TAPAS

IST-2001-34069

Trusted and QoS-Aware Provision of Application Services

TAPAS
Dynamic Trust Management

Nicola Mezzetti[a] **Licia Capra**[b]

Fabio Panzieri[a] **Wolfgang Emmerich**[b]

[a]Dipartimento di Science dell'Informazione, Università di Bologna, Bologna, Italy

[b]Department of Computer Science, University College London, London, UK

Report version: Deliverable D6.1
Report Delivery Date: 31 March 2005
Classification: Public
Contract Start Date: 1 April 2002 **Duration:** 36m
Project Co-ordinator: Newcastle University
Partners: Adesso, Dortmund - Germany; University College London - UK; University of Bologna - Italyl University of Cambridge - Uk

Project funded by the European Community under the Information Society Technology Programme (1998-2002)

Contents

Chapter 1

Introduction

In a Virtual Society (VS) context, possibly federated and mutually distrustful entities (i.e., individuals and resources) dynamically join and leave the system and interact with each other (providing and consuming services) in order to achieve their own goals. In order for these interactions to take place, such entities need to be aware of the trustworthiness of each other; recently, reputation management has been identified as a possible mean to enable entities to decide whether, and under which conditions, to engage in interactions with other entities; moreover, reputation management can provide an entity within a group with an expectation about the result of a possible interaction with any other entity.

Traditional solutions for trust management in distributed scenarios (e.g., [9, 8, 40]) rely on a static finite set of Trusted Third Parties (TTP) which are supposed to always behave in a dependable manner. Based on these static trust relationships, complex protocols built the interaction rules to satisfy the security properties (see Figure 1.1); for these properties to be always satisfied, the TTPs are required to remain trustworthy.

A generic VS is characterized by an arbitrary large population in which individuals, which dynamically leave and join the system, possibly exhibit dynamic behaviours, i.e., their trustworthiness may dynamically change because of failures, or because biased by their personal interests. Within this context, it is clear that the maintenance of static trust relationships would not provide a solid infrastructure on top of which to implement trustworthy interactions; therefore, it is necessary to rely on the deployment and management of locally-available information. In order to support interactions based on a dynamic form of trust, we designed the framework that is shown in Figure 1.2. The implementation of interaction policies based on dynamic trust requires both dynamic-trust criteria and evaluation criteria to be implemented. The first criteria identify decision semantics which enables an entity to decide whether to consider a second entity trusted or not, once estimation about that entity behaviour is provided. The second criteria enables an entity to evaluate the outcome of an interaction and produce the trust information that

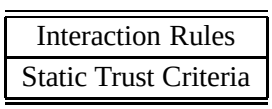

Figure 1.1: Framework supporting interactions based on static trust.

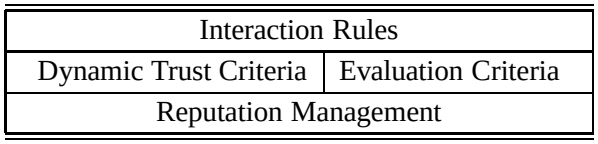

Figure 1.2: Framework supporting interactions based on dynamic trust.

is needed to adapt the estimations about other entities' behaviours; in fact, entities have opinions and form their own expectations using only their own knowledge, which embodies evaluations and expectations. Hence, having dynamic-trust criteria implies the existence of evaluation criteria.

Both the dynamic-trust criteria and the evaluation criteria interact with a Reputation Management layer; this layer of the framework is responsible for aggregating single trust evaluations produced by the evaluation criteria into expectations about entities behaviour. In turn, this aggregated information is the input which enables the dynamic-trust criteria to take a decision about whether, and at which extent, to trust a given entity. Since each single entity will exhibit a coherent behavior (i.e., if its trustworthiness does not change, then its behaviour is coherent, in a statistical manner, with that trustworthiness) within the same context, within the Reputation Management layer only 'objective behaviour' representations have to be implemented; that is, this layer has to provide the dynamic-trust criteria with objective evaluations of entities behaviour. For instance, an objective evaluation may be the aggregation of trust values so as to approximate the overall probability that an entity will completely satisfy another entity if an interaction with him was engaged. Then, the dynamic-trust criteria will translate these objective evaluations into a trust opinion which depend on the trust criteria of each single entity; for instance, Alice may consider trustworthy an entity which is expected to satisfy a given request with a probability between 0.4 and 0.6, while Bob may require a probability between 0.5 and 0.7 so as to consider the same entity trustworthy. We believe that having a common semantics which enables entities to be provided with objective evaluation helps in decentralizing the trust criteria and solves the problem of subjectivity.

When reading the contributions about the computational models of reputation [1, 3, 4, 6, 7, 10, 11, 13, 14, 28, 24, 26, 27, 32, 39], it is clear that a complete reputation model which enables the implementation of an adaptive, robust and scalable Reputation Management System (RMS) is still missing. In fact, examining the proposed trust and reputation models, it is possible to observe that some of them do not adequately capture the adaptive nature of trust [1, 3, 4, 6, 7, 10, 11, 39, 28, 24, 26, 27, 32], some are vulnerable to the presence of noise (i.e., random information) that is injected within the RMS [4, 10, 11, 13], and some others enable only implementations which may not scale when deployed within large communities [7, 13, 14, 32, 39]. Thus, when speaking about a reputation model, we basically require it to satisfy the following properties:

Adaptability: reputations have to adapt according to the behaviour that the individuals exhibit within the system and to the time passing;

Robustness: assuming the presence of a certain amount of noise, the reputation model has to identify it and prevent it from affecting trustworthy information;

Scalability: the reputation model must not require the maintenance of any global knowledge.

Within the reputation management context, we developed the Socially-Inspired Reputation model (SIR), a computational model of reputation which is inspired by the social and cognitive way in which reputation forms in real societies. SIR is based on three trust abstractions: reputation, direct trust, and indirect trust. Direct trust encodes trust based on the trustor's own experience; on the other hand, indirect trust encodes trust based on the recommendations the trustor receives from the other principals. SIR uses adaptive criteria both for aggregating fresh trust evaluations into direct trust, and for composing direct and indirect trust to form reputation, while meeting the requirements of adaptability, robustness, and scalability.

To demonstrate that our architecture meets the above mentioned requirements, the SIR computational model of reputation has been instantiated within the Trust-Aware Naming Service (TAw). TAw is a hybrid peer-to-peer architecture specifically designed for enabling trust reasoning within existing naming services. TAw has not been designed to play any active role in managing computer security; it simply enables each entity in the system with information about another entity's behaviour, to decide whether to engage in interactions with it. For instance, TAw can be employed in order to maintain information which pertains the reliability of a service replica in a cluster, as well as to set adequate security mechanisms when a number of entities engage in an interaction. The simulation experiments we have performed show that TAw provides service consumers and producers with an adaptive, robust, and scalable trust-aware service; the high scalability is due to the adoption of the peer-to-peer design principle and the epidemic dissemination model (also called gossiping). Moreover, TAw has been designed to provide flexibility for the application context and to enable interoperability between different naming service architectures.

The SIR reputation model and framework have proved to be well suited in traditional distributed settings; however, their direct applicability in more dynamic and unstructured environments, such as mobile ad-hoc networks, is less clear. In particular, 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, the risk perceived during interactions in mobile ad-hoc settings is much higher. As the rapid and enormous success of wireless networking technologies and portable devices suggests that pervasive services will play a key role in the future, we have devoted efforts in developing hTrust, a novel trust management model and framework that build on a common understanding of trust as the SIR model, but that specifically target the pervasive scenario. hTrust aims to capture the actual human disposition to trust of the user of the mobile device, by enabling high degrees of customisation of the framework. At the same time, it relieves the programmer from dealing with trust formation, trust dissemination, and trust evolution issues, by means of precise algorithms have maintain trust information dynamically.

The reminder of this report is structured as follows: Chapter 2 describes, in details, the SIR reputation model and the principles it is based upon. In Chapter 3 we illustrate the TAw architecture and discuss the results of a simulation-based evaluation that demonstrates how the SIR reputation model and framework achieve adaptability, robustness, and scalability. Chapter 4 presents the hTrust trust management model and framework, and illustrates how it can be used to support the development of trust-aware systems and applications for the ubiquitous environment.

Finally, Chapter 5 concludes the draft and highlights the major contributions of the TAPAS project with respect to dynamic trust management.

Chapter 2

The SIR Reputation Model

2.1 Trust, Trustworthiness and Reputation

In order for this work to be fully understood, it is worth for us to provide the reader with the definitions below, which clarify the meaning of terms such as trust, trustworthiness and reputation. First, we define a principal to be an entity that can play a role (i.e., trusting or being trusted) within a trust relationship; the set of all principals will be referred to as P. Second, we say Alice trusts Bob within a given context if Alice expects Bob to exhibit a dependable behaviour (i.e., competent and reliable) when acting within that context; thus, trust is a measure of how much reliance a trustor (e.g., Alice) can justifiably place on the dependability of a trustee (e.g., Bob) behaviour within a specific context. In the context of this work, we associate the contexts with service interfaces; specifically, since each service interface can be either produced or consumed, we say that for each service interface i, both the contexts c(i) and p(i), which respectively mean consumption and provisioning of interface i, are defined. Moreover, we define the subcontexts j(c(i)) and j(p(i)) to evaluate the trustworthiness of a principal in providing recommendations respectively within the contexts c(i) and p(i). We assume that a principal can consume or produce more than one interface; thus, it can be trusted within more than one context. Third, we say that a principal is trustworthy, within a given context, if it actually justifies reliance to be put on the dependability of its behaviour within that context; thus, we define trustworthiness to be a private and secret property of each principal and, therefore, neither known to other principals nor provable. Finally, we define the reputation of a given principal within a specific context to be the subjective evaluation of its trustworthiness that is built from a given, possibly partial, amount of information that describes its behaviour within that context.

2.2 Foundations of Trust

Initially, we assume trust to be defined as a ternary relation $\mathcal{T}(\alpha, \beta, \phi)$, where α and β are two principals and ϕ is a context. The presence of the triple $(Alice, Bob, c(i))$ in $\mathcal{T}$ indicates that Alice trusts Bob for *consuming* interface i.

A *trust system* is defined as the triple $(P, \Phi, \mathcal{T})$, where P is the set of principals and Φ is the set of contexts on which the trust relation $\mathcal{T}$ is defined. A trust relation $\mathcal{T}$ defined over $P \times P \times \Phi$ may satisfy the following properties:

reflexivity: *A trust relation $\mathcal{T}$ is reflexive in context ϕ if, for every principal $\alpha \in P$, the triple (α, α, ϕ) is in $\mathcal{T}$.*

symmetry: *A trust relation $\mathcal{T}$ is symmetric in context ϕ if, for every pair of principals $\alpha, \beta \in P$, if $(\alpha, \beta, \phi) \in \mathcal{T}$ then $(\beta, \alpha, \phi^{-1}) \in \mathcal{T}$ (i.e., each has assessed the trustworthiness of the other).*

Reflexivity is also called implicit trust. Basically, the reflexive property enables the existence of self confidence; that is, after two principals engaged an interaction with each other, each of them is enabled with much experience and, thus, is enabled with more data to compute a more precise "a-priori" estimation of other principal behaviours.

For instance, let us say that Alice carries out an interaction with Bob because she wanted interface i to be provided and Bob was able to satisfy her request. Hence, after the interaction Alice has a new trust evaluation which can be used to provide an estimation about the error that Alice made by referring to the trust information she had a priori.

