unreliable authentication mechanisms, ensuring exclusion remains an open issue for research in MAS.

4.3.3 Mixed Perspective: Micro-Macro Loop

In previous sections, we presented two approaches by means of which a group of agents deals with the autonomy of its members to ensure social order. In abstract terms, the difference between these approaches are best described by the question "social order is the result of an external entity imposing control on the behaviour of agents (via organisations or via norms), *or* reflects patterns which appears from the interaction of these agents (via norm emergence and self-organising mechanisms)".

However, in what Keith Sawyer describes as "a fundamental intellectual problem" [Sawyer, 2003], both approaches fail to adequately address the real interplay between *top-down* and *bottom-up*. Also known as the *micro-macro* phenomena, some recent approaches rely on this phenomena to seamlessly combine the previous approaches. Among these initiative, we are particularly interesting in works about *reorganisation* and those that address norms adaptation [Alechina et al., 2013].

4.3.3.1 Reorganisation

Reorganisation is defined as process by means of which an organisation is changed to a new one. As presented in Section 4.3.1.1, an organisation can be described via two dimensions, namely structural and functional. In the light of this, a reorganisation can affect both dimensions. For instance, an organisation can change its structure by changing the groups and the roles that compose it. An organisation can also be reorganised by changing the missions that agents must achieves.

In [Hübner et al., 2004], the authors identified three types of reorganisation; *predefined*, *controlled* and *emergent*. Predefined reorganisation is a planned reorganisation that is automatically triggered when the conditions are met. For instance, an organisation can be specified to change its structure after a certain time of execution. Controlled reorganisation will be triggered by an entity (an agent of the system or a system designer). Emergent reorganisation is performed by one or many agents of the system based on their individual beliefs and objectives. These two last forms of reorganisation reflect the micro-macro loop we are interested in. In [Alberola et al., 2011], the authors presented and compared existing approaches of reorganisation in multi-agent systems.

4.3.3.2 Norms Adaptation

Norms adaptation tries to combine the mechanisms used in legal norms with those used to build, update and maintain social norms. This approach is currently considered as one of the most challenging properties of multi-agent systems that operate in dynamic, unpredictable and eventually hostile environments.

Like reorganisation, norms adaptation can be achieved in three different ways: *predefined*, *controlled* and *emergent*. Here again, we are interested in controlled and emergent norms adaptation that can be achieved via a micro-macro loop.

With controlled adaptation, a system designer can allow its agents to dynamically revise the norms to which they are subject in response to changes in the environment. In [Lacey and Hexmoor, 2003], the authors presented a model in which agents are endowed with capabilities to adapt norms to which they are subject when some conditions in the environment hold.

In contrast, in emergent norms adaptation, agents are the masters of their own fate. Each agent can reason on the norms governing its behaviour and can revise it if necessary. In absence of a central authority, the updated norm is propagated and disseminated using the mechanisms described in the social influence theory 4.2.

4.4 Conclusion

In this chapter, we reviewed existing literatures with respect to the social dimension of trust management. Our objective was to analyse how research to date was dealing with situations in which the decision made a member of a social structure (e.g. group, community, coalition,etc.) affects all members of the structure. We presented in sections 7.5.1 and 7.5.2 two predominant approaches for dealing with this issue and discuss their applicability in the context of virtual communities.

In Section 4.2, we explored the *social science* discipline looking for explanations about how *humans* take decisions within social structures. With respect to that, works from the *social influence theory* answered most of our questions. The theories that have been proposed confirmed the limits of existing trust management systems with respect to the lack of consideration of the social dimension in the trust decision making.

Finally, in Section 4.3 we presented basic concept of *multi-agent systems* and discussed the adequacy of their techniques with the *social influence theory*. In the light of that, and with respect to our objectives (cf. Section 1.3), and given the agents properties we presented in 4.3, agents technology seems to be the perfect abstraction for the implementation of *adaptive and socially-compliant* trust management systems.

Previously, we discussed how norms and organisations have been used to *explicitly* regulate multi-agent systems by making agent comply with the behaviour expected by the system designer. However, the use of such approach often requires the introduction of special entities that are responsible of issuing, maintaining and in most of the case enforcing the organisation specification and norms (sanctions, rewards or exclusions). However, in complex distributed system, identifying such a unique entity is neither possible nor desirable. Impossible because it would be very hard to reach an agreement about the entity responsible of issuing norms, and not desirable because such systems would not meet the requirements of scalability due to the risk of bottlenecks and the presence of a unique point of failure and attack. Moreover, systems in which the control reflects a unique perspective (the one of the system designer her) would not be more powerful than that unique perspective. In other words, the control mechanism will never be more efficient than that unique perspective [Dubois, 2011]. However, this *system-level control* is the unique way to guarantee that unknown agents will behave as expected as it imposed constraints on their behaviour.

4.5 French Summary

Dans le chapitre précédent, nous avons pu constater que l'objectif des systèmes de gestion de la confiance existants est de faire en sorte que les décisions prises par leurs utilisateurs soient sûres à titre individuel. Dans ce chapitre, nous allons nous intéresser aux mécanismes qui permettraient à ces systèmes d'être sûre à titre collective également.

En effet, à partir du principe que les individus ont tendance à aller plus loin lorsqu'ils conjuguent leurs efforts, le concept de communauté est très à la mode de nos jours afin de stimuler la collaboration dans les communautés réelles et virtuelles. Or, dans ce type de situation, les individus rejoignent les communautés avec des expériences, compétences et politiques diverses. Tandis que l'hétérogénéité en matière d'expérience et de compétence est considérée comme étant la richesse et la force de ces communautés, hétérogénéité dans leurs politiques peut constituer un frein à la collaboration si celles-ci sont en conflit et qu'aucun mécanisme n'a été prévu pour résoudre ces conflits.

Dans ce chapitre, nous explorons la littérature à la recherche de mécanismes qui seraient susceptibles d'apporter une solution à ce problème. Nous somme particulièrement focalisés à apporter une solution à ces trois questions : (1) comment les membres de communauté virtuelles prennent des décimions quand les conséquences de celles-ci peuvent affecter les autres membres, (2) est que le fait d'être dans une communauté affect la manière dont l'individu prend ses décisions et comment, et (3) est ce qu'un individu peut affecter la manière dont les autres membres de la communauté prennent leurs décisions et comment cela est-il possible.

Dans la suite de ce chapitre, nous complétons d'abord notre revue de la littérature en matière de gestion de la confiance afin d'analyser comment les chercheurs de la discipline ont essayé de répondre à la problématique soulevée plus haut. Ensuite, nous analyserons quelques théories et modèles qui ont été proposés en *sociologie* et implémentés dans les *systèmes multi-agent* afin de voir si ces disciplines ont sur répondre à cette problématique [Coleman, 1990, Yao, 2004].

4.5.1 La dimension sociale dans la gestion de la confiance

Dans la gestion de la confiance et le contrôle d'accès, les politiques sont utilisés pour prendre des décisions qui proviennent et expriment les exigences d'une seule entité (le propriétaire). Cependant, lorsqu'il s'agit de structures collectives comme les communautés virtuelles, cette démarche ne s'applique plus. Les conséquences de la décision d'un individu sont supportées et affectent tous les membres de la communauté. Cela implique que la prise en compte de la politique de chaque individu lors de la décision est un élément-clé de la cohésion de la communauté. Dans cette perspective, nous avons recensé deux approches (*combinaison et intégration*) qui ont été proposé dans le domaine de la gestion de la confiance pour faire face à cette problématique.

4.5.1.1 Combinaison

La combinaison a été proposé dans le cadre de l'infrastructure XACML. Dans ce langage, il existe trois niveaux de granularité pour définir des politiques : *des conditions*, *des politiques* et des *ensembles de politiques*. Ainsi, le résultat de l'évaluation d'une politique est obtenu en combinant les résultats des l'ensemble des conditions (ou des politiques) composant la politique (ou l'ensemble des politiques). La contribution consiste alors à proposer différentes heuristiques afin de proposer à l'utilisateur des approches de combinaisons qui couvent tous ces besoins (e.g., si une des décisions acceptes, la décision finale est d'accepter) [Li et al., 2008, Rao et al., 2009].

La limite de cette approche réside dans le fait que la combinaison nécessite que l'ensemble des individus impliqué dans la décision soient présents et fournissent leurs évaluations respectives pour que la décision puisse être prise. Or, dans un système ouvert et distribuée, une telle condition serait difficile à satisfaire. De plus, cette approche implique une complexité si on prendre en compte la négociation et mobilise l'ensemble des membres ce qui peut poser des problèmes de rendement et de passage à l'échelle.

4.5.1.2 Intégration et Composition

L'intégration et la composition font référence à la même approche: construire une nouvelle politique à partir de politiques qui ont été spécifié par plusieurs individus. Ainsi, dans l'intégration (nous utiliserons ce terme dorénavant), la décision qui peut concerner tout le monde est prise en se basant sur une politique intégrée qui n'est que le produit de l'intégration des politiques individuelle de chaque membre. Une fois cette politique intégrée obtenue, son évaluation permet d'obtenir une décision qui reflète l'avis de tous ou du moins la majorité.

Bonatti et ses collègues [Bonatti and Samarati, 2002] furent les premiers à explorer cette piste. Ils ont proposé une algèbre de composition de politiques qui permet d'additionner, de conjuguer et de soustraire des politiques entre elles. L'addition produit l'union des politiques, la conjugaison permet d'obtenir leur intersection et la soustraction permet de soustraire d'une politique les effet non existants dans l'autre. Par exemple, l'addition de deux politiques aura pour effet de créer une politique qui va autoriser toute ce que les deux politiques autorisaient. Ainsi, c'est un addition des autorisations véhiculés par les les politiques qui est effectuée.

Plus récemment, Mazzoleni et ses collèges [Mazzoleni et al., 2006] se sont inspiré des travaux de Bonatti afin de proposer un mécanisme similaire pour l'intégration de politiques XACML. Dans leurs travaux, l'intégration ne prend plus la forme d'opérateur dans une algèbre de politique mais d'algorithmes qu'ils appellent heuristiques de combinaison [Rao et al., 2011].

4.5.2 La théorie de l'influence sociale

D'après Allport [Rao et al., 2011], "La psychologie sociale tend à comprendre et à expliquer comment les pensées, les sentiments et les comportements moteurs des êtres humains sont influencés par un autrui, réel, imaginaire ou implicite". Ainsi, la psychologie sociale s'intéresse aux interactions de l'individu afin de comprendre comment est-ce qu'il agit ou réagit à l'égard d'autrui. Ce phénomène est appelé *influence sociale*. Dans notre thèse, nous nous somme concentré sur l'étude de l'influence qui s'exerce entre un individu et le groupe auquel il appartient. Dans ce type d'influences, les travaux de chercheurs comme Sherif, Asch, Milgram, Moscovici ou plus récement Latané, on permit d'identifier deux formes d'influence : *l'influence du groupe sur l'individu* et *l'influence de l'individu sur le groupe*. La première est appelée *l'influence majoritaire* et la seconde *l'influence de la minorité*. Dans ce qui suit, nous décrivons brièvement chaque type d'influence en précisant les modèles et théorie qui y ont été proposés.

4.5.2.1 L'influence majoritaire

L'influence majoritaire a été définie comme étant la tentative de la majorité des membres d'un groupe (ou une figure les représentants) d'imposer son point de vue, son attitude ou son avis sur un individu ou sur une minorité dissidente. Dans cette section, nous étudions les principaux travaux qui ont permis d'identifier puis d'établir l'existence de cette influence.

La première expérience dans ce sens fut menée par Muzafer Sherif au travers de son effet autocinétique. Dans ces expériences [Sherif, 1936, Sherif, 1937], Sherif observa que les individus avaient tendance à construire des normes communes quand ils sont au sein de groupes. En réalité, chaque individu était apte à fléchir sa position pour s'approche de cette adoptée par les autres créant de ce fait un capital commun différent de la position initiale de chaque individu.

L'expérience de Asch quelques années après à permise de mettre en évidence le phénomène de conformité (ou l'effet Asch). En effet, 80% des individus évalué avait tendance à changer leur position pour adopter celle prise par un groupe.

Enfin, dans l'expérience de Milgram, le chercheur a mis en évidence que la conformité pouvait s'opérer également en présence d'un unique individu mais uniquement quand celui-ci constituait une figure qui représentait la majorité (e.g., autorité). Là aussi, bien que contraires à leurs principes, les individu ont montré une nette tendance à changer de position pour se conformer (obéir) à l'avis de la majorité représenté ici par l'autorité.

4.5.2.2 L'influence minoritaire

Dans l'influence majoritaire, les chercheurs ont essayé de comprendre pourquoi un individu était amené à changer sa position pour se conformer (ou obéir) à une majorité. Or, tout le monde sait que tout comme la majorité, une minorité est également capable d'influencer la majorité. Nous expérimentons tous les jours ce phénomène à travers les concepts de modèle, de star ou de figure publique influente. Ainsi, à partir des années 1960, le français Serge Moscovie a commencé à étudier le processus réciproque de l'influence majoritaire. Ses expériences [Moscovici, 1969] ont permis de démontrer que quand une minorité est constante, elle peut amener une partie de la majorité à se convertir en adoptant leur position. Poussé à l'extrême, ce processus permettrait à long terme de faire en sorte qu'une minorité devienne la majorité.

Quelques années plus tard, Nemeth [Nemeth, 1986] poursuivit ces expériences afin d'expliquer davantage ce phénomène. Nemeth expliquait que face à une minorité conflictuelle et consistante, la majorité était poussée à se questionner à la position de cette minorité, et c'est justement ce questionnement qui pousse certains d'entre eux à se convertir.

Nous avons utilisé ces résultats dans notre modèle (cf. Chapitre 7)afin de permettre à des membres de communauté virtuelle d'adapter leurs politiques collectives.

4.5.3 Les systèmes multi-agent

Les systèmes multiagents (SMA) sont un paradigme de programmation dont les origines se trouvent dans le domaine de l'intelligence artificielle. La recherche en SMA se nourrit des problèmes rencontrés lors de l'implémentation des systèmes robustes et capables de passer à l'échelle. Ainsi, les chercheurs ont proposé l'utilisation d'entité computationnelle distribuée qui seraient capables d'entrer en collaboration avec d'autres unités ainsi que son environnement [Jennings and Wooldridge, 1998]. Ces entités sont appelées agents et sont souvent caractérisées à travers leur autonomie, réactivité et proactivité. Les agents sont également caractérisés par leur dimension sociale car ils évoluent souvent au sein de structures collectives au sein desquelles ils collaborent afin de réaliser un objectif commun. Sur cet aspect, les SMA se rapprochent beaucoup du concept de communauté virtuelle ce qui a motivé davantage notre objectif d'en analyser le mode de fonctionnement.

Les SMAs ont été souvent utilisé pour implémenter des systèmes ouverts et décentralisé dans lesquels un juste milieu entre autonomie et coopération est requis (e.g., coalitions, organisations virtuelles et plus récemment communautés virtuelles). Or, ces systèmes sont souvent développés par différentes personnes et dont ne peuvent être supposés fonctionner comme prévu (par inadvertance ou délibérément). Ce problème a été défini par Castelfranchi par *l'ordre social* [Castelfranchi, 2000].

Le problème d'ordre social a enthousiasmé la recherche dans les SMAs durant la dernière décennie. Durant cette période, plusieurs travaux ont été réalisés dans la perspective de mettre en place des mécanismes de contrôle afin de résoudre ce problème. On distingue deux types de mécanismes : *le contrôle descendant* et *le contrôle ascendant*. Dans la première, le problème de contrôler le comportement des agents se traduit par des approches explicites qui imposent le comportement escompté aux agents du système. Nous citons à titre d'exemples les approches par organisation ainsi que celles par normes. Or, ces deux approches requièrent l'introduction d'une entité spéciale qui a la charge de spécifier, maintenir et dans le plus souvent appliquer ces mécanismes. En réponse, dans la seconde approche, le contrôle repose sur l'implication de chaque agent dans le contrôle des autres et ne nécessite donc aucune entité tierce. Dans cette catégorie, nous avons présenté les travaux sur l'émergence de normes ainsi que ceux liés au concept d'ordre social [Dubois, 2011, Koehler and Giblin, 2003].

Enfin, dernière approche en cours d'expérimentation vit à équilibrer les deux approches en proposant un bouclage individu-collectif dans lesquelles. Nous évoquons à titre d'exemple les travaux sur la réorganisation [Hübner et al., 2004] ainsi que ceux sur l'adaptation de normes [Alechina et al., 2013].

Part III

The ASC-TMS Model

Chapter 5

A Multi-Agent-Based Virtual Community

The adoption of agent and multi-agent technologies is a trending approach for modelling and implementing collaborative virtual communities (VC) as well as social networks (SN) in general [Camarinha-Matos et al., 2003, Rupert et al., 2007, Gupta and Kim, 2004]. As illustrated in Chapter 4, multi-agent systems as a virtual workspace where agents are interacting, competing, cooperating and negotiating for the fulfilment of their (individual and collective) goals are a good abstraction model for supporting virtual communities.

In this chapter, we will describe from a theoretical point of view, fundamental concepts that we will rely upon in Chapters 6 and 7. First, we will introduce in Section 5.1 a general view of the multi-agent-based framework that underpins the contribution of this thesis. Then we will delve into more details to present each of the elements that comprise this framework. We start our description in Section 5.3 with the *environment* in which the *agents* evolve, then we present, respectively, in Section 5.4 and Section 5.5, the agents that represent human users and the *communities* to which they belong. Finally, Section 5.6 is dedicated to the *interactions* that are taking place between the three concepts aforementioned. We conclude this chapter by a summary in which we discuss what is essential to be retained from these descriptions.

Before proceeding, however, we will present a motivating example that we will use throughout this chapter and the subsequent chapters to illustrate the forthcoming concepts.

Example 5.0.1 (Running Example) *In this example, we consider a set of participants (e.g. Alice, Bob, Chris, Dave, Eric and Felix) which aims at developing* mobile *applications. To that aim, these members tend to join together and create dynamic* open source *communities wherein applications are developed collaboratively.*

Open source software development is one of the most successful collaboration models. Based on this model, a number virtual communities have been formed in the last decade to make distributed *individual collaborate for the production of a common good. The number of people participating in* open source *projects is very large and the created communities are* open *without any* central authority*. In such context, the development of trust is especially important as empirical evidences [Osterloh and Rota, 2005] show that many participants in* open source *project are* conditionally cooperative *and that* trust *represents the principal motivation leading these individuals to cooperate.*

We will refer to this example in subsequent chapters whenever it serves to illustrate our approach.

5.1 The System Model Specification

In this section, we present the multi-agent-based virtual community model $\mathcal{S}$. As discussed in Section 4.4, we have chosen a *normative* multi-agent system in order to put in place a mixed control loop using the *top-down* and *bottom-up* mechanisms presented in the previous chapter (cf. Section 4.3). These mechanisms will be used later in Chapter 7 to enable *social* and *context* awareness trust management.

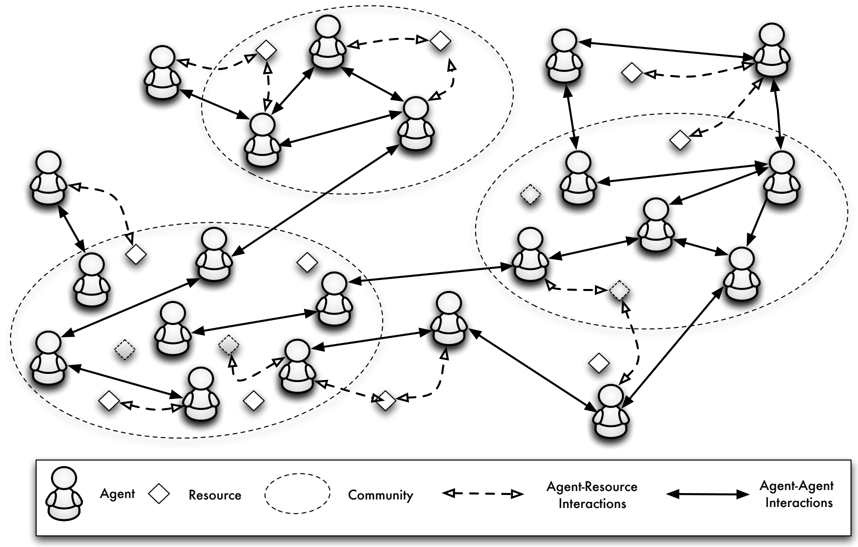


Figure 5.1 – An illustrative example of a population of agents with three multi-agent communities.

As illustrated in 5.1, agents are used as first level abstraction of virtual communities participants. This system $\mathcal{S}$ can be defined at a time t by a 5-uplet:

$$\mathcal{S}^t = \langle \mathcal{A}, \mathcal{R}, \mathcal{C}, \mathcal{I}, \Delta \rangle^t$$

where:

- $\mathcal{A}$ is the set of agents involved in $\mathcal{S}$,
- $\mathcal{R}$ is the set of resources that constitute the agents environment,
- $\mathcal{C}$ is the set of communities in which agents are organised,
- $\mathcal{I}$ is the set of interactions involving agents,

- Δ is the domain ontology.

In order to reduce the complexity of our descriptions, we wilfully refrain from delving into MAS detailed properties. Instead, we focus on the *agent* $(\mathcal{A})$, the *environment* $(\mathcal{R})$, the organisation $(\mathcal{C})$ and the *interaction* $(\mathcal{I})$ elementary dimensions with respect to the VOWELS paradigm [da Silva and Demazeau, 2002]. Before detailing each dimension, we start by defining the domain ontology Δ on which most of the other definitions rely.

5.2 Ontology

The ontology represents the agents' universe of discourse. It encapsulates all the knowledge that an agent must be aware of to achieve its activities.

Definition 15 (Ontology) *The domain ontology Δ is defined by:*

$$\Delta = \langle T, A, R \rangle$$

Where:

- *T and A are disjoint sets which elements are, respectively, terms and assertions (ABox). These concepts are commonly called TBox for terms, and ABox for assertions.*
- *$R = R_T \cup R_A \cup R_{TA}$ is a set of relations between, respectively, elements from T, elements from A and elements from $T \cup A$ such as :*
  - *$\{\sqsupset, \equiv, \not\equiv\}$ are, respectively, subsumption, equivalence and disjunction relations that compose R_T,*
  - *$\{\neq\}$ is the $DistinctFrom$ relation that compose R_A and which is used to differentiate elements from R_A,*
  - *$\{\doteq\}$ is the IsA relation that links each element from R_A to its corresponding class from R_T.*

The TBox T of the ontology is invariant. It contains statements that describe the terms (i.e. ontological concepts) used by the agents. The assertional box A can change over time, it contains assertions (i.e. ontological instances). The set R contains relations that may exist between elements from T, elements from A and elements from $T \cup A$. For instance, $\sqsupset$: $A \rightarrow A$ defines a partial order, called taxonomy or concepts hierarchy, over the elements of T. Similarly, the relation is an instance of (IsA) $\doteq$: $A \rightarrow T$ maps each assertion from A to the term that characterises its class.

Example 5.2.1 (Ontology) *For instance, the term image belongs to the TBox T of the ontology Δ, while the relation $JPEG \doteq image$ states that every assertion $JPEG$ is considered as an image.*

The elements that constitute the ontology will be progressively defined in the remaining of this chapter.

5.3 Resources

The system *environment* is defined as a dynamic set $\mathcal{R}$ of passive entities called resources. $\mathcal{R}$ is dynamic because the resources are unpredictably created, updated and destroyed.

Definition 16 (Resources) Resources *are artefacts that agents create, update, share and use. Each resource r is defined by:*

$$\forall r \in \mathcal{R}, r = \langle \varepsilon, \theta, P \rangle$$

where ε is the resource unique identifier[1] *in $\mathcal{S}$, θ is the content of the resource, P is the set of properties that r possesses.*

The *content* that resources encapsulate represents data, services or any functionality that supports agents individual and collective activities.

Remark 5.3.1 *From now onward, we will use the standard dot "." notation to refer to the property of an entity (resource, agent or community). For example, we will use $r.\varepsilon$ to refer to the resource identifier.*

Example 5.3.1 *Let us consider r_1 = "prog-v.13.jar" a file containing the Java source codes of a particular application, while r_2 = "calculator" is a web service that provides a scientific calculator. Both resources are uniquely identified using their URIs (Uniform Resource Identifier) as follows:*

$$r_1.\varepsilon = \texttt{http//www.emse.fr/yaich/Eureka/ABC/prog} - \texttt{v.13.jar}$$

$$r_2.\varepsilon = \texttt{http//www.emse.fr/calculator.jsp}$$

The content of r_1 constitutes the source code of the Java application, while the content of r_2 is the whole calculation methods provided by the calculator.

Resource Properties

Each resource is associated with a set of properties P that characterises it with respect to other resources.

Definition 17 (Resource Properties) *We define $P = \{\tau, \varphi, \Omega, \varsigma, \nu\}$ as the set of mandatory properties that any resource r from $\mathcal{R}$ possesses. τ refers to the resource type, φ is the identifier of the resource owner, Ω is the set of operations that can be performed on the resource, ς is its sensitivity and ν is the resource business value.*

These resource properties are paramount as our trust model relies on them. Of course, this set of properties can be further extended if other *application-specific* properties need to be added.

[1]For simplicity, we will use interchangeably the name of the resource and its identifier. Also, all identifiers correspond to unforgeable and verifiable via a public/private keys pairs.

Remark 5.3.2 *In the remainder, instead of using $r.P.x$ when referring to the property x of the resource r, we will use a contracted form $r.x$ to simplify our notation. For instance, $r.\tau$ refers to the type of the resource r.*

As defined above, each resource has a type that indicates which kind of content it encapsulates. The type of the resource determines also the nature of the operations that can be performed on it.

Definition 18 (Resources Types) *The type of a resource is defined by:*

$$\forall\, r \in \mathcal{R}, r.\tau \in \Delta_\tau$$

where Δ_τ is the sub-ontology that defines all resources types ($\Delta_\tau \subset \Delta$).

Example 5.3.2 *In our illustrative example, the considered set of resources types is $\Delta_\tau = \{text,$ $image,\ software,\ service\}$. This set is inspired from the taxonomies defined in the DCMI (Dublin Core Metadata Initiative [Weibel, 2000]) and from the MIME (Multipurpose Internet Mail Extensions [Borenstein and Freed, 1996]) as well. Of course, the resource types we give here are only examples, in reality we can have as many types as we need.*

Definition 19 (Resource Owner) *Each resource $r \in \mathcal{R}$ has an owner φ. The resource owner may be an agent or a community and is defined as follows:*

$$\forall r \in \mathcal{R}, r.\varphi \in \mathcal{A} \cup \mathcal{C}$$

A resource is considered as an *individual resource* when its owner is an agent. And a resource is considered as a *collective resource* when its owner is a community (i.e. it is owned by all the agents of the community).

Definition 20 (Resource Operations) *The operations an agent can perform on a resource are defined by:*

$$\forall r \in \mathcal{R}, r.\Omega = \{\omega_1, \ldots, \omega_m\}/r.\Omega \subset \Delta_\omega^{\mathcal{R}} \subset \Delta_\omega$$

where the ontology Δ_ω describes all possible operations agents can perform in $\mathcal{S}$, and the sub-ontology $\Delta_\omega^{\mathcal{R}}$ contains the operations that can be performed on a resource.

Operations represent the actions that agents perform to get advantage of services and functionalities that these resources offer. They represent also low level actions agents can use to sense and manipulate the environment where they evolve.

Example 5.3.3 *Let us consider the Jar file r_1 aforementioned. r_1 contains several Java files, each representing a* text document *that contains the source code of the program that Alice is developing. For instance, let us consider r_3 one of these files. The type of r_3 is text and the operations that can be performed on this type of resources are defined as follows:*

$$r_3.\Omega = \{create, read, update, delete\}$$

Of course, it belongs to the resource owner to allow such operations to be performed or not to others. This issue is not addressed by the system model $\mathcal{S}$ but by the trust model we will introduce in Chapter 6.

Definition 21 (Resource Sensitivity) *The resource sensitivity refers to the degree to which a resource can be* vulnerable *with respect to malicious behaviours. It is defined by:*

$$\forall r \in \mathcal{R}, r.\varsigma \in \Delta_\varsigma$$

The resource sensitivity ($\varsigma \subset \Delta$) expresses the negative consequences that an agent must assume if its resource is affected by a malicious behaviour. For simplicity, resources sensitivity constitutes an integer value that ranges from 0 to 10 ($\Delta_\varsigma = [0, 10]$). 0 means that the resource is not sensitive and 10 means that the resource is extremely sensitive. Based on this property, resources fall into two categories: *public* and *private*. Public resources are *non sensitive* resources that can be manipulated by any agent, while the manipulation of sensitive resources is limited to the resource owner and trusted agents. The model that agents use to decide which agent can be trusted constitute one of contributions of this thesis. It will be described in details in Chapter 6.

Definition 22 (Resource Value) *The value of a resource expresses the benefit an agent gains after manipulating it. We define this value by:*

$$\forall r \in \mathcal{R}, r.\nu \in \mathbb{N}.$$

The value of a resource is domain-dependent and application-specific. For some resources, the value is moral while for others it is purely economic. Therefore, and to refrain from going into such details, we use a simple numeric value to represent it . This value is of course dynamic; it increases if the resource has been further improved, and decreases if the resource was deteriorated.

Example 5.3.4 *The value of the resource r_1 corresponds to the value of the program conveyed by this resource. Thus if the program is further developed and improved, its value will increase. Contrariwise, if the program source code is* corrupted, vandalised *or* plagiarised *its value will decrease.*

Now that all resource properties have been introduced, we can present an example to illustrate what we have presented so far with a complete example about a resource.

Example 5.3.5 (resource) *Let us take the resource r_1 aforementioned. r is described in the system as follows:*

$$\langle r, \theta, P \rangle$$

where:

- $r_1.\varepsilon =$ `http://www.emse.fr/yaich/Eureka/ABC/prog-v.13.jar`

- *θ is the* `prog-v.13.jar` *file containing the source code of the software program*
- $P = \{alice, \texttt{text}, \{\texttt{read, update}\}, 5, 10\}$

In this example, the resource r_1 is identified by its URI. r_1 is owned by the agent alice. r is a text document ($r.\tau =$ `text`) that other agents can `read` *and* `update`*. However, as r is* sensitive *(i.e $r.\varsigma > 0$) only the agents that alice trusts are able to perform* `read` *and* `update` *operations on r_1.*

5.4 Agents

Agents are the core entities in the system $\mathcal{S}$. Agents are autonomous software entities able to perform some tasks in line with their goals (cf. Section 5.4).

Definition 23 (Agent) *We denote the set of agents $\mathcal{A}$ as a finite set such that is composed of at least two agents ($|\mathcal{A}| \geq 2$). Each agent a_i is defined by:*

$$\forall a_i \in \mathcal{A},\ a_i = \langle \varepsilon, G, \Lambda, L, R, C, P \rangle$$

where ε is the agent unique identifier[2] in $\mathcal{A}$, G is the set of goals it is trying to achieve, $\Lambda \subset \Delta_\omega$ is the set of actions (i.e. operations) that the agent a_i can perform, L is a set of plans a_i can accomplish to achieve its goals, R is a set of identifiers of resources that a_i owns, C is the set of identifiers of communities to which a_i belongs, and P is a set of properties that characterise a_i.

Each agent has a set G of goals to which the agent has to some extent committed. The achievement of each goal involves the performance of a sequence of actions (i.e. plans) on resources. These resources may be personal (i.e. owned by the agent) or non personal. Therefore, the fulfilment of the goals of the agent may requires the access to sensitive resources that are owned by other agents, making trust issue central in $\mathcal{S}$. Goals and plans will be presented later on in Section 5.5 and detailed in Chapter 9.

Agent Properties

$\mathcal{A}$ represents an heterogeneous population of agents. The heterogeneity of these agents is characterised in terms of *properties* they possess. We will present hereafter this notion which is fundamental in our thesis as the trust model we propose is inspired from the attribute-based access control models (cf. Section 3.1.6). Agent properties from P represent the description of the prevailing traits of an agent within the system.

[2]Again, for simplicity we will use interchangeably the name of an agent a_i and its identifier ε.

Definition 24 (Agent Properties) *Each property $p \in P$ is a property-value pair defined by :*

$$p = \langle \eta, v \rangle$$

where $\Delta_p \subset \Delta$ corresponds to the set of all properties that agents can have in $\mathcal{S}$, $p.\eta \in \Delta_p.T$ is the property name (i.e. the term defined in the ontology), and $p.v \in \Delta_p.A$ is the property value. We represent the properties of an agent a_i in the following format:

$$a_i.P = \{\langle p_1, v_1 \rangle, \langle p_2, v_2 \rangle, \langle p_3, v_3 \rangle, \ldots, \langle p_n, v_n \rangle\}$$

where p_j indicates the j^{th} property of the agent a_i and $p_j.v$ the value of the j^{th} property of a_i.

Properties are assigned values when the system is initialised. Any agent cannot have more than one value for each property it possesses. Property values express how much credit is accorded to the agent for each specific property. Importantly, we assume a total order over the set $\Delta_p.A$ such that any two distinct values can be compared. At this point, the operators we use for comparing such values are not important. These operators and the algebra we use for comparing properties values are outlined in Chapter 6.

Example 5.4.1 (Agent Properties) *For instance, Alice which is member of the ABC community can be characterised as follows:*

$$alice = \langle alice17@eureka, \Delta_\lambda, \{g_1, g_2, g_3\}, \Lambda, \{l_1, l_2, l_3\}, \{r_1\}, P, \{ABC\} \rangle$$

where

- *$alice17@eureka$ is Alice's identifier in the Eurêka platform,*
- *$alice.\Lambda$ is the set of actions that Alice can perform. $alice.\Lambda = \Delta_\lambda$ means that Alice can perform all possible actions,*
- *$alice.g_1$ = "earn money", $alice.g_2$ = "gain esteem" and $alice.g_3$ = "develop requested application" are the goals to which Alice committed to,*
- *l_i is the plan that alice uses to achieve its goals. For instance, $l_1 = \{r_1.read(), r_1.update\}$ is the plan by means of which Alice continuously updates its application r_1.*
- *$alice.r_1$ is the jar of the application developed by Alice (cf. Example 5.3.3),*
- *$alice.P = \{\langle Identity, F \rangle, \langle Competency, G \rangle,$ $\langle Experience, G \rangle, \langle Recommendation, B \rangle, \langle Reputation, G \rangle\}$,*
- *ABC is the community to which Alice belongs.*

For simplicity, we considered in our example that all properties have the same set of values $\mathcal{V} = \{N,\ VB,\ B,\ F,\ G,\ VG\ \}$ that refers, respectively to *null*, *very bad*, *bad*, *fair*, *good* and *very good* values. We also assume a total order over these values when an agent has the value N means that it has the worst value for p and VG means that the agent has the best value that an agent can have for that specific property.

5.5 Communities

In this thesis, we are particularly interested by the situations where, for some specific reasons, agents from the global population $\mathcal{A}$ form several temporary ad-hoc groups. We denote the set of all possible groups in $\mathcal{A}$ by $\mathcal{C}$ where $\mathcal{C} = \{c_1, c_2, c_3, \ldots, c_m\}$. We refer to each member c_i of $\mathcal{C}$ as a *virtual community.*

Definition 25 (Communities) Virtual communities *(communities for short) are sporadic and ephemeral organisations in which agents group together to achieve a common* objective. *Each* community *c is defined by:*

$$c = \langle \varepsilon, OS, A, R, plays, commits \rangle$$

where:

- *ε is the community unique identifier in $\mathcal{C}$,*
- *OS is the community organisational specification,*
- *A the set of agents identifiers (these agents are called the community members),*
- *R is the set of resources identifiers. These resources represent collective resources that are owned by the community, and by extension by all its members,*
- *$plays : \Re \rightarrow 2^A$ is a function that maps each role to the set of agents playing that role in the community c (cf. Definition 27),*
- *$commits : \mathcal{M} \rightarrow 2^A$ is a function that maps each mission to the set of agents that are responsible of achieving the mission.*

We do not require that the set of communities $\mathcal{C}$ completely cover the set $\mathcal{A}$ of agents. However, we suppose for the moment that $\forall c_i, c_j \in \mathcal{C}, c_i \neq c_j \implies c_i.A \cap c_j.A = \emptyset$. Informally, that means that an agent can be in only one community at a time.

The above community definition is based on the $\mathcal{M}oise$ meta-model [Hubner et al., 2002, Hubner, 2011] and its recent extension [Hübner et al., 2010]. In the following, we adapt the *organisational specification* as defined in $\mathcal{M}oise$ to virtual communities.

Definition 26 (Organizational Specification) *An organisational specification (OS) is defined by three dimensions (cf. Section 4.4):*

$$OS = \langle SS, FS, NS \rangle$$

where:

- *SS is the structural specification,*
- *FS is the functional specification,*

- *NS is the normative specification.*

As its name indicates, the community organisational specification is used by the agents to understand and reasons about the structure, the functioning and the norms of the communities to which they belong.

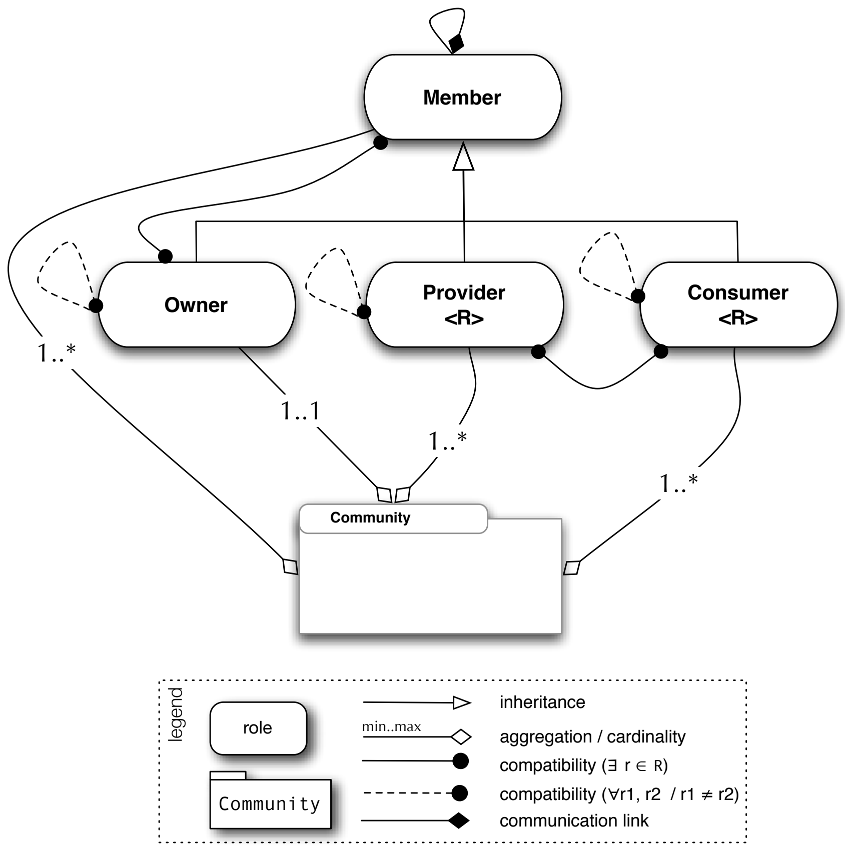


Figure 5.2 – The structural specification of virtual communities.

Definition 27 (Structural Specification) *The structural specification SS (cf. Figure 5.2) of a community is defined by [Hubner, 2011]:*

$$\langle \Re, \sqsubset, GS \rangle$$

where:

- $\Re = \{owner, member, provider, consumer\}$ *is the set of roles available in each community,*

- $\sqsubset$ *is an inheritance relation among roles,*

- *GS is the specification of the root group of the community.*

Roles are descriptions of responsibilities and/or expected functions of agents adopting them. A community structural specification is composed of four basic roles (cf. Figure 5.2): *owner*, *member*, *provider* and *consumer* to which agents can commit. These roles constitute the minimal set of roles that a community can propose. Of course, it can be further extended locally by any community whenever there is a need for such an extension. An agent can adopt as many roles as it wants but every agent must adopt at least the role *member*.

Definition 28 (Roles) *We define a role by:*

$$\forall r \in \Re, r = \langle \varepsilon, AR, LR \rangle$$

where ε is the role identifier, AR are the conditions that an agent must fulfil to adopt the role and LR is the set of conditions that an agent must satisfy to leave the role.

We briefly describe hereafter each of the roles that comprises our structural specification:

Member. Once an agent joins the community, it automatically adopts the role of *member*. This basic role permits agents to get access to the resources of the community.

Owner. This role is systematically adopted by the agent that created the community. It gives him the power to admit new members. As our communities are decentralised, every new members can adopt the role of community owner and have the same rights as the owner.

Provider. Whenever an agent creates a resource it adopts the role of resource *provider*. Committing to this role means that the agent is responsible of answering access requests to the resource it is sharing. Subsequently, it is also responsible of issuing and enforcing policies. Using these policies, the agent is able to select trustworthy agents to which it will provide access.

Consumer. This is the role that agents adopt when they want to use a resource.

The roles of *owner*, *provider* and *consumer* are compatible. Thus, each owner participates in the activities of the community as any other member. Based on Figure 5.2, the following inheritance relationships exist:

$$owner \sqsubset member, provider \sqsubset member, consumer \sqsubset member$$

Definition 29 (Group Specification) *Based on $\mathcal{M}oise$ and its extensions, our group specification (GS) is defined by the elements of the following tuple:*

$$\langle Gr, SG, L^{inter}, L^{intra}, C^{inter}, C^{intra}, cg, cr \rangle \tag{5.1}$$

where:

- *Gr is the set of roles available in the group ($Gr \in \Re$); in our model, communities and groups have the same set of roles,*

- *SG is the set of subgroups of the group; in our model, there is no subgroups therefore $SG = \emptyset$,*

- *$L^{inter} = L^{intra} = \{com\}$ are, respectively, the inter-group and intra-group links such defined in [Hübner et al., 2010] (The communication link is the only link that exists between agents in $\mathcal{S}$),*

- *$C^{inter} = \emptyset$ is an inter group compatibility constraint,*

- *$C^{intra} = \{consumer \overset{r}{\bowtie} provider\}$[3] is an intra-group compatibility constraint,*

- *$cg : Gr \rightarrow \mathbb{N}$ is a group cardinality function that maps each group with the number of instances that can be created within a community; there can be only one group in a community so $cg = 1$,*

- *$cr : Gr \rightarrow \mathbb{N} \times \mathbb{N}$ is a role cardinality function that maps each role of the group to minimal and maximal number of agents that can adopt it in the community; with respect to that:*

  - $community.cr(owner) = (1, *)$[4]
  - $community.cr(member) = (1, 1)$
  - $community.cr(consumer) = (0, *)$
  - $community.cr(provider) = (0, *)$

Definition 30 (Functional Specification) *The functional specification (FS) of a virtual community is defined by:*

$$FS = \langle G, M, S \rangle$$

where:

- *$G = \{createCom, build, manage, admit, exclude, destroy, cooperate, produce, create, delete, consume, request, perform\}$ is the set of community goals identifiers,*

- *$M = \{mCreat, mCert, mProd, mCont, mCons, mMon\}$ is the set of community missions identifiers,*

- *$S = \{ComCrSch\}$ is the set of scheme specification of the community.*

[3]The expression *consumer* $\overset{r}{\bowtie}$ *provider* means the roles of consumer and provider are compatible only for a specific resource r.

[4]$*$ means that the upper bound cardinality of the role does not have any limit

The functional specification of a community c aims at specifying the common goals $c.G$ that its members $c.A$ try to accomplish, how these goals are subsequently divided into missions to which agents can commit, and the social scheme that defines how goals are scheduled and grouped into missions. Figure 5.3, illustrates the functional specification we defined on the top of the structural specification presented above.

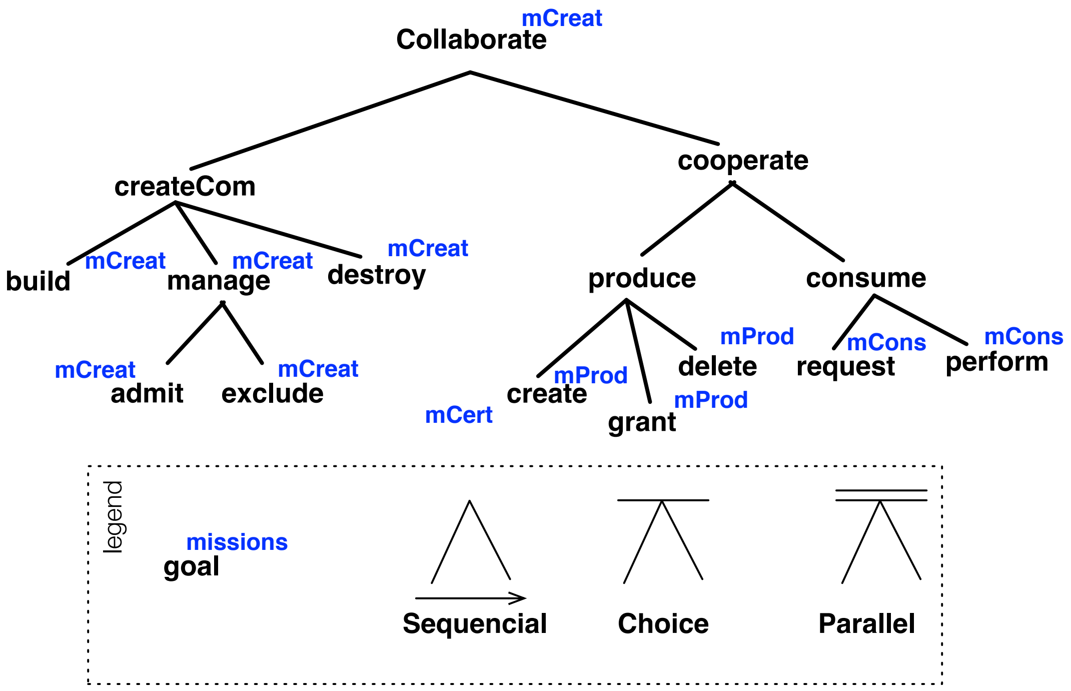


Figure 5.3 – The functional specification of a virtual community.

Definition 31 (Scheme Specification) *A scheme specification is defined through the elements of the following tuple:*

$$S = \langle \varepsilon, max, min, gr \rangle$$

where:

- ε *is a unique identifier of the scheme,*
- $max : M \rightarrow \mathbb{N}$ *is a function that maps each mission from the functional specification* FS *to the maximum number of commitments to that mission in the scheme,*
- $min : M \rightarrow \mathbb{N}$ *is a function that maps each mission from the functional specification* FS *to the minimum number of commitments to that mission to consider the scheme valid,*
- $gr = \{community\}$ *is the root and unique group of the scheme.*

Definition 32 (Goal) *Each agent goal is defined as follows:*

$$\langle \varepsilon, gr, type, card, ttf, P \rangle$$

where:

- *ε is the goal identifier,*
- *$gm : 2^M$ is the set of missions that comprise the goal,*
- *$type \in \{ach, maint\}$ is the type of goal (either achievement on maintenance),*
- *$card \in \mathbb{N}$ is the cardinality of the goal (i.e. how many agents have to achieve the goal for the goal to be considered satisfied),*
- *$ttf \in \mathbb{N}^*$ is the time to fulfil the goal,*
- *P is the plan to achieve the goal.*

Definition 33 (Plan) *A plan is defined by:*

$$\langle h, G_p, op \rangle \tag{5.2}$$

where $h \in G$ is the head goal of the plan is trying to achieve, $G_p = \{g_1, \ldots, g_n\}$ is a set of sub-goals and $op \in \{sequence, choice, parallel\}$ is the operator among the sub-goals. The operator is used to state whether one or all sub-goals have to be achieved to accomplish h and whether their achievement has to be in sequence or in parallel.

The complete function specification of our model is depicted in Figure 5.2.

Roles are defined in term of responsibilities and/or expected functions but also in terms of *norms* that governs them. We use norms to exert *informational decentralised control* by specifying what constitutes acceptable behaviours within the community.

Definition 34 (Normative Specification) *The normative specification of a community is defined by:*

$$NS = \langle N \rangle$$

where N is a set of norms.

The normative specification aims at bridging roles from the structural specification (SS) and the missions from the functional specification (FS) through deontic operators[Hübner et al., 2010].

Definition 35 (Norm) *A norm is defined by:*

$$\forall n \in N, n = \langle \varepsilon, c, r, t, m, ttf \rangle \tag{5.3}$$

where:

- *ε is the norm identifier (or label),*
- *c is the norm activation condition (i.e. the condition that must hold to make the norm active),*
- *r is the role to which the norm is subject,*
- *$t \in \{obligation, permission, forbidding\}$ is the norm type that corresponds to the deontic operator conveyed by the norm,*
- *$m \in FS.M$ is the mission to which the norm is associated,*
- *ttf (time to fulfil) is the deadline to consider the norm either fulfilled or not.*

The definition of norms given above can be read "when c holds, the agents adopting r are t to achieve the missions m before ttf". The initial norms defined in our model are presented in Table 5.1.

$n.\varepsilon$	condition	role	type	mission	TTF
n_1	—	*owner*	*permission*	$mCreat$	—
n_3	—	*provider*	*obligation*	$mProd$	—
n_5	fulfilled(n_3)	*controller*	*obligation*	$mCont2$	—
n_6	notfulfilled(n_5)	*controller*	$forbidding$	$mCont1$	—
n_7	fulfilled(n_3)	*consumer*	*permission*	$mCons$	—

Table 5.1 – Excerpt from the normative specification of a virtual community.

Whenever an agent adopts a certain role on which a norm is associated, this agent implicitly and automatically adopts the norm to which the role is subject. So concretely, a norm is about specifying what *operations* an agent playing some specific role is *permitted*, *obliged*, *forbidden* to perform.

Permissions specify what is considered by the community members as a normal behaviour. Permissions allows an agent to perform operations it is permitted to perform. In other words, permissions represent the rights an agent has [Boella and Torre, 2005].

Obligations define what the community members are entitled to expect from the agent adopting the role. For instance, in the norm $n_1$the agents adopting the role of *owner* has the *obligation* to commit to the missions $mCreat$ where they commit to create and manage a community.

Forbiddings are obligations to not commit to certain missions or to not perform specific operations. For instance, the norm n_6 forbids agents adopting the role of controller to *grant* or *delegate* the use of a resources without satisfying the norm n_5.

Assumption 5.5.1 *By default, agents are prohibited from performing any operation on any resource unless they have an authorisation.*

Whenever an agent joins a community, it commits to a specific role and starts interacting with another agent, it is systematically governed by some norms that should constraint his behaviour. However, in our system agents (i.e. community members) are autonomous entities that may choose to not comply with the norms of their community. Such situation constitutes a *violation* and motivates the need for incentive mechanisms such as *sanctions* and *rewards*.

Definition 36 (Sanction and Reward) Sanctions *and* rewards *constitute the principal motivation that lead an agent to fulfil the norms.* Sanctions *are negative actions toward the deviant agent, while* rewards *are positive actions towards it. Agents tend to fear* sanctions *and desire* rewards *[Boella and Torre, 2005], therefore we use them as a mechanisms of social control (cf. Section 4.3.2.2).*

5.6 Interactions

Based on the definitions given in Sections 5.3, 5.4 and 5.5, the system $\mathcal{S}$ can be seen as a digital ecosystem where agents organised in communities coexist with the resources that compose their environment. The existence of such an ecosystem presupposes the existence of *interactions* among the entities that evolve in it.

An interaction has been defined in the Merriam-Webster Dictionary as a "mutual or reciprocal action or influence". Our definition of interaction emerges from this.

Definition 37 (Interaction) *An interaction is an interactive and iterative data exchange (computation) that* involves *and* affects *at least one agent. The set of interactions $\mathcal{I}$ is defined by:*

$$\mathcal{I} = \mathcal{I}_r \cup \mathcal{I}_c \cup \mathcal{I}_c$$

where $\mathcal{I}_a$ is the set of interactions that take place between two agents, $\mathcal{I}_r$ is the set of interactions that take place between and the agents and the resources of their environment, $\mathcal{I}_c$ is the set of interactions that take place between the agents and their communities.

Agent-resource interactions refers to the ability of an agent to monitor and change the environment in which it evolves. They represent the interactions that occur between an agent and the resources of its environment. *Agent-agent interactions* refers to the agents' capability to achieve one-to-one communications with each others. *Agent-community interactions* refers to the agents' ability to understand, comply with, violate, reason about, and change the norms to which they are subject. They describe the set of one-to-many interactions that occurs between an agent and the members of its community.

By executing operations on resources, an agent can interact with its environment (cf. Section 5.3). For instance, if the resource is a sensor it will encapsulate an information and by invoking the *read* operation on it, the agent will receive an information regarding its environment. Analogously, if the operation modifies a resource in the environment, the environment will change. Then the agent's (and other agents') representation of the environment will change as well. Consequently, this aspect of the agent interactions will not be further details.

In contrast, agents communicate with each others (whether in a one-to-one or a one-to-many mode) using messages and based on a common protocol.

Definition 38 (Message) *$\mathcal{I}_a \cup \mathcal{I}_c$ represents the set of communications taking place between agents. Each element $m \in \mathcal{I}_a$ constitute a message that is defined as follows:*

$$\langle s, u, r, \varepsilon, \theta, L, \Delta_{\mathcal{I}}, prot, t \rangle$$

This means that $s \in \mathcal{A}$ is using the performative utterance u to communicate with $r \in \mathcal{A}$ in the context of the dialogue identified by ε, with the content θ, in the language L, using the ontology $\Delta_{\mathcal{I}}$, in the context of the protocol prot and at a time t [Pitt and Bellifemine, 1999]. However, for simplicity we will use in what follows a contracted version of the message in which we abstract away, the dialogue identifier ε, the language L, the ontology $\Delta_{\mathcal{I}}$, the protocol prot and the time t. Thus a message can be defined as follows

$$\langle s, u, r, \theta \rangle$$

So agents are interacting with each others through message exchanges. In order to provide interoperability between messages that agents use in their communications, the definition of an interaction protocol is crucial. To that aim, a commonly understood agent communication language (ACL) with a formal semantics has to be used. In MAS, communication language semantics is typically characterised in terms of speech act theory [Austin, 1962, Cohen and Levesque, 1997].

Definition 39 (Agent Communication Language) *The communication language L that agents use in $\mathcal{S}$ is defined by:*

$$L = \langle Protocol, Utterance, Replay \rangle$$

where:

- *$Utterance$ = {`request`, `inform`, `cfp`, `failure`, `inform_if`, `inform_ref`, `refuse`, `agree`, `subscribe`, `request_when`, `request_whenever`, `propagate`, `null`, `not_understood`} is a set of performatives with respect to the semantics defined in FIPA-ACL SL language [FIPA, 2002],*
- *$Protocol$ = {`FIPA_request`, `FIPA_query`} is a set of protocols that agents use,*
- *$replay : Utterance \times Protocol \times \Re \times \mathbb{N} \rightarrow Utterance \times Utterance$ is mapping function that maps each $performative$ from Utterance to the set of performatives that can be used in a reply to it given the role played by the agent.*

The *replay* function is easily determined by turning the *Utterance* set into a finite state diagram. For instance, applying this transformation to the `FIPA_request` protocol is straightforward and gives the result shown in Figure 5.4 [Pitt and Bellifemine, 1999].

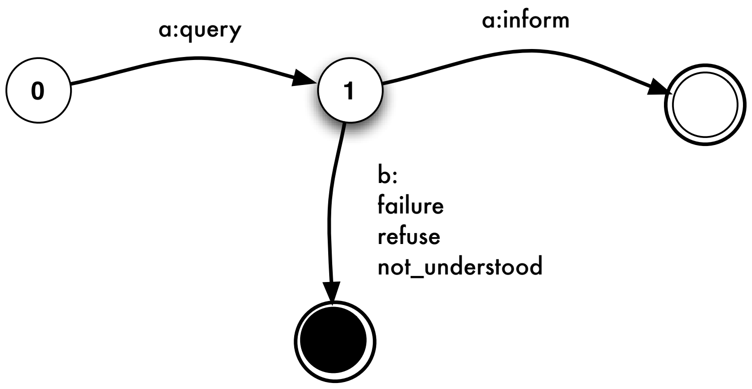


Figure 5.4 – FIPA_request protocol represented using a finite state diagram.

5.7 Conclusion

In this chapter, we have presented the multi-agent-based virtual community framework as well as fundamental concepts that underpin the contributions of this thesis. We adopted a top-down description method in which we started by defining the system as a whole to end up by describing each of the elements that compose it. We have also defined some notational elements that we will rely on in subsequent chapters, and we used some examples to illustrate our choices.

What the reader should keep in mind is that the system is composed of resources, agents and communities, each having specific properties. We presented also how agents can interact with each others and with the resources that compose their environment. While public resources can be freely manipulated, private resources manipulation is limited to agents that are trusted by the resource owner, making the trust issue central in this framework. Also, each agent can be characterised through its properties. So agents are able to discriminate each other based on they possess and those they lack. We will present in the next chapter how we built on these elements to build a *trust management systems* that allows agent to evaluate the trust they put in each others.

5.8 French Summary

L'adoption du paradigme agent et multi-agent est considérée actuellement comme étant une des approches les plus viables pour la modélisation et l'implémentation des communautés virtuelles (CV) [Camarinha-Matos et al., 2003, Rupert et al., 2007, Gupta and Kim, 2004]. Ainsi, dans ce chapitre, nous présenterons notre modèle de communautés virtuelles à base d'agents. Ce modèle comporte les éléments de base de notre mécanisme de gestion de la confiance que nous présenterons dans les chapitres 6 et 7. Cependant, avant de rentrer dans les détails du modèle, nous introduisons l'exemple que nous utiliserons tout au long de ce chapitre et les chapitres suivants pour illustrer les différents concepts abordés.

Example 5.8.1 (Exemple illustratif) *Dans cet exemple, nous considérant un ensemble de participants (e.g., Alice, Bob, Chris, Dave, Eric et Félix) dont l'objectif est de mettre à profit leurs compétences informatiques pour développer des applications mobiles. Pour cela, ces participants ont tendance à vouloir se regrouper afin de créer des communautés "open source" au sein desquels les applications seront développées de manière collaborative.*

Devant le succès grandissant que connaissent les modèles de collaboration du type "open source", plusieurs communautés virtuelles se sont formées durant cette dernière décennie. Leur principal objectif est de permettre à des individus dispersés sur la surface de la terre de collaborer pour la production d'un produit commun. Le nombre de participant étant assez large et les communautés créées étant souvent *ouvertes* et *décentralisées* (i.e., aucune autorité centrale), des études ont démontré que la collaboration des membres de ces communautés est conditionnée par les degrés de confiance qu'ils s'accordent mutuellement. Ainsi, il est évident que la confiance joue un rôle clé dans le succès et l'essor des communautés virtuelles telles que celle qu'on a cité comme exemple.

5.8.1 Spécification de la communauté virtuelle

Dans cette section, nous présentons notre modèle de communauté virtuelle à base d'agents $\mathcal{S}$. Nous avons opté pour un système multi-agent dit *normatif* afin d'être en mesure de mimer le contrôle mixte (*ascendant* et *descendant*) que nous avions décrit dans le chapitre 4 (cf. Section 4.3). C'est également sur ces mécanismes que nous nous appuierons dans le Chapitre 7 afin de mettre en place la sensibilité *sociale* et *contextuelle* dans la gestion de la confiance.

Comme illustré dans la figure 5.5, les agents sont utilisés comme éléments d'abstraction de premier ordre pour représenter les participants dans les communautés virtuelles. Le système $\mathcal{S}$ est ainsi défini a un temps t par un quintuplet:

$$\mathcal{S}^t = \langle \mathcal{A}, \mathcal{R}, \mathcal{C}, \mathcal{I}, \Delta \rangle^t$$

où:

- $\mathcal{A}$ représente l'ensemble des agents impliqué dans $\mathcal{S}$,

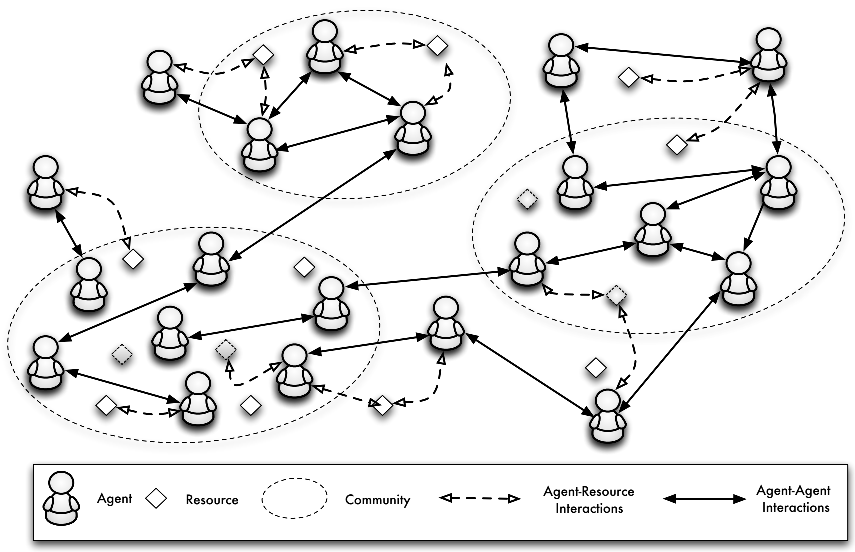


Figure 5.5 – Exemple illustratif d'une population d'agents avec trois communautés.

- $\mathcal{R}$ est l'ensemble des ressources représentant l'environnement des agents,
- $\mathcal{C}$ est l'ensemble des communautés au sein desquels les agents s'organisent,
- $\mathcal{I}$ est l'ensemble des interactions dans lesquels les agents du système sont impliqués,
- Δ est l'Ontologie du domaine.

Dans la suite de ce chapitre, nous allons décrire en détail chaque élément de ce système. Cependant, afin de réduire la complexité de notre description, nous allons nous baser sur l'approche voyelles [da Silva and Demazeau, 2002] afin de nous concentrer sur les *agents ($\mathcal{A}$)*, l'*environnement ($\mathcal{R}$)*, l'*organisation*($\mathcal{C}$) et les interactions ($\mathcal{I}$) de notre système. Néanmoins, avant de présenter chaque élément, nous allons commencer par introduire notre Ontologie Δ car c'est sur celle-ci que les autres définitions reposent.

5.8.2 Ontologie

L'Ontologie représente l'univers du discours des agents du système. Elle encapsule l'ensemble des connaissances que chaque agent du système est censé savoir afin de réaliser ses objectifs.

Definition 40 (Ontology) *L'Ontologie du domaine Δ est définie par:*

$$\Delta = \langle T, A, R \rangle$$

où:

- *T et A sont des ensembles disjoints dont les éléments sont, respectivement, les termes et les assertions. Ces éléments sont communément appelé TBox pour les termes, et ABox pour les assertions.*

- *$R = R_T \cup R_A \cup R_{TA}$ est un ensemble de relations, respectivement, entre les éléments de T, les éléments de A et les éléments de $T \cup A$ tel que:*

  - *$\{\sqsupset, \equiv, \not\equiv\}$ représentent, respectivement, les relations de subsomption, équivalence et disjonction qui constituent R_T,*
  - *$\{\neq\}$ est la relation DistinctDe qui est utilisée pour différencier les éléments de R_A,*
  - *$\{\doteq\}$ est la relation Est qui relie chaque élément de A a sa classe de T.*

5.8.3 Ressources

L'environnement des agents est défini dans notre modèle comme étant un ensemble $\mathcal{R}$ d'entités passives appelées ressources. $\mathcal{R}$ est dynamique car ces ressources sont créées, modifiées et détruites de manière imprédictible.

Definition 41 (Resources) *Les* Ressources *sont des artefacts créées, modifiés, utilisés, partagés et détruits par les agents. Chaque ressource r est définie par :*

$$\forall r \in \mathcal{R}, r = \langle \varepsilon, \theta, P \rangle$$

où ε est l'identifiant unique de la ressource dans $\mathcal{S}$, θ est le contenu de la ressource, P est l'ensemble des propriétés.

Le *contenu* de la ressource représente des données, des services ou tout fonctionnalités permettant aux agents de réaliser leur activités individuelles ou collectives.

Remark 5.8.1 *Dans la suite de ce document, nous allons utiliser la notation pointée "." pour faire référence à la fois aux propriétés des ressources mais également celles des agents et des communautés. Par exemple, nous utiliserons $r.\varepsilon$ pour faire référence à l'identifiant de la ressource r.*

Chaque ressource est associée à un ensemble de propriétés P qui permet de la caractériser.

Definition 42 (Propriétés des Ressources) *Les propriétés sont définis par :*

$$P = \{\tau, \varphi, \Omega, \varsigma, \nu\}$$

où:

- *τ est le type de la ressource (e.g., image, texte, son, vidéo, dessin, etc.),*

- *φ est la propriétés de la ressource. Le propriétaire peut être un agent ou une communauté,*

- Ω *est l'ensemble des opérations qu'admet la ressource. Par exemple, un texte peut être modifié, supprimé, copié.*
- ς *est la sensibilité de la ressource. Cette valeur exprime le degré de vulnérabilité de la ressource quant aux manipulations malveillantes.*
- ν *est la valeur métier de la ressource.*

Example 5.8.2 (ressource) *Prenons l'exemple d'une ressource r_1. r_1 peut être décrit dans notre système par:*

$$\langle r_1, \theta, P \rangle$$

où:

- $r_1.\varepsilon =$ `http://www.emse.fr/yaich/Eureka/ABC/prog-v.13.jar`
- θ *est le fichier* `prog-v.13.jar` *contenant les sources de l'application,*
- $P = \{alice, \texttt{text}, \{\texttt{read, update}\}, 5, 10\}$

Dans cet exemple, la ressource r_1 est identifiée par son URI. r_1 est la propriété de alice, r_1 est un document texte ($r.\tau =$ `text`) que les agents peuvent lire *(read) et modifier (update). Cependant, compte tenu que r_1 est sensible (i.e $r.\varsigma > 0$), seuls les agents en qui alice a confiance peuvent le lire et y écrire. Bien sûre, à ce stade du manuscrit, nous n'avons pas encore abordé les mécanismes sur lesquels alice pourrait s'appuyer pour décider en qui elle peut avoir confiance. Nous les verrons dans le chapitre suivant.*

5.8.4 Agents

Les agents sont l'élément central du système $\mathcal{S}$. Les agents sont des entités logicielles autonomes capables de réaliser des tâches en accord avec leurs objectifs (cf. Section 5.4).

