Diploma Thesis

"AcTrust: A Security Model for Trust Based Computing"

By: Ayşe Moralı

Darmstadt University of Technology
Department of Business Administration, Economics and Law
Institute of Business Administration
Commercial Information Technology I
(Development of Information Systems)
Prof. Dr. Erich Ortner

Advisor: Prof. Carlos Varela

Rensselaer Polytechnic Institute
Computer Science Department
Worldwide Computing Group

19.04.2006, Troy/USA

ABSTRACT

The role of distributed system in our daily life is getting more and more important. The resources of the cyber world are consumed by peers independent from their physical locations, mobile codes interact with each other and their environment on behalf of people. Grid computing provides potential benefits to applications, but as the responsibilities and the intelligence of such systems increase, security threats that they pose to the applications increases, too.

In order to prevent these distributed systems from attacks, they have to be adjusted with secure components. In such systems, openness and dynamic reconfiguration is also inevitable for scalability. Furthermore, transparent information flow between the peers can help increase the efficacy for its participants and reduce user's cognitive load.

In this thesis, we describe a reputation based trust model including a language and its operational semantics, in order to reason about security problems of open and distributed systems, as well as to prove its main security properties. We furthermore, give some examples based on two common application fields of distributed computing: e-commerce and distributed file sharing.

ACKNOWLEDGEMENT

I would gratefully acknowledge Prof. Carlos Varela for advising me throughout this thesis. I would like to thank Prof. Erich Ortner for giving me the opportunity to write my thesis abroad. Furthermore I would like to thank Worldwide Computing Group for the comments they made for the improvement of the examples and the Trusted Actor Language. Last but not least I would like to thank my parents, Nurten and Vedat Morali, for their financial and moral support.

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACL	Access Control List
CMM	Coordination Model Manager
p2p	peer-to-peer
SM	Seller Manager
SMAL	Secure Mobile Actor Language
SM	Store Manager
TAL	Trusted Actor Language
TM	Transaction Manager

1. Introduction

1.1 Motivation

Mobile code technologies provide potential benefits to internet based distributed applications, but as the responsibilities and the intelligence of mobile code increase, security threats that it poses to a system increases, too. In complex open environments, such as multiagent system environments, it is often necessary to make the goals explicit for the agent itself. This means that the agents need to be able to identify and measure the threats it faces, and make risk based decisions. One of the mechanisms in order to measure the threats is to obtain information about the earlier cooperative behavior of certain partners.

In our previous work [19] we have compared the most common security instruments that can be applied to an online market place, where actors, which are computational agents distributed over time and space, represent sellers and buyers of the real world. Complementary to that, in this work we are going to concentrate on one of the security instruments that both fulfill as much security requirements as possible, commonly used in social networks, and is potentially applicable to the mobile actor systems.

If we take a look at the most common online market places, like eBay or Amazon we are going to notice that they use a reputation based trust system in order to achieve a transparent overview of the parties that interact. However, building a trust based reputation system poses difficulties in retrieving reputation and calculating a trustful reputation value. In the last five years many researchers have analyzed this topic from different perspectives and proposed various instruments, like trust propagation, results of interaction, popularity, number of interactions, etc., to overcome problems related to reputation based trust. However, most of them were too complicated and only theoretical approaches, if not inconvenient. Furthermore such approaches fall short to describe the exact behavior of the system, since they lack a notion of protocol, which is a fundamental property when security concerns are present.

From this point of view, this work will first give a broad overview of these research, furthermore, as the contribution, define AcTrust, an actor based trust system for solving the major trust problems. Moreover, we have introduced the Trusted Actor Language for trusted actor communication and given its operational semantics, which are both expressive and simple to apply and, as well as proven the basic trust properties of the language. We furthermore give two examples, in order to point out the major application fields, where AcTrust can be used. We conclude this thesis with some future directions that involve potential improvements of the AcTrust model and expand the application fields of the model.

1.2 Terms and Definition

In electronic commerce settings, peer-to-peer (P2P) communities are often established dynamically by peers that are unrelated and unknown to each other. When a peer interacts with another, it usually assumes that the communication partner is not going to perform any unauthorized operations or that it is going to provide a level of quality of service during an interaction. The ideal solution is to have an environment that is fully trusted by all its entities. However, this ideology cannot be achieved in an open dynamic environment, where simultaneously new peers are created.

In existing literature, trust and reputation are defined and analyzed from many different perspectives. *Trust* is a common criteria in human life activities. In an informal way, trust is defined in Merriam-Websters Collegiate Dictionary as "assured reliance on the character, ability, strength, or truth of someone or something". Deutsch [9] makes one of the widely accepted definition of trust and points especially out that trust is a subjective, or agent-centered notion, one in which most of the choices are based on subjective views of the world. Gambeta defines trust in [13] from the view of the probability theory as follows "... trust is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action...". Mui et al. make a definition in [20] that expresses the importance of expectations: "Trust: subjective expectation an agent has about another future behavior based on the history of their encounters".

According to these definitions trust can be roughly defined as the belief that a peer fulfills it security expectations, *trustworthiness* on the other hand is the confirmation that something earns trust.

Wang and Vassileva suggest in [29] two types of trust for P2P systems. First trust in the quality of the service that a host provides, in other words trust in the competence of a host, and second trust in the quality of recommendation delivered after a transaction. Josang and Pope [15] call these two types of trust as *functional trust* and *referral trust*. As Wang et al., Tajeddine et al. distinguish in [24] between two types of trust and describe quality of recommendation as a trust that is affected by four parameters: activity, similarity, popularity and cooperation.

Reputation in general is defined in Merriam-Websters Collegiate Dictionary as overall quality character as seen or judged by people in general. Wilson makes an operational definition of reputation as a characteristic or attribute ascribed to one person, industry, etc. by another. Operationally, this is usually represented as a prediction about likely future behavior. It is, however, primarily an empirical statement. Its predictive power depends on the supposition that past behavior is indicative of future behavior.

As in trust there are also different types of reputations. Mui et al. suggests a comprehensive differentiation in [20] and studies them in a set of evolutionary games. According to this approach, reputation can be either achieved from an individual peer or a group of peers. They furthermore differ between direct or indirect individual reputation. *Direct reputation* can be driven form an interaction or be achieved through observations. *Indirect reputation* on the other hand can be achieved through prior derived reputation, group derived reputation and propagated reputation.

Turapani et al. [26] suggests increasing the robustness of an agent system by using reputation. Here the authors have simulated two types of reputation. One in which the reputations of all agents are maintained *centrally* and second in which they are *distributed*. They furthermore simulated and compared two ways of achieving distributed simulation management: *post-processing* and *preprocessing*. In the post-processing they suggest voting in order to judge the correctness of the output and in preprocessing they use a probabilistic function in order to select a set of agents based on reinforcement values, which are obtained based on the correctness of the results the agent produces in performing a task that is given to him.

Kinateder et al. defines reputation in [17] as the average trust of all other entities towards a single entity and bridges trust and reputation. They also note that reputation has a global aspect, whereas trust is viewed from both a local and subjective point.

1.3 Structure of the Work

This thesis describes in Chapter 2, the actor model, the secure actor model and the transactors model, which extend the actor model with security and consistency features. Later on in this chapter, we give a broad overview of the most recent and relevant trust management models for P2P communication systems and agent based communication systems. Chapter 2 concludes with the state of the art of reputation based trust systems.

In Chapter 3 we first introduce the main properties of AcTrust, a reputation based trust model for secure actor communication, and later on we give the mathematical model we suggest for the calculation of trust, as well as the trust policy of the model. We conclude Chapter 3 with Trusted Actor Language, which is a layer above an actor language, give its operational semantics, which expresses how trust is represented and propagated in distributed computing, and prove the major properties of AcTrust using the rules of the actor language.

In Chapter 4 we first briefly define the Marketplace Scenario that we have suggested in [19] and BitTorrent, and later on give examples using these models to clarify how AcTrust can be implied to E-commerce and P2P applications.

The thesis is concluded with future directions and possible extensions of the AcTrust Model.

2. Background

2.1 Actors

In this section we are going to describe actors, as Agha et al. described in [1]. We will first give the most specific properties of actors that differ them from passive objects and furthermore we will describe the Secure Mobile Actor Language (SMAL) and it's computational semantics, which is evaluated by Varela [27]. Later on we are going to describe some Properties of Transactors [12], which are actors that fulfill some ineluctable properties like fault-tolerance or global consistency for secure communication in distributed environments.

Actors are computational agents that are distributed over time and space. They encapsulate a state and a thread of control. The communication between the agents is realized with point-to-point messages that are buffered in an input queue. The messages that are received by the interface of an actor are interpreted with public methods, which operate on the state of the actor. Furthermore, the actors are associated with a conceptual location and a unique name, which can be used as a reference by other actors. The actors communicate with other actors or their surrounding in order to fulfill their duties. The local computation of an actor in response to the message received can be modeled by the usual sequential constructs. While processing a message there are four basic actions that an actor may perform (see figure 2.1):

1. changing the local state and becoming ready to process the next message in its mail queue,
2. migrating to a new location,
3. creating a finite set of actors with some initialized behaviors, and
4. sending messages to peer actors.

Individual actors are the smallest coordination unit in the middleware model. The basic actor language proposed by Agha, Mason, Smith and Talcott [2] provides

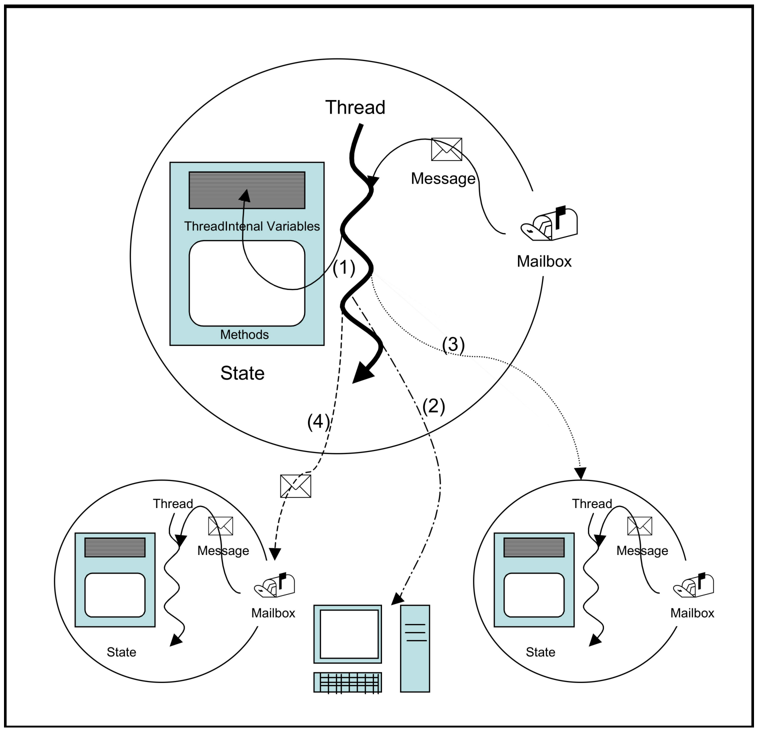

Figure 2.1: Architecture of an actor, and the actions it performs.

a mechanism for specifying the creation and manipulation of actors. Each actor is a unit of computation encapsulating data and behavior. The behavior defines how the actor reacts on receipt of a message [25].

2.2 Secure Mobile Actors

In order to achieve mobility and security towards robust distributed computing systems, Toll and Varela [25] have extended the basic actor language with primitives for mobility and security and then developed an operational semantics for these extensions. Secure Actor Language describes an actor's behavior by a lambda abstraction, which embodies the code to be executed when a message is received.

2.2.1 Model Properties

Actors are inherently independent, concurrent and autonomous which enables efficiency in parallel execution and facilitates mobility. Each actor is a unit of computation encapsulating data and behavior. Communication between actors is purely asynchronous and guaranteed. That means, when a message is sent, the model guarantees that the destination actor will receive the message; however, it does not guarantee the order of message arrival or the order of processing.