Symmetric property states that any consumer/producer interaction has to affect the trust information of both the principals; i.e., both the consumer trust information about the producer and the producer trust information about the consumer. In fact, after an interaction between any two principals, each of them is enabled with more information about the other and can compute a better estimation of its behaviour in the context in which the interaction occurred.

Moreover, within each possible context, each single interaction is an instance of the client/server interaction paradigm; thus, both the client and the server are bound to the same interaction semantics. in this specific case of this work, the two context are both bound to the same interface.

When speaking of trust, it is worth mentioning the first context in which trust appeared, that is *authentication*. We define the context of authentication to be a special context that is equal to its inverse. Having the reflexive property satisfied within authentication expresses the trust that each principal is believed to maintain the secrecy of its private key. More in general, trusting a principal within authentication means trusting the binding between his private key and his identity. On the other hand, having the symmetric property satisfied within authentication indicates that, for any two principals, it is possible for them to mutually authenticate and, thus, implement security requirements such as non-repudiation of origin and integrity which are the base of each effective exchange of information.

A third property, namely *transitivity*, is defined by introducing the *jurisdiction* subcontext; it is used to represent trustworthiness in recommending principals acting within a specific context. Given a context ϕ, the jurisdiction subcontext associated with ϕ is represented in mathematical notation by the symbol $j(\phi)$. A principal having jurisdiction over a context ϕ is trusted for providing reliable trust information about trustees within context ϕ (i.e., whether a given trustee can be trusted or not); such information is referred to as *recommendations*. For example, $(Alice, Bob, j(\phi)) \in \mathcal{T}$ means that Alice places trust in Bob for having jurisdiction over context ϕ and is willing to inherit Bob's trust relationships for context ϕ. Assuming that each principal is enabled to verify the origin and integrity of trust information, transitivity is formally defined as follows:

transitivity: *A trust relation $\mathcal{T}$ is transitive in context ϕ if for every three principals α, β, γ such that $(\alpha, \beta, j(\phi)) \in \mathcal{T}$ and $(\beta, \gamma, \phi) \in \mathcal{T}$ then $(\alpha, \gamma, \phi) \in T \times T$.*

Table 2.1: Structure of Φ augmented with jurisdiction.

I	$::=$	$i \mid h \mid k$
C	$::=$	$p(I) \mid c(I) \mid authentication$
Φ	$::=$	$C \mid j(C)$

In other words, given a trust relation $\mathcal{T}$, transitive in context ϕ, and any three principals α, β, γ, if $\mathcal{T}(\alpha, \gamma, \phi)$ is not defined in $\mathcal{T}$, it can be indirectly computed if both $\mathcal{T}(\alpha, \beta, j(\phi))$ and $\mathcal{T}(\beta, \gamma, \phi)$ are defined; thus, the trust between principals α and γ is defined as $\mathcal{T}(\alpha, \gamma, \phi) = \mathcal{T}(\alpha, \beta, j(\phi))\ \mathcal{T}(\beta, \gamma, \phi)$ in $\mathcal{T}_{\{\alpha,\beta,\gamma\}} \times \mathcal{T}_{\{\alpha,\beta,\gamma\}}$.

Having the transitive property satisfied within authentication enables Certification Authorities and credential chains; a principal Alice, trusting a principal Bob for having jurisdiction within authentication, will inherit all the trust relationships that hold between Bob and any other principal, within the context of authentication. In this case, from Alice point of view, Bob plays the role of Certification Authority. Hence, having a transitivity chain within T enables principals to inherit trust relationships along the chain and thus, at an abstract level, certificate chains to be implemented.

In Tab. 2.1 the set Φ of the possible contexts is defined in terms of the definitions given above; here, I indicates the set of service interfaces and C the basic contexts.

Given the basic notions of trust, we define a *trust system* to be an environment in which trust relationships can be established between that environment's principals; a trust system is specified by the triple $(P, \Phi, \mathcal{T})$ where P is the set of principals, Φ is the above defined set of contexts, and $\mathcal{T}$ is the trust relation.

2.3 The Reputation Model

Above these trust foundations, SIR implements a computational reputation model that is intended to enable the implementation of adaptive, robust, and scalable RMSs.

According to the nature of social trust, in this section we define three trust functions: *reputation*, *direct trust*, and *indirect trust*. In order for the reputation function, $\mathcal{R}$, to be defined both direct trust $\mathcal{D}$ and indirect trust $\mathcal{I}$ have to be known; they both model a trust degree between a trustor and a trustee within a given context. Direct trust encodes trust based on the trustor's own experience. In turn, indirect trust encodes trust based on the recommendations the trustor receives from the other principals.

A generic trust function takes the form described in (2.1); it takes four parameters, which respectively represent the trustor, the trustee, the context, and the time at which trust has to be evaluated, and returns a trust value, which represents the *strength* of the trust relationship between the trustor and the trustee. In order to express this "strength", the trust function returns a real value belonging to the closed set $[0, 1]$ within which the value 1 indicates "full trust" and 0 indicates "absence of trust" (i..e., absence of data which justify reliance to be placed on the

trustee behaviour).

$$\mathcal{T} : \mathcal{P} \times \mathcal{P} \times \Phi \times \tau \longrightarrow [0, 1] \tag{2.1}$$

Legitimately, obsolete information does not enable to accurately describe recent behaviours. Therefore, we assume trust degrees to decay as time passes; specifically, for each context we define a decay function that models the speed with which trust information becomes obsolete within that context.

$$\delta_\phi(t) : \tau \rightarrow [0, 1]$$

Several decay functions are needed because the validity of a trust degree depends on the nature of the context within which it applies; e.g., the more critical (i.e., risky) the context is, the more rapidly the trust decreases with respect to time if no new trust information is available. However, for each of the decay functions we require it to be an homomorphism from $(\mathbb{R}^+ \cup \{0\}, +)$ to $([0, 1], \cdot)$, such as, for example, the exponential function in (2.2).

$$\delta_\phi(t) \ = \ \eta_\phi^t \tag{2.2}$$

The reader can understand the reason of this requirement by referring to [34].

Equation (2.3) models direct trust, which represents the perception of a principal's trustworthiness which is uniquely determined according to the trust evaluations in which the principal computing the reputation took part in the role of trustor. Specifically, if the set E of trust evaluations does not change, the direct trust at time t' is computed by applying the decay function to the last direct trust value $\mathcal{D}(\alpha, \beta, \phi, t)_E$, computed at time t; otherwise, be $tv_{t'} \in [0, 1]$ the fresh trust evaluation obtained at time t', the new direct trust value is computed by the convex combination of both $tv_{t'}$ and the past direct value of $\mathcal{D}(\alpha, \beta, \phi, t)_E$, that is the last direct trust value computed at time $t \leq t'$.

$$\mathcal{D}(\alpha, \beta, \phi, t')_{E'} \ = \ \begin{cases} \mathcal{D}(\alpha, \beta, \phi, t)_E \cdot \delta_\phi(t' - t) & \text{if } E' \ = \ E \\ \\ (1 - \omega) \cdot \mathcal{D}(\alpha, \beta, \phi, t)_E \cdot \delta_\phi(t' - t) + \omega \cdot tv_{t'} & \text{if } E' \ = \ E \cup \{tv_{t'}\} \end{cases} \tag{2.3}$$

In the equation above, the *trust stability* parameter, represented by the notation ω, indicates how much the result of a new interaction affects the trust value that is derived from the previous direct interactions. In principle, the trust stability should depend on how much the fresh trust value differs from the expected one and on whether that difference is positive or negative: as much that difference is higher, a positive one will be discounted while a negative one will be counted.

Equation (2.4) models indirect trust; we define it to be the average reputation that a set Γ of known principals, namely *recommenders*, associate with the trustee (within a given context and at a specific time). In that equation, it is worth noting the use of the notation $\mathcal{R}(\gamma, \beta, \phi, t)$ that stands for the reputation obtained from principal γ about principal β; for clarity, we call recommendation each reputation that a principal obtains from other principals. The weight that is assigned to each single recommendation is the current direct trust between the α, the principal receiving the recommendations, and the recommender which provided that recommendation, within the context of "provisioning of recommendations within context ϕ" which is represented

by the notation $j(\phi)$. Whenever fresh trust information is not available the trust decay applies as in the case of direct trust.

$$\mathcal{I}(\alpha,\beta,\phi,t')_{R'} = \begin{cases} \mathcal{I}(\alpha,\beta,\phi,t)_R \cdot \delta_\phi(t'-t) & \text{if } R' = R \\ \frac{\sum_{\gamma\in\Gamma} \mathcal{D}(\alpha,\gamma,j(\phi),t')_E \cdot \mathcal{R}(\gamma,\beta,\phi,t')}{\sum_{\gamma\in\Gamma} \mathcal{D}(\alpha,\gamma,j(\phi),t')} & \text{otherwise} \end{cases} \tag{2.4}$$

Equation (2.5) defines reputation as the convex combination of direct and indirect trust; the *trust balancing* factor, represented by the notation ψ, is a factor that indicates the subjective weight a specific principal assigns to direct trust with respect to indirect trust; it is individually decided by each principal within the real interval $[0,1]$. Moreover, in this equation E and R indicate, respectively, the set of trust evaluations directly performed by the trustor and the set of recommendations that he collected from a set Γ of recommenders. Similarly to what happens to both direct and indirect trust, if any new trust evaluation is not available, it is possible to compute the current reputation from the previously computed value.

$$\mathcal{R}(\alpha,\beta,\phi,t')_{E',R'} = \begin{cases} \mathcal{R}(\alpha,\beta,\phi,t)_{E,R} \cdot \delta_\phi(t'-t) & \text{if } E' = E \wedge R' = R \\ \psi \cdot \mathcal{D}(\alpha,\beta,\phi,t')_{E'} & \\ +(1-\psi) \cdot \mathcal{I}(\alpha,\beta,\phi,t')_{R'} & \text{otherwise} \end{cases} \tag{2.5}$$

Let any two principals carry out an interaction; after that interaction, each of them can associate a trust value with the other principal, according to his behaviour during that interaction. However, a new trust value does not only contribute to compute the direct trust between a trustor and a trustee; it is also used for computing the direct trust between the trustor and the recommenders. Basically, the smaller the difference between the recommendation and that trustee's direct trust (updated with the latest trust value), the better reputation that will be associated with that recommender by the trustor. Thus, given a trustor α, a recommender β, and a trustee γ, the trust value, tv, that α associates with a recommendation from β regarding γ trustworthiness within a context ϕ, will be 1 if is is equal to the direct trust after the interaction and will tend to 0 as the difference becomes larger; the new direct trust degree between the trustor and the recommender is to be computed according to (2.3) and (2.6).

$$tv \;=\; \big(1 - \big|\mathcal{R}(\beta,\gamma,\phi,t') - \mathcal{D}(\alpha,\gamma,\phi,t')\big|\big) \tag{2.6}$$

2.3.1 Augmenting Contexts with Attributes

So far, we described how to evaluate a principal's reputation either in consuming or producing a given interface or in providing recommendations within a specified context. Hence, if we know that $\mathcal{R}(Alice, Nicola, p(smtp), t_0) = 0.67$, it is still unclear on which aspect the partial unreliability depends, e.g., whether Nicola's smtp server rarely allows illegitimate access to emails or just because of a low availability of the service. When a trustor finds a trustee's reputation to be unsatisfactory, in order for the trustor to interact with that trustee it is legitimate for him to be aware of the origin of that partial unreliability; attributes are introduced to fill this purpose.