Definition 43 (Agent) *Soit $\mathcal{A}$ l'ensemble des agents du système $\mathcal{S}$. Chaque agent $a_i \in \mathcal{A}$ est défini par:*

$$\forall a_i \in \mathcal{A},\ a_i = \langle \varepsilon, G, \Lambda, L, R, C, P \rangle$$

où ε est l'identifiant de l'agent, G est l'ensemble des buts de l'agent, $\Lambda \subset \Delta_\omega$ est l'ensemble des actions (i.e., opérations) que l'agent a_i est en mesure de réaliser, L est l'ensemble des plans que a_i peut accomplir pour réaliser ses buts, R est l'ensemble des identifiants de ressources que a_i possède, C est l'ensemble des identifiants des communautés auxquelles a_i appartient et P est l'ensemble des propriétés qui permettent de caractériser a_i.

La réalisation des objectifs de chaque agent implique la manipulation de ressources personnelles et non personnelles. Ainsi, pour accéder à des ressources non personnelles, l'agent à besoin que l'agent qui possède la ressource lui fasse confiance pour la manipuler. Ainsi, la problématique de la confiance apparaît ici encore comme un élément central de la réussite à la fois des agents mais également des communautés dans lesquelles ces agents s'activent. Les concepts de but et de plan seront décrit plus tard dans la section 5.5 puis détaillés dans le chapitre 9.

Enfin, chaque agent est caractérisé par un ensemble de propriétés. Ces propriétés constituent une description des traits prévalents de l'agent dans le système.

Definition 44 (Propriétés des agents) *Chaque propriétés $p \in P$ est définie par une paire:*

$$p = \langle \eta, v \rangle$$

où p_j indique la $j^{\text{ème}}$ propriétés de l'agent a_i et $p_j.v$ est sa valeur. On assume un ordre total sur $\Delta_p.A$ tel que n'importe quelle paire de de valeurs puisse être comparés. À ce stade, l'ensemble des opérateurs qu'on utilise pour comparer ces valeurs n'est pas important. Ces derniers seront introduits dans le Chapitre 6.

5.8.5 Communautés

Dans cette thèse, nous somme particulièrement intéressés par les situations dans lesquels les agents forment des groupes temporaires ad-hoc. On représente l'ensemble de ces groupes par $\mathcal{C} = \{c_1, c_2, c_3, \ldots, c_m\}$. Chaque élément de $\mathcal{C}$ constitue une communauté virtuelle.

Definition 45 (Communautés) Les communautés virtuelles *sont des organisations sporadiques et éphémères dans lesquels les agents se regroupent afin de réaliser un objectif commun. Dans notre système, chaque communauté est définie par :*

$$c = \langle \varepsilon, OS, A, R, plays, commits \rangle$$

où:

- *ε est l'identifiant de la communauté dans $\mathcal{C}$,*
- *OS est la spécification organisationnelle de la communauté,*
- *A est un ensemble d'identifiants d'agents (ces agents représentent les membres de la communauté),*
- *R est un ensemble d'identifiant de ressources. Ces ressources sont des ressources collectives car ils appartiennent à l'ensemble des membres de la communauté,*
- *$plays : \Re \rightarrow 2^A$ est une fonction qui associe à chaque rôle l'ensemble des agents qui jouent ce rôle au sein de la communauté c,*
- *$commits : \mathcal{M} \rightarrow 2^A$ est une fonction qui associe chaque mission à l'ensemble des agents qui sont responsables de la satisfaction de cette mission.*

Notre définition de la communauté est basé sur le méta-modèle $\mathcal{M}oise$ [Hubner et al., 2002, Hubner, 2011] ainsi que son extension récente [Hübner et al., 2010]. Pour des raisons de format, nous ne pouvons détailler l'ensemble des spécifications qui constituent la spécification organisationnelle telle que définie dans $\mathcal{M}oise$. Nous invitons le lecteur à lire la version complète du manuscrit (en Anglais) pour plus de détails sur les aspects liés à la spécification de la communauté.

5.8.6 Interactions

D'après les définitions présentées précédemment, le système $\mathcal{S}$ peut être considéré comme un eco-système dans lequel les agents organisés en communautés coexistent avec les ressources qui constituent leur environnement. L'existence d'un tel eco-système présuppose l'existence d'intenses interactions entre les différentes entités qui le composent.

Definition 46 (Interaction) *Une interaction est un échange de donnée itératif qui implique et affect au moins un agent. L'ensemble des interactions $\mathcal{I}$ est défini comme suit:*

$$\mathcal{I} = \mathcal{I}_r \cup \mathcal{I}_c \cup \mathcal{I}_c$$

Ici $\mathcal{I}_a$ dénote l'ensemble des interactions qui prennent lieu entre deux agents, $\mathcal{I}_r$ dénote les interactions qui ont lieu entre les agents et les ressources et $\mathcal{I}_c$ dénote les interactions qui prennent place entre les agents et leurs communautés.

Les interactions *Agent-Ressources* font référence à la capacité des agents à surveiller et affecter l'environnement dans lequel ils évoluent. Les interactions de type *Agent-Agent* référent à la capacité des agents à établir des communication mono-interlocuteur. Enfin, les interactions de type *Agent-Communauté* font référence à la capacité des agents à comprendre, se conformer, violer, raisonner sur modifier les normes qui lui sont imposées par la communauté à laquelle il appartient. Ce type d'interaction fait également référence à la capacité des agents à entrer des des communications multi-interlocuteur.

En exécutant des opérations sur des ressources, l'agent est capable de d'interagir de manière simple avec son environnement. Par ailleurs, les communications entre agents (que ce soit mono ou multi-interlocuteur) s'opèrent via un échange de messages.

Definition 47 (Message) *$\mathcal{I}_a \cup \mathcal{I}_c$ représente l'ensemble des communications qui ont lieu entre agents. Chaque élément de $m \in \mathcal{I}_a$ représente un message qu'on défini comme suit:*

$$\langle s, u, r, \varepsilon, \theta, L, \Delta_{\mathcal{I}}, prot, t \rangle$$

Cette définition peut être traduite par l'agent $s \in \mathcal{A}$ qui utilise performative u pour communiquer avec $r \in \mathcal{A}$ dans le contexte du dialogue identifié par ε. Le message est utilisé pour transmettre le contenu θ, dans le langage L et en utilisant l'Ontologie $\Delta_{\mathcal{I}}$, dans le contexte du protocole $\Delta_{\mathcal{I}}$ au moment t. Pour des raisons de simplicité, nous définissons un message par la forme contractée suivante:

$$\langle s, u, r, \theta \rangle$$

Pour assurer l'interopérabilité des messages, la définition d'un protocole de communication est une étape primordiale. Dans les systèmes multi-agent, la sémantique des messages est caractérisée en matière de théorie d'acte de langage [Austin, 1962, Cohen and Levesque, 1997].

Definition 48 (Langage de Communication entre Agents) *Le langage de communication entre agents L utilisé dans $\mathcal{S}$ est défini comme suit:*

$$L = \langle Protocol, Utterance, Replay \rangle$$

où:

- $Utterance = \{$`request`, `inform`, `cfp`, `failure`, `inform_if`, `inform_ref`, `refuse`, `agree`, `subscribe`, `request_when`, `request_whenever`, `propagate`, `null`, `not_understood`$\}$ *est l'ensemble des performatives dont la sémantique est définie par le langage FIPA-ACL [FIPA, 2002],*
- $Protocol = \{$`FIPA_request`, `FIPA_query`$\}$ *est un ensemble de protocoles que les agents utilisent,*
- $replay : Utterance \times Protocol \times \Re \times \mathbb{N} \rightarrow Utterance \times Utterance$ *est une fonction de correspondance qui associe chaque $performative$ à l'ensemble des performatives que l'agent interlocuteur peut utiliser pour répondre à un message donné.*

Chapter 6

The Trust Management System

We framed in Chapter 5 a *decentralised* virtual community model that aims to make cooperate distributed participants based on common interests and/or objectives. In such socio-technical systems (i.e. systems involving human users and man-made agents) participants are massively interacting with each other through messages exchange and resources sharing [Sterling and Taveter, 2009]. An interaction always bears the risk that one partner exhibits uncooperative or malicious behaviour, making trust a central issue for each participant [Falcone and Castelfranchi, 2001].

To this aim, this chapter presents the model used to build and manage trust among members of virtual communities. In the following members will be called individuals to clearly make the difference with communities. The chapter is organized in two separate parts. First we describe our model (Section Section 6.1, 6.2, 6.3 and 6.4). Then we present in Section 6.8 how the virtual community model presented in the previous chapter can be extended to integrate trust management mechanisms proposed in this Chapter.

Finally, we conclude this chapter in Section 6.7 where we sketch the architecture of the trust management system that implements our *trust model* and discuss some of its limits.

6.1 Overview of the Approach

Given the large number of interactions that take place in agent-based virtual communities, and considering that many of these interactions are performed by agents acting on behalf of users, the automation of the decision making process, especially the trust decision one, is a critical concern. Thus, the main objective of our investigations is to design a model that contributes to such an automation. However, before we proceed we will first clarify what we mean by a *trust management model.*

Definition 49 (Trust Management Model) *A trust management model (trust model for short) is used to,* (i) *represent the information based on which trust is evaluated,* (ii) *define a formalism to express constraints on these information, and* (iii) *specify the evaluation scheme used to derive trust from that information.*

Starting from the above definition, we provide in Figure 6.1 an abstract illustration of the use of our trust model in a particular interaction. Here, the interaction involves a *requester* (i.e. resource *consumer* in Section 5.5) and a *controller*. Controller is a role that a virtual community member adopts with respect to a particular resource. Being the controller of a resource means that the member is the owner of the resource (i.e. *provider*) or that this agent has been delegated the right to make decisions about this resource. The requester aims at performing a particular operation on a resource controlled by the controller. To that aim, he must gain a sufficient trust form this latter in order to see its request accepted.

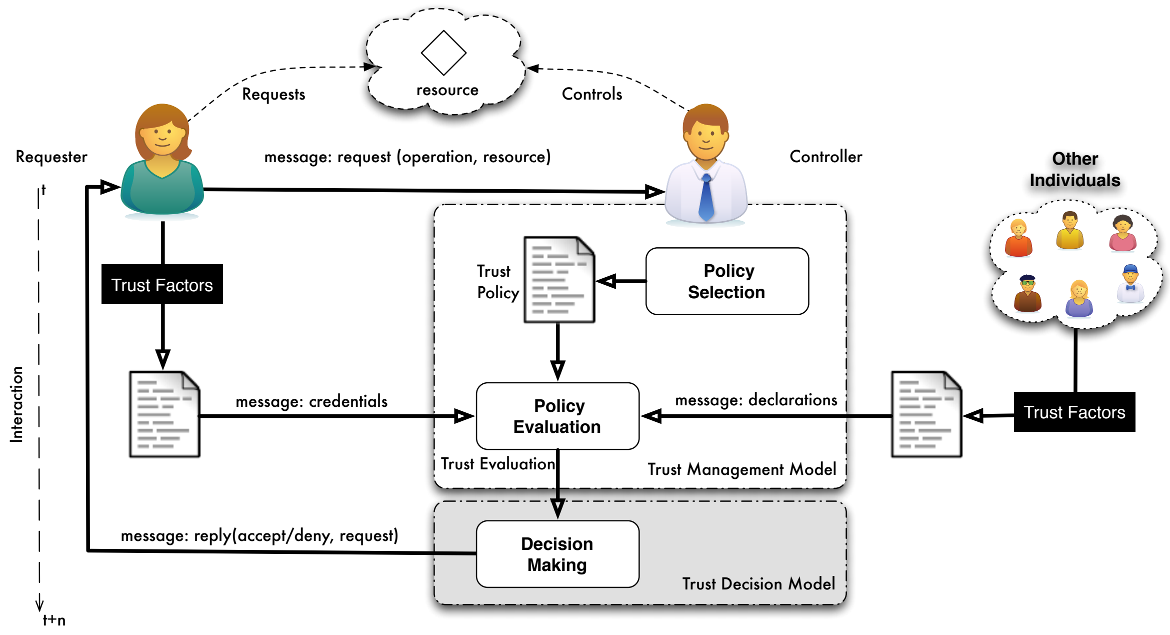


Figure 6.1 – Overview of the trust model.

To gain this trust, the controller selects a policy in which he specifies the conditions that the requester must satisfy in order to be trusted, and consequently granted access to the resource. The controller's policy can be satisfied using information that proves that:

- the requester possesses the properties about which conditions are stated in the policy, and that
- the values of the properties the requester possesses satisfy the threshold value specified by the conditions.

A policy expresses constraints on properties that the controller considers as necessary to grant its trust. So each property that is considered to be pertinent for the trust evaluation constitutes a *trust factor* and its semantic is shared by all the participants in the system. The policy indicates also to the controller what is the information he must acquire in order to evaluate the policy. This information is either provided by the requester itself (i.e. credentials) or aggregated based on other agents' testimonials (i.e. declarations). Once all required information are collected, the controller can evaluate the policy. During the policy evaluation, the controller checks whether the collected information satisfies each of the conditions stated in its policy. If so, the controller will trust the requester to perform the requested operation. Otherwise, the requester will not be trusted and its request to manipulate the resource will be denied.

This model reproduce is similar to the abstract architecture used by almost all trust management system (cf. Chapter 3). The evaluation of the trustworthiness level of an interlocutor is thus separated of the decision of trusting him/her. As explained in the introduction, our work is focused on the evaluation, letting out the decision since it is strongly connected to applicative issues. Thus, the decision model which is presented in Section 6.6 aims only at illustrating how the elements all put together work.

In the following, we build upon the above abstract model to present how trust is concretely managed within virtual communities.

6.2 Trust Factors

In human-based virtual communities, participants rely on a wide range of trust factors to evaluate trust they put in each others. Most of these factors, like proximity [Bruneel et al., 2007], feelings [Nepal et al., 2011] or moods [Dunn and Schweitzer, 2005] (cf. Chapter 2) are subjective and thus cannot be easily captured. Therefore, in our model we adopt a property-based approach (cf. Section 3.1.6), a.k.a. attribute-based, and limit ourselves to trust factors that represent individuals' properties (c.f. Section 5.4). These factors are split into *proofs* and *indicators*. *Proofs* are trust factors which evaluation requires a *credential*, while *indicators* are evaluated by *declarations*.

In the following, we first define this concept of trust factors and illustrate our definition with an example. Then in Section 6.2.4, we describe how agents use *trust information* to communicate about these factors. Trust informations are split into *credentials* and *declarations*. *Credentials* are issued by certification authorities, while *declarations* are stated by agents based on their *opinions* (i.e. direct past experience). The information an agent gathers about its interlocutor is stored in a *profile*.

6.2.1 Definition

Trust factors are the properties based on which virtual community members make their trust evaluation.

Definition 50 (Trust Factors) *We define the set $\mathcal{F}$ of trust factors as follows:*

$$\mathcal{F} \subseteq \mathcal{P}$$

where $\mathcal{P}$ is the set of properties that individuals can have (e.g. those presented in Section 5.4).

Of course the set $\mathcal{F}$ is application-specific and can evolve over time. We assume that the set $\mathcal{F}$ is built by the participants by identifying correlations between properties and trustworthy behaviours. However, the definition of these mechanisms is out of the scope of our investigations so they have not been addressed in this thesis.

Example 6.2.1 *Based on Example 5.4.1, Alice has the following properties:*

$$\begin{pmatrix} Name & Id & Competence & Experience & Recommendation & Reputation \\ Alice & F & G & G & B & G \end{pmatrix}$$

The choice of using properties as trust factors in the context of trust management is quite realistic. Indeed, using its experience, an individual can easily infer that individuals with some specific property p_i tend to misbehave if trusted in some context, while those having another property p_j happens to perform well (cf. Chapter 2). For example, one can say that individuals having diploma in medicine are more likely to perform well if trusted for making medical prescriptions. Similarly, in a medical virtual community it is totally excluded to find any correlation between the trustworthiness of an individual and the fact that he owns a car or not.

6.2.2 Proofs and Indicators

To break the "distrust"[1] deadlock during an interaction, the information that the interacting partners exchange must be acquired from a "Third Party", preferably a *"Trusted Third Party" (TTP)*, a *presumably trusted* entity that is not involved in the current interaction. As highlighted in Chapters 2 and 3, there exist in the literature two approaches for using TTP in trust management: *certification authorities (CA)* and *cross-certification (CC)* [Linn, 2000]. Both approaches rely on the realm of cryptographic techniques involving public and private keys. However, the first approach relies on the trust that exist between an individual and the organisation to which it belongs, while the second drew trust from the experience of others (cf. Section 3.2.3.1). Consequently, and depending on the nature of the TTP, the trust factors fall into two categories: *proofs* and *indicators.*

Definition 51 (Proofs and Indicators) *Proofs are trust factors that can be certified by a certification authority, while indicators are trust factors that are validated using others' statements.*

Example 6.2.2 *The driving licence is a certificate which states that the individual holding it may operate a motorised vehicle, such as a motorcycle, car, truck or a bus, on a public roadway. This certificate is issued, for instance, by the police authorities which play the role of certification authority in this situation.*

[1]An individual gains the trust of its interlocutor by satisfying this latter's policy. In order to ensure this, he must provide information that satisfies each of the constraints expressed in its policy. But at the beginning of an interaction, the interlocutors do not yet trust each others and thus, the information they will provide to each other will not be trusted as well.

6.2.3 Trust Factors Ontology

As discussed in Section 3.4.1, using CA to certify agent's properties for trust management in distributed systems is particularly suitable [Blaze et al., 1999a]. However, individuals involved in virtual communities are *heterogeneous* and so are their properties (cf. Section 5.4).

Therefore, trust management systems need to understand and reason on these properties and which ones could be used as trust factors in the system. This is the main objective of the *trust factors ontology*. The whole set of trust factors forms the trust factors ontology denoted by Δ_f as shown in Figure 6.2.

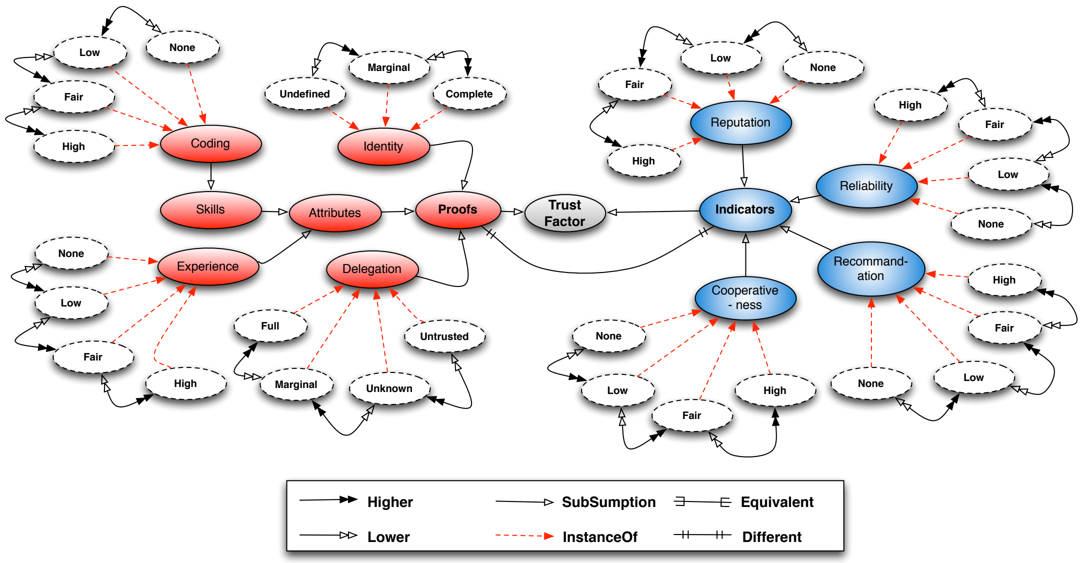


Figure 6.2 – A fragment of the trust factors ontology.

Definition 52 (Trust Factors Ontology) *The trust factors ontology Δ_f represents a formal specification of the set of all trust factors $\mathcal{F}$. $\forall f \in \mathcal{F}$, f is an ontological concept defined by:*

$$f = \langle T, A, R \rangle$$

where T is the term that corresponds to the trust factor's name (TBox), A is the domain of the trust factor (i.e. the assertions that represent the value the trust factor can take), and $R \subseteq \{\Delta.R \cup \{Higher, Lower, Different, Equivalent\}\}$ is the set of relations that elements from A can have between them (cf. Section 5.2).

The role of the trust factors ontology in our model is to define and capture essential features that are the basis for trust establishment between two agents. The root concept is a generic trust factor concept. It is further divided into two sub-types (*proofs* and *indicators*) that represent the trust factors categories introduced in the previous section. The relations *Higher*

($>$) ,*Lower* ($<$), *Different* ($\neq$) and *Equivalent* ($=$) define an order over the values that each trust factor can have.

Example 6.2.3 *For instance, the trust factor representing the diploma degree can be defined as follows:*

$$f = \langle StudiesDegree, \{VG, G, F, B, VB\}, \{VG > G, G > F, f > B, B > VB\}\rangle$$

6.2.4 Trust Information

If the aforementioned trust factors are essential for trust management, the means that agents use to communicate about them are, in reality, more important. Agent uses what we call *trust information* to exchange information about the values of the properties they possess.

6.2.4.1 Types of Trust Information

Trust information falls into two types: *credentials* and *declarations*. Credentials (also called *Electronic Credentials*) constitute the counterpart of the paper credential we use in the real world.

Definition 53 (Credential) *Credentials represent digital certificates that are signed by certification authorities in order to establish properties for their holder.*

For example, `X.509` is one of the most popular credential formats used, nowadays, in the Internet. A credential remains valid until its expiration or its revocation.

Definition 54 (Declaration) *Declarations are testimonial statements issued by individuals to exchange information about other individuals' properties.*

When a member wants to validate the properties of its interlocutor, he broadcasts a request wherein he asks his fellows in the community to provide him with the information that confirms or not that property. Likewise a declaration remains valid until a fresher information is received.

6.2.4.2 Representing Trust Information

The particular form in which trust information are represented is not critical in our model. However, for uniformity, we adopt a generic representation form that we will use to illustrate our approach.

Definition 55 (Trust Information) *Each trust information is defined by a quadruplet as follows:*

$$ti = \langle t, i, s, f, v\rangle$$

where $t \in \{credential, declaration\}$ is the trust information type, i is the individual issuing the trust information, s is the individual concerned by the information, f is the trust factor the information is about, and $v \in f.A$ is the value that i associates to s for the trust factor f.

Example 6.2.4 (Trust Information) *With respect to our running example, the trust information* $\langle declaration, Carl, Alice, reputation, High\rangle$ *is a* declaration *in which* Carl *testifies that* Alice *has a good reputation.*

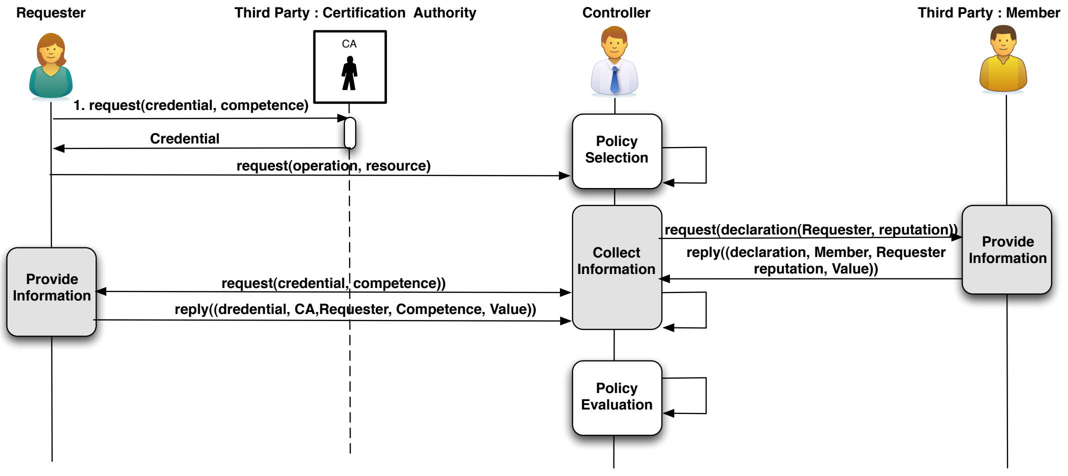

Requester
Third Party : Certification Authority
CA
Controller
Third Party : Member
1. request(credential, competence)
Credential
request(operation, resource)
Policy Selection
request(declaration(Requester, reputation))
reply((declaration, Member, Requester reputation, Value))
Collect Information
Provide Information
Provide Information
request(credential, competence))
reply((dredential, CA,Requester, Competence, Value))
Policy Evaluation

Figure 6.3 – The process of trust information acquisition.

The process that controllers use to acquire these trust information is relatively straightforward. It is illustrated in Figure 6.3. When a controller receives a request, it selects the policy and identifies whether the conditions stated in the policy requires to proofs or indicators.

If the condition requires a proof, the controller requests from the requester the credential that satisfies the condition. Once the controller received all required credentials, he checks each credential is still valid or not (cf. Section 6.2.4).

Assumption 6.2.1 (Trust Information Validation) *We assume that any member of the community is able to convert any "valid" credential or declaration to the representation form defined above. With this assumption we ignore the mechanisms of encryption and decryption that are inherent to this task. For instance, the validation of credentials can be achieved using the mechanisms presented in [Lee, 2008].*

Similarly, for each indicator the controller asks the other participants (the other members are the trusted third party) to provide him with testimonials about the requester. The received testimonials are grouped and aggregated to form unique value that the controller use to evaluate its policy. In the following, we describe how individuals are able to issue testimonials (cf. Section 6.2.4.3)

6.2.4.3 Opinions

In previous section, we presented how declarations are used as third party testimonials that participants use to evaluate their policies.In this section, we will show how each participant builds these testimonials.

Declarations are built based on *opinions* that participants maintains about each others. These opinions are made based on the individual past experience and individuals' observations as well. Observations are rudimentary opinions such as the value of a property (e.g. the participant age), while experiences are computed (e.g. reputation).

Definition 56 (Opinions) *Each individual maintains a set Θ of information about individuals it interacted with/observed previously. Each opinion $\theta \in \Theta$ is defined by:*

$$\theta = \langle i, f, v \rangle \tag{6.1}$$

where i is the subject of the opinion (the individual the opinion is about), f is the trust factor mentioned in the opinion, and v is the value the individual associated to the subject for the trust factor f. For observation opinions, only the newest observation is stored in Θ, while for experience, the new experience is aggregated with the old ones to compute the updated value. Therefore, opinions are not time-stamped.

Example 6.2.5 *For instance, Carle, Dave and Elvis can have the following opinions about Alice :*

- *Carl :* $\langle Alice, reputation, 80\% \rangle$
- *Dave :* $\langle Alice, reputation, 65\% \rangle$
- *Elvis :* $\langle Alice, reputation, 70\% \rangle$

Once the controller receives all the declarations, it aggregates them based on the trust factor they certify. For instance, if the controller requested the reputation of a requester, each participant extracts from its opinions the one that relates to reputation and that concerns the requester. Then this opinion is formatted as a declaration and sent to the controller. Once the controller receives all declarations for the reputation, it uses its own *aggregation function* to compute a reputation value.

We assume that each individual is capable to build the profile of its interlocutor. Once all requested information received[2] and their content validated, the controller is able to build the profile of the requester.

6.2.4.4 Profiles

Profiles are data structures in which an individual keeps track of all information he gathers and that concerns his interlocutor.

[2]We suppose that after a certain time period, the trust information request is considered as "unsuccessful" and the individual proceeds with the policy evaluation without the requested information

Definition 57 (Profile) *A profile ($\psi^{a_j} \in a_i.\Psi$) built by an individual a_i about an other individual a_j is defined by the triplet:*

$$\psi^{a_j} = \langle r, a_j, S \rangle$$

where r is a request identifier and S is a set of statements wherein each statement s_i is represented as follows:

$$S.s_i = \langle f.T, f.A \rangle$$

such as $f.T$ is a trust factor type and $f.A$ is the value of the trust factor (cf. Section 6.2.3).

The set of profiles an individual builds are stored in its *profile repository* (Ψ).

Example 6.2.6

$$Credential(CA1, Alice, age, 30) \tag{6.2}$$

$$Declaration(Carl, Alice, reputation, 80\%) \tag{6.3}$$

$$Declaration(Dave, Alice, reputation, 65\%) \tag{6.4}$$

$$Declaration(Elvis, Alice, reputation, 70\%) \tag{6.5}$$

Based on the trust information (6.2), (6.3), (6.4) and (6.5), Bob can build the following profile in which it stores information it collected and that concern Alice.

$$\langle req1304, Alice, \{\langle age, 30\rangle, \langle reputation, 72\%\rangle\}\rangle$$

6.3 Trust Policies

As defined in Chapter 3 (cf. Section 6.3), policies are statements that specify under which conditions an individual can be trusted for a specific issue. In our thesis, we represent policies as a collection of conditions, called *trust criteria*.

Definition 58 (Policy) *A policy π is defined by:*

$$\pi_{Issuer}^{Pattern} = \{tc_1, tc_2, \ldots, tc_n\}$$

where $Issuer$ is the individual which issues the policy, $Pattern$ is the Trust Pattern to which the policy applies, and $\{tc_1, \ldots, tc_n\}$ is a set of trust criteria.

Policies are used to state what properties an interlocutor must possess in order to be considered trustworthy. However, we defined in Chapter 2 trust as a decision that involve a particular issue (cf. Definition 1). To that aim, we use the concept of *trust pattern*.

Definition 59 (Trust Pattern) *Trust patterns (or patterns, for short) are used to specify the issue of the decision a policy can be used for. They are defined as a pair:*

$$\langle action, target \rangle \tag{6.6}$$

Where $action$ represent the action the requester requested to perform, and $target$ defines the entity (i.e. resource, agent, community) on which the action will be performed.

Based on the *target* field of the trust pattern, two types of policies can be defined; *specific* or *general*. Specific policies are policies in which the pattern refers to a particular resource, community or agent (via their identifiers). In contrast, *general* policies are specified for a group of resources (via their types) or a group of individuals (via their roles).

Example 6.3.1 (Trust Pattern) *For instance, the target* $\langle \texttt{read}, text\rangle$ *is used to specify a policy that manages* `read` *requests on text documents. This pattern is different from this on* $\langle \texttt{read},\, r_1\rangle$ *which is used in the policy that accepts or not requests about reading the file* r_1*.*

We believe that most of the policies that are used in virtual communities are/should be general policies. Indeed, given the intense pace of resources creation, communities building, and agents turnover. We think that stating a specific policy each time a new resources is created, a new community is built or a new agent is admitted seems simply unrealistic. Nevertheless, the policy formalism we used admits to express both even if policies we will use in the remainder of this thesis are *general policies.*

6.3.1 Trust Criteria

Trust criteria, denoted *tc*, are the building blocks of policies. They are atomic conditions used to express threshold values over *trust factors.* With atomic we mean that these constraints cannot be further divided into other smaller conditions.

Definition 60 (Trust Criteria) *Each trust criteria tc is defined by :*

$$tc = \langle f, op, v, w, t\rangle \tag{6.7}$$

where

- f *is a trust factor such as* $f \in \Delta_f$*,*
- $op \in \{=, >, <, \leq, \geq, \neq\}$ *is an operator,*
- v *is the criterion threshold value where* $v \in f.A$*,*
- $w \in \mathbb{Z}$ *is the criterion weight,*
- $t \in \{m, o\}$ *is the criterion type; "m" means that the trust criteria is mandatory, while "o" means that it is optional.*

Trust factors specify what propertie values an interlocutor must satisfy in order to be considered as trustworthy. Operators are used to define the nature of the conditions that the individual wants to express. Basically, an individual can represent six types of conditions:

Upper-bound conditions (UBC) are expressed using the operator "$\geq$". They are used to specify that acceptable values for that specific trust factors must be superior or equivalent to the threshold value stated in the trust criterion.

Strict upper-bound conditions (SUBC) are expressed using the operator "$>$". They are used to specify that the values must be strictly superior to the threshold value fixed in the trust criterion.

Lower-bound conditions (LBC) are expressed using the operator "$\leq$". They are used to specify that acceptable values for that specific trust factors must be inferior or equivalent to the threshold value stated in the trust criterion.

Strict lower-bound conditions (SLBC) are expressed using the operator "$<$". They are used to specify that the values must be strictly inferior to the threshold value fixed in the trust criterion.

Exception conditions (EPC) are expressed through the operator "$=$". They are used to specify that the only acceptable value is the value fixed by the threshold in the trust criterion.

Exclusion conditions (ELC) are expressed by the operator "$\neq$". They are used to specify that all values are acceptable except the value stated in the trust criterion.

Here, "superior" and "inferior" should not be understood only from a pure mathematical semantics. Their meaning depends on the order relation that is stated (or not) in the trust factors ontology.

The relationships between the concepts we saw previously (i.e. properties, trust factors and trust information) and the concept of *trust criteria* that we introduced in this section are illustrated in Figure 6.4.

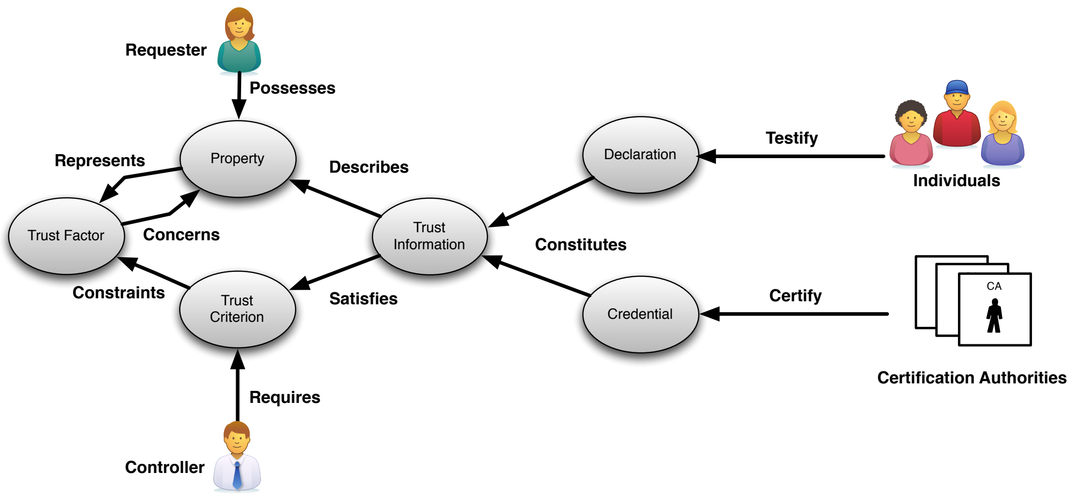


Figure 6.4 – The correlation between the concepts of trust factors, trust criteria and trust information.

Properties are individual's characterising features. When a correlation exists between a property and the trustworthiness of individuals that possess it, the property is considered as a trust factor. Using trust factors, individuals can express trust criteria (i.e. constraints or conditions) to specify from which value the property holder can be considered as trustworthy.

Subsequently, trust criteria can be satisfied (or not) using trust information. These trust information can be declarations or credentials. Declarations constitute other members testimonials that are directly provided to the controller, while credentials are certificates delivered by certification authorities to the controller which decides to disclose them to the requester or not.

Example 6.3.2 *Let us take the example of the following trust criterion:*

$$tc = \langle age, Op, 18, 2, m \rangle \tag{6.8}$$

Now let us see how the type of the operator can affect the nature of the condition stated by the trust criterion:

- *$Op =$ "$\geq$" signifies that only adult individuals satisfy the trust criteria (UBC),*
- *$Op =$ "$<$" signifies that only minors satisfy the trust criteria (SLBC),*
- *$Op =$ "$=$" signifies that only individuals aged 18 satisfy the criteria (EPC),*
- *$Op =$ "$\neq$" signifies all individuals except those that are aged 18 satisfy the criteria (ELC).*

6.4 Trust Mechanisms

A trust mechanism refers to the scheme used by the controller to derive trust evaluations based on its policies and the information it collected (i.e. credentials and declarations). To that aim, we specified a policy evaluation function that computes to which degree the trust information collected satisfy the policy of the controller. Worth noting that this function does not take any trust decision, but instead computes a trust measure that is later used by the trust decision model to derive a trust decision (cf. Section 6.6). This trust evaluation represent the amount of trust that the *controller* accords to the *requester*.

Definition 61 (Policy Evaluation Function) *The policy evaluation function is defined as follows:*

$$\mathcal{E} : a.\Pi \times a.\Psi \rightarrow [0, 1]$$

The above evaluation function is used by an individual a to compute the extent to which the information he collected about his interlocutor (i.e. the profile $a.\Psi$) satisfies the trust criteria stated in its policy (i.e. a.Π). To that aim, the function $\mathcal{E}$ maps a policy and the corresponding profile to the weighted sum that corresponds to the policy evaluation.

Concretely, let π_a^x be the policy used by the individual a to handle requests of the pattern x (e.g. $x = \langle read, text\rangle$). Now let us consider the request r initiated by b. After receiving the request, the controller a collects the trust information required by its policy and builds the requester's profile $a.\psi^b$.

The evaluation of π_a^x is achieved by applying the function $\mathcal{E}(\pi_a^x, \psi^b)$ as sketched by the formula 6.9 below, knowing that:

$$\pi_a^x = \{\langle f_1, op_1, v_1, w_1, t_1\rangle, \ldots, \langle f_n, op_n, v_n, w_n, t_n\rangle\}$$

is the policy under evaluation, and

$$\psi^b = \langle q, b, \{\langle f_1, v'_1\rangle, \ldots, \langle f_m, v'_m\rangle\}\rangle$$

is the profile that a built about b. The policy evaluation is then performed as follows:

$$\mathcal{E}(\pi_a^x, \psi^b) = \begin{cases} \frac{\sum_{i=1,j=1}^{n} E(\langle f_i, op_i, v_i, w_i, t_i\rangle, \langle f_j, v'_j\rangle)}{\sum_{i=1}^{n} w_i} \\ \text{where } \langle f_i, op_i, v_i, w_i, t_i\rangle \in \pi_a^x, \langle f_j, v'_j\rangle \in \psi^b \text{ and } f_i = f_j \\ 0 \text{ if } \exists\ tc_i \in \pi_a^x /\ E(\langle f_i, op_i, v_i, w_i, t_i\rangle, \langle f_j, v'_j\rangle) = 0 \wedge t_i = m \end{cases} \tag{6.9}$$

The function $E(\langle f_i, op_i, v_i, w_i, t_i\rangle, \langle f_i, v'_i\rangle)$ valued in $[0, w_{f_i}]$ performs the evaluation of a trust criterion tc_i with respect to a trust information ti_j. The correspondence between the trust criterion and the trust information is done based on the common trust factor they use. For instance, a trust criterion that constraints the age of the interlocutor is evaluated with the trust information that proves or testifies such trust factor. So for each trust criterion, the function $\mathcal{E}$ looks for the corresponding trust information and evaluates whether this information satisfies the criterion or not. If it satisfies it, the computed value is 1 which is further multiplied by the weight of the criterion w_i, otherwise the computed value is 0. A trust criterion is computed to zero either because the trust information failed satisfying the criterion, or because the appropriate trust information has not been acquired. Further, the results of the evaluation of each trust criteria are summed and divided by the sum of all the weights. So the result represents the mean of the weighted sum and constitutes the satisfaction level of the controller policy with respect to the requester profile. Finally, if one of the criteria that are not satisfied is mandatory, the evaluation of the policy fails and returns 0.

Example 6.4.1 *Let us consider the following policy* [3]
$\pi_{Bob}^{\langle join, community\rangle} = \{\langle age, \leq, 33, 2, m\rangle, \langle age, \geq, 18, 2, m\rangle, \langle identity, >, complete, 1, o\rangle,$
$\langle reputation, >, 70\%, 1, o\rangle\}$
and let

$$Bob.\psi^{Alice} = \langle q, Alice, \{\langle age, 22\rangle, \langle identity, marginal\rangle, \langle reputation, 65\%\rangle\rangle$$

be the profile Bob collected about Alice who wants to access the "juniors" community he created in the Eurêka platform. The evaluation of the policy proceeds as follows:

[3] With $identity.A = \{ultimate > complete > marginal > unknown > untrusted\}$ (cf. Section 3.2.3.1).

- $22 \leq 33 \implies E(\langle age, \leq, 33, 2, m\rangle, \langle age, 22\rangle) = 2$
- $22 \geq 18 \implies E(\langle age, \geq, 18, 2, m\rangle, \langle age, 22\rangle) = 2$
- $marginal < complete \implies E(\langle identity, >, marginal, 1, o\rangle, \langle identity, marginal\rangle) = 0$
- $65\% < 70\% marginal \implies E(\langle reputation, >, 70\%, 1, o\rangle, \langle reputation, 60\rangle) = 0$

The above example is illustrated in Figure 6.5

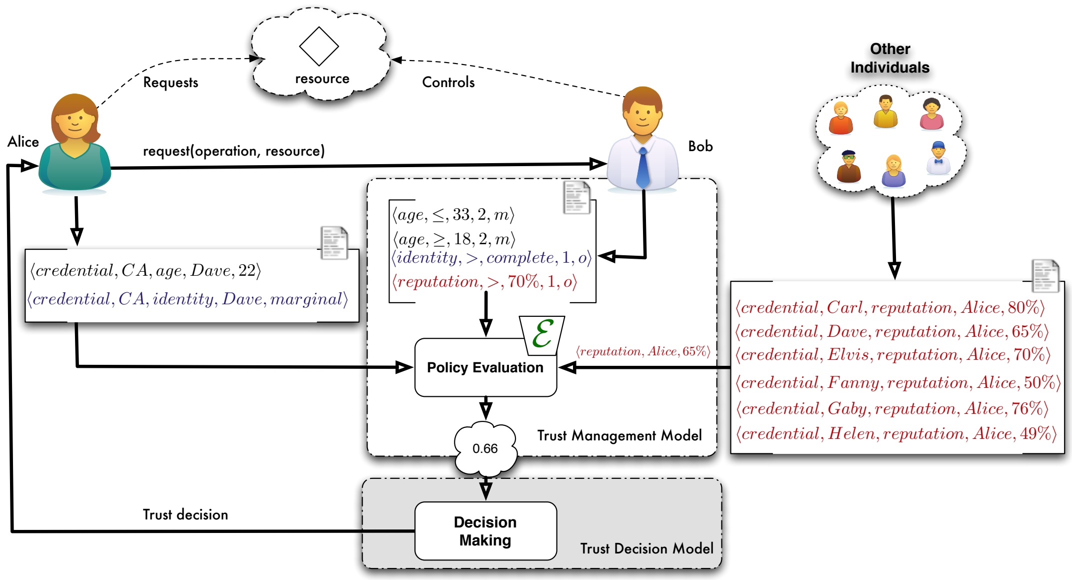

resource
Requests
Controls
Other Individuals
Alice
request(operation, resource)
Bob
$\langle age, \leq, 33, 2, m\rangle$
$\langle age, \geq, 18, 2, m\rangle$
$\langle identity, >, complete, 1, o\rangle$
$\langle reputation, >, 70\%, 1, o\rangle$
$\langle credential, CA, age, Dave, 22\rangle$
$\langle credential, CA, identity, Dave, marginal\rangle$
$\mathcal{E}$
$\langle reputation, Alice, 65\%\rangle$
$\langle credential, Carl, reputation, Alice, 80\%\rangle$
$\langle credential, Dave, reputation, Alice, 65\%\rangle$
$\langle credential, Elvis, reputation, Alice, 70\%\rangle$
$\langle credential, Fanny, reputation, Alice, 50\%\rangle$
$\langle credential, Gaby, reputation, Alice, 76\%\rangle$
$\langle credential, Helen, reputation, Alice, 49\%\rangle$
Policy Evaluation
0.66
Trust Management Model
Trust decision
Decision Making
Trust Decision Model

Figure 6.5 – Illustration of the policy evaluation process

In this example, the comparison between the profile values and the threshold values in the trust criteria is performed based on the relations that exist between these values in the trust factors ontology. For instance, it is not because 33 is greater than 22 that the statement $22 \leq 33$ is computed to true. This is due to the fact that, there is a "*Higher*" relationship from 33 to 22 stated in $\Delta_f.R$ of the trust factors ontology. This mechanism allows us to deal with all kind of values (intervals, boolean, ...). The evaluation of the above policy with respect to the information contained in the profile would be as follows $l = (2 + 2 + 0 + 0)/6 = 0.66$ These results corresponds to the satisfaction level of the policy. Now, it remains to the *trust decision model* to make a trust decision based on this evaluation.

6.5 Trust Decision Making

As stated before, the process of evaluating trust is distinct from the one of making the trust decision. In our thesis we only focus one the first one while the second one is left to the final user. However, in order to be able to evaluate our model, we make use in this section of a simple decision model. This model is encapsulated in the *trust decision function* defined below.

Definition 62 (Trust Decision Function) *The trust decision function is defined by:*

$$\mathcal{W} : [0,1] \rightarrow \{permit, deny\} \tag{6.10}$$

How this trust decision function really works is not relevant in our investigations. The only assumption we make is that this function makes use of the trust evaluation computed before and that this value is compared with respect to a decision threshold fixed by the individual (or agent) making the trust decision.

Example 6.5.1 *Let us consider the case of Example 6.4.1, and let us suppose that Bob, the individual taking the decision, decided to accept all requests from individuals that satisfy his policies by at least* 60%*). In this situation, the trust decision function* $\mathcal{W}$ *will compute the decision permit for the request* q *which was evaluated in that example to* 66%*.*

When a request is accepted, the individual that initiates the request is permitted to perform the operation he requested on the resources his is interested about. The acceptance of a request is concretely materialised by the establishment of an *authorisation* (i.e. credential) in which the resource controller testifies that the requester is trusted to perform what he is authorised for. Once he gets this authorisation, the requester will be allowed by the resource to perform the operation he is trusted for.

6.6 Building Trust Experiences

When a controller trusts a requester, the former exposes himself to the betrayal of its interlocutor. Thus, after each interaction, controllers evaluate the behaviour of their interlocutors with respect to the trust decision they have made. To that aim, each individual makes use of its personal *outcome evaluation function* to assess whether the interaction was satisfactory or not.

Definition 63 (Outcome Evaluation Function (OEF)) *The outcome evaluation function, denoted by* Ξ*, is defined by:*

$$\Xi : \{permit, deny\} \rightarrow \{-1, 0, +1\}$$

-1 means that the outcome was negative, while $+1$ is used when the outcome is positive. The neutral outcome 0 means that the interaction did affect neither negatively nor positively the individuals involved in it.

Based on the outcome evaluation function Ξ, each individual maintains an *experience set* (Γ) in which its stores its individual experiences. Experiences represent traces of interaction undertaken by the individual.

Definition 64 (Experience) *Each experience e is defined by :*

$$e = \langle i, s, l, d, o, \pi, \psi \rangle \tag{6.11}$$

where

- *i is the interaction identifier,*
- *s is the subject (i.e. interlocutor) which whom the individual is interacting,*
- *l is the trust level computed by the policy evaluation function $\mathcal{E}$,*
- *d is the decision derived by the trust decision function $\mathcal{W}$,*
- *o is the outcome computed by the outcome evaluation function Ξ,*
- *π is the policy used during the interaction,*
- *ψ are the profile it used to evaluate the policy.*

We will see in the next chapter, in Section 7.2.1, how these experiences are used to improve the quality and pertinence of trust policies.

6.7 The Trust Management Cycle: From Trust Factors to Trust

Now that all the elements that comprise our trust model has been introduced, we are in position to present how we put all these elements together to define the architecture of the *trust management system* that every agent of our communities relies on when making its trust decisions.

As depicted in Figure 6.6, first (1), the user of the TMS (e.g. the controller) initialises it by specifying the policies that the TMS will use to handle each type of request (cf. Section 6.3). Once the TMS initialised, it is now able to handle the requests that the controller receives (2). The request management is then responsible of extracting the trust pattern from the request and forwards it to the policy management. Based on this pattern, the policy management extracts from the policies repository the policy that best handles the received request (4). Once this policy selected (5), the policy management module triggers the policy evaluation function (cf. Section 3.3). The policy evaluation function triggers the trust information collection (6) which, depending on the nature of the trust criteria (i.e. proofs or indicators) specified in the policy, requests the appropriate trust information (i.e. credentials or declarations) (7). Now, the policy evaluation function is able to use the profile built by the trust information collector (8) with the policies selected by the policies management. The result of the evaluation is then passed to the trust decision function that computes the final trust decision (10). If the requester has been trusted, an authorisation is established. In (11), the outcome evaluation function tries

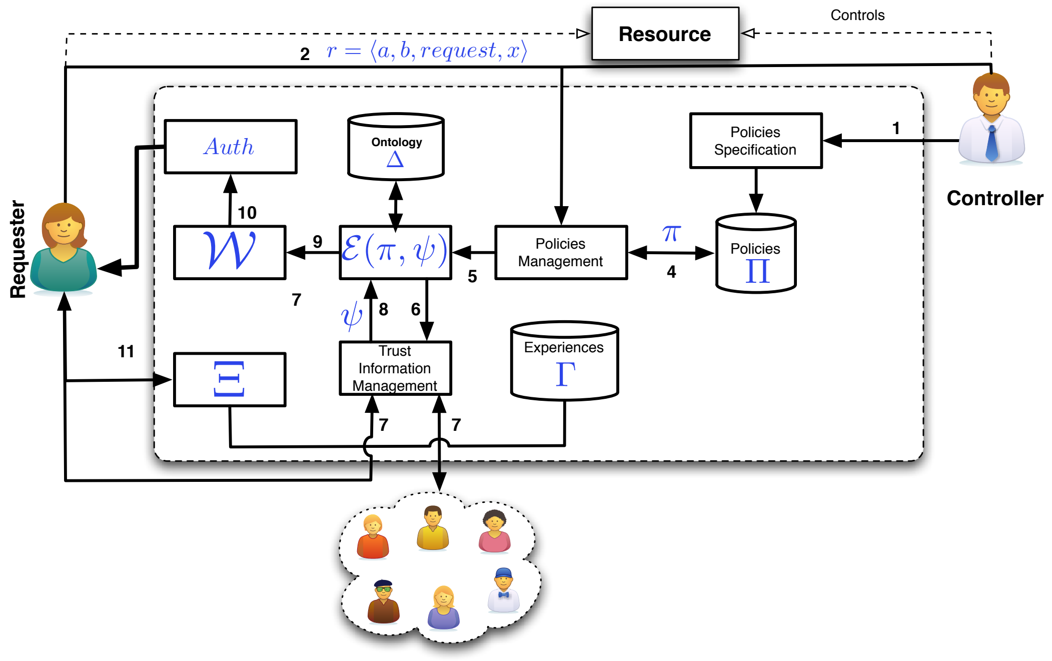


Figure 6.6 – Abstract representation of our trust model.

to assess the outcome of the interaction that has been accepted by the TMS and updates the experiences repository in consequence (12).

In the next chapter we present, in more details, the process behind the *policies management phase* of the model above. That is actually where policies are built, maintained and, most importantly, adapted. It is this adaptation mechanism that represents our solution to the limits of trust management systems highlighted in Chapter 3.

6.8 Bridging the trust model and the virtual community model

We present in this last section how the concepts we used to describe the multi-agent-based virtual community framework (cf. Chapter 5) have been extended to integrate the concepts related to the trust management that we introduced in the previous sections.

Again, we will rely on the VOWELS approach to describe what should be added in each of its dimensions to bridge the two models. But first, we start by extending the definition of the ontology to integrate trust concepts.

6.8.1 Ontology

The trust factors ontology $\mathcal{T}$ defined in Section 6.2.3 is integrated to the domain ontology Δ defined in Chapter 5. The relatively straightforward mapping between the definition of $\mathcal{T}$ and the definition of Δ is as follows:

$$\Delta_f \subset \Delta \implies \begin{cases} \Delta_f.T \subset \Delta.T \\ \Delta_f.A \subset \Delta.A \\ \Delta_f.R \subset \Delta.R \end{cases}$$

6.8.2 Resources

At the resource level, we extend the definition of resources given in Section 5.3 to endow each resource with an access control mechanism that enables it to protect its content from not permitted operations (cf. Section 3.1). This mechanism is symbolised by the authorisation function *Auth*.

Definition 65 (Authorization function) *The authorisation function Auth is defined by:*

$$Auth : A \rightarrow O$$

The function *Auth* maps agents to the operation they are permitted to perform. Consequently, resources are now defined as follows:

$$\forall r \in \mathcal{R}, r = \langle \varepsilon, \theta, P, Auth \rangle$$

6.8.3 Agents

The definition of agents is extended as follows:

$$\forall a \in \mathcal{A},\ a = \langle \varepsilon, G, \Lambda, L, R, C, P, \Theta, \Pi, \Psi, \Gamma, \mathcal{E}, \mathcal{W}, \Xi \rangle$$

where $\varepsilon, G, \Lambda, LR, C$ and P are, respectively, the agent identifier, its goals, its plans, its resources, its communities and its properties as defined in Section 5.4. To these elements, we add the following concepts introduced in this chapter:

- Π which is the set of policies that the agent a uses in its trust decisions,
- Θ is a set of opinions it maintains about other agents of the community,
- Ψ is the set of profiles the agent gathered during its interactions,
- Γ is the experience that the agent built over its interactions,
- $\mathcal{E}$ is the policy evaluation function,
- $\mathcal{W}$ is the trust decision function,
- Ξ is the outcome evaluation function.

6.8.4 Interaction

In terms of interactions, we have made two important extensions to the interaction language we have introduced in Section 5.6. First, we add two new performatives that agents use to communicate about trust decisions: `permit` and `deny`.

In addition, we extended the agents communication language with a new protocol, called `TrustNegotiation`. This protocol is used by the agents in the position of controller to request credentials from the requesters.

Thus, the extension we have made to the interaction dimension are stated as follows:

- $Perf = Perf \cup \{\texttt{permit}, \texttt{deny}\}$
- $Prot = Prot \cup \{\texttt{TrustNegotiation}\}$

The $TrustNegotiation$ protocol will be defined in the next Chapter (cf. Section 7.4.2).

6.8.5 Organization

At the organization level, we need to extend the structural, functional, and normative specifications we presented in Section 5.5. The extension of the structural specification is justified by the need for agent that endorse the role of *certification authority* and to express delegation between agent. And the extension of the normative specification is mainly motivated by the need for collective policies.

6.8.5.1 Structural Specification Extension

In the structural specification, we added three new roles to the specification of the community as depicted in Figure 6.7.

The first added role is the *controller* role. This role is used to implement the delegation of rights between agents. A *provider* or a community *owner* can delegate part of their right to another agents. Consequently, this agent can handle requests about this resources/community and issues permissions to other agents. Moreover, the agents adopting the role of *authority* are responsible of issuing, validating and revoking *credentials* for other members. Thus they act as *certification authorities*. Nevertheless, we assume that this role is incompatible with the role of *consumer*. So an agent playing the role of *authority* cannot use the credentials he issued when he is playing the role of *provider* and *controller* as well. Therefore, there is a compatibility link between these roles. Consequently, there should be at least two agents playing the role of *authority*. So an agent can decide which agents he trusts as an authority and which ones he does not. However, this issue is not addressed in this thesis. Finally, the last role is an abstract role (i.e. a role that cannot be adopted by any agent) that we used to structure the *provider* and the *controller* roles which overlaps in the functional specification as we will see in the next section.

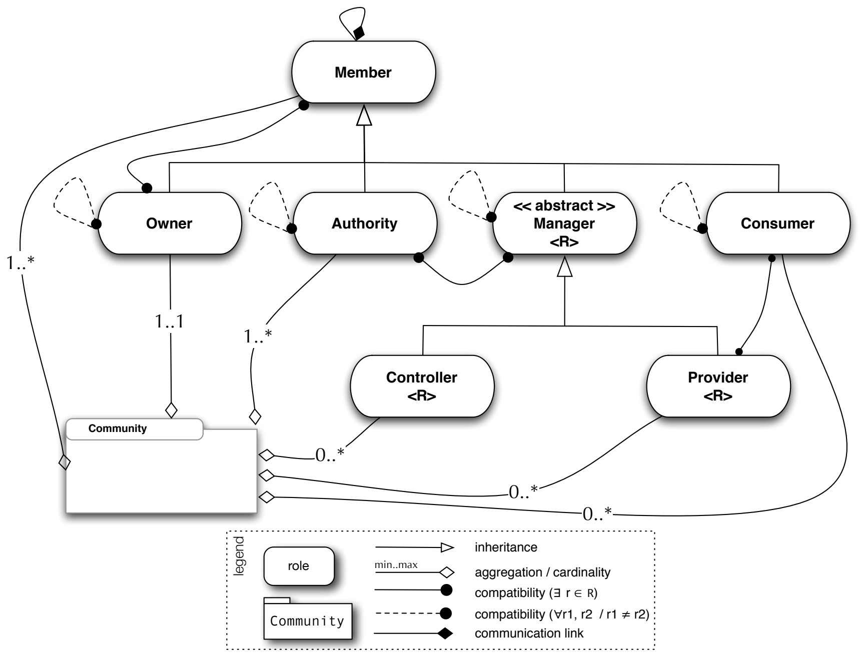


Figure 6.7 – Extended structural specification of virtual communities.

6.8.5.2 Functional Specification Extension

As we added three new roles, we need to update the functional specification of the virtual community to add the goals and missions that belong to each of the three roles. The extended functional specification is depicted in Figure 6.8 below.

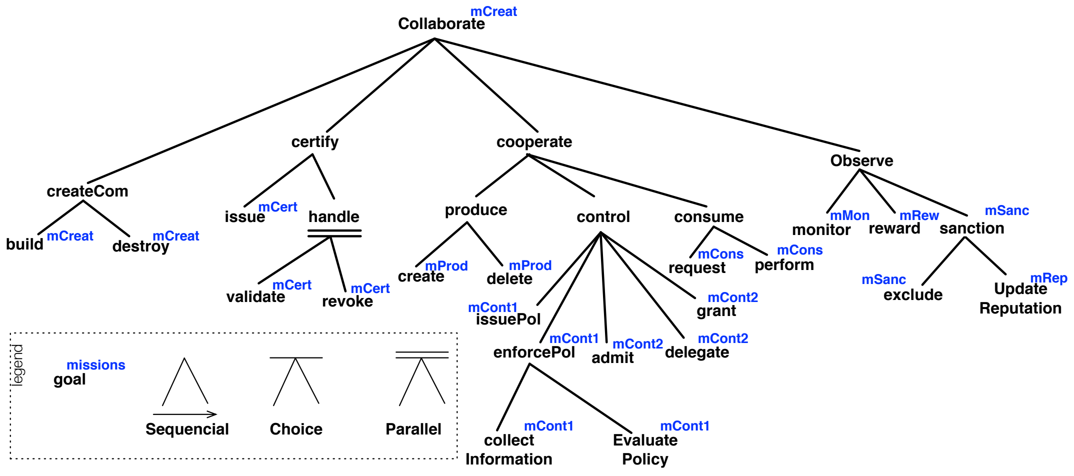


Figure 6.8 – Extended functional specification of a virtual community.

As illustrated in Figure 6.8, we extend the functional specification with the $certify$ goal which is further divided into *issue*, *validate* and *revoke* credentials. We add also a goal *monitor* by means of which agents are able to monitor each other's behaviour and sanction malevolent agents (e.g. reputation loss, exclusion). We will present in the next section the norms for which an agent can be sanctioned. Finally, we added a new goal *control* by means of which agents are responsible of answering requests (e.g. join, access control and delegation). To that aim, they need to specify policies, enforce these policies and make decisions (admit, delegate or grant).

Consequently, the new functional specification is defined as follows:

$$FS = \langle G, M, S \rangle$$

Where:

- $G = \{createCom, build, manage, admit, exclude, destroy, certify, issue, handle, validate, revoke, cooperate, produce, create, delete, control, issuePol, enforcePol, delegate, grant, consume, request, perform, monitor, reward, sanction\}$ is the set of community goals identifiers,
- $M = \{mCreat, mCert, mProd, mCont, mCons, mMon\}$ is the set of community missions identifiers,
- $S = \{ComCrSch, \}$ is the set of scheme specification of the community.

6.8.5.3 Normative Specification Extension

The extension of the normative specification is twofold. First, we added the norms by means of which we match each role with the related missions as presented in Table 6.1.

$n.\varepsilon$	condition	role	type	mission	TTF
n_1	—	*owner*	*permission*	$mCreat$	—
n_3	—	*provider*	*obligation*	$mProd$	—
n_4	fulfilled(n_3)	*controller*	*permission*	$mCont1$	—
n_5	fulfilled(n_3)	*controller*	*obligation*	$mCont2$	—
n_6	notfulfilled(n_5)	*controller*	$forbidding$	$mCont1$	—
n_7	fulfilled(n_3)	*consumer*	*permission*	$mCons$	—
n_8	notfulfilled(n_i)	*member*	*permission*	*mSac*	—
n_9	fulfilled(n_3)	*member*	*obligation*	*mRew*	—

Table 6.1 – Extended normative specification of a virtual communitiy.

Second, we enriched the definition of community with a set of policies Π that the community members must use in their trust decisions alongside with their individual policies. As a result, policies fall into two categories: *individual policies* and *collective policies*.

Individual policies are the policies that agents individually issue and use in their trust decision.

Collective policies are policies that are issued, by consensus, by all members of the community. These policy are then used by everyone during its trust evaluation.

Example 6.8.1 *To illustrate the need for collective policies let us consider our running example described in Section 5.0.1). Suppose that Alice, Bob and Carol are three members that joined together and created a community called ABC. Any trust decision made by a community member may impact the other members, making their agreement a prerequisite when taking such decisions. For instance, if Alice decides to make public the source code of the application she is involved in, this can affect the whole group.*

In the above example, asking each member his opinion when making decisions that can affect all the group is not realistic. Besides being a bottleneck strategy, in such approach no one can guarantee that all members are online when the decision has to be made. A more credible approach would be to make the community members agree on a policy based on which they can make trust decisions. Accordingly, the opinion of each member of the community has to be taken into consideration when issuing such *collective policy*. The consideration of each member's *individual policy* when building such a collective policy is paramount for the community cohesion and the coherence of the decision undertaken within it.

Definition 66 (Collective Policy) *We use policies to represent the constraints imposed by the members of a community. So each community c is associated with a set of collective policies that agents from $c.A$ are* **obliged** *to use when making decisions about common resources ($\forall r \in c.R$).*

Assumption 6.8.1 (Compliance Norm) *From the Definition 66, we advocate that there should be a norm that imposes to each member of the community the use of the appropriate collective policy when making decisions about shared resources. Consequently, every agent that trusts agents which do not satisfy the collective policy violates the compliance norm.*

The problem that can arise using norms such the one presented in Assumption 6.8.1 is the problem of compliance checking. In other words, how can we guarantee that the agents of the community will respect the norm and use the collective policies during their interactions. Indeed, virtual communities are decentralised environments wherein no entity can play the role of central authority and verify whether the imposed norms are respected or not. Therefore, the only way to perform this task is to rely on a *social control* approach [Falcone and Castelfranchi, 2001, Jøsang et al., 2007]. For that, we use a norm that obliges agents detecting norms violation to sanction the agents committing the violation.

Assumption 6.8.2 (Compliance Control Norm) *We propose a norm that obliges the community members to* **monitor** *and* **sanction** *all agents making decisions against the collective policy.*

The questions of “how collective polices are generated” and “how they are used alongside with individual policies” will be discussed later on in Chapter 7. All we need to know for the moment is that communities are endowed with collective policies. Consequently, the definition of a community become as follows:

$$\forall c \in \mathcal{C},\ c = \langle \varepsilon, OS, A, plays, commits, R, \Pi \rangle$$

where $\varepsilon, OS, A, plays, commits$ and R are the concepts defined in Definition 28, while Π is a set of policies that agents use to protect the resources they have in common and their personal resources as well.

6.9 Conclusion

In this chapter, we specified a trust management based on which members of the communities presented in Chapter 5 evaluate the trust they are willing to put in each others.

The most novel aspect of this model lies in its *expressive*, *flexible* and *semantic* policy language that can express both individual and collective policies. *Expressive* as the *conditions* the conditions used to express trust conditions (i.e. trust criteria) make use of a wide range of trust factors (i.e. proofs and indicators), *flexible* as the evaluation of these policies is inspired from *weighted-logics*, and thus fuzzified. Finally, this language is *semantic* as the conditions express by these policies are defined using the terms and values of an ontology (trust factors ontology).

Based on this policy language, we framed a trust management system (cf. Figure 6.6) that allows virtual communities members to specify *trust policies* based on which they make their trust decisions. In this system trust decisions are automated via the use of *trust decision function* but our model only manages the evaluation of trust based on which these decisions are made.

Finally, we presented in the last section, how the concepts introduced in this model and the ones we defined in the multi-agent-based virtual community framework we presented in Chapter 5 can be weaved. The integration of these models allows us to build a generic trust management system that can assist virtual communities human users in their trust decisions.

In the next section, we will see how this policy language and the model we framed in this section are used to achieves the objectives of this thesis (cf. Section 1.3).

6.10 French Summary

Dans le hapitre 6, nous avons présenté un modèle multi-agent de communautés virtuelles qui permettait à des individus distants géographiquement de coopérer dans le cadre d'un objectif commun. Toute coopération comporte une part non négligeable de risque qu'un des partenaires exhibe un comportement non coopératif ou malveillant. Ainsi, la problématique de la confiance se trouve être au centre des préoccupation de chaque membre de ces communautés. À cet effet, nous présentons dans ce chapitre le modèle que nous proposons afin de construire et gérer les relations de confiance entre les membres de communautés virtuelles.

6.10.1 Aperçu de l'approche

Les communautés virtuelles sont le lieu d'intenses interactions dont la majorité concerne des accès à des ressources potentiellement sensibles. De plus, la plupart de ces actions sont réalisées par des agents assistants qui agissent au nom des utilisateurs auxquels ils sont affectés, l'automatisation de la prise de décision, et plus particulièrement les décisions de confiance, est une préoccupation majeure de la conception de ces systèmes. Ainsi, l'objectif central des contributions présenté dans ce chapitre vise à concevoir un modèle de gestion de la confiance qui contribuera à cette automatisation. Cependant, avant de présenter notre modèle nous commençons par clarifier ce que nous voulons dire par modèle de confiance.

Definition 67 (Modèle de confiance) *Un modèle de confiance sert à (i) identifier les informations à partir desquels la confiance est évaluée, (ii) définir un formalisme pour exprimer les contraintes sur ces informations, (iii) spécifier le schéma d'évaluation qui sera utilisé pour dériver une mesure de confiance à partir de ces informations.*

À partir de cette définition, nous proposons dans la Figure 6.9 une illustration abstraite de comment notre modèle pourrait être utilisé dans le cadre d'une interaction particulière. Ici, l'interaction implique un demandeur (i.e., *requester*) et un contrôleur (i.e., *controller*). Le *demandeur* a pour objectif d'exécuter une opération sur la ressource contrôlée par le contrôleur. Pour cela, il doit gagner la confiance de ce dernier.

