Ontologies for Agents: Theory and Experiences

Valentina Tamma
Stephen Cranefield
Timothy W. Finin
Steven Willmott
Editors

Birkhäuser Verlag
Basel · Boston · Berlin

Editors:

Valentina Tamma
University of Liverpool
Agent Applications,
Research and Technology Group
Department of Computer Science
Chadwick Building
Liverpool L69 7ZF
Great Britain

Stephen Cranefield
University of Otago
Department of Information Science
PO Box 56
Dunedin
New Zealand

Timothy W. Finin
University of Maryland
329 Information Technology and Engineering
Baltimore County
1000 Hilltop Circle
Baltimore MD 21250
USA

Steven Willmott
Universitat Politècnica de Catalunya (UPC)
Dept. Llenguatges i Sistemes Informatics
Modul C5, 211b
Campus Nord
08034 Barcelona
Spain

2000 Mathematical Subject Classification 68T35, 68U35, 94A99, 94C99

A CIP catalogue record for this book is available from the
Library of Congress, Washington D.C., USA

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie;
detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

ISBN 3-7643-7237-0 Birkhäuser Verlag, Basel – Boston – Berlin

Cover design: Micha Lotrovsky, CH-4106 Therwil, Switzerland
Printed on acid-free paper produced from chlorine-free pulp. TCF ∞
Printed in Germany
ISBN-10: 3-7643-7237-0
ISBN-13: 978-3-7643-7237-8

9 8 7 6 5 4 3 2 1 www.birkhauser.ch

Contents

Foreword

There is a growing interest in the use of ontologies for multi-agent system applications. On the one hand, the agent paradigm is successfully employed in those applications where autonomous, loosely-coupled, heterogeneous, and distributed systems need to interoperate in order to achieve a common goal. On the other hand, ontologies have established themselves as a powerful tool to enable knowledge sharing, and a growing number of applications have benefited from the use of ontologies as a means to achieve semantic interoperability among heterogeneous, distributed systems.

In principle ontologies and agents are a match made in heaven, that has failed to happen. What makes a simple piece of software an *agent* is its ability to communicate in a "social" environment, to make autonomous decisions, and to be proactive on behalf of its user. Communication ultimately depends on understanding the goals, preferences, and constraints posed by the user. Autonomy is the ability to perform a task with little or no user intervention, while proactiveness involves acting autonomously with no need for user prompting. Communication, but also autonomy and proactiveness, depend on knowledge. The ability to communicate depends on understanding the syntax (terms and structure) and the semantics of a language. Ontologies provide the terms used to describe a domain and the semantics associated with them. In addition, ontologies are often complemented by some logical rules that constrain the meaning assigned to the terms. These constraints are represented by inference rules that can be used by agents to perform the reasoning on which autonomy and proactiveness are based.

In practice, the application areas of these technologies often overlap, for example: e-commerce, intelligent information integration, and web services. Increasingly, the multi-agent systems and ontology research communities are seeking to work together to solve common problems. A key focus to this joint working is emerging in ideas for the semantic web. Both ontologies and agent technologies are central to the semantic web, and their combined use will enable the sharing of heterogeneous, autonomous knowledge sources in a scalable, adaptable and extensible manner.

This volume collects the most significant papers of the AAMAS 2002 and AAMAS 2003 workshop on ontologies for agent system, and the EKAW 2002 workshop on ontologies for multi-agent systems. The workshops were taking different perspectives to the topic of using ontologies in the framework of a multi-agent system. On the one hand, there is the knowledge modelling perspective; *i.e.* how ontologies should be modelled and represented in order to be effectively used in agent systems. On the other hand, there is the agent perspective; what kind of capabilities should be exhibited by an agent in order to make use of ontological knowledge and to perform efficient reasoning with it.

The volume aims at providing a comprehensive review of the diverse efforts covering the gap existing between these perspectives. The papers cover a wide range of topics but can mainly be grouped in three categories: modelling principles for building and reasoning with ontologies for agents, semantic interoperability between different agents, and applications of ontologies in agent systems.

Modelling ontologies entails dealing with the problems of building ontologies, and establishing ontological commitment. Semantic interoperability includes reasoning with ontological knowledge that agents may use to proactively overcome differences in their conceptualisation of the world, and applications of ontologies concern real life examples of how ontologies can be used in agents.

For what concerns modelling and representing ontologies, Cranefield and colleagues in their first contribution propose to reduce the degree of human interpretation currently necessary to understand an interaction protocol, by describing at an abstract level the required agent actions that must be 'plugged into' the protocol for it to be executed. In particular, this can be done by designing and publishing ontologies describing the input and output data that are processed during the protocol's execution, together with the actions and decisions that the agents must perform.
Nodine and Fowler concentrate on ontological commitment, or the agreement to have applications and users conform to a common domain understanding, as encapsulated in one or more shared ontologies. They present their experiences in building ontology-based agent systems in multiple domains and illustrate the problems arising when a new application aims to locate and conform to some existing ontology or ontologies within its domain. The authors propose guidelines for ontology development and evolution, which should facilitate ontology reuse that may underpin a *usage model* for ontologies; one that enables the application designer to reuse ontological concepts from multiple ontologies in a more flexible manner, while retaining the essentially good properties of ontology sharing and reuse.
Pazienza and Vindigni also concentrate on ontological commitment, and in particular, on the lack of a shared knowledge model that can be assumed as a default *ontological commitment*. They propose a communication model based on the use of natural language, that predicates a strong separation among terms and concepts. In order to support the proposed communication model, the authors present a novel agent architecture able to deal with possible linguistic ambiguities by focusing on the conversational level.

An important part of this volume is devoted to approaches aimed at finding an ontological model that is shared by all the agents composing a system. These approaches become particularly important when agents commit to heterogenous ontologies. Dou and colleagues present an approach to ontology translation, one of the hardest problems agents must cope with. In their approach, the merging of two related ontologies is obtained by taking the union of the terms and the axioms defining them. Bridging axioms are added, not only as bridges between terms in two related ontologies, but also to make this merge into a completely new ontology, which can subsequently be merged with other ontologies. Translation is implemented using an inference engine (OntoEngine), running in either a demand-driven (backward chaining or data-driven (forward chaining) mode.

Leen-Kiat Soh contribution describes a multiagent framework for collaborative understanding of distributed ontologies. The framework aims to investigate and identify how agents collaborate to understand each other under resource constraints and operational setups, and to examine how agents manage and share their distributed ontologies triggered by various queries. To facilitate collaborative understanding, each agent maintains an ontology and a translation table with other agents or neighbors.
In Lister and colleagues, the authors address the problem of semantic interoperability on the web, and present their research experiments suggesting that as yet unaddressed issues should be considered; such as reconciling implicit ontologies, evolving ontologies, and task-oriented analysis. The authors consider the role of semantic interoperation in multi-agent systems, and describe strategies for achieving it via the ROADMAP methodology.
Stuckenschmidt and colleagues concentrate on the problem of answering queries over multiple data sources in a dynamic environment, where it is no longer realistic to assume that the involved data sources act as if they were a single (virtual) source, modelled as a global schema. In their contribution, they propose an alternative approach where they replace the role of a single virtual data source schema with a peer-to-peer approach relying on limited, shared (or overlapping) vocabularies between peer agents.
Chris van Aart and colleagues present an approach to agent communication, based on message content ontologies that specify the meaning and intention of messages. By committing to a shared ontology, several agents can reach an agreement on different agent communication languages.