The universal actors extend actors with locations, mobility, and the concept of universal names and universal locations. *Locators* represent references that enable communication with universal actors at a specific location. An actor's location abstracts over its position relative to other actors. Each location represents, like an actor's configuration, an actor's run-time environment and serves like an encapsulating unit for local resources. Ubiquitous resources have a generic representation. Resource-to-actor translations are stored in *resource maps*, which are maps between global names and the names of local actors, who fill a resource's role. Actors keep references, which get updated upon migration to resources at new locations. They can also keep references to non-ubiquitous resources by using resource attachment and detachment operations. Actors do not only interact with other actors, but also with their environments.

Actors in secure mobile actor model restrict communication and migration behaviors to other actors within specific *Access Control Lists* (ACL). Using this method, no unprivileged actor can gain access to a resource. Every actor or location has an ACL, which contains a list of actors allowed to migrate into the location or send/receive messages. ACLs can only be changed by locations that the actor is in consideration, or by a resident actor in case of passive locations.

2.2.2 Secure Mobile Actor Language

Secure Mobile Actor Language (SMAL) is a language that has been expanded in layers. In the the first layer there is the call-by-value λ-calculus, which is extended with primitives for actor communication to actor language, with mobility primitives to mobile actor language and with security primitives to SMAL. In this thesis we are only going to describe SMAL, but the Interested readers are referred to [25] for the operational semantics of actor language or mobile actor language.
The primitives of actor communication are:

- `new(b)` creates a new actor and returns its address. It is also possible to create multiple actors with this schema. The parent actor knows the name of the newly created actors, therefore it can send messages to each of them with each others names.

- `send(`v_0`,`v_1`)` creates a new message and puts it in the message delivery system, where v_1 is the content and v_0 the destination actor.
- `ready(b')` ends the current execution and makes the actor ready to receive a new message using behavior `b'`.

The primitives for mobility are:

- `newloc(Y)` indicates the appearance or creation of a new location, with an initial resource map denoted by `Y`.
- `migrate(l')` moves an actor from its current location to one denoted by `l'`.
- `attach(v)` saves a resource denoted by `v` into the actor's resource map.
- `detach(v)` removes a resource denoted by `v` from the actor's resource map.
- `register(`v_0`,`v_1`,l)` adds the mapping of a resource name v_1 to an actor v_0 in a location `l`.
- `deregister(`v_0`,`v_1`,l)` removes the mapping of a resource name v_1 from an actor v_0 in a location `l`.

The primitives for security are:

- `allow(v)` changes the actor's ACL to include actor `v`.
- `allowloc(v)` changes the actor location's ACL to include actor `v`.
- `disallow(v)` changes the actor's ACL to exclude actor `v`.
- `disallowloc(v)` changes the actor location's ACL to exclude actor `v`.

If there are no restrictions on messaging or migration of an actor, then as an argument `allow` or `allowloc` primitives receive a null ACL, which is represented by $\perp$.

2.2.3 Actor Configuration

Assuming that two sets AT (atoms) and X (variables) are given and V, E and M are defined as follows:

Definition 1: **V E M**

The set of values **V**, the set expressions **E** and the set of messages **M** are defined inductively as follows:

$\mathbf{V} = \mathbf{At} \cup \mathbf{X} \cup \lambda\mathbf{X.E} \cup pr(\mathbf{V,V})$

$\mathbf{E} = \mathbf{V} \cup app(\mathbf{E,E}) \cup \mathbf{F}_n(\mathbf{E}^n)$ where $\mathbf{F}_n(\mathbf{E}^n)$ is all arity-n primitives.

$\mathbf{M} = \langle \mathbf{V} \Leftarrow \mathbf{V} \rangle_X$

Actor names are denoted with variables. At any given point an actor can be either ready to receive a message, `ready(e)`, or currently busy with execution of some expression `e`. And $< v_0 \Leftarrow v_1 >$ denotes a message v_0 sent to an actor v_1.

The set **X** of variables represent both actors and locations, where the set **L**, with $\mathbf{L} \subseteq \mathbf{X}$, is the set of locations, and the set **R**, with $\mathbf{R} \subseteq \mathbf{X}$, is the set of resource identities. The structure of **M** allows selection of valid senders via ACLs, where list of ACLs is defined as: $\mathbf{ACL} \in \mathbf{P}_\omega[Dom(\alpha) \cup \{\perp\}]$. α is defined as: $\alpha \in \mathbf{X} \xrightarrow{f} (\mathbf{E} \times \mathbf{L} \times (\mathbf{R} \xrightarrow{f} \mathbf{X}) \times \mathbf{ACL})$. According to that $\alpha(a) = e \times l \times r \times c$ implies that actor a has behavior e and is currently executed at location l with resource map r and ACL c.

A universal actor configuration is a global snapshot of a group of actors. It consists of an actor map, α, multi-set of messages in transit, μ, a set of mappings from locations to resource maps, π ($\pi \in \mathbf{L} \xrightarrow{f} (\mathbf{R} \xrightarrow{f} \mathbf{X})$), a set of receptionists, ρ, and a set of external actors, χ, and written as: $\langle \alpha \mid \mu \mid \pi \rangle^{\rho}_{\chi}$, where ρ, $\chi \in \mathbf{P}_\omega[\mathbf{X}]$, $\alpha \in \mathbf{X} \xrightarrow{f} (\mathbf{E} \times \mathbf{L} \times (\mathbf{R} \xrightarrow{f} \mathbf{X}) \times \mathbf{ACL}$, $\mu \in \mathbf{M}_\omega[\mathbf{M}]$, $\pi \in \mathbf{L} \xrightarrow{f} ((\mathbf{R} \xrightarrow{f} \mathbf{X}) \times \mathbf{ACL})$, and let $\mathbf{A} = Dom(\alpha)$. Let $\mathbf{P}_\omega[\mathbf{X}]$ be the set of finite subsets of **X**, $\mathbf{M}_\omega[\mathbf{M}]$ be the set of multi-sets with elements in **M**, $\mathbf{X}_0 f \xrightarrow{f} \mathbf{X}_1$ be the set of finite maps from $\mathbf{X}_0$ to $\mathbf{X}_1$, Dom(f) be the domain of f, and **FV**(e) be the set of free variables in e. Furthermore, $\downarrow_i$ indicates that the projection of a cross product onto its i^{th} coordinate. This rule denotes that actor names and locations are disjoint. According to these, the actor configurations in SMAL are:

1. $\rho \subseteq A$, and $A \cap \chi = \emptyset$.
2. if $a \in A$, then $FV(\alpha(a)) \subseteq A \cup \chi$, and if $< v_0 \Leftarrow v_1 > \in \mu$ then $FV(v_i) \subseteq A \cup \chi$ for $i < 2$,
3. $Range(\alpha) \downarrow_i \cap A = \emptyset$.

2.2.4 Operational Semantics

In this subsection we are going to give the operational semantics that Toll and Varela [25] have developed for SMAL. The notation $R[e]$ represents a redex `e` in a reduction context R.

`fun` a

$$e \xrightarrow{\lambda}_{Dom(\alpha)\cup\{a\}} e' \Rightarrow \langle \alpha\{[e,\ l,\ r,\ c]_a\} \mid \mu \mid \pi\rangle^{\rho}_{\chi} \ \mapsto\ \langle \alpha\{[e',\ l,\ r,\ c]_a\} \mid \mu \mid \pi\rangle^{\rho}_{\chi}$$

`new` a, a' (`$a'$ means fresh $a$`)

$$\langle \alpha\{[R[new(e)],\ l,\ r,\ c]_a\} \mid \mu \mid \pi\rangle^{\rho}_{\chi} \mapsto \langle \alpha\{[R[a'],\ l,\ r,\ c]_a, [e,\ l,\ r, \{a,\ a'\}]_{a'}\} \mid \mu \mid \pi\rangle^{\rho}_{\chi}$$

`send` a, v_0, v_1

$\langle \alpha\{[R[send(v_0,\ v_1)],\ l,\ r,\ c]_a\} \mid \mu \mid \pi\rangle^{\rho}_{\chi} \ \mapsto$

$\langle \alpha\{[R[nil],\ l,\ r,\ c]_a\} \mid \mu \uplus < v_2 \Leftarrow v_1 >_a \ \mid \pi\rangle^{\rho}_{\chi}$

if $r(v_0) = v_2$

$\langle \alpha\{[R[nil],\ l,\ r,\ c]_a\} \mid \mu \uplus < v_3 \Leftarrow v_1 >_a \ \mid \pi\rangle^{\rho}_{\chi}$

if $\pi(l)(v_0) = v_3$

$\langle \alpha\{[R[nil],\ l,\ r,\ c]_a\} \mid \mu \uplus < v_0 \Leftarrow v_1 >_a \ \mid \pi\rangle^{\rho}_{\chi}$

if $v_0 \notin Dom(()r)$ and $v_0 \notin Dom(()\pi)$

`receive` v_0, v_1

$\langle\ \alpha\{[R[ready(v)],\ l,\ r,\ c]_{v_0}\} \mid\ < v_0 \Leftarrow v_1 >_a \uplus \mu \mid \pi\rangle^{\rho}_{\chi} \ \mapsto$

$\langle\ \alpha\{[R[app(v,\ v_1)],\ l,\ r,\ c]_{v_0}\} \mid \mu \mid l\rangle^{\rho}_{\chi}$

if $a \in c$ or $c = \perp$

`out` v_0, v_1

$\langle\ \alpha \mid \mu \uplus < a \Leftarrow v_0 >_a \ \mid \pi\rangle^{\rho}_{\chi} \ \mapsto\ \langle\ \alpha \mid \mu \mid \pi\rangle^{\rho'}_{\chi}$

if $a \in \chi$, and $\rho' = \rho \cup (FV(v_0) \cap Dom(\alpha))$

`in` v_0, v_1

$\langle\ \alpha \mid \mu \mid \pi\rangle^{\rho}_{\chi} \ \mapsto\ \langle\ \alpha \mid \mu \uplus < a \Leftarrow v_0 >_{a'}\ \mid \pi\rangle^{\rho}_{\chi \cup (FV(v_0) - Dom(\alpha))}$

if $a \in \rho$, then $FV(v_0) \cap Dom(\alpha) \subseteq \rho$

`migrate` l'

$\langle\alpha\{[R[migrate(l')],\ l,\ r,\ c]_a\} \mid \mu \mid \pi \uplus [c',\ r']_l\rangle^{\rho}_{\chi} \ \mapsto$

$\langle\alpha\{[R[nil],\ l',\ r,\ c]_a \mid \mu \mid \pi \uplus [c',\ r']_{l'}\rangle^{\rho}_{\chi}$

if $a \in c'$ or $c' = \perp$

`newloc` Y `(`a' `means fresh` a`)`

$\langle\alpha\{[R[newloc(Y)],\ l,\ r,\ c]_a\} \mid \mu \mid \pi\rangle^{\rho}_{\chi} \ \mapsto$

$\langle\alpha\{[R[l'],\ l,\ r,\ c]_a \mid \mu \mid \pi \uplus [\{a\},\ Y]_{l'}\rangle^{\rho}_{\chi}$

`attach` a'

$\langle\alpha\{[R[attach(a')],\ l,\ r,\ c]_a\} \mid \mu \mid \pi\rangle^{\rho}_{\chi} \ \mapsto$

$\langle\alpha\{[R[nil],\ l,\ r \cup (a' \rightarrow a''),\ c]_a \mid \mu \mid \pi \uplus [c',\ r' \cup (a' \rightarrow a'')]_{l'}\rangle^{\rho}_{\chi}$

`detach` a'

$\langle\alpha\{[R[detach(a')],\ l,\ r \cup (a' \rightarrow a''),\ c]_a\} \mid \mu \mid \pi\rangle^{\rho}_{\chi} \ \mapsto$

$\langle\alpha\{[R[nil],\ l,\ r,\ c]_a \mid \mu \mid \pi\ \rangle^{\rho}_{\chi}$

`register` a', a'', l'

