Engineering Human Trust in Mobile System Collaborations

Licia Capra*
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
L.Capra@cs.ucl.ac.uk

ABSTRACT

Rapid advances in wireless networking technologies have enabled mobile devices to be connected anywhere and anytime. While roaming, applications on these devices dynamically discover hosts and services with whom interactions can be started. However, the fear of exposure to risky transactions with potentially unknown entities may seriously hinder collaboration. To minimise this risk, an engineering approach to the development of trust-based collaborations is necessary. This paper introduces hTrust, a human trust management model and framework that facilitates the construction of trust-aware mobile systems and applications. In particular, hTrust supports: reasoning about trust (trust formation), dissemination of trust information in the network (trust dissemination), and derivation of new trust relationships from previously formed ones (trust evolution). The framework views each mobile host as a self-contained unit, carrying along a portfolio of credentials that are used to prove its trustworthiness to other hosts in an ad-hoc mobile environment. Customising functions are defined to capture the natural disposition to trust of the user of the device inside our trust management framework.

1. INTRODUCTION

Portable devices, such as palmtop computers, mobile phones, personal digital assistants, digital cameras and the like, have gained wide popularity. Their computing capabilities are growing quickly, while their size is shrinking, allowing their pervasive use in our everyday life. Wireless networks of increasing bandwidth allow these mobile units to aggregate and form complex distributed system structures, as well as to seamlessly connect to fixed networks while they change location. The combined use of these technologies enables people to access their personal information, as well as public resources and services *anytime* and *anywhere*.

*The author has been partly funded by the TAPAS European Project (IST-2001-34069).

Each time an interaction takes place, we face an inherent risk, as we can never be certain of the *trustworthiness* of the entities we interact with, or that mediate the interaction. This risk is not peculiar to mobile settings only, but it characterises any distributed setting, that is, any situation where we are giving up complete control, thus becoming vulnerable to somebody's else behaviour. For example, during an e-commerce transaction, we trust the service provider will not divulge or misuse our credit card details in any way; when downloading and executing a piece of software, we trust it will not harm our device, and so on. In mobile ad-hoc settings, however, the perceived risk is higher, because of the easiness with which services and information can be accessed, the anonymity of the entities we interact with, the speed at which new entities come into reach while others disappear, and so on.

In order to advance the goal of anywhere-anytime computing, and to fully exploit the potential of current technologies to promote non-trivial interactions among entities, the exposure to risky transactions has to be reduced as much as possible. An engineering approach to the development of trust-based collaborations is thus necessary. This requires the existence of a *trust management framework* (TMF) that enables devices to form, maintain and evolve trust opinions. These opinions are of fundamental importance as they can be used to drive the configuration of the system in a variety of ways: for example, to decide from where to download a file, what service provider to contact, what access rights to grant, and so on. Trust is obviously not the only aspect that must be taken into account when making these decisions: the perceived risk inherent to a transaction, and the quality of service (QoS) requirements, will all contribute to the final configuration decisions. For example, low risk transactions can still take place in untrusted environments, while high risk transactions may not take place even in highly trusted environments. Also, a transaction that requires the investment of large amounts of resources (e.g., bandwidth and battery) to be carried on in a low trusted environment may be blocked, because of QoS constraints. Feelings of trust, risk and QoS can be formed independently of each other, and thus dealt with separately, before being combined. In this paper, we are concerned with trust management only.

In traditional distributed systems, trust decisions were mainly centralised: the existence of clearly-defined administrative boundaries, and the limited mobility of entities, allowed a trusted third party (TTP) (e.g., the local administrator) to

store information about the entities belonging to that domain; the TTP was then contacted when a trust decision had to be made (for example, to establish the access rights of an entity to a resource). This trust management model is based on assumptions that do not hold in the mobile setting: first, a globally available infrastructure that holds trust information is missing, and thus the centralised approach is inapplicable. Second, while entities are mostly fixed and known in a centrally administered distributed setting, entities are dynamic and anonymous in mobile settings. In the first case, knowing an entity usually coincides with trusting an entity; in the second case, we often find ourselves having to make a trust decision about entities we have never seen, or about whom we have only partial knowledge. Simply distrusting the unknown would cut down the possibility to engage fruitful interactions; on the other hand, we cannot blindly trust everyone, as anonymity is very appalling to malicious entities against which we want to be protected. To foster the vision of the mobile device as an 'extension' of the human being, and to promote complex and safe interactions in the mobile and pervasive scenarios, a human-tailored trust management model and framework have to be developed that programmers can use to build *trust-aware* systems and applications.

A trust management model for mobile ad-hoc systems must be *subjective*, that is, it must enable the (user of the) mobile application to form its trust opinions; delegating trust decisions to an external entity (the TTP) would in fact inevitably lose the individuality that is the essence of human trust. Decentralised approaches have been proposed, where trust decisions are locally made based on recommendations that are spread across the network via recommendation exchange protocols. However, current approaches fail to be fully satisfactory for the following two reasons: first, they completely rely on the assumption that entities have a social conscience that will make them exchange trust information whenever asked, although no incentives are provided to induce entities to do so. In a resource constrained environment, *selfishness* is likely to prevail over cooperation, for example, to save battery power. Second, the trust decision that each entity locally makes tends to be fully automated, with very little or no customisation allowed. We argue that a trust management model should be highly *customisable* instead, to capture the varying and complex natural disposition of an individual to trust into computer models. This should be achieved without causing disruption to the device computation and communication resources.

Our research goal is to develop a formal abstract treatment of trust that meets the above requirements, and offer programmers an infrastructure they can use to build trust-aware systems. In this paper, we advance this goal by introducing *hTrust*, a human-based trust management model and framework that a mobile system can exploit to form trust opinions, without the burden of, for example, keeping trust information updated, or propagating trust information in the network. The model is completely decentralised: each host acts as a self-contained unit, carrying along a *portfolio of credentials* derived from its past interactions, and that it uses to prove its trustworthiness to others. *Customising functions* are used to capture the natural disposition to trust of the user of the device, thus enabling human-tailored trust reasoning. Although dangers are an intrinsic part of mobile settings, and thus cannot be completely eliminated, our trust management model exploits these customising functions to dynamically detect malicious behaviours, and consequently isolate untrusted entities from future interactions.

The paper is further structured as follows: in Section 2 we discuss strengths and limitations of current approaches to trust management. Section 3 provides a definition of trust and spells out the principles and assumptions our model is based upon. In Section 4, we describe our trust management model in details, and in Section 5 we discuss its suitability to the mobile setting. Finally, Section 6 concludes the paper and illustrates some future works.