With respect to applications, Annamalai and Sterling investigate the possibility for agent systems aiding with collaboration among Experimental High-Energy Physics (EHEP) physicists. They argue that a necessary component is an agreed scientific domain ontology, which must include concepts that rely on mathematical formulae involving other domain concepts, such as energy and momentum. In this work, previous efforts on representing mathematical expressions are adapted to produce a set of representational primitives and supporting definitions for modelling complex mathematical relations.
Chen and colleagues investigate the use of ontologies in a multi-agent system providing brokering services for pervasive computing. Cranefield and colleagues, in their second contribution, propose the use of a UML profile for ontology modelling, to represent an ontology for travel booking services, and automatically derive an object-oriented content language for this domain. This content language is then used to encode example messages for a simple travel booking scenario, and it is shown how this approach to agent communication allows messages to be created and analysed using a convenient object-oriented, agent-specific application programmer interface. Dickinson and Wooldridge present a belief-desire-intention (BDI) approach to the problem of developing an agent-assisted travel scenario, and ask what role ontologies would have in supporting the agent's activity. To

this end, their contribution discusses the Nuin agent platform, and illustrates various ways in which ontology reasoning supports BDI-oriented problem solving and communications by the agents in the system.
Sashima and colleagues focus on the problem of achieving coordination in ubiquitous computing, and in particular. bridging the coordination gap separating devices, services, and humans. They propose an agent-based coordination framework for ubiquitous computing to solve this human-centered service coordination issue.
Zimmerman and colleagues present agent-based supply chain monitoring system for tracking orders, in which communication is enabled through the definition of a shared ontology. The paper discusses the design of the ontology and its use for inter-agent communication is illustrated with the help of AUML models of the agent-interactions in the supply chain monitoring system.

Valentina Tamma

Valentina Tamma
Department of Computer Science
University of Liverpool
Chadwick Building
L69 7ZF Liverpool
Great Britain
e-mail: `V.A.M.Tamma@csc.liv.ac.uk`

Stephen Cranefield
Department of Information Science
University of Otago
Dunedin
New Zealand
e-mail: `scranefield@infoscience.otago.ac.nz`

Timothy W. Finin
Information Technology and Engineering
Baltimore County
Baltimore
USA
e-mail: `finin@cs.umbc.edu`

Steven Willmott
Dept. Llenguatges i Sistemes Informatics
Universitat Politecnica
Barcelona
Spain
e-mail: `steve@lsi.upc.es`

Ontologies for Interaction Protocols

Stephen Cranefield, Martin Purvis, Mariusz Nowostawski and Peter Hwang

Abstract. In this paper we propose reducing the degree of human interpretation currently necessary to understand an interaction protocol by describing at an abstract level the required agent actions that must be 'plugged into' the protocol for it to be executed. In particular, this can be done by designing and publishing ontologies describing the input and output data that are processed during the protocol's execution together with the actions and decisions that the agents must perform. An agent (or agent developer) that has previously defined mappings between the internal agent code and the actions and decisions in an ontology would then be able to interpret any interaction protocol that is defined with reference to that ontology. The discussion is based on the use of Coloured Petri Nets to represent interaction protocols and the Unified Modeling Language (UML) for ontology modelling. An alternative approach using Agent UML (AUML) is also outlined.

1. Introduction

Agent communication languages (ACLs) such as the Knowledge Query and Manipulation Language (KQML) [11] and the Foundation for Intelligent Physical Agents (FIPA) ACL [15] are based on the concept of agents interacting with each other by exchanging messages that specify the desired 'performative' (inform, request, etc.) and a declarative representation of the content of the message. Societies of agents cooperate to collectively perform tasks by entering into conversations—sequences of messages that may be as simple as request/response pairs or may represent complex negotiations. In order to allow agents to enter into these conversations without having prior knowledge of the implementation details of other agents, the concepts of agent conversation policies [17] and interaction protocols have emerged[1]. These are descriptions of standard patterns of interaction between

[1]Some authors treat these terms as synonymous, while others [13] make a distinction between them in terms of the type and generality of the representation formalism used.

two or more agents. They constrain the possible sequences of messages that can be sent amongst a set of agents to form a conversation of a particular type. An agent initiating a conversation with others can indicate the interaction protocol it wishes to follow, and the recipient (if it knows the protocol) then knows how the conversation is expected to progress. A number of interaction protocols have been defined, in particular as part of the FIPA standardisation process [16].

The specification of the individual messages comprising an interaction protocol is necessarily loose: usually only the message performative, sender and receiver are described. This is because an interaction protocol is a generic description of a pattern of interaction. The actual contents of messages will vary from one execution of the protocol to the next. Furthermore, the local actions performed and the decisions made by agents, although they may be related to the future execution of the protocol, are traditionally either not represented explicitly (e.g. in an Agent UML sequence diagram representation [28]) or are represented purely as labelled 'black boxes' (e.g. in a Petri net representation [5]).

In this paper we argue that the traditional models of interaction protocols are suitable only as specifications to guide human developers in their implementation of multi-agent systems, and even then often contain a high degree of ambiguity in their intended interpretation. Here we are not referring to the necessity for an interaction protocol to have formal semantics (although that is an important issue). Rather, we see a need for techniques that allow the designers of interaction protocols to indicate their intentions unambiguously so that a) other humans can interpret the protocols without confusion, and b) software agents can interpret protocols for the purposes of generating conversations. Ideally, an agent would be able to download an interaction protocol previously unknown to it, work out where and how to 'plug in' to the protocol its own code for message processing and for domain-specific decision making, and begin using that protocol to interact with other agents.

We propose reducing the degree of human interpretation currently necessary to understand an interaction protocol by describing at an abstract level the required agent actions that must be plugged into the protocol for it to be executed. In particular, this can be done by designing and publishing ontologies describing the input and output data that are processed during the protocol's execution together with the actions and decisions that the agents must perform. An agent (or agent developer) that has previously defined mappings between the internal agent code and the actions and decisions in an ontology would then be able to interpret any interaction protocol that is defined with reference to that ontology.

For example, consider a protocol describing some style of auction. Inherent in this protocol are the concepts of a bid and response and the actions of evaluating a bid (with several possible outcomes). There are also some generic operations related to any interaction protocol such as the parsing of a message to check that it has a particular performative and that its content can be understood by the agent in the current conversational context, and the creation of a message.

In the next section, we motivate our work by discussing an example: the FIPA Request Interaction Protocol, as specified by FIPA using Agent UML (AUML). This specification lacks a number of important details that would be required for an agent to use the protocol without additional human interpretation. In Section 3 we illustrate how a more detailed model of this interaction protocol can be defined using a coloured Petri net, and explain the role that an ontology plays in the definition. Section 4 then presents two approaches to defining the internal operations that an agent must implement in order to play a particular role in an interaction protocol. In Section 5 we return to consider the use of AUML for interaction protocol modelling, due to the current effort underway in FIPA to enhance and fully specify the language. We give an overview of our recent work on the development of an agent conversation controller that directly interprets AUML sequence diagrams, based on techniques similar to those discussed in Sections 3 and 4 in the context of coloured Petri nets. Section 6 concludes the paper.