$\langle\alpha\{[R[register(a',\ a'',\ l')],\ l,\ r]_a\} \mid \mu \mid \pi \uplus [c',\ r']_{l'}\rangle^{\rho}_{\chi} \ \mapsto$

$\langle\alpha\{[R[nil],\ l,\ r \cup (a' \rightarrow a'')]_a\} \mid \mu \mid \pi \uplus [c',\ r' \cup (a' \rightarrow a'')]_{l'}\rangle^{\rho}_{\chi}$

if $(a' \rightarrow a'') \in r$

`unregister` a', a'', l'

$\langle\alpha\{[R[unregister(a',\ a'',\ l')],\ l,\ r]_a\} \mid \mu \mid \pi \uplus [c',\ r' \cup (a' \rightarrow a'')]_{l'}\rangle^{\rho}_{\chi} \ \mapsto$

$\langle\alpha\{[R[nil],\ l,\ r]_a\} \mid \mu \mid \pi\ \rangle^{\rho}_{\chi}$

if $(a' \rightarrow a'') \in r$

`allow` a'

$\langle\alpha\{[R[allow(a')],\ l,\ r,\ c]_a\} \mid \mu \mid \pi\rangle^{\rho}_{\chi} \ \mapsto$

$\langle\alpha\{[R[nil],\ l,\ r,\ c \cup \{a'\}]_a \mid \mu \mid \pi\ \rangle^{\rho}_{\chi}$

`allow` $\perp$

$$\langle\alpha\{[R[allow(nil)],\ l,\ r,\ c]_a\} \mid \mu \mid \pi\rangle^\rho_\chi \ \mapsto$$
$$\langle\alpha\{[R[nil],\ l,\ r, \perp]_a\} \mid \mu \mid \pi\ \rangle^\rho_\chi$$

`allowloc` a'

$$\langle\alpha\{[R[allowloc(a')],\ l,\ r,\ c]_a\} \mid \mu \mid \pi \uplus [c',\ r']_l\rangle^\rho_\chi \ \mapsto$$
$$\langle\alpha\{[R[nil],\ l,\ r,\ c]_a\} \mid \mu \mid \pi \uplus [c' \cup \{a'\},\ r']_l\ \rangle^\rho_\chi$$

`allowloc` $\perp$

$$\langle\alpha\{[R[allowloc(nil)],\ l,\ r,\ c]_a\} \mid \mu \mid \pi \uplus [c',\ r']_l\rangle^\rho_\chi \ \mapsto$$
$$\langle\alpha\{[R[nil],\ l,\ r,\ c]_a \mid \mu \mid \pi \uplus [\perp,\ r']_l\ \rangle^\rho_\chi$$

`disallow` a'

$$\langle\alpha\{[R[disallow(a')],\ l,\ r,\ c \cup \{a'\}]_a\} \mid \mu \mid \pi\rangle^\rho_\chi \ \mapsto$$
$$\langle\alpha\{[R[nil],\ l,\ r,\ c]_a\} \mid \mu \mid \pi\ \rangle^\rho_\chi$$

`disallow` $\perp$

$$\langle\alpha\{[R[disallow(nil)],\ l,\ r,\ c]_a\} \mid \mu \mid \pi\rangle^\rho_\chi \ \mapsto$$
$$\langle\alpha\{[R[nil],\ l,\ r, \emptyset]_a\} \mid \mu \mid \pi\ \rangle^\rho_\chi$$

`disallowloc` a'

$$\langle\alpha\{[R[disallowloc(a')],\ l,\ r,\ c]_a\} \mid \mu \mid \pi \uplus [c' \cup \{a'\},\ r']_l\rangle^\rho_\chi \ \mapsto$$
$$\langle\alpha\{[R[nil],\ l,\ r,\ c]_a\} \mid \mu \mid \pi \uplus [c',\ r']_l\ \rangle^\rho_\chi$$

`disallowloc`$\perp$

$$\langle\alpha\{[R[disallowloc(nil)],\ l,\ r,\ c]_a\} \mid \mu \mid \pi \uplus [c', r']_l\rangle^\rho_\chi \ \mapsto$$
$$\langle\alpha\{[R[nil],\ l,\ r,\ c]_a\} \mid \mu \mid \pi \uplus [\emptyset,\ r']_l\ \rangle^\rho_\chi$$

2.3 Transactors

Another model that concerns actors, as loosely-coupled distributed components running in an unreliable environment such as internet into systems that reliably maintain globally consistent distributed state, particularly concentrated on fault-tolerance problem is the Transactors Model evaluated by Field and Varela [12].

This model allows the elements of traditional transaction processing to be composed modularly. Transactors Model extends the lambda-calculus based on the actor model and formalizes it with the τ-calculus.

2.3.1 Model Properties

Distributed state denotes that the states of distributed components in a network-connected system are interdependent on one another. The transactor-model maintains these interrelated states in a wide-area network in a consistent way by exposing key semantic concepts related to distributed state in a common language.

Transactors extend the actor model by explicitly modeling node failures, network failures, persistent storage, and state immutability. Just like actors, a transactor encapsulates state and communicates with other transactors via asynchronous message passing. Besides these, in response to a message they may create new transactors, send messages to other transactors, migrate to a new location or modify their internal state. In addition to these inherited actor operations, a transactor may stabilize, checkpoint, or rollback.

Stabilization is a transactors commitment not to modify its internal state until a subsequent checkpoint is performed or until another peer actor causes it to rollback due to semantic inconsistencies. After a transactor enters a stable state it can still process messages, it simply cannot change its own state. Different from stabilization, a *checkpoint* is a consistency guarantee. It creates a copy of the transactor's current state, which can be recovered in the event of temporary failures. In order to checkpoint only the globally consistent states, dependence information is carried along with messages. Checkpoint can be thought of as the second phase of a two-phase commitment protocol, where stabilization is the first phase. A *rollback* brings a transactor back to its previously check-pointed state or makes it disappear, if there is not any such state. Node failures are modeled as rollbacks.

The operational semantics for the τ-calculus helps to prove important soundness and liveness properties by showing that globally consistent checkpoints have equivalent execution traces without any node failures or application-level failures. It furthermore shows the possibility to reach globally consistent checkpoints provided there is some bounded failure-free interval during which check pointing can occur.

Under certain reasonable preconditions, check pointing in the transactor model is possible. *Soundness* under these conditions refers that a trace containing node failures and inconsistencies is equivalent to a normal trace (i.e., one containing no node failure) but with possible message losses. In order to show this, Field and Varela prove that arbitrary τ-calculus traces can be simulated by traces containing only the node failure free subset of the τ-calculus. This shows also how global reasoning about state inconsistencies can be reduced to local reasoning about the possibility of message loss.

Liveness, as the other τ-calculus property, refers for example to the fact that using the transactor model operational semantics global checkpoints can be reached. But not all transactor programs can reach global checkpoints. Therefore Field and Varela introduce a *Universal Check Pointing Protocol*, which assumes a set of preconditions that will entail global check pointing for a set of transactors, and under those preconditions if the protocol terminates then a global check pointing is reached.

2.3.2 Operational Semantics

Field an Varela define the notational preliminaries of transactors that are not standard as follows: Sets are defined using context-free **grammars**, and the set of all terms derived from that non-terminal are represented by a non-terminal of the grammar. Given a set of S, $[S]$ denotes the set of **lists** defined over S, where $[]$ denotes an empty set, $s :: l_s$ a list cell, *len(l)* the length of l and *lastn(n,l)* the list consisting of he last n elements of l. Given a set S_1 and S_2, $S_1 \xrightarrow{f} S_2$ denotes the set of **finite partial maps** from S_1 to S_2, where *dom(m)* denotes the domain of m and and *ran(m)* the range of m, $\emptyset$ the empty map, $m(x)$ element to which m maps, $m[x \mapsto e]$ the map that is the same as m except that x is maped to e.If S is a set then S denotes the set of **multisets** and $\biguplus$ symbolizes the **multiset union**.

Each transition rule of operational semantics refer to a particular redex term within the lambda term encoding a transactor's behavior. To distinguish the redex on which the transition rule will operate the notion of **reduction contexts** of the form $\mathcal{R}(\square)$ is used. Each reduction context is a special term with a single "hole" element $\square$, which is defined such that a transactor behavior can be uniquely decomposed into exactly one redex and one reduction context.

2.3.2.1 Tau-Calculus

The extensional constructs of the τ-calculus can be divided into two categories: those that encode the traditional actor semantics with explicit state management and those that support distributed state maintenance.
Traditional actor constructs:

- `trans` $\texttt{e}_1$ `init` $\texttt{e}_2$ `snart` creates a new transactor with behavior $\texttt{e}_1$, and initiate state $\texttt{e}_2$
- `send v to t` sends a message with content `v` to the transactor `t`
- `ready` a transactor waiting to process the next incoming message
- `self` yield the transactor's own name
- `setstate(v)` updates a transactor's state to the value `v`
- `getstate` retrieves the value of the state

Distributed state maintenance constructs:

- `stabilize` current transactor becomes stable/immobile
- `checkpoint` creates checkpoint if stable and consistent
- `dependent?` checks whether the transactor is dependent on other transactors
- `rollback` reverts the transactor to its previous checkpoint

2.3.2.2 Transactor Configuration

The semantic domains that the τ-calculus operational semantics manipulate are:

$$\begin{array}{llll}
\mathcal{W} & ::= & \mathbf{V}(\mathcal{N})|\mathbf{S}(\mathcal{N}) & \text{Volatility value} \\
\mathcal{H} & ::= & \langle \mathcal{W}, [\mathcal{N}] \rangle & \text{Transactor history} \\
\Delta & = & \mathcal{T} \xrightarrow{f} \mathcal{H} & \text{Dependence Map} \\
\mathcal{S} & ::= & \langle \mathcal{V}, \mathcal{V}; \mathcal{E}, \mathcal{V}; \triangle, \triangle, \triangle \rangle & \text{Transactor} \\
\mathcal{M} & ::= & \mathcal{T} \Leftarrow \langle \mathcal{V}, \triangle \rangle & \text{Message} \\
\Theta & = & \mathcal{T} \xrightarrow{f} \mathcal{S} & \text{Name services} \\
\mathcal{K} & ::= & \{\{\mathcal{M}\}\}|\Theta & \text{Transactor configuration}
\end{array}$$

Volatility value encodes that a transactor is either **volatile** ($w = \mathbf{V}(n), n \geq 0$) or **stable** ($w = \mathbf{S}(n), n \geq 0$). In the event of τ's failure the value that it caries gets lost. The value of n will be referred to as an **incarnation**.

History encodes the checkpoint history of a transactor. A history $h = \langle w, l_h \rangle$ shows that the transactor h has volatility value w, and has check pointed $len(l_h)$ times since it's creation. The history operations that the τ-calculus semantics define are:

- When a transactor is created it's history is set to $\langle \mathbf{V}(0), [] \rangle$.
- When a transactor with history $\langle \mathbf{V}(n), l_h \rangle$ rolls back, its history becomes $\langle \mathbf{V}(n+1), l_h \rangle$.
- If a transactor with history $\langle \mathbf{V}(n), l_h \rangle$ stabilizes, its history becomes $\langle \mathbf{S}(n), l_h \rangle$.
- If a transactor with history $\langle \mathbf{S}(n), l_h \rangle$ checkpoints, its history becomes $\langle \mathbf{V}(0), n :: l_h \rangle$.