Table 2.2: Structure of Φ augmented with attributes.

I	$::=$	$i \mid h \mid k$
A	$::=$	$a \mid b \mid d$
G	$::=$	$p(I) \mid c(I) \mid authentication$
C	$::=$	$a(G) \mid G$
Φ	$::=$	$C \mid j(C)$

We define an *attribute* to be a property that is relevant within a given context, such as availability, confidentiality or authentication. Attributes are introduced into our model by defining associated subcontexts, that are used to specialize the generic contexts.

In Tab. 2.2 the grammar describing the structure of Φ with attributes is formally described; here, I indicates the set of interface names, A the set of attribute names, G the set of simple contexts and C augments G with subcontexts.

Such a design approach enables us to simultaneously maintain both overall reputations and specialized ones in an efficient manner; in fact, when a principal is assigned a new trust value in a generic context $\phi \in \Phi$, if it corresponds to a specialized context, then the corresponding reputation can be updated and the associated generic context can be retrieved in order for it to be updated as well. Hence, the redefined definition of Φ allows us to manage specialized reputations preserving the validity of the formulas presented in the previous section. The inverse is not defined for attributes.

2.4 Related Work

In [1, 3], Abdul-Rahman and Hailes employ a reputation-based infrastructure for information retrieval. Here, reputation is represented in a manner that makes no distinction about which trust evaluations happened far in the past and which ones happened recently, therefore it impossible for the trustee to understand the behavioural pattern of the trustee and to get a picture of its current dependability. In [4], Aberer and Despotovic introduce a complaint-based reputation system to be applied within peer-to-peer infrastructures for file sharing. We believe this complaint based reputation model is not adequate to represent social reputation in that it is not representative of the whole interaction history; in fact, since the reputation model is not adaptive with time, it is impossible for a principal which committed mistakes to rebuild a good reputation. Moreover, the model does not provide any mean to distinguish legitimate trust information from malicious ones. In [6], Azzedin and Maherswaran develop a socially inspired reputation model that is similar to the one proposed in this article. According to this model, a recommender is evaluated as honest if in the past history it returned recommendations that describe a quite stable behaviour of a given trustee; however, in order for this criteria to be valid it requires the trustee to really exhibit a stable behaviour. This contradicts the dynamic nature of reputation. a reputation system

would become useless if it was guaranteed that principals always exhibit the same behaviour. Moreover, this model does not consider trust information to become obsolete as time passes and the semantics of the direct-trust update does not consider the behavioural history of the trustee. In [7, 39], Beth and Yahalom describe a formal reputation model for open networks, which is founded on bayesian probability theory and reliability analysis techniques For this model to be employed, each principal trustor has to be aware of all the direct-trust relationships which form the trust network; hence, this model does not scale as the trust-network complexity and the trust contexts grow. Moreover, this model does not consider trust to adapt with time. In [10, 11], Buchegger presents a reputation which is based on the beta probability density function. This model does not implement trustworthiness of recommendations; instead, it identifies as malicious recommendations the ones which substantially differ from the direct trust. Hence, if a recommendation reports in the trustee a behaviour change which recently happened and which has not yet been identified by the trustor, this trustworthy information would be identified as malicious by the trustor and thus discarded. Moreover, we find quite debatable the approach adopted to model trust adaptability with time: firstly, the decay does not depend on time but on the position of the trust evaluation within the sequence; second, when the decay is applied, the aging does not applies on the whole evaluation history but rather on either the sequence of successes or the sequence of failures, according to the last trust evaluation. In [28, 24, 26], Jøsang defines a trust model which is based on a triple of probability values describing the behaviour that a specific principal will assume when an interaction with him is engaged; respectively, this triple specifies the probability of engaging a positive interaction, the probability of engaging a negative one, and the uncertainty. To our opinion, this model completely hides the information about the interaction history of the trustee, preventing the trustor to understand the behaviour that the trustee could currently exhibit. Trust adaptability with time has not been considered in the model. In [27], Jøsang proposes a reputation model based on the beta probability density function that does not distinguish between direct and indirect forms of trust. Moreover, trust adaptability presents the same problems as in [11].

Related Publications

- Mezzetti, N.: Towards a Model for Trust Relationships in Virtual Enterprises. In Proceedings of 14th Database and Expert Systems and Applications Workshops, pag. 420–424, Prague, September 2003.

- Mezzetti, N.: A Socially Inspired Reputation Model. In Proceedings of the 1st European PKI Workshop (EuroPKI 2004), Lecture Notes in Computer Science (Springer-Verlag), vol. 3093, pag. 191–204, June 2004.

Chapter 3

The TAw Architecture

3.1 The TAw Architecture

The TAw architecture is a hybrid peer-to-peer architecture that has been designed to enable trust-aware interactions within collaborative environments; using TAw, a principal is provided with trust information which enables him to decide whether, and at which conditions, to engage in interactions with other principals. The TAw architecture implements the abstraction of a virtual society where principals interact with each other exchanging local trust information; hence, each principal aggregates the available trust information so as to approximate the trustworthiness of the principals with which he is engaging in interactions. The Virtual Society Service (VSS) is responsible for enabling the principals to aggregate themselves forming a graph which represents a virtual society, on which trust information is exchanged. In TAw, each principal is enabled with a specific TAw peer that implements the social behaviour (e.g., SIR) on behalf of that principal; that is, it locally collects and maintains trust information, computes reputations and propagates them on his behalf. A piece of trust information that is collected by a peer is either originated by the owner principal as evaluation of a performed interaction, or obtained via the trust propagation protocol described further in this section.

3.1.1 Virtual Society Service

The main function of the VSS is to enable the implementation of the virtual society; specifically, it enables the principals to aggregate with each other, forming the social infrastructure over which trust information is exchanged. To enable this service, the VSS employs a directory service which maintains information about all the principals in the virtual society according to the contexts in which they will consume or produce interfaces. The directory service implements the four levels Directory Information Tree (DIT) represented in fig. 3.1: the root represents the starting point to access the stored information; the first level maintains the possible service interfaces; the second level implements the contexts generated by each service interface; and, finally, the third level maintains the principals which either consume or produce services in the specific context.

To contribute to the trust information exchange, a principal joining the virtual society has to be provided with knowledge about how to propagate trust information. We assume to represent the virtual society as a directed random graph in which each node represents a principal and the

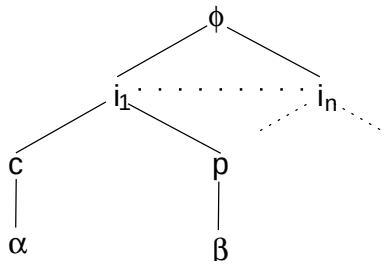

Figure 3.1: Directory Information Tree implemented by the VSS service.

edges represent the direction in which trust information is propagated; given a principal p, we call the *neighborhood* of p the set of principals that are directly connected to p. Reading the literature about epidemic data-dissemination paradigm [15, 29], it is known that a n-nodes direct random graph with degree $O(\log n)$ has diameter in $O(\log n)$; that specific node degree takes the name of *fan-out*. Therefore, in order to minimize the overhead introduced by the propagation of trust information, we require the neighborhood size to be in $O(\log n)$; thus, when a principal joins the virtual society, a random set of neighbors is associated with him for the recommendations to be propagated.

In order for trust information to be propagated uniquely towards principals which are concerned with it, in the virtual society principals are grouped according to the specific contexts in which they engage interactions; hence, we require the neighborhood to be composed by principals which are concerned to the same set of contexts in which the respective principal is.

Due to the dynamic nature of VSs, principals that dynamically join and leave the system may enforce frequent changes in the fan-out; as a consequence, it may be required for all the principals in the system to frequently update their neighborhood in order to meet the above mentioned efficiency requirements. We prevent such updates by enforcing the neighborhood size to be larger than the gossiping fan-out and, at each round of the trust propagation protocol, to enforce the spreading of trust information only to a subset of the neighborhood of size equal to the fan-out.

For TAw to be more robust against byzantine trust information (i.e., noise or malicious reputations), the VSS enables each principal to update its neighborhood with a new one, which is randomly decided by the VSS itself; the neighborhood update limits the influence that such byzantine information have on other principal's ability to compute representative reputations.

We use the term session to identify the sequence of activities that a principal performs, starting with his to the virtual society and ending with the respective leaving. In order to control the maximum influence that a principal can have on the spreading of trust information over the social network, we require the VSS to implement an access control policy so at to enforce each principal to maintain at most one open session at once. In fact, allowing a principal having to maintain several open sessions enables him to appear as multiple principals; that would prevent the other principals to associate to the same principal all the behaviours exhibited by that group of principals and, consequently, to correctly compute that principal reputation. Therefore, in order to enable principal authentication and to verify the origin and the integrity of propagated trust information, each principal which is legitimate to employ TAw services is uniquely identified by

a long-term PK certificate.

So as for the principals to manage its activity within the VS, the VSS provides them with an interface to manage the membership within the virtual society and to enable further exchange of trust information; specifically, that interface enables the principals with the following operations:

`startSession(Certificate c):` Enables a principal to initiate a new TAw session and access the services provided by the VSS;

`endSession(Certificate c):` Terminates an existing TAw session;

`bind(Context c, String reference, NetAdd address):` Enables a principal to register both its naming reference and the address for the reputation exchange to the VSS according to the contexts within which it will consume or produce services; if any context does not already exist, it is created and the attributes of that principal are inserted into the directory structure under that context. This operation returns information about the exit status of the bind operation on the LDAP service;

`lookup(Context c):` Given an object `c` of type `Context`, this operation returns to the principal the naming reference of a principal which might be trusted within that context;

`updateNeighborhood(Context c):` Given an object `c` of type `Context`, this operation returns to the principal a new randomly selected neighborhood (according to the above mentioned criteria);

`unbind(Context c, String reference):` Enables a principal to remove its naming reference from the directory structure. This operation returns information about the exit status of the unbind operation on the LDAP service;

Essentially, the VSS enables the reputation system to start from scratch, connecting principals with each other forming a social network for the propagation of trust information; moreover, in order for the VSS to tolerate failures and not to constitute a bottleneck, it can be implemented in a distributed and trustworthy manner. For instance, the VSS can be implemented as a cluster of servers, which have to agree (e.g., by voting) on the decisions about the management of TAw system (e.g., which principals can legitimately access TAw services).

3.1.2 TAw Peer

The TAw peer is the abstraction of a principal within a virtual society; it is a proxy client which mediates between the principal itself and the naming service and employs the services enabled by the VSS for implementing the principal social behaviour.

Each peer embodies a data structure, namely the *trust repository*, that is used to maintain trust information; basically, it is a collection of tuples $(\alpha, \beta, \phi, t, p)$ where $\alpha \in \mathcal{P}$ is the trustor, $\beta \in \mathcal{P}$ is the trustee, $\phi \in \Phi$ is the context, t is the time to which the trust degree refers and $p \in [0, 1]$ is the trust degree associated with $\mathcal{R}(\alpha, \beta, \phi, t)$. A peer acquires trust information both when the associated principal consumes (provides) services from (to) other principals, and when the trust degrees are collected from the other peers, via the trust propagation protocol which we describe below.