Tel illustré dans la figure 6.9, le processus d'établissement de la confiance commence lorsque le contrôleur prend connaissance de la volonté du demandeur à manipuler sa ressource. Le contrôleur extrait alors la politique de confiance qui s'applique à au type de manipulation et au type de la ressource qui fait l'objet de la demande. Après, le contrôleur et le demandeur entrent dans une phase d'interaction visant à satisfaire chacune des conditions exprimée dans la politique du contrôleur. Le demandeur peut satisfaire la politique du contrôleur si ce dernier est convaincu que :

- le demandeur possède les propriétés sur lesquels les conditions de sa politique sont exprimées, et que

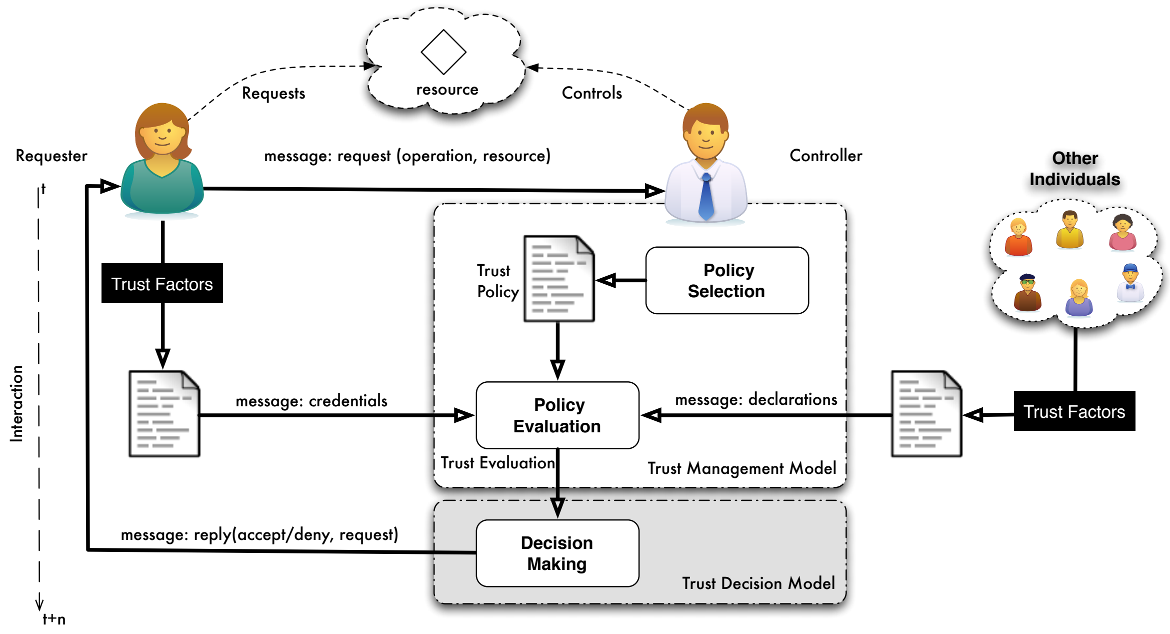


Figure 6.9 – Overview of the trust model.

- les valeurs de chacune de ces propriétés satisfont les seuils de valeurs exprimées dans les conditions.

Ainsi, une politique exprime des contraintes sur les propriétés des partenaires que le contrôleur considère comme étant un préalable à l'établissement de la relation de confiance.

Ainsi, nous considérons chaque propriété dont la pertinence pour l'évaluation de la confiance est partagée par tous comme étant un *facteur de confiance*. La politique indique également au contrôleur quelles informations ce dernier doit collecter afin d'évaluer la confiance. Cette information peut être soit fournie par le demandeur lui-même (e.g., credentials) ou bien agrégée à partir les témoignages des autres (i.e., déclarations). Une fois que toutes les informations requises pour l'évaluation de la politique collectées, le contrôleur peut évaluer la satisfaction de sa politique. Durant cette phase, le contrôleur évalue si les informations collectées satisfont les conditions exprimées dans la politique. Si c'est le cas, le contrôleur va autoriser le demandeur à exécuter l'opération souhaitées. Sinon, la relation de confiance ne s'établira pas et le requête du demander sera rejeté.

Ce modèle reproduit fidèlement l'architecture abstraite des systèmes de gestion de la confiance tel que nous l'avions décrit dans le Chapitre 3. L'évaluation du degré de confiance est complètement séparée du mécanisme utilisé pour la prise de décision effective de faire confiance ou non. Comme nous l'avions expliqué dans l'introduction de ce manuscrit, nous nous intéressons plus particulièrement à l'évaluation de la confiance et non à la prise de décision de confiance. Cette dernière étant dépendante du domaine métier de l'application, nous nous limiterons à un mécanisme de décision simple à base de seuil d'acceptation. Dans la suite, nous présenterons rapidement les éléments-clés de ce modèle de gestion de la confiance.

6.10.2 Les facteurs de confiance

Dans cette section, nous introduisons le concept de *facteur de confiance*. Les *facteurs de confiance* représentent les propriétés à partir desquels les membres des communautés virtuelles réalisent leur évaluation de confiance.

Definition 68 (Facteurs de Confiance) *Nous définissons l'ensemble des facteurs de confiance $\mathcal{F}$ comme suit:*

$$\mathcal{F} \subseteq \mathcal{P}$$

où $\mathcal{P}$ est l'ensemble des propriétés qu'un individu peut avoir (e.g. ceux présentées dans la Section 5.4)

Bien sûr, cette ensemble est dépendant du domaine d'application et peut évoluer dans le temps. On assume également que cet ensemble est construit par les participants à travers un processus d'apprentissage des corrélations entre des propriétés est le degré de confiance d'un individu ainsi que son comportement. La pratique d'utiliser les propriétés des individus en tant facteur de confiance est assez réaliste. En effet, en se basant sur son expérience, l'individu peut facilement déduire que les partenaires ayant une certaine propriété p_i ont tendance à trahir notre confiance dans certains contextes, alors que d'autres ayant une propriété p_j se trouvent être dignes de confiance (cf. Chapter 2).

6.10.2.1 Preuves et Indicateurs

On a vu précédemment que dans notre modèle, pour que le contrôleur fasse confiance au demandeur il faut qu'il puisse collecter les informations nécessaires à l'évaluation de sa politique. Or dans un climat de méfiance, dû à la nature virtuelle des interactions, ces informations doivent provenir d'un tiers de confiance puisque les deux parties ne sont guère confiance. Comme vu dans les chapitres 2 et 3, il existe deux approches d'utiliser des tiers de confiance: *autorité de certification* (AC) et la *certification croisée* (CC) [Linn, 2000]. Dans la première, la validité de l'information repose sur la confiance qu'un individu accore aux organisations auxquelles il appartient, alors que dans la seconde repose sur la confiance que porte un individu à l'expérience des autres. Ainsi, en fonction de l'origine de l'information nécessaire pour sa satisfaction, les facteurs de confiance peuvent être classés en deux catégories: *preuves* et *indicateurs*. Par exemple, un permis de conduire est une *preuve* de la capacité d'un individu à conduire une voiture alors que la réputation est l'agrégation des avis qu'ont les gens à propos d'un individu sur sa capacité à conduire.

6.10.2.2 Ontologie des Facteurs de Confiance

Compte tenu de l'hétérogénéité des membres qui composent les communautés virtuelles en matière de propriétés, il est nécessaire que notre système soit capable d'identifier, comprendre et raisonner sur ces propriétés afin de savoir lesquels sont/peuvent être utilisé comme facteurs de confiance confiances. Ceci est l'objectif principal de l'Ontologie des Facteurs de Confiance. Une partie des facteurs de confiance proposé dans l'Ontologies est illustrée dans la Figure 6.10.

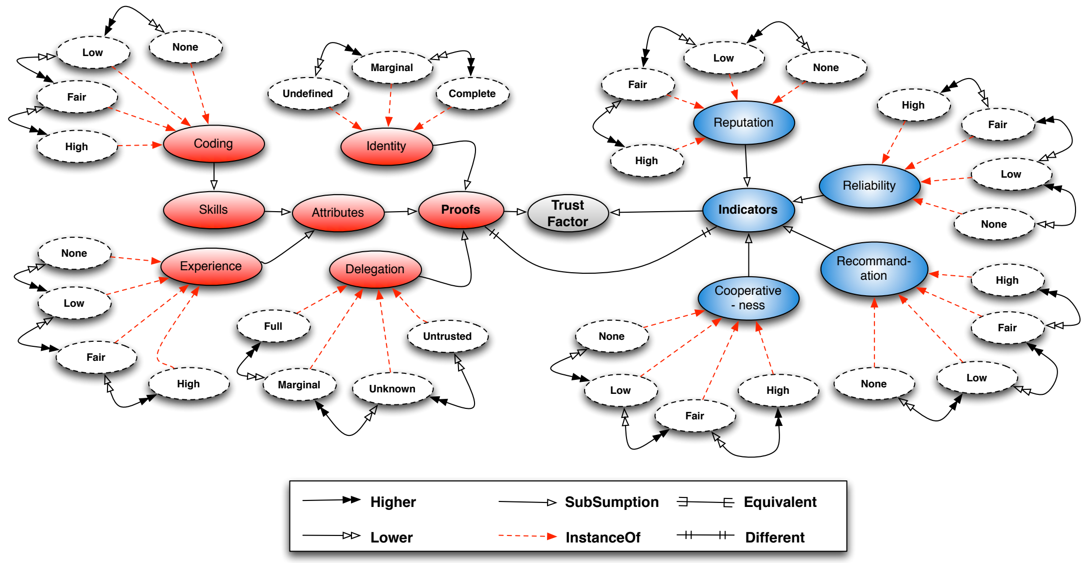


Figure 6.10 – Un fragment de L'Ontologie des Facteurs de Confiance

La racine de l'Ontologie représente un facteur de confiance générique. Ce concept est en fait sub-divisé en deux sous concepts (*preuves* et *indicateurs*) qui représente les deux catégories définies plus haut. *Higher* ($>$) ,*Lower* ($<$), $Different$ ($\neq$) et *Equivalent* ($=$) définissent un ordre sur les valeurs que chaque facteur de confiance peut avoir.

6.10.2.3 Informations de confiance

Dans cette section, nous introduisons le concept d'information de confiance. En faut, les informations de confiance servent à communiquer à propos des facteurs de confiance. Là aussi, en fonction de la nature du facteur de confiance sur lequel les informations portent, les informations de confiance sont classé en deux catégories: *credentials* et *déclarations*. Les *credentials* sont des certificats électroniques établis par des autorités de certification. Leur but est de certifier que l'individu porteur du certificat possède bien la propriété certifiée. Par ailleurs, les *déclarations* sont des témoignages établis par des individus pour échanger des informations sur les propriétés des autres.

Les informations de confiance sont définis par un quadruplet:

$$ti = \langle t, i, s, f, v \rangle$$

où $t \in \{credential, declaration\}$ est le type de l'information, i est l'auteur de l'information, s est l'individu concerné par l'information (i.e., sujet), f est le facteur de confiance à propos duquel l'information est établie, et $v \in f.A$ est la valeur que i associe à s à propos du facteur de confiance f.

6.10.3 La politique de confiance

Une politique de confiance constitue une spécification des conditions à partir desquels un individu peut être considéré comme étant digne de confiance par l'auteur de la politique. Dans notre modèle, nous avons décidé de représenter les politiques de confiance par un ensemble de conditions appelées *critères de confiance.*

Definition 69 (Politique) *Une politique π est défini par:*

$$\pi_{Issuer}^{Pattern} = \{tc_1, tc_2, \ldots, tc_n\}$$

où $Issuer$ est l'individu ayant établi la politique, $Pattern$ est le type de requête auquel s'applique la politique, et $\{tc_1, \ldots, tc_n\}$ est un ensemble de critères de confiance.

Le $Pattern$ représente la finalité de la décision concernée par la politique. Ainsi, chaque pattern est défini comme étant une paire $\langle action, target \rangle$ permettant de spécifier quelles actions va être réalisé et sur quel type de ressource. Par ailleurs, les critères de confiance constituent les briques de base de chaque politique. Elles permettent de spécifier des conditions atomiques sur les facteurs de confiance.

Definition 70 (Trust Criteria) *Chaque critère de confiance tc est défini par :*

$$tc = \langle f, op, v, w, t \rangle \tag{6.12}$$

où:

- *f est un facteur de confiance tel que $f \in \Delta_f$,*
- *$op \in \{=, >, <, \leq, \geq, \neq\}$ est un opérateur de comparaison des valeurs,*
- *v est le seuil pour les valeurs admises tel que $v \in f.A$,*
- *$w \in \mathbb{Z}$ est le poids du critère,*
- *$t \in \{m, o\}$ est le type du critère; "m" veut dire que le critère est obligatoire, alors que "o" veut dire qu'il est optionnel.*

6.10.4 Le schéma de confiance

Dans cette section, nous allons présenter le schéma utilisé dans notre modèle pour calculer une valeur de confiance à partir d'une politique et des informations de confiance collectées. Pour cela, nous avons défini une fonction d'évaluation des politiques de confiance. Le résultat de cette évaluation représente le degré de confiance que le *contrôleur* accorde au *demandeur*. Cette valeur ne présage en rien du fait que le contrôleur va effectivement lui faire confiance et accepter sa requête. Comme dit précédemment, ceci est du ressort du modèle de décision qui n'est pas abordé ici.

Definition 71 (Fonction d'Évaluation des Politiques) *La fonction d'évaluation des politiques de confiance est définie comme suit:*

$$\mathcal{E} : a.\Pi \times a.\Psi \rightarrow [0, 1]$$

Cette fonction est utilisée par un individu a pour calculer le degré auquel les informations qu'il a collecté satisfont les critères de confiance spécifiés dans sa politique (i.e. a.Π). Les informations que le contrôleur collecte à propos du demandeur dans le cadre de l'évaluation de la requête de ce dernier sont stocké dans une structure qu'on appelle *profile* ($a.\Psi$) [4]. Ainsi, la fonction $\mathcal{E}$ à pour tâche de faire correspondre une politique et un profil à la somme pondérée qui correspond à l'évaluation de cette politique.

[4]Pour plus de détails sur les profils nous invitons le lecteur à se référer à la Section 6.2.4.4 du manuscrit.

Chapter 7

Adaptiveness and Social-Compliance in Trust Management

In Chapter 6 we proposed a policy language that that VCs use to specify the properties they require in their partners to consider them trustworthy. Then we extended this language to address the *social* (and collective) dimension of these environment. We have made our language able to express collective policies that express common requirements of the members of community.

In this chapter, we describe how we used this policy language to make our trust management system *social* and *context* aware. To that aim, we first motivate why we considered the objectives of this thesis as *adaptation* problems. With respect to that, we classify the types of pressures to which our policies get adapted and the required adaptation mechanisms. In Section 7.2, we present the mechanism to adapt policies to the *social* and *business* context. Finally, we conclude the chapter in Section 7.6.

7.1 Adaptation Types

From the perspective of behaviour science and psychology [Cervenka et al., 2006, Rakotonirainy et al., 2009], changes in the behaviour of individuals is often correlated to, and can be inferred by (to some extent), the information related to the context wherein its interaction is undertaken. Dey defined context as :

> Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction.

So although not explicitly mentioned in the above definition, *social-context* can arguably be consider as a relevant information to the interaction, and thus included in the above definition. Consequently, *social* and *business* can be considered as equally relevant from humans interactions as both influence and constraints individuals behaviour.

In the light of that, we consider *social-awareness* and *context-awareness* as a unique problem of *policies adaptation* to the interaction context. In Figure 7.1 we characterise this context and analyse the different types of pressures exerted on individuals, and hence their policies, within virtual communities. We identify two kind of pressures: *social pressure* (c.f. Figure 7.1-2 and 7.1-4) and *environment pressures* (cf. Figure 7.1-1 and 7.1-3). *Social pressures* represent the *social* nature of virtual communities emphasised in the thesis introduction (cf. Chapter 1), while *environment pressures* materialise their *dynamic aspect.*

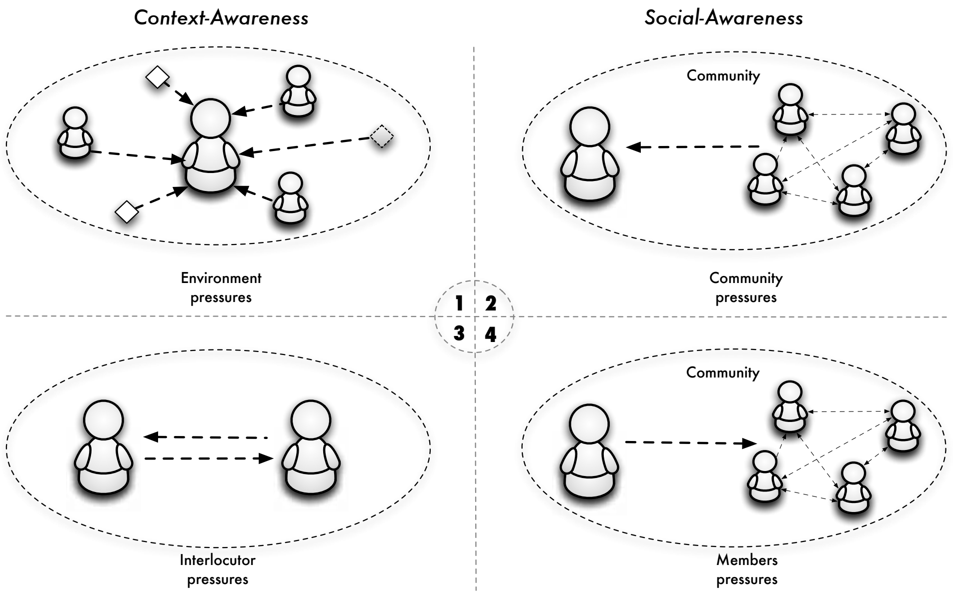


Figure 7.1 – Pressures that may be exerted on policies.

Each of the above pressures affects policies in use within the community (either individual or collective) and thus requires a dedicated adaptation mechanism. In the following, we motivate the need for each type of adaptation and illustrate it with examples.

7.1.1 Business-Context Adaptation

During a trust interaction, the decisions of a trustor are generally affected by the context in which the interaction is undertaken. This context is materialised by the resources that constitute the agent environment (cf. Figure 7.1-1) and the other agent it is interacting with (cf. Figure 7.1-2). In this section, we explain how these elements of the system $\mathcal{S}$ may affect trust policies and suggest specific adaptation mechanisms.

7.1.1.1 Instantiation

In Chapter 6 we presented a policy language based on which two types of policies can be stated; *specific policies* and *generic policies*. Specific policies are stated with respect to a particular resource and a particular operation. Contrariwise, generic policies are stated with respect to a whole category of resources (based on resources type τ). We motivated the need for such generic policies with the high pace of resources creation that makes impossible the definition of policies for each created resources. Accordingly, these policies are generic and requires to be adapted to best fit the specific context in which they will be applied. We call such adaptation *instantiation* (cf. the concept of policies refinement in [Barrett, 2004]).

Moreover, even specific policies require to be *instantiated* when the context for which they are applied is different from the context in which they have been specified. For instance, the particular resources concerned by the policy may evolve in terms of value or in terms of sensitivity. This issue is illustrated in the following example.

Example 7.1.1 *Let us take again the resources r_1 owned by Alice in Example 5.3.3. Initially, the file contains only an abstract description about what Alice wants to develop. Then progressively, these ideas will be implemented and become more elaborated. Thus, intuitively, one can agree that the policy that Alice uses to govern the decision she makes about r_1 would be different for each version.*

In the above situation, it is the file update, which represents here the *business-context* pressures, that motivates the need for policy adaptation. Given the intense and sustained pace of interactions and activities that characterises virtual communities, policy changes such as the ones illustrated in Example 7.1.1 would be so frequent that the human intervention is neither possible nor desirable. Impossible because including a human in the agent decision loop represents a bottleneck strategy as the human can not be present each time an agent has to make a decision [Grandison, 2003, Artz and Gil, 2010, Seamons et al., 2002, Blaze et al., 1999a]; and not desirable as then the human intervention, besides being time-consuming and error-prone, means a loss of any automation advantage made possible by using trust management systems.

7.1.1.2 Negotiation

The trust model we framed in Section 3.2.1 assumes an *asymmetric* conceptualisation of trust (cf. Section 2.3.3.1). Thus in order to gain the trust of the controller, a requester (or a resource consumer) needs to disclose its credentials, making itself vulnerable as well. Indeed, the credentials that can be released to gain access to other's resources can be as sensitive for the requester as the resource itself. Therefore, in most of the cases, controllers and requesters rely on trust negotiation to establish trust gradually by disclosing only credentials for which policies have been satisfied by the interlocutor (cf. Section 3.2.4). The following example aims at illustrating such issue.

Example 7.1.2 *Alice wants to access a resource owned by Bob. But the policy specified by Bob requires that Alice discloses her identity credential to prove that she is an adult. The problem is that Alice's identity credential is a sensitive resource as it contains her personal and private information such as her age, her address and her photo. The disclosure of such information can be harmful for Alice as it represents a violation to her privacy. So using negotiation, Alice is able to propose to Bob alternative credentials (e.g. social security credential) that minimises her privacy loss.*

In the above example, *Alice*'s alternative proposition may represent a pressure to *Bob*'s policies. Thus a successful negotiation must necessary take into account such pressures, otherwise negotiation would be reduced to strategies compatibility which has its limits as discussed in Section 3.4. In the light of this, we propose a mechanism by means of which agents adapts their policies during a negotiation.

7.1.2 Social-Context Adaptation

We presented in Section 4.2, social mechanisms that explains how norms emerges (cf. Section 4.2.1.1), how individuals tend to conform to the norms of a community (cf. Section 4.2.1.2) and how in certain circumstances, individuals succeed in changing the norms of the community. In this section, we analyse the pressure exerted on policies in social context in the light of these social theories. Consequently, we identify *top-down pressure* (cf. Figure 7.1-2) by means of which the community forces its members to adapt their individual policies with respect to a collective one (cf. Section 7.1.2.1), but also a *bottom-up pressure* (cf. Figure 7.1-4) in which the individual policies of each member of the community shapes the collective policy. This bottom-up pressure is then further detailed into two sub-pressures in reference to the process by means of which agents builds the collective policy (cf. Section 7.1.4), and those they use to change it (cf. Section 7.1.3).

7.1.2.1 Combination

We introduced in Chapter 5 the concept of *collective policies* that the members of the same community use to constraint their autonomy and harmonise the trust decisions they make. However, we omitted to mention where does these collective policies come from. This issue is illustrated in the following example.

Example 7.1.3 *Let us consider the community ABC composed of Alice, Bob and Carol. The three members have a common project to develop an Android application. But even if the three members share the same objective, they differ in terms of policies they use. For instance, Bob is willing to accept new members having a reputation of* 0.6*. In turn, Alice does not trust other agent having a reputation below* 0.7*. Finally, Carol does not rely on reputation and prefers a marginal identity.*

In the above example, if each member uses only its individual policy, request to join the community will be accepted or rejected but not for the same reasons. Such situation can be harmful for the community cohesion and stability. To avoid that, the members of the community must agree on minimal conditions that a requester must meet to see its request accepted. As illustrated in Figure 7.1, these conditions represent collective policies that emerge from agents' interactions in which the individual policy of each agent constitutes the pressure that influences the way this collective policy is built. In the light of this, we propose to use *combination* as a way by means of which agents make emerge a collective policy by adapting their individual policies (cf. Section 4.1.1).

7.1.2.2 Integration

Having collective policies is paramount for the community stability. But if these policies are not used by the members, they become useless and the community would still be prone to instability. Therefore, each member of the community has the obligation (via norms) to comply with these collective policies, otherwise he should be punished (cf. Section 4.3.2.2 and Assumption 6.8.2). To illustrate this issue, let us consider the example described hereafter.

Example 7.1.4 *Bob received a request from an agent with a marginally trusted identity and a reputation of* 0.6*. If Bob relies only on his individual policy, he would have trusted this agent and accepted his request. However, the collective policy that Bob agreed on with Alice and Carol states that a new member must have a marginally trusted identity and a minimum reputation of* 0.7*.*

In the above example, it remains to *Bob* to comply or not with this collective policy. Complying with the collective policy means adapting its own one with respect to this policy. So this collective policy aims at dealing with agents autonomy by prescribing trust decisions that are too fare from the decisions that other agents would have made. Therefore, we consider collective policies as an indirect form of social pressure that the individual policy is subject to. To that aim, we suggest a third adaptation mechanism that is inspired from the policies *integration* algorithms presented in Section 7.5.2. The objective is to consider trust management as normative systems in which collective policies constitute the norms that all TMS must comply to. In the light of this, *integration* endeavours at enabling agents to make their individual policies comply with the collective ones when making decisions that may impact the whole community.

Furthermore, policies integration can also be used by agents when making decisions that do not necessary affect other members. For instance, a member can benefit from the experience of others (via the collective policy) to make better trust decisions by integrating its policy with the collective one.

7.1.3 Evolution

Now that the motivations for *instantiation*, *negotiation* and *integration* are established, one can wonder whether *collective policies* should be adapted, and if so, why and how this can be achieved. The following example illustrates some situations requiring collective policies adaptation.

Example 7.1.5 *Let us consider the situation in which the members of the ABC community need to enrol new members to help them developing the application they are involved in. However, the collective policy they use to admit new members is too restrictive and prevents them to achieve their goal. While this policy fulfils well its role in preventing the membership of untrusted agents, the conditions this policy conveys are far too restrictive for any trusted agent.*

The above example illustrates that just like individual policies, collective policies require to be adapted when it prevents the community members to achieve their goals. In the light of that, we propose a policy adaptation mechanism by means of which agents can trigger changes in the collective policy they are subject to. These adaptation can be triggered when the collective policy is *too restrictive* like in the previous example; but it can also be triggered in situations in which the collective policy is *too permissive.* For instance, one can imagine the situation in which a majority of agents noticed that all bad experiences they had involved agent having a particular property (e.g. agents having no recommendations). In such situations, the TMS these agents are using should be able to reason on collective policies, detect eventual adaptation and perform this adaptation in agreement with the other members of the community (via their TMS). We refer to this mechanism as *policies evolution.*

7.2 Policy Adaptation

In this section, we present how agents are able to react to each of the above pressures by adapting their policies. To that aim, we use *adaptation meta-policies* that each trust management systems (ASC-TMS) relies on to automatically change running policies.

The objective is to establish a process, similar to the one used in norms adaptation (cf. Section 4.3.3.2) and apply it to policies adaptation. By making *adaptation meta-policies* explicit, it becomes a relatively simple matter to guarantee that these agents are always using the policies that best handles the context in which they evolve.

Definition 72 (Meta-policy) *Let $\mathcal{M}$ be the set of meta-policies. Each meta-policy $m \in \mathcal{M}$ is defined as a function :*

$$m : \mathcal{I}_a \times \Pi \times \mathcal{S}^t \times \mathcal{K} \to \Lambda^\Pi$$

where the function m maps an interaction $\mathcal{I}_a$ [which is in most of the cases a request], a policy, an image of the system $\mathcal{S}$ at a time t and a condition expression to the appropriate set of actions to be performed on the policy.

So based on their meta-policies, agent are able to trigger changes on there policies when some conditions of the context hold. In the following, we present based on what information (cf. Section 7.2.1) and how these conditions are specified (cf. Section 7.2.2). Then in Section 7.2.3 we describe the kind of adaptation a meta-policy can bring to policies on which they have been triggered. Finally, in Section 7.2.4 we present the formalism used to express these meta-policies and illustrates with some examples.

7.2.1 Adaptation Conditions

The ASC-TMS is has been designed to monitor and reason about both the *business-context* and the *social-context* in which the trust evaluation has to be made. The objective is to capture all information that are relevant to policies adaptation. However, and in order to make our proposal realistic, we assume that only some types of information that are available in S are concretely monitored by the ASC-TMS for adaptation purpose. To that aim, the ASC-TMS relies on a common ontology to know what information should be captured.

Definition 73 (Context Ontology) *We define the context ontology $\Delta_{ctx} \subseteq \Delta$ as a triplet:*

$$\langle T, A, R \rangle$$

where T represents the term that describes the context information to be captured, A are the context values ($\Delta_{ctx}.A = \{x / \exists y \in \Delta_{ctx}.T \wedge x \doteq y\}$) and R are relationships between elements from $T \times A$ (such as $\doteq$).

The ontology Δ_{ctx} constitutes a data structure in which all pertinent contextual information with respect to policies adaptation are collected. The information captured by Δ_{ctx} fall into four categories, namely *internal variables*, *interlocutor variables*, *environment variables* and *social variables.* These *variables* are presented in the next sub-sections.

7.2.1.1 Internal Variables

The first class of information used in adaptation concerns the *agent itself.* This class of information includes the agent personality traits that affect the way that the agent adapts it policy. These variables include but are not limited to:

- **Cautiousness** expresses the degree to which an agent is sensitive to threats such as *collisions*, *credentials forgery*, etc.
- **Opportunism** expresses the degree to which an agent is keen to achieve its interactions, an thus relax its policy.
- **Cooperativeness** determines whether an agent is *altruist*, *cooperative* or *selfish.*
- **Compliance** expresses to which extent an agent is willing to comply with the collective policy.

Example 7.2.1 (Internal Variables) *With respect to our running example, Alice is very cautious. So each time she instantiates a policy she tends to restrict it. But Alice has also a high degree of opportunism which enables her to succeed in her negotiation by relaxing its policy when the pressure of its interlocutor is very high.*

7.2.1.2 Interlocutor Variables

The second class of information belongs to the business-context of the interaction. It concerns the *subject of the interaction* (i.e. the interlocutor). This information includes the general properties available in the system, collected from its profile or gathered from other agents. Here, we identify three main information that may affect/influence the way an agent adapts its policy.

- **Proximity** specifies the nature of social relation that links the resource controller and its interlocutor.
- **History** expresses whether the past experiences with the interlocutor were good or not.
- **Reward** represents the pay-off that the agent will (potentially) gain if the interaction succeed. It can be computed based on the interlocutor competence degree, but it can also be stated by the interlocutor itself at the beginning of the interaction or during a negotiation.

Example 7.2.2 (Interlocutor Variables) *Agents that interact with Alice fall into three categories,* acquaintances*,* fellows *and* strangers*. Based on that Alice affects to each agent a* proximity *value that determines how Alice will adapt (or not) is a specific way for each type of* proximity*.*

7.2.1.3 Environment Variables

The third category of context information completes the previous ones in characterising the business-context of the interaction. It contains information that relates to the *object of the interaction* (e.g. resource, community). It includes the following variables:

- **Scarcity** is computed by the agent and expresses the scarcity of the resources in the system.
- **Demand** expresses the demand rate. This demand rate can be computed by each agent based on the requests it receives or aggregated based on other agents testimonials.
- **Type** corresponds to the resource type ($\tau \in \Delta_\tau$) presented in Definition 18.
- **Sensitivity** corresponds to the resource sensitivity ($\varsigma \in \Delta_\varsigma$) presented in Definition 21.
- **Value** corresponds to the resource value ($\nu \in \Delta_\nu$) presented in Definition 22.

Example 7.2.3 (Environment Variables) *The policy that Alice uses to grant access to its application materialised by* r_1 *(Example 5.3.3) is directly influenced by variables such as* scarcity *and* demand*. For instance, if few participants manifested interest to Alice project, she would be accommodating and her policy less restrictive. Analogously, Alice would be less accommodating if her resource is rare and very valuable.*

7.2.1.4 Social Variables

This last class of contextual information includes general properties of the system $\mathcal{S}$ and the activities that are undertaken in it. It relates also to reported facts and incidents such as *collusion*, *credentials forgery* or *norms violation*. We summarise these variables as follows:

- **Agents population** contains the agents global population (i.e. $|\mathcal{A}|$).
- **Communities count** contains the number of communities (i.e. $|\mathcal{C}|$) that are available in the system.
- **Community attractiveness** is computed based on the value of the collective resources shared among the community, the value of the resources of each member of the community and the perceived competence of the members that compose the community.
- **Community size** contains the number of agents that are members of the community (i.e. $\forall c \in C$, Community Size $= |c.A|$).
- **Collusions** expresses the number of collusion incidents reported in $\mathcal{S}$.
- **Credentials forgery** expresses the number of incidents that are related to the misuse of credentials (*fake*, *forged*, *untrusted CA*).

Example 7.2.4 (Social Variables) *Alice, which is a compliant member, makes use of the* community size *and the* community attractiveness *to decide to comply or not with the collective policy of her community. So these two variables determines when Alice policy could be adapted with respect to the collective policy.*

7.2.2 Expressing Condition

Now that the context ontology $\Delta_c tx$ has been defined. We present hereafter how the elements of this ontology are used to express conditions over the context. These conditions states what value the variables we presented above should have to make the adaptation necessary.

Definition 74 (Context Condition) *Let* $\mathcal{K}$ *be the set of all context conditions, each condition* $k \in \mathcal{K}$ *is defined by*

$$k = (X.type\ operation\ value) \tag{7.1}$$

where $X \in \mathcal{S}$ is the agent, the resource, the community or any other element of $\mathcal{S}$ on which the condition is stated, type $\in \Delta_{ctx}.T$ is the type of the condition, operation $\in \Delta_{ctx}.A$ is the comparison operator, and value $\in \Delta_{ctx}.A$ is the threshold value.

Example 7.2.5 *For instance, a condition k_1 that checks whether the size of the community c is above a certain threshold (e.g. 100 agents) is defined as follows:*

$$k_1 = (c.ComSize \geq 100) \tag{7.2}$$

In this section, we presented some examples of variable that the ASC-TMS relies on to determine when policies should be adapted. These variables are mandatory in the context of our model but they can be easily extended or substituted with other application-specific variables. Knowing when to adapt a policy is important, but more importantly, the ASC-TMS needs to know how this can be done. To that aim, we introduce in the next section the adaptation operators we endowed the ASC-TMS to make policies adaptation effective.

7.2.3 Adaptation Operations

In order to make policies adaptation possible, it is necessary to endow the ASC-TMS with a set of adaptation operations that can be applied to policies. To that aim, we extend Λ, the set of action that agents can perform, with the set Λ^{Π} of adaptation operations. Each operation is an action by means of which an agent can change the policy in a particular way.

Definition 75 (Adaptation Operation) *Adaptation operations are defined as follows:*

$$\forall \omega_i \in \Lambda^{\Pi},\ \omega_i : 2^{\Pi} \rightarrow \Pi$$

In Definition 75, we consider adaptation operations as functions that map a set of policies (eventually composed of one policy) to another policy, that represent the adapted policy. Based on this definition, adaptation operations are split into two categories: *simple operations* and *complex operations*. We assume in the following that a policy π is adapted, resulting in an adapted policy $\pi^{'}$. The definitions below details the preconditions and effects of each operations.

7.2.3.1 Simple operations

Simple operations are used to change a policy in three difference ways.

- A *trust criterion* that compose the policy can be deleted or a new trust criterion can be added.
- The *values* of existing trust criteria can be altered and new values can be used.
- The *weights* of the trust criteria can be changed to make some trust criterion less or more important during the policy evaluation.

To that aim, we defined seven *simple operations* that we present hereafter:

1. **AddCriterion**(π, tc_i): this primitive adds, to the individual policy π, the trust criterion tc_i.

   Precondition: $\forall tc_j \in \pi, tc_j.f \neq tc_i.f$

   Effect: $\pi^{'} = \pi \cup \{tc_i\}$

2. **DelCriterion**(π, f_i): this primitive looks for a trust criterion of type f_i and removes it from the policy π.

   Precondition: $\exists\ tc_j \in \pi/tc_j.f == f_i$

   Effect: $\pi^{'} = \pi - \{tc_j\}$

3. **UpdateCriterion**$(\pi, f_i, op_i, v_i, w_i, t_i)$: this primitive looks for a trust criterion of type f_i and changes it with the new values. If such a criterion does not exist, it adds it to the policy π.

   Precondition: $\exists\ tc_j \in \pi/tc_j.f == f_i$

   Effect: $\pi^{'} = \{\pi - \{tc_j\}\} \cup \{\langle f_i, op_i, v_i, w_i, t_i\rangle\}$

4. **RestrictCriterion**(π, f_i): this primitive looks for a trust criterion of type f_i and restricts its value v. The restriction of v is performed by assigning it a new value $v^{'}$ that is considered as *Lower* ($<$) than v (i.e. Δ_f.*R.Lower*) with respect to the trust factors ontology Δ_f.

   Precondition: $\exists\ tc_j \in \pi/tc_j.f = f_i$

   $\exists\ v' \in \Delta_f.A\ /v \doteq f_i \wedge v' < v$ if tc_i.op is "$\leq$" or "$<$"

   $\exists\ v' \in \Delta_f.A\ /v \doteq f_i \wedge v' > v$ if tc_i.op is "$\geq$" or "$>$"

   Effect: $tc_j.v \leftarrow v^{'}$

   Effect: HigherCriterion(π, f_i) otherwise.

5. **RelaxCriterion**(π, f_i): this primitive looks for a trust criterion of type f_i and relaxes its value v. This is performed by assigning it a new value $v^{'}$ that is considered as *Higher* ($>$) than v with respect to the trust factors ontology Δ_f.

   Precondition: $\exists\ tc_j \in \pi/tc_j.f = f_i$

   $\exists\ v' \in \Delta_f.A\ /v \doteq f_i \wedge v' < v$ if tc_i.op is "$\leq$" or "$<$"

   $\exists\ v' \in \Delta_f.A\ /v \doteq f_i \wedge v' > v$ if tc_i.op is "$\geq$" or "$>$"

   Effect: $tc_j.v \leftarrow v^{'}$ if tc_i.op is "$\leq$" or "$<$" or "$\geq$" or "$>$"

   Effect: LowerCriterion(π, f_i) otherwise.

6. **LowerCriterion**(π, f_i): this primitive looks for a trust criterion of type f_i and lowers its weight w.

   Precondition: $\exists\ tc_j \in \pi / tc_j.f = f_i$

   Effect: $tc_j.w \leftarrow Max(w-1, 1)$

7. **HigherCriterion**(π, f_i): this primitive looks for a trust criterion of type f_i and highers its weight w.

   Precondition: $\exists\ tc_j \in \pi / tc_j.f = f_i$

   Effect: $tc_j.w \leftarrow Min(w+1, 10)$

All these adaptations affect (by weakening or strengthening) the running policy making it more restrictive or less restrictive. So these operations should used within cautious strategies. Otherwise, using the adapted policy can lead to negative consequences. For instance, based on the general rule "more restrictive policies are safer", many are those who minimise the risk in using too restrictive policies. However, it has been proven too *permissive* policies are breaches to security and too *restrictive* policies are brakes to cooperation [Vogel and Giese, 2012]. So to avoid both situations, the above adaptation operations are used based on a *parsimonious strategy*; only in specific circumstances (cf. Section 7.1) and only under particular conditions (cf. Section 7.2.1).

7.2.3.2 Complex operations

While *simple operations* adapt a policy to build another policy, *complex operations* adapt at least two policies to build another policy. Moreover, complex operations are considered complex because they use also simple operations to achieve their adaptation. These operations are essentially used to implement the mechanism to adapt individual policies under the pressure of collective policies. However, up to now, we omitted to talk about where do these collective policies come from.

1. **Integrate**$(\pi_1, \pi_2, \texttt{ih})$: this primitive integrates two policies using the integration heuristic `ih`. How the integration is performed needs detailed explanations. Therefore, we leave its description to the Section 7.5.2 in which we provide the algorithm we used for that.

   Precondition: $\exists\ \pi_1, \pi_2 \in \Pi$

   Effect: $\pi' = \pi_1 \bowtie \pi_2$

2. **Combine** (Π', c, mh, π'): this operation is used to combine a set of policies $\Pi' = \{\pi_n, \pi_m\}$ (where $n \neq m$ are agents identifiers) into a policy π' representing the collective policy of the community c. 7.5.1

   Precondition: $\Pi' = \{\pi_n,\ ...,\ \pi_m\}\ /\ \forall i \in [n, m], \pi_i.Issuer \in c.M$

Effect: $\pi' = \pi_n \uplus ... \uplus p_n$

In Section 7.2.1, we presented the context information that defines **what** conditions the context must satisfy in order to make the agent adapt its policy. Then we introduced in Section 7.2.3 the adaptation operators that describe **how** agents can bring adaptation to policies. In the next section, we illustrate how these concepts (i.e adaptation conditions and adaptation operations) are used alongside to define *adaptation meta-policies*.

7.2.4 Adaptation Meta-Policies

Meta-policies are defined in our system using *event-condition-action* rules (ECA) [Russell and Norvig, 2010]. The use of ECA rules is a well-known approach for enabling human users to specify to agents the desired actions to be performed when they are subject to specific constraints. The most salient feature of ECA-based meta-policies is their declarative semantics. When one of the aforementioned pressures is detected (cf. Section 7.1), the matching meta-policy is triggered and the corresponding adaptation operations are executed, if the conditions hold.

Each meta-policy is composed of three parts: *the event*, *the conditions* and *the actions* as illustrated hereafter:

$$\langle event \rangle : \langle condition \rangle \leftarrow \langle action \rangle$$

The intuitive reading of an ECA meta-policy is *"if the event occurs in a context where the conditions are true then the actions must be executed"*. The event is generated by the TMS whenever it identifies one of the situations requiring policy adaptation (i.e. instantiation, negotiation, combination, integration, and evolution). The $\langle condition \rangle$ is a possibly empty set of conditions $k_i \in \mathcal{K}$. These conditions represent filters over the agent's *context* $\mathcal{S}^t$, the $\langle action \rangle$ is a sequence of adaptation operations through which the agent can change its policy.

The $\langle event \rangle$ is the triggering event that will launch the adaptation. The triggering event allows to associate a name to each adaptation meta-policy.

The above meta-policies syntax consists of three basic predicates: *event predicates* ($\mathcal{S}^t$), *action predicates* (Δ^{Π}_{ω}) and *context condition predicates* ($\mathcal{K}$).

All predicates are system-defined and are given to the virtual community users through their agents in order to allow them define adaptation meta-policies. Event, action and condition predicates can be of different arities.

The *event predicates* represent basic events that trigger the adaptation phase. For example, $InstanciatePolicy(\Pi_{a_i}^{\langle r.\tau, r.o \rangle}, q, a_j, r, o, S^i)$ is the event symbol that triggers the instantiation of the policy $\Pi_{a_i}^{\langle r.\tau, r.o \rangle}$ with respect to a requester a_j that sends the request r in which he asks to perform the operation o on the resource r in a context $\mathcal{S}^t$ (voir Section 7.4.1).

The context conditions represent expressions containing constants and variables. The variables can be related to the arguments that appear in the event part of the policy. For instance, one can express conditions on the agent properties (i.e. internal variables) resource r (i.e. environment variables) attributes, the partner a_j properties (i.e. interlocutor variables), the community c_i (i.e. community variables) properties and/or on the attributes of any elements composing the context S^i.

Example 7.2.6 *In order to illustrate how meta-policies are defined, let us consider the following examples :*

$$Instantiate(\Pi_{Bob}, \_, \_, R, \_, \_) : R.value^t > R.value^{t-1}$$

$$\leftarrow RestrictCriterion(\Pi_{Bob}, reputation).$$

In this example, Bob uses a meta-policy that systematically restricts the reputation of the selected policy each time the requester resource value increases.

We presented here how meta-policies can be specified and how, based on these meta-policies, policies are adapted in response to context constraints. Now, we need to think about the means by which an agent can coordinate the various adaptations he needs to perform in order to make its policy fit the constraints he is subject to. To that aim, we need to specify a policy adaptation cycle that makes the agent aware of the ordering of process that a policy goes through since its selection to its evaluation.

7.3 The Policy Adaptation Cycle

In this section, we extend the policy evaluation and selection mechanisms of ASC-TMS (cf. Section 6.7) to support policy adaptation. The resulting architecture is depicted in Fig. 7.2 below.

As illustrated in Figure 7.2, now the policy repository contains both individual (i.e. Π_a) and collective policies (i.e. Π_c). We added also a *meta-policy repository*. Finally, we added a *context management module* which aims at collecting context information that are used to verify whether context conditions stated in the meta-policies are satisfied or not.

The adaptation of policies is entirely handled by the policies management module. The objective of this component is to automate the adaptation process based on a predefined adaptation cycle which is inspired from *case-based reasoning* (CBR) [Aamodt and Plaza, 1994].

CBR is a reasoning type that uses cases (past experiences) to solve new problems by adapting solutions that have been used to solve similar problems in the past. A case can be defined as an experience, and is composed of three elements: a description of the initial problem, the solution that provides the sequence of actions carried out in order to solve the problem, and the final state which describes the outcome of the used solution.

The functioning of the ASC-TMS depicted in Figure 7.2 reproduces a CBR cycle in which solutions represent *trust policies*, problems constitute *trust requests* and cases are the *agent experience* (cf. Definition 64). The way in which experiences (cases), requests (problems) and policies (solutions) are managed is called the CBR cycle. This cycle is depicted in Figure 7.3 below.

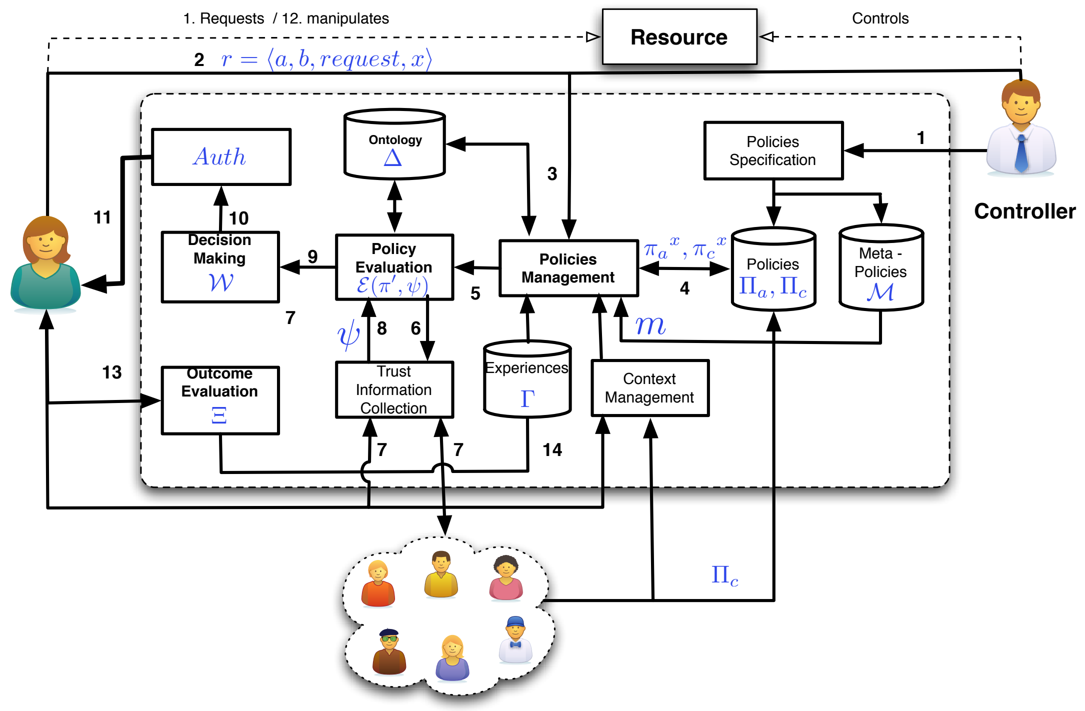


Figure 7.2 – Extension of the trust model illustrated in Figure 6.6.

The policies adaptation cycle shown in Figure 7.3 consists of five phases: *select*, *reuse/adapt*, *evaluate*, *decide* and *retain*. The *select* phase starts once a new request is received. The request is then parsed and an appropriate algorithm is applied to retrieve, from the policy repository, the policy which best matches the *pattern* of the received request. Once the most appropriate policy has been retrieved, the *reuse/adapt* phase can start. In this phase, the selected policy is adapted to obtain a policy that best handles the running constraints. The adaptation is performed by the heuristics defined by the VC member using adaptation meta-policies (cf. Section 7.2). If the policy cannot be further adapted, it is reused as it is without change. The evaluation phase consists of applying the policy evaluation function (cf. Definition 61) to assess to which degree the partner's profile information satisfies the resulting policy. In the *decide* phase, the agent decides whether to trust or not the requester, and thus accept its request.

Finally, based on the interaction outcome, the agent updates in consequence its experience repository and decides whether the policy that he used during the interaction should be retained or not.

As the *retrieve* and *evaluate* phases has already been presented in Chapter 6, we dedicate the next section to the presentation of the mechanisms and process we use to achieve the *adapt* phase. Then we present how these adaptation meta-policies can be used by VC members to

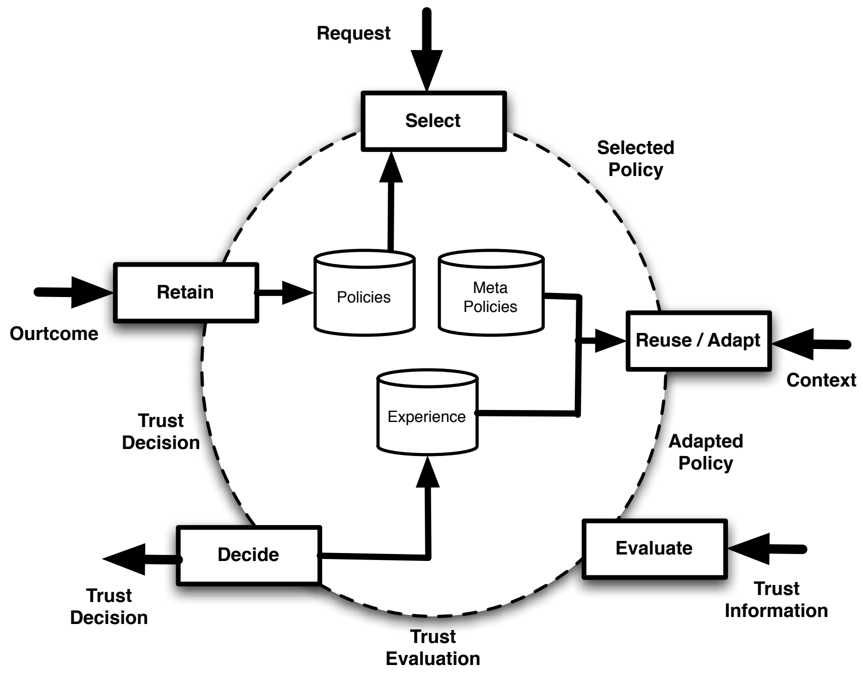


Figure 7.3 – Policies adaptation cycle based on the CBR model.

define adaptation heuristics. Finally, the *retain* phase will be presented later on in Section 7.5.

7.4 Adaptiveness: Adaptation to Business-Context

In our approach, the instantiation of a policy is performed with respect to contextual information as highlighted above. The main novelty of the ASC-TMS relies in its capacity of monitoring the constraints to which the policy issuer is subject and to adapt consequently the policies it is using. As illustrated in Figure 7.1, virtual communities include four types of constraints to which a policy can be subject. Each type of pressure needs a specific adaptation heuristics. Thus, we identified in Section 7.1 four kinds of adaptation, namely *instantiation*, *negotiation*, *integration* and *combination*. In this section, we describe how the meta-policies and the adaptation operations introduced previously are used to specify adaptation heuristics with respect to environment changes (the left part of Figure 7.1). In this form of adaptation, we distinguished instantiation from negotiation. Instantiation will be described in the next section while negotiation will be detailed later in Section 7.4.2.

7.4.1 Instantiation

Instantiation refers to the concept of exploiting environment information to produce accurate trust policies. So instantiation aims at transforming running policies in order to fit environment changes. Our hypothesis is that using the context information to tailor trust policies should result in the most appropriate one.

The need for adapting trust policies based on the context in which the interaction is undertaken has been long recognised. However, most of the proposals are ad-hoc or manual. For these reasons, we propose here the use of *environment variables* (cf. Section 7.2.1.3), a.k.a. context information, to dynamically and automatically trigger *addition*, *removal*, *restriction*, …,*relaxation* of the trust criteria contained in the selected trust policies.

In the following, we first define the concept of instantiation then we detail how the meta-policies formalisms can help agent maintaining their trust policies.

Definition 76 (Instantiation) *Instantiation is the process or outcome of a process by which trust policies are adjusted in response to environment properties and stimuli.*

In order to illustrate the benefit that agent can gain using meta-policies alongside with trust policies, we start by presenting the following scenario.

Example 7.4.1 *As an illustrative example, suppose that the community members have the obligation to monitor and report any collusion incident* [1] *[Liu et al., 2008]. Let us say that up to 5 collusion incidents can be reported in normal circumstance. Now let us consider Dave, a member who would like to restrict the use of its resources* r *to the members having a reputation score above* 0.7*. Dave selected this score because he knew that this score means that the trusted individual is a good member in the community. However, in presence of collusion , Dave's policy will be no longer valid as malicious members car easily gain such score. So ideally, Dave would like to use a* meta-policy *that would guarantee that the conditions under which Dave issued the policy remains valid, and to react consequently if not.*

In the above example, the collusion incidents represent the *context filter* property on which conditions will activate the meta-policy. So there is a context term associated to this condition, say *Collusion*, in the contextual information ontology Δ_{ctx}. So in the condition part of the meta-policy we will need to check the count of *collusion incidents* to perform the appropriate adaptation action. To that aim we need to extend the *Collusion* symbol with an attribute named *count*. In general, context elements will have a set of attribute names associated with them, each one having an associated type (cf. Chapter 5). For example, the type of *count* will be an integer. We will use *Collusion.count* to refer to the count attribute of the collusion incidents context attributes. Now, suppose we define an action that changes the required reputation score from the policy. The action signature is $UpdateCriterion(\Pi_{Dave}^{\langle \tau_r, o \rangle}, v_{f_i})$. When this action

[1]Collusion detection is a trending research issue that has been studied in many fields including PageRank [Zhang et al., 2004], online auction systems and P2P file-sharing networks [Lian et al., 2007]

is executed, the value of the reputation trust criteria expressed in the policy will take the new value. The following meta-policy covers the actions required when the collusion incidents are abnormal (i.e. exceeds 5).

$$\begin{gathered} Instantiate(\Pi_{Dave}^{\langle r.\tau, r.o\rangle}, q, a_j, r, o, S^i) : S^i.Collusion.count > 5 \\ \wedge\ \Pi_{Dave}^{\langle r.\tau, r.o\rangle}.reputation < 0.8 \\ \leftarrow UpdateCriterion(\Pi_{Dave}^{\langle r.\tau, r.o\rangle}, reputation, 0.8, 1) \end{gathered}$$

The above meta-policy can also be made generic. By generic we mean that its application is not specific to some kind of policy but general to all running policies regardless of the policy pattern, the requester and the request to which is applies. Such meta-policy can be defined as follows:

$$\begin{gathered} Instantiate(\Pi_{Dave}^{\langle \_,\_\rangle}, \_, \_, \_, \_, S) : S.Collusion.count > 5 \\ \wedge\ \Pi_{Dave}^{\langle \_,\_\rangle}.reputation < 0.8 \\ \leftarrow UpdateCriterion(\Pi_{Dave}^{\langle \_,\_\rangle}, reputation, 0.8, \_) \end{gathered}$$

This meta-policy states that whenever the amount of reputation collusion exceeds the normal ratio, the reputation value will be changed to 0.8 and its weight will be reduced to its minimal value. The policy update is performed using the adaptation operation *UpdateCriteria*. This operation is a part of a set of adaptation operations that we presented in the previous section.

7.4.2 Trust Negotiation

Traditionally, *negotiation* was used in trust management to avoid unnecessary credentials discolour and to prevent the untrusted agent from accessing the sensitive information conveyed by these credentials (cf. Section 3.2.4). Based on that, several well-established negotiation models have been proposed. In our thesis, we propose a new *automated trust negotiation strategy* that, in addition to the focused disclosure of credentials, allows agents to adapt their policies based on alternative disclosures proposed by their interlocutors. To that aim we have made the choice to address the negotiation issue from a game theory perspective. In trust negotiation, game theory can be applied to address two main concerns [Parsons and Wooldridge, 2002]:

1. the design of *negotiation protocols* that will govern the interactions between negotiating participants,

2. the design of *optimal strategies* that agent can use while negotiating.

In the remainder of this section, we first define the protocol that our negotiation process is built upon, then we present how we used the meta-policies we defined above to specify negotiation strategies based on game theory foundations.

7.4.2.1 Trust Negotiation Protocol

The trust negotiation protocol TN aims at enabling the interacting agents to construct offers and counter-offers and communicate about them in a structured manner [Yu et al., 2001]. Concretely, agents negotiate by sending messages. With respect to that, a negotiation protocol defines the type and the ordering of the information conveyed by these messages. The type of a message is determined by the utterance used in it. For instance, when the `propose` utterance is used in a message, it specifies that the concerned message represent an offer in the negotiation dialogue. The set of utterances we conceived for the trust negotiation process represent an extension to the agent communication language $\mathcal{L}$ (cf. Section 5.6) that we defined by :

$$\mathcal{L} = \langle Protocol, Utterance, replay\rangle$$

We specify here the negotiation utterances, denoted, $\mathcal{U}_n$, which represent an extension of the utterances already included in the set $\mathcal{L}.Utterance$. $\mathcal{U}_n$ is defined as follows: $\mathcal{U}_n$ = {`START-NEGOTIATION`, `REFUSE-NEGOTIATION`, `QUERY-IF`, `CONFIRM`, `DISCONFIRM`, `PROPOSE`, `ACCEPT-PROPOSAL`, `REJECT-PROPOSAL`, `END-NEGOTIATION`} (inspired from [Seamons et al., 2002] [Winsborough and Li, 2006] and [Bhargav-Spantzel et al., 2007]).

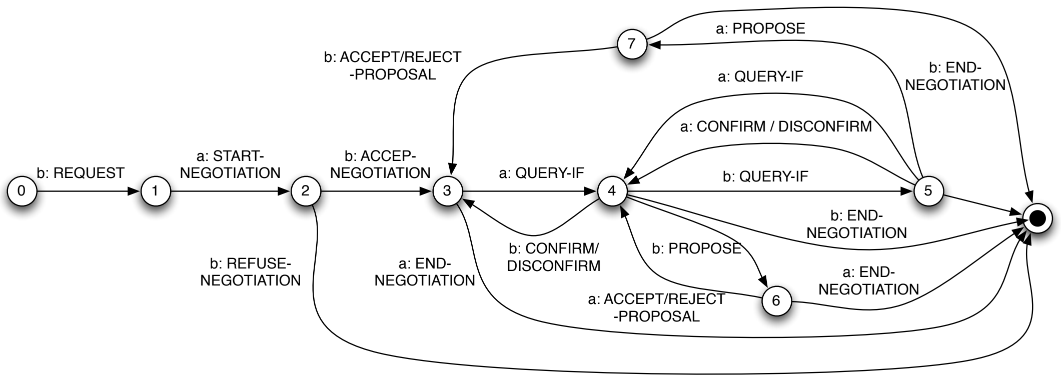


Figure 7.4 – The ASC-TMS trust negotiation protocol TN.

Starting from this new set of utterances, we redefined the function $L.replay$ so that it can map each utterance used during the negotiation to the set of utterance from $\mathcal{U}_n$ that an agent can use to reply to it. Here again, the determination of the $L.replay$ is made easy by turning the $\mathcal{U}_n$ set into a Finite State Diagram. Figure 7.4 illustrates how the utterances set $\mathcal{U}_n$ can be used to build the negotiation protocol to be used by an agent playing the role of resource controller.

Example 7.4.2 *Now let us consider the scenario described in Example 7.1.1.2. In this example, Alice wants to use the resource provided by Bob but she is not willing to loose her privacy by disclosing the private information contained in her* identity credential. *In such situation,* *trust negotiation is the unique solution that could lead Alice and Bob agree on what each party is willing to disclose to make there interaction successful.*

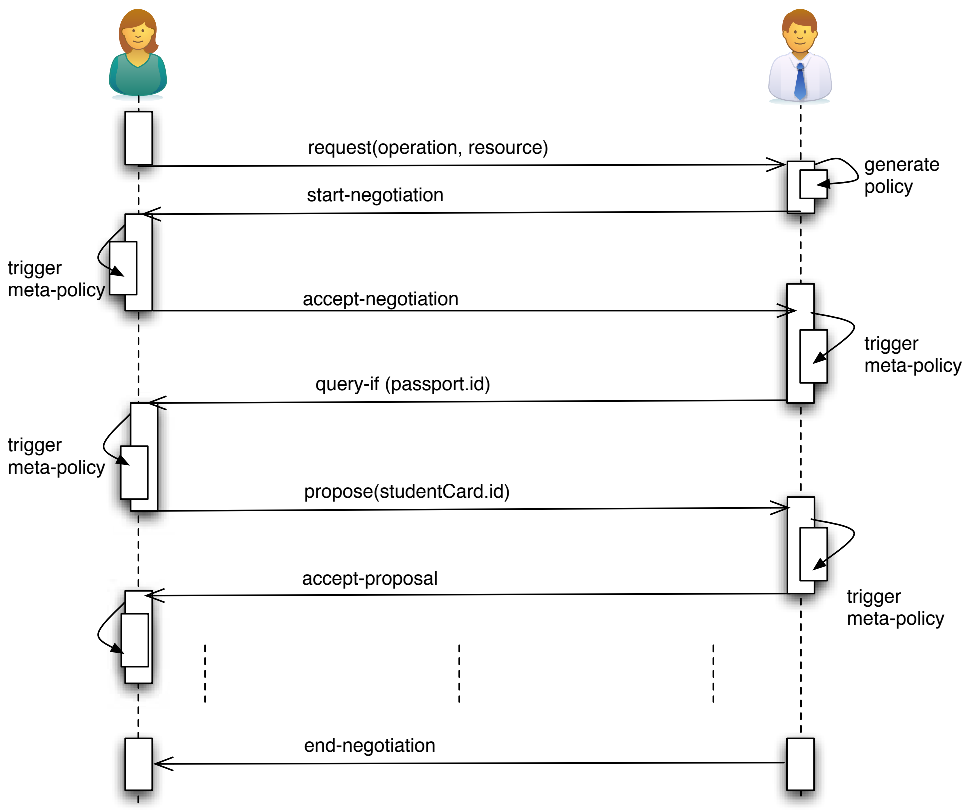


Figure 7.5 – A sample execution of the ASC-TMS trust negotiation protocol.

Figure 7.5 illustrates how the trust negotiation protocol we proposed above can help *Alice* and *Bob* exchange offers and counter-offers to reach (if possible) an agreement. In this example, the policy used by *Bob* requires a *passport* credential. However, *Alice* is not willing to disclose such a sensitive credential. Instead, *Alice* proposes to provide her student card which proves that she is an adult. Here *Bob* can either accept *Alice* proposal and query the next credential. He can also request another credential (e.g. social security card) or he can reject *Alice*'s proposal and quit the negotiation.

The negotiation protocol we presented above defines the moves that an agent is allowed to perform during a negotiation. However, each agent participating in a negotiation might have different requirements for the decisions he will make during the negotiation (e.g. how much and which credentials he is willing to disclose). For such decisions, each agent relies on its trust negotiation strategies. Now we define how the negotiating agents can reason strategically to make his negotiation successful. We call this strategy the *adaptive negotiation strategy*.

7.4.2.2 The Adaptive Negotiation Strategy

We present in this section the *adaptive negotiation strategy* that ASC-TMS relies on during the negotiation. This strategy is used by the requester and the controller for different reasons within the same negotiation. The objective of the controller is to maximise the guarantees and minimise the risk of the decisions he makes. To that aim, the strategy used by the controller will always try to keep the conditions stated in its policy unchanged. This issue is particularly important since this policy may contain part of the conditions stated by the collective policy (cf. Section 7.5.2). So, changing this policy may lead the agent to violate the compliance norm (cf. Definition 6.8.1) and thus exposes him to *sanctions*. In contrast, the objective of the requester is to prevent privacy breaches by minimising the number of credentials he discloses during the negotiation.

In this thesis, we propose an adaptive negotiation strategy that is inspired from game theory. To that aim, we model the negotiation as an extensive game in which we assume that agents' decisions are always rational. The game is played by two agents $\{controller, requester\} \in \mathcal{A}$ each having a set of alternative moves with respect to the protocol defined above. The moves an agent can make represent offers and counter-offers that are made based an *expected utility function*.

Definition 77 (Expected Utility Function) *The expected utility function determines the potential outcome of each move in the negotiation process. Let us consider a negotiation in which the requester wants to update a resource controlled by a requester. We assume that $\{x_1, \ldots, x_n\}$ is the set of credentials required by the policy of the controller, while $\{y_1, \ldots, y_m\}$ is the set of credentials that unlock the release of credentials from $\{x_1, \ldots, x_n\}$. As the controller and the requester have different objectives in the negotiation process, they use different utility functions, defined as hereafter.*

The expected utility function of a controller is defined as follows:

$$u_c = (O^{\langle update,r \rangle}_{controller} + \sum(x_i.\nu)) - (r.\varsigma + r.\nu + sanctions) \tag{7.3}$$

where

- $O^{\langle update,r \rangle}_{controller}$ *is the benefit that the resource controller can get if he accepts the request,*
- $\sum(x_i.\nu)$ *is the value of the credentials provided by the requester,*
- $r.\varsigma$ *is the sensitivity level of the requested resource,*
- $r.\nu$ *is the value of the requested resource,*
- *sanctions is the cost of the sanctions if the action violates a collective policy.*

The expected utility function of a requester is defined as follows:

$$u_r = (O^{\langle update,r \rangle}_{requester} + \sum(x_i.\nu)) - \sum(y_i.\varsigma) \tag{7.4}$$

where

- $O_{requester}^{\langle update, r\rangle}$ *is the benefit this agent will gain if his request was accepted,*
- $\sum(y_i.\nu)$ *is the value of the credentials provided by the controller,*
- $\sum(x_i.\varsigma)$ *is the sum of the sensitivity of the credentials required by the controller.*

Based on the expected utility function used by the controller (cf. Equation 7.3) and the one used by the requester (cf. Equation 7.4) we are able to define preferences on each move these agents are able to perform in the game. So for each utterance provided by the $L.replay$ function, the agent are able to compute the expected utility and build consequently a *negotiation game tree* based on which they will decide. For example, Figure 7.6 depicts a negotiation game played by a *controller* and a *requester* about the disclosure of the *identity* credential.