2. Example: the FIPA Request Interaction Protocol

Figure 1 shows the FIPA Request Interaction Protocol[2] [14] using AUML [28]. This protocol defines a simple interaction between two agents. One agent plays the Initiator role and sends a request for an action to be performed to another agent which plays the Participant role. The protocol illustrates that there are three alternative responses that the participant can make after receiving the request: it can refuse or agree to the request or it may signal that it did not understand the request message. If it agreed, it subsequently sends a second response: a message indicating that its attempt to fulfil the request action failed, a message signalling that the action has been performed, or a message containing the result of performing the requested action.

There are some aspects of this protocol that are not specified. For example, it is not specified that each of the `not-understood`, `agree` and `refuse` messages should contain part of the original request in their content tuple, along with an additional proposition (representing respectively an error message, a precondition for the action to be performed, and a reason for refusal). To make the specification more precise there needs to be a way of annotating the protocol with constraints on the contents of the messages and the relationships between them. These constraints would need to be expressed in terms of a vocabulary relating to the structure of messages, i.e. an ontology for messages.

Furthermore, the underlying *intention* of this protocol is not explicitly specified. In order to customise this protocol to a particular domain, a request initiator agent must 'plug in' domain-specific procedures at six different points: the handling of `not-understood`, `refuse` and `failure` messages, analysing an `agree`

[2]The figure shows Version F of the protocol. Since this paper was written a more recent version of the protocol has been released by FIPA and promoted to standard status. The new version has removed the `not-understood` message and made the `agree` message optional.

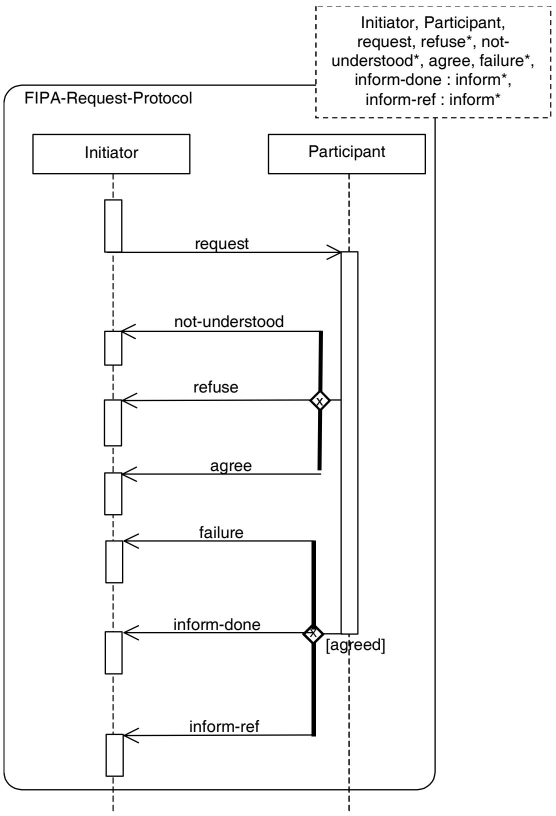
Initiator, Participant, request, refuse*, not-understood*, agree, failure*, inform-done : inform*, inform-ref : inform*
FIPA-Request-Protocol
Initiator
Participant
request
not-understood
refuse
agree
failure
inform-done
[agreed]
inform-ref

Figure 1. The FIPA Request Interaction Protocol defined using AUML

message to check if a precondition is specified by the participant, and the handling of the two different types of response that indicate the action was successfully performed. Similarly, there are three pieces of domain-specific functionality that an agent wishing to play the role of participant must supply: the parsing of the action being requested (possibly resulting in a failure to understand the message), choosing whether to agree to the request, and the choice between an `inform-done` response or an `inform-ref` message describing the action's result. We believe that an interaction protocol is not completely specified until the interface between the domain-specific agent-supplied code and the generic interaction protocol is defined. Clearly, interaction protocols should remain as generic as possible, making no commitment to any particular agent platform or implementation language. Thus the specification of this interface should be in terms of a programming-language independent representation. Furthermore, the agent operations related to a particular protocol will be related to the types of entity involved in the execution of that protocol, e.g. the notion of a bid in a 'call for proposals' protocol. This model of protocol-related concepts and the operations that act on them is an ontology that needs to be supplied along with the interaction protocol to give it a full specification.

3. A Coloured Petri Net approach

The discussion above was based on an analysis of an interaction protocol expressed as an AUML sequence diagram. However, the version of AUML used in the current FIPA standards has some shortcomings for further investigation of these ideas. First, this version of AUML is underspecified and the intended interpretation of an AUML sequence diagram is not always clear. Second, there are no graphical editors that directly support the use of AUML. Finally, these AUML sequence diagrams do not have a way of explicitly modelling the internal actions of agents[3]—which are exactly the points of the protocol at which we need to attach annotations refering to an ontology. We therefore adopted an alternative modelling language for our initial research in this area: coloured Petri nets.

Petri Nets [24] are a formalism and associated graphical notation for modelling dynamic systems. The state of the system is represented by *places* (denoted by hollow circles) that can contain *tokens* (denoted by symbols inside the places). The possible ways that the system can evolve are modelled by defining *transitions* that have input and output arcs (denoted by arrows) connected to places. The system dynamics can be enacted non-deterministically by determining which transitions are *enabled* by the presence of tokens in the input places, selecting one and *firing* it, which results in tokens being removed from its input places and new tokens being generated and placed in its output places.

Coloured Petri nets (CPNs) [18] are an elaboration of ordinary Petri nets. In a coloured Petri net, each place is associated with a 'colour', which is a type (although the theory of CPNs is independent of the choice of type system). Places can contain a multiset of tokens of their declared type. Each input arc to a transition is annotated with an expression (possibly containing variables) that represents a multiset of tokens. For a transition to be enabled, it must be possible to match the expression on each input arc to a sub-multiset of the tokens in the associated input place. This may involve binding variables. In addition, a Boolean expression associated with the transition (its *guard*) must evaluate to true, taking into account the variable bindings. When a transition is fired, the matching tokens are removed from the input places and multiset expressions annotating the output arcs are evaluated to generate the new tokens to be placed in the output places. If the expression on an output arc evaluates to the empty multiset then no tokens are placed in the connected place.

[3]UML activity diagrams have this capability and can be used on their own or in conjunction with sequence diagrams to specify the internal agent processing [28, 20].