2. RELATED WORK

Trust comes into play in any computational interaction, that is, in any situation where full control is given up, and we have to rely on the behaviour of someone else to complete a task. The problem of trust management is thus very broad and it has been dealt with by different research communities. However, a common limitation to many approaches is that they deal with only a very narrow subset of the overall trust management problem, or they provide solutions that are not consistent with human intuitions of trust (that is, they fail to capture various facets of human trust).

In distributed systems, the issue of trust has often been regarded as how to increase the client's trust in the behaviour of the server component. Signatures have been proposed to convey trust in the code (e.g., Signed Java Archives for Java and Authenticode for ActiveX); however, a signature can only convey trust in the identity of the signer: whether the signer is trusted or not, is an entirely different problem. Similar considerations apply to public key certificates [13, 2, 18, 7]) that aim to solve the problem of authentication in distributed settings; however, a signed public key certificate does not tell you whether the owner is a straight person or a crook. To increase the client's confidence in the correctness of third party component's implementations, various solutions have been advanced. Proof-carrying code techniques [20] have been proposed to foster the acceptance of mobile agents; when the component implementation is inaccessible, and thus independent verification cannot be performed, approaches based on run-time component interface violations have been suggested (e.g., [9, 16]). The computational overhead that these approaches impose is, however, unbearable for mobile devices and thus can only be applied to traditional distributed systems. More importantly, none of these approaches say much about the wider notion of an entity's trustworthiness: what trust is made of, how it can be formed, how it evolves. A wider approach to the problem of trust management is thus necessary, to enable mobile systems and applications to dynamically reason about trust.

Sultan [15] is a trust management framework that allows the specification, analysis and management of trust relationships. Its functioning is based on a central specification server where trust information is stored and used both for decision making and analysis. Although well-suited for use by the system administrator, its applicability to the mobile distributed setting is thus limited. PolicyMaker [5] takes a distributed approach to the problem of trust management; it

binds keys to actions that the possessor of the key is trusted to do. When asked to determine whether a key is allowed to perform an action, PolicyMaker uses these bindings and a set of locally defined policies to answer. Similar credential-based distributed policy management approaches have been defined (e.g.,[4, 10]); however, issues such as trust evolution and subjective trust reasoning have not been tackled.

In [1], a trust management model is proposed to give autonomous entities the ability to reason about trust, without relying on a central authority. Based on direct experiences and recommendations, each entity is able to derive trust measures, thus being responsible for its own fate. The approach relies on the assumption that entities will behave socially, exchanging recommendations when requested to do so, although no incentives are provided for this to happen. Also, no mechanism to dynamically re-evaluate trust decisions is discussed. In [6], a mechanism to detect and isolate misbehaving nodes at the network (routing and forwarding) level is proposed. The main advantage of the mechanism is that it works even without assuming the cooperativeness of the nodes; however, decisions about what nodes to isolate are performed in a completely automatic and homogeneous way. While this approach may work well at the network level, its lack of subjectivity severely limits its applicability at the application level, where the user's disposition has to be accounted for.

As part of the SECURE project [11, 22], a trust management model has been defined that uses local trust policies to form and dynamically re-evaluate trust, based on past personal observations and recommendations. The computed trust values are then exploited to assess risks involved in the transaction, and then determine what behaviour the entity must adhere to. While moving a step closer to the definition of human trust than previous approaches, details about the local policies and the way they influence trust formation and evolution are missing. Moreover, the issue of malicious behaviours is left behind the scene.

Social control mechanisms have been proposed to automatically isolate malicious entities and exclude them from future interactions, without having to rely on a trusted third party (e.g., [21]). The underlying assumption is the view of a mobile system as an ecologic system [19], where the interaction of the participants determines the success of the individual participant. Although sharing the same basic assumption, approaches developed to date are fairly limited, in that they do not capture a variety of aspects peculiar to human trust; for example, ways to recover from a bad reputation, and natural disposition to trust unknown entities. In this paper, we propose a more comprehensive trust management model that includes these aspects of human trust.

Various formalisms of trust have been proposed, in order to help reasoning about trust relationships. In [17], an opinion model based on subjective logic is discussed that can assign trust values in the face of uncertainty; however, the approach does not describe how to compute these values. In [8], a formal model for trust formation/negotiation, evolution and propagation is presented; however, the protocols for exchanging recommendations and for dynamically re-evaluating trust relationships are not provided. Similar considerations hold for the formal trust models described in [3] (based on probability theory) and [24] (based on lattice, denotational semantics and fixed point theory). In this paper, we tackle the problem of trust management from an engineering point of view, and provide a more operational model that programmers can actually exploit to build trust conscious systems.

3. PRINCIPLES OF TRUST

Prior to describing our trust management model, we define what we mean by *trust*, and spell out some commonly accepted characteristics of trust. Despite extensive studies from sociologists, philosophers, and psychologists, a unique, precise, and universally accepted definition of trust is still missing. One of the most commonly accepted definition, and the one we refer to, is from sociologist Diego Gambetta [12]:

> "... trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action, both before [we] can monitor such action (or independently of his capacity of ever be able to monitor it) and in a context in which it affects [our] own action."

A first important observation is that trust is *subjective*: in particular, it is the degree of belief about the behaviour of other entities, also called 'agents', upon which we depend (for example, to have a service delivered). These beliefs regard both the intentions (not) to cheat, and the skills to perform adequately to intentions. Trust is *asymmetric*, that is, two agents need not have similar trust in each other, and *context-dependent*, in that trust in a specific environment does not necessarily transfer to another. Trust is *dynamic* and it tends to be reduced if entities are misbehaving; viceversa, it increases if agents are doing well. There is no absolute definition of what 'doing well' means, and therefore different observers may have different perceptions of the same agent's trustworthiness.

A trust management framework thus characterises as a self-adjusting system used to form, exchange and evolve trust information about the agents that populate the network. We refer to this network as to the *social context*, in order to distinguish it from the *transactional context*, that is a network of producers/consumers that interact to deliver services. A transactional context uses the trust information available from the social context in order to customise the way transactions take place, that is, to configure the system. As pointed out before, risks and QoS issues must also be taken into account when customising a transaction. In this paper, we are not concerned with how these three parameters, that is, trust, risk and QoS, can be combined to dynamically configure the system. Even before this can happen, trust opinions about other agents must be formed. In this paper, we define such a trust management framework.