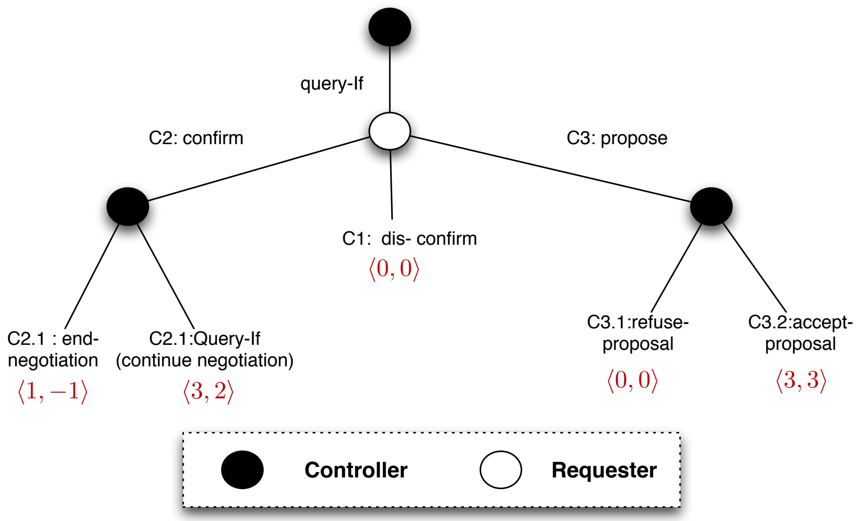


Figure 7.6 – The negotiation extensive game tree between a controller and a requester.

The game starts with the *controller* that the *requester* whether he is willing to disclose a particular credential. For illustration, we suppose that the controller requires that the requester provides a *passport credential* in order to prove that he is an adult. Once the *requester* receives the $query - if$ message asking him to disclose a *private credential* (i.e. a sensitive credential), he invokes the $\mathcal{L}.replay$ function to retrieve the moves he can make in this game and decides which move to select based on its strategy. In the game depicted in Figure 7.6 the requester may $refuse$ the negotiation (Case C1); he can also *accept* to disclose the required credential or he may *propose* another credential which is less sensitive than the passport and that certifies the same property (e.g. a student card).

Now it is the *controller*'s turn. He can *accept* the proposal (Case C2.1), $refuse$ the proposal (Case C3.2), continue the negotiation with another $query - if$ (Case C2.1) or to *end* the negotiation (having the credential but without delivering the resource) (Case C2.2). In this scenario, both the *controller* and the *requester* would agree on.

The representation of extensive games into logic has been investigated in many works [van Benthem, 2003, van Benthem et al., 2011, van Benthem, 2001, Harrenstein et al., 2002, Parikh, 1985]. We get inspired from these works to translate the above negotiation game tree into meta-policies. In the following, we illustrate the situation in which the game leads the *controller* to relax its policy by accepting the proposal of the *requester*.

$$\begin{aligned}
&propose(TrustFactor, Proposal, Request) : step(S)\\
&getUtility(\texttt{propose}, TrustFactor, Proposal, Request, S, PayOff)\\
&\wedge isMaxUtility(PayOff, Request, S) \leftarrow policy(Policy, Request);\\
&\texttt{accept-proposal}(Request, S);\\
&UpdateCriteria(Policy, TrustFactor, proposal)
\end{aligned} \tag{7.5}$$

The above meta-policy is triggered each time a `propose` utterance is received. Here, the meta-policy is a generic policy that will systematically update the active policy (the one associated to the particular request) with the received proposal if the expected utility of this proposition outperforms all alternatives the *controller* can perform (i.e. `reject-proposal` and `end-negotiation`). To that aim the meta-policy makes use of two internal actions in which the expected utility of the next move is computed (i.e. *getUtility*). Then this value ($PayOff$) is compared to the utility of the other alternatives at the same step s (i.e. *isMaxUtility*).

7.5 Social-Compliance: Adaptation to Social-Context

We presented in Chapter 4 how the social impact theory can help the description and understanding of the process of influence that affects individual's decisions and behaviours within virtual communities (cf. *majority influence* in Section 4.2.1). It describes also the reciprocal mechanism by means of which individuals can modify the decisions and behaviours of other individuals with whom they interact (cf. *minority influence* in Section 4.2.2). In this thesis, we build on these theories and apply them to the trust management. First, agents make use of *policies combination* to build their collective policies. Then, each agent use *policies integration* to make its individual policy comply/or not with the collective policy. Finally, when collective policies need to be updated, agents use *evolution* mechanisms to bring changes to the collective policies they are subject to. The following sections describes how *combination*, *integration* and *evolution* are performed in ASC-TMS.

7.5.1 Combination

In Section 4.1.1, we saw that *combination* of policies refers to the process, proposed by Bertino and colleagues in [Li et al., 2009], of deriving a decision based on decisions derived from several policies using a combination heuristic. In our proposal, we use this concept in reference to the process of building a policy from several policies. The policies to be combined are individual policies of the members of the community. To achieve this combination, we draw inspiration from the works presented in Section 4.1.2.1 to specify the combination Algorithm 1 sketched hereafter.

Algorithm 1: combine($\Pi, Community, Pattern, Heuristic$)

```
P ← ∅ ; P.Issuer ← Community; P.Pattern ← Pattern;
foreach $\pi_i \in \Pi$ do
    foreach $tc_i \in \pi_i$ do
        if $\exists\, tc_j \in P, tc_i.f = tc_j.f$ and $tc_i.op = tc_j.op$ then
            $P \leftarrow P/\{tc_j\}$
            if $Heuristic == h1$ then
                $P \leftarrow MostRestrictive(tc_i, tc_j)$
                ComputeWeights();
            end
            if $Heuristic == h2$ then
                $P \leftarrow LeastRestrictive(tc_i, tc_j)$
                ComputeWeights()
            end
            $IP \leftarrow IP/\{tc_i\}$
        end
        else
            $P \leftarrow tc_i$
            ComputeWeights()
            $IP \leftarrow IP/\{tc_i\}$
        end
    end
end
return $P$
```

Informally, the algorithm generates the combined policy by making the union of the trust criteria contained in the policies to be combined. When two criteria are stated using the same trust factor (e.g. identity or reputation), the algorithm makes use of particular heuristics to merge these two criteria.

Two heuristics have been specified: $h1$ and $h2$. $h1$ selects the criterion that is the most restrictive, while $h2$ selects the least restrictive criterion. Thanks to the trust factors ontology, a criterion tc_i is considered as more restrictive than another criterion tc_j iff:

$$\exists\, ts_i \in Pr, ts_i \models tc_i \implies ts_i \not\models tc_j \tag{7.6}$$

where ts_i is a trust statement and $ts_i \models tc_i$ means that ts_i satisfies the trust criteria tc_i.

The heuristic determines also whether the criterion should be mandatory or optional. If a mandatory trust criterion has to be combined with an optional one, $h1$ will make the combined criterion mandatory, while $h2$ will make it optional.

Finally, the weight of each trust criterion is computed. Computing weight aims at preserving the proportion of the criterion in the policy. To that aim, the weights of all criterion are summed over the same trust factor.

Now that we presented how the *combine* adaptation operation is achieve, we present in what follows an example of the meta-policies that an agent a_i uses to trigger the collective policy creation. First, a_i initiates the combination process by sending a message to all the members of the community. In this message, the agent asks the members of its community to start the combination. To that aim, the agent makes use of the following meta-policy

$$\begin{aligned} &createCollectivePolicy(c, \langle x, y\rangle) : \forall \pi_i \in c.\Pi, \pi_i.Pattern \neq \langle x, y\rangle) \\ &\leftarrow \forall a_j \in c.M, \langle a_i, inform\_if, a_j, combine(\pi, \langle x, y\rangle, h1))\rangle \end{aligned} \tag{7.7}$$

With the above meta-policy, the agent a_i (i.e. the combination initiator) informs the members of its community that he is starting the creation of a collective policy for a particular pattern $\langle x, y\rangle$ and that he is willing to use the heuristic $h1$.

Now, each agent receiving the message from a_i has to decide whether he wants to be involved in building the collective policy. To proceed, the recipient agents make use of the following meta-policy.

$$\begin{aligned} &combine(\pi, \langle x, y\rangle, heuristic) : \forall \pi_i \in c.\Pi, \pi_i.Pattern \neq \langle x, y\rangle) \\ &\leftarrow \forall a_j \in c.M, \langle a_i, inform, a_j, combine(\pi, \langle x, y\rangle, h1))\rangle \end{aligned} \tag{7.8}$$

In this meta-policy, the agent first checks whether a collective policy that handles the same pattern does not exist. If it is the case, the agent sends to the other members of the community its individual policy for that particular pattern along with the heuristic he wants to use. This choice of the heuristic is made based on the agent characterising properties (cf. Definition 24 in Section 5.4). When the agents prefers to use distinct heuristics, the conflict is resolved by using the heuristic stated by the majority. After a certain period (e.g. fixed with a timer), all members of the community should have sent their policies. Now each agent makes use of the forthcoming meta-policy to combine all received policies using the most popular combination heuristic.

$$\begin{aligned} &\mathit{combinePolicies}(\Pi', c, \langle x, y\rangle, \mathit{Heuristic}) : \\ &\leftarrow \mathit{combine}(\Pi, c, \mathit{heuristic}) \end{aligned} \tag{7.9}$$

In this meta-policy, the set Π' represents the policies received by the agent a_i, c is the community to which a_i belongs, $\langle x, y\rangle$ is the pattern for which the collective policy is going to be built and *heuristic* is the heuristic to be used during the combination process. In Section 9.2.1.2, we will illustrate this combination process with more details.

7.5.2 Integration

In this section, we present how we use policies *integration* to make individual policies comply with collective ones. The integration process is fairly similar to one used for combination except that integration considers only two policies. In addition, the integration consider two additional heuristics which offers to the agent a better control on the outcome of the integration process. The heuristics are used during the integration process to answer two important questions:

1. What to do when the trust criteria of the policies to be integrated are stated using the same trust factor (in policies that converge, shuffle, extend and restrict each other):
    - *keep the most restrictive criterion?*
    - *keep the least restrictive criterion?*
2. What to do with trust criteria that are stated using a trust factor that is not used by both policies (in policies that shuffle, extend, restrict and diverge each other):
    - *keep all trust criteria?*
    - *keep those stated in one (particular) policy?*
    - *do not keep any criteria that is not stated in both policies?*

The above questions suggest the existence of six ways in which the integration could be performed. Among these ways, we discarded the ones which do not keep trust criteria that are not stated by both policies. We can easily prove that these ways are not relevant to the integration as they can result in an integrated policy that rejects (resp. accepts) request that both the individual policy and the collective policy would have accepted (resp. rejected). Thus, ASC-TMS does not have any reason to provide such integration. The four remaining ways have been implemented as heuristics into the integration algorithm sketched below.

The behaviour of the four heuristics can be summarised as follows:

1. Heuristic 1 ($H1$): the resulting policy will be at least as restrictive as the most restrictive policy.

Algorithm 2: integrate($\pi_1, \pi_2, Pattern, Heuristic$)

```
1  $P \leftarrow \emptyset$ ;
2  foreach $tc_i \in \pi_1$ do
3      if $\exists\, tc_j \in \pi_2 / tc_i.f == tc_j.f$ and $tc_i.op == tc_j.op$ then
4          if $(Heuristic == h1)$ or $(Heuristic == h3)$ then
5              $P \leftarrow MostRestrictive(tc_i, tc_j)$
6          end
7          if $(Heuristic == h2)$ or $(Heuristic == h4)$ then
8              $P \leftarrow LeastRestrictive(tc_i, tc_j)$
9          end
10     end
11     else
12         if $Heuristic \neq h4$ then
13             $P \leftarrow tc_i$
14             $IP \leftarrow IP/\{tc_i\}$
15         end
16     end
17 end
18 ComputeWeights()
19 return $P$
```

2. Heuristic 2 ($H2$): the resulting policy will be at most as restrictive as the least restrictive policy.

3. Heuristic 3 ($H3$): the resulting policy will be at least as as restrictive as the selected policy.

4. Heuristic 4 ($H4$): the resulting policy will be at most as restrictive as the selected policy.

The idea behind $H1$ is to build policies that guarantee that the agent will never accept requests that would be denied by both policies. At the opposite, $H2$ builds policies that guarantee that the agent, using the integrated policy, will never deny a policy that would accepted by both policies. $H3$ and $H4$ give respectively the priority to one policy: $H3$ guarantees that the integrated policy never accepts request that the policy having the priority would deny, while $H4$ guarantees that the integrated policy never denies a request that the policy having the priority would accept.

Our objective is not to specify when to select a particular heuristic but instead we propose a toolkit for trust policies integration in order to allow agents to make *socially-compliant trust decisions*. In the following, we present an example of a meta-policy in which we used findings of social impact theory (cf. Section 4.2) about how the community context influences the way the agent complies with the collective policy (cf. Section 8.3.1.2 for more details about how this meta-policy has been implemented).

$$\begin{aligned}&integratePolicy(\pi, c, \langle x, y\rangle, Heuristic) :\\&population(c, Po)\&((Po > 3)\&(Po < 20))\\&\leftarrow integrate(\pi_c, \pi_a, \langle x, y\rangle, Heuristic).\end{aligned} \tag{7.10}$$

The above meta-policy makes the agent comply with the collective policy using the heuristic $H3$ whenever the population of this community satisfies the compliance conditions as states by Latané in [Latané, 1981] (cf. Section 4.2). Here the agent is called *conditionally* compliant. In other words, its compliance is conditioned by the community context. Works on sociology demonstrated that the attitude of an individual with respect to compliance fall into three categories: *compliant*, *deviant* and *conditionally compliant*. *Deviants* are agents that refuse the collective policy and consequently will always neglect it. In contrast, *compliant* agent will blindly follow comply the collective policy each to avoid being in conflict with the other community fellows. We will illustrate in Section 8.3.1.2 the meta-policies used by the agents to exhibit each type of compliance.

7.5.3 Evolution

We presented in Section 7.5.1 how collective policies are built. But for many reasons (as illustrated in Section 7.1), a collective policy can be brought to become obsolete. In this section, we present our proposition to make the members of a community to adapt their collective policies. The idea in this section is to be able to reproduce the *minority influence* phenomena in terms of trust management. To that aim, we addressed collective policies evolution from the *social choice theory* perspective. Consequently, we modelled the evolution of collective policies as a voting process that helps distributed agents to produce consensus about when and how their collective policies can be adapted.

The evolution process is specified using three kinds of meta-policies. The first kind of meta-policies are responsible of detecting the situations necessitating the adaptation of collective policy and to trigger the adaptation. These meta-policies are stated as follows:

$$\begin{aligned}&evolution(\pi^{Pattern}, c) : k_0, \ldots, k_n \leftarrow \forall a_i \in c.A\\&\langle a_j.\varepsilon, inform, a_i.\varepsilon, cfe(Pattern, Adaptation, TrustFactor)\rangle\end{aligned} \tag{7.11}$$

In this meta-policy, the context conditions $k_0, \ldots, k_n$ captures the context in which the collective policy has to be adapted. This context can be generic (e.g. performed systematically by a new member) or specific to a particular situation (e.g. after several failures in negotiation because of the same trust criterion). If the context holds, the initiator agent (i.e. a_j) informs its community fellows ($\forall a_i \in c.A$) about the adaptation he wants to perform on the collective policy (i.e. $(Pattern, Adaptation, TrustFactor)$).

When a members of the community receives the call for evolution (i.e. $cfe(Pattern, Adaptation, TrustFactor)$), the corresponding meta-policy is triggered to evaluate whether the agent personal conditions to accept such proposal are met. Such meta-policy is stated as follows:

$$\begin{aligned} &cfe(Pattern, Adaptation, TrustFactor) : k_0, ..., k_m \leftarrow \forall a_i \in c.A \\ &\langle a_j.\varepsilon, inform, a_i.\varepsilon, vote(agree, Pattern, Adaptation, TrustFactor)\rangle \\ &countVotes(Pattern, Adaptation, TrustFactor) \end{aligned} \tag{7.12}$$

With the above meta-policy, the agent receiving the call for evolution will accept the call and send an agreement statement that every agent will receive. Of course, the agreement of an agent may influence the agreement of another agent but we do not consider/handle such influence in our thesis. If the evolution conditions stated by the agent do not hold, the agent will disagree and notify other agents in consequence. Worth noting that the set of conditions used by the agent a_i to trigger the evolution process are different from those used by each agent to accept or not such proposition.

Once the agents have finished voting[2], the evolution initiator and every member of the community count the votes that agree with the proposition and those which do not. Then each agent makes use of a voting system to determine whether the adaptation has been accepted by the community members or not. This can be done via the following meta-policy:

$$\begin{aligned} &countVotes(Pattern, Adaptation, TrustFactor) : \\ &count(vote(\_, Pattern, Adaptation, TrustFactor)) \geq (|c.A| * 2/3) \\ &\wedge (count(vote(disagree, Pattern, Adaptation, TrustFactor)) > \\ &count(vote(agree, Pattern, Adaptation, TrustFactor))) \\ &\leftarrow makeEvolution(Pattern, Adaptation, TrustFactor) \end{aligned} \tag{7.13}$$

This meta-policy counts the total votes and compares it with the community population. If two-thirds of the population have voted, the vote is considered to be valid. When the vote is valid and the agreeing voters are more outnumber the disagreeing voters, the collective policy is adapted. When the proposition to adapt the collective policy is accepted, each agent (those who agreed and those who did not) makes use of the following meta-policy to make the adaptation of the collective policy effective.

$$\begin{aligned} &makeEvolution(Pattern, Adaptation, TrustFactor) : policy(P, Pattern) \\ &\wedge P.Issuer == c \wedge Adaptation == \texttt{"restrict"} \\ &\leftarrow RestrictCriterion(P, TrustFactor) \end{aligned} \tag{7.14}$$

[2]We assume that agents make use of a timer to time the voting process.

This last meta-policy selects the collective policy that corresponds to the pattern concerned by the adaptation. Then the adaptation operation is performed on the policy. Of course, this meta-policy applies only to the evolutions that makes the collective policy more restrictive. Similarly, other meta-policies are defined for each adaptation operation.

7.6 Conclusion

In this chapter, we presented our proposition for the automation of policies adaptation. The adaptation of a policy is made with respect to two kinds of contexts, namely *business-context* and *social-context*. The business-context relates to the individual and the object of the interaction, while the social-context is materialised by the community to which the agent belongs to.

We considered *social-awareness* as an adaptation process in which policies of each member are *adapted* to *specify* (Objective O1.1) and *enforce* (Objective O1.2) collective policies. Indeed, individual policies are *adapted* and *weaved* to enable the *specification* of the collective policy. This process mimics the model reported by Sherif in the *autokinetic effect*. Likewise, the policy used by an individual is *adapted* to make it compliant with the collective policy of its community. Here we implement *Asch* [Asch, 1955] theory in which he proved that an individual changes its opinion to make it comply with the one of the group. We also get inspiration from *social influence theory* to propose a mechanism that makes VCs members trigger evolution (i.e. adaptation) on their collective policies (Objective O2.2).

So we draw inspiration from the *social impact theory* to put in place a *micro-micro* loop for trust policies adaptation. On the one hand, the *micro-macro* adaptation represents the mechanisms we proposed to specify (cf. Section 7.5.1) and enforce (cf. Section 7.5.3) the collective policy of a community. On the other hand, the *macro-micro* represent the mechanism that agent use to make their policies comply with the collective policy (cf. Section 7.5.2).

Also, we presented *adaptation* mechanisms used by VCs members to *adapt* their policies in reactions to changes in their environment (cf. Section 7.4.1) and those imposed by their interlocutor (cf. Section 7.4.2).

Finally, we stress two important remarks with respect to the policies evolution process we proposed above. First, we intentionally omitted to describe the voting scheme used by the agents. In our proposal, we do not impose neither a voting scheme nor a voting protocol. The only requirement we make is that all agents use the same voting scheme and protocol.

7.7 French Summary

Dans le chapitre précédent, nous avons présenté notre modèle de gestion de la confiance. Ce modèle comportait une ontologies des facteurs de confiance, un langage de spécification des politiques de confiance et d'un schéma d'évaluation de ces politiques. Les politiques sont utilisés dans notre modèle pour exprimer les conditions qu'un individu ou une communauté estiment être nécessaires pour la prise de décision de la confiance. Dans ce chapitre, nous allons présenter comment nous avons utilisé ce langage de politique pour faire en sorte que notre gestion de la confiance soit sensible à l'environnement social et métier des communautés virtuelles. Pour cela, nous allons d'abord justifier notre choix de considérer les objectifs de la thèse comme une problématique d'adaptation. Ensuite, nous allons classifier les différents catégories d'adaptations que nous adressons dans ce manuscrit. Enfin, nous présenterons les mécanismes d'adaptation que nous proposons afin d'adapter les politiques aux deux types de contexte (social en environnemental) ciblés.

7.7.1 Types d'adaptation

Un des points fondamentaux de cette thèse réside dans le fait que nous accordons le même degré d'importance au *contexte sociale* et au *contexte environnemental*. Cela résulte de notre analyse des travaux réalisés en informatique que ce soit sur les systèmes de gestion de la confiance (cf. Chapitre 3) ou les systèmes multi-agent (cf. Chapitre 4) mais également et sur tout de notre étude des travaux réalisés en sociologie (cf. Chapitre 4).

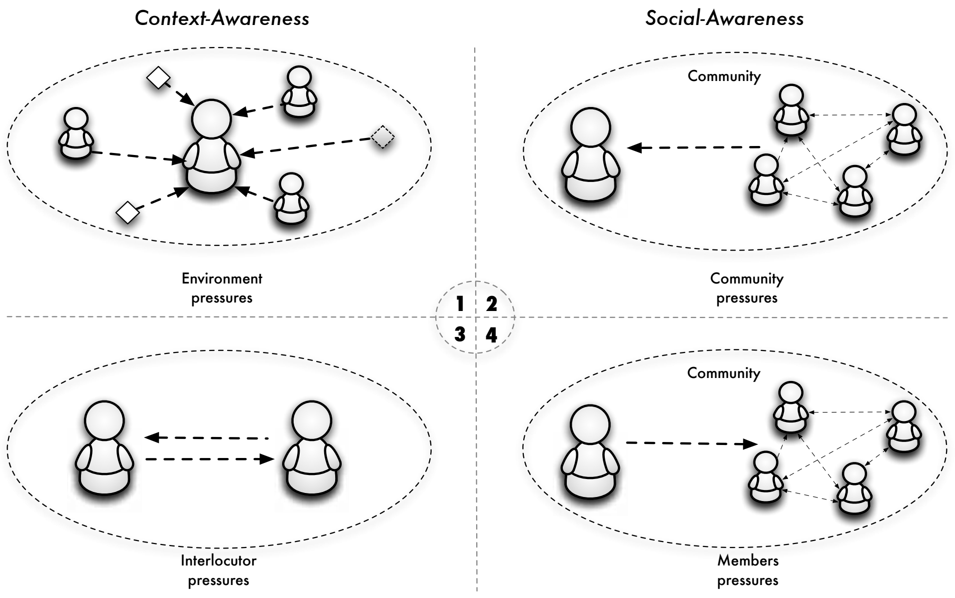


Figure 7.7 – Les pressions exercées sur les politiques.

Ainsi, sous l'éclairage de ces études, nous avons décidé de considérer *la sensibilité au contexte sociale* et *la sensibilité au contexte environnemental* comme étant un seul et unique problème d'adaptation des politiques. Dans la Figure 7.7, nous caractérisons la notion de contexte et analysons les différents types de pressions auxquelles sont sujets les membres, et donc les politiques au sein des communautés virtuelles. On identifies principalement deux types de pressions : les pressions sociales (c.f. Figure 7.7-2 et 7.7-4) et les pressions environnementales (cf. Figure 7.7-1 et 7.7-3). Les pressions sociales reflètent la nature sociale des communautés virtuelles que nous avions souligné dans l'introduction du manuscrit tandis que les pressions environnementales se réfèrent aux aspects dynamiques de ces communautés.

Chacune de ces pressions affects les politiques utilisées dans la communauté (que ce soit individuelle ou collective) et de ce fait nécessitent un mécanisme d'adaptation dédié. Dans ce qui suit, nous motivons brièvement la nécessite de chaque type d'adaptation.

7.7.1.1 Adaptation métier

Lors d'une interaction autour de la confiance, la décision de faire confiance est souvent influencée par le contexte métier dans lequel cette décision a été prise. Ce contexte est dans la plupart du temps matérialisé par les ressources qui constituent l'environnement de l'agent (cf. Figure 7.7-1) ainsi que l'agent interlocuteur (cf. Figure 7.1-3). Dans cette section, nous allons voir comment ces éléments de notre système $\mathcal{S}$ affectent les politiques de confiance et proposer en conséquence des mécanismes d'adaptation dédiés.

- **Instanciation** : Le langage de politiques que nous avons présenté dans le chapitre précédent utilisé le concept de *Pattern* pour définir le type de décision à laquelle s'applique la politique. Or, le pattern permet de spécifier une opération et une ressource/communauté particulière mais permet également de spécifier des politiques génériques qui s'appliquent à l'ensemble des ressources d'un type donné (en se basant sur le le type τ). Ce choix se justifie par le fait que l'intensité de la fréquence à laquelle les ressources sont créées au sein de communautés virtuelles. Ainsi, il est quasi impossible et assez contraignant que les membres spécifient une politique pour chaque ressource ils créent. Ainsi, il serait plus judicieux d'utiliser la même politique pour tout un ensemble de ressources. Cependant, ces politiques se doivent d'être adaptés pour coller au mieux à la réalité du monde dans lequel elles seront utilisées. De ce fait, l'*instanciation* permet ce type d'adaptation.

- **Négociation :** Le modèle de confiance que nous avons repose sur une conceptualisation asymétrique de la relation de confiance. Ainsi, pour qu'un demandeur se voit accorder l'accès à une ressource, celui-ci doit montrer ces certificats au contrôleur. Or, ces certificats peuvent contenir des informations sensibles ce qui rend le demandeur vulnérable vis-à-vis du contrôleur. Pour éviter tout risque, le contrôleur et le demandeur passent par une phase de négociation afin qu'une relation de confiance graduelle puisse s'établir. Dans ce contexte, plusieurs travaux ont été proposé et que nous avions détaillé dans le Chapitre 2. Dans notre approche, nous avons, en plus des mécanismes de négociation de base proposées dans les autres travaux, introduit la notion d'adaptation de politiques au cours de la négociation. Par exemple, considérons la situation ou le contrôleur A souhaite que le demandeur B lui fasse part de son passeport pour prouver qu'il est adulte. Or, ce dernier possède d'autres certificats (e.g., carte de sécurité sociale, carte de crédit, permis de conduire) qui sont à son sens moins sensibles que le passeport et qui permettent autant que le passeport de prouver qu'il est un adulte. Ainsi, notre approche va lui permettre de faire savoir sa volonté au contrôleur ce qui pourrait entraîner l'adaptation de la politique de ce dernier aux exigences du demandeur.

7.7.1.2 Adaptation sociale

Nous avons présente en Section 4.2 les mécanismes sociaux qui expliquent comment les normes émergent (cf. Section 4.2.1.1), comment les individus au sein de communautés avaient tendance à se conformer à ces normes (cf. Section 4.2.1.2) et comment dans certaines circonstances, ces individus réussissent à changer les normes auxquelles ils sont sujets. Dans cette section, nous analysons les pressions exercées sur les politiques dans la lumière de ces théories. Par conséquent, nous identifions les pressions *descendantes* à travers lesquels une communauté impose à ses membres une politique collective afin de les forcer à changer leurs politiques individuelles en conséquence (cf. Section 7.1.2.1), et les pressions *ascendantes* par lesquelles les politiques individuelles des membres sont utilisées pour créer et mettre à jour les politiques collectives. Cette pression ascendante est par la suite subdivisée en deux sous-pressions en référence au processus avec lequel les membres construisent leurs politiques collectives (cf. Section 7.1.4) et celui avec lequel ils la mettent à jour en cas de besoin (cf. Section 7.1.3).

- **Combinaison** : Dans notre modèle, les politiques de confiance sont spécifié à deux niveaux : *individuel* et *collectif.* Les politiques individuelles reflètent les conditions personnelles d'un individu pour ce qui concerne les décisions de confiance. Par ailleurs, les *politiques collectives* permettent aux membres d'une communauté d'harmoniser les décisions de confiance prises par chaque membre d'une manière complètement décentralisée. Ainsi, nous proposons dans cette thèse un mécanisme d'adaptation qui va permettre aux membres de la communauté de façonner une politique collective à partir de la politique individuelle de chaque membre.

- **Intégration** : L'existence d'une politique collective implique également la nécessité d'un mécanisme permettant aux membres des communautés d'en prendre connaissance et de l'utiliser. Ainsi, chaque individu doit être en mesure de considérer, à la fois, sa politique individuelles et la politique collective de sa communauté. Cette étape va l'amener à adapter la politique qu'il utilise pour qu'elle soit en conformité avec la politique collective. Ici, notre objectif est de pouvoir proposer un mécanisme d'adaptation qu'on appelle *intégration* en références aux travaux vus dans le chapitre 4.

- **Évolution** : Maintenant que l'intérêt des mécanismes d'*instanciation*, *négociation*, *com-*

binaison et *intégration*, il est légitime de se demander si les politiques collectives à leur tour doivent/peuvent être adaptées. Notre réponse est oui. Tout comme les politiques individuelles, les politiques collectives nécessitent des mécanismes d'adoption leur permettant de répondre au mieux aux exigences des membres de la communauté. Pour cela, nous présentons un mécanisme d'adaptation dédié aux politiques collectives que nous appelons évolution. Ce mécanisme va permettre à tout membre de la communauté de déclencher une reconsidération de la politique en modifiant un des critères de confiance qui la composent.

7.7.2 Adaptation des politiques

Dans cette section nous présentons comment notre système est capable de réagir aux pressions discutées plus haut et de mettre en œuvre les types d'adaptations correspondantes. L'idée ici était d'utiliser des méta-politiques d'adaptation en s'inspirant des mécanismes utilisés dans la communauté multi-agent pour mettre en œuvre l'adaptation des normes (cf. Section 4.3.3.2). En rendant ces méta-politiques explicites, il devient plus simple de s'assurer que les agents encapsulant les systèmes de gestion de la confiance utilisent constamment les politiques les plus appropriées au contexte courant.

Definition 78 (Méta-politiques) *Soit $\mathcal{M}$ l'ensemble des méta-politiques du système. Chaque méta-politiques $m \in \mathcal{M}$ est définie comme suit:*

$$m : \mathcal{I}_a \times \Pi \times \mathcal{S}^t \times \mathcal{K} \rightarrow \Lambda^\Pi$$

Où la fonction m fait correspondre une interaction $\mathcal{I}_a$ [qui est dans la plupart des cas une requête], une politique, une image du système $\mathcal{S}$ à un temps t et une expression de conditions à l'ensemble des actions qui doivent être exécutées sur la politique afin que celle-ci s'adapte à son contexte.

Dans ce qui suit, nous présentons les conditions qui sont utilisées pour déclencher les méta-politiques ainsi que les opérations d'adaptation réalisées sur les politiques.

7.7.2.1 Conditions d'adaptation

Notre système de gestion de la confiance (ASC-TMS) a été conçu afin qu'il puisse être en mesure de surveiller et raisonner sur le contexte (social et environnemental) au sein duquel l'évaluation de la confiance va avoir lieu. Cependant, il est assez difficile de faire en sorte que ce système puisse surveiller et raisonner sur l'ensemble des informations disponibles dans $\mathcal{S}$. Ainsi, nous avons recensé puis circonscrit les éléments du contexte à ceux qui ont un lien direct avec les pressions listées plus haut. Le fruit de ce travail constitue l'Ontologie du contexte Δ_{ctx} qui est en fait un sous-ensemble de l'ontologie du domaine Δ.

7.7.2.2 Opérations d'adaptation

Afin que l'adaptation des politiques soit possible, il est nécessaire de doter notre système de gestion de la confiance d'un ensemble d'opérations à travers lesquels il pourrait changer les politiques qu'il manipule. À cet effet, nous avons avons proposé une extension de l'ensemble A, l'ensemble des actions qu'un agent peut réaliser, avec l'ensemble Λ^{Π} contenant uniquement les opérations qui s'appliquent à des politiques.

Definition 79 (Opération d'adaptation) *Les opérations d'adaptation sont définies comme suit:*

$$\forall \omega_i \in \Lambda^{\Pi},\ \omega_i : 2^{\Pi} \rightarrow \Pi$$

Nous identifions deux types d'opérations: *les opérations simples* et *les opérations complexes*. Les opérations simples permettent d'ajouter ou de supprimer un critère de confiance. Elles permettent aussi de modifier la valeur et/ou le poids d'un critère existant. Par ailleurs, les opérations complexes permettent de modifier ou générer une politique à partir d'au moins deux politiques.

7.7.3 Les méta-politiques d'adaptation

Dans notre modèle, nous avons utilisé des règles événement-condition-action pour la mise en œuvre effective des méta-politiques. Ainsi, chaque méta-politique est composée de trois parties: *l'événement*, les *conditions* et les *actions* tel que c'est illustré ci-après:

$$\langle \textit{événement} \rangle : \langle \textit{conditions} \rangle \leftarrow \langle \textit{actions} \rangle$$

Ces types de méta-politiques se lisent comme suit "si l'événement a lieu dans le contexte où les conditions sont satisfaites alors les actions doivent être exécutées". Nous illustrons l'utilisation de ces méta-politiques par un exemple.

$$Instantiate(\Pi_{Bob}, \_, \_, R, \_, \_) : R.value^{t} > R.value^{t-1}$$
$$\leftarrow RestrictCriterion(\Pi_{Bob}, reputation).$$

Dans cet exemple, la méta-politique est une méta-politique d'instanciation. Elle sera utilisée pour rendre le critère sur la réputation plus restrictif si la valeur de la ressource demandée a augmentée.

7.7.4 Le processus d'adaptation

Dans la section précédente, nous avons présenté comment les méta-politiques d'adaptation sont spécifiées et comment, à partir de ces méta-politiques, les politiques sont adaptées en réponse à un contexte particulier. Désormais, il est nécessaire de présenter comme est ce que l'agent encapsulant le système de gestion de la confiance sera en mesure de coordonner les différentes actions d'adaptation. En effet, il est plus que probable que pour une décidions donnée plus d'une politique d'adaptation sera active. Ainsi, nous présentons dans cette section le cycle d'adaptation qui va permettre à chaque agent de savoir dans quel ordre déclencher les différentes adaptations depuis la sélection de la politique à sont évaluation. L'architecture décrivant ce cycle est illustrée dans la Figure 7.8.

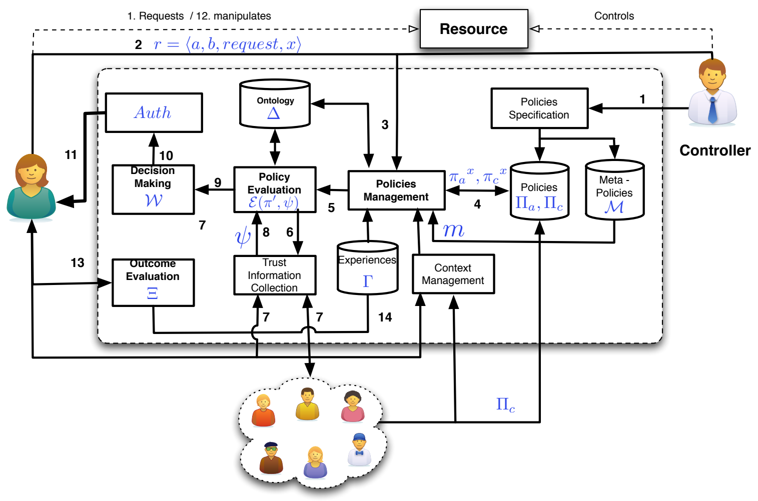


Figure 7.8 – Architecture du système de gestion de la confiance ASC-TMS

Comme illustré dans la figure 7.8, le répertoire des politiques contient à la fois les politiques individuelles (i.e. Π_a) et les politiques collectives (i.e. Π_c). Nous avons également les méta-politiques qui sont stockés par le système dans un répertoire dédie. L'utilisateur dans un premier temps va donc spécifier à son système ses politiques individuelles ainsi que les méta-politiques. Le système aura ensuite pour objectif de récupérer les politiques collectives si celles-ci existent. Une fois la requête reçu (2), le système va tout d'abord sélectionner les politiques individuelles et collectives. Ces politiques vont servir à générer une politique adaptée qui sera évaluée. Enfin, l'évaluation sera utilisée pour prendre une décision. Cette décision servira ensuite à établir une autorisation.

Dans cette architecture, l'ensemble des mécanismes d'adaptation est géré par le module de gestion des politiques (i.e., Policies Management). Ainsi, l'objectif de ce module est d'automatiser entièrement les différentes étapes d'adaptation introduites plus haut. Ainsi, nous avons défini le fonctionnement du module en nous basons sur un cycle inspiré des travaux sur le raisonnement à base de cas [Aamodt and Plaza, 1994].

Part IV

Implementation, Application and Experimentation of the ASC-TMS

Chapter 8

ASC-TMS Implementation

In this chapter, we provide technical insights into the implementation of the agent-based virtual community and the ASC-TMS we deployed on it. To that aim, we introduce in Section 8.1 the multi-agent programming platform we used to implement and deploy our ASC-TMS. In Section 8.2 we present the general architecture of the *JaCaMo* based virtual community framework. Then in Section 8.3, we delve into the details of our implementation and illustrate different aspects of the implementation with codes. Finally, we conclude this chapter in Section 8.4 with a summary and a discussion on the limits of our implementation.

Before we delve into implementation details, however, we introduce in the next chapter *JaCaMo*, the multi-agent programming platform we used to implement ASC-TMS.

8.1 JaCaMo Framework

The *JaCaMo* is a multi-agent programming platform that originates from the integration of three existing platforms, namely *Jason*, *CArtAgO* and $\mathcal{M}$*oise*. So programming programming in *JaCaMo* is: (a) programming agents and agent-agent interactions based on the Jason agent-programming platform, (b) programming environments and agent-environment interactions using the *JaCaMo* platform, and (c) programming multi-agent systems (i.e. organisations) and agent-organisation interactions using the $\mathcal{M}$*oise* framework.

Our main motivation in using *JaCaMo* is the seamless and synergistic integration of all the dimensions of the system model we framed in Chapter 5. The full description of the *JaCaMo* platform and the integration of the three multi-agent programming technologies is out of the scope of this thesis. We refer interested readers to the existing literature or to the project web site[1]. We present, however, in the following a brief description of the three components of this platform.

8.1.1 Programming Agents with Jason

Jason is a java-based extension of AgentSpeak [Rao and Georgeff, 1995]. AgentSpeak(L) represents a computational formalisation of the BDI model. Figure 8.1 shows the architecture of a Jason agent as well as the component functions that are executed during the reasoning cycle. In

[1]`http://jacamo.sourceforge.net/`

this architecture, rectangles represent data structures that contain the main components that determine the agent state (i.e. belief base, set of events, plans library and set of intentions), while rounder boxes, diamonds and circles represent functions used in the reasoning cycle.

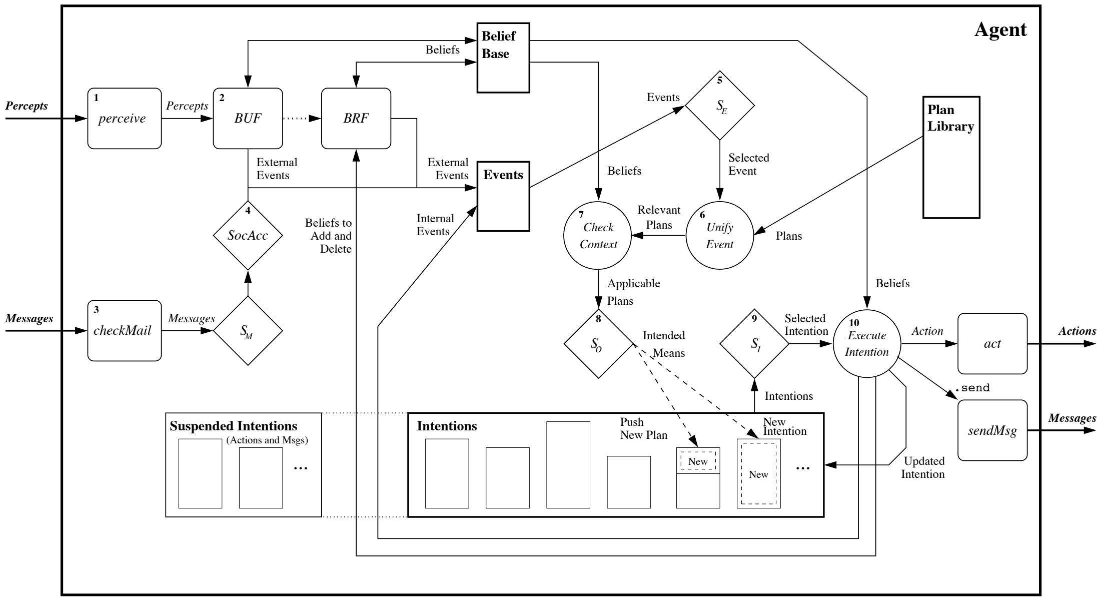


Figure 8.1 – The execution cycle of the AgentSpeak Architecture

Explaining each of the 10 steps of the reasoning cycle used by Jason agents is out of the scope of this thesis as we do not make any contribution on this cycle. Instead, we are more interested in defining the five core elements of a Jason agents that we will rely upon in the next section to describe our implementation.

- ***beliefs*** represent the information known by the agent that it regularly updates based on its perception. A Belief is represented in Jason as a ground (first-order) predicate called belief atom and can be combined to form beliefs using logic connectives of negation (not) and conjunction (∧ or &). For example, owner(resource1) is a belief atom, and **not** owner(resource1) & owner(resource2) is a belief [Meneguzzi, 2009] .

- ***events*** represent the notification an agent receives when changes occur in the environment of in one of its data structures. These events are used to trigger the adoption of a plan by an agent if the situation for which this plan applies holds. Jason propose four types of events corresponding to the addition, deletion of beliefs and plans. The plus (+) sign corresponds to the addition of a belief or a goal, whereas the minus (-) sign corresponds to the deletion of a belief or a goal.

- ***plans*** represent the behaviours of the agent. Plans specify the means (it terms of actions or other plans) for achieving a particular goal whenever certain events occur and under a certain circumstance.Each plan has three distinct parts: the triggering event, the context and the body. The three parts are separated by ":" and "$< -$" as follows:

$$triggering - event : context < -body. \tag{8.1}$$

- ***intentions*** are instantiated plans that are currently adopted by the agent. When the agent adopts a plan, that means that the agent commits to execute plan to completion. Executing the plan means executing each interaction contained in the plan and those contained in the sub-plans of the executed plan.

- ***goals*** correspond to a desired state the agent wants to reach by triggering the associated plan. Two types of goals can be used in Jason; *achievement goals* are represented by a predicate that is preceded by an exclamation mark (e.g. `!adaptPolicy(Policy)`), and *test goals* are represented by a predicate that is preceded by a question mark (e.g. `?owner(resource2)`). Test goals are used by the agent to test if a belief is true, whereas achievement goals are used to achieve a certain state of affairs [Bordini et al., 2007].

Now that we introduced the main components of the above Jason architecture, we briefly describe the reasoning cycle used by Jason agents. We summarise this cycle into three phases:

1. The agent senses (or **percepts**) the environment for new percepts (1). Percepts are symbolic representation of particular property of the current state of the environment [Bordini et al., 2007]. Each percept can represent the adding, update or deletion of a belief (i.e. update in the belief base), or it can represent an event based on which goals are added or deleted (i.e. update in its events bases).

2. The agent selects an event (if the event queue is not empty). If the event concerns a belief update, the beliefs base is updated in consequence. Otherwise, the event is unified with the head of a plan from the ones present in its plans library. If the unification succeeds, the plan is made active and thus becomes an intention.

3. The agent selects one of the active plans and executes the actions contained in the plan one by one. Each action can generate an internal event (if it is a sub-goal), an update of the beliefs base or an interaction with the environment (if the agent can perform such action) or with other agents (typically messages exchange).

Even if we don't leverage all aspects of the above BDI architecture, using this approach to implement ASC-TMS was particularly interesting to provide adaptiveness features. This approach allows ASC-TMS to sense its environment (1), to reason on its context (5,6 and 7) and take the more appropriate action to perform (8 and 9). Thanks to the BDI architecture, ASC-TMS constitute the first trust management system that is able to exhibit goal directed functioning.Thus ASC-TMS is perpetually trying to achieve objectives such as maintaining the context, providing opinions or computing reputation.

8.1.2 Programming Environments with CArtAgO

CArtAgO (Common ARTifact infrastructure for AGents Open environments) is a generic framework which makes it possible to program and execute virtual environment for multi-agent systems [Ricci et al., 2011]. The platform is based on the Agent & Artefacts (A&A) meta-model developed by Omicini and colleagues [Omicini et al., 2008]. This meta-model aims at designing multi-agent systems in which environment entities are considered as first level abstraction concepts. A&A introduced a high-level that draws inspiration from the human cooperative environment in which artefacts are used to support individual and collective activities. Analogously, *CArtAgO* allows multi-agent programmer to develop working environment as a set of artefacts (representing resources and tools), collected in workplaces that agents can join and leave and wherein they can create, use and manipulate artefacts in order to realise their individual and collective goals.

Artefacts are non-proactive entities on which agents rely to achieve their goals [Esparcia and Argente, 2010]. As illustrated in Figure 8.2, artefacts are provided with *properties*, *operations* and *functional descriptions*.

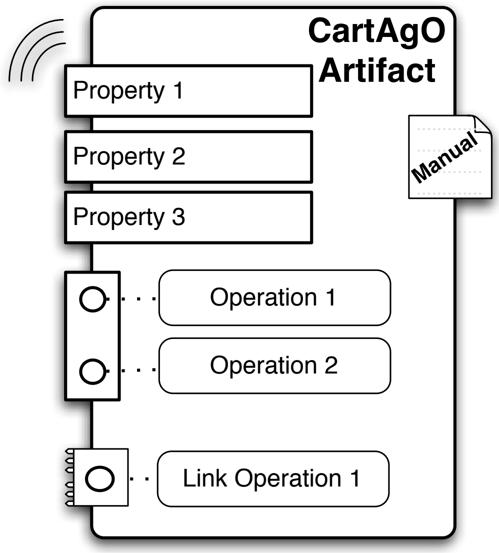


Figure 8.2 – Illustration of an Artefact (Adapted from [Ricci et al., 2011])

Operations are the means by which agents interact with the artefacts that compose their environment. There is also a special kind of operation (link operation) by means of which artefacts can manipulate other artefacts to provide complex or composite resources. In complement, *artefacts* provides a mechanism by means of which agents can be informed about the state of the artefact without performing any operation, this mechanism is implemented using *observable properties*. Finally, artefacts are enhanced with *functional description* that are used as manuals by means of which an agent can discover new artefacts and reason on their description in order to evaluate whether they are of interest to the fulfilment of its goals. Recently, a semantic description of these manuals has been proposed. With this extension, manuals are described though OWL Ontologies which ease the reasoning on their capabilities.

8.1.3 Programming Organisations with $\mathcal{M}oise$

We presented in Chapter 4 the $\mathcal{M}OISE^+$ meta-model which allows the specification of organisations. Then we showed in Chapter 5 how we used this model for the definition of multi-agent communities. In this section, we present the framework that aims at supporting the execution of normative multi-agent organisations based on the $\mathcal{M}OISE^+$[Hübner et al., 2009a, Hubner, 2011, Boissier, 2011, Hubner et al., 2002].

In the actual current version of *JaCaMo*, $\mathcal{M}oise$ organisation are interpreted and managed using three types of *organisational* artefacts, namely *OrgBoard artefact*, *GroupBoard artefact* and *SchemeBoardartefact*. In particular, these artefacts provide the following functionalities:

- *GroupBoard artefact* is used to manage an instance of the group of agent that constitute the community based on the structural specification defined in Section 5.5. For instance, this artefact is used by the agents to adopt roles.
- *SchemeBoard artefact* is used to manage an instance of the social scheme which is defined by the community functional specification. For instance, using this artefacts, an agent commits to a mission or informs its the members of a community community about the fulfilment of a mission.
- *OrgBoard artefact* is used to manage the general information about the current state of the community. These artefacts are also used as a global registry in which instances of the *GroupBoard* and the *SchemeBoard* are referenced as observable properties.

As a specific types of artefacts, organisational artefacts possess also observable properties for making the dynamic state of a community observable by the agent. For instance, it is using such observable properties that agents can be informed about the *community variables* (cf. Section 7.2.1.4) that trigger their policies adaptation. It is also using theses observable properties that agents are informed about roles, missions and norms that are available in their communities.

One of the most interesting features in $\mathcal{M}OISE^+$, regards the implementation of the ASC-TMS, is that $\mathcal{M}OISE^+$ allows agents to change the organisation and parts of its specifications. Of course, such changes are controlled via mechanisms that checks whether theses changes do not violate norms. It is this feature that we rely on to implement the process by means of which agents adapts the collective policies of their communities.

8.2 The JaCaMo-Based Community Architecture

In this section, we present how the system model $\mathcal{S}$ we framed in Chapter 5 and the ASC-TMS we proposed in chapters 6 and 7 have been implemented using the *JaCaMo* platform. As illustrated in Figure 8.3, we used *Jason* to develop and run our agents, we used *CArtAgO* to develop and execute artefact-based environments where artefacts are used to wrap the resources of the system, and we used $\mathcal{M}$*oise* to model and manage open and decentralized virtual communities as an instance of the virtual community meta-model we framed in Section 5.5.

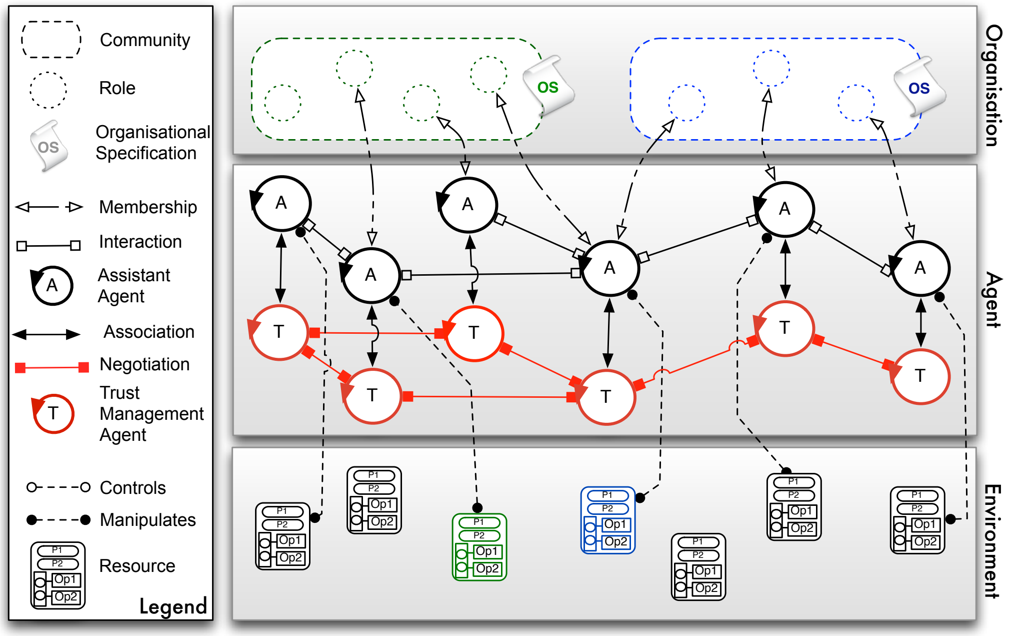


Figure 8.3 – The JaCaMo-Based Community Architecture

Thanks to *JaCaMo*, the proposed layered architecture matches perfectly the four dimensions of the virtual community model we framed in Chapter 5. At first, the *agents layer* is composed of the agents of the system (i.e. $\mathcal{A}$). To make our system application generic, we have made the choice to split the agents into two distinct types; *assistants* and *trust management agents*. An *assistant* is an agent that assists virtual communities participants in their activities. Thus this agent is endowed with application specific goals. The *trust management agent* integrate the ASC-TMS and thus is responsible of the trust management issue. The objective of this agent is to assist the assistant agent in its trust decision making.

Second, we defined the *environment layer* which integrates resources that *assistants* manipulate to during their activities. Resources have been implemented as *artefacts* and their implementation will be discussed in Section 8.3.2.

Further, agents are grouped within organisational structures that represent the communities.

Each community is defined based on its organisational specification and the set of resources shared among the community (we use a different colour for each community). Section 8.3.3 details how communities are implemented using $\mathcal{M}$*oise*. Here, we used colours to indicate the entity to which this resource belongs. For instance, green resources belong to the left community while black resources are owned by a single agent.

Finally, *interactions* do not have a dedicated layer in Figure 8.3 as this dimension is transverse to the other three layers. Agents interact with each other through communication and negotiation and they interact with resources via operations.

8.3 Implementation Details

In this section, we will delve into the details of the implementation of the agent-based virtual community and ASC-TMS. Our objective is to show how elements of the above architecture have been implemented using the *corresponding JaCaMo* technologies. To that aim, we start from agents (cf. Section 8.3.1), continue with the environments (cf. Section 8.3.2), and finish with the interactions (cf. Section 8.3.4) and the organisations (cf. Section 8.3.3).

8.3.1 Implementing Agents in Jason

In this section, we present the implementation of the *assistant* and the *trust management* (TMA) agents. We are particularly motivated in showing what functionalities are provided by the *assistant* and the TMA and how these agents collaborate to achieve the community participant goals. To proceed, we start by showing the implementation of the *assistant* in the next section we present the implementation of the TMA in Section 8.3.1.2.

8.3.1.1 Implementing the Assistant

The objective of the *assistant* is to help virtual communities members in their activities. To that aim, the agent is provided with functionalities that are split into four categories: *activities management*, *interactions management*, *trust decision management* and *authorisations management* .

- *activities management* is an application specific functionality. It encapsulates plans that relates to how assistants create, join and leave communities. It contains also plans for resource creation, manipulation and destruction.
- *interactions management* is an application independent functionality. It contains plans that the assistant use to interact with other assistants via message exchange as specified in Section 5.6. In the remainder of this chapter, we will particularly get interested in a particular form of interactions that are *requests*.
- *trust decision management* contains the plans that constitute the trust decision model mentioned in Section 6.7. Even if we did not propose any contribution with respect to

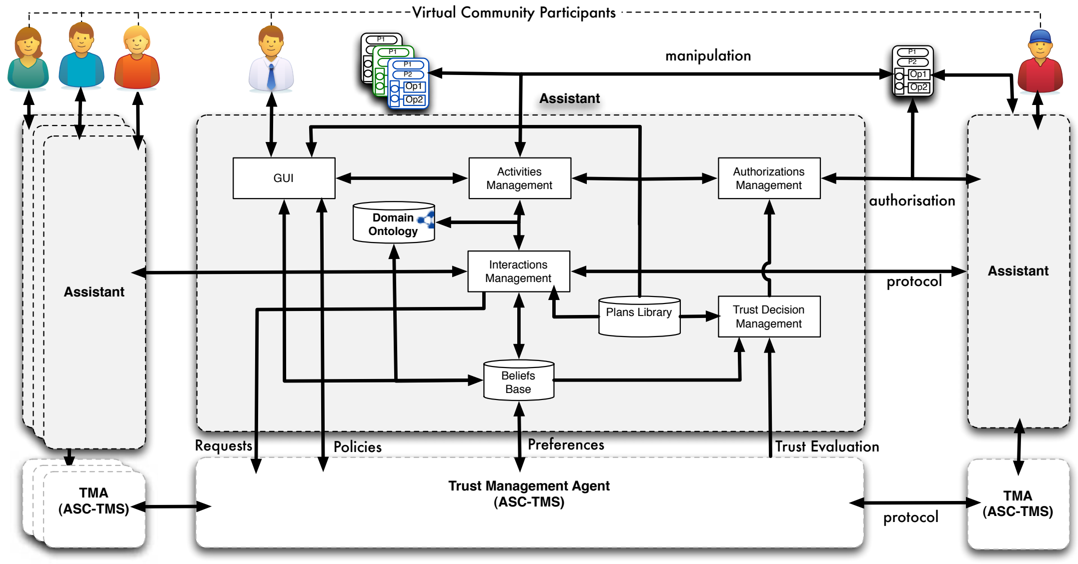


Figure 8.4 – Architecture of the Assistant Agent

the specification of a trust model, we will describe how a trust decision can be automated using Jason plans.

- *authorisation management* allows the assistant agent to issue *authorisations* based on the trust decision made by the previous functionality. This functionality is also application generic and is achieve using specific plans.

In the following, we present how *interaction*, *trust decision* and *authorisations* are made. The *activities management* will be illustrated in both sections 5.3 and 5.5. In Section 5.3 which will show how an agent creates, manipulate and destroy a resource, while in Section 8.3.3 we will present how agents create, joins and destroys communities.

Initialisation

Before that the *assistant* agent would be able to provide the aforementioned functionalities, it needs to be initialised. To that aim, each assistant is launched with an abstract goal (i.e. `!start`) that will be used to trigger the initial plan to be executed. This plan is presented hereafter.

```
+!start <-
    !build_tma;
    !work.
```

Listing 8.1 – Example of an initialisation plan

What the above plan states is that as soon as the assistant agent starts, he creates his personal TMA. The creation of the TMA is performed using the following plan.

```
 1 @Initialisation
 2 +!build_tma
 3 <- .my_name(Me);
 4    .concat(Me,"_TMS",TMS);
 5    .create_agent(TMA, "src/asl/asc−tms.asl",[agentArchClass("c4jason.CAgentArch")]);
 6    +myTMA(TMA);
 7    .send(TMA,tell,assistant(Me));
 8    .findall(policy(P)[issuer(Me), flexible(F), pattern(X,Y)], policy(P)[flexible(F), pattern(X,Y)], Policies);
 9     for(.member(policy(P)[issuer(Me), flexible(F), pattern(X,Y)], Policies)){
10    .send(TMA,tell, policy(P)[issuer(Me), flexible(F), pattern(X,Y)])}.
```

Listing 8.2 – Plan used to build the TMA

All TMAs are created using the same Jason code (i.e-. `"src/asl/asc-tms.asl"`). So what differentiates a TMA from another TMA is its name and the preferences that guides its behaviour. The instructions at lines 5 and 6 are used to keep track of the name of the TMA used by the assistant and to inform the TMA for which *assistant* he is associated to. Finally, the instruction at line 7 looks for policies that have been specified by the user to the *assistant* and transmits theses policies to the TMA in lines 8 and 9.

Once the TMA is created, the *assistant* is ready to work. Working means that the *assistant* is ready to execute orders transmitted by the user via the GUI (cf. Figure 8.4). That means also that the *assistant* is now able to reply to *requests* coming from other agents. In our thesis we consider three kind of requests, namely *access request* (i.e. request to access to a resource owner by the assistant), *release requests* (i.e. request to release a credential) and *membership requests* (i.e. requests to join the community).

Interactions Management

The *interactions management* functionality implements the management of interaction protocols (cf. Section 5.6 in Chapter 5) based on which *assistants* communicate and interact with each others. The presentation of the interaction protocol will be detail in Section 8.3.4. Therefore, we will focus in this section on how the *assistant* handles requests and interacts with its TMA to evaluate the trust he can put in the agent initiating the request (i.e. the *requester*). In the following, we present the plan that the *assistant* uses to handle request about resources (i.e. access requests).

```
@InteractionsManagement
+request(Operation, Object)[source(Ag)] :
                        owner(Object) || controller(Object) // conditions that make the plan active
                        <− ?myTMA(TMA);
                         utils.req2id(Agent,Operation,Object,ReqId);
                         reqPattern(ReqId, pattern(Operation, utils.getType(Object)));
                         .send(TMA,achieve, handle(Agent, Operation, Object, ReqId)).
```

Listing 8.3 – Request Management Plan

In this plan, the *assistant* first checks whether he is the owner or the controller of the requested resource. It not, the *assistant* simply ignores the request. Otherwise, it first generates (in line 3) the id of the request based on the *requester*, resource, and the operation names. This id will be used in subsequent interactions with the *requester* and the TMA as well. Then the *assistant* generates a belief to match the request id with the particular pattern the request belongs to. Recall that the pattern indicates to the agent which kind of decision he has to make, and thus which kind of policy his TMA should use (cf. Chapter 5). To that aim, we use the action `utils.getType` to retrieve the type (resource or community) of the object for which the request has been made. Then, in line 5, the *assistant* asks the TMA to handle the request. Here the message type is `achieve`, so when the TMA receives this message it will have `+!handle(Agent, Operation, ResName, ReqId)[source(Assistant)]` as a new goal. This new goal will trigger the plans that will derive a trust evaluation. The plans used by the TMA to perform these valuations will be presented in Section 8.3.1.2. Before, however, we describe how the *assistant* agent makes a trust decision based on the TMA trust evaluation.

Trust Decision Management

Once the TMA has finished the trust evaluation, it transmits this value to its *assistant*. This measure is then used by the *assistant* to decide whether to make a trusting decision or not. Defining a trust decision model that determines how the *assistant* use the trust measure provided by the TMA was out of the scope of our thesis. However, we assume in our implementation that the *assistant* is provided by the (VCs human) user with threshold values that defines its preferences about how the measure provided by the TMA should be used. These preferences are expressed as follows:

```
decisionThreshold(pattern(read, _), 0.75).
decisionThreshold(pattern(update, _), 0.6).
decisionThreshold(pattern(delegate, _), 0.9).
decisionThreshold(pattern(join, community), 0.8).
decisionThreshold(pattern(delegate, community), 0.9).
```

Listing 8.4 – Example of the agent's preferences

The above believes associate each pattern to a decision threshold value. For example, the first belief states the acceptance threshold for requests that tries to read the content of any all types of resources. Then based on these values, the *assistant* makes its trust decision. The plan used by this agent to accept a request that satisfies the minimum threshold is stated hereafter.

```
@TrustDecisionManagement
+trustLevel(ReqId, TL)[source(TMA)]: myTMA(TMA) & pattern(ReqId, Pattern) &
                    decisionThreshold(Pattern, T) &
                    TL >= T
```

```
                <− !trust(ReqId); −trustLevel(ReqId, TL)[source(TMA)].
```

Listing 8.5 – The trust decision plan

With this plan, the *assistant* checks whether the trust level computed by its TMA satisfies the threshold fixed by the user. If so, the plan will be active and another plan (`!accept(ReqId)`) will be triggered. The last instruction performed by this plan deletes the belief containing the trust level to avoid to use it in next evaluations.

Authorisations Management

Once the trust level of a request exceeds the threshold fixed by the user, we say that the requester is trusted for the request he has made. The trust decision is then transformed into a trusting action which is performed by the following plan.

```
1 @AuthorisationManagement
2 +!trust(ReqId) <− id2req(ReqId, Agent, Operation, Object);
3           !authorise(Agent, Operation, Object);
4           .send(Agent,tell, accept(Operation,Object)).
```

Listing 8.6 – The authorisation establishment plan

In this plan, the trusting acting is achieved by *granting* access to the *requester* (line 3) and informing it about the status of its request (line 4). To grant access to the requester object, the *controller* needs to manipulate artefacts. To that aim, we will detail this issue in Section 8.3.2.

8.3.1.2 Implementing Trust Management Agent

In the previous section, we described the *assistant* architecture that we illustrated by some source codes. We showed how this agent builds the TMA, interacts with it and makes trust decisions based on its trust evaluations. In this section, we build on that and describe the behaviour of the TMA. Concretely, the functionalities provided by the TMA fall into four categories: *requests management*, *instantiation management*, *social compliance management*, *negotiation management*, and *policy evaluation management* (cf. Figure 8.5).

The above architecture reproduces the steps described in the *adaptive and socially compliant trust management system* we framed in Section 7.3.

- *Requests Management* contains the plans that the TMA uses to select the policy that best handles the received request (based on the request pattern).
- *Instantiation Management* contain the meta-policies based on which the TMA achieves the instantiation of policies as introduced in Section 7.4.1.
- *Social Compliance Management* contains the meta-policies that the TMA use to achieve *combination*, *integration* and *evolution*. Therefore, these plans make use of both individual and collective policies as illustrated in the above architecture.

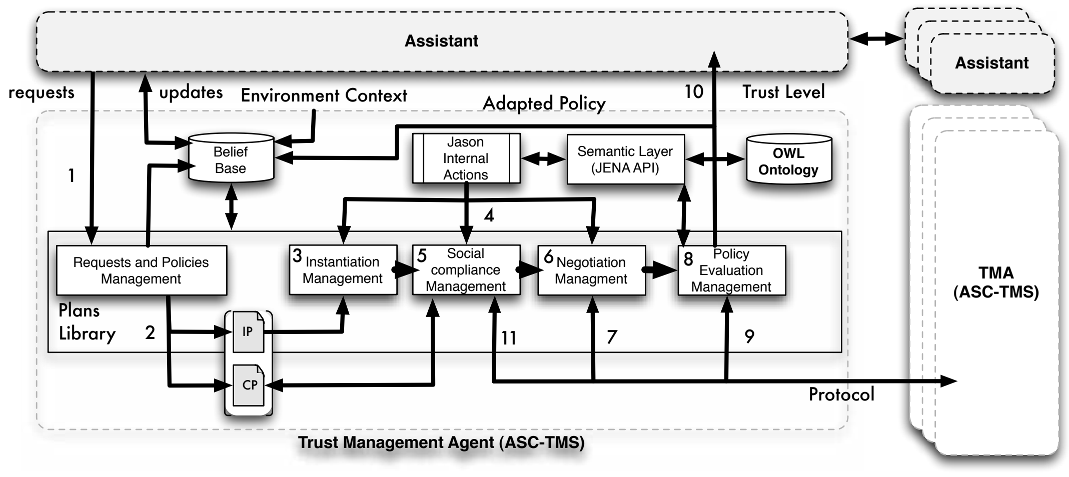


Figure 8.5 – Architecture of the Trust Management Agent

- *Negotiation Management* contains plans and meta-policies used by the TMA to achieve automated trust negotiation.
- *Policy Evaluation Management* integrate plans used by the TMA to evaluate the policies.

In the following, we describe how each functionality (except negotiation which will be presented in Section 8.3.4) has been implemented and illustrate our explanations with the corresponding Jason Code. Before we proceed, however, we briefly describe how policies and meta) policies have been implemented in Jason.

Policies Implementation

We have made the choice to implement policies as structures in Jason. A structure starts with an atom (called *functor*) and is followed by a number of terms (called arguments) separated by commas and enclosed in parentheses. The difference between predicates that are used to represent beliefs and structures is purely semantic; a structure is used as a term to represent a policy, whereas a fact is used as a logical proposition that can be evaluated to be trust of false. So policies are structures which *functor* is “policy” and their arguments are quaternary predicates representing the trust criteria conveyed by the policy. To differentiate policies, we use annotation to express the policies issuer, the pattern the policy handles and whether the policy is flexible (i.e. can be adapted) or not. This annotation mechanism is further used for many other reasons such as the specification of the combination algorithm (Cf. Section 4.1.1) or the id of the request for which the policy has been instantiated as we will see in the next section.