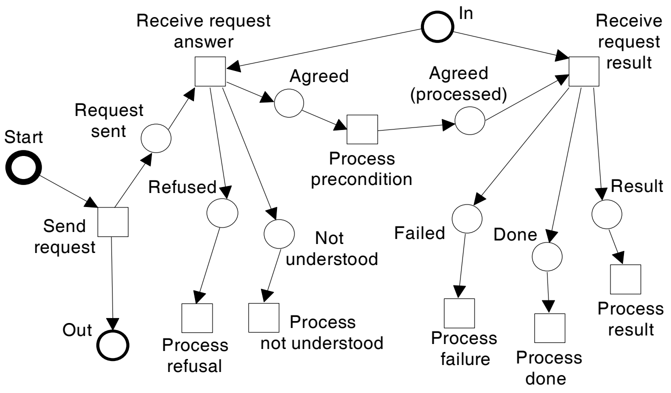

FIGURE 2. The Initiator role for the Request protocol as a CPN (outline only)

The coloured Petri net formalism provides a powerful technique for defining system dynamics and has previously been proposed for use in modelling interaction protocols [5, 19]. In this paper we take a different approach from our previous work [26, 25, 31] in the application of CPNs to interaction protocol modelling. We choose to model each side of the conversation (a *role*) using a separate CPN (we have also discussed this approach elsewhere [34, 35, 33, 32], and a variety of approaches have been used by other researchers [30, 23, 21]). Figure 2 shows an overview of the net for the Initiator role of the FIPA Request protocol (the 'colours' of places, the arc inscriptions and the initial distribution of tokens are not shown). In the figure, places are represented by circles and transitions are represented by squares. No tokens are shown. The places labelled In and Out are *fusion* places: they are shared between all nets for the roles the agent can play (in any interaction protocol). The agent's messaging system places tokens representing received messages in the In place and removes tokens from the Out place (these represent outgoing messages) and sends the corresponding messages.

The fully detailed version of this Petri net encodes the following process. The Initiator begins the conversation by sending a request with its `reply-with` parameter set to a particular value. When an answer with a matching `in-reply-to` parameter value is received, the Receive request answer transition is enabled and can subsequently fire at the agent's discretion. This transition generates a single token that is placed in one of the Agreed, Refused or Not understood places, depending on the communicative act of the reply (the remaining two output arcs each generate an empty multiset of tokens, i.e. no tokens are placed in their output places). In the case that the other agent agreed to the request, another message is subsequently expected from that agent, containing the result of the requested action. This is handled by the right hand side of the net in a similar fashion.

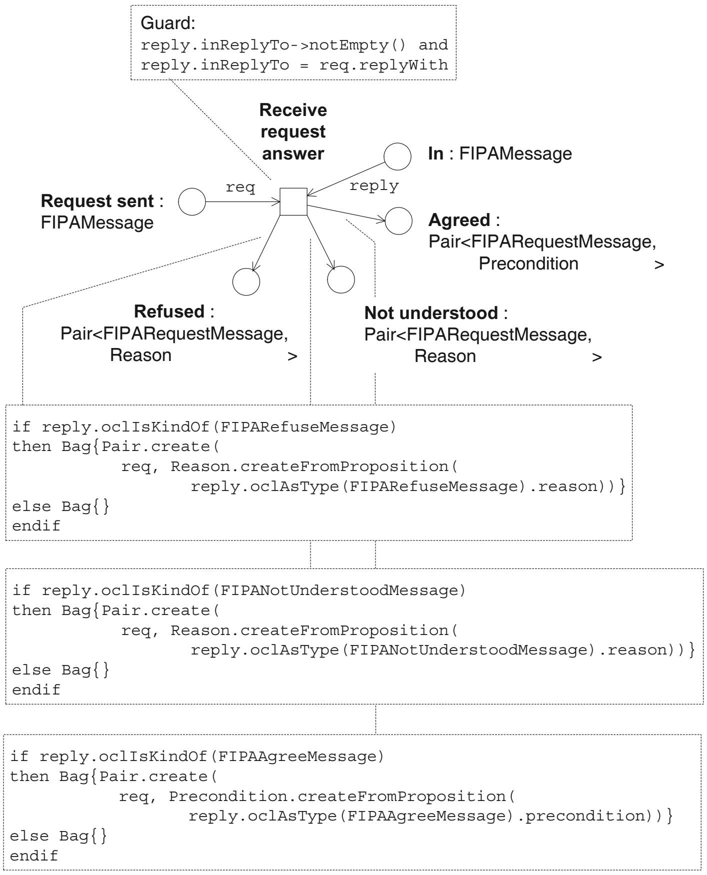
Guard:
reply.inReplyTo->notEmpty() and
reply.inReplyTo = req.replyWith
Receive request answer
In : FIPAMessage
req
reply
Request sent : FIPAMessage
Agreed : Pair<FIPARequestMessage, Precondition >
Refused : Pair<FIPARequestMessage, Reason >
Not understood : Pair<FIPARequestMessage, Reason >
if reply.oclIsKindOf(FIPARefuseMessage)
then Bag{Pair.create(
req, Reason.createFromProposition(
reply.oclAsType(FIPARefuseMessage).reason))}
else Bag{}
endif
if reply.oclIsKindOf(FIPANotUnderstoodMessage)
then Bag{Pair.create(
req, Reason.createFromProposition(
reply.oclAsType(FIPANotUnderstoodMessage).reason))}
else Bag{}
endif
if reply.oclIsKindOf(FIPAAgreeMessage)
then Bag{Pair.create(
req, Precondition.createFromProposition(
reply.oclAsType(FIPAAgreeMessage).precondition))}
else Bag{}
endif

FIGURE 3. Details of the 'Receive request answer' transition

In this Petri net we have included transitions that correspond to internal actions of the agent, such as those labelled Process refusal and Process not understood. These are not part of the protocol when it is viewed in the pure sense of simply being a definition of the possible sequences of messages that can be exchanged. However, we believe these 'internal' transitions communicate the underlying intent of the protocol: there are a number of points at which the agent must invoke particular types of computation to internalise and/or react to the different states that can occur. In the example shown, most of these internal transitions occur after the final states of the protocol. However, this is not necessarily the case, e.g. the Process precondition transition gives the agent a chance to reason about the precondition that may be specified by the other agent when it agrees to the request. This precondition must become true before the other agent will fulfil the request (in the simple case it can just be the expression `true`). Although the Request interaction protocol does not allow for any extra communication between the two agents regarding this precondition, an agent might wish to do something outside the scope of the conversation to help satisfy the precondition (e.g. perform an action). Therefore, the initiator needs an opportunity to notice the precondition.