Closely related to trust management is the issue of identification: we must be able to bind a trust opinion to an identity; however, the nature of mobile settings is such that creation and deletion of identities is very quick and easy, and malicious agents could exploit it to repeatedly misbe-

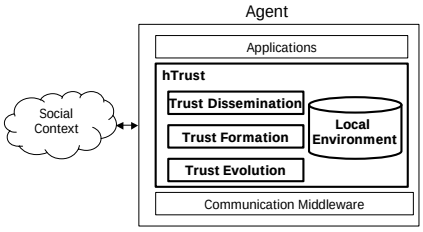

Figure 1: Trust Management Model Overview.

have, without ever being isolated. Authentication systems based on public-key cryptography (e.g., [7, 18]) do not solve the issue, as there is nothing there preventing an agent to have multiple pairs of keys, thus multiple identities, so that misbehaviors related to one identity cannot be traced back to another. We do not try to address the issue of identification in our trust management framework. We assume each agent has got a pair of public/private keys (perhaps more than one), that is managed via an independent public-key management system (we do not bind ourselves to any in particular). However, based on this assumption, our TMF aims to detect potential cheaters (i.e., agents that maliciously create new public/private keys to conceal past misbehaviors), and alert the application about their presence.

4. HTRUST

An overview of the trust management model we have developed is depicted in Figure 1. The model comprises three core components: *trust dissemination*, *trust formation*, and *trust evolution*. Whenever an agent a, called the *trustor*, has to decide whether to trust another agent b, called the *trustee*, trust information about b has to be collected. In our model, sources of trust information are both direct experiences and recommendations. Direct experiences represent the agent's history; the trust management framework keeps them in the agent's *local environment*. Recommendations come from other agents in the social context; the TMF propagates them by means of the trust dissemination component. Trust information is then processed by the trust formation component, to predict the trustee's trustworthiness. As previously discussed, the decision of whether to interact or not depends on both the trust opinion just formed, the risks involved in the interaction, and the QoS requirements at stake. Assuming the interaction takes place, feedback about b's trustworthiness, as perceived by a, is given in input to the trust evolution component, whose goal is to keep a's local environment updated.

In the following sections, we describe the core components of our trust management framework, that is, trust formation, trust dissemination and trust evolution. As the picture shows, these components sit in between the applications and a communication middleware that enables the agent to interact with other agents in the system. The goal of the TMF is to take the burden of maintaining and propagating trust information away from the application; at the same time, subjective reasoning is made possible by means of a number of customising functions that capture the human disposition to trust of the entity involved in the trust decision process (i.e., the user of the device). These functions will be discussed in the following sections when encountered. While the discussion proceeds, we will also incrementally describe what information constitutes an agent's local environment.

4.1 Trust Formation

We call *trust formation* the process that enables a trustor agent to predict a trustee's trustworthiness before the interaction takes place. In this section, we describe both the information that the TMF uses to predict the trustworthiness of an agent, and the trust formation function that the TMF provides to compute a prediction.

4.1.1 Trust Data Model

A trustor a forms a trust opinion about a trustee b based on: *aggregated trust information*, that is, trust information locally kept by the TMF and mainly based on a's past direct experiences with b (i.e., interactions happened in the past between a and b in their transactional context); and *recommendations* sent to a by other agents in the social context, who have interacted with b in the past.

Aggregated trust information is kept in the trustor's local environment as a set of tuples:

$$[a,\ b,\ l,\ s,\ c,\ k,\ t],$$

meaning that agent a trusts agent b at level l to carry on service s in context c. The trust level l varies in a range $[-1, 1]$, with -1 meaning total distrust, and 1 meaning blind trust. The trust level will be higher if interactions happened in the transactional context have been positive experiences, and viceversa. Because in mobile ad-hoc settings agents can have only a partial knowledge of their surroundings, their trust opinions contain a level of uncertainty. In order to distinguish between 'don't trust' (i.e., trust-based decision) from 'don't know' (i.e., lack of evidence), we explicitly model the degree of knowledge k in the trust opinion expressed. This knowledge varies in a range $[0, 1]$, with 0 meaning unknown, and 1 meaning perfect knowledge; the higher the number of direct experiences happened between the trustor and the trustee, the higher the degree of knowledge. The distinction between trust and knowledge has been explicitly made in [8] first. However, they do not model a third important parameter, that is time. The trustor's knowledge k decays with time; we thus associate, to each tuple, a timestamp t, indicating at which time the knowledge k refers to. In the reminder of the paper, we are only interested in the parameters l, k and t; we thus simplify the notation and describe an aggregated trust opinion made by agent a about b as $o = [a, b, l, k, t] \in \mathcal{O}$, $\mathcal{O}$ being the set of all trust opinions. We will discuss in Section 4.3 how the TMF on each agent maintains aggregated information updated.

Recommendations have a very similar structure to the one presented above. The main difference is that each recommendation is signed with the recommender's private key, in order to prove its authenticity. For example, a recommendation sent by agent x about agent b would look like:

$$[x,\ b,\ l,\ s,\ c,\ k,\ t]_{SK_x}$$

or, simply, $[x, b, l, k, t]_{SK_x}$. We refer to x and b as to the agent's names; they are the piece of information that is publicly attached to their public key through the public-key management infrastructure. A recommendation is thus

simply computed by signing the local aggregated trust tuple; we will discuss in Section 4.2 the protocol used by the TMF to exchange recommendations across the social context. Recommendations are used to form a trust opinion in two circumstances: to predict the trustee's behaviour in case the trustor has never interacted with him before (i.e., there is no aggregated trust information to base on); and to (partially) rely on third-party assessments to form a trust opinion (trust delegation). For example, an agent a may be willing to interact with agent b provided that he has been recommended by agent x (i.e., a delegates to x the formation of a trust opinion about b).

Not all recommendations are used to predict trust. In human interactions, we tend to weigh more recommendations coming from people who have given us good recommendations in the past (i.e., people with whom we shared opinions), while discarding recommendations coming from unknown recommenders, or from recommenders with whom we have divergence of opinions. Information about the trustworthiness of an agent as a recommender is thus kept in an agent's local environment as a set of tuples:

$$[a,\ x,\ l,\ s,\ c,\ k,\ t].$$

We refer to these tuples as to tacit information, as the TMF tacitly extracts them from observations of direct experiences using a process that will be described in Section 4.3. The interpretation of this tuple is the same provided for aggregated trust tuples: a trustor a trusts agent x at level l to provide recommendations (service s) in a certain context c (e.g., you may trust x on recommendations about restaurants but not about what stock market to invest in). The trustor has knowledge k at time t about this information. The higher the number of good recommendations received by x in the past, the higher the trust level, and viceversa. Also, the higher the number of recommendations received from x, the better the knowledge of x; as before, this knowledge decays with time, and we thus have to record at what time the information contained in the tuple refers to. To simplify the discussion, we will refer to a recommender's trust tuple as $r = [a, x, l, k, t] \in \mathcal{R}$, $\mathcal{R}$ being the set of all trust opinions about recommenders.