In the following, we present how the policy we used in Example 6.4.1 can be represented in Jason.

```
1 policy(
2 tc(passport.age,“>=”, 33, 2, m),
3 tc(passport.age,“<=", 18, 2, m),
4 tc(mail.identity,“>=", complete, 2, o),
5 tc(reputation,“>=", 70, 1, o),
6 )[issuer(bob), flexible(yes), pattern(join, community)].
```

Listing 8.7 – Example of a policy stated in Jason

Lines 1 and 6 represent the policy declaration, while lines 2, 3 and 4 contains the trust criteria stated by the policy. As said before, trust criteria are expressed as quaternary predicates in which the predicate name is `tc`. The first literal of the predicate resents the n this predicate, the first literal represents the trust factor ($f \in \Delta_f.T$) on which the condition is formulated, the second literal represent the comparison operator used to set the acceptable values, the third literal represent the minimum value based on which the condition is satisfied (($v \in \Delta_f.A/v \doteq f$)), the fourth literal represent the weight the condition during the policy evaluation, and the last literal specifies whether the condition is mandatory or not. Here, the policy makes use of three Lower Bound Conditions and one Upper Bound condition (Cf. Section 6.3).

Requests Management

With this functionality, the TMA handles the received request by selecting the most appropriate policy. The TMA selects the policy which pattern corresponds to the pattern of the request as performed in the plan below.

```
+!handle(Agent, Operation, Object, ReqId)[source(Assistant)]:
        myAssistant(Assistant)
        <- reqPattern(ReqId, pattern(Operation, utils.getType(Object)));
        ?pattern(ReqId, Pattern);
        ?policy(P)[request(ReqId), issuer(Me), pattern(Pattern)];
        ?policy(P)[request(ReqId), issuer(Me), pattern(Operation, Object)];
        !instantiate(P, Agent, Operation, Object);
        !integrate(P, Agent, Operation, Object).
```

Listing 8.8 – Request Management Plan

At first, the plan checks whether the agent sending the request is the *assistant* that created the TMA. Then, the request pattern is identified and a new belief is added correspondingly. The plan makes use of a test goal (`?pattern(ReqId, Pattern)`) to retrieves at first the pattern and use this pattern to select the policy which is associated to the pattern. We used two times the test goal to retrieve policy to select the most specialised policy. In other words, if a policy is defined for a particular resource type and another policy is specified for a particular resource which belongs to to this type, it is this second policy that will be selected as it is considered as the most appropriate policy. Finally, once the policy has been selected, the plan triggers the instantiation of the policy and its integration with the collective policy. These last goals are executed in sequence; the integration will not take place until all instantiation plans have finished.

Instantiation Management

This functionality assume the instantiation of the selected policy to fit the context in which the trust decision has to be made. To that aim, we make use of several meta-policies that are actually expressed as plans. Each meta-policy constitute a *known-how* for the TMA with respect to *how* selected policies could be adapted in response to the *business-context*. In our implementation, we adapt the policies to three types of environment changes, namely *collusion*, *credentials forgery* and *resource updates*. The following policy illustrates how we the meta-policy that has been defined to react to resource update is specified.

```
+!instantiate(Policy[pattern(Pattern)], Agent, Operation, Object) :
        value(Old)[pattern(Pattern)] & valueOf(New)[artifact_name(Object)] & New > Old
        <- Delta = New − Old ;
        for(Delta>0){
        policy.restrict(Policy, all);
        Delta = Delta −1;
        }
        for(Delta<0){
        policy.relax(Policy, all);
        Delta = Delta + 1;
        }
        −+value(New)[pattern(Pattern)].
```

Listing 8.9 – Policy Instantiation Plan

In the above meta-policy, the belief `value(Old)[pattern(Pattern)]` contains the value of the resource for which the policy has been used last time. This value is compared to the value of the current resource for which the policy is instantiated. Then the delta between these values is computed and the policy is either restricted or relaxed consequently. Finally, the value belief is updated with the value of the current resource. The idea of this meta-policy is to make the policy evolve with the resource it is protecting. So the more the resource becomes valuables, the more the policy will be restrictive. Therefore, the keyword `all` is used instead of a particular *trust factor* meaning that all trust criteria that can be changed will be adapted.

In a similar way, we defined three other meta-policies to demonstrate the benefit of the instantiation. Of course, the user can define its own meta-policies via the assistant agent. To that aim, the assistant makes use of the *tellHow* message as follows.

```
.send(TMA, tellHow, "!instantiate(Policy, Agent, Operation, Object): friend(Agent)
        <- policy.delCriterion(P,identity).")
```

Listing 8.10 – How the assistant personalises the meta-polices of the TMA

Once then the assistant agent transmits this policy using a message. When the above instruction is executed, the string in the message content will be parsed into Jason meta-policy and added to the TMA meta-policies library.

Adaptation actions (e.g. `policy.relax(...)`) are not natively provided by Jason. However, as Jason itself is developed in Java, the framework allows developer to specify user-defined internal actions. Unlike standard internal action (e.g. `.send`, `.my_name` or `.count`) a user-defined internal actions are accessed by the name of the library, followed by ".", followed by the name of the action. Libraries are defined as Java packages and each action in the user library is a Java class within that package; the names of the package and class are the names of the library (Adaptation Actions in Figure 8.5) and action as they will be used in Jason Plans.

Social Compliance Management

The *social compliance* functionality implements the adaptation to *social-context* as defined in Section 7.5. This functionality is achieve by the TMA with three different types of meta-policies; *combination*, *integration* and *evolution*. *Combination* allows the agent to build a collective policy based on the policies provided by the members of its community, *integration* make the agent complying or not with the collective policy of its community.

Combination meta-policies are triggered by the plans used in the *activities management* of the *assistant agent*. It is used by the TMA to *combine* the policies transmitted by the *assistant* to build a collective policy. This is performed using the following meta-policy.

```
@socialComplianceManagement
+!doCombination(Policies, Community, Pattern, heuristic)[source(Assistant)] :
                              myAssistant(Assistant)
                              <− policy.combine(Policies, Pattern, Heuristic, CollectivePolicy);
                              .send(Assistant, tell, newCollectivePolicy(CollectivePolicy)[issuer(Community),
                                     pattern(Pattern)]);
                              −+policy(CollectivePolicy)[issuer(Community), pattern(Pattern)].
```

Listing 8.11 – How the assistant personalises the meta-polices of the TMA

The plan is triggered with a set of policies, a name of a community, a pattern, and a combination heuristic. The TMA combines the set of policies to create a collective policy. The combination is performed by the TMA following the algorithm presented in Section 7.5.1 which has been implemented in Java and made available to the TMA via an internal action (i.e. `polic.combine(...)`). Once the TMA executes this internal action, the outcome of this execution is used to ground the variable Collective policy with the policy resulting from the integration. Then this policy is sent to the *assistant* which will make this newly generated collective policy active. Finally, the TMA keeps track of the collective policy in order to use it during *integration*.

Integration is performed in a similar way. However, *integration* is automatically triggered by the TMA each time a request is received (cf. Requests Management). Recall, we defined in Section 7.5.2 four types of heuristics, each generating an integrated policy that possess some desired properties (i.e. accepts when both policies accept, reject when both policies reject, accept all requests accepted by one policy and reject all request rejected by a policy). Thus the choice of the heuristic determines the behaviour exhibited by the *assistant* (compliant or deviant). However, we demonstrated experimentally in [Yaich et al., 2013] that *heuristic 1* and *heuristic 2* were inappropriate for compliance issues as they do not guarantee any property when the policies to be integrated conflict (one accepts and the other one rejects the request). Therefore, the integration is achieved using heuristic 3 and 4 based on beliefs that determines the user preferences about how compliant he wants to be. These beliefs are stated as follows.

```
complianceProfile(compliant).
complianceProfile(deviant).
complianceProfile(conditionally).
```

Listing 8.12 – Compliance preferences

So by stating one of the above beliefs, the *assistant* transmits the preferences of the user to the TMA. If the first preference is used, the agent will always and blindly comply with the collective policy of its community. In contrast, if the second preference is used, the agent will use only its individual policy and thus not complying with the collective policy. Finally, if the agent makes use of the third preference, it will conditionally comply (i.e. when some conditions are met). In the following, we present the three meta-policies that correspond to the above preferences.

```
 @socialComplianceManagement
!integrate(IP[pattern(Pattern)], Agent, Operation, Object): complianceProfile(deviant) <- true.
```

Listing 8.13 – Compliance Plan

The above meta-policy implements the simplest case in which the agent do not want to comply with the collective policy. Therefore, the individual policy is kept as it is without being integrated with the collective policy.

```
 @socialComplianceManagement
!integrate(IP[pattern(Pattern)], Agent, Operation, Object) : complianceProfile(compliant) &
                                   myCommunity(Com) & member(Agent, Com)
                                  <- ?collectivePolicy(CP)[pattern(Pattern), issuer(Com)];
                                  policy.integrate(CP, IP Pattern, h3).
```

Listing 8.14 – Compliance meta-policy

With the above policy, the agents that want to exhibit a compliant behaviour will always use integration. The difference between this policy and the one we present hereafter lies in the fact that the above policy is used for requests that originates from agents that are not members of the community. Therefore, the meta-policy makes use of the heuristic `h3` which guarantees that the integrated request will reject any request rejected by the collective policy. However, the heuristic allows the agent to be more restrictive than the community; the integrated policy may reject requests that the collective policy would have accepted.

```
 @socialComplianceManagement
!integrate(IP[pattern(Pattern)], Agent, Operation, Object) : complianceProfile(compliant) &
                              myCommunity(Com) & not member(Agent, Com)
                             <- ?collectivePolicy(CP)[pattern(Pattern), issuer(Com)];
                             policy.integrate(CP, IP, Pattern, h4).
```

Listing 8.15 – Meta-policy used for integration

Besides, the above meta-policy is to be used when interacting with agents that are members of the same community. Indeed, members of the same community may legitimately expect that the policy used to handle their request are not more restrictive than the collective policy. Nevertheless, a member is not obliged to have such a behaviour, one can use the same integration meta-policy in both cases.

Finally, the agents that are conditionally complying with the collective policy make use of the following meta-policy.

```
 1  @socialComplianceManagement
 2 !integrate(IP[pattern(Pattern)], Agent, Operation, Object) : complianceProfile(conditionally)
 3                              & myCommunity(Com)
 4                              & collectivePolicy(CP)[pattern(Pattern), issuer(Com)]
 5                             <- getPopulation(Com, Population);
 6                             if((Population<3) ||(Population>=20)){
 7                             policy.integrate(IP, CP, Pattern, h3)
 8                             }
 9                             else((Population > 5) & (Population<20)){
10                             policy.integrate(CP, IP, Pattern, h3)
11                             }.
```

Listing 8.16 – Compliance meta-policy

In this policy, if the agent is in a context that is suitable for compliance, with respect to the conditions stated by Latané in its *social impact theory*, the agent will use heuristic h3 to avoid entering in conflict with its community. Recall, h3, will never accept a request that the policy having the priority would have rejected. Here the policy having the priority (line 7) is the collective policy as it is the first argument of Algorithm 2 (cf. Section 7.5.2). Otherwise, if the condition of compliance are not met, the meta-policy give the priority to the individual policy during the integration. Giving the priority to the individual policy means that the agent accepts to be in conflict with its community and thus exposes itself to sanctions.

Policy Evaluation Management

Policy evaluation management consist in *collecting* the trust information required for the evaluation of the policy, and *computing* the extent to which these information satisfy the *trust criteria* express in the policy. To that aim, this functionality is achieve in several steps as specified in the following plan.

```
@PolicyEvaluation
+!policyEvaluation(Policy, ReqId) <-
                +profile([])[request(ReqId)];
                !getTrustInformation(ReqId);
                !collectInformation(ReqId);
                .wait(100);
                !evaluatePolicy(ReqId).
```

Listing 8.17 – Evaluation Plan

First, the TMA creates a belief (i.e. `profile`) in which he will store all information (i.e. credentials and declarations) required for the policy evaluation. This profile is annotated with the request id to avoid confusion when several requests are handled in parallel. The second step consist in creating a new goal (i.e. getTrustIInformation) which parse the policy to identity the trust criteria that require credentials *credentials* (i.e. proofs) and those that require *declarations*. *Credentials* are requested from the TMA of the *requester*, while *credentials* are requested from the agents composing the system. The following plan illustrates how this issue is achieved.

```
@PolicyEvaluation
+!getTrustInformation(ReqId): policy(Policy)[request(Req)] & ReqId== Req
            <- utils.getRequester(ReqId, Requester);
            for(.member(tc(Type, Value, Weight), Policy))
              {
              if(semanticLayer.isProof(Type))
                {
                .send(Requester, achieve, getCredentials(ReqId, List)
                }
              else
                {
                .broadcast(achieve, getCredentials(ReqId, List)
                }
              }.
```

Listing 8.18 – Evaluation Plan

Once this plan executed, the TMA executes the plan by means of which the received information is processed (i.e. `collectInformation`). Declarations are aggregated to derive a unique value, while credentials are verified with respect to the *public* key used by the certification authority issuing the credential. Then for each type of information, a belief having `ti(Type, Value)` is added to the corresponding profile. Once the profile is built, the *evaluatePolicy* plan is triggered. This plan computes the extent to which the information contained in the profile satisfy the corresponding policy. The code of these plans is presented hereafter.

```
+!evaluatePolicy(ReqId): ?profile(Profile)[request(ReqId)] & policy(Policy)[request(ReqId)]
          <-
          +weights(0)[request(ReqId)];
          +sum(0)[request(ReqId)];
          for(.member(tc(T, O, V, W, M), Policy))
```

```
{
?weights(Ws)[request(ReqId)];
neWeight = Ws + W;
−+weights(neWeight)[request(ReqId)];
for(.member(tic(Type,Value), Profile))
{
if((Type == T)) & (semanticLayer.satisfy(Type, Value, V, O)))
  {
?sum(S)[request(ReqId)];
newSum= S + W;
−+sum(newSum)[request(ReqId)];
}
else{
if(M == m)
+unsatisfiedMandatory[request(ReqId)];
}
}
?sum(Sum)[request(ReqId)];
?weights(Weight);
+trustLevel(Sum/Weight)[request(ReqId)].
```

Listing 8.19 – Evaluation Plan

The above plan implements the *policy evaluation function* presented in Section 6.4. The plan makes use of the ontology to check whether the values contained in the collected information satisfy the threshold values set in the policy. In the current implementation of the ASC-TMS, the ontology has been implemented in OWL DL using the Protege framework [The Standfod Center for Biomedical Informatics Research (BMIR), 2000]. The mapping between Jason plans and the ontology is achieved by the semantic layer (cf. Figure 8.5) which has been implemented as a library of internal actions (e.g. `isProof`, and `satisfy`)[2].

The outcome of this evaluation represent computed trust level which is added to the belief base of the TMA. The addition of this belief generates a new event which will trigger the plan that informs the *assistant* about the outcome of the evaluation. This last step is achieve by this plan.

```
1 + trustLevel(TL)[request(ReqId)]: not unsatisfiedMandatory[request(ReqId)] & ?assistant(Assistant)
2                  <− .send(Assistant, tell, trustLevel(ReqId, TL)).
3
4 + trustLevel(TL)[request(ReqId)]: unsatisfiedMandatory[request(ReqId)] & ?assistant(Assistant)
5                  <− .send(Assistant, tell, trustLevel(ReqId, 0)).
```

Listing 8.20 – Plan used by the TMA to send the computed trust evaluation to the assistant

With the above plan, the TMA informs its *assistant* about the result of the trust evaluation. If one of the mandatory *trust criteria* was not satisfy, the final outcome of the evaluation is considered to be null (line 5). Based on this evaluation, the *assistant* makes a trust decision as presented in Section 8.3.1.1.

[2]This library makes use of the Jena library to manipulate the OWL ontology

8.3.2 Implementing Resources in Artefacts

Artefacts are used to implemented the resources of our model (i.e $\forall r_i \in \mathcal{R}$). As presented in Section 8.1.2, artefacts are characterised by a set of observable and non observable properties, a set of operations, and a functional description.

The artefact private attributes represent *non observable* properties, the public attributes represent the *observable properties* and the methods represent the operations provided by the resource type.

We present hereafter how the concepts we used to describe the resources in our model are mapped to the concepts used in *CArtAgO*.

- Observable properties have been used to implement the resource *owner* ($r_i.\varphi$), *sensitivity* ($r_i.\varsigma$) and its *value* ($r_i.\nu$). In their most general form, observable properties are represented by a tuple, with a functor and one or multiple arguments, of any type. In our implementation, *sensitivity* and *value* are have integer value, while *owner* is an *ArrayList* containing the identifiers of the resource owner.

- Non observable properties are classical Java object attributes which have been used to implement the *content* ($r_i.\theta$) of the resource. We will see later in this section why we have made the choice to implement part of the resource properties as observable properties, while we kept the *content* non observable.

- Both observable and non observable properties are accessed via operations. So here the mapping between the resources operations used in our model and the operations provided by *CArtAgO* is straightforward.

- The authorisation function ($r_i.Auth$) introduced in Section 6.8 is also implemented as a particular operation called `grant`.

In the following, we will describe how the *assistant*, implemented in Jason, interacts with the resources that are implemented in *CArtAgO*. We are particularly interested in showing how these agents *create*, *manipulates* and *grants access to* the resources that compose its environment.

Resource creation

In *CArtAgO*, artefacts are structured in *workspaces*. Each workspace constitute a sub-environment containing dynamic set of artefacts. In our implementation, we have made the choice to implement the environment dimension ($\mathcal{R}$) as a unique workspace that is accessible to all agents of the system ($\forall a_i \in \mathcal{A}$). So each resource that is available in the system belongs to the default workspace.

In order to create a resource, the *assistant* agent exploit the `makeArtifact` action. The following plan presents the way resources are created.

```
@ResourceCreation
+!createResource(ResNam, ResType) : true
                <- .my_name(Me);
                makeArtifact(ResName,ResType,[Me, 0,0],ResId).
```

Listing 8.21 – Plan used by the TMA to send the computed trust evaluation to the assistant

With this plan, the *assistant* creates a resources of type `ResType` which name is `ResName`. The list `[Name, 0,0]` corresponds to the arguments transmitted after its creation. These arguments corresponds to the values of the observable properties, namely *owner*, *sensitivity* and *value*. The *operations* admitted by the resources depend on the type of this resources. In fact, the type of the resource corresponds to the Java class which inherits from the default *Artifact* class provided by *CArtAgO*. The name of the class corresponds to the artefact type.

Resource discovery

Without the resource identifier, an *assistant* will not be able to manipulate it. An agent can discover the resources that are available in its environment (i.e. workspace) by means of the following plan.

```
@ResourceDiscovery
+?availableResource(Res, Operation) : true
                <- lookupArtifact(Res, ResId);
                !accessResource(Operation, ResId).

-?availableResource(Res, Operation) : true
                <- .wait(100);
                ?availableResource(Res).
```

Listing 8.22 – Plan used by the TMA to send the computed trust evaluation to the assistant

Resources are discovered based on their name or their type. If a resource with the specified name or type exists, its identifier is bound to the variable `ResId`. In case several the resource types is specified and several resources of the same type are available, one resource is randomly chosen. The second plan is used to manage the situation in which the first plan fails finding a corresponding resource. In this case, the agent will wait for a certain period (e.g. 100 milliseconds), then try to discover the resource again. If the resource is found, the goal `accessResource(ResId)` triggers the access the resource.

Resource Manipulation

Once the resource discovered, the *assistant* is able to manipulate it. The manipulation of the resource is made based on the type of the resource. Each resource type offers a set of operations by means of which the content of the resource can be updated.

We present hereafter a plan used to read and update the content of a `TextFile` resource.

```
 1 @ResourceManipulation
 2 +!accessResource(ResId) : profile(altruist)
 3            <- read(Content)[artifact_id(ResId)];
 4            utils.improve(Content, Content2);
 5            update(Content2)[artifact_id(ResId)].
 6
 7 +!accessResource(ResId)[error_msg(Msg), operation(Op)] :
 8              Msg =="not_authorized" <-
 9              getControllers(Controllers)[artifact_id(ResId)] ;
10              ?myTMA(TMA);
11              for(.member(Agent, Controllers))
12              {
13              .send(Agent, tell, request(Operation, ResName));
14              .send(TMA, tell, pendingRequest(Agent, ResId, Operation));
15              }
```

Listing 8.23 – Plan used by the TMA to send the computed trust evaluation to the assistant

With the first plan, the *assistant* agent, which has a *altruist* profile, wants to read the content of a resource (line3), improve its quality (line4) and update the resource consequently (line 5). However, the resource reveals to be *private*, and thus its manipulation is limited to trusted agents. The failure of the first plan automatically triggers the execution of the second plan. In this second plan, the agent will pars the error message to evaluate the reason of the failure of the first plan [3]. Here, the message corresponds to an *unauthorised* access. In such situation, the agent will retrieve from the resource the list of agents controlling it (line 13) and sends a request to each agent of the list. Finally, the agent informs its *TMA* (line 14) about pending request so that this latter accepts the related negotiation demands.

Resource authorisation

We presented in Section 8.3.1.1, how the *assistant* agent makes trust decision based on the *trust evaluation* performed by its TMA. In this section we will see how the decision is enforced by the *assistant*. Once the `+!authorise(Agent, Operation, Object)` added to the *assistant* belief base, the agent triggers the following plan

```
@ResourceAuthorisation
+!authorise(Agent, Operation, Object) <- grant(Agent, Operation)[artifact_id(Object)].
```

Listing 8.24 – Plan used by the TMA to send the computed trust evaluation to the assistant

Granting access to the resource is performed by invoking the operation `grant` which is common to all resources. By default, *CArtAgO* has been designed to support *role-based access control*. However, these mechanisms has not been implemented in the current version of the distribution. Additionally, even if it was implemented, we would have to extend it in order be

[3]The message is sent automatically by the artefact whenever their is an unauthorised manipulation of the resource. The artefact sends also the name of operation for which the signal was generated

able to issue *authorisation* that entails on individual instead of a whole group. To that aim, we implemented our *resources* with a hybrid (IBAC and RBAC) *access control mechanism*.

Consequently, the first argument of the grant *operation* can be either an agent identifier, or the identifier of a community. In this case, every agent composing the community is granted access to the resource. This feature is only used when the resource is a collective resource.

8.3.3 Implementing Virtual Communities with $\mathcal{M}oise$

In our implementation of the *virtual community* functional specification defined using $\mathcal{M}oise$, we grouped the *OrgBoard*, *GroupBoard* and the *SchemeBoard* into one unique artefact called `ComArt` (i.e. *community artefact*). In addition to the functionalities provided by these artefacts, the *community artefact* encapsulates the *collective policies* that are in use within the community and the operations based on which the members of the community are able to *retrieve* and *update* these policies. Finally, we provide in the `ComArt` a `grant` operation by means of which the community owner can grant privileges (e.g. membership, delegation, etc.) to other agents. Worth noting that we consider that the *owner* of a community does not have any power on other members. Also, we assume that when a community is initiated by several agents, only one agent creates the community but the role of *owner* is adopted by all.

In the following, we present the plans used by *assistant* agents to create, configure and join communities. These plans are generic and are embodied by all *assistants*.

```
1 @CommunityCreation
2 +!createCom(ComName) <−
3             makeArtifact(ComName, ‘‘ComArt’’, ["community−os.xml", ComId);
4             adoptRole(owner)[artifact_id(ComId)];
5             .my_name(Me);
6             +owner(ComId, Me);
7             +myCom(ComId);
8             !admitMembers(ComName, ComId).
```

Listing 8.25 – Community Creation Plan

With the above plan, the *assistant* agent creates a community by instantiating the `ComArt` artefact. Like any artefact, the `ComArt` is created using the *makeArtifact* action. Once the community created, the initiator of the community adopts the role of *owner* (line 4). As stated in the definition of the functional specification (cf. Definition 30), the owner of a community can admit new members. To that aim, the above plan triggers a new goals by means of which the *owner* will admits new members (line 5). The plan used to handle this new plans is detail in what follows.

```
1 @CommunityManagement
2 +!admitMembers(ComId): trust(Candidate, join, community) &
3                 owner(ComId, Me) & .my_name(Me)
4                 <− grant(membership, Ag)[artifact_id(ComId)];
5                 − trust(Candidate, join, community);
```

```
                .send(Candidate, achieve, adoptRole(member, ComId));
                !admitMembers(ComId).
```

Listing 8.26 – Plan for admitting new members

Here, the *assistant* agent playing the role of owner will systematically admit members that have been trusted to be members of the community he created. Of course, trusting a members means that follows the same process described in Section 8.3.1.2 in which the *assistant* makes use of the evaluation of its TMA to decide whether the agents should be trusted or not. In the above plan, the *assistant* invokes the *grant* to add the new member to the list of agents that are admitted to perform the operation *adoptRole*. This agent is then requested (using an achieve message) to adopt the role of member. The new member adopts this role in the same way that the community creator adopted the role of *owner* in Listing 8.32. This is illustrated in the following plan.

```
@AdoptingRole
+!adoptRole(member, ComId) <− adoptRole(member)[artifact_id(ComId)].
```

Listing 8.27 – The plan for adopting the member role

8.3.4 Implementing Interactions

The way agents interact with their environments is performed by *CArtAgO* through *Artefacts*. This is achieved via their operations that allow agents to interact with the resources that compose their environment. In this section we present how these agents communication with each others.

As illustrated in Figure 8.1, at the beginning of each reasoning cycle, agents check for messages they might have received from other agents using the checkMail (step 3). Any message received by an agent has the following structure [Bordini et al., 2007]:

$$\langle receiver, performative, content\rangle \tag{8.2}$$

The sender is an atom representing the id by means of which the agent receiving the message is uniquely identified in the multi-agent system. The performative denotes the intention of the sender (cf. Section 5.6), while content is a predicate which interpretation depends on the performative used in the message.

Message are sent with a pre-defined *Jason* internal action that is used to inter-agent communication. So to send messages like the one presented above, the agent has to use this internal action within a plan. The following plan illustrates how a message is sent by an agent.

```
+!event:
<− .send(receiver, performative, content)
```

Listing 8.28 – A generic plan for sending a message

In the above plan, the receiver represent the id of the agent that will received the message. In Jason, provides nine performatives, namely *tell*, *untell*, *achieve*, *unachieve*, *askOne*, *askAll*, *tellHow*, *untellHow*, *askHow*. In the following, we briefly describe the semantic of each performative.

- `tell` (resp. `untell`): the sender intends the received (resp. not) to believe that the predicate of in the message content is true.
- `achieve` (resp. `unachieve`): the sender requests the receiver to achieve (resp. drop) a goal.
- `askOne`: the sender wants to know if the content is true for the receiver.
- `askAll`: the sender wants to know all the answer to a question.
- `tellHow`: the sender informs the receiver about a plan.
- `untellHow`: the sender request that the receiver deletes a plan from its library.
- `askHow`: the sender requests the receiver to provide him with all plans that are relevant for the triggering event provided in the content.

The above performatives provided by Jason are different from the ones we specified in Section 5.6 and Section 7.4.2.1. So to implement the *interaction protocol* and the *negotiation protocol* we developed in this thesis we had two alternatives; (a) changing the Jason interpreter to allow or the interpretation of specific performatives, or (b) implement the protocol using Jason plans to interpreter received messages. We have made the choice to implements *interaction* and *negotiation* protocols as plans library that are included at the plan library of any agent when it starts running. This approach has been also adopted by the authors of Jason to implement an example about Contract Net Protocol [Bordini et al., 2007].

Consequently, all performatives used in our model (Section 5.6 and Section 7.4.2.1.) are implemented using `tell` messages. In the following, we present two plans to show how the sender and the receiver interact using the same protocol.

```
1 @negotiationProtocol
2 +!Negotiate(Receiver, ReqId): true
3             <- .send(Receiver, tell, start-negotiation[request(ReqId)]).
```

Listing 8.29 – Two plans implementing the initiation of a negotiation

All negotiation process is conducted by the TMAs, so the sender and the receiver of the above plans represent the TMA of the requester and the TMA of the controller. The first plan (line 2) is triggered by the TMA of the controller after performing the integration of the individual and the collective policy (cf. Section 8.3.1.2). The plan execution will result in the addition of the belief `start-negotiation[source(Sender), request(ReqId)]` to the belief base of the TMA of the *requester*.

Having this belief in its beliefs base, the TMA of the requester will have to find a plan that will allow him to act so as to handle the new event (i.e. belief addition). So the first thing this agent will do is to find in the *plan library* (cf. Figure 8.1) all plans which are relevant for the given plan. This step corresponds to the execution of the *L.Replay* function we defined in the interaction protocol (cf. Section 5.6). This is achieve by retrieving all plans that have a triggering event that can be unified with the `+start-negotiation` event.

Once the TMA retrieves all relevant plans, the agent has to determine which ones are applicable. Recall, plans have a *context* part which states when a plan can be used. It is these information that are used to active or not a given plan.

The *interaction protocol* has been designed so that only one plan is possible at any moment of the interaction. However, the *trust negotiation protocol* is strategic. So the agent has to evaluates several alternatives and select the one that maximises its *expected utility*. Consequently, the TMA has to handle the situations in which several plans are applicable alternatives to handle the selected event (e.g. here the TMA of the requester can answer with `accept-negotiation` and `refuse-negotiation`). To that aim, we implemented the *expected utility function* specified in Section 7.4.2.2 as an internal action that the TMA agent uses during the negotiation.

```
@negotiationProtocol
+start−negotiation[source(Sender), request(ReqId)] :
                          getUtility(ReqId, accept−negotiation, U1) &
                          getUtility(ReqId, refuse−negotiation, U2) &
                          U1 >= U2
                          <− .send(Sender, tell, accept−negotiation[request(ReqId)]);
                                +currentNegotiation(requester, ReqId).
@negotiationProtocol
+start−negotiation[source(Sender), request(ReqId)] :
                          getUtility(accept−negotiation, U1)
                          & getUtility(refuse−negotiation, U2)
                          & U1 < U2 <− .send(Sender, tell, refuse−negotiation[request(ReqId)]).
```

Listing 8.30 – Two plans implementing the initiation of a negotiation

The above plans illustrates how the TMA of the requester selects the next move in the negotiation game. Here, the agent makes use of the `getUtility()` internal action which computes the *expected* if the agent accepts or refuses the negotiation. Then this value is compared with each of the alternatives at the negotiation stage. The plan having the utility that outperforms the other alternative are triggered.

Of course, the main contribution we we want to stress here not the *negotiation protocol* itself. Rather, we prefer focusing on the functionality by means of which an agent is able to adapt its policy when this adaptation improves its utility. This issue is illustrated by the following plan.

```
@negotiationProtocol
+propose(Criterion, Value)[source(Sender), request(ReqId)] :
                           getUtility(ReqId, accept−proposal, U1) &
                           getUtility(ReqId, reject−proposal, U2) &
                           getUtility(ReqId, end−negotiation, U3) &
```

```
6                        (U1 > U2) & (U1 > U3)
7                         <- ?policy(P)[request(ReqId)];
8                         policy.updateCriterion(P, Criterion, ,Value, , );
9                         .send(Sender, tell, accept-proposal[request(ReqId)]).
```

Listing 8.31 – Two plans implementing the initiation of a negotiation

With the above plan, the TMA of the controller evaluate the utility of the three alternatives to a proposal. This plan is triggered when the utility of accepting the proposal outperforms the utility of rejecting the proposal or ending the negotiation (lines 3 to 6). Once the proposal accepted, the agent update its policy consequently. To that aim, he makes use of the `policy.updateCriterion()` internal function (line8). Then he informs the TMA of the controller about its decision. In the next section, we illustrate this negotiation process using a concrete example.

8.4 Conclusion

In this chapter, we discussed the implementation of ASC-TMS and the agent-based virtual community platform on which it has been deployed. The overall system is used to build *virtual communities* which user's are assisted in their trust decisions by ASC-TMS. The main advantage of ASC-TMS embodied in the trust management agent lies in its ability to unload VCs users from trust management tasks. Thus these members can completely focus on their activities while having the guarantees that their trust evaluations will remain accurate.

To that aim, we first we presented the *JaCaMo* multi-agent programming platform we used to implement our system. We showed the adequacy of this platform with the architecture of the system we presented in Chapter 5 and present the features that interested us in *JaCaMo*.

Then, we delve into implementation details to present how *agents* has been implemented in *Jason*. We showed how the architecture of the system respects the principal of separation of concerns as the functionalies that are application specific and those that relate to the trust management issue are encapsulated in two different agents (i.e. assistant agent and trust management agent). Then we presented how *resources* has been implemented into *CArtAgO* artefacts and how the M$\mathcal{OISE}$ specification of communities was executed by *Jason* agents. Finally, we described how the *interaction* and negotiation protocol introduced in this thesis was implemented.

Our implementation of ASC-TMS has been made deliberately generic to allow the widest possible application of our system.

8.5 French Summary

Dans ce chapitre, nous fournissons des détails techniques sur l'implémentation de la communauté virtuelle d'agents et le système de gestion de la confiance (ASC-TMS) que nous avons déployée dessus. Pour cela, nous commençons par décrire la plateforme de développement multi-agent *JaCaMo* utilisée pour l'implémentation de notre système.

JaCaMo est né de l'intégration de *Jason*, *CArtAgO* et $\mathcal{M}$*oise*, trois plateformes de développement multi-agent. Ainsi, *JaCaMo* c'est : (i)programmer des agents et les interactions agent-agent car c'est ce que permet de faire *Jason*, (ii) programmer des environnements et des interactions agent-environnement en utilisant la plateforme *CArtAgO*, (iii) programmer des des organisations d'agents ainsi que des interactions agents-organisation à travers la plateforme $\mathcal{M}$*oise*.

Dans ce qui suit, nous détaillerons l'implémentation de chaque partie de l'architecture décrite dans le chapitre précédent (Cf. Figure 7.8).

8.6 L'architecture *JaCaMo* d'une communauté virtuelle

Dans cette section, nous présentons comment le modèle de communauté virtuelle représentant notre système $\mathcal{S}$ dans le chapitre 6 a été implémenté en utilisant la plateforme *JaCaMo*.

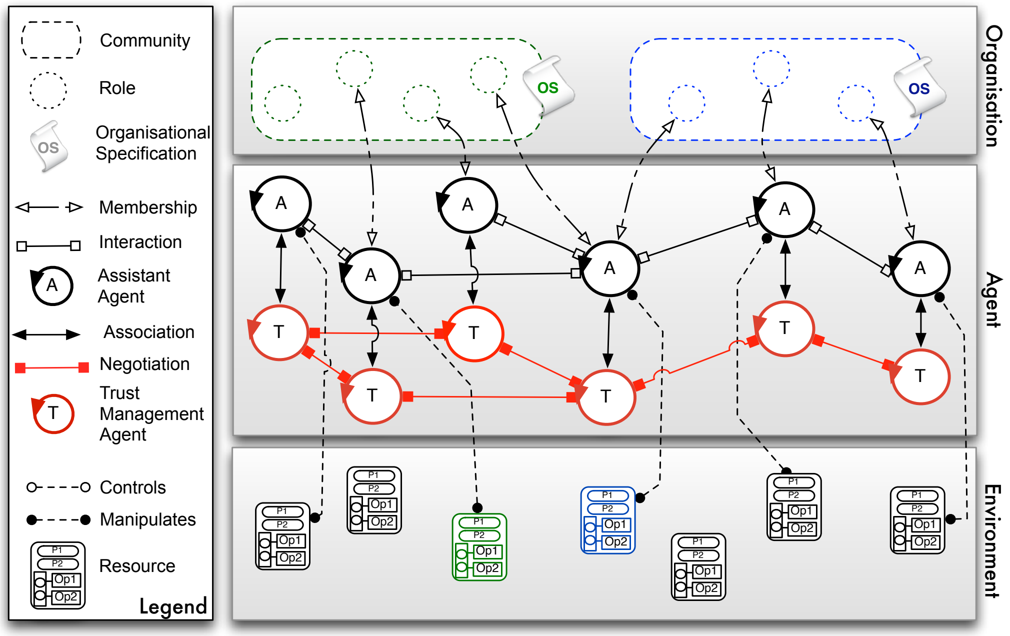


Figure 8.6 – L'architecture JaCaMo du système $\mathcal{S}$

Comme illustré dans la figure 8.6, Nous avons utilisé *Jason* pour développer et exécuter nos agents, nous avons également utilisé *CArtAgO* pour développer et faire vivre des environnements à base d'artefacts. Ici les artefacts sont utilisés pour encapsuler les ressources de notre système. Enfin, nous avons utilisé $\mathcal{M}$*oise* pour pour modéliser et gérer des communautés virtuelles ouvertes et décentralisées en suivant le modèle décrit dans le chapitre 5.

Grâce à l'utilisation de JaCaMo, l'architecture en couches illustrée dans la figure 8.6 se trouve être en parfaite adéquation avec les quatre dimensions que comporte le modèle présenté dans le chapitre 5.

Premièrement, la couche des agents est composée des agents du système (i.e. $\mathcal{A}$) tel qu'on les avaient décrit dans la Section 5.4. Afin de rendre notre système indépendant de toute application métier, nous avons fait le choix de considérer deux types d'agents: *assistants* et *gestionnaires de confiance*. Un *assistant* est un agent dont la principale tâche est d'assister un membre de la communauté dans ses activités. Ainsi, cet agent est doté de but spécifiques à l'application métier cible. Par ailleurs, l'agent *gestionnaire de confiance* encapsule l'architecture du système de gestion de la confiance présentée dans le chapitre 7 et de ce fait est responsable des tâches liées à la gestion de la confiance (i.e., instanciation, négociation, combinaison, intégration et évaluation des politiques de confiance). L'objectif principal de cet agent est donc d'assister l'agent assistant dans ses décisions de confiance.

Deuxièmement, dans la couche environnement (c.f. la couche basse de la figure 8.6) nous avons placés les ressources que les agents assistant doivent manipuler dans le cadre de leurs activités. Ces ressources ont été implémentes en tant qu'artefacts *CArtAgO* comme évoqué plus haut.

Troisièmement, les agents sont groupés au sein de structures organisationnelles qui représentent les communautés. Chaque communauté est définie à travers sa spécification organisationnelle (c.f. $\mathcal{M}$*oise*) ainsi que l'ensemble des ressources partagées au sein de cette communauté. Nous avons utilisé une couleur différente pour différencier les communautés dans la figure 8.6. Ici la couleur indique la communauté à laquelle appartient chaque ressource.

Enfin, dans la figure 8.6, les interactions n'ont pas de couché dédiée car cette dimension est transversale aux trois autres. Ainsi, les agents communiquent entre eux grâce à un protocole de communication ou un protocole de négociation. Ils interagissent avec les ressources via les opérations et ils peuvent interagir avec leur communauté via la spécification organisationnelle.

8.7 Détails d'implémentation

Dans cette section, nous décrivons brièvement comment les éléments de l'architecture décrite plus haut (c.f., Figure 8.6) ont été implémentés en utilisant la technologie *JaCaMo* correspondante.

8.7.1 Implémentation des agents

Dans cette section, nous présentons comment l'agent assistant et le gestionnaire de la confiance sont implémenté. nous sommes particulièrement intéressés par montrer quelles fonctionnalités sont assurées par chaque type d'agent.

8.7.2 Agent Assistant

L'agent assistant a pour objectif d'aider les membres de communauté virtuelles dans leurs activités. Pour cela, l'agent repose sur l'architecture illustrée dans la Figure 8.7.

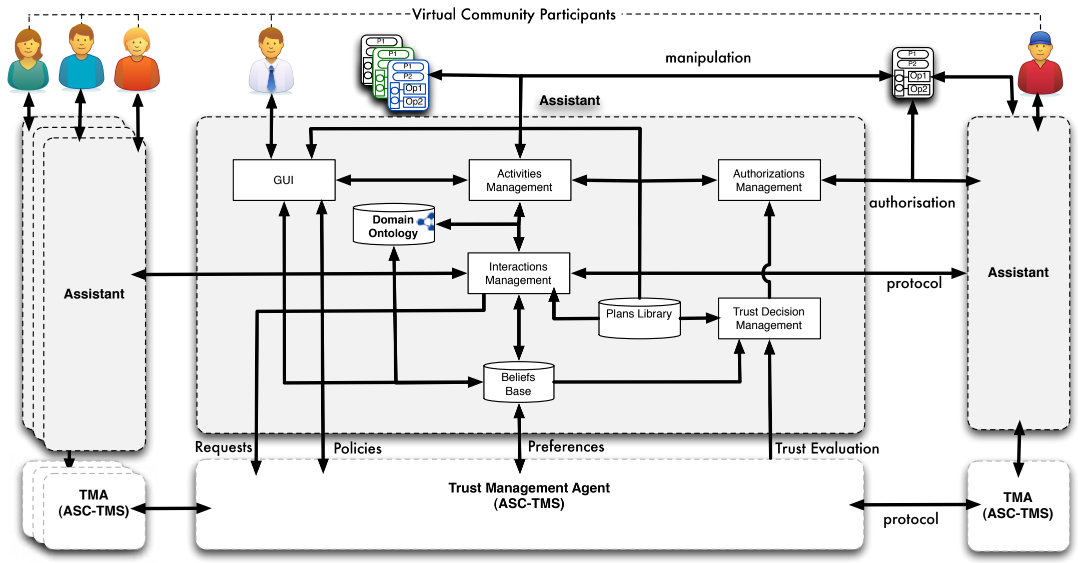


Figure 8.7 – Architecture de l'agent assistant

Les fonctionnalités offertes par cet agent sont au nombre de quatre et nous les résumons comme suit :

- *activities management* : cette fonctionnalité permet à l'agent de réaliser des objectifs métier. Ainsi, ce module encapsule les plans (écrits en *Jason*) qui permettent à l'assistant de créer, rejoindre et quitter les communautés. Ces plans lui permettent également de créer, manipuler et détruire des ressources.
- *interactions management* : cette fonctionnalité contient des plans lui permettant d'interagir avec les autres agents assistant en échangeant de messages.
- *trust decision management* : cette fonctionnalité encapsule les plans à partir desquels l'assistant est en mesure de prendre des décisions à la place de l'utilisateur humain. Ces plans représentent de ce fait le modèle de confiance évoqué au chapitre précédent.

- *authorisation management* : une fois qu'une décision de faire confiance a été prise, cette fonctionnalité permet à l'assistant d'établir des autorisations qui vont permettre à l'agent à qui il a décidé de faire confiance de manipuler la ressource convoitée.

Chacune des quatre fonctionnalités décrites plus haut est concrètement implémentée grâce à des plans *Jason*. Nous invitons le lecteur à voir les exemples illustrant ces fonctionnalités dans le manuscrit original.

8.7.3 Gestionnaire de la confiance

Dans la section précédente, nous avions décrit l'agent assistant était en mesure de prendre des décisions à la place des membres humains de la communauté en se basant sur le travail du gestionnaire de confiance. Dans cette section, nous allons donc voir comment ce dernier est conçu pour permettre ce travail.

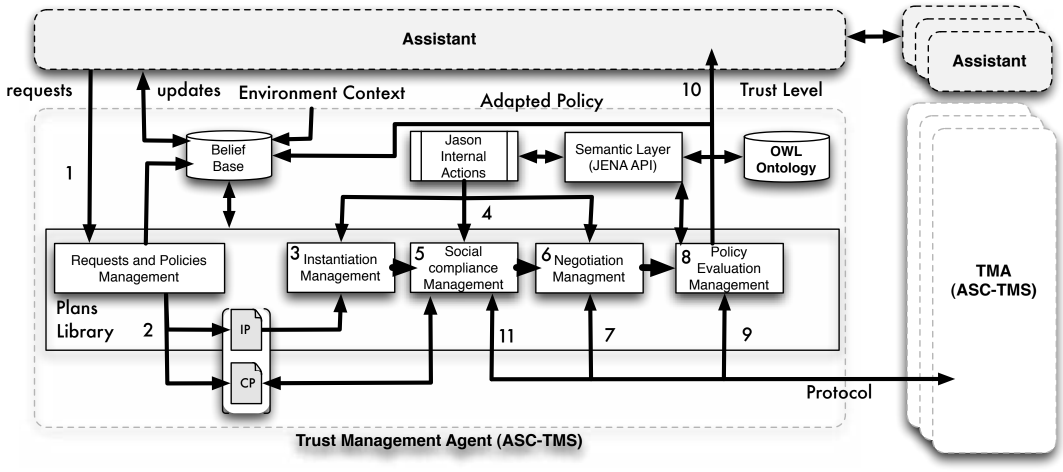


Figure 8.8 – Architecture du gestionnaire de confiance

Comme l'illustre la figure 8.8, le travail du gestionnaire de confiance peut être décrit à travers cinq fonctionnalités qu'on résume comme suit:

- *Requests management*: cette fonctionnalité assure le traitement des requêtes reçues. Pour cela, les requêtes seront interprétées et les politiques correspondantes associées.
- *Instantiation Management*: cette fonctionnalité est assurée grâce aux méta-politiques d'instanciation vues dans le chapitre 9.
- *Social Compliance Management*: cette fonctionnalité assure la combinaison, l'intégration et l'évolution des politiques. Pour cela, le gestionnaire repose sur les méta-politiques correspondantes ainsi que les politiques individuelles et les politiques collectives.

- *Negotiation Management*: cette fonctionnalité assure la négociation avec les autres gestionnaires. Des méta-politiques dédiées sont utilisées ici aussi en conjonction avec le protocole de négociation présenté dans le chapitre 8.

- *Policy Evaluation Management*: cette fonctionnalité assure enfin l'évaluation des politiques.

Là aussi, les fonctionnalités ont été implémentée sous forme de plans *Jason*. Le format du résumé ne permet pas de les présenter c'est pourquoi nous invitons le lecture à consulter les exemples fournis dans le chapitre 8

8.7.4 Implémentation des ressources

Les artefacts *CArtAgO* sont utilisés pour implémenter les ressources de notre système $\mathcal{S}$. Les artefacts sont caractérisés par un ensemble de propriétés observables et non-observables, un ensemble d'opérations ainsi qu'une désinscription fonctionnelle.

Les attributs privés de l'artefact représentent les propriétés observables, alors que les attributs publics représentent les propriétés non-observables.Enfin, les méthodes constituent les opérations fournies par chaque type de ressource. Dans ce qui suit, nous présentons comment les concepts utilisés pour décrire les ressources dans notre modèle (cf. Chapitre 5) sont traduits en terme de concepts *CArtAgO*.

- Les propriétés observables des artefacts ont été utilisé pour implémenter *le propriétaire* ($r_i.\varphi$), *la sensibilité* ($r_i.\varsigma$) et *la valeur* ($r_i.\nu$) d'une ressource. Dans notre implémentation, la sensibilité et la valeur d'une ressource sont matérialisés par des entiers alors que la propriété *propriétaires* est implémentée par une liste (i.e. `ArrayList` contenant les identifiants des propriétaires d'une ressource. Les propriétaires peuvent être nombreux lorsqu'il s'agit d'une ressource collective ce qui explique ce choix de structure de données.

- Les propriétés non-observables sont des attributs Java classiques. Ils sont utilisés pour implémenter le contenu ($r_i.\theta$) des ressources.

- Les propriétés observables tout comme les propriétés non-observables sont accessibles via des opérations. Ainsi, la correspondance entre le concept opération utilisé dans notre modèle et les opérations utilisés dans *CArtAgO*.

- Enfin, la fonction d'autorisation $r_i.Auth$) introduite dans la Section 6.8 est également implémenté avec une opération spéciale appelée `grant`.

8.7.5 Implémentation des communautés

Dans $\mathcal{M}oise$, les spécifications organisationnelles sont implémentés en utilisant des artefacts appelés *boards*. Par défaut, il en existe 3 dans chaque organisation: *OrgBoard*, *GroupBoard* et *SchemeBoard*. Dans notre implémentation, nous avons décidé de regrouper ces trois artefacts en un unique `ComArt` (i.e. Community Artefact). En plus des fonctionnalités fournies par ces artefacts, notre `ComArt` contient les politiques collectives utilisées par les membres de la communauté ainsi que les opérations d'adaptation qui permettent aux membres de ces communautés d'y accéder et de les adapter. Engin, le `ComArt` encapsule également une opération *grant* à travers laquelle le propriétaire de la communauté peut octroyer des privilèges à d'autres agents. Enfin, on assume que lorsqu'une communauté est créée par plusieurs agents, seule un parmi eux exécute le plan permettant sa création par contre tous les agents sont considérés comme étant les propriétaires de la communauté. Pour illustrer cette étape de création, nous présentons ci-dessous le plan utilisé par un agent pour créer une communauté donc le nom est `ComName`.

```
1 @CommunityCreation
2 +!createCom(ComName) <-
3           makeArtifact(ComName, "ComArt", ["community-os.xml", ComId);
4           adoptRole(owner)[artifact_id(ComId)];
5           .my_name(Me);
6           +owner(ComId, Me);
7           +myCom(ComId);
8           !admitMembers(ComName, ComId).
```

Listing 8.32 – Community Creation Plan

Dans ce plan, l'agent assistant crée une communauté en instanciant un artefact de type `ComArt`. Comme tout artefact, le `ComArt` est créée en utilisant l'action `makeArtifact` (ligne 3). Une fois la communauté créée, l'assistant adopte le rôle de propriétaire (Ligne 4) puis se met en attente de requêtes d'admission émanant d'autres agents (Ligne8).

8.7.6 Implémentation des interactions

Les interactions des agents avec leur environnement sont assurées par *CArtAgO* à travers les opérations sur les artefacts. Par contre, les interactions entre agents sont assurées par *Jason*. Comme évoqué dans la Section 5.6, les interactions entre agents s'opèrent via des échanges de messages. Chaque message recu ou envoyée par un agent à la structure suivante [Bordini et al., 2007]:

$$\langle receiver, performative, content \rangle \tag{8.3}$$

Le `sender` est un atome représentant l'identifiant de l'agent recevant le message, le `performative` dénote l'intention de l'émetteur (i.e., force illocutoire) et le contenu est un prédicat dont l'interprétation dépend du performative utilisé. L'enchaînement ainsi que l'ordre des performatifs est défini par le protocole utilisé par les agents. Ainsi, c'est le protocole qui va définir si le message envoyé fait partie d'une session de communication normale ou dans le cadre d'une négociation.

Nous invitons le lecture à consulter le manuscrit pour plus de détails à propos de comment les communications et les interactions sont traduites en terme de plans *Jason*.

Chapter 9

ASC-TMS Application to Open Innovation

Previous chapters presented different aspects of ASC-TMS model, which include a generic multi-agent based virtual community framework (Cf. Chapter 5), an expressive and flexible (individual and collective) policy specification language (Cf. chapter 6) and a policy adaptation mechanism (Cf. chapter 7). Finally, Chapter 8 described the implementation and deployment of ASC-TMS on the *JaCaMo* multi-agent programming platform.

In this chapter, we *emphasise* the applicability of ASC-TMS on real life *virtual communities*. To that aim, we present how ASC-TMS can be used to manage trust in *open innovation* virtual communities. This use case has been selected due to our implication in the WINPIC project funded by the Saint-Etienne Metropole [1].

The objective is to demonstrate how ASC-TMS may be applied in diverse settings to support trust decision making in these systems.

9.1 Open Innovation

Open Innovation is currently recognised as an exciting new way to generate breakthrough innovation at lower cost and fast time. This approach is adopted by companies and organizations to enhance innovation in their R&D departments by harnessing external ideas. This approach was made famous by Henry Chesborough, in his influential book Open Innovation [Chesbrough, 2006] in which he argues that " in today's information rich environment, companies can no longer afford to rely entirely on their own ideas to advance their business" [Chesbrough, 2012b, Chesbrough, 2012a].

Open innovation paradigm treats research and development as an open system [Ahonen and Lietsala, 2007, Vega, 2012]. It suggests that valuable ideas can come from inside and outside companies. So by adopting open innovation, companies and organisations place internal and external ideas at the same level of importance. Open innovation is often conflated with the *open source* approach for software development. Indeed, both approaches rely on the abundance of external source of idea to create value. However, open innovation incorporate the business model in the innovation process, while *open source* approach downplays or denies

[1] https://iscod.emse.fr/winpic/

it [Cooke, 2009, Huizingh, 2011]. With respect to this issue, participants in *open innovation* process are more concerned by *trust* as the business model represents the source of both value creation and capture.

The open innovation activity involves two parties and an intermediary; *Seekers* are looking for solving problems and *Solver* aims at providing solutions to these problems. Traditionally, the role of intermediary has been performed by technology brokers [Vega, 2012]. However, in the last few years, a number of commercial (e.g. Hypios, Nine Sigma or Innocentive) and free (e.g. W3C Community Group) open innovation platforms (called markets) have emerged, often making extensive use of on-line resources [Cooke, 2009]. These platforms plays the role a *technological*, *business* and *trust* intermediary. *Technological* intermediary as they provide tools for collaboration and resource sharing, *business* intermediary as they constitute a showcase for R&D problems, and *trust intermediary* as they act as a trusted third party between *seekers* and *solvers*.

In the next section, we present an *open innovation* virtual community example that we will rely on throughout this Chapter to illustrate the application of ASC-TMS. In this example, we will focus on the trust relationship between *solvers* which is not managed in current *open innovation platforms*. The drawback has been highlighted by recent feedbacks from a number of *solvers* [Cooke, 2009]. In these feedbacks, most of the solver confess that *distrust* was the major factor affecting their decision to discontinue their involvement with innovation platforms.

9.2 Illustrative Example: An Open Innovation Community

In this running example, we consider an open innovation system composed of 13 participants (cf. Figure 9.1). For simplicity, we assume that one participant is playing the role of *seeker* (i.e. michael) and the other participants are playing the role of *solver*.

As illustrated in 9.1, the scenario starts when the seeker proposes a new challenge. A challenge is a problem to which the seeker (i.e. Michael) is willing to give a reward against a solution that satisfies its requirements. Defining how a solution is considered satisfactory by the seeker is out of the scope of this thesis,

Defining how solutions are selected and rewarded is out of the scope of this thesis, but we can imagine that proposed solutions are evaluated by experts and ranked based on their fulfilment of the initial objective of the challenge. Likewise, we do not manage the *trust* relationship between *solvers* and *seekers* as suggested in the previous chapter.

The scenario starts after the introduction of a new challenges. Based on their propensity to collaborate, solvers are split into two categories; *collaborative* and *non collaboratives*. *Collaboratives* solvers tend to prefer forming communities in which they engage to address the submitted challenger collaboratively, while *non collaborative* solvers prefer to work alone. In this scenario, we will focus on *collaborative* solvers but we admit that the system includes *non*

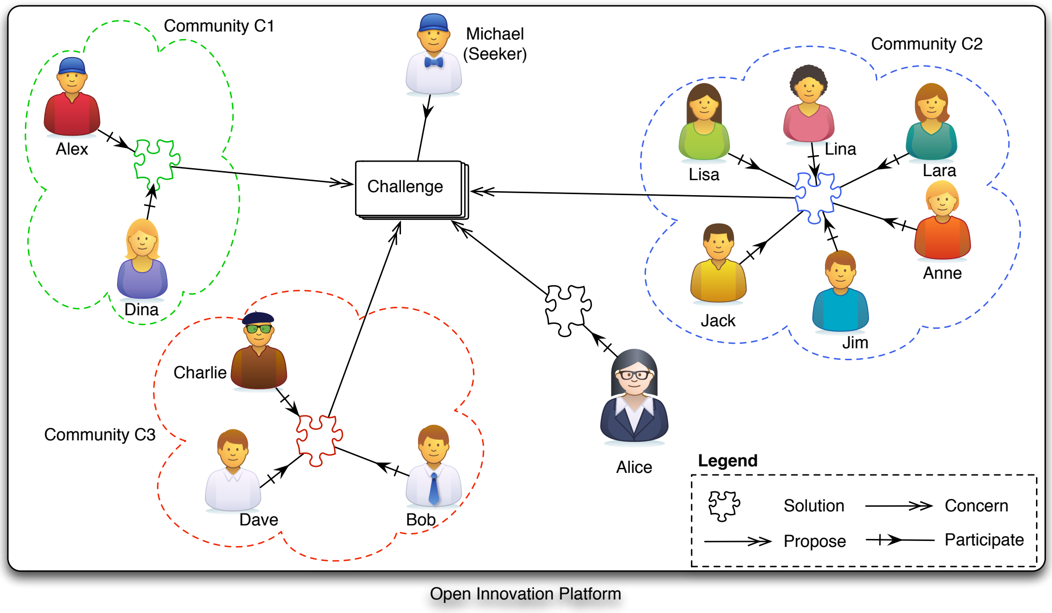


Figure 9.1 – An instantiation of an open innovation challenge

collaborative ones.

So *collaborative* solvers tend to group together and form communities in which the challenges are solved collectively. This task is performed by the *assistant agent* but it is the *human* members which decides at each step. The process of creating a community has not bean investigated in our research. But we present hereafter a simplified protocol that we imagined in order to *emphasise* the trust management issue in this process.

9.2.1 ASC-TMS for Communities Creation

In sum, we assume that participants are able to make estimation about how many person are needed to solve the submitted problem. This estimation is made based on each one's own capabilities (i.e. degree of competence). Then these members make use of their *assistant agent* to broadcast their estimation to the other participants in order to find potential community fellows. Based on these estimation, communities are group are formed but not yet communities. Indeed, *participants* with the same estimation won't be able to until they fulfil two conditions: (1) they mutually satisfy each others collaboration policy, and (2) they succeed in specifying collective policies [Pearlman et al., 2002].

At this stage, already, appears the initial need for ASC-TMS. In the next section, we will illustrate the benefits using ASC-TMS to fulfil the above conditions. To that aim, we illustrate this issue with the formation of the community $C3$ depicted in Figure 9.1. The community is not formed yet and there are four potential candidates that all made an estimation for a community composed of three members. These participants are are *Alice, Bob, Charlie* and *Dave*.

9.2.1.1 Policies Evaluation with the ASC-TMS

Once they consider creating the community, *Alice, Bob, Charlie* and *Dave* enter in a pairwise trust establishment phase in which each member tries to satisfy the other participant collaboration policy. In order to illustrate this process, we present how *Bob* succeed in satisfying *Dave*'s policy and how *Alice* failed in this issue. To that aim, we present the collaboration policy used by *Dave* in Listing 9.1

```
policy([
lbc(mail.identity,“>=”, marginal, 5, m)
lbc(skilfulness, “>=”, fair, 2,m),
lbc(reputation, “>=”, 70 ,2,o),
lbc(recommendation, “>=”, 3,1,o)
])[issuer(dave), pattern(collaborate, _)].
```

Listing 9.1 – Dave’s collaboration policy

In the above policy, *Dave* makes use of four trust factors, namely *identity*, skilfulness, reputation and experience. *Dave* considers only lower-bound conditions and requires that the identity of its trusted collaborators must be at least marginal (with respect to the PGP classification) and that the credential provided to prove it should be her mail address He also requires a fair level of skilfulness. In this scenario, we suppose that there exist an certification entity which certifies the skilfulness of the solvers based on their solving capabilities. The certification is a four level degree (none, bad, fair, good, very good) that the solver receives as a credential. These conditions are mandatory, while the two remaining ones are optional. In these conditions, *Dave* requires a 70 % reputation value and three recommendations. Finally, *Dave* associates different weights to each of the above conditions.

In this demo, we make use of a simple reputation model in which an agent computes the reputation of another agent based on the simple equation:

$$Reputation = (\frac{P+1}{T+2}) \times 100 \tag{9.1}$$

Where P represents the count of past positive experiences and T the total amount of past interactions. So the reputation given to a member with whom we don’t have any prior experience is 0.5. It is also the bootstrap value of the reputation in the scenario.

In order to satisfy *Dave’s* policy, *Alice* privided the credentials presented in 9.2 and *Bob* provided the ones of Listing 9.3.

```
credential(mail.identity, alice, unknown)[issuer(authority1), type(“PGP”)].
credential(certificate.skilfulness,Alice, good)[issuer(authority2), type(“X.509”)].
```

Listing 9.2 – Credentials provided by Alice

```
credential(mail.identity, bob, good)[issuer(authority1), type(“PGP”)].
credential(certificate.skilfulness,bob, fair)[issuer(authority2), type(“X.509”)].
```

Listing 9.3 – Credentials provided by Bob

For simplicity, we only presented the informational consequence of receiving a credential. In fact, once the TMA receives the credential he adds the above beliefs based on which he will evaluate its policy. So the policy evaluation is performed based on beliefs about declarations and credentials the TMA received.

In addition to the above credentials, *Bob* make use of declarations provided by other participants. These declarations are aggregated to compute an unique value for each property. The outcome of this process is presented in Listing 9.4.

```
declaration(reputation,Alice, 50).
declaration(recommendation,Alice,0).
declaration(reputation, bob, 60).
declaration(experience,bob, 2).
```

Listing 9.4 – Declarations collected by Bob

Based on the above policy and the acquired information, the trust management agent uses the policy evaluation function presented in Definition 61 (Cf. Section 6.4 of Chapter 6). The evaluation process undergone by this function for the assessment of Alice's trust level is presented in figure 9.2 hereafter.

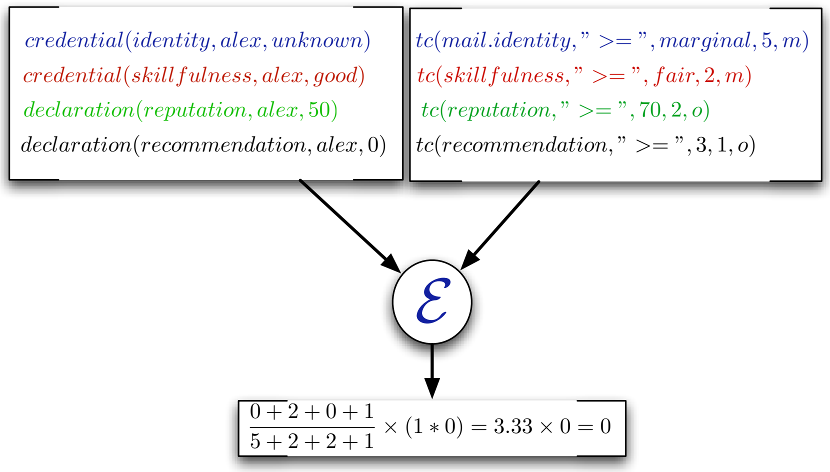

$credential(identity, alex, unknown)$
$credential(skillfulness, alex, good)$
$declaration(reputation, alex, 50)$
$declaration(recommendation, alex, 0)$
$tc(mail.identity, ">=", marginal, 5, m)$
$tc(skillfulness, ">=", fair, 2, m)$
$tc(reputation, ">=", 70, 2, o)$
$tc(recommendation, ">=", 3, 1, o)$
$\mathcal{E}$
$\frac{0+2+0+1}{5+2+2+1} \times (1 * 0) = 3.33 \times 0 = 0$

Figure 9.2 – Evaluation of Alice's trust level by Dave

Here, the computed trust level equals 0 because *Alice* failed satisfying the identity trust criteria that *Dave's* considers to be mandatory. Similar to this process, the four participants (i.e. *Alice, Bob, Charlie* and *Dave*) compute the trust they put in each others. Table 9.1 below, summarizes the trust level computed by each member. Based on the above table, *participants*

Solvers	Alice	Bob	Charlie	Dave
Alice	-	0.6	0.8	0.9
Bob	0.4	-	0.7	0.7
Charlie	0.5	0.5	-	0.6
Dave	0	0.7	0.6	-

Table 9.1 – Summary of the evaluation of the four agent's policies

decides to build a community with the most trusted participants. In our example, the group of agents that can potentially form the community $C3$ reveals to be *Bob, Charlie* and *Dave.*

9.2.1.2 Policies Combination with the ASC-TMS

In the previous section, we saw how ASC-TMS is used to form a group based on which a community can be created. However, this community will not take place until the members builds the collective policies they will use for their trust decision. To that aim, these participants will rely on ASC-TMS and its *combination* operator to build these policies. The policy combination process is achieved using the `combine()` internal action defined in Section 8.3.1.2 based the combination process described in Section 7.5.1. Figure 9.3 illustrates how the process can be instantiated during the creation of the community C3.

The obtained policy constitute a combination of the three policies. In this example, the agents used the heuristic *h1* (cf. Section 7.5.1). So the resulting collective policy is composed of trust criteria that are at least as restrictive as the trust criteria stated in the policies of the three participants. Thus the resulting policy will never accept a request that one of the participants would have rejected.

9.2.2 Request Management with the ASC-TMS

Previous section described how helpful ASC-TMS can be in situation of community creation in order to bootstrap a minimum level of trust between its members. However, requests management remains the primary situation in which the use of ASC-TMS can be better stressed.

In this section, we will consider the scenario in which *Dave* has to make a decision with respect to a request that originates from *Alice*. *Alice* wants to read a common resource shared among C3 members. This request belongs to the pattern $pattern(read, text)$.

The second request concerns *Bob* who wants to read a personal resource owned by *Dave.* Here, both requests belongs to the same pattern that is So based on this pattern, the TMA

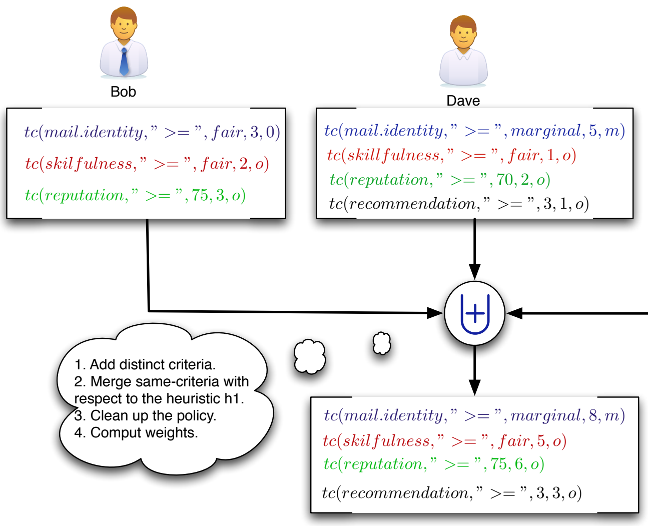

Bob
Dave
Charlie
tc(mail.identity," >= ", fair, 3, 0)
tc(skilfulness," >= ", fair, 2, o)
tc(reputation," >= ", 75, 3, o)
tc(mail.identity," >= ", marginal, 5, m)
tc(skillfulness," >= ", fair, 1, o)
tc(reputation," >= ", 70, 2, o)
tc(recommendation," >= ", 3, 1, o)
tc(skilfulness," >= ", fair, 2, o)
tc(reputation," >= ", 65, 1, o)
tc(recommendation," >= ", 2, 2, o)
1. Add distinct criteria.
2. Merge same-criteria with respect to the heuristic h1.
3. Clean up the policy.
4. Comput weights.
tc(mail.identity," >= ", marginal, 8, m)
tc(skilfulness," >= ", fair, 5, o)
tc(reputation," >= ", 75, 6, o)
tc(recommendation," >= ", 3, 3, o)

Figure 9.3 – Illustration of the policy combination during the creation of Community C3

is able to retrieve the policy that handles this kind of request. In its policy, *Dave* uses three trust factors, namely identity, reputation and cooperativeness. Using these trust factors, *Dave* specified three lower-bound trust criteria which threshold values and weights are presented in Listing 9.5.

```
policy([
tc(mail.identity,“>=”, fair,2,m)
tc(reputation,“>=”, 70 ,2 ,o),
tc(cooperativeness,“>=”, 65, 3, o)
])[issuer(dave), pattern(read, text)].
```

Listing 9.5 – Policy used by Dave for the pattern read-text

In the following, we illustrate how the TMA will *instantiate* this policy with respect to the context in which the interaction is undertaken, adapts the policy during the negotiation phase, and integrates it with the collective policy of the community C3.