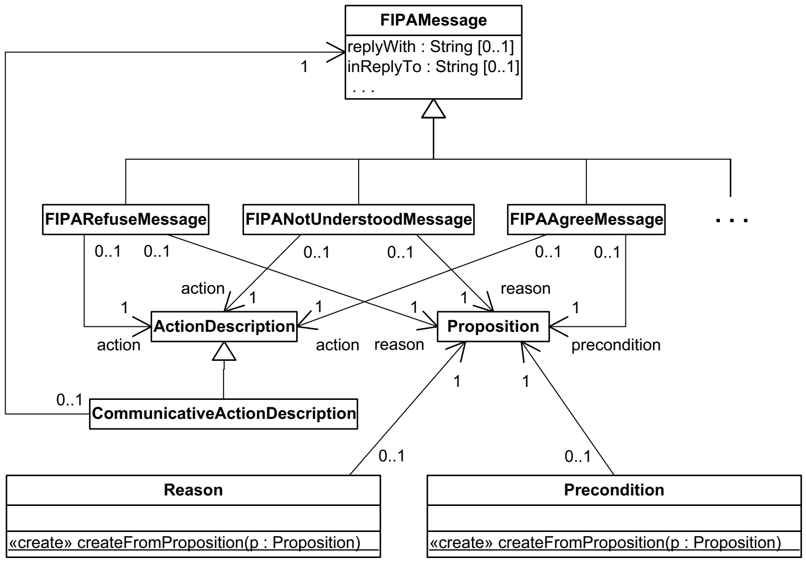

Figure 4. A partial ontology for the Request interaction protocol

Figure 3 shows the details of the Receive request answer transition. This is where we make the connection with ontologies: the types used as place colours and within arc expressions are concepts in an associated ontology (a portion of which is shown in Figure 4). We use the object-oriented Unified Modeling Language (UML) [3] to represent the ontology and UML's associated Object Constraint Language (OCL) [36] as our expression language. For brevity, we adopt the convention that a variable `x` appearing on an input arc represents the singleton multiset `Bag{x}` (`Bag` is the OCL type corresponding to a multiset).

In the case of the Request protocol, the concepts that need to be defined in the associated ontology are message types. Figure 4 therefore defines an inheritance hierarchy of FIPA ACL message types[4] (generalisation relationships are represented by arrows with triangular heads, while associations are represented by arrows with open heads). In addition, we have chosen to explicitly model the concepts of a reason and a precondition that are associated with the Request protocol. Within a FIPA ACL message these are both represented as propositions, but

[4]A more complex UML model for FIPA messages has been presented elsewhere [6], but that serves a different purpose. The model in this paper is not intended as an update of that previous work, but instead provides a different view of FIPA message types.

here the `Reason` and `Precondition` classes can be used (via their constructors) to achieve an additional level of interpretation of a proposition. Note that although the ontology is shown here as being a monolithic model, in practice some of the classes shown would be imported from a separate UML package.

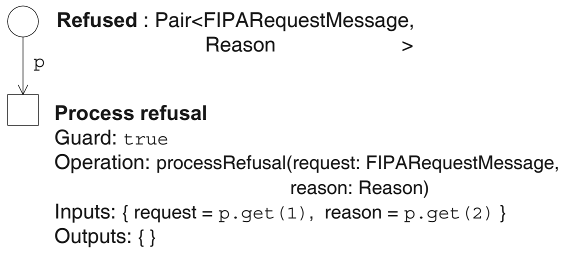
Refused : Pair<FIPARequestMessage,
Reason >
p
Process refusal
Guard: true
Operation: processRefusal(request: FIPARequestMessage,
reason: Reason)
Inputs: { request = p.get(1), reason = p.get(2) }
Outputs: { }

FIGURE 5. Details of the 'process request refusal' transition

In addition to the classes shown, the ontology is assumed to include a UML *class template* called `Pair`. A class template is a class that is defined in terms of one or more other classes, which are specified only as parameter names. When it is used (as in Figure 3) specific types must be supplied to instantiate the parameters. `Pair` represents a pair of elements with the type of each argument being the corresponding supplied parameter.

The arc expressions in Figure 3 use the operations `oclIsKindOf` and `oclAsType`. These are predefined OCL operations used for run-time type checking and type casting respectively.

4. Modelling Internal Agent Operations

In Figure 3, all processing represented by the transition is performed by the guard and the output arc expressions. This is not always the case. Consider the Process request refusal transition from Figure 2 (shown in detail in Figure 5). This represents the computation that an agent must do to react to the participant's refusal of the request. Although any future actions of the initiator agent are outside the scope of the Request protocol, in order for the protocol model to act as a stand-alone specification (without relying on implicit assumptions about the meaning of certain places) it should define the way in which the agent transfers information from the Petri net to its own internal processes. To support this, we optionally associate an *operation* with each transition, specifying the inputs to the operation as OCL expressions and providing a list of variables to which the outputs should be assigned (note that UML allows multiple output parameters in an operation). Figure 5 illustrates this for the Process request refusal transition.

The operations required to interface an agent with the CPN for a given role constitute part of the ontology for the protocol. In this section we describe two

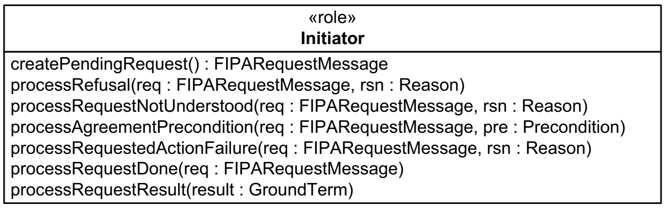
«role»
Initiator
createPendingRequest() : FIPARequestMessage
processRefusal(req : FIPARequestMessage, rsn : Reason)
processRequestNotUnderstood(req : FIPARequestMessage, rsn : Reason)
processAgreementPrecondition(req : FIPARequestMessage, pre : Precondition)
processRequestedActionFailure(req : FIPARequestMessage, rsn : Reason)
processRequestDone(req : FIPARequestMessage)
processRequestResult(result : GroundTerm)

FIGURE 6. 'Static' specification of the Initiator role for the Request protocol

approaches for using UML to model the operations required for particular roles: a simple 'static' approach and a more flexible but complex 'dynamic' approach.

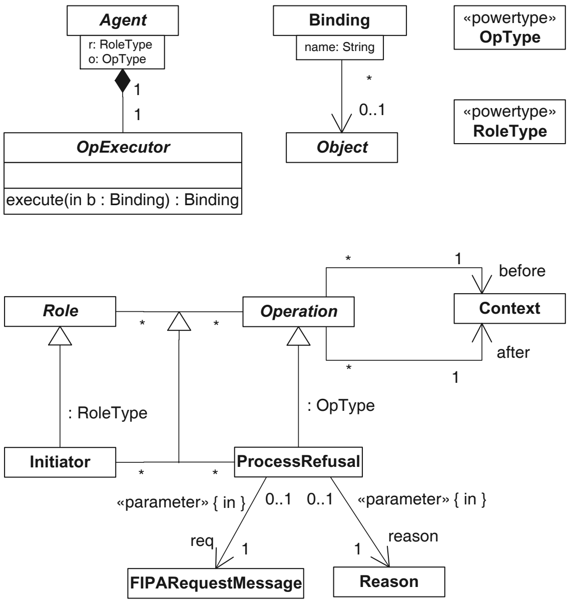

FIGURE 7. Specifying operations as first-class objects

4.1. The Static Approach

Figure 6 illustrates the static approach to including a role's operations in a UML ontology. This figure shows a class (annotated with the «role» stereotype) representing the role and containing all required operations. Although this looks like the specification of an application programmer interface rather than an ontology, it is not intended that an agent must implement operations with the same signatures as shown here. Instead an agent may be able to map these operations into those it does possess. To do this, the description of the role's required operations would need to include some information about their semantics, possibly using OCL pre- and postcondition expressions. This is a subject for future research.