4.1.2 Trust Formation Function Υ

A trustor a willing to predict the trustworthiness of a trustee b, uses the trust formation function Υ our framework provides. A formal definition of Υ can be found in Figure 2. Given in input an aggregated trust information tuple $o \in \mathcal{O}$, taken from a's local environment $e \in \mathcal{E}$, and a set of recommendations $O \in \wp(\mathcal{O})$ coming from the social context, the trust formation function Υ returns a *range* of predicted trust values. We represent the prediction as an interval, rather than as a single value, to cater for the approximate nature of trust due to incomplete knowledge. It is up to the mobile application to decide what value to use from this range.

First, given the aggregated local tuple $o = [a, b, l, k, t]$, a predicted trust value f is computed, based on the past trust level l, the trustor's knowledge k, and the time elapsed since the last time the tuple o was updated with fresh information. T is a parameter belonging to the trustor local environment e, and represents the time interval during which interactions are observed. A predicted trust range is then derived, based on the trust level l experienced in the past, and the discrepancy between this value and the prediction f (function Υ_{op} - prediction based on one trust opinion).

Second, a predicted trust range is computed based on the recommendations received from the social context (function Υ_{rec} - prediction based on m recommendations). For each recommendation $o_i \in O$, the function Υ_{op} is used to derive a predicted trust range; a new lower limit l_{low} and upper limit l_{high} are then computed as a weighed average of the individual ranges lower and upper limits. These weights are computed taking into consideration the recommender i's trustworthiness l'_i, the trustor's knowledge k'_i about the recommender, and the time elapsed since when i's last recommendation was received. The tacit information tuple $r'_i = [a, i, l'_i, k'_i, t'_i]$ is retrieved from a's local environment e using an auxiliary function $lookup(i, e)$. Note that not all recommendations are used to compute a prediction: only recommendations coming from recommenders whose *quality* q_i is greater than a minimum level η contribute to the final result. Also, to avoid considering the same recommendation more than once, we assume that only recommendations with timestamp $\pi_{time}(o_i) > \pi_{time}(r_i)$ are processed (π_{time} is an auxiliary function used to project a tuple onto its timestamp field, that is, $\pi_{time}([x, y, l, k, t]) = t$).

Finally, the trust formation function Υ is used to derive a predicted range of trust values, based on the two ranges previously computed. We make use here of a *customising function* h_1 that synthesises a trust range, given two different trust ranges. This function will vary from agent to agent, and depending on the natural disposition to trust of the agent itself. For example, h_1 may be chosen to consider the local aggregated trust information only, thus relying solely on the trustor's past experiences (*trust reflexivity*); viceversa, recommendations alone can be used (*trust transitivity*), for example, when the trustor has no previous knowledge of the trustee. More generally, a combination of the two will be used:

$$h_1([l_1, l_2], [l'_1, l'_2]) = [\ h_2([l_1, l_2]) - |h_2([l_1, l_2] - h_2([l'_1, l'_2])|,\ h_2([l_1, l_2]) + |h_2([l_1, l_2] - h_2([l'_1, l'_2])|\ ]$$

where h_2 is another customising function used to compute a trust value out of a trust range. For example, a cautious agent that tends to distrust other agents may choose $h_2(l_1, l_2) = l_1$ (that is, the lower trust value of a range); another agent who naturally tends towards trust may choose $h_2(l_1, l_2) = l_2$ instead. More generally, $h_2(l_1, l_2) = w_1 * l_1 + w_2 * l_2$, with $0 \leq w_1, w_2 \leq 1$, $w_1 + w_2 = 1$; the weights w_1 and w_2 are chosen depending on the importance given to trust reflexivity (w_1) and to trust transitivity (w_2).

4.2 Trust Dissemination

The trust formation function described before uses recommendations to predict the trustworthiness of a trustee b; these recommendations become particularly important when b is unknown to the trustor a; a protocol for the dissemination of recommendations is thus necessary. In this section, we describe a recommendation exchange protocol that guarantees a a minimum set information upon which to base the prediction, even in the case agents are selfish and, having to decide whether to answer a request for recommendations, or to save battery power, choose the latter.

$$
\begin{aligned}
\Upsilon_{op} &: \mathcal{O} \to \mathcal{E} \to [-1,1] \times [-1,1] \\
\Upsilon_{op}[\![[a,b,l,k,t]]\!]_e &= [\max(-1, l-|l-f|), \min(l+|l-f|,1)] \ , \quad f = l*k*\max\left(0, \frac{T-(t_{now}-t)}{T}\right) \\
\Upsilon_{rec} &: \wp(\mathcal{O}) \to \mathcal{E} \to [-1,1] \times [-1,1] \\
\Upsilon_{rec}[\![\{o_i | i \in [1,m]\}]\!]_e &= [l_{low}, l_{high}] \ , \\
& l_{low} = \frac{\sum_i \{\pi_1(\Upsilon_{op}[\![o_i]\!]_e) * q_i | q_i > \eta\}}{\sum_i (q_i | q_i > \eta)}, \quad l_{high} = \frac{\sum_i \{\pi_2(\Upsilon_{op}[\![o_i]\!]_e) * q_i | q_i > \eta\}}{\sum_i (q_i | q_i > \eta)} \\
& q_i = \max(\eta, l_i' * k_i' * \max\left(0, \frac{T-(t_{now}-t_i')}{T}\right)) \ , \quad r_i' = lookup(i,e) = [a, i, l_i', k_i', t_i'] \\
\Upsilon &: \mathcal{O} \times \wp(\mathcal{O}) \to \mathcal{E} \to [-1,1] \times [-1,1] \\
\Upsilon[\![(o,O)]\!]_e &= h_1(\Upsilon_{op}[\![o]\!]_e, \Upsilon_{rec}[\![O]\!]_e)
\end{aligned}
$$

Figure 2: Trust Formation Function Υ. The following auxiliary functions are used: $\pi_1(l_1, l_2) = l_1$; $\pi_2(l_1, l_2) = l_2$

4.2.1 Exchange of Recommendations

We assume that each trustee agent b carries with him (i.e., in his local environment) a *portfolio of credentials*, that is, a set of letters of presentation that represents the history of the agent itself. Each letter contains the following information:

$$[x,\ b,\ l,\ s,\ c,\ k,\ t]_{SK_x}$$

This letter has to be interpreted as done before for recommendations: agent x says he trusts b at level l to carry on service s in context c; at time t, x was confident in the trust opinion expressed at degree k. The letter of presentation is authentic, that is, it can only come from x, as it has been signed with his private key SK_x.