9.2.2.1 Instantiation

Now we present two sub-scenarios in which the selected policy is instantiated with respect to environmental pressures, namely resource value and reputation collusions. The adaptation meta-policy used to illustrate these scenarios are listed here below.

```
+!instantiate(Policy[pattern(Pattern)], Agent, Operation, Object) :
        value(Old)[pattern(Pattern)] & valueOf(New)[artifact_name(Object)] & New > Old
        <− Delta = New − Old ;
        for(Delta>0){
            policy.restrict(Policy, all);
            Delta = Delta −1;
            }
            for(Delta<0){
            policy.restrict(Policy, all);
            Delta = Delta + 1;
            }
            −+value(New)[pattern(Pattern)].
```

Listing 9.6 – Adaptation meta-policy for resource value

This meta-policy has already been presented in Section 8.3.1.2 of the previous chapter. With this above meta-policy the selected policy is instantiated with respect to the value of the requested resource. The hypothesis that motivates this meta-policy is that the more a resource is valuable the more its policy becomes restrictive. Analogously, the less a resource is valuable the less its policy should be restrictive. With this meta-policy, the restrictiveness of the policy is general. So all trust criteria used in the policy are adapted with more restrictive values. The resulting *instantiated* after policy is illustrated below.

```
policy([
lbc(mail.identity,marginal,2,o)
lbc(reputation, 71 ,2 ,o),
lbc(cooperativeness, 66 , 3, o)
])[issuer(dave), pattern(read, text)].
```

Listing 9.7 – Instantiated Policy

In the same way, another meta-policy has been defined to make the policy react to collusions attacks. These attacks affect the way the participants consider the *testimonials* provided by other participants.

```
+!instantiate(Policy[pattern(Pattern)], Agent, Operation, Object) :
value(collusion, Old, New) &
threshold(collusion, T) & C = New − Old & C >= T
<−for(C>0){policy.restrict(ReqId,reputation), C=C−1;}
```

Listing 9.8 – Adaptation meta-policy for reputation collusions

In the above policy, when collusions attacks are detected, ASC-TMS restricts all trust criteria making use of *indicators* (i.e. testimonials). In its policy, *Dave* makes use of two types of *indicators*, namely *reputation* and *cooperativeness*. Consequently, the value of these trust criteria are made more restrictive. The resulting policy is illustrated below.

```
policy([
lbc(identity,mail,2,o)
lbc(reputation, 75 ,2 ,o),
lbc(cooperativeness, 70 , 3, o)
])[issuer(dave), pattern(read, text)].
```

Listing 9.9 – Instantiated Policy

ASC-TMS makes use of several meta-policies that behave in a similar way. Some of them are application specific and hence should be specified by the user, while others are generic and can be considered as system level meta-policies.

Importantly, we specified rules to update and maintain the beliefs base on which instantiation meta-policies operate. Listing 9.10 describes an example of such rules and beliefs.

```
1 threshold(collusion, 2).
2 value(collusion,New, Collusion)[Pattern]:−
3 .count(alert(collusion)[source(Agent), Pattern], Collusions) &
4 value(collusion, Old, New)[Pattern] & reputation (Agent, R) & R > 0.5).
```

Listing 9.10 – Beliefs and rules used to build and maintain the environmental context

Here, the belief `threshold(collusion, 2)` represents the tolerance threshold for collusion incidents. These beliefs indicates to the TMA that if the number of collusions reported increased by two since the last interaction (with the same pattern), the reputation threshold value should increase accordingly. The rule (Line 2-4) updates this belief based on the collusions reported by agents which reputation is above 0.5 (i.e. 60 %). With this mechanisms, an agent can filter collusion reports.

9.2.2.2 Integration

Once the policy has been instantiate, ASC-TMS integrate the collective policy of the community to which *Dave* belongs (i.e. community $C3$). To proceed, the *operation* `integrate(...)` that implements the integration algorithm described in Section 7.5.2 is used. The figure 9.4 illustrates the integration of the individual policy used by *Dave* with the collective policy used by its community.

In our example, *Dave* is willing to comply with the collective policy as long as the community remains attractive. To that aim, he use a rule to compute the attractiveness of the community to which he belongs. He uses also a beliefs by means of which he states to the TMA what is the threshold value to consider a community attractive. Based on this, *Dave* specified the collective policy illustrated in Listing 9.11.

```
!integrate(IP[pattern(Pattern)], Agent, Operation, Object) : complianceProfile(compliant) &
                                    attractiveness(C, Att) &
                                    threshold(attractiveness, TA) & Att >= TA
                                    myCommunity(Com) & not member(Agent, Com)
                                   <− ?collectivePolicy(CP)[pattern(Pattern), issuer(Com)];
                                   policy.integrate(CP, IP, Pattern, h3).
```

Listing 9.11 – Adaptation meta-policy for resource value

Using the above meta-policy, ASC-TMS used by *Dave* guarantees that *Dave* will not accept any request which the collective policy would not have accepted. The resulting policy is illustrated in Listing 9.12.

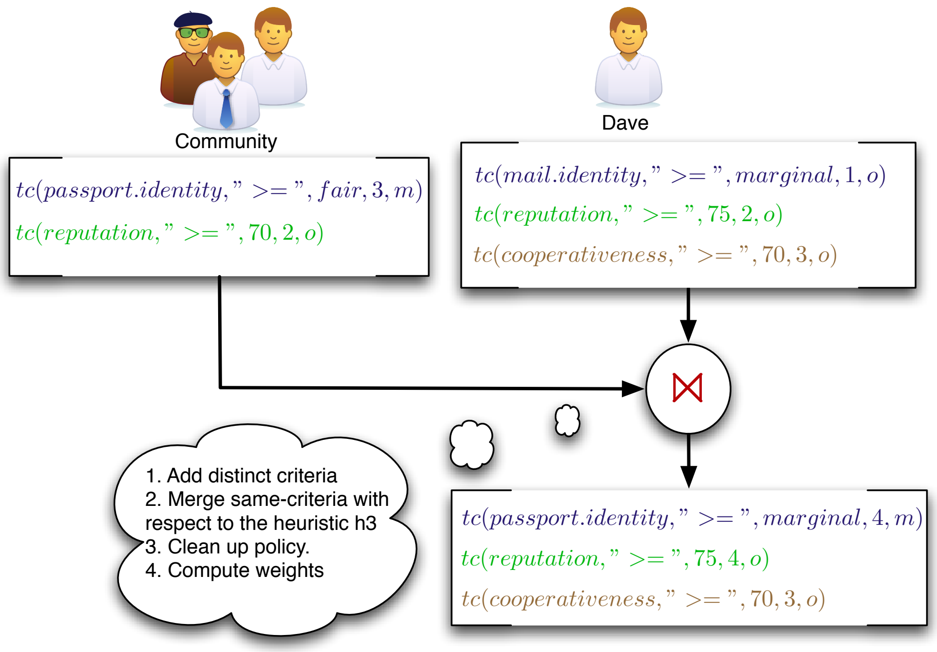

Community
Dave
$tc(passport.identity, ">=", fair, 3, m)$
$tc(reputation, ">=", 70, 2, o)$
$tc(mail.identity, ">=", marginal, 1, o)$
$tc(reputation, ">=", 75, 2, o)$
$tc(cooperativeness, ">=", 70, 3, o)$
1. Add distinct criteria
2. Merge same-criteria with respect to the heuristic h3
3. Clean up policy.
4. Compute weights
$tc(passport.identity, ">=", marginal, 4, m)$
$tc(reputation, ">=", 75, 4, o)$
$tc(cooperativeness, ">=", 70, 3, o)$

Figure 9.4 – Illustration of Dave's policy with the collective policy used in C3

```
policy([
tc(passport.identity,“>=”, marginal ,4 ,m),
tc(reputation,“>=”, 75 ,4 ,o),
tc(cooperativeness,“>=”, 70 , 3, o)
])[issuer(dave), pattern(read, text)].
```

Listing 9.12 – Integrated Policy

So as long as the community *C*3 remains attractive, *Dave*, as a compliant member, will comply with the collective policy. This behaviour ensures the community cohesion and avoids conflictual decisions. However, using the heuristic h3 means also that the integrated tend to be more restrictive which can be a brake for the *compliant agent* as we will see in the next section.

In this example, *Dave* managed a request originating from a participant that is not a member of the community. However, one can imagine that *Dave* would have to handle request from participants that are inside the community. Of course, these members will not request *Dave* access to collective resources as they are granted access to them *de facto*. Instead, we are more interesting in the situations in which a member of a community request access to *Dave* individual resource. In such situation, one can think that *integration* would be useless for *Dave*. However, the requesting member (e.g. *Bob*) would expect that *Dave* will use a policy that is not too much restrictive compared to the collective policy they agree on. However, *h*3 do not prevent *Dave* from having a policy which is too restrictive if his policy is more restrictive than the collective policy. Here, the most appropriate policy integration heuristic to be used would be $h2$. Indeed, this heuristic guarantees that the integrated policy would never reject any request that would be accepted by the collective policy. That means that *Dave* need to have another meta-policy which is dedicated to requests originating from the members of its community.

9.2.2.3 Negotiation

At this stage, *Dave* generated an integrated policy that he will use with his interlocutors. In this section, we show the negotiation that *Dave* will conduct with *Alice*. The one with *Bob* proceeds in an analogous way. The negotiation between *Dave* and*Alice* is depicted in Figure 9.5

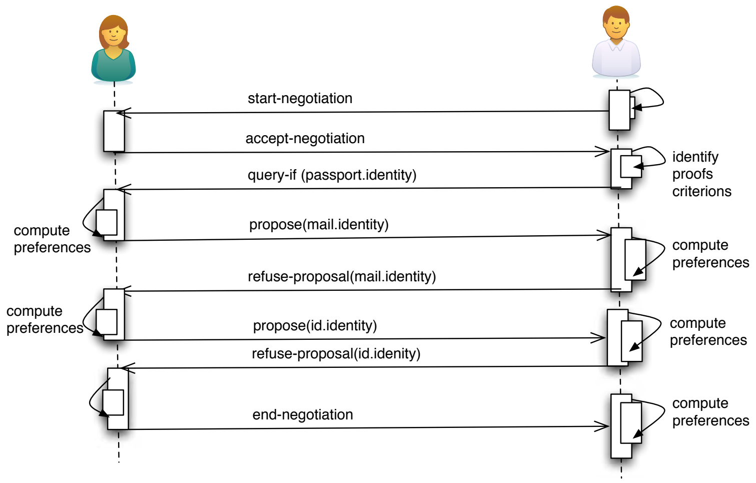


Figure 9.5 – The negotiation between Dave and Alice

In this example, each time *Dave* has to make a decision he builds the extensive game tree in order to select the best alternative (cf. Figure 9.6). The value represent the payoffs of each player (i.e. Dave and Alice) computer using the *expected utility function* defined in Definition 77. Howe these values are not very relevant. All what we need to show is that at some point, each participant is able to compute the utility of each of the moves he can make in the game and selects the moves that maximises its utility.

In this game, *Alice* is trying to minimise her privacy exposure by proposing the least sensitive credentials that proves her identity (i.e. mail with sensitivity of 1). However, *Dave* will systematically refuse the counter-offers of *Alice* because the initial identity he requested (i.e. passport) is one of conditions stated by the collective policy. Thus when *Dave* computes his preferences, the *punishment* cost he will have to assume if he accepts *Alice* proposal exceeds the potential utility he will gain by providing *Alice* access to the resource. Thus, the only

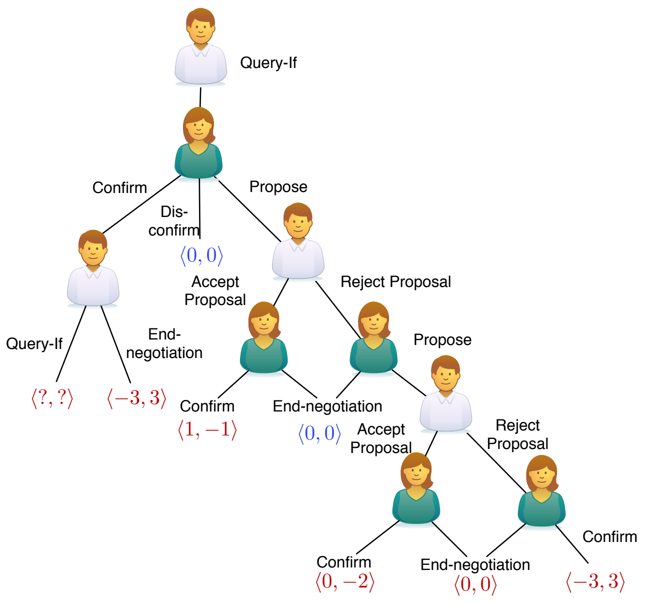


Figure 9.6 – The negotiation game between Dave and Alice

alternative that *Dave* would have accepted would be the case in which *Alice* confirms that she will provide the requester passport credential. However, *Alice* is not willing to accept it because the privacy breach exceeds the utility she may gain by accessing the resource.

No matter what *Alice* will provide as credentials, the evaluation of *Dave′s* policy will always compute to zero as the trust criterion on which the negotiation was performed was mandatory. Consequently, *Dave* will systematically refuse *Alice′s* request, as well all requests originating from individuals that will refuse to deliver their *passport* credential.

9.2.2.4 Evolution

In the previous section, *Dave* was unable to achieve its negotiation due the restrictiveness of the collective policy of its community. If such situation becomes frequent, *Dave* will either violate the *compliance norm* of its community or ask for the collective policy evolution. Here, *Dave* is a compliant agent so he will always prefer to make his community follows adapt their collective policy, otherwise he prefers to quit the community than violating the community norms (and thus the collective policy).

We suppose that *Dave* detected in the last example that the *identity* trust criterion of the collective policy was too restrictive. This observation can be the fruit of a careful count of interactions that *Dave* refused because of this criterion. Based on this observation, *Dave* maintains a belief that he will use as a condition in a meta-policy that aims at relaxing such criterion. This meta-policy is illustrated in Listing 9.13 below.

```
+!evolve():
failures(Pattern, Criterion, Adaptation, Count) & Count > 3
<− .broadcast(tell, evolve(relax, Criterion); .wait(1000);
!countVotes(Pattern, Scheme).
```

Listing 9.13 – Evolution meta-policy

With the above meta-policy *Dave* states that he is willing to miss up to 3 interactions because of the collective policy. So after three missed interactions, *Dave* will systematically try to change the collective policy of its community using the voting scheme presented in Section 7.5.3. Worth noting that this meta-policy is generic, so it will be triggered for any adaptation. In the above example, the "Adaptation" refers to the "relaxCriterion". To make the collective policy evolve, *Dave* will send a request to its community follows to know whether they are keen to adapt the collective meta-policy as stated by the above meta-policy.

Once the other agents receive *Dave's* proposal to vote for the collective policy evolution, they will evaluate whether they have been also restricted by the same criterion at least one time (i.e. $Count >= 1$). If so, the agents will accept the proposed adaptation, otherwise they will refuse it. Such behaviour is also stated using a meta-policy which we illustrate in the following meta-policy.

```
+evolve(Pattern, Criterion, Adaptation):
failures(Pattern, Criterion, Adaptation, Count) & Count >= 1
<− .broadcast(tell,agree(Pattern, Criterion, Adaptation)).
```

Listing 9.14 – Collective Decision Taking

Based on the other agents responses, *Dave* and each member of the community triggers the voting scheme to evaluate whether the community agree about evolving the policy as suggested by *Dave* or not (cf. Section 7.5.3). To that aim, each agent makes use of the following meta-policy.

```
+!countVotes(Pattern, Scheme):
.count(agree(Pattern, Criterion, Adaptation), V) & value(comPopulation, P)
& V >= (P/2) & Scheme == "majority"
<− !AdaptPolicy(Pattern, Criterion, Adaptation).
```

Listing 9.15 – Collective Decision Taking

Using the above meta-policy, all agents of the community collect and count the votes that agree with the proposed adaptation. Then the specified voting scheme is used to compute a decision based on the votes counts. In this example, the agents make use of a majority scheme which evaluates whether the number of agents agreeing with the adaptation exceeds half of the community population. If so, each agent will adapt the collective policy in consequence. To that aim, the following meta-policy is used.

```
 !AdaptPolicy(Pattern, Criterion, Adaptation):
Adaptation =="relaxCriterion" & policy(Pattern, P)[Pattern]
```

```
<- policy.relaxCriterion(P, Criterion).
```

Listing 9.16 – Collective Decision Taking

Thus we showed in this section that using four relatively simple meta-policies, how we make agents coordinate about collective policies adaptation. Of course, the above policy should be specified for each type of adaptation. The testing condition determines which kind of adaptation should be triggered (i.e. $Adaptation ==''\, relaxCriterion''$).

9.3 Conclusion

In this chapter, we illustrated the applicability of ASC-TMS on an *open innovation* virtual community scenario. Nevertheless, we believe that ASC-TMS is general enough to be applicable to many other different scenarios involving social structures.

This chapter aimed at illustrated how the contribution proposed in this thesis can be used to meet the objectives of this thesis. We showed how *ASC-TMS* can be used by the participant in innovation process to select the partners which whom that want to collaborate. We showed also how these members can rely on it to build the collective policies of the communities they create.

We demonstrated also how ASC-TMS prevent virtual community members from making decisions that goes against the collective policies in their communities, and how in case this policy is inadequate they can adapt it.

Also, this chapter aimed at illustrating the use of the trust negotiation strategy presented in Chapter 7. The novel aspect of this strategy that we believe to be more *adaptive* as it considers the possibility to adapt the policy if such adaptation is possible (e.g. do not violate the collective police).

In the next section, we further motivate the benefits of ASC-TMS and evaluate experimentally the performance of virtual communities using ASC-TMS compared to those that are not using it.

9.4 French Summary

Dans les chapitres précédents, nous avons présenté notre modèle de communautés virtuelles à base d'agents (cf. Chapitre 5), notre langage de spécification des politiques – individuelles et collectives – (cf. Chapitre 6) ainsi que notre mécanisme d'adaptation des politiques (cf. Chapitre 7). Enfin, dans le chapitre précédent, nous avons décrit comment ces différentes contributions ont été implémentées. Dans ce chapitre, nous allons mettre l'accent sur l'application de notre approche aux communautés virtuelles de la vie de tous les jours. Pour cela, nous présenterons comment nous avons utilisé notre système de gestion de la confiance pour gérer la confiance dans des communautés virtuelles d'innovation ouverte. Ce scénario a été choisis suite à notre collaboration avec le projet WINPIC financé par St-Étienne Métropole. [2]

9.5 Innovation Ouverte

L'innovation ouverte est une nouvelle manière de générer des idées révolutionnaires rapidement et à moindres coûts. Cette approche est de nos jours adoptée massivement par les entreprises et les organisations pour stimuler l'innovation dans leur département R&D en mobilisant les compétences externes à leurs locaux. Historiquement, le concept d'innovation ouverte à été rendu célèbre par Henry Chesborough dans son livré "Open Innovation" [Chesbrough, 2006], devenu une référence depuis.

Le paradigme de l'innovation ouverte considère la recherche et le développement comme un système ouvert [Ahonen and Lietsala, 2007, Vega, 2012]. Cette approche suggère que les bonnes idées peuvent émerger à la fois du sein de l'entreprise mais également de l'extérieure de celles-ci. Par exemple, le principe de l'open source est souvent donné comme exemple d'une application réussie des principes de l'innovation ouverte. La seule différence majeure entre l'open source et l'innovation ouverte réside dans le fait que la seconde intègre le modèle économique dans son mode de développement alors que la seconde a tendance à le rejeter [Cooke, 2009, Huizingh, 2011]. Au vu de cette caractéristique, et du point de vue de la gestion de la confiance qui nous concerne, l'innovation ouverte est de notre point de vu plus sensible au mode de gestion de la confiance que ne l'est l'open source, bien que ce dernier n'y soit pas insensible.

L'activité d'innovation ouverte implique souvent deux types d'acteurs et un intermédiaire. Les *Seekers* sont les individus à la recherche d'idée pour résoudre des problèmes particuliers, tandis que les *solvers* sont les têtes pensantes dont l'objectif est de résoudre ces problèmes. Aussi, le rôle de l'intermédiaire consiste à fournir des moyens techniques et technologiques afin de permettre à ces deux types d'acteurs de réaliser leurs objectifs. Ainsi, au cours des dernières années, plusieurs plateformes commerciales (e.g. Hypios, Nine Sigma or Innocentive) ou gratuites (e.g. W3C Community Group) ont vu le jour dans la perspective de jouer le rôle d'intermédiaire à la fois technique, mais également économique et sur tout un intermédiaire de

[2]http://iscod.emse.fr/winpic/

confiance. Intermédiaire technique car ils fournissent des outils de collaboration et de partage de ressources qui permettent au *solvers* et aux *seekers* de maintenir leur activités. Intermédiaires économiques car ils représentent une vitrine pour les problèmes mais également pour les chercheurs de solutions. Enfin, intermédiaires de confiance car ils font office de tiers de confiance entre *solvers* et *seekers*. Cependant, aucune des plateformes que nous avions ne permet de gérer explicitement la relation de confiance entre les différents *solvers*.

Dans la section suivante, nous allons présenter l'exemple de communauté virtuelle d'innovation que nous allons utiliser tout au long de ce chapitre pour illustrer l'application de ASC-TMS. Dans ce scénario, nous allons nous intéresser essentiellement à la gestion des relations de confiance entre *solvers* qui fait fait défaut aux plateformes existantes comme précisé plus haut. En effet, dans une étude récente [Cooke, 2009], la plupart des *solvers* avouent que le manque de confiance fut le raison principalement de leur désintéressement des plateformes d'innovation ouverte. Dans la suite de ce chapitre, nous expliquerons comment nos contributions constituent une solution à ce problème.

9.6 Communauté d'Innovation Ouverte

Dans cet exemple, nous considérons une communauté d'innovation ouverte composée de 13 membres. Pour des raisons de simplicité, nous assumons que seule un membre joue le rôle de *seeker* (i.e., Michael). Dans ce scénario, le *seeker* cherche à trouver une solution à un problème donné contre une récompense. Cette situation est illustré dans la figure 9.7

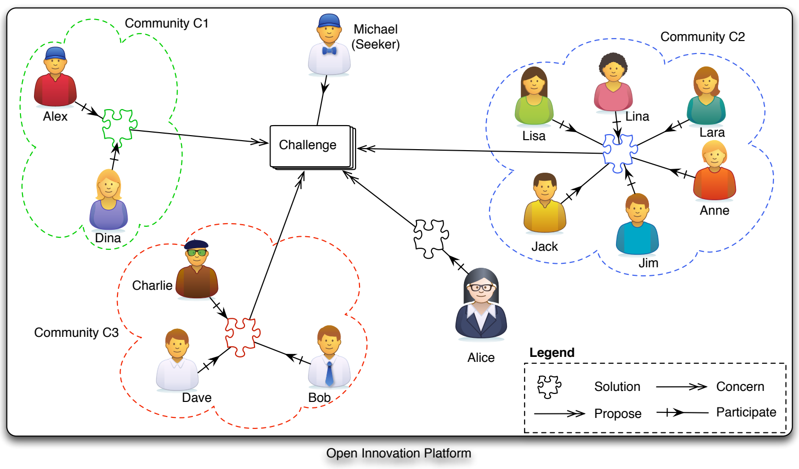


Figure 9.7 – Une instanciation d'une communauté d'innovation ouverte

Le scénario commence lorsque le *seeker* introduit dans le système un nouveau challenge – description du problème auquel la communauté va essayer d'apporter une solution – .

9.6.1 Utilisation du ASC-TMS pour la création des communautés

En fonction de leurs aptitudes, les agents ont tendance à se grouper pour créer des communautés au sein desquels ils vont collaborer pour relever le challenge. À ce stade déjà apparaît le premier besoin de la gestion de la confiance car les participants ont tendance à choisir leur collaborateur en fonction du degré de confiance qu'ils leur accordent.

9.6.1.1 Évaluation de politiques

Comme évoqué précédemment, avant de se mettre d'accord pour créer une communauté, les participants doivent d'abord s'assurer que leurs collaborateurs satisfont leurs politiques de confiance. Ainsi, le processus d'évaluation des politiques de chaque membre est un préalable à la formation de toute communauté. Dans cette section, nous décrivons succinctement comment le ASC-TMS va décharger les membres (humains) des communautés virtuelles ainsi que leurs assistants (agent) de cette tâche complexe et fastidieuse. Pour cela, nous présentons la politique utilisé par *Dave* pour prendre des décisions à propos de collaboration.

```
policy([
lbc(mail.identity,“>=”, marginal, 5, m)
lbc(skilfulness, “>=”, fair, 2,m),
lbc(reputation, “>=”, 70 ,2,o),
lbc(recommendation, “>=”, 3,1,o)
])[issuer(dave), pattern(collaborate, _)].
```

Listing 9.17 – Politique de collaboration utilisée par Dave

Maintenant, on considère la situation dans laquelle *Dave* souhaite évaluer le degré de confiance qu'il peut accorder à *Alice* dans la perspective de former une communauté avec elle. Pour cela, *Dave* doit collecter les informations nécessaire à l'évaluation de sa politique. Là aussi c'est le ASC-TMS qui s'en charge et on suppose que le listing ci-dessous présente l'ensemble des informations (credentials et déclarations) que *Dave* a récollté sur *Alice* (i.e. son profil).

```
credential(mail.identity, alice, unknown)[issuer(authority1), type(“PGP”)].
credential(certificate.skilfulness,Alice, good)[issuer(authority2), type(“X.509”)].
declaration(reputation,Alice, 50).
declaration(recommendation,Alice,0).
```

Listing 9.18 – Le profile d'Alice

À partir de ces informations, et en utilisant la fonction d'évaluation présentée dans le Chapitre 7, le ASC-TMS va calculer le degré de confiance que *Dave* accorde au profile d'*Alice*. Ce processus d'évaluation est décrit dans la figure 9.8

Lors de cette évaluation, on voit qu'Alice n'a pas été en mesure de satisfaire l'ensemble des conditions spécifié par *Dave* dans sa politique. Le degré de confiance accordé à *Alice* est de zéro car le certificat d'identité que celle-ci possède ne satisfait pas la condition correspondante dans la politique de *Dave*. Et comme cette condition est obligatoire dans sa politique, le résultat de

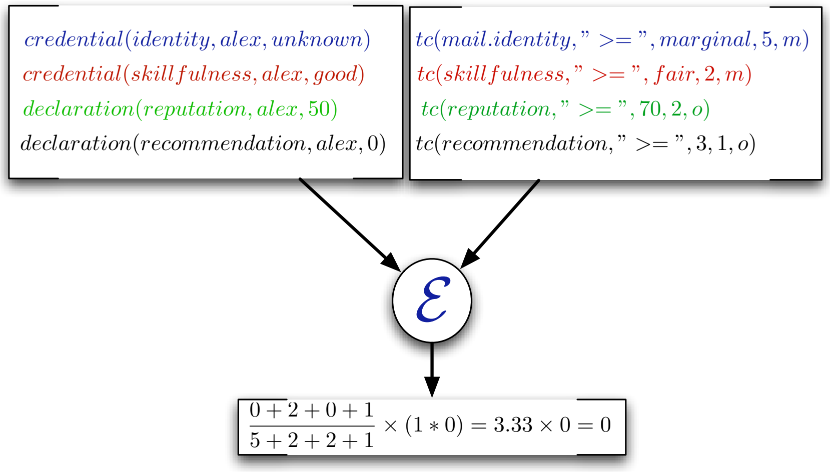

$credential(identity, alex, unknown)$
$credential(skillfulness, alex, good)$
$declaration(reputation, alex, 50)$
$declaration(recommendation, alex, 0)$
$tc(mail.identity, ">=", marginal, 5, m)$
$tc(skillfulness, ">=", fair, 2, m)$
$tc(reputation, ">=", 70, 2, o)$
$tc(recommendation, ">=", 3, 1, o)$
$\mathcal{E}$
$\frac{0+2+0+1}{5+2+2+1} \times (1*0) = 3.33 \times 0 = 0$

Figure 9.8 – Evaluation du degré de confiance accordé à Alice

l'évaluation est zéro. De manière analogue, les autres participants (i.e. Alice, Bob, Charlie et Dave) ont put chacun calculer le degré de confiance qu'il s'accordent mutuellement. Le résultat de ces différentes évaluations est illustré dans le tableau 9.2.

Solvers	Alice	Bob	Charlie	Dave
Alice	-	0.6	0.8	0.9
Bob	0.4	-	0.7	0.7
Charlie	0.5	0.5	-	0.6
Dave	0	0.7	0.6	-

Table 9.2 – Résumé des évaluations respectives de Alice, Bob, Charlie et Dave

En se basant sur ce tableau, et grâce au travail du ASC-TMS, les participants ont décidé de construire une communauté de confiance dans laquelle seuls les participant ayant obtenu le plus haut degré de confiance sont admis. Ainsi, la communauté formée est composé de *Bob, Charlie* et *Dave*. Mais avant que cette communauté ne puisse réellement prendre vie, les membres de cette communauté doivent d'abord se mettre d'accord sur une base commune qui va leur servir à prendre des décisions de confiance. Cette base représente les politiques de confiance et là également apparaît l'intérêt du ASC-TMS car il va permettre à ces membres de pouvoir construire de manière décentralisée leurs politiques collectives. Ensuite, une fois que ces politiques collectives construites, le ASC-TMS va leur permettre de les appliquer conjointement avec leurs politiques individuelles et les faire évoluer quand cela est nécessaire (cf. le manuscrit).

Chapter 10

Evaluation of ASC-TMS

In Chapter 8 we presented the implementation of the ASC-TMS and in Chapter 9 we illustrated its application to an open innovation community. In this chapter, we will focus our attention upon the evaluation of the ASC-TMS. Our implementation is based on the Recursive Porous Agent Simulation Toolkit (Repast) .

The objective of this chapter is to have a systemic evaluation of the benefit of the mechanisms proposed in this thesis. We are particularly interested in evaluating the benefits of using the mechanisms drawn from *social influence theory* (i.e. *combination, integration* and *evolution*) in terms of trust management within large systems. To that aim, we *emphasise* the impact of these mechanisms on virtual communities in terms of *stability*.

To that aim, we first present in Section 10.2 the simulation model we implemented in our simulator. Then, we present in Section 10.3 the scenario we used in our evaluation. This scenario is inspired from the running example presented in the previous chapter. In 10.4, we describe the settings, parameters and metrics we used in our evaluation. In Section 10.12, we introduce our *working hypothesis* then we present the results we obtained and discuss their relevance with respect to our *hypothesis*.

Before, we present in the next section the Repast Symphony platform on which the system has been implemented.

10.1 Repast Platform

Repast [Collier, 2003] is a widely used free and open-source agent-based modelling and simulation platform. RePast was developed in 2003 by the Social Science Research Computing Lab of the University of Chicago specifically for creating agent based simulations in social sciences. The framework makes use of a fully concurrent discreet event scheduling. It provides rich logging and graphing built-in tools while allowing the direct connection with external analysis tools such as MatLab, R, Spreadsheet (e.g. Excel) and Weka. Repast has multiple implementations in several languages (e.g. Java, Logo, .NET languages, Lisp, Prolog and Python). In our implementation, we used the *Repast-J*, which is the Java based implementation of Repast.

Our principal motivation in using Repast lies in the fact that this platform is currently considered as the most suitable simulation framework social scientific agent based computer simulation [Tobias and Hofmann, 2004]. This makes Repast the most appropriate candidate for experimenting the benefits of using the Social Influence Theory in Trust Management.

10.2 Simulation Model

The model we implemented in Chapter 8 based on the specification made in Chapters 6 and 7 is quite complex to be implemented and properly evaluated as it is. Several aspects used in the model are technically hard to implement into Repast (e.g. the BDI decision engine). Therefore, we implemented a simplified version of the ASC-TMS which covers the contributions of our model in a minimal way. Also, we have made the choice to merge the *assistant* and the *trust management* agent into one single agent representing the *solvers*.

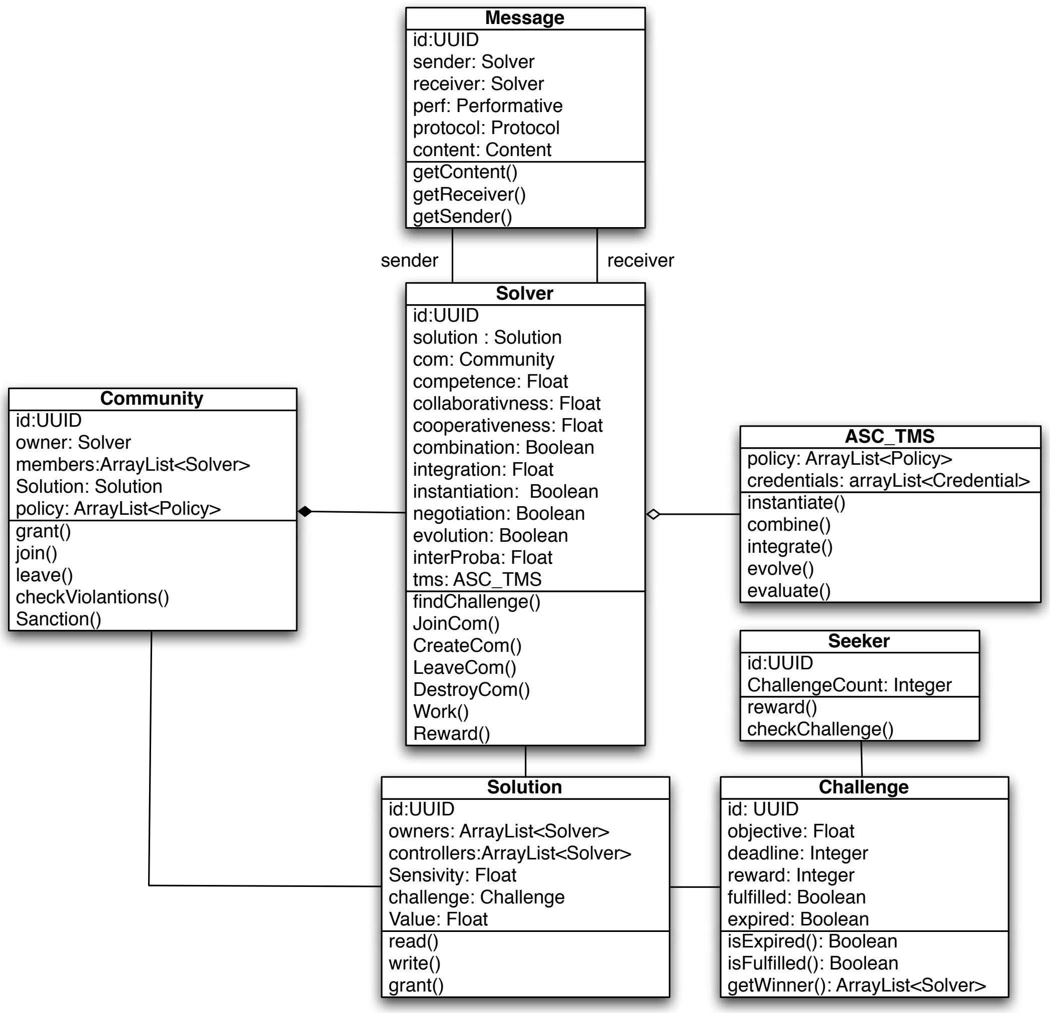


Figure 10.1 – Abstract UML class diagram of the simulated model

Figure 10.2 illustrates the principal Java class our simulation relies on. The simulated model is composed of two classes, each representing a type of agent (*Solver* and *Seeker*), a class representing communities (*Community*), a class representing the solutions produced by the *Solvers*, a class representing the challenges that are introduced by *Seekers* and class encapsulating the functionalities provided by the ASC-TMS. Finally, we provide an *message* class by means of which *solvers* interacts with each others. The remainder of this section is dedicated to the description of these classes (except the message class).

10.2.1 Solutions

Solutions correspond to the resources of our model ($\forall r_i \in \mathcal{R}$). In our simulation, we consider only one type of resource which contains a solution to a specific problem. For instance, *solutions* can represent notes on which specify their ideas to tackle the proposed problem. Each *solution* is characterised by an identifier[1], a list of owners, its sensitivity and its value. The description of these attributes and the values they can take are reported in Table 10.1.

	Attribute	Values	Description
Solution	`id`	`UUID`[2]	Corresponds to the id (i.e. $r_i.\varepsilon$) of the solution in the system.
	`owners`	Solver	Corresponds to the owner of the solution ($i.e.r_i.\varphi$). This value can be the identifier of an agent if the solution is an individual individual or an identifier of a community if it is a collective solution.
	`value`	`[0,10`^{4}`]`	Corresponds to the value of the solution. This value materialises the extent to which the solution fulfils the objective of the challenge (cf. Section 10.2.2).
	`sensitivity`	`[0,100]`	Corresponds to the *sensitivity* (i.e. $r_i.\varsigma$) of the resources as defined in the model.

Table 10.1 – The variables of a challenge

Solvers can manipulate *Solutions* via four methods, namely `read()`, `update()`, `grant()` and `notify()`. These methods corresponds to the operations of the resources as defined in the model (i.e. $r_i.\omega$). The `grant()` method is used by the *solver* to grant access a solution to another *solver*. Access to solutions can be made via `read()` or `write` methods. The first method is used by the solver to retrieve the content (i.e. value) of the solution, while the second one allows the *solver* to modify it. The modification of a solution can be *positive* or *negative*. To that aim, solutions are endowed with the `notify()` that is triggered by the solution after each access. The method informs the resource owner(s) about the nature of the access performed

[1] A Java class that represents an immutable universally unique identifier (UUID). A UUID represents a 128-bit value.

on it. This information constitute the feedback of the trust decision as defined in Section 7.3 (cf. Chapter 7). Based on this feedback, an agent can decide to sanction or reward the agent that he trusted.

10.2.2 The Challenge

Challenges represent the problems to which *solvers* are trying to provide solutions. *Challenges* are proposed by *solvers* are goals they will try to achieve (i.e. $g_i \in a_j.G$). They are introduced in the system by *Seekers* by instantiating the *Challenge* class. An instance of a challenge defines the *objective*, *deadline* and *reward* of the problem. The description of these attributes and their corresponding values are reported in Table 10.2.

	Attribute	Values	Description
Challenge	`id`	`UUID`	Corresponds to the unique identifier of the solution in the system.
	`Objective`	`[0,` 10^4`]`	Corresponds to the value that a solution should reach to fulfil the challenge
	`Reward`	`[0,`10^3`]`	Corresponds to the utility the agent(s) that proposed the solution will gain in terms of utility. If the challenge is fulfilled by the members of a community, they will share the utility.
	`Deadline`	`[0,`10^3`]`	Corresponds to the number of simulation steps that the challenge will remain active before its expiration.

Table 10.2 – The variables of a challenge

The method, `isExpired()` and `isFulfilled()` prevents agent from working on *challenges* that are expired or have been already met. When a challenge is expired, it is retrieved from the system and a new *challenge* is introduced by the *seeker*. Similarly, if a challenge is fulfilled, the *Seekers* rewards the *solvers* participating in the challenge and introduced a new one. So at any moment of the simulation, only one challenge is made active. This choice has been made to maximise the response time and to prevent inter-challenges conflicts (e.g. an agent abandoning the current challenge for an easier challenge or one with a better reward). Experimenting situations in which challenges are proposed by many agents is out of the scope of our thesis. This issue was left as a perspective and will be discussed in Section 11.2.

10.2.3 The Community

The *community* class encapsulates attributes and methods that are necessary to emulate the concept of community ($\forall c_i \in \mathcal{C}$) as defined in Section 5.5. A community is defined by its unique identifier, a set of members, a set of policies and a solution. We report in Table 10.3 the description of these attributes and their corresponding values.

	Attribute	Values	Description
Community	`id`	`UUID`	Corresponds to the unique identifier (i.e. $c_i.\varepsilon$) of the community in the system.
	`owner`	Solver	Corresponds to Solver that created the community.
	`members`	Solver	Corresponds to the set of solver that are members of the community.
	`policies`	Policy	Corresponds to the set of collective policies used in the community.
	`solution`	Solution	Corresponds to collective solution developed by the members of the community.

Table 10.3 – The variables of a challenge

Each community provides a set of methods by means of which a Solver can join (i.e. `join()`), leave (i.e. `leave()`) and destroy (i.e.`destroy()` a community. The community can be destroyed by its owner, or by a solver which is he is the unique member of the community. Here we simplified our model by choosing to implement only two roles, namely owner and member. Also, norms that are used to evaluate deviation have been implemented in the `control()` method. This method is automatically triggered after each access to determine if the access was positive or negative. In this last situation, the *checkViolation()* method is invoked to evaluate whether the collective policy has been violated while granting access to the resource. If so, the solver that made the decision is sanctioned and the violation is reported to the other members of the community.

10.2.3.1 The Solvers

Solvers are the agents responsible of providing solutions to challenges (i.e. problems) proposed by *seekers*. The solver class is defined by a set of attributes and methods that determines the solver's behaviour in the system. These attributes are grouped into two categories: *innovation* and *trust management* attributes.

Innovation attributes are application specific. They determine how the solver behaves with respect to the innovation process. These attributes are presented in Table 10.4.

Based on their *cooperativeness* value, solvers are split into four categories:

- **Altruist.** Are solvers which cooperativeness value is 1. These solvers will devote all its competence to improve the solution he is accessing. He will also benefit from the resource it is accessing to. For example, if a solver accesses a solution which value exceeds the value of its solution, he will benefit from this solution to make his solution as valuable as the solution he accessed to.

- **Cooperative**. Are solvers which cooperativeness value belongs to the interval $]0,1[$. This value represent the proportion of competence used by the solver to improve solutions of

	Attribute	Values	Description
Solver - **(Innovation)**	`id`	`UUID`	Corresponds to the unique identifier (i.e. $c_i.\varepsilon$) of the community in the system.
	`competence`	`[-1,1]`	Corresponds to the solver's competence degree. It determines to which extent the solver is able to improve a solution.
	`collaborativeness`	`[-1,1]`	Expresses to which extent a solver is willing to work collaboratively. It represents the probability to see the solver creating/joining a community.
	`cooperativeness`	`[-1,1]`	Determines how the solver behaves when accessing resources that he does not own.
	`Solution`	`[-1,1]`	Corresponds to the *solution* owed by the *solver*.

Table 10.4 – The variables of a challenge

other solvers. For instance, if a solver cooperativeness value is 0.2, that means that this solver will use 20% of its competence capabilities to improve the solution to which he has access. It will also benefit from the resource.

- **Selfish.** Are solvers which cooperativeness value is 0. These solvers will never improve solutions of others. Instead, they benefit from the value of these solutions to improve their own solution.

- **Malicious.** Are solvers which cooperativeness value is below belongs to the interval $]0, -1]$. These solvers will first benefit from the solution of others, then they vandalise it. Vandalising a solution means that the solver will make this solution loose part of its value. For instance, a solution that has been manipulated by a solver which cooperativeness value is -0.5 will loose half of its value.

Trust management attributes are used by the simulation to know which feature of the ASC-TMS model are used by the agents. These attributes are summarised in Table 10.5.

The above attributes of the class *Solver* are used to activate and deactivate specific features of our model. The objective is to be able to evaluate more accurately different aspects of the *adaptation* mechanisms proposed in Chapter 7. For instance, *instantiation* attribute is set to `false` or `true` depending on whether the solver should be sensitive to the changes in its environment. Likewise, *combination* and *evolution* are attributes expresses the propensity of the *solver* to use the *feature*. Moreover, each *solver* is endowed with a set of methods that allows him to find a challenge to address (i.e. `findChallenge()`), create a community (i.e. `createCom()`), join a community (i.e. `joinCom()`), leave a community (i.e. `leaveCom()`) and destroy a community (i.e. `destroyCom()`).

A *solver* can also grant access to the solution he owns or to a community he belongs to. A *solver* can *reward* (i.e. `reward()`) or *sanction* (i.e. `sanction()`) an other *solver* based on

	Attribute	Values	Description
Solver - (**Trust Management**)	`id`	`UUID`	Corresponds to the solver's unique identifier in the system.
	`combination`	`Boolean`	Corresponds to the solver's ability to combine several policies.
	`integration`	`[0,1]`	Corresponds to the solver's propensity (i.e. probability) to comply with the collective policy of its community. These features expresses the ability of the agent to *integrate* the collective policy with its individual one.
	`evolution`	`Boolean`	Corresponds to the solver's propensity to trigger the adaptation of the collective policy of its community.

Table 10.5 – The variables of a challenge

this latter's behaviour. They also *solvers* rate each others (i.e. `rate()`) with respect to the *cooperativeness* attitude of the *solver* that manipulated their *solution.* This rating determines if this *solver* exhibited an *altruist* (+2), a *cooperative* (+1), a *selfish* (-1) or a *malicious* (-2) behaviour. Based on these ratings, an aggregated reputation value is shared among the members of the system.

10.2.3.2 The Seeker

Seekers are the agents responsible of providing challenges in the simulation The attributes used to instantiate seekers are defined in Table 10.6

	Attribute	Values	Description
Seeker	`id`	`UUID`	Corresponds to the seeker's unique identifier (i.e. $a_i.\varepsilon$) in the system (i.e. $\mathcal{S}$).
	`chCount`	Integer	Corresponds to the number of challenges each seeker will introduce in the simulation.

Table 10.6 – The variables of a challenge

Each seeker is provided with a method that evaluates whether a challenge is met (i.e. `checkChallenge()`), and a method used to reward the winners of the challenge (i.e. `reward()`).

10.3 Simulation Scenario

In the previous section, we defined the attributes and methods of the classes used in our simulation. In this section, we describe the innovation process used in our simulations. This scenario is inspired from the running example presented in Chapter 9.

The simulation is initialised by creating one *seekers* and a given number of *solvers*. The objective of the *seeker* is to find a solution to a given number of problems (i.e. `chCount`). Solvers aspire to optimize their individual utility. This utility can be increased by winning a challenge and obtaining the reward, or by being a member of a community which won a challenge.

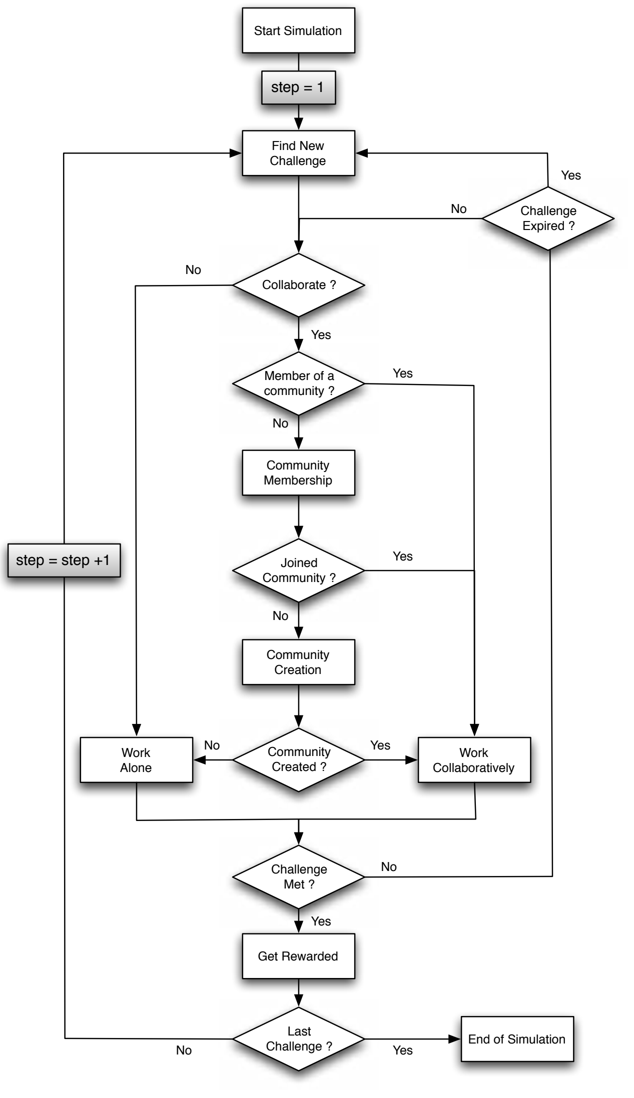


Figure 10.2 – Flowchart describing the abstract innovation process

As illustrated in Figure 10.2, the simulation starts when a new challenge is made available by a *seeker*. Once a new challenge found, the *solver* decides whether to *work collaboratively* or to *work alone*. This decision is defined by its propensity to collaborate that we express as a probability drawn from a uniform distribution between 0 and 1 (i.e. *collaborativeness*). If the *solver* decides to collaborate, he first checks whether he is not already member of a community. If not, the *solver* will try to join a community otherwise, he will try to create a new one. At this stage, if the member succeed to join or create a community, he will *work collaboratively*, otherwise, he will *work alone*. A round of this simulation model is completed when every *solver* has been given exactly one opportunity to work. Once the *solver* finishes working, he evaluates whether the value of solution he participated reached the *objective* of the challenge. If it is the case, the state of the challenge is set to *met* and the agent (or community of agent) is rewarded. If not, the *seeker* checks whether the challenge did not expire, otherwise he introduces a new one. If the *seeker* reaches the number of challenge he wanted to solve, the simulation stops.

Up to now, we describe the process followed by the *solver* during the simulation in abstract terms. For instance, we did not detail what is really happening in the working (alone or collaboratively) step. To that aim, we present in Figure 10.3 how the *cooperativeness* attribute influences the behaviour of solvers.

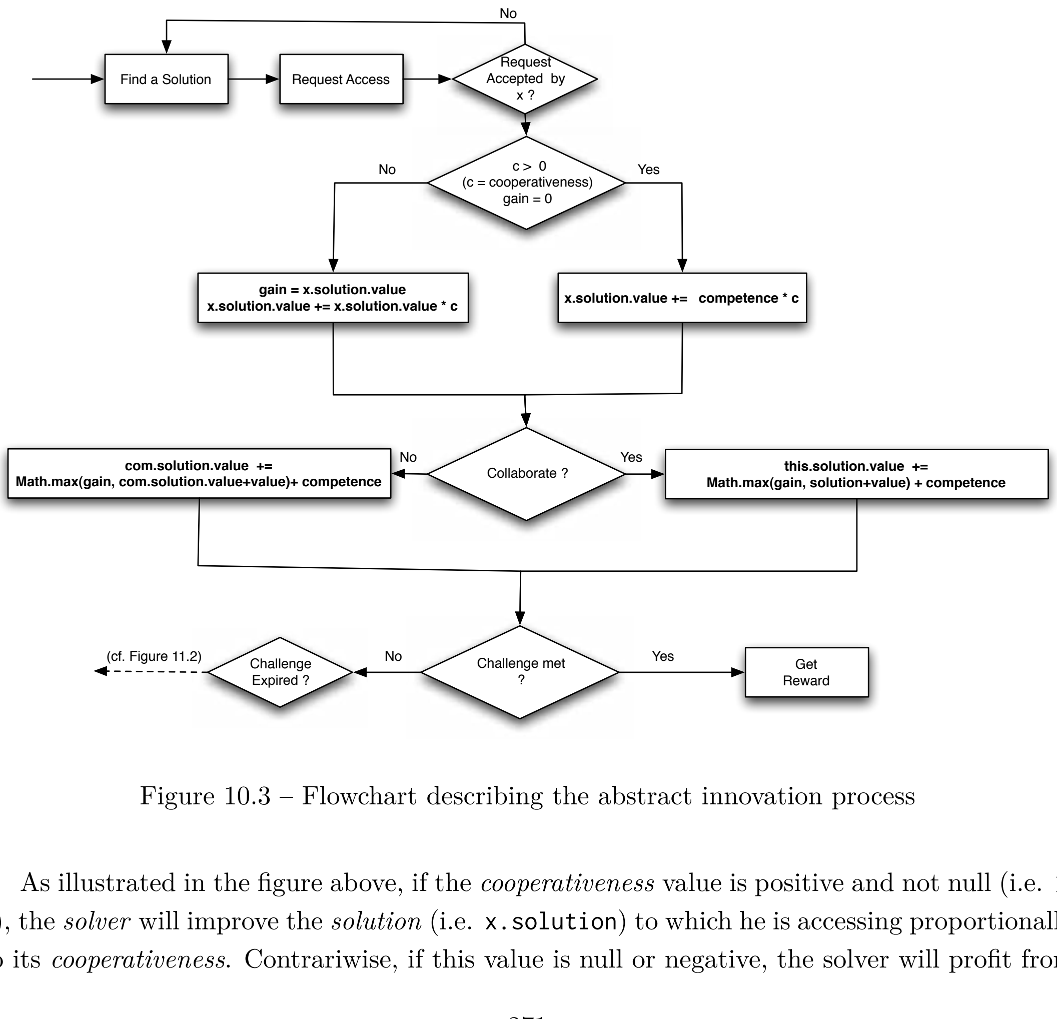

No
Find a Solution
Request Access
Request Accepted by x ?
No
c > 0 (c = cooperativeness) gain = 0
Yes
gain = x.solution.value x.solution.value += x.solution.value * c
x.solution.value += competence * c
com.solution.value += Math.max(gain, com.solution.value+value)+ competence
No
Collaborate ?
Yes
this.solution.value += Math.max(gain, solution+value) + competence
(cf. Figure 11.2)
Challenge Expired ?
No
Challenge met ?
Yes
Get Reward

Figure 10.3 – Flowchart describing the abstract innovation process

As illustrated in the figure above, if the *cooperativeness* value is positive and not null (i.e. > 0), the *solver* will improve the *solution* (i.e. `x.solution`) to which he is accessing proportionally to its *cooperativeness*. Contrariwise, if this value is null or negative, the solver will profit from the solution and vandalise it proportionally to its *cooperativeness* value. *Rewards* are determined in terms of utility. At the beginning of the simulation, each *solver* is provided an initial utility score of 100 units. Then this score is affected (positively or negatively) by the decisions made by this *solver*. The way the decisions of the *solver* affect its utility are summarised in Table 10.7.

Name	Utility	Description
`RESOURCE_IR`	`+ Value gain`	Reward gained by the *solver* which resource has been improved by another *solver*.
`RESOURCE_CR`	`+ (Value gain / size)`	Reward gained by the *solver* which the resource of its community has been improved by another *solver*.
`RESOURCE_G`	`+ Value gain`	Reward gained by the *solver* following the access to a resource that improved its own/collective resource.
`RESOURCE_IS`	`- Value Loss`	Utility loss in consequence of granting access to an agent that vandalised the individual solution.
`RESOURCE_CS`	`- (Value Loss / population)`	Utility loss in consequence of granting access to an agent that vandalised the solution of the community.
`CP_VIOLATION`	`- Value Loss`	Sanction imposed to *solvers* that violates collective policies. This sanction corresponds to the loss of value caused by the decision made by the *solver*.
`CH_IWIN`	`+ Reward`	Reward gained by a *solver* that won a challenge.
`CH_CWIN`	`+ (Reward / size)`	Reward gained by a *solver* that is a member of a community that won a challenge.

Table 10.7 – The utility payoffs during the simulation

A *solver* can gain utility units by winning a challenge or participating in a community which won a challenge. A *solver* can also gain utility when accessing valuable resources or by letting *competent* and *cooperative solvers* manipulate its resources, or the resource of its community.

A *solver* can also loose *utility* units. The *utility* of a *solver* decreases in three situations: (1) when his *solution* has been vandalised by a malicious *solver*, (2) when the solution of its community has been vandalised, and (3) when he violates the collective policy of its community.

10.4 Simulation Settings

In this section, we present the simulation settings we used to evaluate the ASC-TMS.

10.4.1 Simulation Parameters

In this section, we present the parameters used to setup our simulation. The parameters of the the experimentations fall into two categories; parameters that relates to the challenges and those that relates to the solvers. These parameters represent are default settings if not stated otherwise:

- **Challenges Parameters**

  1. `Challenges Count` - The number of challenges introduced.
  2. `Challenges Objectives` - What is the objective (i.e. resource value) that the agents much reach to win the challenge. The objective of each challenge was set to 10^4. This value was fixed experimentally to have sufficiently long challenges.
  3. `Challenges Rewards` - What is the reward that agents will gain in winning the challenge. The reward of each challenge was set to 1000.
  4. `Challenges Deadline` - What is the deadline of the challenge. The deadline of each challenge was set to 1000.

- **Solvers Parameters:**

  1. `Policies` - Each solver is endowed with two policies which *type*, *value* and *weight* are randomly generated. The first policy is used for making decisions about access to resources, while the second is used for to make decision about whom to collaborate with (i.e. community membership). Policies are defined using four types of *trust factors*; two *proofs*, namely *identity*, *competence*, and two *indicators*, namely *reputation* and *recommendation*.
  2. `Credentials`- Each solver is provided with 10 credentials, five about its identity and five about its competence. Consequently, more that one credential can be used to vouch for the same property (i.e. identity or competence). However, these credentials differ in terms of sensitivity.
  3. `Competence` - The *competence* of *solvers* is drawn from a *normal* distribution between 0 and 1.
  4. `Collaborativeness` - The *collaborativeness* of *solvers* is drawn from a uniform distribution between 0 and 1.
  5. `Cooperativeness`- The *cooperativeness* of solvers was set manually in order to have a population that is composed of 10% altruist, 40% cooperative, 40% selfish and 10% malicious solvers. This population was selected experimentally.
  6. `Interaction`: This parameter determines the probability of a solver to interact with other solves (i.e. request access to solution). This parameter (i.e. `interProba`) has been fixed to 0.8.

10.4.2 Simulation Metrics

Metrics represent the statistics that we rely on in the evaluation of our approach. These metrics are presented in what follows:

- `sovCount`: the number of solvers in the system.
- `comsCount`: the number of communities
- `comsSize`: the population of each community

10.5 Evaluations

This chapter aims at evaluating the benefits in using the mechanisms drawn from *social influence theory* in terms of trust management. The empirical evaluation of our system has been performed based on the scenario presented in Section 10.3. We are particularly interested in assessing whether the following working hypothesis are supported by ASC-TMS.

- *Hypothesis 1:* Communities that are formed by *combining solvers* are more stable than communities formed by *non combining* agents.
- *Hypothesis 2:* Communities formed of compliant agent are more stable that communities formed of non compliant agents.
- *Hypothesis 3:* Communities stability is function of the proportion of *compliant* members composing it.
- *Hypothesis 4:* The agents ability to make evolve their collective policies affects positively the stability of these communities.

Communities stability is thus an important evaluation indicator in our experiments. However, this concept is very confusing as it has been used differently in several researches. For instance, in [Merida-Campos and Willmott, 2007], the authors assimilated communities (or coalitions) stability to a stable marriage problem. They considered a community to be stable when any member do not prefer to be in another coalition rather staying in his current community. Such approach could not be used in our experiments as our agents (solvers) do not have any preferences with regard communities. Therefore, in our experiments, we will use a mix between the number of communities, the mean size of each community and the *social welfare* metrics to evaluate community stability.

10.5.1 Results

In this section, we discuss the results we we obtained in the evaluation of the *hypothesis* formulated in the previous section. In this evaluation, we used 1000 *solvers* and 1 *seeker* which was initialised with 10 challenges. Each challenge has a deadline of 1000, thus the average duration of a simulation is around 10000 steps. The objective has been fixed with respect to the standard settings presented in Section 10.4.1 (i.e. 10^4). Also, communities do not necessary explode after the end of each challenge as as long as there is a challenge to meet, members prefers to remain in a community rather then leaving it.

In the evaluation of the above *hypothesis*, we employ five experimental populations. A summary of the parameters used for each population is provided in Table 10.8.

population	Combination	Integration	Evolution
population 1	`false`	0	`false`
population 2	`true`	0	`false`
population 3	`true`	0.5	`false`
population 4	`true`	0.8	`false`
population 5	`true`	1	`false`
population 6	`true`	0.8	`true`
population 7	`true`	1	`true`

Table 10.8 – Populations used in the simulation

- Population 1: *Solvers* are are endowed with an *ASC-TMS* in which *combination* and *integration* features were deactivated. Thus theses *solver* are neither able to build collective policies, nor to exhibit *compliant* behaviour.
- Population 2: *Solvers* are able to build collective policies, but they are not able to integrate these collective. Thus the decision they make only considers their individual policies.
- Population 3: *Solvers* are endowed with *combination* and *integration* mechanisms. However, they tend to comply with a probability of 0.5. So globally, 50% of the population will comply.
- Population 4: *Solvers* are able to build collective policies, and they exhibit a compliant behaviour with a probability of 0.8. This value corresponds to the proportion of *compliant* members in real settings as evidenced in the *social influence theory* (cf. Section 4.2).
- Population 5: *Solvers* are able to build collective policies and they always *comply* with the collective policy.
- Population 6: *Solvers* are able to build and make evolve the collective policies to which they are subject. However, they comply with these policies with a probability of 0.8.
- Population 7: *Solvers* systematically build, integrate and make evolve their collective policies. This population is the perfect population with respect to the *social influence theory*.

In our scenario, we make use of random draws which make our simulation *stochastic*. As a *stochastic* simulation, our evaluation will produce different outcomes for different random number streams, which are generally driven by choosing different random seeds. For instance, some *solvers* may interact only with good *solvers* (i.e. *altruist* or *cooperative*) or bad *solvers* (i.e. *selfish* or *malicious*) *solver*. In addition, the result of each *solver* is also the consequence of the values they have been initialised with. So in order to minimise this effect on our results, and to explore the space of all possible outcomes, all graphs presented in the next sections plot the mean of 100 execution of the simulation. Then we compared the results of each population in terms of *utility*, *social welfare* and communities *stability*. The results of these evaluations are presented in the next section.

10.5.1.1 Hypothesis 1

In this section, our objective is to evaluate if virtual communities members (i.e. solvers [3].) would benefit from the ability to build collective policies. To that aim, we used populations 1 and 2 described above. *Population 1* is used for control purpose, while *population 2* is used to assess the benefit of combination. As in *population 1* there is no collective policy, agents are excluded when they make three bad decisions (i.e. trust decisions with negative consequence) and a community collapse when it loose all its members. In turn, agents from *population 2* are sanctioned when they make bad decisions (i.e. decisions that conflict with the collective policy and which consequences are negative) and after three sanctions, they are excluded.

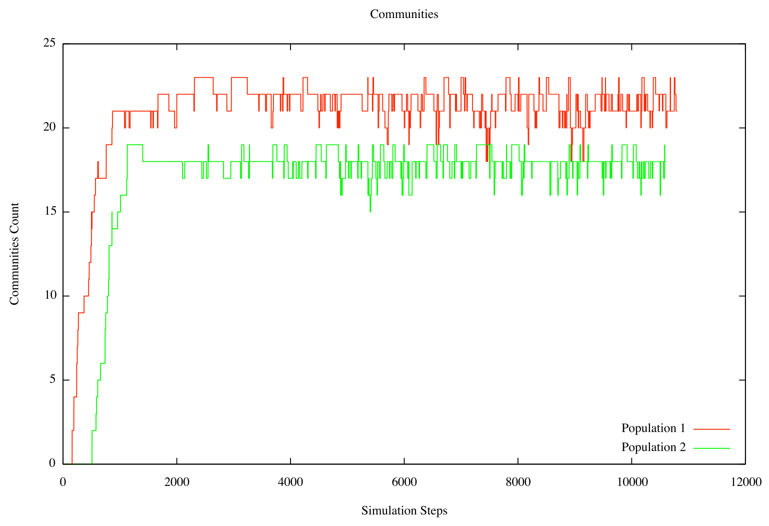


Figure 10.4 – Evolution of communities

As showed in Figure 10.4, the effect of *combination* on *population 2* was quite negative. This effect is not surprising as the objective of collective policies is to bring stability, but without integration mechanisms, this feature will only have a prescribing effect. So agents are more frequently sanctioned and communities collapse more often.

[3]In the remainder of this chapter, we will use interchangeably the terms *member*, *solver* and *agent* when referring to virtual communities members

Also, in population 1, communities are formed more rapidly compared to the communities formed in population 2. This effect can be explained by the process that agent use to form these communities (cf. Section 9.2.1). Indeed, in population 2, agent need more time to trust each others and agree on a collective polity. This process disadvantage these agents which explains the delay that take communities to form in this population. In contrast, agents from population 1 get formed into communities only based on their respective competence and the trust level they accord to each other. Thus these agents get more rapidly involved in communities.

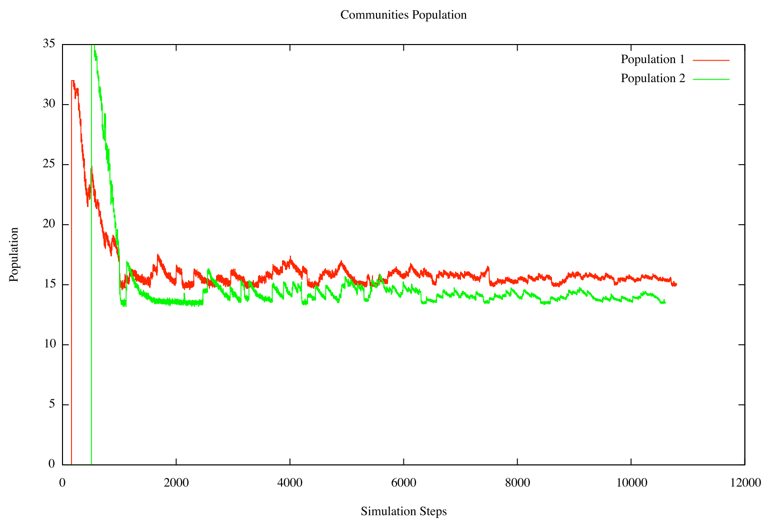


Figure 10.5 – Population evolution

The negative effect of *combination* of collective policies on virtual communities is further confirmed when analysing the average size of these communities. The average size of communities formed in *population 2* remains very low (around 12) compared to this value in population 1 (around 15). So the population turnover is higher in *population 2* than in *population 1*, but in both cases, *communities* prove to be unstable.