The representation in Figure 6 does not model the operations required for a given role as first class objects in UML, but as features of a class representing the role. Although this has the benefit of simplicity, it has a number of shortcomings. Essentially it treats a role as an interface that an agent must implement if it wants to act in that role. We call this the static approach because it does not accommodate in a straightforward way the possibility of agents dynamically changing the roles they support. The UML object model does not allow classes (or agent types in this scenario) to change their set of implemented interfaces at run time. Also, the notation does not show graphically the relationships between the operations and the ontological concepts on which they depend.

4.2. The Dynamic Approach

Figure 7 shows an alternative approach that addresses the concerns raised above. The majority of the figure represents a base ontology containing classes to which a specific role ontology would make reference. Only the four classes at the bottom of the figure represent a specific ontology: a portion of the ontology for the Request interaction protocol.

Modelling both entities and the operations that act on them as first class objects is difficult to do in a straightforward way without departing from a "strict metamodelling architecture" where there is a firm distinction between instances and classes [1]. In this case, to allow the use of associations to define the types of operation arguments, each operation must be defined as a class. The abstract class `Operation` represents the concept of an operation that is associated with a role and which relates two contexts: the relevant local states of the world before and after the operation is performed. Particular operations are modelled as subclasses of `Operation` with their input and output arguments represented by associations labelled with the UML stereotype «parameter» and one of the Boolean 'tagged values' `in` or `out`

If operations are classes, we need to consider what their instances are. The answer is that the instances represent snapshots of the operation's execution in different contexts and with different arguments, in the same way that a mathematical function can be regarded as the set of all the points on its graph. However, the operation class only serves as a description of the operation: it will not be instantiated by an agent. Instead we model an agent as containing a collection of `OpExecutor` objects, each being an instance of some class that implements an operation. These objects are indexed by role and operation (this is shown using UML's qualified association notation). Roles and operations are both modelled as classes, so the types for these association qualifiers must be powertypes of `Operation` and `Role`. A powertype is a class whose instances are all the subclasses of another class [22, Chapter 23].

To invoke an operation, an agent calls `execute` on an `OpExecutor` object. The arguments to this method must be completely generic, so a binding structure is provided as an argument. This maps the operation's argument names to objects. The operation returns another binding list specifying values for any output parameters.

5. AUML Revisited

Section 3 motivated our use of coloured Petri nets by discussing some shortcomings of AUML as used in the FIPA interaction protocols. Recently FIPA has formed the Modeling Technical Committee [12] which is developing a new version of AUML, based on UML 2.0 and various UML-inspired modelling notations that have been defined by researchers in the area of agent-based software engineering. To date, work has focussed on the graphical notation, with a metamodel to be developed later. In parallel with (and informed by) the FIPA work on notation, we have have undertaken a preliminary investigation of how interaction protocols can be represented by sequence diagrams in sufficient detail to allow them to be directly interpreted by agents (once the application-specific code has been 'plugged in' to the appropriate points in the protocol) [9]. This required developing a mechanism for specifying the connection between the contents of incoming messages, the parameters and results of agents' internal operations, and the contents of outgoing messages—similar to the technique presented in Section 4 for use with coloured Petri nets. We also developed a metamodel for AUML sequence diagrams (based on a subset of the UML 2.0 metamodel), and implemented a library[5] for the Opal agent platform [29] that allows an agent to read an interaction protocol sequence diagram encoded in the XMI format, plug in methods that correspond to the internal actions specified in the protocol, and take part in a conversation according to that protocol by directly interpreting the sequence diagram.

Figure 8 illustrates the sequence diagram notation handled by our tool. The rectangles appearing on the lifelines represent the operations performed by agents when processing incoming messages and generating responses. These are specified by an operation name (not shown in the figure) and *start* and *end constraints*, denoted by dog-eared rectangles. These constraints specify how the inputs and outputs of the operations are related to the contents of the messages and variables of the protocol. In general, we believe that OCL would be the apppropriate language for expressing these constraints. However, as our tool is based on the Eclipse Modelling Framework (EMF) [8], and an OCL interpreter for EMF [27] was not yet available at the time this work was done, we used Java statements (interpreted by BeanShell [2]) for our prototype AUML interpreter. Figure 9 shows the declaration of the inputs and outputs of the `createProposal` operation from Figure 8: the operation's inputs are the two expressions in the content of the incoming CFP message (an action description and a referential expression describing a property that any responding proposal should satisfy), and the outputs are the action the

[5]The current library is an early prototype that supports only two-party conversations and a subset of sequence diagram features.

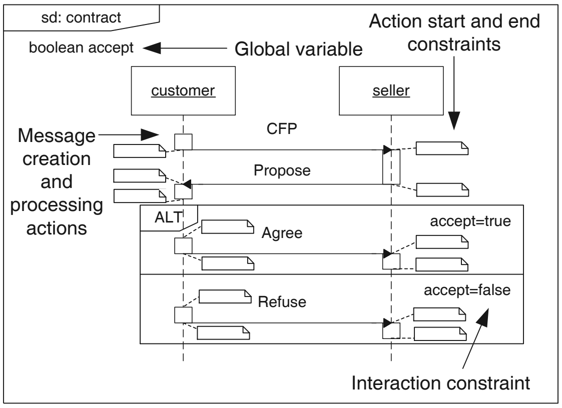

Figure 8. An executable AUML interaction protocol

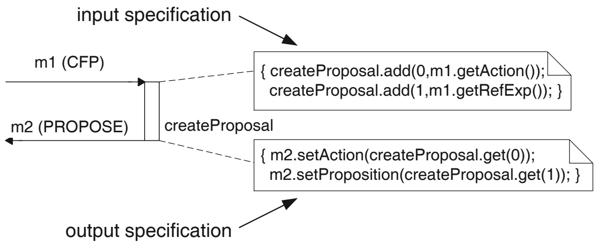
input specification
m1 (CFP)
{ createProposal.add(0,m1.getAction());
createProposal.add(1,m1.getRefExp()); }
m2 (PROPOSE)
createProposal
{ m2.setAction(createProposal.get(0));
m2.setProposition(createProposal.get(1)); }
output specification

Figure 9. Declaring operation inputs and outputs in AUML

seller will propose to do (probably the same as that requested) and a proposition expressing the terms of the bid.

6. Conclusion

In this paper we have identified two weaknesses in traditional mechanisms for specifying agent interaction protocols: a lack of precision in defining the form of messages that are exchanged during the protocol and the relationships between them, and the lack of any explicit indication of where and how the protocol interfaces with an agent's internal computation. We have proposed the use of an ontology associated with a protocol to define the relevant concepts and the internal operations that an agent needs in order to partake in a conversation using that protocol, and have illustrated how this information can be represented in interaction protocols defined using coloured Petri nets and an extended version of AUML.

We note that some uses of interaction protocols are not concerned with the internal actions of agents, e.g. external monitoring of conversations for the purpose of compliance testing or debugging multi-agent systems [30]. For this type of application it may be beneficial to provide a simpler view of protocols that abstracts away the transitions representing internal actions.

Two techniques were proposed for modelling the agent internal actions necessary to use an interaction protocol: a static model and a dynamic model. We believe the dynamic model, although more complex, is more flexible and has more scope for adding semantic annotations to define the operations—an extension necessary to enable agents to deduce how to use their existing operations to implement those required by an interaction protocol.