Autonomously each TAw peer disseminates trust information towards a subset of its neighborhood through the trust propagation protocol (see below). Such a protocol is triggered if either of the following conditions are met:

1. a fixed amount of time elapsed since the last protocol run and some trust information has been updated;
2. a fixed amount of trust information has been acquired since the last protocol run.

A TAw peer provides the client application with two interfaces, respectively the *naming interface* and the *trust management interface*. The naming interface provides the associated principal with the operations that are required to access the naming service in a trust-aware manner; precisely, it provides the principal with the operations described below.

`login`: Enables a principal to initiate a new TAw session and access the services provided by the VSS. An invocation of this operation on the TAw peer implies a corresponding invocation of the `login` operation on the VSS;

`logout`: Terminates an existing TAw session. An invocation of this operation on the TAw peer implies a corresponding invocation of the `logout` operation on the VSS;

`bind(Context c, String reference)`: Enables a principal to register itself to the VSS according to a context within which it will consume or produce services; for each context implemented by a principal a single binding can be simultaneously established. An invocation of this operation on the TAw peer implies a corresponding invocation of the `bind` operation on the VSS, whose return value is returned from this operation as well;

`trustAwareLookup(Context c)`: Given an object `c` of type `Context`, this operation returns to the principal the reference to a principal within that context, according to the following semantics:

1. reputations are updated according to currently available information;
2. if there are trustees in that context, the one with the best reputation is returned,
3. otherwise the `lookup` operation is invoked on the VSS to obtain the reference to a principal which implements the requested context; its direct trust is set to 0 to indicate that no information is available about its past behaviour.

`unbind(Context c, String reference)`: Enables a principal to remove its naming reference from the directory structure. An invocation of this operation on the TAw peer implies a corresponding invocation of the `unbind` operation on the VSS; the value returned by that operation is returned to the calling principal.

The naming interface provides the associated principal with the operations that are required to retrieve and introduce trust information about a principal within a given context and adapt the reputations according to the new evaluations; precisely, it provides the principal with the operations called `getTrust` and `putTrust`:

Table 3.1: Structure of a trust propagation protocol message.

$Reputation$	::=	$\langle trustee, time, reputationValue \rangle$
$Context$	::=	$\langle context, length, Reputation^{+} \rangle$
$TrustPropagationMessage$	::=	$\langle trustor, length, Contexts^{+} \rangle$

`getTrust(String reference, Context c)`: Given the reference to a principal and a context, this operation returns the reputation that is locally associated to that principal within that context. This operation implements the following semantics:

1. the reputation table is updated according to the currently available trust information;
2. the computed reputation is returned.

`putTrust(String reference, Context c, TrustDegree t)`: Given the reference to a principal, a context and a trust evaluation, this operation updates the trust information (i.e., direct trust, indirect trust and reputation) of that principal and returns the new reputation value. Moreover, it possibly triggers the trust propagation.

1. the reputation table is updated according to the currently available trust information;
2. if condition 2 for propagation is met, then the reputation table is then propagated towards other principals.

3.1.3 TAw Trust Propagation Protocol

In order for trust information to be distributed over the system, TAw implements an epidemic-like dissemination technique; specifically, each TAw peer periodically sends the newly computed reputations to a random subset of its neighborhood, whose size corresponds to the fan-out. The receiving neighbors will store these tuples in the respective trust repositories where they will be employed for computing fresh reputations.

Table 3.1 describes the structure of a generic trust propagation message; so as to minimize the size of the message, in a generic message trust information is grouped by the reference context. Thus, each message specifies the trustor, the number of contexts included within the message and, for each context, the number of reputation tuples included in the message and the tuples themselves.

When the trust propagation protocol is triggered, either because a timeout occurred or enough trust information has been directly updated, then the protocol behaves as described by the pseudocode reported in Tab. 3.2. In words, each time a protocol run is triggered, a subset of the neighborhood is randomly selected and the set of trust information that has been updated since last propagation round is identified. Then, for each neighbor in the selected fan-out, a protocol message, containing the trust information in which that neighbor is concerned, is built and sent to that neighbor.

```
1    randomly select a fan-out from the neighborhood;
2    select the recently updated reputations;
3    for each neighbor n in the selected fan-out:
4        build an appropriate protocol message;
5        send that protocol message to neighbor n;
```

Table 3.2: Pseudocode describing the trust propagation protocol.

The adoption of the epidemic model gives probabilistic guarantee to maintain a social network whose diameter (i.e., the maximum social distance between any two individuals) counts $O(\log n)$ edges, with n being the number of individuals in the society. Never in two successive propagation rounds the same information is propagated. At each round each individual propagates its own opinion about the known principals, according to the social human behaviour; however, it is worth noting that such an epidemic technique allows a fresh trust information to reach all the TAw peers within a number of propagation rounds that depends on the logarithm of group size.

The adoption of such an epidemic model for trust information spreading does not provide the guarantee that all the principals compute the same trust degree about a specific principal; however, according to the preliminary results in [35], we expect trust degrees to tend towards the mathematical ideal described in Sec. 2.3, in a manner that depends on the social distance between the trustor and the trustee (i.e., the average number of recommenders which stand between the trustor and the trustee).

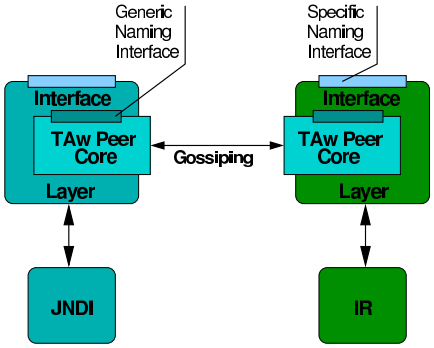

Figure 3.2: Interoperability among TAw peers.

3.1.4 Interoperability

Interoperability is a key issue to address in a wide area system; in this section we outline some design issues that enable TAw (i) to be suitable for extending every existing naming and directory service and (ii) to allow a global virtual society to be built upon several domains which make use of different naming and directory technologies. Both of the mentioned interoperability aspects are addressable by having a common underlying trust semantics; this semantics also allows a common internal representation of trust information. Thus, we address interoperability by dividing the individual in two logical units:

TAw Core peer: The core peer implements the trust semantics described in Sec. 2.3. Each peer implements both a *generic naming interface* and a *peer interface*. The former enables a principal to query the trust repository for service providers according to the semantics explained in Sec. 3.1.2. The interface is called "generic" because each specific naming-and-directory service interface can be mapped into a sequence of invocations to this interface. The peer interface implements the trust propagation protocol; this interface is not affected by the underlying naming and directory service. In fact, the common trust semantics and representation allow peers to exchange trust information independently from the interface implemented by the underlying naming service.

Interface layer: The interface layer implements the underlying service specific interface with a set of invocations to the generic naming interface that is provided and implemented by the core peer. In addition, this layer possibly translates the results returned by the invocations of the generic naming interface into a set of invocations on the actual instance of the underlying naming service.

Figure 3.2 shows two peers are interacting with each other; the first is deployed on JNDI whereas the second is deployed on Corba IR. The common trust semantics enables the TAw Core peers to communicate with each other, exchanging trust information. The peer interface layer enables the respective applications to transparently manage trust information, providing the same interface of the underlying naming and directory services. This design approach addresses the second concern as well; because of the common trust semantics and representation, the core peers are able to exchange trust information with each other directly as long as they are able to identify each principal by resource identifiers that are independent from any specific naming and directory service.

3.2 Evaluation of TAw

Extensive simulations have been carried out so as to evaluate how the TAw implementation of SIR behaves in a producer-consumer scenario. The results that we are going to present have been computed considering a generic producer/consumer scenario in which 1000 consumers join the TAw infrastructure in order to access services provided by a set of producers; the behaviour of the consumers, the size of this set, and the behaviour of the producers vary according to the purpose of the experiment. Specifically, each consumer is characterized by a maliciousness degree, that is the probability that it will propagate randomly generated reputations instead of the correct ones; on the other hand, each provider is described by its trustworthiness, i.e., the probability that it will exhibit a dependable behaviour in delivering a service, and a behavioural pattern, that is a function that describes how the trustworthiness changes depending on time (this feature has been introduced to specify failures, i.e. byzantines and crashes, as well as failure recoveries). When a provider is not fully dependable (i.e., its trustworthiness is below 1), we assume that it may exhibit a failure with a uniform probability probability distribution whose average value is equal to its trustworthiness. Similarly, at each propagation round a malicious recommender (i.e., its maliciousness is above 0) will decide whether to spread out either wrong or correct trust information using a uniform probability distribution whose average value is equal to its maliciousness.

Since the VSS implements a social network for each context within which principals are currently bound, the results of the simulations are not affected by the number of contexts that are defined; hence, we decided to define a unique service interface with the associated contexts (i.e., consumption, provisioning and recommending) focusing our attention on the properties of the reputation management system. Within that context, each consumer generates independent service requests towards the most trustworthy provider. The time interval between the generation of any two consecutive requests is generated through a negative exponential distribution with mean 30 time units; the negative exponential distribution models has the property to model the time between the occurrence of two stochastically independent events.

Within TAw, performing a trust-aware interaction corresponds to the following sequence of actions, whose semantics is described in the previous section.

1. `ref = trustAwareLookup(c)`: the principal requires the best provider within context c;
2. `res = ref.invoke()`: the service is invoked on the provider;
3. `tv = evaluate(res)`: a trust value is associated with the performed interaction;
4. `rep = putTrust(ref,c,tv)`: a new trust value for trustee ref within context c triggers trust information update.

Moreover, each peer periodically propagates trust information towards its neighborhood; the interval between any two consecutive propagations is 120 time units. The experiments that we are going to describe enabled us to assess at which extent TAw satisfies adaptability and robustness within a generic VS environment. Since TAw has been designed in order not to require a global knowledge, the scalability of the system is satisfied by definition of the architecture; in addition, the properties of the epidemic propagation technique guarantee that the system behaves scalably as the number of principals joining TAw increases.

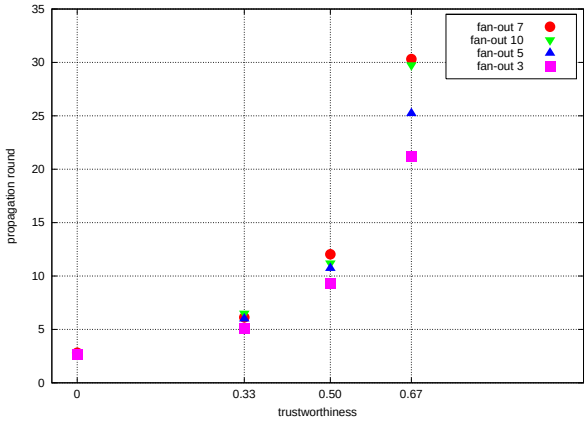

Figure 3.3: Reconfiguration in presence of failure without malicious consumers.