Dependence maps map each transactor name t to a history value h. Dependency maps are associated with three explicit semantic components of a transactor:

- **Creation dependency map**: Transactors on which t is dependent for it's existence
- **State dependency map**: Transactors on which t's current state depends
- **Behavior dependency map**: Transactors on which the value of the current redex depends

A **transactor** is a 7-tuple ($\tau \in \mathcal{S} = \langle b, s_{\surd}; e, s; \delta_s, \delta_c, \delta_b \rangle$) containing the following components:

- **Behavior component** (b) encodes the response to the incoming messages.
- **Persistent state component** ($s_{\surd}$) encodes the last value of s stored by a checkpoint.
- **Evaluation state** (e) encodes the expression representing the current state of the evaluation of a transactor's behavior.
- **Volatility value component** (s) encodes the volatile value and contains τ's current state.
- **State dependence map component** (δ_s) encodes the fact that the state of τ depends on the state of the transactors in $dom(\delta_s)$, whose histories are encoded in the map.
- **Creation dependence map component** (δ_c) encodes the fact that the state of τ depends on the state of the parent transactor that has created it.
- **Behavior dependence map component** (δ_b) encodes the behavior dependencies of the transactor τ.

Message ($m \in \mathcal{M}$) contains the name of the target transactor, which is it's destination, the payload, and the dependancy map, which encodes the transitive closure of transactors on which the payload depends.

Name Services maps each transactor name t to a transactor s.

Transactor configuration ($k \in \mathcal{K}$) encodes a network, which is a multiset of messages, and the mapping of the transactor name to transactors.

2.3.2.3 History Operations

In this subsection we are going to describe the basic history operations and the relations on histories with more detail.

Let $h = \langle w, l_h \rangle$ be a history. If $w = \mathbf{S}(n)$ a history h is **stable** and is denoted as $\diamond(h)$, otherwise h is **volatile**. If l_h is nonempty, then h is furthermore **persistent** ($\surd(h)$), otherwise it is **ephemeral**.

The transitions that a history associated with a single transactor are:

$\looparrowright$: encodes that a transactor has rolled back

$\rightarrow_{\diamond}$: encodes that a transactor has stabilized

$\rightarrow_{\surd}$: encodes that a transactor has checkpointed

$\leadsto_{*}$: encodes a partial order of history

$\ltimes$: encodes that a transactor is superseded by on other transactor

The above conditions satisfy the following simple and complex relations:

$$\begin{array}{lcl}
\langle \mathbf{V}(n), l_h \rangle & \looparrowright & \langle \mathbf{V}(n+1), l_h \rangle \\
\langle \mathbf{S}(n), l_h \rangle & \looparrowright & \langle \mathbf{V}(n+1), l_h \rangle \\
\langle \mathbf{V}(n), l_h \rangle & \rightarrow_{\diamond} & \langle \mathbf{S}(n), l_h \rangle \\
\langle \mathbf{S}(n), l_h \rangle & \rightarrow_{\surd} & \langle \mathbf{V}(0), n :: l_h \rangle \\
\leadsto & \triangleq & (\looparrowright \cup \rightarrow_{\diamond} \cup \rightarrow_{\surd}) \\
\ltimes & \triangleq & \looparrowright \cdot \leadsto_{*}
\end{array}$$

Two histories h_1 and h_2 are **comparable** if they are valid histories for the same transactor and if one of the following relations exist between them: $h_1 \leadsto_{*} h_2$ or $h_2 \leadsto_{*} h_1$. Furthermore, two histories are **consistent** if neither supersede the other. The state represented by h_2 supersedes the absolute state represented by h_1 ($h_1 \ltimes h_2$), if h_2 is a transactor history that rolled back from the state represented by history h_1. Given consistent histories h_1 and h_2, the **sharpening operation** is defined as follows:

$$h_1 \sharp h_2 = \begin{cases} h', & \text{if there exists } h' \text{ such that } h_1 \rightarrow_{\diamond} h' \leadsto_{*} h_2 \\ h_1, & \text{otherwise.} \end{cases}$$

2.3.2.4 Dependency Map Operations

Let δ_1 and δ_2 be dependency maps, then $\delta_1 \ltimes \delta_2$ denotes that δ_1 is **invalidated** by $\delta_2 \leftrightarrow \exists\, t | t \in dom(\delta_1) \cap dom(\delta_2)$ and $\delta_1(t) \ltimes \delta_2(t)$. The union of δ_1 and $\delta 2$ ($\delta_1 \bigoplus \delta_2$) is defined as follows:

$$(\delta_1 \bigoplus \delta_2)(t) = \begin{cases} max \rightsquigarrow_* (\delta_1(t), \delta_2(t)), & \text{when } t \in dom(\delta_1) \cap dom(\delta_2) \text{ and} \\ & \delta_1(t) \text{ and } \delta_2(t) \text{ are compairable,} \\ \delta_1(t), & \text{when } t \in dom(\delta_1), t \notin dom(\delta_1), \\ \delta_2(t), & \text{when } t \notin dom(\delta_1), t \in dom(\delta_1), \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

The expended version of the sharpening operation of the histories with consistent dependence maps is as follows:

$$(\delta_1 \sharp \delta_2)(t) = \begin{cases} \delta_1(t) \sharp \delta_2(t), & \text{when } t \in dom(\delta_1) \cap dom(\delta_2) \\ \delta_1(t), & \text{otherwise.} \end{cases}$$

A dependency map δ is **independent** ($\diamond\delta$), if $\forall\, t \in dom(\delta)$, $\diamond(\delta(t))$.

2.3.2.5 Transactor Configuration Transition Rules

In this subsection we are going to give some of the transactor configuration transition rules defined by Field and Varela, which are going to be adopted to AcTrust model.

The pure reduction rules for lambda terms encoding transactor behaviors are as follows:

$$\begin{array}{lrcl}
[pur1] & ((\lambda x,\ e)\ v) & \rightarrow_\lambda & e[v/x] \\
[pur2] & \mathbf{fst}(\langle v_1,\ -\rangle) & \rightarrow_\lambda & v_1 \\
[pur3] & \mathbf{snd}(\langle -,\ v_2\rangle) & \rightarrow_\lambda & v_2 \\
[pur4] & \mathbf{if}\ \text{true}\ \mathbf{then}\ e_1\ \mathbf{else}\ -\ \mathbf{fi} & \rightarrow_\lambda & e_1 \\
[pur5] & \mathbf{if}\ \text{false}\ \mathbf{then}\ -\ \mathbf{else}\ e_2\ \mathbf{fi} & \rightarrow_\lambda & e_2 \\
[pur6] & \mathbf{letrec}\ x = v\ \mathbf{in}\ e\ \mathbf{ni} & \rightarrow_\lambda & e[(v[(\mathbf{letrec}\ x = v\ \mathbf{in}\ e\ \mathbf{ni})/x])/x] \\
[pur7] & f(v_1, ..., v_n) & \rightarrow_\lambda & (f \in F, v = [\![f]\!](v_1, ..., v_n))
\end{array}$$

In the transition rules for basic transactor semantics the relation $\xrightarrow[\text{l}]{\text{t}}$ is a single step transition relation, which will be annotated with both the name l the applicable rule and a distinguished transactor name t to which the relation will be said to apply. The rules that are listed below augment the basic actor transitions - pure, new, send, receive, get, set, self - with additional rules for managing distributed state. Here we are going to give the transition rules and avoid a detailed description of these rules, interested readers may refer to [12].

`pure` Evaluation pure redex.

$$\frac{e \rightarrow_\lambda e' \quad e \in \mathcal{E}_P^{rdx}}{\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[e], s;\ \delta_s, \delta_c, \delta_b \rangle] \xrightarrow[\text{[pure]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[e'], s;\ \delta_s, \delta_c, \delta_b \rangle]}$$

`new` Creation of a new transactor. $(t' \notin dom(\theta) \cup \{t\}$ and $\delta' = [t' \mapsto \mathbf{H}_0])$

$$\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{trans } b' \text{ init } s' \text{ snart}],\ s;\ \delta_s[t \ \mapsto\ h],\ \delta_c,\ \delta_b \rangle] \xrightarrow[\text{[new]}]{\text{t}}$$
$$\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[t'], s;\ \delta_s[t \ \mapsto\ h],\ \delta_c,\ \delta_b \oplus \delta' \rangle][t' \mapsto \langle b', \texttt{nil}; \text{ready}, s';\ \delta',\ \delta_c \oplus \delta_b \oplus [t \ \mapsto\ h], \emptyset \rangle]$$

`send` Send message.

$$\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{send } v_m \text{ to } t'], s;\ \delta_s[t \ \mapsto\ h], \delta_c, \delta_b \rangle] \xrightarrow[\text{[send]}]{\text{t}}$$
$$(\mu \uplus \{t' \Leftarrow \langle v_m, \delta_c \oplus \delta_b \oplus [t \mapsto h] \rangle\}) \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\texttt{nil}], s;\ \delta_s[t \ \mapsto\ h], \delta_c, \delta_b \rangle]$$

`rcv1` Message dependencies not invalidated by transactor and transactor dependencies not invalidated by message, so the message will be normally processed. $(\delta_{sc} = \delta_s \oplus \delta_c)$

$$\frac{\urcorner(\delta_{sc} \ltimes \delta_m) \qquad \urcorner(\delta_m \ltimes \delta_{sc})}{(\mu \uplus \{t' \Leftarrow \langle v_m, \delta_m \rangle\}) \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{ready}], s;\ \delta_s, \delta_c, - \rangle] \xrightarrow[\text{[rcv1]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd}; (b\ \ v_m), s;\ \delta_s \# \delta_m, \delta_c \# \delta_m, \delta_m}$$

`get` Retrieve state.

$$\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{getstate}], s;\ \delta_s, \delta_c, \delta_b \rangle] \xrightarrow[\text{[get]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[s], s;\ \delta_s, \delta_c, \delta_b \oplus \delta_s \rangle]$$

`self` Yields references to own name.

$$\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{self}], s;\ \delta_s, \delta_c, \delta_b\rangle] \xrightarrow[\text{[self]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[t], s;\ \delta_s, \delta_c, \delta_b\rangle]$$

`set1` Transactor is volatile, so that sating state succeeds.

$$\frac{\neg\diamond(h)}{\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{setstate}(s)], -;\ \delta_s[t \mapsto h], \delta_c, \delta_b\rangle] \xrightarrow[\text{[set1]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\texttt{true}], s;\ \delta_s[t \mapsto h] \oplus \delta_b, \delta_c, \delta_b\rangle]}$$

`set2` Transactor is stable, so that attempt to set state fails.

$$\frac{\diamond(h)}{\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{setstate(-)}], s;\ \delta_s[t \mapsto h], \delta_c, \delta_b\rangle] \xrightarrow[\text{[set2]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\texttt{false}], s;\ \delta_s[t \mapsto h], \delta_c, \delta_b\rangle]}$$

`sta1` Transactor is volatile, so that stabilization causes it to become stable.

$$\frac{h \ \rightarrow_{\diamond}\ (h')}{\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{stabilize}], s;\ \delta_s[t \mapsto h], \delta_c, \delta_b\rangle] \xrightarrow[\text{[sta1]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\texttt{nil}], s;\ \delta_s[t \mapsto h'], \delta_c, \delta_b\rangle]}$$

`sta2` Transactor is currently stable, so that stabilization is a no-op.

$$\frac{\diamond(h)}{\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{stabilize}], s;\ \delta_s[t \mapsto h], \delta_c, \delta_b\rangle] \xrightarrow[\text{[sta2]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\texttt{nil}], s;\ \delta_s[t \mapsto h], \delta_c, \delta_b\rangle]}$$

`chk1` Transactor is stable and independent, so that checkpoint succeeds.

$$\frac{\diamond(\delta_s[t\ \mathit{mapsto}\ h] \oplus \delta_c h) \qquad h \rightarrow_{\surd} h'}{\mu \mid \theta[t \mapsto \langle b, -;\ \mathcal{R}[\text{checkpoint}], s;\ \delta_s[t \mapsto h], \delta_c, -\rangle] \xrightarrow[\text{[chk1]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s;\ \text{ready}, s;\ [t \mapsto h'], \emptyset, \emptyset\rangle]}$$

`chk2` Transactor is dependent or volatile, so that checkpoint behaves simply like ready.

$$\frac{\neg\diamond(\delta_s \oplus \delta_c h)}{\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{checkpoint}], s;\ \delta_s, \delta_c, -\rangle] \xrightarrow[\text{[chk2]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \text{ready}, s; \delta_s, \delta_c, \emptyset\rangle]}$$