The type of ontology discussed in this paper combines descriptions of concepts and operations that act on them in a single model. In the knowledge acquisition research community there has been considerable study of techniques for building libraries of reusable problem-solving methods, and work has been done on combining such libraries and ontologies in a single system [10]. This research may provide some insights into the problems of integrating action descriptions into ontologies.

The aim of the work described in this paper is to reduce the degree of human interpretation required to understand an interaction protocol. The solution proposed here achieves this by including more detailed information about the actions that participating agents must perform. The use of an associated ontology provides terminology for describing how the messages received and sent by agents are related to each other, and also allows signatures to be defined for the operations that agents must be able to perform in order to use the protocol for its intended purpose. These signatures provide a syntactic specification for the points in the protocol at which the agents must provide their own decision-making and information-processing code, and agent developers could use this to bind internal agent code to these points in the protocol. There is further work to be done to find ways of defining the meaning of these operations so that this binding can be performed on a semantic rather than syntactic basis. This will provide the ability for agents to engage in previously unknown interaction protocols by interpreting the specifications of the protocol and its associated ontologies.

References

[1] C. Atkinson and T. Kühne. Processes and products in a multi-level metamodelling architecture. *International Journal of Software Engineering and Knowledge Engineering*, 11(6):761–783, 2001.

[2] Beanshell: Lightweight scripting for Java. http://www.beanshell.org, 2003.

[3] Grady Booch, Ivar Jacobson, and James Rumbaugh. *The Unified Modeling Language User Guide.* Addison-Wesley, 1998.

[4] C. Castelfranchi and W. L. Johnson, editors. *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002).* ACM Press, 2002.

[5] R. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Using colored Petri nets for conversation modeling. In Dignum and Greaves [7], pages 178–192.

[6] S. Cranefield and M. Purvis. A UML profile and mapping for the generation of ontology-specific content languages. *Knowledge Engineering Review*, 17(1):21–39, 2002.

[7] F. Dignum and M. Greaves, editors. *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Artificial Intelligence*. Springer, 2000.

[8] Eclipse modeling framework. http://www.eclipse.org/emf, 2004.

[9] L. Ehrler and S. Cranefield. Executing agent UML diagrams. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*. ACM Press, 2004. To appear.

[10] D. Fensel, M. Crubezy, F. van Harmelen, and M. I. Horrocks. OIL & UPML: A unifying framework for the knowledge web. In *Proceedings of the Workshop on Applications of Ontologies and Problem-Solving Methods, 14th European Conference on Artificial Intelligence (ECAI 2000)*, 2000. http://delicias.dia.fi.upm.es/WORKSHOP/ECAI00/14.pdf.

[11] Tim Finin, Yannis Labrou, and James Mayfield. KQML as an agent communication language. In J. M. Bradshaw, editor, *Software Agents*. MIT Press, 1997. Also available at http://www.cs.umbc.edu/kqml/papers/kqmlacl.pdf.

[12] FIPA Modeling Technical Committee Web site. http://www.auml.org/auml/modelingtc, 2004.

[13] R. A. Flores and R. C. Kremer. To commit or not to commit: Modelling agent conversations for action. *Computational Intelligence*, 18(2):120–173, 2002.

[14] Foundation for Intelligent Physical Agents. FIPA request interaction protocol specification, version F. http://www.fipa.org/specs/fipa00026, 2001.

[15] Foundation for Intelligent Physical Agents. FIPA ACL message representation in string specification. http://www.fipa.org/specs/fipa00070, 2002.

[16] Foundation for Intelligent Physical Agents. FIPA interaction protocol library. http://www.fipa.org/repository/ips.html, 2002.

[17] M. Greaves, H. Holmback, and J. Bradshaw. What is a conversation policy? In Dignum and Greaves [7], pages 118–131.

[18] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume 1: Basic Concepts*. Monographs in Theoretical Computer Science. Springer, 1992.

[19] F. Lin, D. H. Norrie, Weiming Shen, and Rob Kremer. A schema-based approach to specifying conversation policies. In Dignum and Greaves [7], pages 193–204.

[20] J. Lind. Specifying agent interaction protocols with standard UML. In M. Wooldridge, G. Weiß, and P. Ciancarini, editors, *Agent-Oriented Software Engineering II*, volume 2222 of *Lecture Notes in Computer Science*, pages 136–147. Springer, 2002.

[21] S. Ling and S. W. Loke. A formal compositional model of multiagent interaction. In Castelfranchi and Johnson [4], pages 1052–1053.

[22] J. Martin and J. J. Odell. *Object-Oriented Methods: A Foundation*. Prentice Hall, Englewood Cliffs, NJ, UML edition, 1998.

[23] H. Mazouzi, Amal El Fallah Seghrouchni, and S. Haddad. Open protocol design for complex interactions in multi-agent systems. In Castelfranchi and Johnson [4], pages 517–526.

[24] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4), 1989.

[25] M. Nowostawski, M. Purvis, and S. Cranefield. A layered approach for modelling agent conversations. In *Proceedings of the 2nd International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, 5th International Conference on Autonomous Agents*, 2001. http://www.cs.cf.ac.uk/User/O.F.Rana/agents2001/papers/06_nowostawski_et_al.pdf.

[26] M. Nowostawski, M. Purvis, and S. Cranefield. Modelling and visualizing agent conversations. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 234–235. ACM Press, 2001.

[27] Object constraint language library. http://www.cs.kent.ac.uk/projects/ocl, 2003.

[28] J. J. Odell, H. Van Dyke Parunak, and B. Bauer. Representing agent interaction protocols in UML. In Paolo Ciancarini and Michael Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 121–140. Springer, 2001. (Draft version at http://www.auml.org/auml/working/Odell-AOSE2000.pdf).

[29] Opal agent platform. http://sourceforge.net/projects/nzdis, 2004.

[30] D. Poutakidis, L. Padgham, and M. Winikoff. Debugging multi-agent systems using design artifacts: the case of interaction protocols. In Castelfranchi and Johnson [4], pages 960–967.

[31] M. Purvis, S. Cranefield, M. Nowostawski, and D. Carter. Opal: A multi-level infrastructure for agent-oriented software development. Discussion Paper 2002/01, Department of Information Science, University of Otago, PO Box 56, Dunedin, New Zealand, 2002. http://www.otago.ac.nz/informationscience/publctns/complete/papers/dp2002-01.pdf.gz.

[32] M. Purvis, M. Nowostawski, S. Cranefield, and M. Oliveira. Multi-agent interaction technology for peer-to-peer computing in electronic trading environments. In G. Moro, C. Sartori, and M. Singh, editors, *Proceedings of the 2nd International Workshop on Agents and Peer-to-Peer Computing, 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 103–114, 2003.

[33] M. Purvis, M. Nowostawski, M. Oliveira, and S. Cranefield. Multi-agent interaction protocols for e-business. In *Proceedings of the 2003 IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003)*, pages 318–324. IEEE Press, 2003.