3.2.1 Adaptability Test

With this experiment we have been studying how TAw enables system reconfiguration in presence of failures that occur within the system; specifically, given any consumer which currently refers to a dependable service provider, if a failure occurs that compromises the dependability of this provider, we have been evaluating the mean number of propagation round that are required for that consumer to select a new dependable provider. For this experiment we defined 50 providers, 5 of which are fully dependable (i.e., trustworthiness 1), and the remaining are affected by randomly generated byzantine failures that reduce their dependability (i.e., trustworthiness $\in [0, 1)$). Within this setting, once all the consumers are stabilized on those five most dependable providers, a byzantine failure is triggered on one of these providers so as to enforce the consumers to reorganize themselves so as to refer to the remaining four dependable providers. In this experiment, we considered four different kind of failures, that respectively reduce the trustworthiness of the selected correct provider to 0.67, 0.5, 0.33, and 0. Figure 3.3 shows the mean number of propagation rounds that are required for reconfiguration, when the above mentioned failures occur, and setting the fan-out size respectively to 3, 5, 7, and 10. The reader can observe that the more the failure affects the provider the faster the consumers reorganize so as to refer to a correct provider; the reason of this behaviour is that the more a failure is noticeable by all the principals, the faster is for them to achieve a good approximation of the failure. The reader note that the fan-out of 10 corresponds to the logarithm of the number of nodes in the social network and, according to the results presented in [15, 29], such a fan-out reduces the diameter of the social network to the order of the logarithm of the number of nodes in that network, thus optimizing the number of instances of the propagation protocol that are required for a piece of trust information to cross the entire network.

3.2.2 Robustness Test

With this experiment we have been studying the robustness of TAw against malicious principals which propagate wrong trust information within the system; specifically, we have been evaluating at which extent any consumer can approximate the trustworthiness of a given provider which exhibit a dynamic behavioural pattern. Hence, for this experiment we defined a single provider with the following behavioural pattern: initially, it behaves dependably with trustworthiness 1; at interaction round 60 it suffers a crash failure; at the interaction round 120 the crash recovers and it behaves dependably with trustworthiness 0.8; at round 180 another failure happens that reduces its trustworthiness to 0.6; finally, at round 240, the provider is fully recovered and its trustworthiness is 1. Within this setting, we estimate at which extent the provider's reputation, as computed by the consumers, is affected as the number of malicious consumers increases.

In Fig. 3.4 (a) it is possible to see the average reputation of the provider, computed over the 1000 consumers when no byzantine information is introduced within the system. How it is possible to observe, the approximated pattern follows the specified behavioural pattern for the faulty principal. In addition, the adaptability rate of the trust function can be modified by setting the trust decay parameter according to the specific application context. In Fig. 3.4 (b) shows the variance computed on the reputations of Fig. 3.4 (a), which affects only the second decimal digit of the computed reputation of each principal. Specifically, it is possible to see

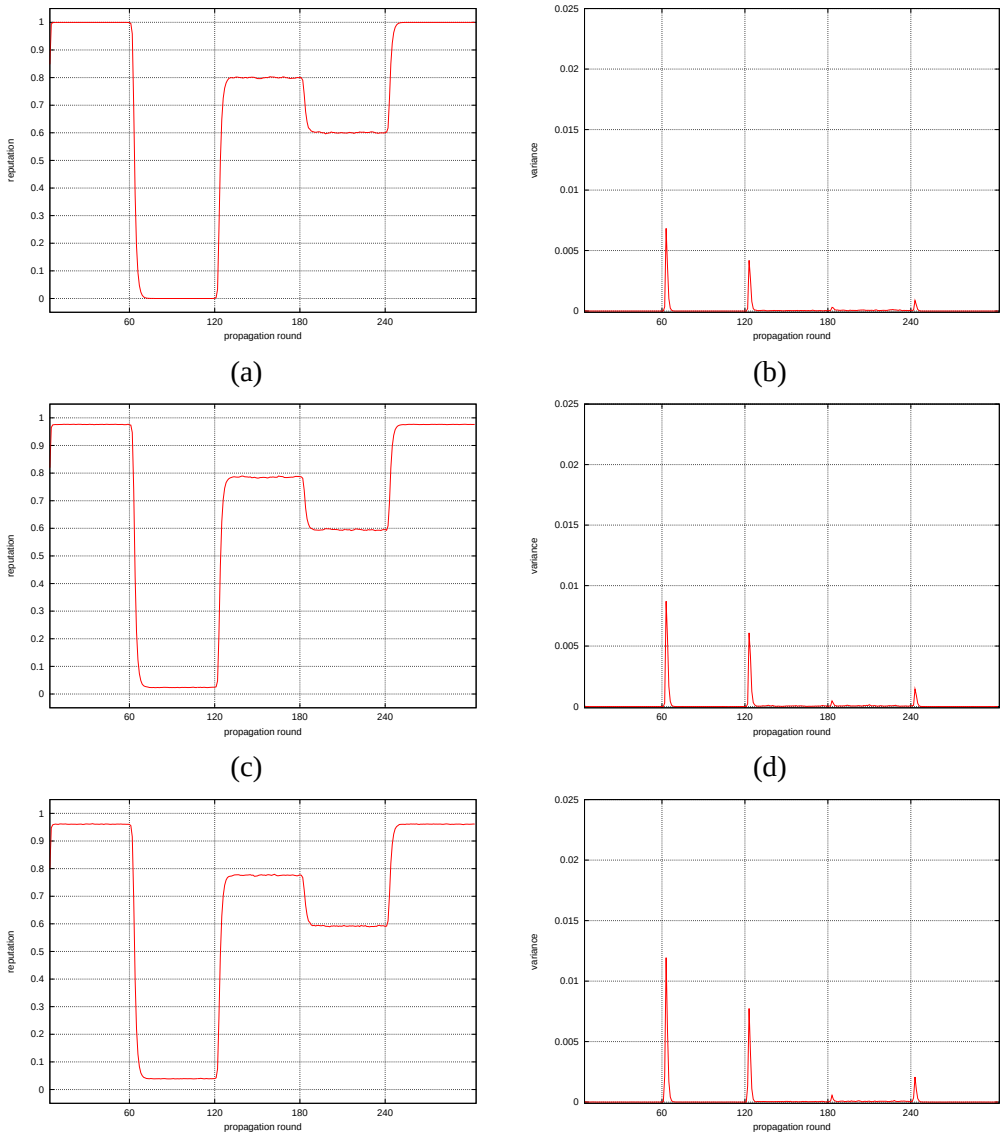

Figure 3.4: Robustness test with a single provider in presence of failures.

that, as the provider changes its trustworthiness (i.e., the behaviour in the intervals between propagation round 30 and 60 and between 90 and 150), there is a variance peak which represents the asynchrony with which all the principals discover the change in the provider behaviour; after the peak, the variance's convergence to zero represents an implicit and distributed commit by all the principals on the approximation of that provider's trustworthiness. Hence, even with a byzantine behaviour that the principal exhibit to each possible consumer, TAw enables all the consumers to agree on the reputation to assign to that provider, which approximates the provider's actual trustworthiness with a good degree of precision.

In figures 3.4(c), 3.4(d), 3.4(e), and 3.4(f) it is possible to observe the average reputation and the respective variance computed, respectively, when the 30% and the 50% of the consumers propagate byzantine (i.e., uniformly random in $[0, 1]$) trust information with a random probability. By looking at the average reputations and the variance charts, we can claim that, even in presence of byzantine recommenders, all the correct consumers reach a good approximation of the provider's trustworthiness.

Related Publications

- Mezzetti, N.: Enabling Trust-Awareness in Naming Services. In Proceedings of the 1st International Conference on Trust and Privacy in Digital Business, Lecture Notes in Computer Science (Springer-Verlag), vol. 3184, pag. 20-29, 2004.

- Mezzetti, N.: Design and Evaluation of a Trust-Aware Naming Service. To appear in Computer Systems Science and Engineering Journal, 2005.

Chapter 4

The hTrust Model

4.1 Introduction

In recent years, 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*.

Applications developed for this kind of devices will be both clients and providers of services; in order to foster coordination among these application services, service level agreements (SLAs) will have to be specified and monitored at run-time, to ensure that the Quality-of-Service (QoS) constraints stated in the SLA are being met. Moreover, trust specifications will have to be provided and analysed to predict the trustworthiness of application services.

One of the main achievements of the TAPAS project has been the development of QoS-enabled middleware services capable of meeting Service Level Agreements (SLAs) between application services; moreover, the trust model and architecture described in the previous chapters foster the deployment and interaction of components across organisational boundaries by enabling components to assess the trustworthiness of other peers. Although these approaches to dynamic QoS management and trust reasoning are well-suited in a distributed setting, their applicability is somewhat limited in more dynamic and unstructured environments, such as mobile ad-hoc networks.

Each time an interaction takes place, a mobile application service faces 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 misuse our credit card details; 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 this chapter we present our attempt to extend the trust computational model thoroughly discussed and analysed in the previous chapters to more open environments. In particular, in order to advance the goal of anywhere-anytime computing, and to fully exploit the potential of current wireless technologies to promote non-trivial interactions among entities, we have developed a trust management framework (TMF) that enables portable devices to form, maintain and evolve trust opinions in a very light-weight, and completely decentralised manner. 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 ones. 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. At the moment 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 system, 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 have to make a trust decision about entities we have never seen, or about whom we only have partial knowledge. Distrusting the unknown would cut down the possibility to engage fruitful interactions; however, we cannot blindly trust everyone, as anonymity is very appealing 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 pervasive scenario, 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 has been to develop a formal abstract treatment of trust that meets the above requirements, and offer programmers an infrastructure they can use in practice to build trust-aware systems. We have advanced 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, hTrust exploits these functions to dynamically detect malicious behaviours, and consequently isolate untrusted entities from future interactions.

The remainder of the chapter is structured as follows: Section 4.2 revisits the definition of trust provided in the previous chapters, and spells out the principles and assumptions our model is based upon. In Section 4.3 we describe our trust management model in detail, focusing on trust formation, trust dissemination and trust evolution. Section 4.4 discusses the suitability of hTrust to the mobile setting and reports on some open issues; finally, Section 4.5 compares our approach to related works.

4.2 Principles of Trust

Prior to describing our trust management model, we revisit the definition of trust previously given, so to spell out the characteristics of trust that are most relevant to the mobile setting scenario. Despite extensive studies from sociologists, philosophers, and psychologists, a universally accepted definition of trust is still missing[1]. One of the most commonly accepted definitions, and the one we refer to, is from sociologist Diego Gambetta [19]:

> "... 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 observation is that trust is *subjective*: 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; vice-versa, it increases if

[1] See [33] for an analysis of various definitions of trust.

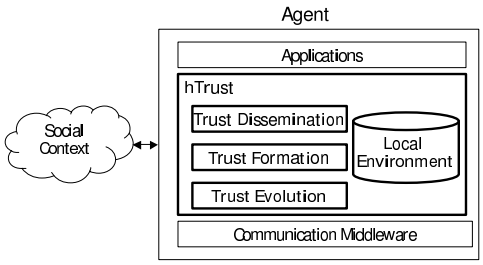

Figure 4.1: Trust Management Model Overview.

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 for mobile ad-hoc systems 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 work, 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 the remainder of this chapter, 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 easy, and malicious agents could exploit it to repeatedly misbehave, without being isolated. Authentication systems based on public-key cryptography (e.g., [12, 30]) do not solve the issue, as there is nothing there preventing an agent to have multiple pairs of keys, thus multiple identities, so that misbehaviours 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, self-organised public-key management system, and that the agent is known in the system by means of a pseudonym that may be the public key itself. 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 misbehaviours), and alert the application about their presence.

4.3 hTrust

An overview of the trust management model we have developed is depicted in Figure 4.1. 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: *direct experiences*, *credentials* and *recommendations*. Direct experiences represent an agent's history of interaction (in this case, past interactions between a and b); they are kept in the trustor's *local environment* by the TMF. Credentials are somehow the symmetric of direct experiences: they represent what other agents thought of us in previous interactions (for example, what agent x thought of b); they are kept in the trustee's local environment by the TMF. Recommendations are trust information coming 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 of the trustor agent, to predict the trustee's trustworthiness. 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 update a's local environment.