`rol1` Transactor is stable, so that rollback behaves simply like ready.

$$\frac{\diamond(h)}{\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{rollback}], s;\ \delta_s[t \mapsto h], \delta_c, -\rangle] \xrightarrow[\text{[rol1]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \text{ready}, s;\ \delta_s[t \mapsto h], \delta_c, \emptyset\rangle]}$$

`dep1` Transactor is independent, so that it yields false.

$$\frac{\diamond((\delta_s \oplus \delta_c) \setminus t)}{\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{dependent?}], s;\ \delta_s, \delta_c, \delta_b\rangle] \xrightarrow[\text{[dep1]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{false}], s; \delta_s, \delta_c, \delta_b\rangle]}$$

`dep2` Transactor is dependent, so that it yields true.

$$\frac{\neg\diamond((\delta_s \oplus \delta_c) \setminus t)}{\mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{dependent?}], s;\ \delta_s, \delta_c, \delta_b\rangle] \xrightarrow[\text{[dep2]}]{\text{t}} \mu \mid \theta[t \mapsto \langle b, s_{\surd};\ \mathcal{R}[\text{true}], s; \delta_s, \delta_c, \delta_b\rangle]}$$

`lose` Message loss.

$$(\mu \uplus \{m\}) \mid \theta \xrightarrow[\text{[lose]}]{\text{t}} \mu \mid \theta$$

2.4 Related Work

Besides the semantics of the actor and transactors models that are related with the semantic goal of this thesis, there are some researchers working on reputation based trust, which concerns the technical and mathematical goal of the thesis. The main reason for the high activity of research in this field is it's similarity to the social relations of the real world and applicability to the cyber world. In this section we are going to present the major work done on retrieving feedbacks, calculating trust values and other auxiliary metrics.

The first simple reputation mechanism proposed in the literature is developed by Zacharias et al. and is called **Sporas** [33]. Sporas can be implemented irrespectively of the number of rated interactions. The reputation service that Sporas provides is based on the following principles:

- New users start with a minimum reputation value, which is increased with their activity on the system.
- The reputation value of a user never falls below the reputation of a new user.
- After each transaction, based on the feedback provided from the transaction partner, the reputation value of the user is updated.

- Two users can rate each other only once. If they have interacted more than once, then the most recent feedback is used.
- As the number of ratings a user has increases, the effect of each rating decreases.

Histos [33] is a more complex reputation mechanism, which is also developed by Zacharias et al.. Histos assumes that the system has been somehow bootstrapped so that there is a large quantity of rated interactions to create a dense web of pair wise ratings. This model is applicable to p2p E-commerce communities where peers are equal in their roles and are independent entities, thus no peers can serve as trusted third parties or intermediaries [30].

Besides this research that focused on building trust through trusted third parties or intermediaries Xiong et al. develop **PeerTrust** [31], which is a coherent adaptive trust model for quantifying and comparing the trustworthiness of peers, based on the evaluation of peer in terms of level of reputation it receives in providing service to other peers in the past. For PeerTrust, Xiong et al. introduce transition context factor and community context factor as adaptive factors, as well as three basic trust parameters in computing trustworthiness of peers:

- Feedback a peer receives from other peers
- Total number of transactions a peer performs
- Credibility of the feedback source

Derbas et al. make a comparison of numerous trust models and using this comparison develop **TRUMMAR** [24] as a truly comprehensive trust model that concerns all of the trust metrics used in different models for mobile agent systems based on reputation. The trust system of TRUMMAR takes the following in to account:

- **Hierarchy of trust** is used to classify the peers in a system in different groups depending on the possibility of trustworthiness. Derbas [8] for example describes hierarchy of trust in four levels: self; other hosts on its own network that are under the same administrative control (neighbors); hosts from different networks that are under different, but trusted administrative control (friends); and hosts that are willing to volunteer information (strangers). Different hierarchical levels refer to different levels of trust in the information supplied from peers in different levels.

- **Position of members in community** refers to rating for example a feedback from a trust center more then a feedback from a normal peer.
- **Prior-driven reputation** is a method for a posterior identification of malicious hosts to build a trust policy. Here reputation can for example be driven looking at the position of members in a community before having any transaction with that peer previously.
- **Trust propagation** allows diffusion of trust according to whom the trusted peers trust. For example if a peer X trusts another peer Y, and if this peer Y trusts a third peer Z, then peer X trusts peer Z according to trust propagation.
- **First impression** is used when a new peer wants to join the system and none of the peers have had a transaction with it before. Some first impression methods that can be used in such situations are using an initial test period and using a predefined random trust value.
- **Result of interaction** is the feedback sent to a peer after an interaction referring to the level of satisfaction during the interaction.
- **Fixed decay factor** allows the submitted feedbacks to loose their effect linearly with the time.
- **Dynamic decay factor** allows the submitted feedbacks to loose their effect with time depending on some dynamic factors.
- **Trusting vs. suspicious** is described in [6] as defining either the list of hosts that has to be visited or the list of hosts that the agent does not want to communicate with again.

- **Similarity** between two hosts depends on the similarity of their reputation values. The closer their reputation the closer their evaluation and the more recommendations will be credible to the other.

- **Activity** depends on the total number of interactions with other hosts, which is highly affected by the size of a community. Yu describes this fact in [32] as the difference between the number of reputations an agent is going to get in a society with 3000 agents and 300,000 agents. According to his suggestion, depending on the activity level of an host, it is possible to define how much it's information is reliable and up-to-date.

- **Popularity** is a metric that looks at the number of transactions the peer took part in a given time period. The higher this value is, more popular the peer becomes and gets better accepted among other hosts.

- **Cooperation** is described in [11] as the metric that depends on the number of times this host interacted compared to the number of times it has asked for an interaction. Through cooperation the willingness of a host to provide service to others can be defined.

- **Number of interactions** is a metric that looks at the number of transactions the peer took part in during his life time.

Guha et al. made a comprehensive research on propagation of trust and distrust in [14]. They suggest four possibilities of atomic propagation for trust and distrust. Atomic propagation means that a conclusion is reached based on a single argument. For example if it is given that a peer a trust another peer b, and the goal is to apply the knowledge that peer b trusts c. Atomic trust allows in one propagation step to guess that a trusts c. The atomic propagations Guha et al. describe are:

- **Direct propagation:** Suppose peer a trusts peer b, and peer b trusts peer c, then this implies that peer a trusts peer c.

- **Co-citation:** Suppose peer a_1 trusts peers b_1 and b_2 and peer a_2 trusts b_2. Under co-citation, they consider that a_2 should also trust b_1.

- **Transpose trust:** If peer a trusts peer b then trusting b implies trusting a.
- **Trust coupling:** Peers a and b trust peer c, so trusting a implies trusting b.

2.5 State of the Art

As mentioned in section 2.4 there is an extensive amount of research focused not only on building trust for electronic markets through trusted third parties or intermediaries on p2p E-commerce communities, but also where peers are equal in their roles and are independent entities. In this section we are going to present the problems of such trust systems and suggest methods to overcome these problems.

2.5.1 Quality of feedback

Basic reputation mechanisms like the one suggested by Chen et al. [4] computes a reputation for a rater peer based on quantity of the ratings it gives. However, the method is based on the assumption that the ratings are of good quality if they are consistent to the majority opinions of the rating. Which allows peers that submit fake or misleading feedbacks still to gain a good reputation simply by submitting a large number of feedbacks and becoming the majority opinion.

Dellarocas et al. [7] suggest using the credibility of the feedback source as one of the basic trust parameters when evaluating the trustworthiness of a peer. They furthermore suggest using cluster-filtering techniques to detect and filter out the dissenter feedbacks.

Venkatraman [28], Tajeddine et al. [24] and Lin [18] suggests instead of eliminating the exceptional feedbacks completely out, to use the social clusters in order to leaver the effects of the feedbacks from different clusters. The credibility of the feedback source is defined according to it's contacts to the peers inside the same cluster and outside the cluster. Venkatraman et al.'s suggestion, is based on the assumption that a peer, whose social contacts are in the same cluster is worse than some other peer, who has social contacts in different clusters. Tajeddine on the other hand describes social clusters in four levels: self, neighbors, friends, and strangers; and suggest to trust itself the most, neighbors more than friends and friends more than peers it does not know. Lin et al. present a distributed reputation and trust management framework for web applications and the performance of the system by simulations. Parallel to Tajeddine et al. they define a hierarchy of trust that has direct and institutional trust clusters.

Similar to Tajeddine et al., Pujol [21] calculates the value of the ratings by the topological information, instead of using the feedbacks after interaction between agents. A communitys social network has been used as a measure of each members reputation within a community.

2.5.2 Lack of temporal adaptivity

With time, the quality of service a peer accomplishes may change in a positive or a negative way, but the recent trust management systems usually do not consider this change. Or the ratings from a untrustworthy peer may probably be less honest then the feedback from a trustworthy peer. Therefore a trust system that either to consider only the recent reputations or all of the reputation in the history equally or maybe even with changing affectivity would give more realistic results. Xiong [30] works on this problem and evaluates PeerTrust with a metric that allows giving higher weight to feedbacks from users with better reputations. They also use the transaction value as a metric that affects the trust value in sense of temporal adaptivity.

2.5.3 Lack of rating incentive

Reputation based systems are dependent on feedbacks. However, since the feedback is not a real component of the transaction, but related more with social responsibility, peers either do not want to waste time with something that does not bring a direct benefit or forget sending feedbacks. Another aspect is that the peers that are satisfied with the transaction usually forget submitting feedback, as the ones who had problems do not. As a result, the feedbacks in the system may end up being from the same group of peers or mostly from those who had bad experiences. Xiong et al. [30] suggests adding a reward as a community context for peers, who submit feedbacks.

2.5.4 Storage of feedback

The most common e-marketplaces, like Amazon and eBay, use centralized reputation. Such systems have two main deficits. On one hand they are easier to attack, since vulnerable data is stored in one location. On the other hand they are inflexible, because centralized authority may be subject to the scalability problem. Distributed trust, which allows storing trust information separately for each peer may be a solution to this problem.

However, distributed trust mechanism also has its challenges, like the propagation of reputation, or the evaluation of the trustworthiness of a party with information gathered from potentially untrustworthy parties. As a solution the problem of eliciting honest feedbacks Jurca et al. suggested in [16] some incentive mechanisms through some side payment mechanisms.

2.5.5 Calculating the trust value of new peers

In distributed mobile systems, such as digital agent systems, peers usually do not live very long and besides that frequently new peers are generated. The trust value of a peer in most of the trust systems is based on the feedbacks a peer receives from the interactions in his history, which is not possible for new peers. Therefore in many such systems the starting reputation is set low, but this method causes a barrier for taking part in interactions. Tajeddine et al. suggest in [24] sending nonessential test data with known results during a test period, which differs according to the peer in order to figure out how the new peer is behaving. Another alternative is suggetsed by Falcone et al. in [10], which concerns defining a fixed trust value for each peer according to the its hierarchical cluster.

3. Reputation Based Trust for Secure Actors

The main focus of this thesis is the design and development of AcTrust - a model for calculation of reputation values and the determination of trust decisions. AcTrust extends AMST [2] with using trust value information as a criteria for communication. Trust information is similar to dependence maps in transactors [12], which is described in Subsection 2.3.

Trust maps have a list of the actors, which the actor is allowed to and not allowed to communicate with. After each transaction the trust value of the actors that take part in the interaction is calculated and the trust map is updated with this new information. For this calculation actors use the the feedback about the interaction and some other metrics that overcome different problems of reputation-based trust.

In this chapter, we are furthermore going to introduce a trust policy in order to indicate, constrain and identify the type of trust that is going to be achieved. Moreover, we are going to introduce the Trusted Actor Language and expand the operational semantics of the actor model in order to express how AcTrust behaves under different conditions.