[34] M. K. Purvis, S. J. S. Cranefield, M. Nowostawski, and M. A. Purvis. Multi-agent system interaction protocols in a dynamically changing environment. In T. Wagner, G. Vouros, and S. Smith, editors, *Proceedings of the Workshop – Toward an Application Science: MAS Problem Spaces and their Implications to Achieving Globally Coherent Behavior, 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2002.

[35] M. K. Purvis, P. Hwang, M. A. Purvis, S. J. Cranefield, and M. Schievink. Interaction protocols for a network of environmental problem solvers. In *Proceedings of*

the 2002 iEMSs International Meeting: Integrated Assessment and Decision Support (iEMSs 2002), volume 3, pages 318–323. The International Environmental Modelling and Software Society, 2002. http://www.iemss.org/iemss2002/proceedings/pdf/volume%20tre/214_purvis.pdf.

[36] J. B. Warmer and A. G. Kleppe. *The Object Constraint Language: Getting your models ready for MDA*. Addison-Wesley, 2nd edition, 2003.

Stephen Cranefield, Martin Purvis, Mariusz Nowostawski
Department of Information Science
University of Otago
PO Box 56
Dunedin, New Zealand
e-mail: {`scranefield,mpurvis,mnowostawski`}`@infoscience.otago.ac.nz`

Peter Hwang
Knowledge Engineering and Discovery Research Institute
AUT Technology Park, 581-585 Great South Road
Penrose, Auckland, New Zealand
e-mail: `peter.hwang@aut.ac.nz`

On the Impact of Ontological Commitment

Marian H. Nodine and Jerry Fowler

Abstract. Ontological commitment, or the agreement to have your applications and users conform to a common domain understanding as encapsulated in one or more shared ontologies, is a noble goal and essential for open agent systems. Our experiences building ontology-based agent systems in multiple domains have shown us that the intention for a new application to locate and conform to some existing ontology or ontologies within its domain has many impediments to its success. For instance, the goals of the designer of a domain ontology include developing a complete and comprehensive domain description; however, the application developer may only require a small fragment of that ontology. Multiple applications that conform to the ontology may, in fact, use completely orthogonal fragments of the ontology, and not be able to interact at all. Users may insist on importing into the ontology sets of terms that are neither logically consistent nor easily modelable.

With these issues in mind, we propose here some guidelines for ontology development and evolution that should facilitate ontology reuse. These guidelines could underpin a *usage model* for ontologies; one that enables the application designer to reuse ontological concepts from multiple ontologies in a more flexible manner, while retaining the essentially good properties of ontology sharing and reuse. These guidelines affect both the design and use of ontology-based applications, as well as the way applications advertise themselves to other agents with which they may interoperate.

Mathematics Subject Classification (2000). 68U35 Information Systems.

Keywords. Ontologies, ontological commitment.

1. Introduction

The goal of knowledge representation is to make explicit the semantics of a particular domain of interest for the purposes of sharing the knowledge among humans and computer artifacts. Sowa [21] subdivides knowledge representation into categories:

EDEN was funded jointly by the DOD, DOE and EPA.

Logic provides the formal structure and rules of inference.
Ontology defines the kinds of things that exist in the application domain.
Computation supports the applications that distinguish knowledge representation from pure philosophy."

There is a strong relationship between some specific ontology and the logical rules and computational artifacts that use that ontology, in that when they communicate among themselves, they have some level of assurance that the same terms have the same meanings to all. However, this use requires that the logical rules and the computational artifacts have explicit linkages with the ontology; often in the form of hard-coding the ontological terms into the rules and/or the application code itself.

In an agent-based system, common ontologies specify the ontological commitments of a set of participating agents [10]. An ontological commitment is an agreement to use a vocabulary in a way that is consistent with an ontology. An agent or human committed to an ontology understands (some subset of) the ontology and agrees to use it in a manner consistent with the semantics of the ontology. Agents and humans committed to the same ontology can share knowledge among themselves with some confidence that they share an underlying understanding of what is being said. Commitment to common, shared ontologies facilitates openness in an agent-based system.

We examine the conflicting requirements and goals of ontology designers, ontology-committed applications, and ontology-aware users, and their respective impact on the problem of ontology commitment and reuse. This conflict is evident, but unresolved, in the guide for the OWL Web Ontology Language [24], which states at one point, "... the development of an ontology should be firmly driven by the intended usage", and at another, "in order for ontologies to have the maximum impact, they need to be widely shared". The first statement implies that application designers and users need to have an impact on the development of an ontology, and the second implies that application designers and users should use pre-existing ontologies, which were designed without considering their needs. As ontological sharing and reuse increases, the gap between the ontology designers and the ontology users grows larger.

Our goal is to analyze what issues inhibit reuse and to propose strategies for facilitating reuse. In particular, we consider the problem of reuse of ontologies whose specification is complete, for applications whose requirements were not considered during the design of the ontology. This problem is not addressed in the ontology design methodologies summarized in [8]. We develop guidelines and approaches for agents to use existing ontologies in a more flexible manner.

We seek to relate the issues to real issues we have encountered within the context of one of our applications, EDEN [5]. EDEN is an agent-based system developed for the purpose of inter-organizational sharing of environmental data collected, stored and monitored by multiple government agencies and non-government scientists spread throughout the US and Europe, and relating information from these disparate data sources and schemas at a semantic level as needed by the users. EDEN uses ontologies to represent the semantics of the underlying information in several large, diverse production databases to populate those ontologies with instances.

2. Ontological Commitment

Because ontologies are meant to facilitate sharing and reuse of knowledge, it is important that the ontology and its collection of users (both human and agent) align themselves to a shared view of the domain during the process of designing and evolving the ontology for that domain. However, many existing ontologies have been developed either by designers attempting to characterize a domain (with no real computational applications that use them) or by application developers to support individual applications (with no real sharing of the ontology with other applications). The plethora of existing ontologies argues that many concepts are already represented within some ontology, so reuse of these ontologies, increasing the ontological commitment level, is now feasible. For example, some ontologies such as the Unified Medical Language System (UMLS) Metathesaurus [14] have achieved a higher level of commitment.

There are several issues that impede this sharing and knowledge. First are issues related to the conflicting goals and objectives of ontology definers, ontology-committed applications, and users of such applications. Second are issues of the mutual conflict between ontological commitment and ontological evolution. Third are issues of conceptual mismatches between ontologies and the applications and users that use them. Unfortunately, these issues stem from fundamental issues and characteristics of the problem of sharing ontologies so broadly. We discuss the first two issues in the remainder of this section, and the third (which has a more evident impact on the application designers and users) in a later section, "Impedance Mismatch".

2.1. Conflict: Definer, Application, User

The commitment to ontologies is hampered by the conflicting goals of ontology definers, developers of ontology-committed applications and ontology-committed application users, and often by the confusion of users and definers over the demands of ontology-committed applications. Here, we use the adjective "ontology-committed" to mean that something purports to use the terms in the ontology(-ies) in a manner consistent with its (their) definitions.

The goal of the *ontology designer*, working towards maximizing the usefulness of his ontology to a wide variety of applications, is to completely characterize a particular domain at the semantic level. The ontology designer needs expertise in knowledge representation and in the domain of the ontology. His intent is to develop a comprehensive and up-to-date ontology, with a broad set of acceptable