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.

4.3.1 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.

Trust Data Model

As we said before, a trustor a forms a trust opinion about a trustee b based on a's direct experiences with b, b's credentials, and recommendations coming from the social context.

The trustor's direct experiences with b are processed by the TMF and kept locally in the form of a single *aggregated trust information* tuple:

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

The meaning of the tuple is as follows: agent a trusts agent b at level l to carry on service s in context c. For example, we may specify that Alice (a) trusts Bob's eBookshop (b) at level 0.8 (l) to sell books (s) that are specialised on travel (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 vice-versa. 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 [13] 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. If only an agent's direct experiences are aggregated, we talk about *trust reflexivity* (i.e., the agent trusts only his own experiences). However, in some circumstances, we may want to take other agents' experiences into account as well; we talk, in this case, of *trust transitivity*. Both direct and indirect experiences may thus be combined in a single aggregated tuple: we will discuss in Section 4.3.3 how the TMF on each agent maintains this information updated. In this work, we are not interested in the specific ontology used to encode service and context information (parameters s and c respectively); we thus simplify the notation and describe an aggregated trust tuple made by agent a about b as $at = [a, b, l, k, t] \in \mathcal{A}$, $\mathcal{A}$ being the set of all aggregated tuples.

Aggregated tuples are the primary source of information to predict trust. When this information is not available (e.g., because no interaction has ever happened in the past between the trustor and the trustee), or in case this information is not reliable (e.g., because it is outdated), the trustor may ask other agents in the social context to provide him with recommendations. For example, Alice may be willing to buy books from Bob's eBookshop provided that it has been recommended by Clare (agent x). A recommendation tuple sent by x about agent b looks like:

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

or, simply, $rt = [x, b, l, k, t]_{SK_x} \in \mathcal{R}$, $\mathcal{R}$ being the set of all recommendations. A recommendation is thus computed by signing the local aggregated tuple; a signature is necessary to prove the recommendation's authenticity. We refer to x and b as to the agent's names; they are the piece of information they are known for in the system (e.g., their public key). We will discuss in Section 4.3.2 the protocol used by the TMF to propagate recommendations in the social context.

Aggregated tuples and recommendations condense an arbitrary long history of direct and indirect interactions into a single tuple. While this has the advantage of minimising the memory required to store trust information, it has also the drawback of losing proof of individual interactions. This approach has been proven to have a number of limitations [37]: for example, an agent cannot defend himself against defamation, nor can he self-recommend himself. On the other hand, dealing only with evidences of single interactions would saturate the system quickly. hTrust overcomes the limitations of these two extreme approaches by combining aggregated tuples and recommendations with credentials. Each agent b carries with him (i.e., in his local environment) a *portfolio of credentials*, that is, a set of letters of presentation detailing how trustworthy b has been in one or more previous interactions. Each credential looks like:

$$[x,\ b,\ l,\ s,\ c,\ n_{from},\ n_{to},\ t]_{SK_x},$$

meaning that agent x considers b trustworthy at level l to carry on service s in context c. This trust level refers to a set of transactions happened in the past between x and b; assuming each

agent counts the number of interactions occurred with any other agent in the system, the trust level in the credential refers to the sequence of transactions numbered n_{from} to n_{to}, the last of which happened at time t. When n_{from} coincides with n_{to}, the credential is a proof of an individual transaction. For example, Bob's eBookshop may have a credential, given to him by Clare, stating that Clare herself considered Bob trustworthy at level 0.9 when she bought there travel books during their 10^{th} transaction ($n_{from} = n_{to} = 10$) which happened at time t. Each credential is signed with the *accreditor*'s (i.e., x) private key, so to prove its authenticity; the trustee b can thus use its set of credentials both to self-recommend himself to other agents, and to defend himself against defamation. We will detail in Section 4.3.2 how credentials are formed and exchanged. To simplify the discussion, we will describe a credential tuple given by agent x to b as $ct = [x, b, l, n_1, n_2, t] \in \mathcal{C}$, $\mathcal{C}$ being the set of all credentials.

We said before we are not interested in particular services and contexts to which trust information refers. There is, however, a service of particular importance, provided by virtually every agent in the system, that we model explicitly: the service of supplying the recommendations themselves. In human interactions, we tend to value 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. Agents are thus judged based on the quality of the recommendations they give, in the same way they are assessed for any other service they provide. Information about the trustworthiness of a recommender agent x can thus be kept by a trustor agent a in the form of an aggregated tuple, where the service parameter s is set to 'recommender':

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

The interpretation of this tuple is the same provided for aggregated tuples: a trusts x at level l to provide recommendations (service s) in a certain context c. For example, Alice may trust Clare at level 0.7 to recommend travel books; this, however, does not imply that Alice trusts Clare also on recommendations regarding what stock market to invest in, as contexts are different. The trustor has knowledge k at time t about this information. The higher the number of good recommendations received from x in the past, the higher the trust level, and vice-versa. 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. We refer to these tuples as to *tacit information* tuples, both to distinguish them from other aggregated information and to cater for the fact that, as we will show in Section 4.3.3, the TMF tacitly extracts them from observations of direct experiences. We will refer to a tacit information tuple as to $tt = [a, x, l, k, t] \in \mathcal{T}$, $\mathcal{T}$ being the set of all tacit information tuples.

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 4.2; before describing step by step the computation it performs, we illustrate the two auxiliary functions Υ_1 and Υ_n that Υ is based upon. In the following, when a formula applies to both aggregated tuples, credentials and recommendations, we will simply refer to a generic tuple $gt \in \mathcal{G}$, $\mathcal{G}$ being $\mathcal{A} \cup \mathcal{C} \cup \mathcal{R}$.

Function Υ_1. When predicting trust, the trust level l expressed in a generic tuple gt has to be scaled down, depending on the knowledge the tuple embeds, and on the time elapsed since when that trust opinion was formed. The function Υ_1 serves this purpose: given in input a tuple gt, Υ_1 first computes a new predicted trust value f that takes into consideration the uncertainty inherent to gt; it then derives a predicted trust range, based on the trust level $\pi_l(gt)$ experienced in the past, and the discrepancy between this value and the prediction f. T and N are parameters belonging to the trustor local environment e, T representing the time interval during which interactions are observed, N representing the number of interactions after which knowledge can be considered maximum. For example, let us consider the aggregated tuple $at = [a, b, 0.6, 0.9, t]$ that states that a is very confident ($k = 0.9$) of the trust opinion ($l = 0.6$) he has in b at time t; when applying function Υ_1 to at after an infinitesimal amount of time has passed (i.e., we do not consider time uncertainty), we obtain a very close range: $[0.6 - |0.6 - 0.6 * 0.9|,\ 0.6 + |0.6 - 0.6 * 0.9|\ ] = [0.54, 0.66]$. However, the lower the knowledge, the wider the range that Υ_1 computes; so, for example, given the aggregated tuple $at = [a, b, 0.6, 0.4, t]$, we obtain $[0.6 - |0.6 - 0.6 * 0.4|,\ 0.6 + |0.6 - 0.6 * 0.4|\ ] = [0.24, 0.96]$.

Function Υ_n. As previously discussed, we value trust opinions coming from other agents (i.e., credentials and recommendations) differently, depending on the *quality* of these agents (i.e., whether, in the past, they have provided several and accurate recommendations). Given a set of n credentials or recommendations, Υ_1 is first applied to each of them, to obtain a set of trust ranges that cater for time and knowledge uncertainty; Υ_n is then computed to derive a unique range, by means of a *weighed* average of the individual ranges lower and upper limits. Each weight q_i in the formula represents the quality of the accreditor/recommender i and it is computed based on his trustworthiness l'_i, the trustor's knowledge k'_i about him, and the time elapsed since when i's last credential/recommendation was received. The tacit tuple $tt'_i = [a, i, l'_i, k'_i, t'_i]$ is retrieved from a's local environment e using an auxiliary function $lookup(i, e)$. For example, given a recommendation $rt = [x, b, 0.6, 0.9, t]$ (so that $\Upsilon_1[\![rt]\!]_e = [0.54, 0.66]$), and a tacit tuple $tt = [a, x, 0.9, 0.8, t]$ (so that $q_x = 0.9 * 0.8 = 0.72$, without considering time uncertainty), we obtain, in a single step of the summation, the addends $0.54 * 0.72 = 0.39$ and $0.66 * 0.72 = 0.48$, for the lower and upper limits respectively (i.e., the breadth of the range is preserved, although scaled down to cater for the quality of the recommender). Note that not all credentials/recommendations are used to compute a prediction: only those coming from agents whose quality q_i is greater than a minimum level η contribute to the final result. Also, to avoid considering the same tuple more than once, we assume that only credentials/recommendations with timestamp $\pi_t(gt_i) > \pi_t(tt_i)$ are processed.

Trust Formation Function Υ. Given in input an aggregated tuple $at \in \mathcal{A}$, taken from a's local environment $e \in \mathcal{E}$, a set of credentials $C \in \wp(\mathcal{C})$ supplied by the trustee b, and a set of recommendations $R \in \wp(\mathcal{R})$ coming from the social context, the trust formation function Υ first computes three independent trust ranges, by means of the auxiliary functions Υ_1 and Υ_n. It then uses a *customising function* h_1 to synthesise one trust range, given three ranges in input. This function will vary from agent to agent, depending on the natural disposition to trust of the agent itself. For example, h_1 may consider the local aggregated trust information only, thus relying solely on the trustor's past experiences (*trust reflexivity*); vice-versa, credentials and recommendations alone can be used (*trust transitivity*), for example, to delegate trust to third parties. More generally, a combination of the three will be used; for example,

$$
\begin{aligned}
\Upsilon_1 &: \mathcal{G} \to \mathcal{E} \to [-1,1] \times [-1,1] \\
\Upsilon_1[\![gt]\!]_e &= \begin{cases} [\max(-1, \pi_l(gt) - |\pi_l(gt) - f|), \min(\pi_l(gt) + |\pi_l(gt) - f|, 1)] \,, \\ f = \pi_l(gt) * \pi_k(gt) * \max\left(0, \frac{T-(t_{now}-\pi_t(gt))}{T}\right) \text{ if } gt \in \mathcal{A} \cup \mathcal{R}, \\ f = \pi_l(gt) * \frac{(\pi_{to}(gt)-\pi_{from}(gt)+1)}{N} * \max\left(0, \frac{T-(t_{now}-\pi_t(gt))}{T}\right) \text{ if } gt \in \mathcal{C} \end{cases} \\
\Upsilon_n &: \wp(\mathcal{G}) \to \mathcal{E} \to [-1,1] \times [-1,1] \\
\Upsilon_n[\![\{gt_i | i \in [1,m]\}]\!]_e &= \begin{cases} [l_{low}, l_{high}] \,, \\ l_{low} = \frac{\sum_i \{\pi_1(\Upsilon_1[\![gt_i]\!]_e) * q_i | q_i > \eta\}}{\sum_i (q_i | q_i > \eta)}, \;\; l_{high} = \frac{\sum_i \{\pi_2(\Upsilon_1[\![gt_i]\!]_e) * q_i | q_i > \eta\}}{\sum_i (q_i | q_i > \eta)} \\ q_i = \max(\eta, l'_i * k'_i * \max\left(0, \frac{T-(t_{now}-t'_i)}{T}\right)) \,, \;\; tt'_i = lookup(i,e) = [a, i, l'_i, k'_i, t'_i] \end{cases} \\
\Upsilon &: \mathcal{A} \times \wp(\mathcal{C}) \times \wp(\mathcal{R}) \to \mathcal{E} \to [-1,1] \times [-1,1] \\
\Upsilon[\![(at, C, R)]\!]_e &= h_1(\Upsilon_1[\![at]\!]_e, \Upsilon_n[\![C]\!]_e, \Upsilon_n[\![R]\!]_e)
\end{aligned}
$$

Figure 4.2: Trust Formation Function Υ. The following auxiliary functions have been used: $\pi_1(l_1, l_2) = l_1$; $\pi_2(l_1, l_2) = l_2$; also, π_{field} projects a tuple onto the specified field, that being the trust value l, the knowledge k, the time t, or the range of transactions a credential refers to (i.e., $from$ and to).