3.1 Model Properties

In this section, we are first going to present the general properties of AcTrust, which allows trustworthy communication in various applications, such as E-Commerce or Grid Computing. Afterwards we are going to define the trust parameters that AcTrust implies in order to achieve a higher level of trustworthiness and we will conclude the section with the general trust policy of AcTrust.

3.1.1 General Properties

In this subsection we are going to describe the feedback collection and storage as well as trust propagation mechanism and how trusted communication is realized.

In AcTrust every actor is associated with a unique name, a trust map and a trust thresh-hold. They furthermore encapsulate a state and a thread of control. The state of an actor can be ready or busy processing a message. The actors communicate with other actors or their surrounding in order to fulfill their duties. The communication between the actors is realized with point-to-point messages that are buffered in an input queue.

Every message is a triple of sender's name, receiver's name and content. The messages that are received by the interface of an actor are interpreted with public methods, which operate on the state of the actor. Before a message is interpreted the trustworthiness of the sender is tested. To do this trust maps are used.

Every entry in the trust map is annotated with an actor's unique name and its trust value. Only the actors that are in the trust map and have a higher trust value then the thresh-hold of that actor are trusted. An actors thresh-hold is defined at its creation, just like its name, and is given by the parent. When an actor X creates another actor Y, the trust map of X is extended with Y and the trust map of Y is inherited from X.

In reaction to a message an actor in AcTrust can change its present state and becoming ready to process the next message in its mail queue, creating a finite set of actors with some initialized behaviors as well as a *threshold*, send messages to peer actors, and perform one of the three following trust operations:

- Enter a new peer to its trust map,
- Updating the trust value of a peer in its trust map, or
- Query the trustworthiness of a peer in it's trust map

3.1.1.1 Collecting the Reputation Information

As described in Section 1.2 there are two different types of trust in the literature: functional trust and referral trust . *Functional trust* is the trust in the quality of service an actor accomplishes. *Referral trust*, on the other hand, is the trust in the actor's competence and credibility to give trusted feedback.

In AcTrust functional trust depends on the behavior of an actor during an interactions and in order to fulfill predefined duties. Functional trustworthiness is realized by fulfilling a specific task up to the expectations of the initiating peer. Every peer reviews the behavior of the peers that have taken part in interactions with him.

Referral trust is questioned only then when a peer X wants to interact with peer Y and X is unknown to Y. In that case Y asks his trusted parties for reputation information of X. Trusted parties are those peers that Y trusts based on his previous interactions. So, peer Y trusts those peers that have interacted trustworthy both in giving reputation and have performed some specific tasks up to the expectations of peer Y. This property of AcTrust motivates the peers to submit reputations when desired. If the peer is not known to any peer in a network, then it's interaction request is rejected.

3.1.1.2 Storing the Trust Information

In reputation based trust models there are two possibilities to find out whether a peer is trustworthy or not: using a central authority or storing the information in a distributed way. In distributed reputation each actor stores the trust information separately. The alternative to that is to have a central trust authority, which calculates the trust values and stores these in a central database. If we consider the fact that peers do not only interact with those peers that they know but also with those peers that they confront for the first time, we will see that it is hard to protect the trust information that they carry with themselves. On the other hand storing all of the vulnerable information in a single place makes that place a center of malicious attacks and a single point of failure. Besides that in a central reputation system the actors will be querying the trust center all the time, which will increase the message overload in the system.

In AcTrust we used a combination of these two possibilities, and achieved a more robust system against certain manipulations of malicious actors. Every peer stores the trust information separately and refers to that information source in the first position. Furthermore, if the information in that source is not sufficient, then it asks the other peers and only considers the feedback from those peers which are trusted. Every actor has its own list of trusted actors, which is updated after each succeeded or failed transaction.

In AcTrust, if a malicious peer manages to manipulate the trust information in one of the peers, since the trust information of an actor is usually not saved only in one information source, the trust system will not be completely manipulated.

3.1.1.3 Trust Propagation

TheAcTrust model allows propagating trust between peers. Guha et al. suggests in [14] four possibilities of horizontal atomic propagation for trust and distrust. From these four we have applied only the direct propagation to AcTrust, because the other three propagations suppose that if peer X trusts peer Y then Y trusts X, too. Accepting a trust propagation like this can strongly harm the level of security in actor based environments. According to these rules, if an untrustworthy peer adds all trustworthy peers in his trusted peers list, which can easily be done by setting his trust threshold to the lowest trust value, then all peers in the system have to trust this untrustworthy peer.

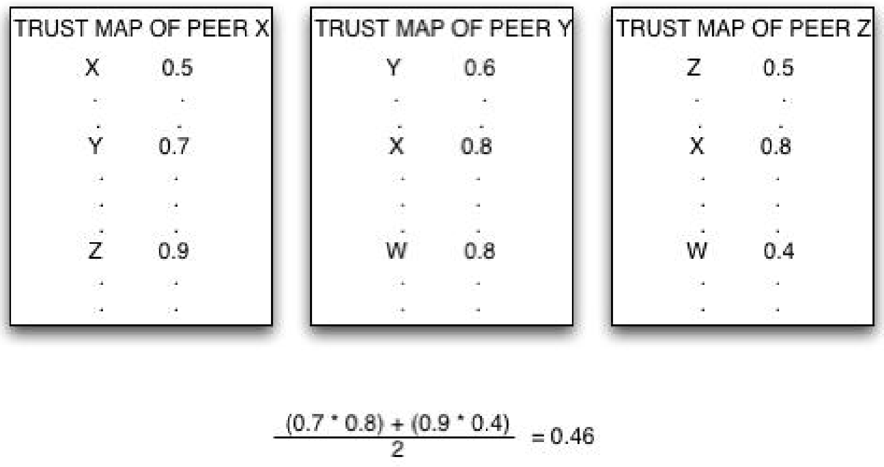

Figure 3.1: Calculation of the trust value of new peer W, for the trust map of X, using the feedback retrieved from peers Y and Z.

Direct propagation extends the set of trusted actors according to the following commutability property: If peer X trust peer Y and if peer Y trust peer Z, then according to direct propagation peer X trusts peer Z, too. If two trusted peers X and Y have conflicting trust information on peer Z, then we computes a weighted average of the feedbacks, based on trust placed on each peer. An example to this calculation is given in the Table3.1.

Through direct propagation of trust in AcTrust, the actors can build social network between other members of the marketplace without raising the message traffic much.

3.1.1.4 Trusted Communication

In this subsection, we are going to show how a peer responds under different conditions to a message from another peer.

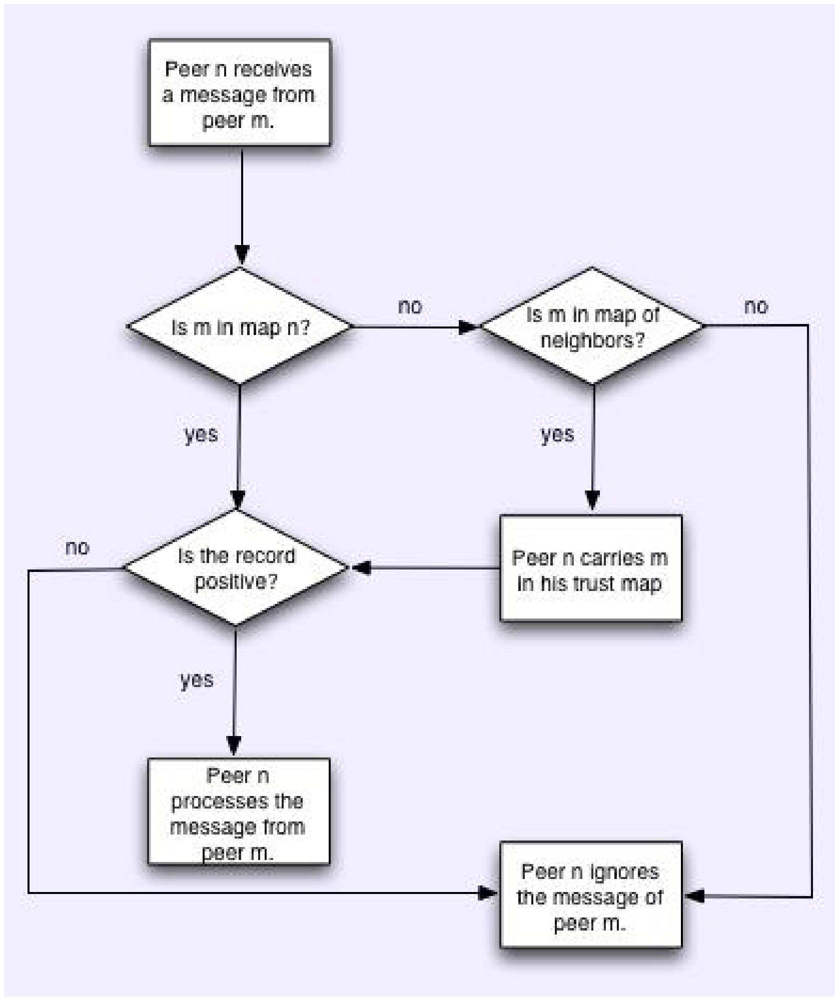
Peer n receives a message from peer m.
Is m in map n?
no
Is m in map of neighbors?
no
yes
yes
no
Is the record positive?
Peer n carries m in his trust map
yes
Peer n processes the message from peer m.
Peer n ignores the message of peer m.

Figure 3.2: Determination of the trust value of a peer n that is willing to communicate with another peer m.

As shown in Figure 3.2, if a peer X receives a message from peer Y, peer X looks first whether Y is in his trust peers list, which is also called *trust map of X*. If Y is in the trust map of X and has a lower trust value then the threshold of X, then Y is *untrustworthy*, and X ignores the message. If Y is in the trust map of X and has a higher trust value then the threshold of X, then Y is *trustworthy* and, X processes the message. If Y is not in the trust map of X, then he multicasts to the trustworthy peers in his trust map asking for reputation information of Y. X calculates a trust value using the replies from his trusted parties and adds the mapping of this trust value and Y to his trust map. If this trust value is higher then the threshold of X, then X processes the message. If none of the trusted peers have Y in their trust maps, then the message is from an actor that is new to the system. In that case X ignores the message.

3.1.2 Calculation of Trust

In this subsection, we are going to define the trust parameters of AcTrust that are used in order to calculate the trust value of a peer.

The trust parameters that AcTrust implies in order to calculate a trust value are as follows:

- Reputation an actor receives from other actors
- Total number of transactions an actor performs
- Temporal adaptivity of a reputation

The basic trust metric on AcTrust is as follows.

$$T(a) = \sum_{i=1}^{I(a)} R(a,i) \tag{3.1}$$

In this metric $T(a)$ represent the trust value for actor a, and the rest of the parameters are denoted as follows:

- $I(a)$ the total number of transaction the actor a has joined,
- $R(a,i)$ the reputations that the actor a receives for the transaction i.

According to this metric, reputations are simply added to each other and, trust can change between -1 and 1.

3.1.2.1 Reputation an Actor Receives

The reputation received from other actors is the main trust parameter. Reputation in this context is the reflection of an actor's satisfaction related to a transaction with another actor. This parameter is represented in the basic trust metric of AcTrust with $R(a)$ and is equal to "$\frac{1}{2}$" if the actor is satisfied and "$-\frac{1}{4}$" other wise. By giving unequal reputation values we allow the trust metric to grow slower in case of failure and faster in case of success.

3.1.2.2 Total Number of Transactions

Most of the P2P E-commerce systems of these days like E-bay or Yahoo collects the feedbacks from peers that take part in interactions and add these to each other in order to achieve a trust value. Reputation values calculated with these feedback-only binary systems can only give correct results if each peer has equal number of transactions, and otherwise the resulting outcome is unfair.

In reputation systems that calculate the trust value of an actor just by adding up the submitted reputations, malicious actors can hide their frequent bad behavior by increasing the number of reputations. For example, if an actor had 2 bad reputations and 5 good then his trust value would be equal to 3 as the trust value of another actor who only has 2 interactions both of which are graded as good has the trust value 2. In order to disable an actor of hiding it's bad reputations, and betraying other actors, AcTrust includes the total number of transactions as a scope factor for comparing the feedback in terms of degree of satisfaction among different peers.