10.5.1.2 Hypothesis 2

The disappointing results obtained in the previous section led us to ask whether it is worth to endow *solvers* with *combination* features despite the apparent benefit advocated throughout this thesis. In this section, we try to answer this question by evaluating whether *collective policies* have any real impact on the stability of communities. For this purpose, we simulated *population 5* in the same settings that we used to simulate *population 1* and *population 2* in the previous section. The results of these evaluations are plotted in Figures 10.6 and 10.7. Recall, *population 5* is composed of *solvers* that always comply with the collective policies of their communities. So these agents never violates these policies. Thus this population constitute the best choice to evaluate the benefit of *collective policies* if there were any.

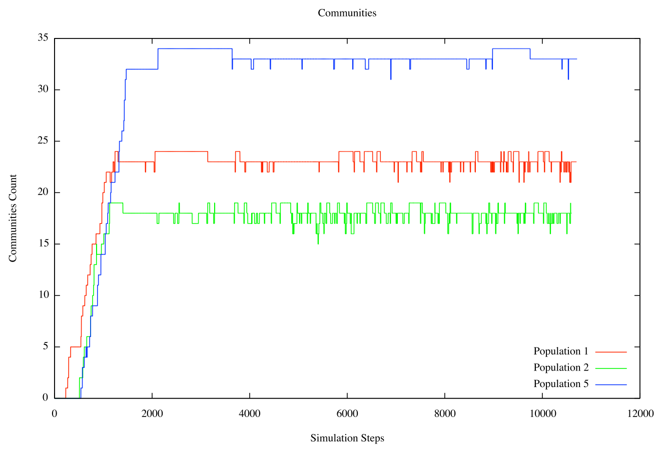


Figure 10.6 – Evolution of communities in populations 3, 4 and 5.

As expected, *populations 5* produces the best results. The communities created in this population are more stable as their members systematically comply with collective policies. The average numbers of communities ranges from 30 to 35, while this value did not exceed 25 in the populations composed of *non compliant* members. This observation applies equally to the population evolution as shown in Figure 10.7.

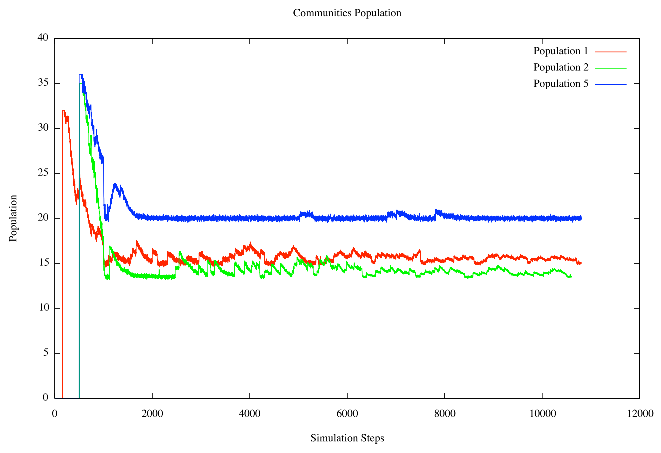


Figure 10.7 – Evolution of communities size in populations 3, 4 and 5.

In the same simulation, the size of the communities formed in *population 1* and *population 2* remains very low compared to size of communities formed in *population5*. These results also shows that proportion of *solvers* that are members of a community in *population 5* (around 560) is in average more important that those from *populations* 1 (around 380) and 2 (around 250).

Based on these indicators, we assume that *population 5* reveals to be more stable. This stability is explained by the fact that the *solvers* that compose these communities makes less violations with respect to their collective policies and thus are less frequently ejected. However, these results show that *population 5* is less idyllic than what we could expect from a population composed of agents that always comply with their collective policies. With respect to this issue, much work have to be done to understand what prevents these communities from being stable.

10.5.1.3 Hypothesis 3

The experiments presented in the previous section showed that *communities* that are composed of 100% of compliant *solvers* reveal to be more stable and less prone to *population* turnover. We explained such results by the fact that these *solvers* are never sanctioned and thus, they are never excluded. However, the *social impact theory* showed that such *population* never exists in real setting. The experiences that were conducted by *Ash* [Asch, 1955] (cf. Section 4.2.1.2) report a proportion of 80% of *compliant* individual in real life settings.

In the light of that, our main concern in this section is to evaluate whether *populations* in which the *proportion* of *compliant* solver falls below 100% are able to maintain *stability* property. To that aim, we make use of *population 3* and *population 4* with possess different proportion of *compliant* solvers. Recall, *population 3* is composed of agent which propensity to comply with the collective policy of 0.5. That means that these agent will integrate the collective policy with a probability of 0.5. So at any time, *population 3* will have 50% of its members complying with the collective policy. Similarly, *population 4* will have a proportion of 80% of complying agents. This population corresponds to the normal settings as discussed above.

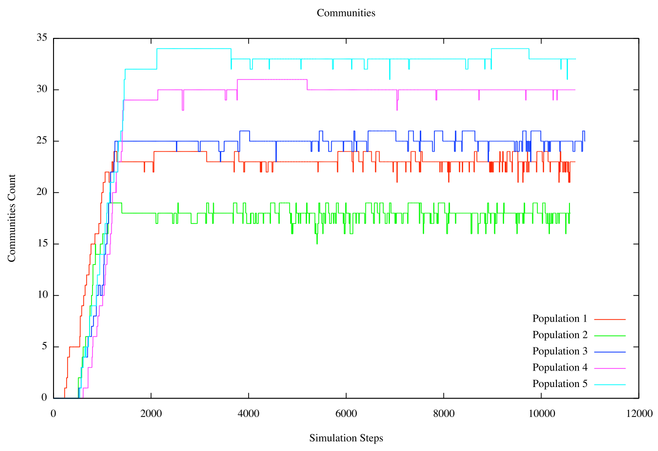


Figure 10.8 – Evolution of communities in populations 5, 6 and 7.

The average values of the number and the size of the communities formed in *populations* 1, 2, 3, 4 and 5 is plotted in Figures 10.8 and 10.9. *Populations* 1,2 and 5 have been added as control population.

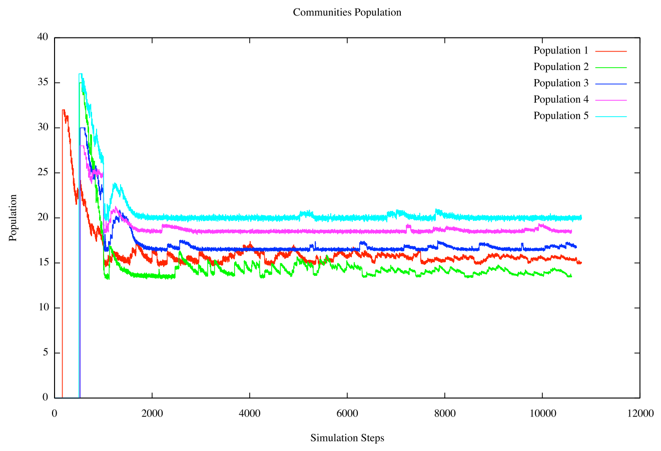


Figure 10.9 – Evolution of communities size in populations 5, 6 and 7.

The results obtained in these experiments showed that *communities* stability is positively affected by the amount of *compliant* agent composing the population. This result also reinforced the findings of the previous section in the sense that the propensity of *solvers* to comply with the collective policies of their communities affects significantly the stability of their communities. This tendency is confirmed also in terms of *communities size* as depicted in Figure 10.9.

The average of the *communities size* is directly affected by the percentage of compliant solvers. However, these values are not *proportional*; If we compare the number of agents that belong to a community in *population 3* (around 425) and the those from *population 4* (around 600) we notice that *population 4* performs better than expected (this value should be around 550), if we consider a strict correlation between the number of communities and the size of each community.

10.5.1.4 Hypothesis 4

In previous sections, *population 5* reveals to be the *population* sample having the most stable communities. However, the experiments conducted in these sections showed also that this population was not so idyllic as one could expect.

As discussed so far, a *solver* can leave a community because it has been *excluded* after having violated the collective policy, or because the agent decided to leave the community because of a too restrictive or a too permissive collective policy. The first situation is not relevant for *population 5* as the *solvers* of this population never violate their collective policies. Thus, the solely explanation that we can defend is that these *solvers* leave their communities because they enter in conflict with the *collective policies* in use in their communities. It is these issues that we will investigate in this section. Our objective is to see whether putting in place the complete *social compliance* micro-macro loop will reduce the number of agents leaving their population, and thus participate in the improvement of communities stability.

For this purpose, we make use of *population 6* and *population 7* aforementioned. These populations correspond to, respectively, *population 4* and *population 5* in which the *evolution* features has been activated. So the *solvers* of both populations are able to trigger evolutions in the collective policies of their communities. The results of these populations, compared to *population 5* represents the base case of all previous evaluations are plotted in Figures 10.10 and 10.11.

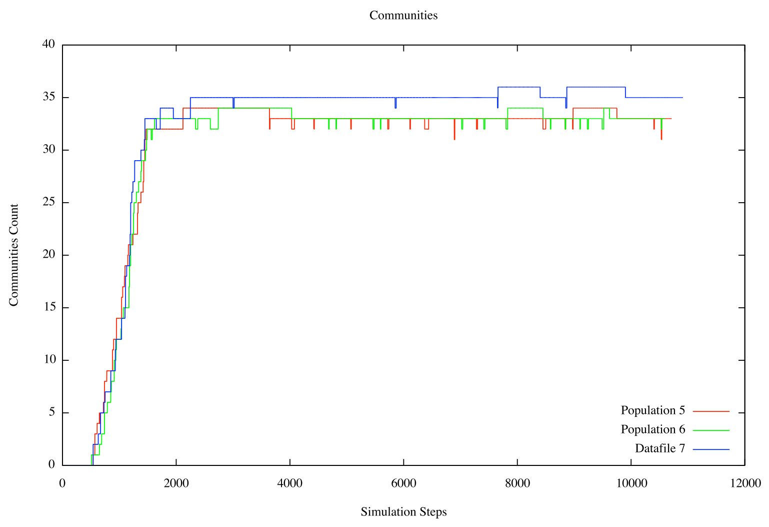


Figure 10.10 – Evolution of communities in populations 5, 6 and 7.

As shows in Figure 10.10, after allowing *solvers* to change their collective policies, the number of communities in *population 4* (6 in this evaluation) had risen above 30, while this number exceeded 35 for *population 5* (7 in this evaluation). These results showed that *populations* in which only 80% of *complying solvers* reveal to be able to have as much stable communities as *communities* which members were 10% complying.

The analyse of the above results in terms of *communities size* revealed that the *evolution* feature reduces substantially the members turnover. Surprisingly, *population 6* (former 4) was even able to perform better that *population 5* as the average population of each community was almost as good as in *population 7*. While, our experiments do not allow a explain these effects, we thinks that the *ability* of solver to adapt their collective policies makes improves their stability. This stability participates in the growth of the size of these communities, and makes these *communities* scale.

In sum, based on the above evaluation, we assume that *evolution* has a positive impact on the stability and even *scalability* of virtual communities. These results confirmed also the

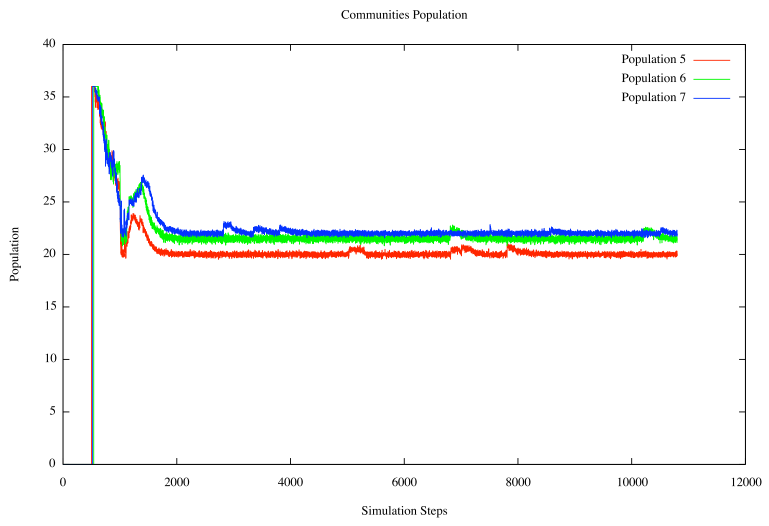


Figure 10.11 – Evolution of communities size in populations 5, 6 and 7.

benefit of applying *social influence theory* to trust management as *socially compliance solvers* tend to form more stable and bigger communities.

10.6 Conclusion

In this chapter, we report part of the preliminary results towards the evaluation of ASC-TMS. This evaluation was made based on the *open innovation* scenario presented in this chapter. Our main concern was the assessment of the impact of applying *social influence theory* in trust management within virtual communities. To that aim, we have empirically evaluated our approach, and the results of our experiments show that *integration* and *evolution* significantly improve the stability of *virtual communities*. In additions, the results show that *combination* could be detrimental to communities stability unless it is used jointly with *integration*.

Of course, these results remains incomplete and more sound a clearer evaluations of the impact of applying ASC-TMS to virtual communities has carried out to definitively confirm whether the *hypothesis* formulated in this chapter are supported by ASC-TMS.

Also, several research question remain open and motivate for further investigation. For instance, the populations used in this evaluation was selected manually based on *social science theories* (e.g. [Moscovici, 1969, Bowser, 2013]). However, to be sound these evaluation should consider a richer pool and heterogeneous (in terms of collaboratives, cooperativeness and competence) set of populations.

10.7 French Summary

Le chapitre précédent nous a permis d'évaluer en partie nos contributions en les confrontant aux contraintes d'un scénario tiré de la vie réelle. Néanmoins, cette évaluation demeure incomplète si on la couple pas à une évaluation à plus grand échelle de notre système. Ainsi, l'objectif de ce chapitre est de tester l'impact à grand échelle de notre système sur les communautés virtuelles. nous sommes plus particulièrement intéressés par l'évaluation des avantages de l'utilisation des concepts inspirés de la théorie d'influence sociale (i.e., combinaison, intégration et évolution) sur la gestion de la confiance au sein de systèmes larges comptant non pas des dizaines ou des centaines de membres mais des milliers. Notre évaluation sera donc centrée sur l'impact de notre système sur la stabilité et la dynamique des communautés le mettant en œuvre.

10.8 La plateforme de simulation multi-agent Repast

Repast [Collier, 2003], est une plateforme de simulation multi-agent libre et open source assez connue. La plateforme a été développée en 2003 par le laboratoire de recherche en sciences sociales de l'université de Chicago pour appliquer la simulation à base d'agent aux sciences sociales. L'outil permet une simulation à événement discret ainsi que toute une panoplie d'outils de monitoring et de visualisation de résultats. La plateforme possède plusieurs versions dont chacune est dédiée à un langage particulier (e.g. Java , Logo , . NET, Lisp , Prolog et Python). Dans notre implémentation, nous avions utilisés la version Repast-J qui est la version Java de Repast.

Le choix de Repast nous est apparu évident de par sa vocation à réaliser des simulations multi-agent en relation avec les théories émises par les chercheurs en sciences sociales. Ainsi, souhaitant étudier l'impact de l'application des théories sociales de l'influence sociale à la gestion de la confiance, Repast nous semble être la meilleur solution pour cela.

10.9 Modèle de simulation

Le modèle dont nous avions décrit l'implémentation dans le chapitre 8 est relativement complexe pour être implémenté et évalué convenablement. Un bon nombre de concepts utilisés dans ce modèle seraient techniquement difficiles à implémenter sous Repast (e.g. Le moteur BDI des agents). Pour cela, nous avons identifié le sous-ensemble minimal du modèle qui permettrait son implémentation sous Repast tout en reprenant l'esprit des contributions apportées dans cette thèse. Par exemple, fusionner l'agent assistant et l'agent gestionnaire de confiance dans une seule entité nous parait être un choix qui n'affecterait pas grandement les résultats de notre évaluation. De manière analogue, nous avons fait un certain nombre de simplifications de conception afin de rendre cette simulation réalisable. La figure 10.13 illustre les principales classes Java utilisées dans notre simulation.

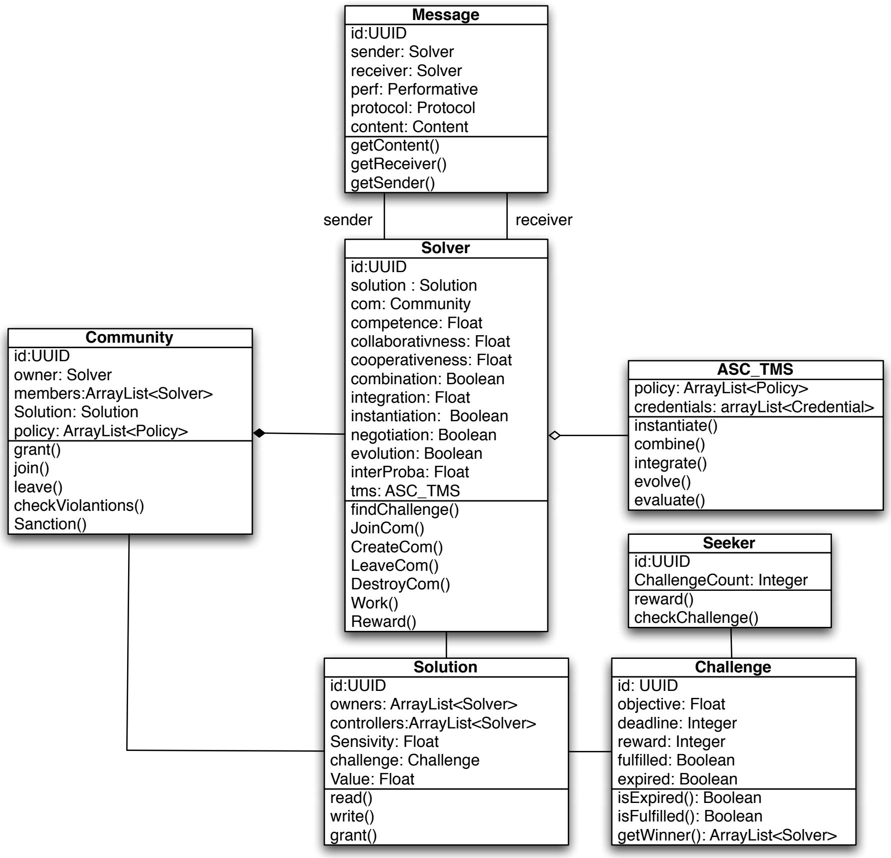


Figure 10.12 – Diagramme de classes en UML des éléments du modèle de simulation

Le modèle se compose de deux classes d'agent; *seeker* et *solver*, d'une classe est utilisé pour représenter les communautés (i.e. Community), d'une classe représentant les solutions proposées par les *solvers*, une classe représentant les *challenges*. Enfin, nous avons encapsulé toutes les fonctionnalités proposées par le ASC-TMS dans une classe dédié à laquelle la classe *solver* délègue les questions liées à la gestion de la confiance. Les détails de chaque classe sont présenté dans le manuscrit d'origine.

10.10 Scénario de simulation

Dans cette section, nous décrivons le scénario utilisé lors de nos simulations. Ce scénario est inspiré de l'exemple de communauté virtuelles d'innovation vu dans le chapitre précédent (cf. Chapitre 9).

Comme illustré dans la Figure 10.2, la simulation est initialisée par la création d'un *seeker*

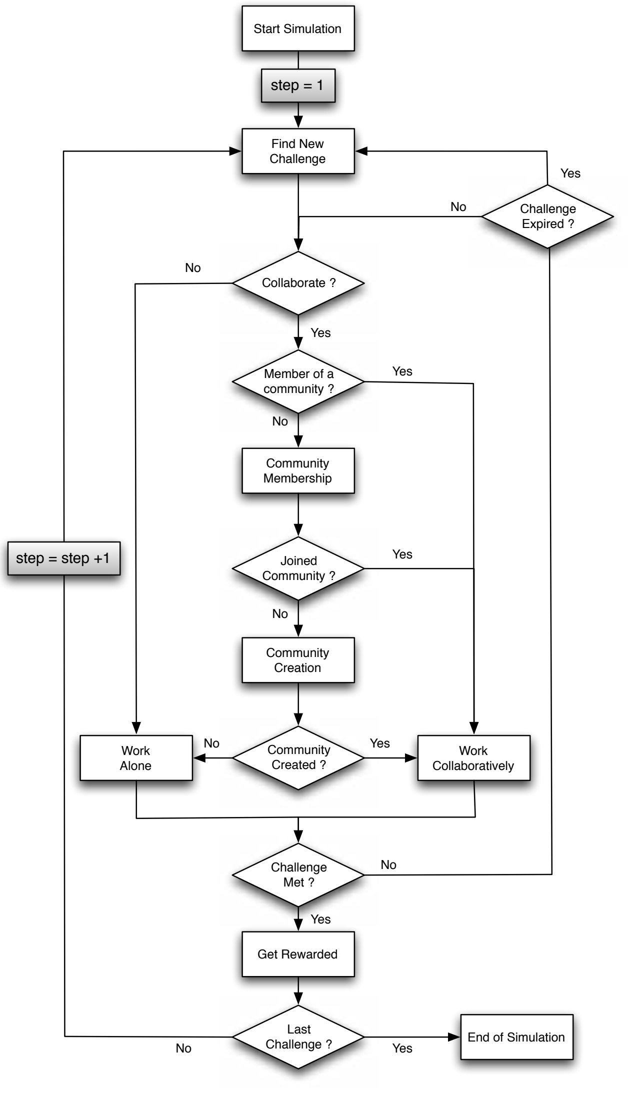


Figure 10.13 – Flowchart describing the abstract innovation process

et un certain nombre de *solvers*. L'objectif du *seeker* est de trouver des solutions à un certain nombre de challenges (i.e. `chCount`). De leur côté, les *solvers* cherchent à optimiser leur utilité en gagnant des challenges individuellement, ou en étant membres d'une communauté qui a gagné un challenge.

La simulation commence donc quand le premier challenge est introduit dans le système. L'objectif initiale de chaque *solver* est dont de trouver un challenge à relever. Dès lors, chaque *solver* décide soit de travailler individuellement soit de collaborer avec d'autres au sein d'une communauté. Cette décision, est propre à chaque *solver* en fonction de sa tendance à vouloir collaborer ou non. Cette tendance, suit une loi uniforme entre 0 et 1.

Une fois la décision de collaborer (ou non) prise, chaque *solver* se met à la recherche d'une communauté, sinon il va en créer une. À ce stade, si le *solver* à réussi a rejoindre une communauté, il vaut travailler de manière collaborative, sinon il va travailler tout seul. Un cycle de simulation s'achève une fois que chaque *solver* a été en mesure de travailler. Une fois que le *solver* a fini de travailler, il va évaluer si la ressource qu'il a amélioré a atteint l'objectif fixé dans challenge. Si c'est le cas, le statut du challenge est mis à réussit et le ou les *solvers* qui ont participé à sa résolution sont récompensés. Si non, le *seeker* vérifie que le challenge n'a pas expiré, auquel cas il introduit un nouveau jusqu'à épuisement des challenges.

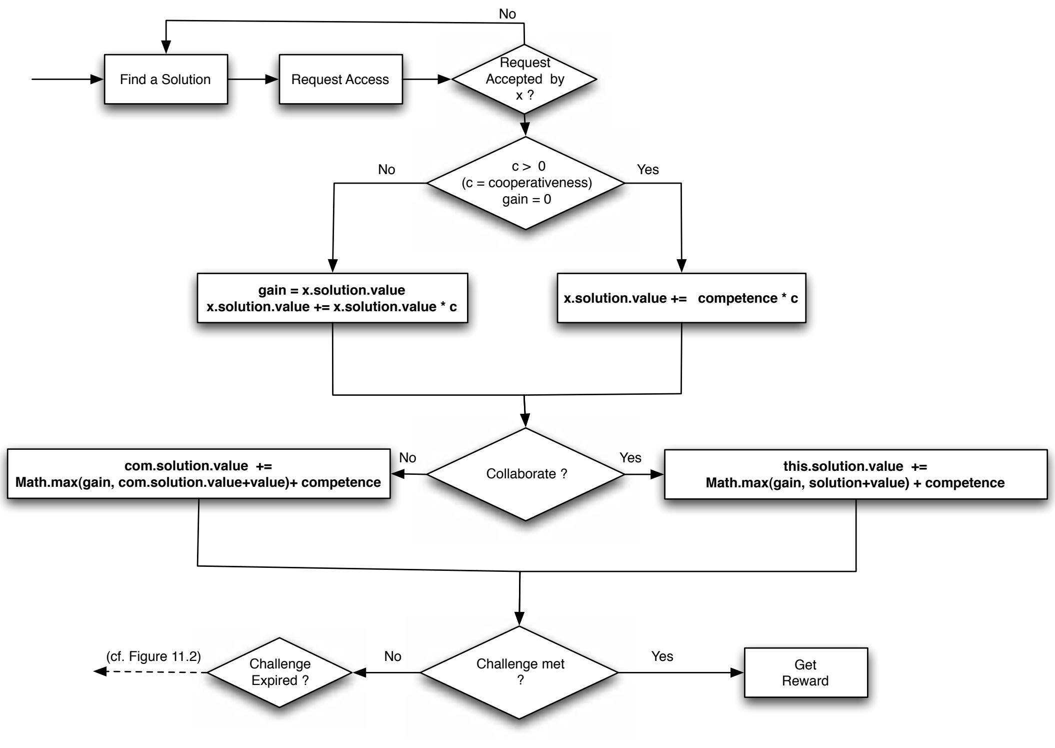


Figure 10.14 – Flowchart describing the abstract innovation process

Dans la figure 10.14, nous détaillons le processus de création présentée de manière abstraite dans la Figure 10.13. L'objectif est d'expliquer comment le degré de coopération des *solvers* (i.g. cooperativeness) influe sur leur comportement quand ils accèdent aux ressources des autres.

Comme illustré dans la figure, si le degré de coopération est strictement positif (i.e. > 0), le *solver* va améliorer la solution à laquelle il accède de manière proportionnelle à cette valeur. Par contre, si cette valeur est négative ou nulle, le *seeker* va profiter de cette ressource est la vandaliser de manière proportionnelle à cette valeur.

Enfin, au début de la simulation, chaque *solver* est doté de 100 unités en matière d'utilité.

La manière avec laquelle cette utilité est affectée par les actions des *solver* est présenté dans le tableau 10.7.

10.11 Réglages de la simulation

Dans cette section, nous passons en revue les principaux réglages utilisés lors de notre simulation. Il s'agit essentiellement des paramètres et ainsi que les indicateurs qui ont servis à l'évaluation du ASC-TMS.

10.11.1 Paramètres de simulation

Les paramètres utilisés lors de notre simulation se regroupent en deux catégories: *les paramètres des challenges* et les *paramètres des solvers.* Ces paramètres sont considérés comme les paramètres par défaut, sauf précisions contraires.

- **Paramètres des challenges**
    1. `Challenges Count` - Le nombre de challenges introduits
    2. `Challenges Objectives` - L'objectif à atteindre pour gagner le challenge. Cette valeur a été fixée de manière expérimentale à 10^4 afin d'avoir des challenges assez longs et disputés.
    3. `Challenges Rewards` - L'utilité que les gagnants d'un challenge vont remporter. Cette valeur a été fixée à 1000.
    4. `Challenges Deadline` - Le temps alloué à chaque challenge. cette valeur a été fixée à 1000.
- **Paramètres de Solvers:**
    1. `Policies` - Chaque *solver* dispose de deux politiques ont les *types*, *poids* et *valeurs de seuil* ont été fixé de manière aléatoire. La première politique est utilisée par les *seekers* pour prendre des décisions sur l'accès aux ressources alors que la seconde est utilisée pour des décisions sur la collaboration.
    2. `Credentials`-Chaque *solver* est doté de 10 *credentials* dont cinq concerne sont identité et cinq concernent ses compétences.
    3. `Competence` - La *compétence* des *solvers* suit un loi normale entre 0 et 1.
    4. `Collaborativeness` - La degré *collaboration* des *solvers* suit une loi uniforme entre 0 et 1.
    5. `Cooperativeness`- Le degré de *coopération* a été fixé manuellement afin d'avoir une population composé de 10% altruistes, 40% coopératifs, 40% égoïstes and 10% malveillants
    6. `Interaction`: Ce paramètre détermine la probabilité à laquelle un *solver* va interagir avec un autre solver. Ce paramètre représente une probabilité qui a été fixée à 0.8.

10.11.2 Métriques de simulation

Les métriques représentent les sorties de notre simulation. Elles constituent les éléments statistiques sur lesquels nous allons reposer lors de notre évaluation. Ces métriques sont décrites dans ce qui suit:

- `sovCount`: le nombre de *solvers* dans la simulation.
- `comsCount`: le nombre de communautés.
- `comsSize`: la population de chaque communauté.

10.12 Évaluation

Dans ce chapitre, nous allons procéder à l'évaluation de l'intérêt des mécanismes inspiré de la théorie de l'*influence sociale* et leur application à la gestion de la confiance au sein de communautés virtuelles.

10.12.1 Hypothèses

Lors de notre évaluation, nous nous sommes particulièrement intéressé à vérifier l'impact de notre système sur la dynamique des communauté dans quatre situations formulées sous forme d'hypothèses. Ces hypothèses sont décrites dans ce qui suit :

- *Hypothesis 1:* Les communautés dans lesquels les membres sont capables de *combiner* ont une dynamique positive.
- *Hypothesis 2:* Les communautés composées d'agents capables de se conformer ont une meilleure dynamique que elles ou les agents en sont incapables de le faire.
- *Hypothesis 3:* La dynamique des communauté est fonction de la proportion d'agents capables de se conformer.
- *Hypothesis 4:* Quand les agents sont capables de faire évoluer leurs politiques collectives, la dynamique de leur communauté s'en trouve davantage améliorée.

10.12.2 Results

Dans cette section, nous présentons une partie des résultats obtenus dans cette thèse de doctorat. Les résultats ont été obtenu en utilisant un population composée de 1000 *solvers* et un *seekers* qui introduit dans le système 10 challenges avec la configuration par défaut. Afin d'évaluer la validité des hypothèses formulées dans la section précédente, nous avons utilisé 7 populations d'agents différentes.

Comme illustré dans le tableau 10.9, les populations se différencient entre elles en terme de fonctionnalité du modèle (combinaison, intégration et évolution). En plus nous avons utilisé des

population	Combination	Integration	Evolution
population 1	`false`	0	`false`
population 2	`true`	0	`false`
population 3	`true`	0.5	`false`
population 4	`true`	0.8	`false`
population 5	`true`	1	`false`
population 6	`true`	0.8	`true`
population 7	`true`	1	`true`

Table 10.9 – Populations used in the simulation

populations qui ont différents degrés d'intégration pour évaluer l'impact de la conformité sur la dynamique des communautés. Aussi, afin de minimiser l'impact des paramètres aléatoires utilisés dans la simulation, nous avons calculé la moyenne de 100 exécutions de de chaque scénario. Dans ce qui suit, nous allons survoler les résultats obtenus pour ces évaluations.

10.12.2.1 Hypothèse 1

Dans cette simulation, la population 1 est utilisée comme population témoin (population de contrôle) alors que la population 2 sera utilisée pour évaluer l'impact de la *combinaison* sur la dynamique des communautés.

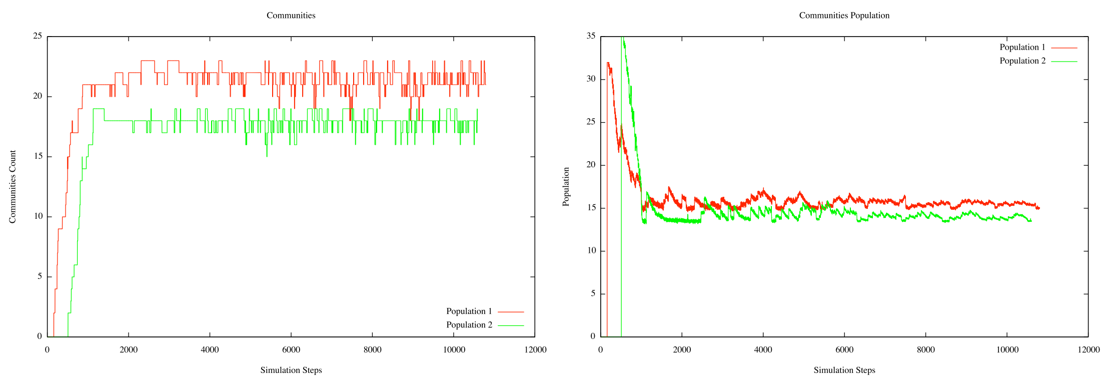


Figure 10.15 – Évolution des communautés et des populations

Comme le montrent les résultats (cf. 10.15), la population 2 est pire que la 1 ce qui peut s'expliquer par le fait que les agents sont pénalisés par le processus de construction des politiques alors qu'ils n'ont tirent aucun profit (pas d'intégration). Donc, l'hypothèse 1 n'est pas vérifiée sans que cela remette en cause notre modèle.

10.12.2.2 Hypothèse 2

La simulation précédente ne nous a pas permis de valider l'impact de la combinaison sur la dynamique des communautés. Nous avions expliqué cela par le fait que les agents étaient incapables de les prendre en considération. Ainsi, cette série de simulations vise à vérifier leur réel impact quand les agents sont capables non seulement de créer leurs politiques collectives mais également et sur tout de les prendre en considération.

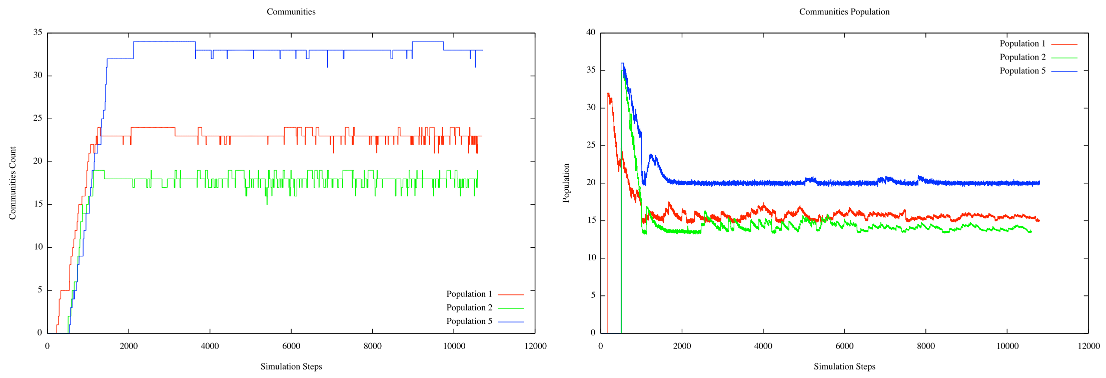


Figure 10.16 – Évolution des communautés et des populations

Comme le montrent les résultats obtenus (cf. 10.15), la dynamique de la population 3 est nettement meilleure que les deux autres populations. Cela par s'explique par la tendance de ces agents à se conformer à leurs politiques collectives. Pour simplifier, ces agents ne violent jamais leur politique et donc ne sont jamais exclus et les communautés n'implosent que rarement.

10.12.2.3 Hypothèse 3

Dans la simulation précédente, nous avions utilisé une population composée d'agents se conformant à 100% avec leurs politiques collectives. Évidemment, cette configuration utopique n'est pas réaliste. C'est pourquoi, dans cette série de simulations nous avons voulus tester différents degrés de conformité afin de voir jusqu'à quel niveau les communautés arrivaient à se maintenir avec des agents anticonformistes.

Les résultats montrent que la dynamique des communautés et leur stabilité est proportionnelle au degré de conformisme de leur population. Ainsi, l'aptitude des agents à se conformer affecte significativement la stabilité de leurs communautés.

10.12.2.4 Hypothèse 4

Jusqu'à maintenant, nous avions passé sous silence les raisons pour lesquels les agents quittaient leurs communautés. Deux raisons expliquent ce comportement, l'exclusion suite à des violations de politiques de confiance ou l'abandon quand l'agent est confronté à des politiques collectives trop restrictives. Ainsi, la capacité des agents à pouvoir changer ces politiques collectives

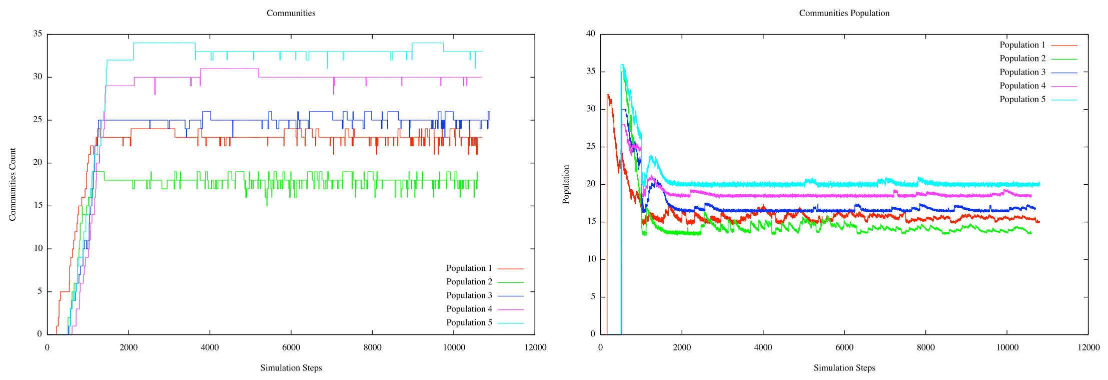


Figure 10.17 – Évolution des communautés et des populations

est normalement un élément favorisant la stabilité des communautés. C'est que nous voulions vérifier en permettant aux agents de la population 4 d'adapter leurs politiques collectives quand c'est possible.

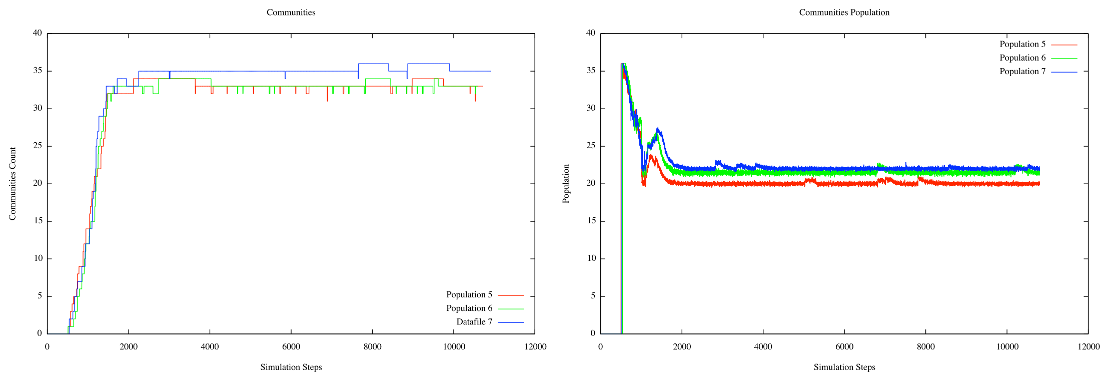


Figure 10.18 – Évolution des communautés et des populations

Comme l'illustre la Figure 10.18), les résultats furent surprenants. La capacité des agents à changeur de manière consensuelle leur politique collective a amélioré de manière substantielle la stabilité des communautés formées dans la population 4. Cette population dépasse même les résultats obtenus avec la population 6 considérée comme idéale en terme de conformité.

Part V

Conclusion

Chapter 11

Conclusion

The various contributions developed in this thesis are linked by the underlying theme of attempting to assist *open* and *decentralised* virtual communities members in their trust decisions. In this conclusion, we bring together these contributions and discuss how they achieve this objective.

11.1 Summary and Contributions

Our main aim at the beginning of this thesis was to develop a system that assists members of open and decentralised virtual communities in their trust decisions. We emphasised how the inherent *social* and *dynamic* aspects of these systems are challenging the ways trust is currently managed in these communities and the necessity to cope with them.

To that aim, we proposed in this thesis *Adaptive and Socially-Compliant Trust Management System* (ASC-TMS). The novelty of ASC-TMS lies in its ability to exhibit *social-awareness* and *context-awareness* features. Social-awareness refers to ability of the trust management system (TMS) to handle the social nature of VCs by making trust evaluations that are collectively harmful, while context-awareness refers to the ability of the system to handle the dynamic nature of VCs by making trust evaluations that are always in adequacy with the context in which these evaluations are undertaken.

The development of as so called *socially-aware TMS* involves two sub-objectives: the *specification of collective policies* and the *enforcement of these collective policies.* To meet these objectives, we draw inspiration from decentralised trust management (cf. Chapter 3) and sociology disciplines (cf. Chapter 4). The contributions made in this thesis towards the fulfilment of these objectives are summarised as follows:

- We believe that this study is a step forward bridging the gap between *social science* and *trust management* disciplines. This thesis also reinforces the interplay between *trust management* and *distributed* artificial intelligence. In the light of that, the state-of-the-art presented in Part II constitutes the initial contribution of this thesis. In Chapter 3 we reviewed and analysed existing trust management systems and stressed how these systems failed addressing the social dimension of nowadays environments. In Chapter 4, we have made explicit the mapping between mechanisms proposed in trust management such *combination* (cf. Section 7.5.1 and *integration* (cf. Section 7.5.2) and corresponding concepts from *social influence theory.* We stressed how these mechanisms, although insufficient, could be adapted and used to implement concepts of *social influence theory* and thus made the first steps towards the fulfilment of objectives O.1.1, O.1.2 and O.2.2.

- As a second step, we developed in Chapter 5 a trust management system in which the social dimension of trust decisions is explicitly considered. To that aim, we designed a policy specification language that is able to express both *individual* and *collective trust policies*. While individual policies represent the personal conditions based on which an individual makes its trust decisions, collective policies represent the conditions that should be considered by every members of the community when making decisions that can affect all the members of this community.

- The ability to specify collective policies calls for a mechanism that makes the specification of such policies by several members, each having its own individual policy, possible. In Chapter 7, we adapted mechanisms used in decentralised trust management systems to attack this problem. The proposed mechanism allows VC members to build their collective policies in a decentralised way. We proposed an mechanism that implements two of the heuristics used in these works: *deny overrides* and *permit overrides*. The first heuristic builds a policy that never accepts a request that one of the members would have accepted (reject overrides), while the second heuristics guarantees that the resulting policy never rejects a request that one member would have accepted (accept overrides). The difference between using integration and directly using the collective policy lies in the fact that when the individual policy is more accurate than the collective policy and does not conflict with it, the individual one influences the most the trust evaluation. For instance, if the collective policy is too permissive, if the individual policy is restrictive enough, the use of integration will prevent the community from wrong decision that would not have been possible only by using the collective policy (Objective O.1.1)

- Once the first sub-objective has been achieved (i.e. specification of collective policies), we tackled the second one that consists in *enforcing* these collective policies. However, as we target open and decentralised way, the enforcement of these collective policies must necessary be done in a decentralised way by each member. To that aim, we proposed an mechanism that VC members can use to *integrate* the collective policies along with their individual ones. The integration is made based on four different heuristics, each implementing a particular form of compliance. So the decision to comply and how to comply is left to the VC member. The main advantage of this mechanism is that it guarantees that, if the member wants, his trust evaluations will never be in conflict with the collective policies of his community.

- Offering to VC members a way to enforce collective policies (i.e. comply) does not necessary mean that they will do. To that aim, we draw inspiration from social sciences to borrow models proposed in the *social influence theory* to understand when and how collective policies are created and enforced in real societies. Based on these models, we selected *multi-agent systems* (MAS) as target technology to allow a decentralised and socially-compliant trust management. At this stage, we were particularly interested in the ability of MAS to express *norms* that oblige VC members to comply with the collective policy of their communities. Thinking ASC-TMS as an agent allowed us to be profit from the adequacy of the mechanisms proposed in MAS with the concepts of social influence theory.

The development of a context-aware TMS involved two sub-objectives: the *adaptation of individual policies* and the *adaptation of collective policies*. In the following, we discuss the contributions made in this thesis toward the satisfaction of these objectives:

- The adaptation of policies, either individual or collective, calls for a policy specification language that makes such adaptation possible. To that aim, our first contribution at this stage was to make our policy language *semantic*. This has been done by describing the terms and values used to specify policies into an ontology (i.e. *trust factors ontology*). This ontology, makes the agents implementing ASC-TMS able to understand and reason on conditions stated by the policies they use.

- Then we defined an algebra that ASC-TMS uses to *adapt* policies. Based on this algebra, ASC-TMS can *relax*, *restrict* and *update* in many different ways the policy it is manipulating. This algebra relies on the aforementioned ontology to bring the desired adaptation to policies (Objective O.2).

- We defined meta-policies that react to environment changes and adapt active policies in consequence. These meta-policies are expressed as *event-condition-action* rules that virtual community members can personalise to adapt their policies in an appropriate way (Objective O.2.1).

- Having this adaptation mechanism, we defined an *adaptive trust negotiation* strategy that is able to *relax* the conditions stated by the policy used during the negotiation. This adaptive strategy allows virtual community members to make their negotiation successful by making compromises (Objective O.2.1).

- Finally, we used the adaptation mechanisms to make virtual communities members trigger adaptation of their collective policies. To that aim, we make use of a voting scheme that members rely on to communicate and agree about the adaptation they want to bring to their collective policies (Objective O.1.2).

Moreover, we showed how the contributions presented in this section have been *implemented* (cf.), *applied* (cf. Chapter 9) and *evaluated* (cf. Chapter 10).

In Chapter 8, we presented the implementation of ASC-TMS and the agent-based virtual community platform. The system has been implemented and deployed using the *JaCaMo* multi-agent programming platform. We motived the convenience of this choice and showed how the implementation of the *theoretical* concepts that underpin our contribution can be simply implemented.

In Chapter 9, we illustrated the applicability of ASC-TMS based with an *open innovation* virtual community scenario. We showed the extent the members of these communities could be benefit from using ASC-TMS. We claimed also that ASC-TMS remains generic enough to be applied to any form of social structure.

Finally, we presented in Chapter 10 some empirical evaluations based on which we tried to observe the impact of using ASC-TMS on large scale populations. The objective of our experiments was to confirm or disconfirm the benefit of integrating *social influence theory* in trust management.

The preliminary results are promising as they show that population in which *combination*, *integration* and *evolution* mechanisms are jointly used reveal to be more stable.

11.2 Open Issues and Future Works

Given the multidisciplinary nature of the theories and techniques used in this thesis, we did not have the opportunity to explore some aspects of our proposal in greater depth. In the following, we recognise the limits of our proposal and discuss how the contributions presented in this thesis can be extended.

Social Influence Theory

Although our evaluations show that our approach has a reasonably positive impact on the *stability* of virtual communities, we have not conducted in depth sophisticated analysis on performance of ASC-TMS. Several questions remain open and motivate for further investigations. For example, a basic question is "what would be the impact of ASC-TMS on populations composed of only *malicious* or *selfish* participants, or different mixtures of such profiles?".

Also, one of the main limits that we have been able to identify when evaluating ASC-TMS relates to the impact of a *malicious minority* on the policy used by the *majority*. Indeed, in our experiments we showed that the ability of agent to adapt their collective policies affects positively the stability of these communities. However, we did not evaluate situations in which malicious members collude to make the collective policy less restrictive.

Finally, more statistically sound and clearer evaluation should be conducted to definitively validate or invalidate whether the hypothesis formulated in Chapter 10 are supported by ASC-TMS or not. More particularly, ANOVA (analysis of variance) should be carried out to test whether our results are statistically significant or not.

Learning Trust Factors

ASC-TMS relies on properties of agents to specify policies. For that purpose, we assume that a certain correlation between partners' properties and their degree of trustworthiness exists.

Such assumption has been proved to be quite realistic in many works [Xin, 2011, Burnett, 2011]. However, in our approach we did not provide any mechanism based on which the users of our system could discover such correlations, making our approaches incomplete. With respect to this issue, the use of *machine learning* techniques could improve the accuracy of our approach and its applicability in the same way [Burnett et al., 2013b].

Ghost Policies

The decentralised of ASC-TMS may lead virtual communities to inconsistant states. For instance, if a member of the community was offline during the evolution of the collective policy, thus the collective policy he will use will be probably obsolete. This situation may bring this agent to violate the new collective policy in use, and consequently exposes him to the violation of the *social compliance* norm (cf. Section 7.4.2.2. These policies are called *ghost policies* and may affect the stability of the community. There exist some solutions in the literature (e.g. [Serban and Minsky, 2009]) that address the safe evolution of policies that we could consider to overcome this issue.

The Ontology

The *trust factors ontology* is cornerstone for ASC-TMS. Based on this ontology, the system is able to understand, reason on and adapt the policies it is manipulating. Moreover, based on this ontology, ASC-TMS accomplishes combination and integration tasks that are essential for the implementation of the concept of social compliance. However, the expressiveness of this ontology is obviously limited. In this regard, much work has to be done towards the identification and specification of all relevant *trust factors* with respect to trust management in VCs.

Furthermore, we assumed in this thesis that all agents are using the same ontology. However, in real settings, such assumption may not hold; people may not agree on the semantic of each trust factor. This issue constitutes a real problem for the *applicability* of our approach in systems in which several ontologies may coexist. Even if we believe that existing *ontologies* alignment techniques could, in theory, resolve such conflicts, we assume that this issue should be explored more seriously to confirm our intuition.

Policy Language

The policy language we specified in Chapter 6 has been designed in the light of the objectives of this thesis. Nevertheless, this language has to be more rigorously specified to be consider its use realistic. Also the use of this language in the context of real-life scenarios (cf. Chapter 9), allowed us to shed light on some of its limits. For instance, our policy language is not able to express alternative policies; it is not possible to specify two policies that apply to the same pattern. If we want to add this features, we would allow ASC-TMS to evaluate all policies that apply to a particular pattern then aggregate the results (e.g. provide the highest value).

Meta-Policies

Another research issue relates to the possibility to activate and deactivate *meta-policies*. In Chapter 8, we showed how *assistant agent* can transmit to its TMA (i.e. ASC-TMS) meta-policies. However, we did not explore how allowing the user to adapt their meta-policies could impact the functioning of the system. In this regard, we could also evaluate whether *meta-meta-policies* could be specified to *automatically activate and deactivate* meta-policies based on the result of the adaptation they brought to policies.

Application areas

ASC-TMS has been specifically designed for VCs. However, we believe that is worth to apply the approach advocated in this system to other areas involving collective structures. For instance, it may be possible to apply our trust management system to multi-agent coalitions. Another interesting area where ASC-TMS can be used is the recently emerged federated cloud [Buyya et al., 2010, Paraiso et al., 2012] (also called cloud federation). In these systems, multiple cloud services group together to match business needs of a client. So a federated cloud is the union of several smaller parts that perform a common action. The functioning of these systems correspond in several perspectives to the one of virtual communities. We expect that the application of our approach to these environments produces the same impacts.

Deployment

We presented in Chapter 8, the implementation of the proof of concept ASC-TMS on the top of the *JaCaMo* platform, a challenging research issue would be to make the mechanisms proposed in this thesis available in real-world applications. Part of this implementation has already been deployed on the top of the Android platform using the *JaCa-Android* [1]. However, this application does not cover all mechanisms provided in this thesis as *JaCa-Android* is built upon *Jason* and *CArtAgO*, thus all aspects that relates to $\mathcal{M}OISE$ have not been deployed.

Finally, another interesting perspective would be to use *JaCa-Web*[2] to develop a web version of ASC-TMS to make it available to the largest public. Such deployment can bring much value to the system and in depth understanding of VC members behaviour. It can also be used to evaluate how the decisions supported by ASC-TMS aided VC members and will probably open opportunities rooted in ground reality.

[1] http://jaca-android.sourceforge.net/
[2] http://jaca-web.sourceforge.net/

11.3 French Summary

Le travail dont nous rendions compte dans ce manuscrit de doctorat avait pour objectif de concevoir un outil pour assister les membres de communautés virtuelles dans leurs décisions de confiance. Dans cette conclusion, nous dressons un bilan sur notre contributions et discutons comment elles ont permis, de notre point de vue, d'atteindre cet objectif.

11.3.1 Synthèse des contributions

Le principal objectif de notre thèse était de développer un système qui assisterait les membres de communautés virtuelles ouvertes et décentralisées dans leurs décisions de confiance. Nous avions souligné dans l'introduction de cette thèse la nature *sociale* et *dynamique* de ces communautés et auxquels il fallait absolument faire face. Pour cela, nous avons présenté un système de gestion adaptatif et conforme socialement ASC-TMS (de l'anglais Adaptive and Socially-Compliant Trust Management System).

Cette thèse est, à notre connaissance, la première à avoir proposé un système de gestion de la confiance doté de propriétés sociales et adaptatives. L'aspect social du ASC-TMS fait référence à la capacité de notre système à prendre des décisions qui soient sûres non seulement pour l'individu mais également et sur tout pour la communauté. Par ailleurs, l'aspect adaptatif du système fait référence à la capacité du système à prendre des décisions qui soient en parfaite adéquation avec l'environnement dans lequel elles sont prises.

Le développement d'un système de gestion avec des capacités d'adaptation au contexte impliquait deux sous objectifs: (i) la capacité d'adapter les politiques individuelles, et (ii) la capacité d'adapter des politiques collectives. Que ce soit pour les politiques individuelles ou collectives, la possibilité d'adapter une politique impliquait un langage de spécification de politiques qui tolérerait une telle adaptation. Ainsi, notre première contribution dans ce sens était de faire en sorte que la sémantique de nos politiques soit accessible à notre système de gestion de la confiance. Pour cela, nous avons utilisé des technologies du web sémantique, notamment des Ontologies, afin de permettre à notre système (et les agents dans lesquels il a été implémenté) de comprendre et raisonner sur les politiques qu'il utilise. Ensuite, il a fallu définir un ensemble d'opérateurs d'adaptation que nous avons appelé *algèbre d'adaptation des politiques*. Ces opérateurs, qui reposent eux aussi sur l'Ontologie sus mentionnée, permettent au système de changer les politiques de différentes manières. Enfin, nous avons étendu notre langage de politiques afin d'exprimer des méta-politiques avec lesquels il était possible d'exprimer quand (en réponse à quel contexte) et comment (en utilisant quels opérateurs d'adaptation) une politique devait être adaptée.

Le développement d'un système de gestion de la confiance avec des capacités sociales impliquait la satisfaction de deux sous objectifs: (i) la capacité de spécifier à la fois des politiques individuelles mais aussi et surtout des politiques collectives, (ii) la capacité à veiller à l'application et au respect de ces politiques collectives. Pour répondre à ces deux objectifs, nous avons puisé notre inspiration dans le domaine de la gestion de la confiance distribuée (cf. Chapitre 3) ainsi que des théories sociales (cf. Chapitre 4). Du premier domaine nous avons repris le concept de politiques ainsi que les mécanismes de combinaison et d'intégration, alors que le second nous a offert des théories qui nous ont permis de comprendre quand et pourquoi la politique d'un individu devait être adaptée pour se conformer à sa communauté. Nous avions également utilisé le mécanisme d'adaptation présenté plus haut afin de permettre, à la fois, à un individu d'adapter sa politique afin de se conformer à la politique collective, mais également pour permettre aux membres d'une communauté virtuelle de déclencher l'adaptation de leurs politiques collectives.

Enfin, nous avons présenté dans la dernière partie de ce manuscrit comment nous avions implémenté notre système de gestion de la confiance en utilisant le paradigme multi-agent, notamment avec la plateforme de programmation *JaCaMo*. Nous avions également évoqué l'applicabilité de notre approche sur un scénario de la vie réelle. Il s'agissait d'un exemple de communauté virtuelle d'innovation ouverte. Puis, nous avions vérifié l'intérêt de notre approche en simulant une communauté virtuelle composé de 1000 membre en utilisant la plateforme de simulation Repast. Les résultats préliminaires de notre évaluation empirique montrent que les communautés utilisant notre modèle avaient une dynamique plus positive en terme de nombre de communautés et de populations.

11.3.2 Questions ouvertes et perspectives

Compte tenu de la nature pluridisciplinaire des théories, modèles et techniques manipulés dans cette thèse, il nous était difficile d'explorer en profondeur chaque aspect de notre approche. Dans cette section, nous discuterons des limites de nos propositions tout en évoquant quelques pistes d'amélioration.

- Bien que les premiers résultats montrent l'effet positif de notre approche sur les communautés virtuelles, par manque de temps, nous n'avions pas pu réaliser des études sophistiquées sur les performances de notre modèle comparées à d'autres approches. Par exemple, nous aurions aimait pouvoir examiner l'impact d'une minorité malveillante sur une politique collective. Aussi, il aurait fallu tester plusieurs types de population pour voir si le modèle n'a pas de contrepartie en terme de performance dans des configurations particulières. Enfin, il aurait été aussi intéressant de réaliser une analyse des données et une étude statistiques plus rigoureuses sur les résultats obtenus (e.g., Analyse de la variance) pour voir si les résultats sont statistiquement fiables et viables.

- Notre système repose sur une Ontologie des facteurs de confiance dans laquelle nous avions recensé les éléments essentiels à l'évaluation de la confiance. Or, ce facteur repose sur l'existence d'une corrélation entre le comportement d'un individu et les propriétés qu'il possède. Ainsi, il nous semble intéressant d'utiliser des techniques d'apprentissage artificiel (i.e., Machine learning) tel que [Burnett et al., 2013b] afin de permettre au système d'enrichir automatiquement cette Ontologie.

- Le caractère décentralisé des communautés virtuelles peut amener le système à créer un état incohérent. Par exemple, quand une politique collective évolue alors que certains membres n'étaient pas connectés. Dans cette situation, il existerait ce qu'on appelle les *politiques fantômes* qui serait forcément en conflit avec la version à jour de la politique collective. Afin d'éviter cette situation, nous avons identifié quelques travaux (e.g., [Serban and Minsky, 2009]) qui ont été utilisés avec succès dans des situations analogues. Ainsi, une perspective intéressante consisterait à appliquer ces solutions à notre modèle afin de le prémunir de ces politiques fantômes.

- Dans notre système, nous supposons que l'ensemble des membres utilisent forcément la même Ontologie. Or, pour être plus réalistes, il aurait fallut envisager la situation dans laquelle plusieurs Ontologies sont utilisés. Bien que nous pensons que les techniques d'alignement d'Ontologies sont capables d'assurer une certaine forme d'interopérabilité, il est nécessaire de vérifier cette piste et de proposer des solutions en conséquence.

- Le langage de spécification des politiques utilisé dans notre système a été conçu de manière explicite pour répondre aux objectifs de cette thèse. Cependant, pour que ce langage soit ait sa place dans des systèmes réels, ce dont nous sommes convaincus, il faut spécifier sa syntaxe et sa sémantique de manière plus rigoureuses et lever les quelques limités qui ont pu être identifiées. Par exemple, il n'est actuellement pas possible de spécifier des politiques alternatives (deux politiques pour la même décision). Bien que les patterns permettent de la faire, nous n'avons pas encore réfléchi quel impact aurait l'évaluation simultanée de deux politiques et si leurs résultats étaient en conflits. Là aussi, les approches de combinaisons de politiques nous semblent assez matures pour les appliquer à ce type de cas de figure, mais tant que nous l'avons pas testé nous ne pouvons tolérer des politiques alternatives.

- Les métas-politiques sont un concept clé de notre contribution. Nous les utilisons pour activer l'adaptation de politiques individuelles et collectives. Or, à ce stade, nous n'avons pas envisagé que des métas-politiques puissent être adaptés de manière automatique. Au moins, il serait intéressant de voir comment est-ce qu'on pourrait utiliser des métas-politiques pour activer ou désactiver d'autre méta-politiques qui s'avérerait être inefficaces.

- Notre système a été conçu explicitement pour les communautés virtuelles. Or, nous pensons que son application pouvait être étendue à d'autres systèmes dans lesquels les dimensions individuelles et collectives existent. Par exemple, nous pensons que des systèmes comme les coalitions multi-agent ainsi que les organisations virtuelles sont de bons candidats pour étendre le champ d'application de notre approche. Une autre piste d'application serait le domaine récent des fédérations de cloud (Federated Cloud) [Buyya et al., 2010, Paraiso et al., 2012]. Dans ces systèmes, plusieurs fournisseurs de services collaborent pour répondre à la commande d'un client. Le fonctionnement de ces système rejoint sur plusieurs points celui des communautés virtuelles. Par conséquence, nous avons de bonnes raisons de penser que les résultats que nous avons obtenus avec les communautés virtuelles peuvent être reproduites sur ces systèmes.

Bibliography

Bibliography

[Aamodt and Plaza, 1994] Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7:39–59. (Cited on pages 186 and 208.)

[Abi Haidar et al., 2008] Abi Haidar, D., Cuppens - Boulahia, N., Cuppens, F., and Debar, H. (2008). XeNA: an access negotiation framework using XACML. *Annals of telecommunications - annales des télécommunications*, 64(1-2):155–169. (Cited on page 73.)

[Ahonen and Lietsala, 2007] Ahonen, M. and Lietsala, K. (2007). Managing Service Ideas and Suggestions Information Systems in Innovation Brokering. *Innovation*. (Cited on pages 245 and 259.)

[Alberola et al., 2011] Alberola, J., Julian, V., and Garcia-Fornes, A. (2011). Open issues in multiagent system reorganization. In Perez, J. B., Corchado, J. M., Moreno, M. N., Julian, V., Mathieu, P., Canada-Bago, J., Ortega, A., and Caballero, A. F., editors, *Highlights in Practical Applications of Agents and Multiagent Systems*, volume 89 of *Advances in Intelligent and Soft Computing*, pages 151–158. Springer Berlin Heidelberg. (Cited on page 108.)

[Alechina et al., 2013] Alechina, N., Bassiliades, N., Dastani, M., Vos, M. D., Logan, B., Mera, S., Morris-Martin, A., and Schapachnik, F. (2013). Computational models for normative multi-agent systems. In Andrighetto, G., Governatori, G., Noriega, P., and van der Torre, L. W. N., editors, *Normative Multi-Agent Systems*, volume 4 of *Dagstuhl Follow-Ups*, pages 71–92. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. (Cited on pages 106, 108 and 114.)

[Allport, 1985] Allport, G. (1985). *The historical background of social psychology*, pages 1–46. Random House. (Cited on page 95.)

[Andrighetto et al., 2010] Andrighetto, G., Campenni, M., Cecconi, F., and Conte, R. (2010). The complex loop of norm emergence: A simulation model. In Takadama, K., Cioffi-Revilla, C., and Deffuant, G., editors, *Simulating Interacting Agents and Social Phenomena*, volume 7 of *Agent-Based Social Systems*, pages 19–35. Springer Japan. (Cited on page 102.)

[Ardagna et al., 2007] Ardagna, C. A., Damiani, E., Capitani di Vimercati, S., Foresti, S., and Samarati, P. (2007). Trust management. In Petković, M. and Jonker, W., editors, *Security, Privacy, and Trust in Modern Data Management*, Data-Centric Systems and Applications, pages 103–117. Springer Berlin Heidelberg. (Cited on page 61.)

[Arrow, 1974] Arrow, K. J. (1974). *The Limits of Organization.* W W Norton &amp; Company Incorporated. (Cited on page 20.)

[Arrow, 1984] Arrow, K. J. (1984). *The Economics of Agency.* Institute for Mathematical Studies in the Social Sciences, Stanford University. (Cited on page 20.)

[Artz and Gil, 2010] Artz, D. and Gil, Y. (2010). A survey of trust in computer science and the Semantic Web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):58–71. (Cited on pages 25, 30, 62 and 175.)

[Asch, 1955] Asch, S. E. (1955). Opinions and Social Pressure. *Scientific American*, 193:31–35. (Cited on pages 97, 202 and 279.)

[Ashby, 1947] Ashby, W. R. (1947). Principles of the self-organizing dynamic system. *The Journal of general psychology*, 37(2):125–8. (Cited on page 106.)

[Austin, 1962] Austin, J. L. (1962). *How to do Things with Words: The William James Lectures delivered at Harvard University in 1955.* Clarendon Press. (Cited on pages 133 and 141.)

[Axelrod and Hamilton, 1981] Axelrod, R. and Hamilton, W. (1981). The evolution of cooperation. *Science*, 211(4489):1390–1396. (Cited on page 28.)

[Axelrod and Hamilton, 1984] Axelrod, R. and Hamilton, W. D. (1984). The Evolution of Cooperation in Biological Systems. In *The Evolution of Cooperation*, pages 88–105. Basic Books. (Cited on page 28.)

[Baier, 1986] Baier, A. (1986). Trust and antitrust. *Ethics*, 96(2):231–260. (Cited on page 27.)

[Bandara et al., 2007] Bandara, A., Damianou, N., Lupu, E., Sloman, M., and Dulay, N. (2007). Policy-based management. In Bergstra, J. and Burgess, M., editors, *Handbook of Network and System Administration*. Elsevier Science. (Cited on page 62.)

[Banzhaf, 2009] Banzhaf, W. (2009). Self-organizing systems. *Encyclopedia of Physical Science &amp; Technology*. (Cited on page 106.)

[Barrett, 2004] Barrett, K. (2004). The dynamic adaptation of security policies in pervasive environments , with contextual information as the catalyst. Technical report, M-Zones Research Programme, Waterford Institute of Technology. (Cited on page 175.)

[Becker and Sewell, 2004] Becker, M. and Sewell, P. (2004). Cassandra: Distributed access control policies with tunable expressiveness. In *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '04, pages 159–, Washington, DC, USA. IEEE Computer Society. (Cited on pages 68, 80 and 85.)

[Becker, 2005] Becker, M. Y. (2005). Cassandra: flexible trust management and its application to electronic health records. Technical Report UCAM-CL-TR-648, University of Cambridge. (Cited on pages 51 and 52.)

[Bertino et al., 2003] Bertino, E., Ferrari, E., and Squicciarini, A. (2003). X -tnl: An xml-based language for trust negotiations. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 81–, Washington, DC, USA. IEEE Computer Society. (Cited on pages 35, 46, 71, 80 and 85.)

[Bertino et al., 2004] Bertino, E., Ferrari, E., and Squicciarini, A. C. (2004). Trust-x: A peer-to-peer framework for trust establishment. *IEEE Trans. on Knowl. and Data Eng.*, 16(7):827–842. (Cited on pages 35, 46 and 71.)