$h_1([l_1, l_2], [l'_1, l'_2], [l''_1, l''_2]) = [w_1 * l_1 + w_2 * l'_1 + w_3 * l''_1,\ w_1 * l_2 + w_2 * l'_2 + w_3 * l''_2]$, with $w_1 + w_2 + w_3 = 1$, $0 \leq w_i \leq 1$. We represent the final prediction as an interval, rather than as a single value, to cater for the approximate nature of trust due to incomplete knowledge. Mobile applications can derive a single trust value out of the predicted range, either by means of one of hTrust customising functions (e.g., see h_3 in Section 4.3.3), or by applying some more advanced, application-specific reasoning, if needed. Note that there may be an overlap between credentials and recommendations, that is, the accreditor of a credential may also be a recommender. To avoid the same experience being accounted for more than once by the trust formation function, we require the sets C and R to be properly pruned, so that an agent appears either as an accreditor or as a recommender, but not as both.

4.3.2 Trust Dissemination

The trust formation function described before uses credentials and recommendations to predict the trustworthiness of a trustee b; these tuples become particularly important when b is unknown to the trustor a, so that there is no aggregated trust information to rely on. In these circumstances, the following protocol for the dissemination of credentials and recommendations is used; the protocol 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.

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. $b \to a : \ ct_i,\ i \in [1, m]$. 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 an independent key management system 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 too uncertain. In this case, the TMF queries the social context to receive 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 again 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: $a \rightarrow b : [a,\ b,\ l',\ n,\ n,\ t]_{SK_a}$, and $b \rightarrow a : [b,\ a,\ l'',\ n,\ n,\ t]_{SK_b}$. 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 neighbours. Note that the credential refers only to the just completed interaction (numbered n by both agents, as this is the n^{th} interaction between the two of them). The more an agent interacts, the higher the number of credentials he has to manage; to avoid saturation, a may ask b to provide him with a credential that summarises a sequence of n previous interactions. To do so, a sends b the credentials $[b, a, l_i, i, i, t_i]_{SK_b}$ for $i \in [m, m+n]$, and b replies with a single credential $[b, a, \sum_{i=m}^{m+n} l_i/n, m, m+n, t_{m+n}]_{SK_b}$. For example, given $[b, a, 0.7, 1, 1, t_1]_{SK_b}$,$[b, a, 0.8, 2, 2, t_2]_{SK_b}$,$[b, a, 0.8, 3, 3, t_3]_{SK_b}$, $[b, a, 0.4, 4, 4, t_4]_{SK_b}$ and $[b, a, 0.8, 5, 5, t_5]_{SK_b}$, a single credential $[b, a, 0.7, 1, 5, t_5]_{SK_b}$ can be computed. Note that a cannot cheat and do this computation by himself, as the accreditor's signature is required. Also, although the content of each credential can be read using the accreditor's public key PK_b, a cannot selectively drop credentials (for example, to discard a negative evidence) without being discovered, as they are sequentially numbered; for the same reason, b cannot provide a with a falsified averaged credential. Concealments of individual credentials to agents other than the accreditor are still possible; however, as we have seen in the previous section, an individual credential has much lower weight than an averaged one, and to obtain an averaged one a complete sequence is needed.

The trust formation function Υ is applied at three different stages of the recommendation 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 $h_2 : [-1, 1] \times [-1, 1] \rightarrow \{0, 1\}$ 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: $h_2(l_1, l_2) = 0$ 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 $h_2(l_1, l_2) = 0$ if $l_2 - l_1 > \delta l_{max}$, 1 otherwise; that is, if the range of opinions is too broad, ask for more information to narrow it down.

$$
\begin{array}{rcl}
\Phi & : & [-1,1] \times \wp(\mathcal{C}) \rightarrow \mathcal{E} \rightarrow \mathcal{E} \\
\Phi[\![(\tilde{l}, C)]\!]_e & = & \left\{ \begin{array}{l} e \setminus \{at = [a,b,l,k,t]\} \cup \{at' = [a,b,l',k',t']\} | \\ at = lookup(b,e) \wedge l' = h_4(\tilde{l}, l, h_3(\Upsilon_n[\![C]\!]_e)) \wedge \\ k' = \min(k + k_{min}, 1) \wedge t' = t_{now} \end{array} \right. \\
\Phi[\![(\varepsilon, C)]\!]_e & = & \left\{ \begin{array}{l} e \setminus \{at = [a,b,l,k,t]\} \cup \{at' = [a,b,l',k',t']\} | \\ at = lookup(b,e) \wedge l' = h_4(\varepsilon, l, h_3(\Upsilon_n[\![C]\!]_e)) \wedge \\ k' = k \wedge t' = \max_i(\{\pi_t(ct_i), ct_i \in C\}) \end{array} \right.
\end{array}
$$

Figure 4.3: Aggregation Function Φ.

When entering a social context for the first time, an agent has no history, thus no portfolio he can use to prove his trustworthiness. Having no history is distinctive of both genuine newcomers, but also of cheating agents, that are repeatedly creating new identities to conceal past misbehaviours. 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 encourage interactions.

4.3.3 Trust Evolution

As we discussed in Section 4.3.1, trust is predicted based on the perceived behaviour of agents in their past interactions. A fundamental component of a TMF is thus trust evolution, that is, the continuous self-adaptation of trust information kept 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 tuple). Also, we illustrate how the maintenance of trust information plays a key role in the detection of malicious agents.

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 his credentials.

Figure 4.3 contains a formal definition of the aggregation function Φ. The first equation describes the case where the aggregated tuple $at = [a, b, l, k, t]$, 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 $h_3(\Upsilon_n[\![C]\!]_e)$

(that is, b's trustworthiness computed using the newly received credentials C). Two customising functions are used: first, h_3 computes a trust value out of a trust range. For example, a cautious agent that tends to distrust other agents may choose $h_3(l_1, l_2) = l_1$ (that is, the lower trust value of a range); another agent who naturally tends towards trust may choose $h_3(l_1, l_2) = l_2$ instead. More generally, $h_3(l_1, l_2) = w_1 * l_1 + w_2 * l_2$, with $w_1 + w_2 = 1$, $0 \leq w_i \leq 1$. Second, function h_4 combines the three trust opinions to derive the new one. The general structure of h_4 is the following: $h_4(l_1, l_2, l_3) = w_1 * l_1 + w_2 * l_2 + w_3 * l_3$, with $w_1 + w_2 + w_3 = 1$, $0 \leq w_i \leq 1$, and l_1 being b's trustworthiness as perceived by a in the just completed interaction, l_2 being the opinion previously held by a about b, and l_3 being b's expected trustworthiness based on the received credentials. Different choices of the weights w_i correspond to different dispositions to trust. Examples include: $w_1 = 0.5, w_2 = 0.5, w_3 = 0$, 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 $w_1 = 1/n, w_2 = (n-1)/n, w_3 = 0$, 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 credentials 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 credentials C, without an interaction to have actually occurred (because, for example, the predicted trust was too low). In this case, a new trust opinion is computed solely based on l (i.e., the old trust opinion held by a, if any), and $h_3(\Upsilon_n[\![C]\!]_e)$ (i.e., the predicted trustworthiness computed using the newly received credentials C). Note that, in this case, the trustor's knowledge is not incremented: only direct experiences contribute to uncertainty reduction.

The aggregated tuple can then be signed with the trustor's private key and used during the recommendation exchange protocol to answer request for recommendations. We assume that only credentials whose accreditor's quality q_i is higher than η contribute to the newly synthesised trust value (see Figure 4.2 for a definition of q_i). Also, we assume that only fresh credentials are processed, that is, $\pi_t(ct_i) > \pi_t(at)$, ct_i being a credential in C, and at the aggregated tuple we are updating; we are thus guaranteed that the same credential will not be processed more than once in different executions of the aggregation function (although some previously unprocessed credentials may be discarded). Also, we do not aggregate recommendations, to avoid situations where the same experience is processed more than once (for example, both as a credential and as a recommendation previously formed by aggregating the very same credential).

Tacit Information Extraction Function Ψ

After an interaction, the opinion that the trustor a holds about the trustworthiness of recommenders is updated. Maintaining the tacit information tuples updated is crucial, as it allows the trustor to discriminate between good and bad recommendations in future interactions. This activity is performed by the TMF using a tacit information extraction function Ψ whose formal definition can be found in Figure 4.4. The tacit information tuple $tt_i = [a, i, l_i, k_i, t_i] \in e$, con-

$$\Psi \quad : \quad \mathcal{A} \times \wp(\mathcal{R}) \rightarrow \mathcal{E} \rightarrow \mathcal{E}$$

$$\Psi[\![(at = [a, b, l', k', t'], R)]\!]_e \quad = \quad \left\{ \begin{array}{l} e \setminus \{tt_i = lookup(i, e) = [a, i, l_i, k_i, t_i], \forall rt_i \in R\} \cup \\ \{tt'_i = [a, i, l'_i, k'_i, \pi_t(rt_i)], \forall rt_i \in R\} \mid \\ k'_i = \min(k_i + k_{min}, 1) \ \wedge l'_i = \left\{ \begin{array}{l} \max(-1, h_5(l_i, \delta l_i)) \text{ if } \delta l_i > \delta_{max} \\ \min(h_5(l_i, \delta l_i), 1) \text{ if } \delta l_i \leq \delta_{max} \end{array} \right. , \\ \delta l_i = |l' - h_3(\pi_l(rt_i) - |\pi_l(rt_i) - \pi_l(rt_i) * \pi_k(rt_i) * \frac{T-(t_{now}-\pi_t(rt_i))}{T}|, \\ \pi_l(rt_i) + |\pi_l(rt_i) - \pi_l(rt_i) * \pi_k(rt_i) * \frac{T-(t_{now}-\pi_t(rt_i))}{T}|)| \end{array} \right.$$

Figure 4.4: Tacit Information Extraction Function Ψ.

taining information about the trustworthiness l_i of agent i as a recommender, is updated based on: the new aggregated opinion l' about b with whom a has just interacted, and the recommendation $rt_i \in R$ about b that recommender i has given to a. Note that only recommendations (and not credentials) are processed at this stage: this is because we are interested in identifying agents with whom there is an affinity of (long-term) reasoning, while credentials refer to very specific transactions only. Also, to avoid considering the same recommendation many times, only recommendations rt_i with timestamp $\pi_t(rt_i) > \pi_t(tt_i)$ are processed.