The total number of transactions allow to calculate an average amount of satisfaction actor a receives for each transaction. So the basic trust metric changes as follows, and shows the ratio of the total amount of satisfaction an actor receives over the total number of transactions it has.

$$T(a) = \frac{\sum_{i=1}^{I(a)} R(a,i)}{I(a)} \tag{3.2}$$

3.1.2.3 Temporal Adaptivity of a Reputation

Every reputation that is sent to the Reputation Manager affects the eventual trust record of the related actor. Since an actor can change its behavior during the time in a positive or negative way, a temporal adaptation of the trust score to the more recent reputation would give more precise results.

Xiong suggest for this problem a solution in [30] called *temporal adaptivity*, which works by multiplying the feedbacks in the history and the last feedback with weight factors. This solution has several shortcomings. For example all of the past reputations have to be stored in a database, which increases the memory consumption.

AcTrust realizes the temporal adaptivity of the feedbacks by using the following trust equation.

$$T(a) = \sum_{i=1}^{I(a)} \left(\frac{R(a,i)}{2^{(I(a)-i)}} \right) \tag{3.3}$$

According to this equation, the relevance of each reputation sinks exponentially. Every time a new reputation is received, a fresh trust value is calculated and the old trust value in the database is over written by the new value. This way both of the above-mentioned shortcoming of PeerTrust are solved. Besides that AcTrust minimizes the memory consumption, since the reputations of the old transactions must not be stored any more. Simply by using the following formula a new trust value can be calculated.

Let T' symbolize the new trust value and $R(a, last)$ the last reputation the peer receives about peer a.

$$T(a)' = \frac{T(a)}{2} + R(a, last) \tag{3.4}$$

3.1.3 Trust Policy of AcTrust

In this section, we are going to describe the trust policy of AcTrust in order to define, which actors are allowed to communicate with each other at any given time.

In AcTrust, actors are allowed to communicate only with those actors in their trust maps and at the same time having a higher trust value than the threshold of their communication partners. An actor can either develop trust in a direct way, by registering a trust value for the actors it has communicated with and updating it on subsequent transactions, or in an indirect way, by referring to the trust information it gets from other trusted actors. The indirect way of developing trust refers to the transitivity property of trust, which is explained in Subsection 3.1.1.3.

In AcTrust, we modeled a *trust hierarchy*. Here we concerned about two hierarchical trust levels: network level, and level of neighbors. Neighbors are the set of all peers that are in the trust map of a peer and are tagged as trustworthy. If peer X is questioning the trust level of another peer Y, it first refers to the trust value of Y in his trust map. If Y is in the trust map then he accepts the interaction request. In Y is not in the trust map then he asks the trusted peers in the network. If the majority of the trusted peers say that this peer is trustworthy then X carries Y to his trust map as trustworthy, other wise as untrustworthy.

3.2 Trusted Actor Language

In this section, we are going to describe the Trusted Actor Language (TAL). TAL extends the levels in AMST, which itself extends the λ-calculus.

Trusted Actor Language describes an actor's behavior by a lambda abstraction, which embodies the code to be executed when a message is received. The state of an actor can be *ready* or *busy* processing a message. The messages that are received by an actor are interpreted by applying the actors behavior to the content of the message.

In AcTrust every actor is associated with a unique name, a trust map, and a threshold. Actors restrict communication to other actors within specific trust maps. Before an actor interprets a message it checks the trustworthiness of the sender by using trust map. If the sender actor does not appear in the receiver's trust map, other trusted actors are contacted to provide feedback on the sender's trustworthiness. Using this method, no unprivileged actor can gain access to a resource. Every actor has a trust map and each entry in the trust map is annotated with an actor's unique name and trust information. Only the messages from those actors that are in the trust map of an actor and that have a higher trust value than the threshold are interpreted.

Actors communicate with other actors or their surrounding in order to fulfill their duties. Communication between actors is realized with asynchronous, guaranteed, and point-to-point messaging. Messages are buffered in an input queue. Every message is a triple of sender's name, receiver's name and content.

An actor in AcTrust can perform the following actions in reaction to a message:

1. Change its local state and become ready to process the next message in its mail queue,
2. Send messages to peer actors,
3. Create a finite set of actors with some initial behaviors and thresholds,
4. Update its trust map by including a new actor or changing the trust value of an actor that is already in its trust map, and/or
5. Find out if an actor is trustworthy.

In order to fulfill their duties concurrently, actors may create other actors. In that case the children inherit the trust map from their parent, and include their parent as a trusted peer in their trust map. Furthermore the trust map of the parent is also extended with the child, as a trusted peer.

3.2.1 Language Constructs

The λ-calculus is used to model sequential computation inside an actor. TAL extends it by adding primitives for trusted actor communication as follows:

- `new(b,t)` creates a new actor with behavior `b` and initial threshold `t`
- `send(a,c)` sends a message with content `c` to actor `a`

- `ready(b)` makes an actor ready to receive a new message using behavior `b`
- `register(a,t)` saves the mapping of actor `a` to trust value `t` into the trust map
- `trustworthy?(a)` looks up in the trust map of an actor to see if an actor `a` is trustworthy or not
- `success(a)` updates the actor's trust map after a successful transaction with actor `a`
- `failure(a)` updates the actor's trust map after an unsuccessful transaction with actor `a`

3.2.2 Actor Configurations

Let $\mathbf{T} = \{t | t \in \mathbb{R},\ -1 \leqslant t \leqslant 1\}$ be the set of *trust values*, and assume that sets *atoms* (**AT**) and *variables* (**X**) are given, then the set of *values* (**V**), *expressions* (**E**) and *messages* (**M**) are defined inductively as follows:

$\mathbf{V} = \mathbf{AT} \cup \mathbf{X} \cup \lambda\mathbf{X}.\mathbf{E} \cup pr(\mathbf{V},\mathbf{V}) \cup \mathbf{T}$

$\mathbf{E} = \mathbf{V} \cup app(\mathbf{E},\mathbf{E}) \cup \mathbf{F}_n(\mathbf{E}^n)$, where $\mathbf{F}_n(\mathbf{E}^n)$ is all arity-n primitives.

$\mathbf{M} = \langle \mathbf{V} \Leftarrow \mathbf{V} \rangle_V$

The set **X** of variables represents actor names. At any given point an actor can be either ready to receive a message, `ready(e)`, or currently busy with execution of some expression `e`. And $< a \Leftarrow c >_{a'}$ denotes a message with content c sent to an actor a by an actor a'.

Let $\mathbf{M}_\omega[\mathbf{M}]$ be the set of multi-sets with elements in **M**, $\mathbf{X}_0 \xrightarrow{f} \mathbf{X}_1$ be the set of finite maps from $\mathbf{X}_0$ to $\mathbf{X}_1$, Dom(f) be the domain of f, and **FV(e)** be the set of free variables in e. $\alpha\{[e]_a\}$ defines an extended mapping, which maps a into e, and all other actor names a' into $\alpha(a')$, and $\alpha\{[e]_a, [e']_{a'}\}$ as $\alpha\{\{[e]_a\}\{[e']_{a'}\}\}$ for $a \neq a'$. For a set **S**, we use $|S|$ to denote its size.

Definition 3.2.1. *An actor configuration is a global snapshot of a group of actors. It consists of an actor map, α, which maps each actor name to a behavior; a multi-set of messages in transit, μ; and a global trust map, τ, which relates actor names to* *trust maps, which themselves map their acquaintances into trust values. Accordingly, we write:*

$$\langle \alpha \mid \mu \mid \tau \rangle$$

where $\alpha \in \boldsymbol{X} \xrightarrow{f} \boldsymbol{E}$, $\mu \in \boldsymbol{M}_\omega[\boldsymbol{M}]$, *and* $\tau \in \boldsymbol{X} \xrightarrow{f} \boldsymbol{X} \xrightarrow{f} \boldsymbol{T}$, *such that* $\boldsymbol{A} = Dom(\alpha)$, *then:*

1. *If* $a \in \boldsymbol{A}$, *then* $\boldsymbol{FV}(\alpha(a)) \subseteq \boldsymbol{A}$, *and if* $< v_0 \Leftarrow v_1 >_{v_2} \in \mu$ *then* $\boldsymbol{FV}(v_i) \subseteq \boldsymbol{A}$ *for* $i < 3$.
2. $Dom(\tau) = \boldsymbol{A}$, $\forall\ a \in \boldsymbol{A} : Dom(\tau(a)) \subseteq \boldsymbol{A}$.
3. $\forall\ a \in \boldsymbol{A} : a \in Dom(\tau(a))$.

Actor that are defined in a configuration above can only send messages to the actors that are also defined in the system. Every actor has a trust map and all the actors in the trust map are put to the configuration. Furthermore, trust map of every actor involves the trust value of that actor, which is the trust threshold for that actor.

Let $\alpha(a) =$ `ready(b)` and let $\tau(a)(a) = t$ be the behavior of an actor bound to the name a and its threshold value in some actor configuration $\langle \alpha \mid \mu \mid \tau \rangle$. The behavior of a, i.e., it's response to the next incoming message, is represented by b, and the threshold of a, i.e. the threshold that defines actors below it as *untrustworthy* and the rest as *trustworthy*, is represented by t.

3.2.3 Operational Semantics

To describe the internal transitions between configurations, an expression can be decomposed into a reduction context filled with a redex. The notation $R[\![e]\!]$ represents a redex `e` in a reduction context R.

We define then a transition relation between actor configurations as the least relation satisfying the following rules:

`[pure a]`

$$e \xrightarrow{\lambda}_{Dom(\alpha)\cup\{a\}} e' \Rightarrow \langle \alpha\{[e]_a\} \mid \mu \mid \tau \rangle \quad \mapsto \quad \langle \alpha\{[e']_a\} \mid \mu \mid \tau \rangle$$

`[new b,t']`

$$\langle \alpha\{[R[\![new(b,t')]\!]]_a\} \mid \mu \mid \tau\{[\tau_a\{[t]_a\}]_a\}\rangle \;\mapsto$$
$$\langle \alpha\{R[\![[a']_a,[b]_{a'}]\!]\} \mid \mu \mid \tau\{[\tau_a\{[t]_a,[1]_{a'}\}]_a,[\tau_a\{[1]_a,[t']_{a'}\}]_{a'}\}\rangle \quad a' fresh$$

`[send a',c]`

$$\langle \alpha\{[R[\![send(a',c)]\!]]_a\} \mid \mu \mid \tau\;\rangle \;\mapsto$$
$$\langle \alpha\{[R[\![nil]\!]]_a\} \mid \mu \uplus < a' \Leftarrow c >_a \;\mid \tau\;\rangle$$

`[receive c,a']`

$$\langle\; \alpha\{[R[\![ready(b)]\!]]_a\} \mid\; < a \Leftarrow c >_{a'} \uplus \mu \mid \tau\{[\tau_a\{[t']_{a'},[t]_a\}]_a\}\rangle \;\mapsto$$
$$\langle\; \alpha\{[app(b,c)]_a\} \mid \mu \mid \tau\{[\tau_a\{[t']_{a'},[t]_a\}]_a\}\rangle \quad \text{, if } t' \geqslant t$$

`[ignore c,a']`

$$\langle \alpha\{[R[\![ready(b)]\!]]_a\} \mid\; < a \Leftarrow c >_{a'} \uplus \mu \mid \tau\{[\tau_a\{[t']_{a'},[t]_a\}]_a\}\rangle \;\mapsto$$
$$\langle\; \alpha\{[ready(b)]_a\} \mid \mu \mid \tau\{[\tau_a\{[t']_{a'},[t]_a\}]_a\}\rangle \quad \text{, if } t' < t$$