Whenever a trustor agent a has to form a trust opinion about a trustee agent b, and a cannot rely solely on the aggregated trust information the TMF has kept locally (for example, because there is no such information, or because it is too outdated), the TMF runs the following protocol.

Step 1. $a \to b$: $req\_for\_credentials(m)$. The protocol starts with a sending b (notation $a \to b$) a request to see his portfolio of credentials; the parameter m indicates the maximum number of letters a is willing to receive from b.

Step 2. $b \to a$: $(o_i)_{SK_i},\ i \in [1,m]$. The trustee b replies with a set of at most m letters of presentation (the ones he considers to be the best for his own reputation).

Step 3. The TMF decrypts the letters of presentation received, relying on the public-key infrastructure to receive the public-keys of the agents that have signed the letters. The local trust formation function Υ is then used to form a trust opinion about b, based on the information contained in the decrypted letters. However, this information may not be enough, leaving the predicted value still not satisfactory. In this case, the TMF queries the social context to receive further recommendations about b. Note that this request may bring no additional information to a, in case agents in the social context do not reply. No more information can now be collected; the function Υ is used one last time to synthesise a predicted trust interval.

Step 4. At this point, interaction between a and b may or may not take place in their transactional context; this does not only depend on the result of the trust formation function, but also on risks and QoS needs related to the specific transaction. In case the interaction takes place, our protocol demands that, upon its completion, a and b exchange a letter a presentation: $a \to b : [a,\ b,\ l',\ k',\ t]_{SK_a}$, and $b \to a : [b,\ a,\ l'',\ k'',\ t]_{SK_b}$. The tuple signed by b becomes a letter of presentation for a and vice-versa. This is a far less demanding assumption than requiring an agent to answer all requests for recommendations coming from neighbors.

Note that a can read what b has written in the letter of presentation by decrypting it with PK_b. In this case, if b gives negative feedback to a, a can decide to discard this letter of presentation and not include it in its portfolio. We do not consider this concealment to affect our protocol for the following reasons: if a repeatedly misbehaves, and thus keeps discarding the letters of presentation received, he will not build a portfolio to supply to other trustor agents in order to make himself trusted. Occasional concealments are harder to be discovered; however, the more recommendations are collected from the social context, the easier and quicker the discovery of concealed bad reputation letters. We will detail in Section 4.3 how malicious agents can be detected and isolated.

The trust formation function Υ is applied at three different stages of the recommendations exchange protocol: prior to its execution, after the portfolio of credentials has been obtained from the trustee, and then again once (and if) further recommendations have been received from the social context. Every time, the returned result of the Υ function is used to decide whether to proceed with the following step of the protocol or not. This decision depends on a customising function $h_3 : [-1,1] \times [-1,1] \to \{0,1\}$ that, given in input a predicted trust range, decides whether the prediction is accurate enough (return 1), or whether further information should be acquired (return 0). What exactly is *enough*, depends once again on the natural disposition to trust of the agent. Possible examples of this function include: $h_3(l_1, l_2) = 0$ if $l_1 < 0$ and $l_2 > 0$, 1 otherwise; that is, if the lower bound l_1 is negative (i.e., tends towards distrust), while the upper bound l_2 is positive (i.e., tends towards trust), then ask for more trust information. Another possibility is $h_3(l_1, l_2) = 0$ if $l_2 - l_1 > \delta l_{max}$, 1 otherwise; that is, if the range of opinions is too broad, ask for more

$$
\begin{aligned}
\Phi &: [-1,1] \times \wp(\mathcal{O}) \rightarrow \mathcal{E} \rightarrow \mathcal{E} \\
\Phi[\![(\tilde{l},O)]\!]_e &= e \setminus \{[a,b,l,k,t]\} \cup \{[a,b,l',k',t']\} \mid o = lookup(b,e) = [a,b,l,k,t] \ \wedge \\
&\quad l' = h_4(\tilde{l},l,h_2(\Upsilon_{rec}[\![O]\!]_e)) \ \wedge \ k' = \min(k+k_{min},1) \ \wedge \ t' = t_{now} \\
\Phi[\![(\varepsilon,O)]\!]_e &= e \setminus \{[a,b,l,k,t]\} \cup \{[a,b,l',k',t']\} \mid o = lookup(b,e) = [a,b,l,k,t] \ \wedge \\
&\quad l' = h_4(\varepsilon,l,h_2(\Upsilon_{rec}[\![O]\!]_e)) \ \wedge \ k' = k \ \wedge \ t' = \max(t, \max_i(\{\pi_{time}(o_i), o_i \in O\}))
\end{aligned}
$$

Figure 3: Aggregation Function Φ.

information to narrow it down.

Note that, when entering a social context for the first time, an agent has no history, thus no portfolio he can use to prove his trustworthiness to other agents. Having no history is distinctive of both genuine newcomers, but also of cheating agents, that are repeatedly creating new identities to conceal past misbehaviors. To facilitate the start-up of an agent x without past, an agent a, that is already a member of the social context, may send out an *introductory message* $[a,x,l,s,c,0,t]_{SK_a}$ to the community. The interpretation of the message is the same provided for recommendations; however, the knowledge parameter k is set to zero, to warn the community that the trust opinion l is not based on a direct experience but, for example, on the opinion that a formed about x during a physical encounter. It will then depend on a's trustworthiness as a recommender whether, and how quickly, the newcomer will be accepted by the social context. In the absence of such an introductory message, the newcomer may offer (service-specific) incentives to solicit trust, and thus encourage interactions.

4.3 Trust Evolution

As we discussed in Section 4.1, the trustworthiness of a trustee agent is predicted based on his trustworthiness during past experiences, as perceived by the trustor agent. A fundamental component of a trust management framework is thus trust evolution, that is, the continuous self-adaptation of trust information kept by the TMF in the agent's local environment. In this section, we discuss: an *aggregation function* Φ, used to maintain information about the trustworthiness of an agent as a service provider (i.e., aggregated trust information tuple), and a *tacit information extraction function* Ψ, used to maintain information about the trustworthiness of an agent as a recommender (i.e., tacit information tuples). Also, we illustrate how the maintenance of trust information plays a key role in the detection of malicious agents.