First, the discrepancy δl_i between the trustor's new opinion l' and the recommender's opinion $\pi_l(rt_i)$ is computed. For example, if we consider time uncertainty to be infinitesimal, and the trustor's new opinion l' to be 0.9, then the discrepancy of opinions with respect to recommendation $rt = [x, b, 0.7, 0.8, t]_{SK_x}$ would be: $\delta l_x = 0.9 - h_3(0.7 - |0.7 - 0.7 * 0.8|,\ 0.7 + |0.7 - 0.7 * 0.8|) = 0.9 - h_3(0.56, 0.84)$; for $h_3(l_1, l_2) = (l_1 + l_2)/2$, that would be $\delta l_x = 0.9 - 0.7 = 0.2$. 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 $n = k_i/k_{min}$. Vice-versa, 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 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 vice-versa.

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" [21]), 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 drops 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" [21]). In this model, the punishment 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.

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 malicious recommenders, not of unreliable service providers, through analysis of the recommendations they provide. Detection and isolation of unreliable 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 this work. An unsatisfactory transaction has repercussions on 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 work.

4.3.4 Local Environment

Data	
Aggregated Tuple	$[a, x, l, k, t]$
Tacit Tuple	$[a, x, l, k, t]$
Credential	$[x, a, l, n_1, n_2, t]_{SK_x}$
Parameters	
Interval of Relevant Observations	T
Number of Relevant Interactions	N
Maximum Discrepancy of Opinions	δ_{max}
Single Increment of Knowledge	k_{min}
Minimum Trust Level	η
Customising Functions	
Given 3 ranges, compute a new one	h_1
Given a range, decide whether the prediction is precise enough	h_2
Given a range, compute an opinion	h_3
Given 3 opinions, compute a new one	h_4
Given an opinion and a discrepancy, compute a new opinion	h_5

Table 4.1: Agent a's Local Environment

Table 4.1 summarises the information that forms an agent's local environment: the data is continuously updated by the TMF using the aggregation function, the tacit information extraction function, and during the agent's interactions. 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 work to supply values for the parameters.

4.4 Discussion

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 fate.

Second, the resource demands imposed by an implementation of the framework can be minimal. Direct experiences, for example, are not individually 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 ex-

traction 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 resource capabilities of the device, etc.).

hTrust simulates the human approach to trust management in the computer world 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. Further evaluation is necessary to derive guidelines to write profiles that are both effective and that minimise the burden imposed on the user of the device.

Apart from customising functions, the behaviour of our trust management model depends on a number of parameters, as listed in Table 4.1. We have not discussed in this chapter possible values of these parameters. Further experimentation is needed to empirically associate values to these variables. Initial results obtained during simulation of hTrust confirmed that different choices of these parameters impact on the accuracy of trust prediction as well as on resource usages. A unique optimal choice does not exist, as these parameters are domain-dependent; analysis of the target domain is thus necessary to associate meaningful values to these parameters.

4.5 Related Work

The problem of trust management is very broad and it has been dealt with by different research communities. In this section, we review the literature with a focus on common limitations of currently available approaches when applied to the mobile setting.

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 [20, 5, 30, 12]) 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 [36] 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 runtime component interface violations have been suggested (e.g., [16, 23]). 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.

Sultan [22] 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 [9] 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 distributed policy management approaches have been defined (e.g.,[8, 17]); however, issues such as trust evolution and subjective trust reasoning have not been tackled.

In [2], 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 predict trust, 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. Also, no mechanism to dynamically re-evaluate trust decisions is discussed. In [10], a mechanism to detect and isolate misbehaving nodes at the network (routing and forwarding) level is proposed. The mechanism 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 [18], 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. In [31], a mechanism for the management of distributed reputation in mobile ad-hoc networks is presented, that is able to effectively detect malicious recommenders based on the idea of 'recommendation reputation', a very similar concept to the 'quality' parameter we use in our model. We believe hTrust improves over this model, first by making a clear distinction between trust and knowledge, concepts that appear to be confused in [31], and second by providing a more advanced credential exchange protocol than a simple periodic exchange. In [37], a different approach to distributed reputation management is proposed, that prescribes the use of first-hand experiences only, to circumvent the limitations of recommendations. We believe it to be unfeasible to work with first hand experiences only, as this would quickly saturate the system; hTrust thus proposes an approach that combines both first-hand experiences and recommendations.

Various formalisms of trust have been proposed, in order to help reasoning about trust. In [25], 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 [13], a formal model for trust formation/negotiation, evolution and propagation is presented; however, the protocols for exchanging recommendations and for dynamically re-evaluating trust

are not provided. Similar considerations hold for the formal trust models described in [7] (based on probability theory) and [38] (based on lattice, denotational semantics and fixed point theory). In this work, we have tackled the problem of trust management from an engineering point of view, providing an operational model that programmers can actually exploit to build trust-aware systems.

Related Publications

- L. Capra. Towards a Human Trust Model for Mobile Ad-hoc Networks. In Proc. of the 2^{nd} UK-UbiNet Workshop, Cambridge, UK, May 2004.
- L. Capra. Engineering Human Trust in Mobile System Collaborations. In Proc. of the 12^{th} International Symposium on the Foundations of Software Engineering (SIGSOFT 2004/FSE-12), pages 107-116, Newport Beach, CA, USA, November 2004. ACM Press.

Chapter 5

Conclusions

With respect to the overall TAPAS objective of developing novel methods, tools, algorithms and protocols that support the construction and provisioning of Internet application services, the work described in this deliverable has contributed the following.

The SIR Reputation Model. Our first major contribution is the SIR reputation model, that is, a socially-inspired reputation's model and metric that enable service consumers and producers to be mutually aware of their trustworthiness. SIR has been both formally discussed and practically applied, thus demonstrating its suitability and power for evaluating principal trustworthiness within practical collaborative scenarios. The results that are obtained through its evaluation within the TAw architecture show that SIR enables the implementation of adaptive, robust and scalable reputation management systems. We envision possible instances of the SIR reputation model within other kind of software architectures such as reputation based load-balancing policies and global computing applications (e.g., virtual enterprises, mobile code).

The TAw Architecture. Our second major contribution is TAw, our Trust-Aware Naming Service. To the best of our knowledge, TAw represents the first general-purpose middleware architecture for reputation management. TAw transparently manages trustworthiness of physical and logical resources in middleware systems deployed over large scale distributed contexts as well as for managing the trustworthiness of principals that interact or collaborate via application level services in a business-to-business context; in addition, the experimental results that we presented prove that TAw enables reputation management in virtual enterprises scenarios in an adaptive, robust, and scalable manner. Specifically, the presented results demonstrate that TAw enables principals within a dynamic environment to efficiently approximate other principal trustworthiness even in presence of malicious information. However, more experiments are being performed so as to evaluate robustness in presence of malicious principal collusions instead of the random introduction of noise within the system. Moreover, this work also prove that the SIR reputation model is to be considered a good model for evaluating principal trustworthiness within practical collaborative scenarios. We envision possible applications of the SIR reputation model within other kind of software architectures such as applications for ad-hoc networking, reputation based load-balancing policies, and global computing applications (e.g., virtual enterprises, mobile code).

The hTrust Model. Our last major contribution has been the formalisation of the hTrust trust management model and framework for the development of trust-aware systems and appli-

cations for the ubiquitous environment. The rapid and enormous success of wireless networking technologies and portable devices suggests that pervasive services will play a key role in the future. hTrust provides a framework to develop trust-aware pervasive computing applications and services; in particular, it relieves the programmer from dealing with trust formation, trust dissemination, and trust evolution issues. Various sources of trust information are combined in hTrust: direct experiences, credentials and recommendations; precise algorithms have been illustrated to combine and maintain this information dynamically. 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 customisable functions and parameters.

Bibliography

[1] A. Abdul-Rahman and S. Hailes. A distributed trust model. In *New Security Paradigms Workshop*, 1997.

[2] 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, November 1997.

[3] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *33rd Hawaii International Conference on System Sciences*, 2000.

[4] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *10th International Conference on Information and Knowledge Management*, 2001.

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

[6] F. Azzedin and M. Maheswaran. Trust brokering system and its application to resource management in public-resource grids. In *2004 International Parallel and Distributed Processing Symposium (IPDPS 2004)*, April 2004.

[7] 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, November 1994.

[8] M. Blaze, J. Feigenbaum, and A.D. 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, April 1998. Springer-Verlag.

[9] 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.

[10] S. Buchegger and I.Y. Le 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, March 2003.

[11] S. Buchegger and J.-L. Le Boudec. A robust reputation system for mobile ad-hoc networks. In *P2PEcon*, June 2004.

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

[13] 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, September 2003.

[14] G. Caronni. Walking the web of trust. In *IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'00)*, pages 153–158, 2000.

[15] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *6th ACM Symposium on Principles of distributed computing*, pages 1–12. ACM Press, 1987.

[16] S.H. Edwards, G. Shakir, M. Sitaraman, B.W. 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.

[17] K. Seamons 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.

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

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

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

[21] 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.

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

[23] R. Jagannathan and P.A.G. 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.

[24] A. Jøsang. An algebra for assessing trust in certification chains. In J. Kochmar, editor, *Network and Distributed System Security Symposium (NDSS'99)*. The Internet Society Press, 1999.

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

[26] A. Jøsang. A logic for uncertain probabilities. *International Journal of Uncertainity, Fuzziness and Knowledge-Based Systems*, 9:279–311, 2001.

[27] A. Jøsang. The beta reputation system. In *15th Bled Electronic Commerce Conference*, 2002.

[28] A. Jøsang and S. Knapskog. A metric for trusted systems. In *21st National Security Conference*, 1998.

[29] A. Kermarrec, L. Massoulié, and A. Ganesh. Probabilistic and reliable dissemination in large scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(2), 2003.

[30] 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, November 2001.

[31] J. Liu and V. Issarny. Enhanced Reputation Mechanism for Mobile Ad Hoc Networks. In *Proc. of the* 2^{nd} *International Conference on Trust Management (iTrust)*, volume 2995, pages 48–62, Oxford, UK, March 2004. LNCS.

[32] U. Maurer. Modeling a public-key infrastructure. In E. Bertino, editor, *1996 European Symposium on Research in Computer Security (ESORICS'96), Lecture Notes in Computer Science*, volume 1146, pages 325–350, 1996.

[33] D.H. McKnight and N.L. Chervany. The Meanings of Trust. Management Information Systems Research Center, University of Minnesota, 1996. Working Paper 96-04.

[34] N. Mezzetti. SIR: a model of social reputation. Technical Report UBLCS-2004-15, Department of Computer Science, University of Bologna, October 2004.

[35] N. Mezzetti. A socially inspired reputation model. In S. Katsikas, S. Gritzalis, and J. Lopes, editors, *Public Key Infrastructures, Lecture Notes in Computer Science*, volume 3093, pages 191–204, June 2004.

[36] George 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, January 1997.

[37] P. Obreiter. A Case for Evidence-Aware Distributed Reputation Systems - Overcoming the Limitations of Plausibility Considerations. In *Proc. of the* 2^{nd} *International Conference on Trust Management (iTrust)*, volume 2995, Oxford, UK, March 2004. LNCS.

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

[39] R. Yahalom, B. Klein, and T. Beth. Trust relationships in secure systems – a distributed authentication perspective. In *1993 IEEE Symposium on Security and Privacy*, pages 150–164, May 1993.

[40] W. Yao, K. Moody, and J. Bacon. A model of OASIS role-based access control and its support of active security. *ACM Transactions on Information and System Security*, 5(4), 2002.