[Bhargav-Spantzel et al., 2007] Bhargav-Spantzel, A., Squicciarini, A. C., and Bertino, E. (2007). Trust negotiation in identity management. *IEEE Security and Privacy*, 5(2):55–63. (Cited on page 191.)

[Bhuiyan and Jøsang, 2010] Bhuiyan, T. and Jøsang, A. (2010). Analysing Trust Transitivity and The Effects of Unknown Dependence. *International Journal of Engineering Business Management*, 2(1):23–28. (Cited on pages 27 and 29.)

[Blaze et al., 1999a] Blaze, M., Feigenbaum, J., Ioannidis, J., and Keromytis, A. D. (1999a). The role of trust management in distributed systems security. In Vitek, J. and Jensen, C. D., editors, *Secure*

Internet programming, pages 185–210. Springer-Verlag, London, UK. (Cited on pages 54, 55, 82, 83, 147 and 175.)

[Blaze et al., 1999b] Blaze, M., Feigenbaum, J., and Keromytis, A. (1999b). Keynote: Trust management for public-key infrastructures. In Christianson, B., Crispo, B., Harbison, W., and Roe, M., editors, *Security Protocols*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63. Springer Berlin Heidelberg. (Cited on pages 4, 34, 45 and 63.)

[Blaze et al., 1996] Blaze, M., Feigenbaum, J., and Lacy, J. (1996). Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, SP '96, pages 164–, Washington, DC, USA. IEEE Computer Society. (Cited on pages 4, 34, 45, 54, 55, 62, 80, 82, 83 and 84.)

[Bocchiaro and Zamperini, 2012] Bocchiaro, P. and Zamperini, A. (2012). Conformity, obedience, disobedience: The power of the situation. In Rossi, G., editor, *Psychology - Selected Papers*, pages 275–294. InTech. (Cited on page 107.)

[Boella and Torre, 2005] Boella, G. and Torre, L. (2005). Permission and authorization in policies for virtual communities of agents. In Moro, G., Bergamaschi, S., and Aberer, K., editors, *Agents and Peer-to-Peer Computing*, volume 3601 of *Lecture Notes in Computer Science*, pages 86–97. Springer Berlin Heidelberg. (Cited on pages 131 and 132.)

[Boella et al., 2008] Boella, G., Torre, L. V. D., and Verhagen, H. J. E. (2008). Introduction to the special issue on normative multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 17(1):1–10. (Cited on page 104.)

[Boella and van der Torre, 2005] Boella, G. and van der Torre, L. (2005). Role-based rights in artificial social systems. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IAT '05, pages 516–519, Washington, DC, USA. IEEE Computer Society. (Cited on pages 52 and 107.)

[Boissier, 2011] Boissier, O. (2011). From organisation oriented programming to multi-agent oriented programming. In *Proceedings of the 9th German conference on Multiagent system technologies*, MATES'11, pages 1–1, Berlin, Heidelberg. Springer-Verlag. (Cited on page 215.)

[Boissier et al., 2010] Boissier, O., Bordini, R. H., Dastani, M., Hübner, J. F., and Ricci, A. (2010). Multi-Agent Programming. (Cited on page 103.)

[Bonatti et al., 2010] Bonatti, P., De Coi, J. L., Olmedilla, D., and Sauro, L. (2010). A rule-based trust negotiation system. *IEEE Trans. on Knowl. and Data Eng.*, 22(11):1507–1520. (Cited on page 36.)

[Bonatti et al., 2008] Bonatti, P., Juri, L., Olmedilla, D., and Sauro, L. (2008). Policy-driven negotiations and explanations: Exploiting logic-programming for trust management, privacy &amp; security. *Logic Programming*, pages 779–784. (Cited on pages 74, 80 and 85.)

[Bonatti and Olmedilla, 2005] Bonatti, P. and Olmedilla, D. (2005). Driving and monitoring provisional trust negotiation with metapolicies. In *Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '05, pages 14–23, Washington, DC, USA. IEEE Computer Society. (Cited on pages 74 and 75.)

[Bonatti and Samarati, 2000] Bonatti, P. and Samarati, P. (2000). Regulating service access and information release on the web. In *Proceedings of the 7th ACM conference on Computer and communications security*, CCS '00, pages 134–143, New York, NY, USA. ACM. (Cited on page 36.)

[Bonatti and Samarati, 2002] Bonatti, P. A. and Samarati, P. (2002). Regulating Service Access and Information Release on the Web. *Journal of Computer Security*, 10(3):241 – 271. (Cited on pages 36, 90, 91, 92 and 112.)

[Bordini et al., 2007] Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley and Sons. (Cited on pages 213, 234, 235 and 243.)

[Borenstein and Freed, 1996] Borenstein, N. and Freed, N. (1996). Multipurpose Internet Mail Extensions. (Cited on page 121.)

[Bowser, 2013] Bowser, A. S. (2013). *To Conform or Not to Conform: An Examination of the Effects of Mock Jury Deliberation on Individual Jurors*. PhD thesis, East Tennessee State University. (Cited on pages 99, 102 and 282.)

[Braghin, 2011] Braghin, S. (2011). *Advanced languages and techniques for trust negotiation*. PhD thesis, University degli Studi dell'Insubria. (Cited on pages 62 and 72.)

[Bruneel et al., 2007] Bruneel, J., Spithoven, A., and Maesen, A. (2007). Building trust: a matter of proximity? In *Babson College Entrepreneurship Research Conference*. (Cited on page 145.)

[Burgemeestre et al., 2010] Burgemeestre, B., Hulstijn, J., and Tan, Y.-H. (2010). Towards an architecture for self-regulating agents: a case study in international trade. In *Proceedings of the 5th international conference on Coordination, organizations, institutions, and norms in agent systems*, COIN'09, pages 320–333, Berlin, Heidelberg. Springer-Verlag. (Cited on pages 105 and 107.)

[Burnett, 2011] Burnett, C. (2011). *Trust Assessment and Decision-Making in Dynamic Multi-Agent Systems*. PhD thesis, University of Aberdeen. (Cited on pages 31, 44 and 299.)

[Burnett et al., 2010] Burnett, C., Norman, T. J., and Sycara, K. (2010). Bootstrapping trust evaluations through stereotypes. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 241–248, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems. (Cited on page 25.)

[Burnett et al., 2013a] Burnett, C., Norman, T. J., and Sycara, K. (2013a). Stereotypical trust and bias in dynamic multiagent systems. *ACM Trans. Intell. Syst. Technol.*, 4(2):26:1–26:22. (Cited on page 31.)

[Burnett et al., 2013b] Burnett, C., Norman, T. J., and Sycara, K. (2013b). Stereotypical trust and bias in dynamic multiagent systems. *ACM Trans. Intell. Syst. Technol.*, 4(2):26:1–26:22. (Cited on pages 299 and 303.)

[Burnett et al., 2011] Burnett, C., Norman, T. J., and Sycara, K. P. (2011). Trust decision-making in multi-agent systems. In *IJCAI*, pages 115–120. (Cited on page 25.)

[Buyya et al., 2010] Buyya, R., Ranjan, R., and Calheiros, R. N. (2010). Intercloud: utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'10, pages 13–31, Berlin, Heidelberg. Springer-Verlag. (Cited on pages 300 and 304.)

[Camarinha-Matos et al., 2003] Camarinha-Matos, L., Castolo, O., and Rosas, J. (2003). A multi-agent based platform for virtual communities in elderly care. In *Proceedings of 2003 IEEE Conference on Emerging Technologies and Factory Automation.*, volume 2, pages 421–428. IEEE. (Cited on pages 117 and 135.)

Bibliography

[Campenni et al., 2009] Campenni, M., Andrighetto, G., Cecconi, F., and Conte, R. (2009). Normal = normative? the role of intelligent agents in norm innovation. *Mind &amp; Society*, 8(2):153–172. (Cited on pages 104 and 105.)

[Casimir et al., 2012] Casimir, G., Lee, K., and Loon, M. (2012). Knowledge sharing: influences of trust, commitment and cost. *Journal of Knowledge Management*, 16:740–753. (Cited on page 4.)

[Castelfranchi, 2000] Castelfranchi, C. (2000). Engineering social order. In Omicini, A., Tolksdorf, R., and Zambonelli, F., editors, *Engineering Societies in the Agents World*, volume 1972 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin Heidelberg. (Cited on pages 102 and 114.)

[Castelfranchi and Falcone, 1998] Castelfranchi, C. and Falcone, R. (1998). Principles of trust for mas: Cognitive anatomy, social importance, and quantification. In *Proceedings of the 3rd International Conference on Multi Agent Systems*, ICMAS '98, pages 72–79, Washington, DC, USA. IEEE Computer Society. (Cited on page 24.)

[Castelfranchi and Falcone, 2000a] Castelfranchi, C. and Falcone, R. (2000a). Trust and control: A dialectic link. *Applied Artificial Intelligence*, 14(8):799–823. (Cited on page 24.)

[Castelfranchi and Falcone, 2000b] Castelfranchi, C. and Falcone, R. (2000b). Trust is much more than subjective probability: Mental components and sources of trust. In *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 6 - Volume 6*, HICSS '00, pages 6008–, Washington, DC, USA. IEEE Computer Society. (Cited on pages 23, 24, 43 and 44.)

[Ceri et al., 1989] Ceri, S., Gottlob, G., and Tanca, L. (1989). What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. on Knowl. and Data Eng.*, 1(1):146–166. (Cited on pages 64 and 68.)

[Cervenka et al., 2006] Cervenka, R., Trencansky, I., and Calisti, M. (2006). Modeling social aspects of multi-agent systems: The aml approach. In Muller, J. P. and Zambonelli, F., editors, *Agent-Oriented Software Engineering VI*, volume 3950 of *Lecture Notes in Computer Science*, pages 28–39. Springer Berlin Heidelberg. (Cited on page 173.)

[Chen, 2011] Chen, L. (2011). *Analyzing and Developing Role-Based Access Control Models*. PhD thesis, Royal Holloway, University of London. (Cited on pages 50 and 52.)

[Chesbrough, 2012a] Chesbrough, H. (2012a). Open Innovation. *Research Technology Management*, 55:20–27. (Cited on page 245.)

[Chesbrough, 2012b] Chesbrough, H. (2012b). Open innovation: Where we've been and where we're going. *Research Technology Management*, 55:20–27. (Cited on page 245.)

[Chesbrough, 2006] Chesbrough, H. W. (2006). *Open Innovation: The New Imperative for Creating and Profiting from Technology*. Harvard Business Press. (Cited on pages 245 and 259.)

[Christianson and Harbison, 1997] Christianson, B. and Harbison, W. S. (1997). Why isn't trust transitive? In Lomas, M., editor, *Security Protocols*, volume 1189 of *Lecture Notes in Computer Science*, pages 171–176. Springer Berlin Heidelberg. (Cited on page 29.)

[Chu et al., 1997] Chu, Y.-H., Feigenbaum, J., LaMacchia, B., Resnick, P., and Strauss, M. (1997). Referee: trust management for web applications. *World Wide Web J.*, 2(3):127–139. (Cited on pages 63, 64, 80 and 84.)

[Cohen and Levesque, 1997] Cohen, P. R. and Levesque, H. J. (1997). Communicative actions for artificial agents. In Bradshaw, J. M., editor, *Software agents*, pages 419–436. MIT Press, Cambridge, MA, USA. (Cited on pages 133 and 141.)

[Coleman, 1990] Coleman, J. S. (1990). *Foundation of Social Theory*. Harvard University Press. (Cited on pages 87 and 111.)

[Collier, 2003] Collier, N. (2003). RePast : An Extensible Framework for Agent Simulation. (Cited on pages 263 and 283.)

[Conrad et al., 2012] Conrad, E., Misenar, S., and Feldman, J. (2012). *CISSP study guide*. Syngress. (Cited on page 57.)

[Conte et al., 2009] Conte, R., Andrighetto, G., and Campenni, M. (2009). The immergence of norms in agent worlds. In Aldewereld, H., Dignum, V., and Picard, G., editors, *Engineering Societies in the Agents World X*, volume 5881 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin Heidelberg. (Cited on page 105.)

[Conte et al., 1998] Conte, R., Gilbert, N., and Sichman, J. S. (1998). Mas and social simulation: A suitable commitment. In Sichman, J. S., Conte, R., and Gilbert, N., editors, *Multi-Agent Systems and Agent-Based Simulation*, volume 1534 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin Heidelberg. (Cited on page 101.)

[Cooke, 2009] Cooke, J. (2009). Resolving Open Innovation Contradictions. Technical Report DLM2009-01, Systematic Innovation. (Cited on pages 246, 259 and 260.)

[Corkill, 1983] Corkill, D. D. (1983). *A framework for organizational self-design in distributed problem solving networks*. PhD thesis, University of Massachusetts Amherst. AAI8310275. (Cited on page 103.)

[Cover, 2007] Cover, R. (2007). Extensible Access Control Markup Language (XACML). (Cited on pages 72, 80 and 85.)

[Criado, 2013] Criado, N. (2013). Using norms to control open multi-agent systems. *AI Communications*, 26(3). (Cited on pages 104 and 105.)

[da Silva and Demazeau, 2002] da Silva, J. L. T. and Demazeau, Y. (2002). Vowels co-ordination model. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, AAMAS '02, pages 1129–1136, New York, NY, USA. ACM. (Cited on pages 119 and 136.)

[Damianou et al., 2001] Damianou, N., Dulay, N., Lupu, E., and Sloman, M. (2001). The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, POLICY '01, pages 18–38, London, UK, UK. Springer-Verlag. (Cited on pages 65, 80 and 84.)

[De Coi and Olmedilla, 2008] De Coi, J. L. and Olmedilla, D. (2008). A review of trust management, security and privacy policy languages. In *SECRYPT 2008, Proceedings of the International Conference on Security and Cryptography, Porto, Portugal, July 26-29, 2008, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications*, pages 483–490. INSTICC Press. (Cited on pages 58, 68 and 76.)

[De Coi et al., 2008] De Coi, J. L., Olmedilla, D., Bonatti, P., and Sauro, L. (2008). Protune: A framework for semantic web policies. In Bizer, C. and Joshi, A., editors, *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008)*, volume 401. CEUR Workshop Proceedings. (Cited on pages 36 and 46.)

[DeTreville, 2002] DeTreville, J. (2002). Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, pages 105–, Washington, DC, USA. IEEE Computer Society. (Cited on pages 64, 80 and 84.)

[Deutsch, 2011] Deutsch, M. (2011). Cooperation and competition. In Coleman, P. T., editor, *Conflict, Interdependence, and Justice*, volume 11 of *Peace Psychology Book Series*, pages 23–40. Springer New York. (Cited on pages 18, 21, 24 and 41.)

[Deutsch and Gerard, 1955] Deutsch, M. and Gerard, H. B. (1955). A study of normative and informational social influences upon individual judgment. *The Journal of Abnormal and Social Psychology*, 51(3):629–636. (Cited on pages 18, 21, 24, 26 and 41.)

[Dignum, 2004] Dignum, M. (2004). *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Proefschrift Universiteit Utrecht. (Cited on pages 102, 103 and 104.)

[Dimmock et al., 2004] Dimmock, N., Belokosztolszki, A., Eyers, D., Bacon, J., and Moody, K. (2004). Using trust and risk in role-based access control policies. In *Proceedings of the ninth ACM symposium on Access control models and technologies*, SACMAT '04, pages 156–162, New York, NY, USA. ACM. (Cited on page 52.)

[Dubois, 2011] Dubois, D. (2011). *Self-organizing Methods and Models for Software Development*. PhD thesis, Politecnico di Milano, Dipartimento di Elettronica e Informazione. (Cited on pages 106, 110 and 114.)

[Dunn and Schweitzer, 2005] Dunn, J. and Schweitzer, M. (2005). Feeling and Believing: The Influence of Emotion on Trust. *Journal of personality and social psychology*, 88(5):736–748. (Cited on page 145.)

[Eisenegger, 2009] Eisenegger, M. (2009). Trust and reputation in the age of globalisation. In Klewes, J. and Wreschniok, R., editors, *Reputation Capital*, pages 11–22. Springer Berlin Heidelberg. (Cited on page 107.)

[El Houri, 2010] El Houri, M. (2010). *A Formal Model to express Dynamic Policies Access Control and Trust Negotiation a Distributed Environment*. PhD thesis, Université Toulouse III Paul Sabatier (UPS). (Cited on page 49.)

[Esparcia and Argente, 2010] Esparcia, S. and Argente, E. (2010). A functional taxonomy for artifacts. In Corchado, E., Gra Romay, M., and Manhaes Savio, A., editors, *Hybrid Artificial Intelligence Systems*, volume 6077 of *Lecture Notes in Computer Science*, pages 159–167. Springer Berlin Heidelberg. (Cited on page 214.)

[Esteva et al., 2002] Esteva, M., de la Cruz, D., and Sierra, C. (2002). Islander: an electronic institutions editor. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, AAMAS '02, pages 1045–1052, New York, NY, USA. ACM. (Cited on page 103.)

[Eymard-Duvernay et al., 2003] Eymard-Duvernay, F., Favereau, O., Orléan, A., Salais, R., and Thévenot, L. (2003). Values, Coordination and Rationality: The Economy of Conventions or the

Time of Reunification in the Economic, Social and Political Sciences. *Conventions et Institutions: approfondissements theoriques et contributions au debat politique*, 11. (Cited on page 19.)

[Falcone and Castelfranchi, 2001] Falcone, R. and Castelfranchi, C. (2001). Social trust: a cognitive approach. In Castelfranchi, C. and Tan, Y.-H., editors, *Trust and deception in virtual societies*, pages 55–90. Kluwer Academic Publishers, Norwell, MA, USA. (Cited on pages 23, 24, 25, 26, 43, 44, 143 and 164.)

[Fang et al., 2012] Fang, H., Zhang, J., Sensoy, M., and Thalmann, N. M. (2012). A generalized stereotypical trust model. In *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM '12, pages 698–705, Washington, DC, USA. IEEE Computer Society. (Cited on page 31.)

[Farley and Clark, 1954] Farley, B. G. and Clark, W. (1954). Simulation of self-organizing systems by digital computer. *Information Theory, Transactions of the IRE Professional Group on*, 4(4):76–84. (Cited on page 106.)

[Farrell, 2009] Farrell, H. (2009). *Distrust*, chapter Trust, Distrust, and Power, pages 85–105. Russell Sage Foundation. (Cited on page 27.)

[Ferber et al., 2005] Ferber, J., Michel, F., and Baez, J. (2005). Agre: Integrating environments with organizations. In Weyns, D., Dyke Parunak, H., and Michel, F., editors, *Environments for Multi-Agent Systems*, volume 3374 of *Lecture Notes in Computer Science*, pages 48–56. Springer Berlin Heidelberg. (Cited on page 103.)

[Fernandez-Gago et al., 2007] Fernandez-Gago, M. C., Roman, R., and Lopez, J. (2007). A Survey on the Applicability of Trust Management Systems forWireless Sensor Networks. In *Third International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing (SecPerU 2007)*, pages 25–30. Ieee. (Cited on page 62.)

[Ferraiolo and Kuhn, 2009] Ferraiolo, D. and Kuhn, D. (2009). Role-based access controls. *arXiv preprint arXiv:0903.2171*, pages 554 – 563. (Cited on page 52.)

[Ferraiolo et al., 2001] Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., and Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274. (Cited on page 52.)

[Finin et al., 2008] Finin, T., Joshi, A., Kagal, L., Niu, J., Sandhu, R., Winsborough, W., and Thuraisingham, B. (2008). Rowlbac: representing role based access control in owl. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 73–82, New York, NY, USA. ACM. (Cited on pages 52 and 78.)

[FIPA, 2002] FIPA (2002). FIPA SL Content Language Specification. (Cited on pages 133 and 141.)

[Firdhous et al., 2012] Firdhous, M., Ghazali, O., and Hassan, S. (2012). Trust management in cloud computing: A critical review. *International Journal on Advances in ICT for Emerging Regions (ICTer)*, 4(2). (Cited on page 62.)

[Galinović, 2010] Galinović, A. (2010). Automated trust negotiation models. In *Proceedings of the 33rd International Convention MIPRO*, pages 1197–1202. (Cited on page 56.)

[Gambetta, 2000] Gambetta, D. (2000). Can We Trust Trust? In Gambetta, D., editor, *Trust: Making and Breaking Cooperative Relations*, chapter 13, pages 213–237. Department of Sociology, University of Oxford. (Cited on pages 22 and 43.)

[Genovese, 2012] Genovese, V. (2012). *Modalities in Access Control: Logics, Proof-theory and Applications*. PhD thesis, University of Luxembourg and University of Torino. (Cited on pages 48 and 51.)

[Gerck, 2000] Gerck, E. (2000). Overview of certification systems: X.509, ca, pgp and skip. Technical report, Meta-Certificate Group. (Cited on pages 58 and 83.)

[Golbeck, 2005] Golbeck, J. (2005). *Computing and applying trust in web-based social networks*. PhD thesis, University of Maryland, College Park. (Cited on page 30.)

[Golbeck and Hendler, 2006] Golbeck, J. and Hendler, J. (2006). Inferring binary trust relationships in web-based social networks. *ACM Trans. Internet Technol.*, 6(4):497–529. (Cited on page 25.)

[Governatori and Rotolo, 2009] Governatori, G. and Rotolo, A. (2009). How do agents comply with norms? In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 03*, WI-IAT '09, pages 488–491, Washington, DC, USA. IEEE Computer Society. (Cited on page 104.)

[Grandison and Sloman, 2000] Grandison, T. and Sloman, M. (2000). A survey of trust in internet applications. *Commun. Surveys Tuts.*, 3(4):2–16. (Cited on pages 20, 27 and 30.)

[Grandison and Sloman, 2003] Grandison, T. and Sloman, M. (2003). Trust management tools for internet applications. In *Proceedings of the 1st international conference on Trust management*, iTrust'03, pages 91–107, Berlin, Heidelberg. Springer-Verlag. (Cited on pages 34, 45 and 55.)

[Grandison, 2003] Grandison, T. W. A. (2003). *Trust Management for Internet Applications*. PhD thesis, Imperial College of Science, Technology and Medicine, University of London. (Cited on pages 23, 25, 27, 28, 29, 30, 38, 43, 55, 61, 62, 63, 64, 65, 80, 82, 84 and 175.)

[Gray, 2006] Gray, E. (2006). *A trust-based reputation management system*. PhD thesis, Dept. of Computer Science, Trinity College Dublin. (Cited on pages 27 and 62.)

[Grizard et al., 2007] Grizard, A., Vercouter, L., Stratulat, T., and Muller, G. (2007). A peer-to-peer normative system to achieve social order. In Noriega, P., Vazquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., and Matson, E., editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, volume 4386 of *Lecture Notes in Computer Science*, pages 274–289. Springer Berlin Heidelberg. (Cited on pages 102 and 107.)

[Gupta and Kim, 2004] Gupta, S. and Kim, H. (2004). Virtual community: concepts, implications, and future research directions. In *Proceedings of the 10th American Conference on Information System*, pages 2679–2687. (Cited on pages 117 and 135.)

[Hardin, 1982] Hardin, R. (1982). *Trust*. Polity. (Cited on page 27.)

[Harrenstein et al., 2002] Harrenstein, P., van der Hoek, W., Meyer, J.-J. C., and Witteveen, C. (2002). On modal logic interpretations of games. In van Harmelen, F., editor, *ECAI*, pages 28–32. IOS Press. (Cited on page 195.)

[Harris, 1977] Harris, E. E. (1977). Kants Refutation of the Ontological Proof. *Philosophy*, 52:90. (Cited on page 18.)

[Haynes et al., 2013] Haynes, C., Miles, S., and Luck, M. (2013). Monitoring the Impact of Norms upon Organisational Performance: a Simulation Approach. In *The 15th International Workshop on Coordination, Organisations, Institutions and Norms.* (Cited on page 104.)

[Herzberg et al., 2000] Herzberg, A., Mass, Y., Michaeli, J., Ravid, Y., and Naor, D. (2000). Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 2–, Washington, DC, USA. IEEE Computer Society. (Cited on pages 67, 80 and 85.)

[Herzig and Lorini, 2010] Herzig, A. and Lorini, E. (2010). A logic of trust and reputation. *Logic Journal of IGPL*. (Cited on pages 28, 33 and 45.)

[Herzig et al., 2008] Herzig, A., Lorini, E., H, J. F., Ben-naim, J., Boissier, O., Castelfranchi, C., Longin, D., Perrussel, L., and Vercouter, L. (2008). Prolegomena for a logic of trust and reputation. In Boella, G., Pigozzi, G., Singh, M. P., and Verhagen, H., editors, *Proceedings of the Third International Workshop on Normative Multiagent Systems (NorMAS 2008)*, pages 143–157. (Cited on page 28.)

[Horling and Lesser, 2004] Horling, B. and Lesser, V. (2004). A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*. (Cited on page 103.)

[Hosmer, 1995] Hosmer, L. (1995). Trust: The connecting link between organizational theory and philosophical ethics. *Academy of Management Review*, 20(2):379–403. (Cited on page 19.)

[Hübner et al., 2009a] Hübner, J., Bordini, R., and Picard, G. (2009a). Using Jason and MOISE+ to develop a team of cowboys. *Programming Multi-Agent Systems*. (Cited on page 215.)

[Hübner et al., 2009b] Hübner, J., Lorini, E., Herzig, A., and Vercouter, L. (2009b). From cognitive trust theories to computational trust. In *Proceedings of The 12th Workshop on Trust in Agent Societies*. (Cited on pages 33 and 45.)

[Hubner, 2011] Hubner, J. F. (2011). *Moise specifications*. (Cited on pages 125, 126, 140 and 215.)

[Hübner et al., 2010] Hübner, J. F., Boissier, O., Kitio, R., and Ricci, A. (2010). Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400. (Cited on pages 125, 128, 130 and 140.)

[Hubner et al., 2002] Hubner, J. F., Sichman, J. S., and Boissier, O. (2002). MOISE+ : Towards a structural , functional , and deontic model for MAS organization. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1–2. ACM Press. (Cited on pages 103, 104, 125, 140 and 215.)

[Hübner et al., 2004] Hübner, J. F., Sichman, J. S., and Boissier, O. (2004). Using the moise+ for a cooperative framework of mas reorganisation. In Bazzan, A. L. C. and Labidi, S., editors, *Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence, São Luis, Maranhão, Brazil, September 29 - October 1, 2004, Proceedings*, volume 3171 of *Lecture Notes in Computer Science*, pages 506–515. Springer. (Cited on pages 108 and 114.)

[Hubner et al., 2007] Hubner, J. F., Sichman, J. S., and Boissier, O. (2007). Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *Int. J. Agent-Oriented Softw. Eng.*, 1(3/4):370–395. (Cited on page 103.)

[Huizingh, 2011] Huizingh, E. K. R. E. (2011). Open innovation: State of the art and future perspectives. *Technovation*, 31:2–9. (Cited on pages 246 and 259.)

[Humenn, 2003] Humenn, P. (2003). The formal semantics of XACML. Technical report, Syracuse University. (Cited on pages 72 and 85.)

[Huynh et al., 2006] Huynh, T. D., Jennings, N. R., and Shadbolt, N. R. (2006). An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 13(2):119–154. (Cited on pages 27, 28, 32 and 44.)

[Jennings and Wooldridge, 1998] Jennings, N. R. and Wooldridge, M. (1998). Applications of intelligent agents. In Jennings, N. R. and Wooldridge, M. J., editors, *Agent technology*, pages 3–28. Springer-Verlag New York, Inc., Secaucus, NJ, USA. (Cited on pages 101, 102 and 114.)

[Jongh, 2013] Jongh, M. D. (2013). *Group dynamics in the Citizens' Assembly on Electoral Reform.* PhD thesis, Utrecht University. (Cited on page 98.)

[Jonker and Treur, 2001] Jonker, C. M. and Treur, J. (2001). An agent architecture for multi-attribute negotiation. In *Proceedings of the 17th International Joint Conference on AI, IJCAI'01, 2001*, pages 1195–1201. Morgan Kaufman. (Cited on page 28.)

[Jøsang, 2007] Jøsang, A. (2007). Trust and reputation systems. In Aldini, A. and Gorrieri, R., editors, *Foundations of security analysis and design IV*, pages 209–245. Springer-Verlag, Berlin, Heidelberg. (Cited on pages 22, 30, 55 and 62.)

[Jøsang and Ismail, 2002] Jøsang, A. and Ismail, R. (2002). The Beta Reputation System. In *Proceedings of the 15th Bled Electronic Commerce Conference*, volume 160, pages 324–337. (Cited on pages 31, 32 and 44.)

[Jøsang et al., 2007] Jøsang, A., Ismail, R., and Boyd, C. (2007). A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644. (Cited on pages 27, 30, 55, 62, 82 and 164.)

[Josang and Presti, 2004] Josang, A. and Presti, S. (2004). Analysing the relationship between risk and trust. In Jensen, C., Poslad, S., and Dimitrakos, T., editors, *Trust Management*, volume 2995 of *Lecture Notes in Computer Science*, pages 135–145. Springer Berlin Heidelberg. (Cited on page 22.)

[Kagal et al., 2003] Kagal, L., Finin, T., and Joshi, A. (2003). A policy based approach to security for the semantic web. In Fensel, D., Sycara, K., and Mylopoulos, J., editors, *The Semantic Web - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 402–418. Springer Berlin Heidelberg. (Cited on pages 34 and 66.)

[Kalam et al., 2003] Kalam, A. A. E., Benferhat, S., Miège, A., Baida, R. E., Cuppens, F., Saurel, C., Balbiani, P., Deswarte, Y., and Trouessin, G. (2003). Organization based access control. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 120–, Washington, DC, USA. IEEE Computer Society. (Cited on page 52.)

[Kalam and Deswarte, 2006] Kalam, A. A. E. and Deswarte, Y. (2006). Multi-OrBAC : a New Access Control Model for Distributed, Heterogeneous and Collaborative Systems. In *8th International Symposium on System and Information Security (SSI'2006), Sao Paulo (Brésil), 8-10 Novembre 2006*. (Cited on page 52.)

[Kant, 1788] Kant, I. (1788). *Kritik der praktischen Vernunft*, volume 56. Gutenberg Project. (Cited on pages 18 and 41.)

[Kaur, 2011] Kaur, P. (2011). Supporting users trust decisions on inter-enterprise collaborations. Master's thesis, Aalto University, School of Science. (Cited on pages 24 and 26.)

[Khodyakov, 2007] Khodyakov, D. (2007). Trust as a Process: A Three-Dimensional Approach. *Sociology*, 41(1):115–132. (Cited on pages 19 and 26.)

[Kim et al., 2009] Kim, P. H., Dirks, K. T., and Cooper, C. D. (2009). The Repair of Trust: A Dynamic Bilateral Perspective and Multilevel Conceptualization. *Academy of Management Review*, 34(3):401–422. (Cited on page 25.)

[Koehler and Giblin, 2003] Koehler, J. and Giblin, C. (2003). On autonomic computing architectures. Technical report, IBM Research, Zurich. (Cited on pages 106 and 114.)

[Koster, 2012] Koster, A. (2012). *Trust Alignment and Adaptation : Two Approaches for Talking about Trust in Multi-Agent Systems*. PhD thesis, Universitat Autonoma de Barcelona. (Cited on pages 5 and 33.)

[Krukow et al., 2008] Krukow, K., Nielsen, M., and Sassone, V. (2008). Trust models in ubiquitous computing. *Philosophical transactions of the Royal Society*, 366(1881):3781–3793. (Cited on pages 30 and 62.)

[Krupa, 2012] Krupa, Y. (2012). *PrivaCIAS: Privacy as Contextual Integrity in Decentralized Multi-Agent Systems*. PhD thesis, Ecole Nationale Supérieure des Mines de Saint-Etienne. (Cited on page 33.)

[Krupa et al., 2009] Krupa, Y., Vercouter, L., Hubner, J. F., and Herzig, A. (2009). Trust based evaluation of wikipedia's contributors. In Aldewereld, H., Dignum, V., and Picard, G., editors, *Engineering Societies in the Agents World X*, volume 5881 of *Lecture Notes in Computer Science*, pages 148–161. Springer Berlin Heidelberg. (Cited on page 33.)

[Lacey and Hexmoor, 2003] Lacey, N. and Hexmoor, H. (2003). Norm Adaptation and Revision in a Multi-Agent System. In *The Florida AI Research Society Conference - FLAIRS*, pages 27–31. (Cited on page 109.)

[Lamsal, 2001] Lamsal, P. (2001). Understanding trust and security. Technical report, Department of Computer Science University of Helsinki, Finland. (Cited on page 22.)

[Latané, 1981] Latané, B. (1981). The psychology of social impact. *American Psychologist*, 36:343–356. (Cited on pages 100 and 200.)

[Lee, 2008] Lee, A. J. (2008). *Towards Practical and Secure Decentraliz Attribute-Based Authorisation Systems*. PhD thesis, University of Illinois. (Cited on pages 52, 61 and 149.)

[Lee et al., 2009] Lee, A. J., Winslett, M., and Perano, K. J. (2009). Trustbuilder2: A reconfigurable framework for trust negotiation. In Ferrari, E., Li, N., Bertino, E., and Karabulut, Y., editors, *Trust Management III*, volume 300 of *IFIP Advances in Information and Communication Technology*, pages 176–195. Springer Berlin Heidelberg. (Cited on pages 56 and 70.)

[Lee and Yu, 2009] Lee, A. J. and Yu, T. (2009). Towards a dynamic and composable model of trust. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 217–226, New York, NY, USA. ACM. (Cited on page 26.)

[Lewicki et al., 2006] Lewicki, R. J., Tomlinson, E. C., and Gillespie, N. (2006). Models of Interpersonal Trust Development: Theoretical Approaches, Empirical Evidence, and Future Directions. *Journal of Management*, 32(6):991–1022. (Cited on pages 24 and 26.)

[Li et al., 2002] Li, N., Mitchell, J. C., and Winsborough, W. H. (2002). Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, pages 114–, Washington, DC, USA. IEEE Computer Society. (Cited on pages 52, 68, 80 and 85.)

[Li et al., 2009] Li, N., Wang, Q., Qardaji, W., Bertino, E., Rao, P., Lobo, J., and Lin, D. (2009). Access control policy combining: theory meets practice. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 135–144, New York, NY, USA. ACM. (Cited on pages 73 and 196.)

[Li et al., 2008] Li, N., Wang, Q., Rao, P., Lin, D., Bertino, E., and Lobo, J. (2008). A Formal Language for Specifying Policy Combining Algorithms in Access Control. Technical Report 2008-9, CERIAS. (Cited on pages 88 and 112.)

[Lian et al., 2007] Lian, Q., Zhang, Z., Yang, M., Zhao, B. Y., Dai, Y., and Li, X. (2007). An empirical study of collusion behavior in the maze p2p file-sharing system. In *Proceedings of the 27th International Conference on Distributed Computing Systems*, ICDCS '07, pages 56–, Washington, DC, USA. IEEE Computer Society. (Cited on page 189.)

[Linn, 2000] Linn, J. (2000). Trust models and management in public-key infrastructures. Technical report, RSA Laboratories. (Cited on pages 56, 146 and 168.)

[Liu, 2011] Liu, W. W. (2011). *Trust Management And Accountability for Internet Security*. PhD thesis, Department of Computer Science, Florida State University. (Cited on pages 28 and 62.)

[Liu et al., 2009] Liu, X., Datta, A., Rzadca, K., and Lim, E.-P. (2009). Stereotrust: a group based personalized trust model. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 7–16, New York, NY, USA. ACM. (Cited on page 31.)

[Liu et al., 2008] Liu, Y., Yang, Y., and Sun, Y. (2008). Detection of collusion behaviors in online reputation systems. In *Signals, Systems and Computers, 2008 42nd Asilomar Conference on*, pages 1368–1372. (Cited on page 189.)

[López y López and Luck, 2004] López y López, F. and Luck, M. (2004). A model of normative multi-agent systems and dynamic relationships. In Lindemann, G., Moldt, D., and Paolucci, M., editors, *Regulated Agent-Based Social Systems*, volume 2934 of *Lecture Notes in Computer Science*, pages 259–280. Springer Berlin Heidelberg. (Cited on pages 103 and 107.)

[Lovejoy, 1968] Lovejoy, A. O. (1968). Kant and evolution. In *Forerunners of Darwin (1745-1859)*, pages 173–206. JHU Press. (Cited on page 18.)

[Luhmann, 1990] Luhmann, N. (1990). Familiarity, confidence, trust: Problems and alternatives. In *Trust: Making and breaking cooperative relations*, pages 15–35. Basil Blackwell. (Cited on pages 19, 22 and 43.)

[Mahoney et al., 1994] Mahoney, J., Huff, A., and Huff, J. (1994). Toward a new social contract theory in organizatioon science. *Journal of Management Inquiry*. (Cited on page 19.)

[Manchala, 1998] Manchala, D. (1998). Trust metrics, models and protocols for electronic commerce transactions. In *Proceedings of 18th International Conference on Distributed Computing Systems (Cat. No.98CB36183)*, pages 312–321. IEEE Comput. Soc. (Cited on pages 30, 31 and 44.)

[Mangematin, 1998] Mangematin, V. (1998). La confiance : un mode de coordination dont l'utilisation dépend de ses conditions de production. In Harrisson, D., Mangematin, V., and Thuderoz, C., editors, *Confiance et entreprise*, pages 1–21. Gaetan Morin. (Cited on page 26.)

[Marsh, 1994] Marsh, S. (1994). *Formalising trust as a computational concept*. PhD thesis, Department of Computing Science and Mathematics, University of Stirling. (Cited on pages 6, 22, 28, 30, 38 and 44.)

[Maslow, 1943] Maslow, A. H. (1943). A Theory of Human Motivation. *Psychological Review*, 50:370–396. (Cited on page 17.)

[Mazzoleni et al., 2006] Mazzoleni, P., Bertino, E., Crispo, B., and Sivasubramanian, S. (2006). Xacml policy integration algorithms: not to be confused with xacml policy combination algorithms! In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, SACMAT '06, pages 219–227, New York, NY, USA. ACM. (Cited on pages 91, 94 and 112.)

[Mcallister, 1995] Mcallister, D. J. (1995). Affect- and Cognition-Based Trust as Foundations for Interpersonal Cooperation in Organizations. *The Academy of Management Journal*, 38(1):24–59. (Cited on page 19.)

[Mcallister, 1997] Mcallister, D. J. (1997). The Second Face of Trust: reflections on the Dark Side of Interpersonal Trust in Organizations. *Research on Negotiation in Organizations*, 6:87–111. (Cited on page 19.)

[Mcknight and Chervany, 1996] Mcknight, D. H. and Chervany, N. L. (1996). The Meanings of trust. Technical Report 612, University of Minnesota. (Cited on pages 19, 23 and 43.)

[Meneguzzi, 2009] Meneguzzi, F. (2009). *Extending agent languages for multiagent domains*. PhD thesis, School of Physical Sciences and Engineering Department of Computer Science, University of London. (Cited on page 212.)

[Merida-Campos and Willmott, 2007] Merida-Campos, C. and Willmott, S. (2007). Stable collaboration patterns of self-interested agents in iterative request for proposal coalition formation environments. *International Transactions on Systems Science and Applications*, 2(1):40–45. (Cited on page 274.)

[Meyer et al., 1998] Meyer, J.-J., Wieringa, R., and Dignum, F. (1998). The role of deontic logic in the specification of information systems. In Chomicki, J. and Saake, G., editors, *Logics for Databases and Information Systems*, volume 436 of *The Springer International Series in Engineering and Computer Science*, pages 71–115. Springer US. (Cited on page 104.)

[Meyer and Wieringa, 1993] Meyer, J.-J. C. and Wieringa, R. J., editors (1993). *Deontic logic in computer science: normative system specification*. John Wiley and Sons Ltd., Chichester, UK. (Cited on page 104.)

[Misztal, 1996] Misztal, B. (1996). *Trust in Modern Societies: The Search for the Bases of Social Order*. Wiley. (Cited on page 19.)

[Moscovici, 1969] Moscovici, S. (1969). Studies in Social Influence. *Journal of Experimental Social Psychology*, 16:270–282. (Cited on pages 99, 113 and 282.)

[Nejdl et al., 2004] Nejdl, W., Olmedilla, D., and Winslett, M. (2004). Peertrust: Automated trust negotiation for peers on the semantic web. In Jonker, W. and Petković, M., editors, *Secure Data Management*, volume 3178 of *Lecture Notes in Computer Science*, pages 118–132. Springer Berlin Heidelberg. (Cited on pages 36 and 56.)

[Nemeth, 1986] Nemeth, C. (1986). Differential contributions of majority and minority influence. *Psychological review*. (Cited on pages 100 and 113.)

[Nepal et al., 2011] Nepal, S., Sherchan, W., and Paris, C. (2011). Strust: A trust model for social networks. In *Proceedings of the 2011 IEEE 10th International Conference on Trust, Security and*

Privacy in Computing and Communications, TRUSTCOM '11, pages 841–846, Washington, DC, USA. IEEE Computer Society. (Cited on page 145.)

[Nyanchama and Osborn, 1999] Nyanchama, M. and Osborn, S. (1999). The role graph model and conflict of interest. *ACM Transactions on Information and System Security*, 2(1):3–33. (Cited on page 52.)

[Omicini et al., 2008] Omicini, A., Ricci, A., and Viroli, M. (2008). Artifacts in the A&amp;A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456. (Cited on page 214.)

[Orléan, 2000] Orléan, A. (2000). La théorie économique de la confiance et ses limites. *les Cahiers de Socio-Économie*, pages 59–77. (Cited on page 19.)

[Osterloh and Rota, 2005] Osterloh, M. and Rota, S. (2005). Trust and community in open source software production. CREMA Working Paper Series 2005-11, Center for Research in Economics, Management and the Arts (CREMA). (Cited on page 117.)

[Paraiso et al., 2012] Paraiso, F., Haderer, N., Merle, P., Rouvoy, R., and Seinturier, L. (2012). A federated multi-cloud paas infrastructure. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, pages 392–399, Washington, DC, USA. IEEE Computer Society. (Cited on pages 300 and 304.)

[Parikh, 1985] Parikh, R. (1985). The logic of games and its applications. In *Selected papers of the international conference on "foundations of computation theory" on Topics in the theory of computation*, pages 111–139, New York, NY, USA. Elsevier North-Holland, Inc. (Cited on page 195.)

[Parsons and Wooldridge, 2002] Parsons, S. and Wooldridge, M. (2002). Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, pages 1–14. (Cited on pages 61 and 190.)

[Pasquier et al., 2006] Pasquier, P., Flores, R. A., and Chaib-draa, B. (2006). An ontology of social control tools. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, AAMAS '06, pages 1369–1371, New York, NY, USA. ACM. (Cited on page 107.)

[Pearlman et al., 2002] Pearlman, L., Welch, V., Foster, I., Kesselman, C., and Tuecke, S. (2002). A community authorization service for group collaboration. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, POLICY '02, pages 50–, Washington, DC, USA. IEEE Computer Society. (Cited on pages 6, 13 and 247.)

[Pitt and Bellifemine, 1999] Pitt, J. and Bellifemine, F. (1999). A Protocol-Based Semantics for FIPA'97 ACL and its Implementation in JADE. In *AI*IA*, pages 1–10. The Imperial College of Science Technology &amp; Medicine and CSELT. (Cited on pages 133 and 134.)

[Plato, 1994] Plato (1994). *The Republic*. The Gutenberg Project. (Cited on page 17.)

[Preece, 2001] Preece, J. (2001). Sociability and usability in online communities: Determining and measuring success. *Behaviour &amp; Information Technology*, 20(5):347–356. (Cited on page 3.)

[Preece, 2004] Preece, J. (2004). Online communities: researching sociability and usability in hard to reach populations. *Australasian J. of Inf. Systems*, 11(2). (Cited on page 3.)

[Prohic, 2005] Prohic, N. (2005). Public Key Infrastructures – PGP vs. X. 509. In *INFOTECH Seminar Advanced Communication Services (ACS)*. (Cited on pages 58 and 83.)

[Pynadath et al., 2000] Pynadath, D. V., Tambe, M., Chauvat, N., and Cavedon, L. (2000). Toward team-oriented programming. In Jennings, N. R. and Lespérance, Y., editors, *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, volume 1757 of *Lecture Notes in Computer Science*, pages 233–247. Springer Berlin Heidelberg. (Cited on page 103.)

[Rakotonirainy et al., 2009] Rakotonirainy, A., Loke, S., and Obst, P. (2009). Social awareness concepts to support social computing. In *International Conference onComputational Science and Engineering (CSE'09)*, volume 4, pages 223–228. (Cited on pages 102 and 173.)

[Ramchurn, 2004] Ramchurn, S. D. (2004). *Multi-agent negotiation using trust and persuasion*. PhD thesis, Faculty of Engineering and Applied Science, School of Electronics and Computer Science, University of Southampton. (Cited on page 25.)

[Ramchurn et al., 2004a] Ramchurn, S. D., Huynh, D., and Jennings, N. R. (2004a). Trust in multi-agent systems. *Knowl. Eng. Rev.*, 19(1):1–25. (Cited on pages 25, 36 and 37.)

[Ramchurn et al., 2004b] Ramchurn, S. D., Huynh, T. D., and Jennings, N. R. (2004b). Trust in multi-agent systems. *The Knowledge Engineering Review*, 19(1):1–25. (Cited on pages 36 and 38.)

[Rao and Georgeff, 1995] Rao, A. S. and Georgeff, M. P. (1995). Bdi agents: From theory to practice. In Lesser, V. R. and Gasser, L., editors, *Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA*, pages 312–319. The MIT Press. (Cited on page 211.)

[Rao et al., 2009] Rao, P., Lin, D., Bertino, E., Li, N., and Lobo, J. (2009). An algebra for fine-grained integration of xacml policies. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 63–72, New York, NY, USA. ACM. (Cited on pages 88, 90 and 112.)

[Rao et al., 2011] Rao, P., Lin, D., Bertino, E., Li, N., and Lobo, J. (2011). Fine-grained integration of access control policies. *Computers &amp; Security*, 30(2-3):91–107. (Cited on pages 91, 94 and 112.)

[Reh, 2008] Reh, M. (2008). *Multiagent Trust Modeling for Open Network Environments*. PhD thesis, Czech Technical University in Prague Faculty of Electrical Engineering. (Cited on page 23.)

[Rehm and Endrass, 2009] Rehm, M. and Endrass, B. (2009). Rapid prototyping of social group dynamics in multiagent systems. *AI &amp; Society*, 24(1):13–23. (Cited on page 100.)

[Rempel et al., 1985] Rempel, J. K., Holmes, J. G., and Zanna, M. P. (1985). Trust in close relationships. *Journal of Personality and Social Psychology*, 49(1):95–112. (Cited on pages 18 and 41.)

[Renzl, 2008] Renzl, B. (2008). Trust in management and knowledge sharing: The mediating effects of fear and knowledge documentation. *Omega*, 36(2):206 – 220. Special Issue on Knowledge Management and Organizational Learning. (Cited on page 4.)

[Rheingold, 1993] Rheingold, H. (1993). The Virtual Community. *Addison-Wesley Publishing Co.* (Cited on page 3.)

[Ricci et al., 2011] Ricci, A., Piunti, M., and Viroli, M. (2011). Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192. (Cited on page 214.)

[Rotter, 1967] Rotter, J. B. (1967). A new scale for the measurement of interpersonal trust. *Journal of Personality*, 35(4):651–665. (Cited on pages 18, 24, 26 and 41.)

[Ruohomaa and Kutvonen, 2005] Ruohomaa, S. and Kutvonen, L. (2005). Trust management survey. In Herrmann, P., Issarny, V., and Shiu, S., editors, *Trust Management*, volume 3477 of *Lecture Notes in Computer Science*, pages 77–92. Springer Berlin Heidelberg. (Cited on pages 30, 38 and 62.)

[Rupert et al., 2007] Rupert, M., Hassas, S., Li, C., and Sherwood, J. (2007). Simulation of Online Communities Using MAS Social and Spatial Organisations. *World Academy of Science*, pages 355–360. (Cited on pages 117 and 135.)

[Russell and Norvig, 2010] Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall,. (Cited on page 185.)

[Ryutov et al., 2005] Ryutov, T., Zhou, L., Neuman, C., Leithead, T., and Seamons, K. E. (2005). Adaptive trust negotiation and access control. In *Proceedings of the tenth ACM symposium on Access control models and technologies*, SACMAT '05, pages 139–146, New York, NY, USA. ACM. (Cited on pages 56, 60, 61, 74, 80 and 85.)

[Saadi et al., 2011] Saadi, R., Rahaman, M., Issarny, V., and Toninelli, A. (2011). Composing trust models towards interoperable trust management. *Trust Management V*, 358:51–66. (Cited on pages 26, 38 and 62.)

[Sabater and Sierra, 2001] Sabater, J. and Sierra, C. (2001). Regret: reputation in gregarious societies. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 194–195, New York, NY, USA. ACM. (Cited on page 31.)

[Sabater and Sierra, 2005] Sabater, J. and Sierra, C. (2005). Review on Computational Trust and Reputation Models. *Artificial Intelligence Review*, 24(1):33–60. (Cited on pages 36 and 38.)

[Salais, 1989] Salais, R. (1989). L'analyse économique des conventions du travail. *Revue économique*, 40(2):199. (Cited on page 19.)

[Samarati and Vimercati, 2001] Samarati, P. and Vimercati, S. D. C. d. (2001). Access control: Policies, models, and mechanisms. In *Revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design: Tutorial Lectures*, FOSAD '00, pages 137–196, London, UK, UK. Springer-Verlag. (Cited on pages 57 and 83.)

[Sandhu et al., 1996] Sandhu, R., Coyne, E., Feinstein, H., and Youman, C. (1996). Role-Based Access Control Models. *Computer*, 29(2):38–47. (Cited on pages 51 and 52.)

[Sandhu et al., 2000] Sandhu, R., Ferraiolo, D., and Kuhn, R. (2000). The nist model for role-based access control: towards a unified standard. In *Proceedings of the fifth ACM workshop on Role-based access control*, RBAC '00, pages 47–63, New York, NY, USA. ACM. (Cited on page 52.)

[Sandhu, 1993] Sandhu, R. S. (1993). Lattice-based access control models. *Computer*, 26(11):9–19. (Cited on page 50.)

[Sawyer, 2003] Sawyer, R. K. (2003). Artificial Societies: Multiagent Systems and the Micro-Macro Link in Sociological Theory. *Sociological Methods Research*, 31(3):325–363. (Cited on page 108.)

[Seamons et al., 2002] Seamons, K., Winslett, M., Yu, T., Smith, B., Child, E., Jacobson, J., Mills, H., and Yu, L. (2002). Requirements for policy languages for trust negotiation. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, POLICY '02, pages 68–, Washington, DC, USA. IEEE Computer Society. (Cited on pages 36, 59, 76, 175 and 191.)

[Sensoy et al., 2010] Sensoy, M., Norman, T. J., Vasconcelos, W. W., and Sycara, K. (2010). Owl-polar: Semantic policies for agent reasoning. In Patel-Schneider, P., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J., Horrocks, I., and Glimm, B., editors, *The Semantic Web ISWC 2010*, volume 6496 of *Lecture Notes in Computer Science*, pages 679–695. Springer Berlin Heidelberg. (Cited on page 78.)

[Serban and Minsky, 2009] Serban, C. and Minsky, N. (2009). In vivo evolution of policies that govern a distributed system. In *Proceedings of the 10th IEEE international conference on Policies for distributed systems and networks*, POLICY'09, pages 134–141, Piscataway, NJ, USA. IEEE Press. (Cited on pages 299 and 303.)

[Shapiro, 1998] Shapiro, S. (1998). Places and spaces: The historical interaction of technology, home, and privacy. *The Information Society*. (Cited on pages 19 and 24.)

[Sherif, 1936] Sherif, M. (1936). The psychology of social norms. *Journal for the Theory of Social Behaviour*, 41:53–76. (Cited on pages 96, 97 and 113.)

[Sherif, 1937] Sherif, M. (1937). An experimental approach to the study of attitudes. *Sociometry*, 1:90–98. (Cited on pages 96, 97 and 113.)

[Simpson, 2007] Simpson, J. a. (2007). Psychological Foundations of Trust. *Current Directions in Psychological Science*, 16(5):264–268. (Cited on pages 18 and 41.)

[Singh and Liu, 2003] Singh, A. and Liu, L. (2003). Trustme: Anonymous management of trust relationships in decentralized p2p systems. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, P2P '03, pages 142–, Washington, DC, USA. IEEE Computer Society. (Cited on page 25.)

[Sloman, 1994] Sloman, M. (1994). Policy Driven Management For Distributed Systems. *Journal of Network and System Management, Vol.*, 2(4). (Cited on page 58.)

[Smith and Makckie, 2000] Smith, E. R. and Makckie, D. M. (2000). *Social Psychology*. Psychology Press. (Cited on page 105.)

[Squicciarini et al., 2007] Squicciarini, A., Bertino, E., Ferrari, E., Paci, F., and Thuraisingham, B. (2007). Pp-trust-x: A system for privacy preserving trust negotiations. *ACM Trans. Inf. Syst. Secur.*, 10(3). (Cited on pages 35 and 72.)

[Sterling and Taveter, 2009] Sterling, S. L. and Taveter, K. (2009). *The Art of Agent-Oriented Modeling*. The MIT Press. (Cited on page 143.)

[Suryanarayana and Taylor, 2004] Suryanarayana, G. and Taylor, R. (2004). A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications. Technical Report UCI-ISR-04-6, ISR. (Cited on page 30.)

[Tambe, 1997] Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial INtelligence Research*, 7:83–24. (Cited on page 103.)

[Tambe, 1998] Tambe, M. (1998). Implementing agent teams in dynamic multiagent environments. *Applied Artificial Intelligence*, 12:189–210. (Cited on page 103.)

[Tambe and Zhang, 2000] Tambe, M. and Zhang, W. (2000). Towards flexible teamwork in persistent teams: Extended report. *Autonomous Agents and Multi-Agent Systems*, 3(2):159–183. (Cited on page 103.)

[Taylor, 1941] Taylor, A. E. (1941). Back to Descartes. *Philosophy*, 16. (Cited on page 18.)

[The Standfod Center for Biomedical Informatics Research (BMIR), 2000] The Standfod Center for Biomedical Informatics Research (BMIR) (2000). Protege: open source ontology editor and knowledge-base framework. (Cited on page 229.)

[Tinnemeier et al., 2009] Tinnemeier, N., Dastani, M., and Meyer, J.-J. (2009). Roles and norms for programming agent organizations. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, pages 121–128, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems. (Cited on pages 103 and 104.)

[Tobias and Hofmann, 2004] Tobias, R. and Hofmann, C. (2004). Evaluation of free Java-libraries for social-scientific agent based simulation. *Journal of Artificial Societies and Social Simulation*, 7(1). (Cited on page 263.)

[Udhayakumar et al., 2011] Udhayakumar, S., Chandrasekaran, S., Tamilselvan, L., and Ahmed, F. (2011). An adaptive trust model for software services in hybrid cloud environment. In *Proceedings of the 15th WSEAS international conference on Computers*, pages 497–502, Stevens Point, Wisconsin, USA. World Scientific and Engineering Academy and Society (WSEAS). (Cited on page 25.)

[Uszok et al., 2003] Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., Hayes, P., Breedy, M., Bunch, L., Johnson, M., Kulkarni, S., and Lott, J. (2003). Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 93–, Washington, DC, USA. IEEE Computer Society. (Cited on page 66.)

[Uszok et al., 2004] Uszok, A., Bradshaw, J. M., Johnson, M., Jeffers, R., Tate, A., Dalton, J., and Aitken, S. (2004). Kaos policy management for semantic web services. *IEEE Intelligent Systems*, 19(4):32–41. (Cited on page 66.)

[van Benthem, 2001] van Benthem, J. (2001). Games in Dynamic-Epistemic Logic. *Bulletin of Economic Research*, 53(4):219–248. (Cited on page 195.)

[van Benthem, 2003] van Benthem, J. (2003). Rational dynamics and epistemic logic in games. Technical report, University of Siena, department of political economy. (Cited on page 195.)

[van Benthem et al., 2011] van Benthem, J., Pacuit, E., and Roy, O. (2011). Toward a Theory of Play: A Logical Perspective on Games and Interaction. *Games*, 2(4):52–86. (Cited on page 195.)

[Vega, 2012] Vega, H. L. (2012). *Open innovation: Organizational practices and policy implications*. PhD thesis, Universiteit Hasselt. (Cited on pages 245, 246 and 259.)

[Venanzi et al., 2011] Venanzi, M., Piunti, M., Falcone, R., and Castelfranchi, C. (2011). Facing openness with socio-cognitive trust and categories. In *IJCAI*, pages 400–405. (Cited on page 25.)

[Vercouter and Muller, 2010a] Vercouter, L. and Muller, G. (2010a). L.i.a.r.: Achieving social control in open and decentralised multi-agent systems. *Journal Applied Artificial Intelligence*, 24(8):723–768. (Cited on pages 32 and 44.)

[Vercouter and Muller, 2010b] Vercouter, L. and Muller, G. (2010b). L.I.A.R.: Achieving Social Control in Open and Decentralized Multiagent Systems. *Applied Artificial Intelligence*, 24(8):723–768. (Cited on pages 32 and 37.)

[Villatoro, 2013] Villatoro, D. (2013). Social Norms for Self-Policing Multi-agent Systems and Virtual Societies. *AI Communications*. (Cited on pages 102, 104 and 107.)

[Vogel and Giese, 2012] Vogel, T. and Giese, H. (2012). Requirements and assessment of languages and frameworks for adaptation models. In Kienzle, J., editor, *Models in Software Engineering*, volume 7167 of *Lecture Notes in Computer Science*, pages 167–182. Springer Berlin Heidelberg. (Cited on page 184.)

[Wang and Varadharajan, 2007] Wang, Y. and Varadharajan, V. (2007). Role-based Recommendation and Trust Evaluation. In *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*, pages 278–288. IEEE. (Cited on page 52.)

[Wehmeyer and Riemer, 2007] Wehmeyer, K. and Riemer, K. (2007). Trust-building potential of coordination roles in virtual organizations. *Journal of Organizational Virtualness*, 8(5). (Cited on page 38.)

[Weibel, 2000] Weibel, S. (2000). The Dublin Core Metadata Initiative. *DLib Magazine*, 6(2). (Cited on page 121.)

[Weinstock, 1999] Weinstock, D. (1999). Building Trust in Divided Societies. *Journal of Political Philosophy*, pages 1–27. (Cited on page 26.)

[Wieselquist et al., 1999] Wieselquist, J., Rusbult, C. E., Foster, C. a., and Agnew, C. R. (1999). Commitment, pro-relationship behavior, and trust in close relationships. *Journal of personality and social psychology*, 77(5):942–66. (Cited on pages 18 and 26.)

[Wikipedia, 2013] Wikipedia (2013). Trust management (information system). (Cited on page 55.)

[Williamson, 1993] Williamson, O. E. (1993). Calculativeness, trust, and economic organization. *Journal of Law and Economics*, 36(1):453–486. (Cited on page 19.)

[Winsborough and Li, 2006] Winsborough, W. H. and Li, N. (2006). Safety in automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 9(3):352–390. (Cited on pages 56 and 191.)

[Wooldridge and Jennings, 1995] Wooldridge, M. and Jennings, N. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115—-152. (Cited on page 101.)

[Xin, 2011] Xin, L. (2011). *Trust beyond reputation: Novel trust mechanisms for distributed environments*. PhD thesis, School of Computer Engineering, Nanyang Technological University. (Cited on page 299.)

[Yagüe, 2006] Yagüe, M. (2006). Survey on xml-based policy languages for open environments. *Journal of Information Assurance and Security*, 1:11–20. (Cited on pages 61, 62 and 76.)

[Yaich et al., 2013] Yaich, R., Boissier, O., Picard, G., and Jaillon, P. (2013). Adaptiveness and Social-Compliance in Trust Management within Virtual Communities. *Web Intelligence and Agent Systems (WIAS), Special Issue: Web Intelligence and Communities*, 11(4). (Cited on page 226.)

[Yao, 2004] Yao, W. (2004). Trust management for widely distributed systems. Technical Report UCAM-CL-TR-608, University of Cambridge Computer Laboratory. (Cited on pages 51, 62, 70, 80, 87 and 111.)

[Yew, 2011] Yew, C. (2011). *Architecture Supporting Computational Trust Formation*. PhD thesis, University of Western Ontario. (Cited on pages 27, 36 and 37.)

[Yu, 2003] Yu, T. (2003). *Automated trust establishment in open systems*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA. AAI3102006. (Cited on pages 57, 60 and 62.)

[Yu et al., 2000] Yu, T., Ma, X., and Winslett, M. (2000). Prunes: an efficient and complete strategy for automated trust negotiation over the internet. In *Proceedings of the 7th ACM conference on Computer and communications security*, CCS '00, pages 210–219, New York, NY, USA. ACM. (Cited on page 62.)

[Yu et al., 2001] Yu, T., Winslett, M., and Seamons, K. E. (2001). Interoperable strategies in automated trust negotiation. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, CCS '01, pages 146–155, New York, NY, USA. ACM. (Cited on page 191.)

[Yu et al., 2003] Yu, T., Winslett, M., and Seamons, K. E. (2003). Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security*, 6(1):1–42. (Cited on pages 35, 45, 46, 69 and 80.)

[Yuan and Tong, 2005] Yuan, E. and Tong, J. (2005). Attributed based access control (ABAC) for Web services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE. (Cited on pages 49 and 52.)

[Zhang et al., 2004] Zhang, H., Goel, A., Govindan, R., Mason, K., and Roy, B. (2004). Making eigenvector-based reputation systems robust to collusion. In Leonardi, S., editor, *Algorithms and Models for the Web-Graph*, volume 3243 of *Lecture Notes in Computer Science*, pages 92–104. Springer Berlin Heidelberg. (Cited on page 189.)

[Zhou and Hwang, 2007] Zhou, R. and Hwang, K. (2007). Gossip-based reputation aggregation for unstructured peer-to-peer networks. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007)*, pages 1–10. IEEE. (Cited on page 27.)

[Zhou et al., 2008] Zhou, R., Hwang, K., and Cai, M. (2008). Gossiptrust for fast reputation aggregation in peer-to-peer networks. *IEEE Trans. on Knowl. and Data Eng.*, 20(9):1282–1295. (Cited on page 27.)

[Zucker, 1986] Zucker, L. G. (1986). Production of Trust: Institutional Sources of Economic Structure. *Research in Organizational Behavior*, 8:53–111. (Cited on pages 19 and 41.)

École Nationale Supérieure des Mines de Saint-Étienne

NNT :2013 EMSE 0717

Mohamed Reda YAICH

Adaptiveness and Social-Compliance in Trust Management. A Multi-agent Based Approach

Major in Computer Science

Keywords: Trust Management, Virtual Communities, Multi-agent Systems, Open Decentralized Systems, Open Innovation

Abstract:Virtual communities (VCs) are socio-technical systems wherein distributed individuals (human and/or artificial) are grouped together around common objectives and goals. In such systems, participants are massively collaborating with each other by sharing their private resources and knowledge. A collaboration always bears the risk that one partner exhibits uncooperative or malicious behaviour. Thus, trust is a critical issue for the success of such systems.

The work presented in this dissertation addresses the problem of trust management in open and decentralised virtual communities (VCs). To address this problem, we proposed an Adaptive and Socially-Compliant Trust Management System (ASC-TMS). The novelty of ASC-TMS lies in its ability to exhibit social-awareness and context-awareness features. Social-awareness refers to the ability of the trust management system (TMS) to handle the social nature of VCs by making trust evaluations that are collectively harmful, while context-awareness refers to the ability of the system to handle the dynamic nature of VCs by making trust evaluations that are always in adequacy with the context in which these evaluations are undertaken.

Thus, the contributions made in this thesis constitute an additional step towards the automation of trust assessment. We provided accordingly a novel trust management system that assists members of open and decentralised virtual communities in their trust decisions. The system has been implemented and deployed using the JaCaMo multi-agent platform. We illustrated also the applicability of on a real life open innovation virtual community scenario. Finally, the ASC-TMS has been experimentally evaluated using the multi-agent based Repast simulation platform. The preliminary results show that the use of our system significantly improves the stability of the virtual communities in which it has been deployed.

Version: 21-01-2014

École Nationale Supérieure des Mines de Saint-Étienne

NNT : 2013 EMSE 0717

Mohamed Reda YAICH

ADAPTATION ET CONFORMITÉ SOCIAL DANS LA GESTION DE LA CONFIANCE. UNE APPROCHE MULTI-AGENTS

Spécialité : Informatique

Mots Clefs : Gestion de la confiance, Communautés virtuelles, systèmes multi-agent, Systèmes Ouverts et Décentralisés, Innovation Ouverte

Résumé : Les communautés virtuelles sont des systèmes sociotechniques dans lesquels des entités (humaines et/ou artificielles) répartis à travers le monde se réunissent autour d'intérêts et/ou d'objectifs communs. Afin de réaliser ces objectifs, les membres de la communauté doivent collaborer en partageant leurs ressources et/ou connaissances. Or, toute collaboration comporte une part de risque dans la mesure où les membres peuvent se comporter de manière non coopérative ou malveillante. Dans de tels contextes, où les mécanismes de sécurité standard ne suffissent plus, la confiance est rapidement devenue un facteur déterminant lors de la prise de décision.

Le travail présenté dans cette thèse s'attaque à la problématique de la gestion de la confiance dans les communautés virtuelles ouvertes et décentralisées. Pour cela, nous avons proposé une infrastructure de gestion de la confiance adaptative et conforme socialement (ASC-TMS). L'aspect novateur de ce système réside dans sa faculté à exhiber des propriétés sociales et adaptatives. L'aspect social du ASC-TMS fait référence à la capacité de notre système à prendre des décisions qui soient sûres non seulement pour l'individu mais également et surtout pour les autres membres de la communauté. Par ailleurs, l'aspect adaptatif du système fait référence à la capacité du système à prendre des décisions qui soient en parfaite adéquation avec l'environnement dans lequel ces décisions sont prises.

Ainsi, cette thèse constitue une nouvelle étape vers l'automatisation de l'évaluation de la confiance en assistant les membres des communautés virtuelles ouvertes et décentralisées dans leur prise de décision. Le système a été implémenté et déployé en utilisant la plateforme de développement multi-agent JaCaMo. Nous avons également illustré l'applicabilité de notre approche sur un scénario réel de communauté virtuelle d'innovation ouverte. Enfin, nous avons évalué notre système expérimentalement en utilisant la plateforme de simulation multi-agent Repast. Les résultats obtenus montrent que l'utilisation de notre système avait un impact positif sur la dynamique des communautés dans lesquels il est a été utilisé.

Version : 21-01-2014