4.3.1 Aggregation Function Φ

The TMF of agent a uses the aggregation function Φ to update the perceived trustworthiness of a trustee b when a new direct experience between the two agents occurs. Even in case there is no interaction, the trustworthiness of the trustee may still be updated, based on the recommendations received about b from trusted recommenders; the trust information kept on the mobile device for each agent is thus minimal (a single aggregated tuple). We assume that only recommendations coming from agents with quality $q_i > \eta$ contribute to the newly synthesised trust value (see Figure 2 for a definition of q_i); also, we assume that only fresh recommendations are processed ($\pi_{time}(o_i) > \pi_{time}(r_i)$, o_i being a recommendation in O, and r_i the tacit information tuple related to recommender i in the agent's local environment).

Figure 3 contains a formal definition of the aggregation function Φ. The first equation describes the case where the aggregated tuple $o = [a,b,l,k,t]$, kept in a's local environment e, is updated as a result of an interaction occurred between a and b. The old trust opinion l, held by a prior to this interaction, is replaced by a new value l' that depends on: $\tilde{l} \in [-1,1]$ (that is, b's trustworthiness as perceived by a in the just completed interaction), l, and $h_2(\Upsilon_{rec}[\![O]\!]_e)$ (that is, b's trustworthiness computed using the newly received recommendations O). A customising function h_4 combines this three trust opinions to derive the new one. The general structure of h_4 is the following: $h_4(l_1,l_2,l_3) = \frac{w_1*l_1+w_2*l_2+w_3*l_3}{w_1+w_2+w_3}$, where l_1 corresponds to b's trustworthiness as perceived by a in the just completed interaction, l_2 is the opinion previously held by a about b, and l_3 is b's expected trustworthiness based on the received recommendations. Different choices of the weights w_i correspond to different dispositions to trust of the agent. Examples include: $w_1 = 1, w_2 = 1, w_3 = 0$, that is, equal weight is given to the newly perceived trustworthiness and the old opinion; this reflects a human disposition to change trust opinion fairly quickly. Another example is $w_1 = 1, w_2 = n, w_3 = 0$, with $n = k/k_{min}$, k_{min} representing the increment of knowledge happened in a single transaction; in this case, each of the n experiences happened between a and b has equal weight in computing the new trust l', thus reflecting a more cautious behaviour, not inclined to change opinion rapidly. In both cases, the recommendations are not taken into account, as trust information coming from a direct experience (that is, $\tilde{l}$) is available. Apart from adjusting the trust level, the knowledge a has got about b is increased; also, the timestamp is updated to guarantee the freshness of the information.

The second equation considers the case where trust information about b is updated solely based on the newly received recommendations O, without an interaction to have actually occurred. This is useful, for example, when collecting information about the trustworthiness of other agents with whom there has been no previous interaction. In this case, a new trust opinion is computed solely based on l (i.e., the old trust opinion held by a), and $h_2(\Upsilon_{rec}[\![O]\!]_e)$ (i.e., the predicted trustworthiness computed using the newly received recommendations O). Note that, in this case, the trustor's knowledge is not incremented: only direct experiences contribute to uncertainty reduction.

$$
\begin{aligned}
\Psi &: [-1,1] \times \wp(\mathcal{O}) \rightarrow \mathcal{E} \rightarrow \mathcal{E} \\
\Psi[\![(l', O)]\!]_e &= e \setminus \{r_i = lookup(i,e) = [a,i,l_i,k_i,t_i], \forall o_i \in O\} \cup \{r'_i = [a,i,l'_i,k'_i,t_i], \forall o_i \in O\} \mid \\
& \quad k'_i = \min(k_i + k_{min}, 1) \ \wedge l'_i = \begin{cases} \max(-1, h_5(l_i, \delta l_i)) \text{ if } \delta l_i > \delta_{max} \\ \min(h_5(l_i, \delta l_i), 1) \text{ if } \delta l_i \leq \delta_{max} \end{cases}, \\
& \quad \delta l_i = |l' - h_2(l_i - |\pi_l(o_i) - \pi_l(o_i) * \frac{T - (t_{now} - t_i)}{T}|, l_i + |\pi_l(o_i) - \pi_l(o_i) * \frac{T - (t_{now} - t_i)}{T}|)|
\end{aligned}
$$

Figure 4: Tacit Information Extraction Function Ψ. The auxiliary function π_l has been used to project a tuple $o = [x, y, l, k, t]$ onto the trust value l.

The aggregated tuple can then be signed with the trustor's private key and used at the end of the recommendation exchange protocol to provide the trustee a letter of presentation; also, it is used to answer request for recommendations coming from other agents in the social context.

4.3.2 Tacit Information Extraction Function Ψ

When agent a has to make a trust decision about agent b with whom he has no previous direct experiences, there are only recommendations to rely on. However, as trust is subjective, these recommendations may be conflicting with each other; in these cases, we would like to weigh more recommendations coming from agents we trust as recommenders (i.e., whose recommendations we usually agree with), while weighing less, or even discard, recommendations coming from agents we do not trust (i.e., with whom we do not share opinions). The TMF thus maintains, on behalf of its agent, a set of tuples that assesses the trustworthiness of agents as recommenders. We refer to this set of tuples as tacit information; its content is updated after an interaction has occurred, using a tacit information extraction function Ψ whose formal definition can be found in Figure 4.

The tacit information tuple $[a, i, l_i, k_i, t_i] \in e$, containing information about the trustworthiness l_i of agent i as a recommender, is updated based on: the perceived trustworthiness l' of b with whom a has just interacted, and the recommendation $o_i \in O$ about b that recommender i has given to a. To avoid considering the same recommendation many times, only recommendations o_i with timestamp $\pi_{time}(o_i) > \pi_{time}(r_i)$ are processed. First, the discrepancy δl_i between the observed trustworthiness l' and the opinion l_i provided in the recommendation is computed; the uncertainty that time brings is taken into consideration, as the recommendation may refer to a distant past. A new trust value l'_i for recommender i is then computed, based on both its past trustworthiness l_i and the discrepancy δl_i, using a customising function h_5. An example of h_5 is the following: if the discrepancy of opinions is lower than a tolerance parameter δ_{max}, then i's trustworthiness is increased: $l_i = \frac{n * l_i + \frac{2 - \delta l_i}{2}}{n+1}$, with $n = k_i / k_{min}$. Viceversa, if the discrepancy of opinions is higher than the tolerance, the recommender's trustworthiness is decreased: $l_i = \frac{n * l_i - \frac{2 - \delta l_i}{2}}{n+1}$. In this case, an equal weight is given to every recommendation received from i in the past (cautious change of opinion); another possibility is to weigh the past as a whole and the new recommendation equally (rapid change of opinion), that is, $l_i = \frac{l_i + \frac{2 - \delta l_i}{2}}{2}$. The lower the tolerance, the stricter is the trustor in selecting trustworthy recommenders. To limit the uncertainty of the information processed, as well as the computational complexity of the model, we do not consider recommendations about the trustworthiness of recommenders.

Both the aggregation function and the tacit information extraction function are non-monotone: they adjust the value of an agent's trustworthiness based on its behaviour, so that trust can dynamically be gained (when behaving well) and lost (when misbehaving). Trust information thus keeps evolving throughout the lifetime of an agent: the more frequently the agent interacts, the more accurate the agent's knowledge of the surroundings becomes, and viceversa.

4.3.3 Malicious Agents Detection

In the social context, malicious behaviours refer to the spreading of: *fake bad recommendations*, when a single agent, or a group of agents, start spreading false bad recommendations to damage some other agent; and *fake good recommendations*, when a group of agents aggregate and support each other to create a false good reputation. Detection of these behaviours is particularly difficult, as there is no definite way to distinguish between a simple difference of opinions, and a real threat. Punishment is even more difficult to perpetrate, because of the lack of a central authority to enforce it, and because of the anonymity of agents in mobile systems. We thus favour an anarchic model ("anarchy engenders trust and government destroys it" [14]), where each agent is responsible for his own fate, as far as detection of malicious agents and punishment are concerned. The TMF supports the agent by providing a conflict detection mechanism that relies on the tacit information the TMF maintains on behalf of its agent. When agent a receives new recommendations from agent x about some agent b, the function Ψ is used to compare b's trustworthiness, as perceived by a, with that recommended by x: if conflicts of opinions with x happen sporadically, chances are that they are simply disagreements of trust opinions without any malevolence. However, if they happen frequently, x's trustworthiness as a recommender quickly drop towards -1, that is, total distrust. A boundary value $\eta \in [-1, 1]$ for an agent's trustworthiness is defined, so that when x's trustworthiness drops below η, x becomes a *suspect*: recommendations coming from x are now discarded by the TMF, unless he recovers from this state. It is possible, in fact, that a mistake is made and a wrong opinion about x is formed; however, the more a interacts, the quicker and more likely the TMF will detect these mistakes, and thus have the chance to rectify them ("anarchy engenders cohesion" [14]). In this model, the punishment

Data	
Aggregate Trust Information	$[a, x, l, k, t]$
Tacit Information	$[a, x, l, k, t]$
Portfolio of Credentials	$[x, a, l, k, t]_{SK_x}$
Parameters	
Time Interval of Relevant Observations	T
Maximum Tolerate Discrepancy of Opinions	δ_{max}
Single Increment of Knowledge	k_{min}
Minimum Trust Level	η
Customising Functions	
Given two trust ranges, compute a trust range (used by Υ)	h_1
Given a trust range, compute a trust opinion (used by Υ)	h_2
Given a trust range, decide whether the prediction is precise enough (used by the recommendations exchange protocol)	h_3
Given three trust opinions, compute a new one (used by Φ)	h_4
Given a trust opinion and a discrepancy, compute a new trust opinion (used by Ψ)	h_5

Table 1: Agent a's Local Environment

for being a cheater is thus the loss of trust, which results in isolation from future interactions. After being isolated, an agent may very easily create a new identity; in this case, however, he will have no history, and thus other agents will be reluctant to trust him and start interactions with him.

In order for an anarchic model to work, the assumption that the number of honest agents is higher than the number of malicious agents must hold. Also, the social context should circulate as much trust information as possible, so that concealments are revealed, and conflicts are more quickly detected. To achieve this goal, the recommendation exchange protocol could end with the newly created letters of presentation (step 4) sent to the social context (instead of using a private exchange between the two interacting agents), as part of a more socially inspired protocol. The way the protocol should run is not enforced by our model.

Note that the mechanism described above allows detection of misbehaving recommenders, not of service providers, through analysis of the recommendations they spread around. Detection and isolation of malicious service providers (that is, agents that fail to deliver a service as they have advertised) sits at the boundary of the social and transactional context and is not dealt with in details in this paper. An unsatisfactory transaction has repercussions in the trustworthiness of the service provider as shown by the aggregation function Φ; the extent to which a transaction is considered unsatisfactory, as well as the trust opinion that results from such a transaction, are outside the scope of this paper.

4.4 Local Environment

Table 1 summarises the information that forms an agent's local environment: the data is continuously updated by the TMF using the aggregation function (to maintain aggregated trust information), the tacit information extraction function (to maintain tacit information), and during the agent's interactions (to maintain a portfolio of credentials). Parameters and customising functions enable the customisation of the TMF according to the user's natural disposition to trust. Examples of customising functions have been discussed in the previous sections; it is beyond the scope of this paper to supply values for the parameters.

5. DISCUSSION

We argue that the trust management model we have described in the previous section is particularly well suited for the mobile setting. First, it does not rely on trusted third parties, such as server repositories of trust information, or central authorities with special powers to detect and punish malicious agent; we favour an anarchic model instead, where each agent is solely responsible for his own fate.

Second, the resource demands imposed by an implementation of the framework can be minimal. Recommendations, for example, are not stored in an agent's local environment, but aggregated so that a single tuple is kept, thus minimising memory requirements per agent; the number of agents for which trust information is locally maintained then varies, depending on the trustor's needs. Tuples can also be periodically purged (e.g., when outdated $t_{now} - t_i > T$), to further limit memory requirements. A similar consideration applies to the computational overhead: the more frequently the recommendation exchange protocol is run, the more accurate the trust information received, the higher the load; similarly, the more often the tacit information extraction function is computed, the more precise the selection of good recommenders, the higher the load. However, the frequency with which these tasks are executed is not prescribed in our TMF: each agent has the chance to decide how many resources to invest in trust management (e.g., depending on the risks inherent to the transactions the agent undertakes, the resource capabilities of the device, etc.).

One of the major claims of our trust management model is that it simulates the human approach to trust management in the computer world. It does so by means of the customising functions described in the previous section. A major concern is then how these functions are selected. We expect the user of the device to select the 'trust profile' that better describes his natural disposition, from a list of available profiles that are offered him by the trust management framework. Attached to each profile is a choice of what customising functions to use in practice. The level of abstraction of the profile can vary from very high level (e.g., 'distrust the unknown') for non-expert users, to very low (detailed) level for expert users.

Apart from customising functions, the behaviour of our trust management model depends on a number of parameters, as listed in Table 1. We have not discussed in this paper possible values of these parameters. We have implemented a prototype of our trust management framework in C++, and we now plan to run simulations using the OMNeT++ [23] simulation environment, to empirically derive possible values for these parameters. These simulations will also show the level of accuracy of the computed trust opinions (e.g., no false negatives - all real cheaters should be caught, and no false positives - no benevolent agent should be charged).

6. CONCLUSION AND FUTURE WORK

This paper has discussed hTrust, a trust management model and framework that enables the development of trust-aware systems and applications. In particular, hTrust relieves the

programmer from dealing with trust formation, trust dissemination, and trust evolution issues. While doing so, the framework makes sure to capture the actual human disposition to trust of the user of the mobile device, by means of customising functions.

We are currently evaluating an implementation of our trust management model using the OMNeT++ simulation environment. As discussed in the previous section, the outcomes we expect from this exercise are manifold: empirical values to associate to the parameters that represent variables in our model; quantitative evaluation of the impact of these parameters on the model in terms, for example, of accuracy of trust opinions, resource usage, etc. This may lead to refinements of the model itself. At present, the recommendation exchange protocol propagates trust information in the social context using a simple flooding algorithm. It is our plan to integrate our trust management model with available event-based middleware, and replace the flooding algorithm with available, more efficient, routing algorithms for ad-hoc networking. Although not relevant for the discussion of the trust management model, we realise that an ontology to encode the information contained in the tuples (i.e., service description and context) should be defined.

Our research plans for the future include the run-time monitoring of interactions happening in the transactional context, both to provide feedback to the trust evolution component of our framework, and to detect (and react to) violations to the predicted trust level, prior to the completion of the transaction. On a longer-term basis, we would like to integrate trust, risks and Qos issues to drive the dynamic (re)configuration of a system.

7. REFERENCES

[1] A. Abdul-Rahman and S. Hailes. Using Recommendations for Managing Trust in Distributed Systems. In *Proc. of IEEE Malaysia International Conference on Communication (MICC'97)*, Kuala Lumpur, Malaysia, Nov. 1997.

[2] C. Adams and S. Farrell. Internet X.509 Public Key Infrastructure - Certificate Management Protocols. Technical Report RFC 2510, The Internet Society, Mar. 1999.

[3] T. Beth, M. Borcherding, and B. Klein. Valuation of Trust in Open Networks. In *Proc. of the 3^{rd} European Symposium on Research in Computer Security (ESORICS '94)*, pages 3–18, Brighton, UK, Nov. 1994.

[4] M. Blaze, J. Feigenbaum, and A. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures. In *Proc. of 6^{th} International Workshop on Security Protocols*, volume 1550 of *LNCS*, pages 59–63, Cambridge, UK, Apr. 1998. Springer-Verlag.

[5] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proc. of IEEE Symposium on Security and Privacy*, pages 164–173, Oakland, Ca, May 1996.

[6] S. Buchegger and I. L. Boudec. The Effect of Rumor Spreading in Reputation Systems for Mobile Ad-hoc Networks. In *Proc. of WiOpt 2003: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, Sophia-Antipolis, France, Mar. 2003.

[7] S. Capkun, L. Buttyán, and J. Hubaux. Self-Organized Public-Key Management for Mobile Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 2(1):52–64, 2003.

[8] M. Carbone, M. Nielsen, and V. Sassone. A Formal Model for Trust in Dynamic Networks. In *Proc. of First International Conference on Software Engineering and Formal Methods (SEFM'03)*, pages 54–63, Brisbane, Australia, Sept. 2003.

[9] S. Edwards, G. Shakir, M. Sitaraman, B. Weide, and J. Hollingsworth. A framework for detecting interface violations in component-based software. In *Proc. of the 5^{th} International Conference on Software Reuse*, pages 46–55. IEEE CS Press, June 1998.

[10] K. S. et. al. Requirements for Policy Languages for Trust Negotiation. In *Proc. of 3^{rd} International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, pages 69–79, Monterey, California, June 2002.

[11] V. C. et. al. Using Trust for Secure Collaboration in Uncertain Environments. *IEEE Pervasive Computing Mobile And Ubiquitous Computing*, 2(3):52–61, Aug. 2003.

[12] D. Gambetta. Can we trust trust? . In D. Gambetta, editor, *Trust, Making and Breaking Cooperative Relations*, pages 213–237. Basil Blackwell, Oxford, 1998.

[13] S. L. Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly UK, 1994.

[14] E. Gellner. Trust, Cohesion, and the Social Order. In D. Gambetta, editor, *Trust, Making and Breaking Cooperative Relations*, pages 142–157. Basil Blackwell, Oxford, 1998.

[15] T. Grandison and M. Sloman. Trust Management Tools for Internet Applications. In *Proc. of the 1^{st} International Conference on Trust Management*, Crete, Greece, May 2003.

[16] R. Jagannathan and P. Sivilotti. Increasing Client-side Confidence in Remote Component Implementations. In *Proc. of the 8^{th} European Software Engineering Conference - held jointly with the 9^{th} ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 52–61. ACM Press, 2001.

[17] A. Josang. A Logic for Uncertain Probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):279–311, June 2001.

[18] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks. In *International Conference on Network Protocols (ICNP)*, pages 251–260, Riverside, California, Nov. 2001.

[19] M. S. Miller and K. E. Drexler. Markets and Computation: Agoric Open Systems. In B. A. Huberman, editor, *The Ecology of Computation*, pages 133–176. Elsevier Science Publishers, 1988.

[20] G. C. Necula. Proof-Carrying Code. In *Proc. of the 24^{th} ACM SIGPLAN-SIGACT Symposium on Principles of Programming Langauges (POPL '97)*, pages 106–119, Paris, Jan. 1997.

[21] L. Rasmusson and S. Janson. Simulated Social Control for Secure Internet Commerce. In *New Security Paradigms Workshop*, pages 18–26, Lake Arrowhead, CA, Sept. 1996. ACM Press.

[22] B. Shand, N. Dimmock, and J. Bacon. Trust for Ubiquitous, Transparent Collaboration. In *First International Conference on Pervasive Computing*, pages 153–160, Dallas-Fort Worth, Texas, Mar. 2003.

[23] A. Varga. OMNeT++ Discrete Event Simulation System. http://www.omnetpp.org/.

[24] S. Weeks. Understanding Trust Management Systems. In *Proc. IEEE Symposium on Security and Privacy*, pages 94–105, Oakland, CA, May 2001.