`[trustprop a,a']`

$$\langle \alpha\{[R[\![ready(b)]\!]]_a\} \mid\; < a \Leftarrow c >_{a'} \uplus \mu\rangle \mid \tau\{[\tau_a\{[t]_a\}]_a\}\rangle \;\mapsto$$
$$\langle \alpha\{[ready(b)]_a\} \mid\; < a \Leftarrow c >_{a'} \uplus \mu\rangle \mid \tau\{[\tau_a\{[t]_a,[t']_{a'}\}]_a\}\rangle \qquad , \; a' \notin dom(\tau_a)$$
$$let\ \mathbf{A} = \{b \in dom(\tau_a)\ s.t.\ \tau_a(b) > t, a' \in dom(\tau(b))\}\ ,$$
$$then\ t' = \begin{cases} \frac{\sum_{b \in A}(\tau_a(b) * \tau(b)(a'))}{|A|}, & if\ \mathbf{A}\ \neq \emptyset \\ -1, & \text{otherwise.} \end{cases}$$

`[register a',t]`

$$\langle \alpha\{[R[\![register(a',t)]\!]]_a\} \mid \mu \mid \tau\{[\tau_a]_a\}\rangle \;\mapsto$$
$$\langle \alpha\{[R[\![nil]\!]]_a\} \mid \mu \mid \tau\{[\tau_a\{[t']_{a'}\}]_a\}\rangle$$

`[trustworthy?`$_1$` a']`

$$\langle \alpha\{[R[\![trustworthy?(a')]\!]]_a\} \mid \mu\mid \tau\{[\tau_a\{[t]_a,[t']_{a'}\}]_a\}\rangle \;\mapsto$$
$$\langle \alpha\{[R[\![false]\!]]_a\} \mid \mu\mid \tau\{[\tau_a\{[t]_a,[t']_{a'}\}]_a\}\rangle \quad \text{, if } t' < t$$

`[trustworthy?`$_2$` a']`

$$\langle \alpha\{[R[\![trustworthy?(a')]\!]]_a\} \mid \mu\mid \tau\{[\tau_a\{[t]_a,[t']_{a'}\}]_a\}\rangle \;\mapsto$$
$$\langle \alpha\{[R[\![true]\!]]_a\} \mid \mu\mid \tau\{[\tau_a\{[t]_a,[t']_{a'}\}]_a\}\rangle \quad \text{, if } t' \geqslant t$$

`[success a’]`

$$\langle \alpha\{[R[\![success(a')]\!]]_a\} \mid \mu \mid \tau\{[\tau_a\{[t]_a,[t']_{a'}\}]_a\}\rangle \;\mapsto$$
$$\langle \alpha\{[R[\![nil]\!]]_a\} \mid \mu \mid \tau\{[\tau_a\{[t]_a,[t'']_{a'}\}]_a\}\rangle \quad , t'' = \frac{t'}{2}+\frac{1}{2}$$

`[failure a’]`

$$\langle \alpha\{[R[\![success(a')]\!]]_a\} \mid \mu \mid \tau\{[\tau_a\{[t]_a,[t']_{a'}\}]_a\}\rangle \;\mapsto$$
$$\langle \alpha\{[R[\![nil]\!]]_a\} \mid \mu \mid \tau\{[\tau_a\{[t]_a,[t'']_{a'}\}]_a\}\rangle \quad , t'' = \frac{t'}{2}-\frac{1}{4}$$

- `[pure a]` evaluates pure redex.
- `[new b,t’]` is used to create a new actor with given behavior `b` and threshold `t’`. The set of trust maps in the system is extended with the trust map of `a’`, which is inherited from `a` and extended with `a’`. Furthermore, the trust map of the parent, `a`, is extended with the child, `a’`. Since the parent trusts the child and the child trusts its parent, the trusts value of child in the trust map of the parent, as well as the trust value of the parent in the trust map of the child, is set equal to 1.
- `[send a’,c]` is used to send message to an actor `a’` with content `c`.
- `[receive c,a’]` and `[ignore c,a’]` these rules define when for an actor `a` receives or ignores a message with content `c` from another actor `a’`. If the sender is in the trust map of the actor `a` and has a trust value that is equal to or higher then the threshold of `a`, than the message is accepted. If the actor `a’` is in the trust map but has a lower trust value then the the threshold of `a`, than the message is removed from the message queue and ignored.
- `[trustprop a,a’]` is used in case the actor `a’` send a message to the actor `a`, and `a’` is not in the trust map of actor `a`. Then the message remains in the message queue and the actor `a` computes `a’`’s trust value bound on all the peers in his trust map that have a trust value higher or equal to `a`’s threshold. Actor `a` extends its trust map with this information. If no trusted actors know about a’, it sets the trust value of the actor `a’` to -1 and extends its trust map with this information.

- `[register a',t]` extends trust map of an actor `a` with the mapping (`a'`$\rightarrow$ `t`)
- `[trustworthy? a']` is used to look in the trust map in order to see if an actor `a'` is trustworthy or not. If the trust value in the trust map is higher or equal to the threshold of the actor `a`, than the rule yields `true`, otherwise `false`.
- `[success a']` is used in order to calculates a new trust value for the interaction peer `a'` after a successful interaction and update its trust value in the trust map.
- `[failure a']` is used in order to calculates a new trust value for the interaction peer `a'` after an unsuccessful interaction and update its trust value in the trust map.

3.2.4 Security Properties

In this section we are going to give some basic properties of the Trusted Actor Language and their proofs.

Lemma 3.2.1. *Trust values for peer actors are inherited.*

Proof. During the propagation of trust, according to the rule **trustprp a,a'**, it is shown that the trust value of new peers are calculated by using the following formula:

$$t' = \begin{cases} \frac{\sum_{b\in A}(\tau_a(b)*\tau(b)(a'))}{|A|}, & if\ \mathbf{A}\ \neq \emptyset \\ -1, & \text{otherwise.} \end{cases}$$

Using this formula, the trust value of every other peer in the trust map of an actor is multiplied with 1, because according to rule [**new b,t'**], every child peer trusts its parent, which is denoted by 1 as the trust value of the parent peer in the trust map of a child. Therefore, it is said that trust values for peer actors are inherited. ■

Lemma 3.2.2. *A new actor, which is also not created by an actor in the system, has the lowest trust value in the system.*

Proof. According to Definition 3.2.1, all of the actors in the system are known by at least one other actor. If a new actor **y** wants to join the system, and communicate with other actors, it sends a message to one of the actors, let this actor be **x**, in the system. Since y is a new actor, it is not in the trust map of **x**. Therefore, actor **x** collects feedback from it's trusted peers, using the rule [**trustprop**]. Since actor **y** is new to the system, non of **x**'s trusted peers will send feedback, and because of that actor **x** is going to set the trust value of **y** to -1, which is the lowest value in the system. ■

Theorem 3.2.3. *If neither an actor a, nor its trusted peers know about an actor b, and the threshold of actor a is not equal to -1, then that actor b is untrusted, and therefore b is unable to communicate with a.*

Proof. If a peer **a'** send a message to another peer **a**, according to the rule [**receive c,a'**] the peer **a'** hast to be in the trust map of **a** and the trust value of **a'** has to be higher than the threshold of **a**.

If a peer is not in the trust map, than he can not have a trust value. In that case the message waits in the message que. until a trust value for the sender peer is propagated.

For the propagation of a trust value the [**trustprp a,a'**] rule is used. According to which, the receiver peer asks his trusted parties for trust information and processes this information as follows:

$$t' = \begin{cases} \frac{\sum_{b\in A}(\tau_a(b)*\tau(b)(a'))}{|A|}, & if\ \mathbf{A}\ \neq \emptyset \\ -1, & \text{otherwise.} \end{cases}$$

If non of the trusted parties of receiver peer has trust information on the sender peer, then $\mathbf{A} = \emptyset$ and the trust value of the sender peer is set equal to -1. Since the threshold of the receiver peer is higher then -1, the ignores the message using the [**ignore c'a'**] rule. Which point to the fact that they are unable to communicate. ■

Corollary 3.2.4. *If actor **a** receives a message from actor **b**, then **b** is trustworthy for **a**; i.e., in its trust map or its peer's trust maps.*

Lemma 3.2.5. *Feedback from untrusted peers will not influence communication with a third party.*

Proof. According to [**trustprop a,a'**] only the trust information form those peers are taken in consideration that have a trust value higher then the threshold ($\tau_a(b) > t$). According to the rule [**trustworthy? a'**] trust value of untrusted peers are lower than the threshold. Therefore, the feedback of untrusted peers will not be take to consideration. ∎

3.2.5 Final Remarks

We have developed TAL in an analogous manner to SMAL, described in Section 2.2, which has a layered structure. So other properties, like mobility, of SMAL can easily be imported to TAL.

Furthermore, trust maps work in a similar manner to the dependence maps in the Transactors Model, described in Section 2.3, which map each transactor name to a history value. A trust map maps each actor name a, on which it is defined, to a trust value.

The structure of messages allows selection of trusted senders via trust maps. In every message the sender is annotated. A message receiver, uses his trust map in order to find out if the sender is trustworthy or not. This process is similar to the usage of history values and dependence maps in the transactor model.

4. Examples of Trust Based Communication

In this chapter, we are going to present two examples, which use AcTrust. The first example shows an E-Commerce application with actors, where the actors look for some level of trust in order to communicate with other actors. The second example shows, how trust can be used for secure file sharing in P2P communities.

4.1 E-Commerce

4.1.1 Marketplace Model Overview

One of the main application fields of reputation based trust is E-Commerce. In this section we are going to imply AcTrust to the Marketplace model that we have explained in [19] (See Figure 4.1 for the model.) in order to show, how AcTrust can be used in an E-Commerce application, where the sellers and the buyers of the real world are represented by online actors and interact in means of auctions in order to fulfill their duties.

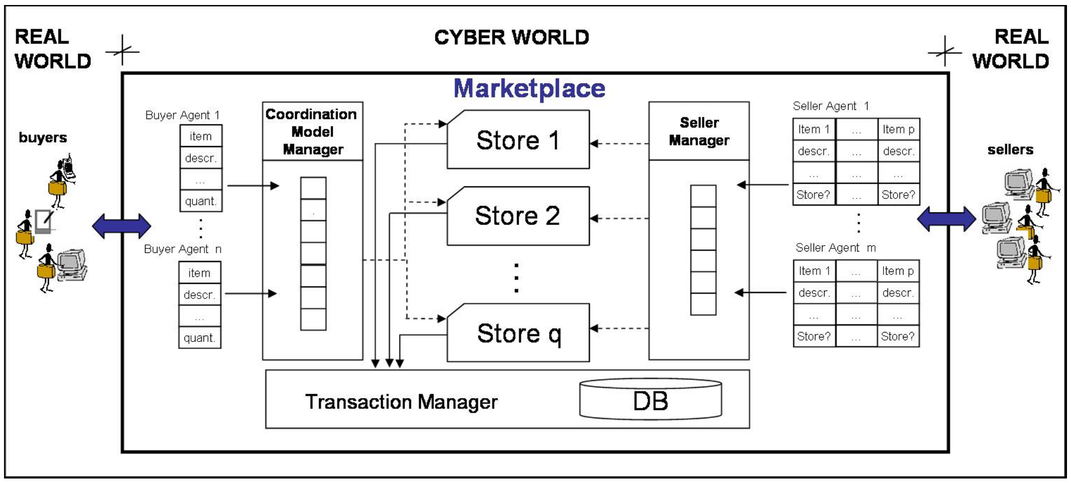

Figure 4.1: The Electronic Marketplace Model.

The Marketplace Model consists of two types of actors; *manager actors* and *representing actors*. Manager actors are Coordination Model Manager (CMM), who is responsible of choosing the most suitable actor coordination model and sending the actors that represent buyers of the real world to stores; Seller Manager (SM), who is responsible of choosing and sending the seller actors to stores, where they can meet with the buyer actors; Store Manager (StM), who is responsible for matching the seller and buyer actors and creation of auctions; and Transaction Manager (TM), who is responsible of the transactions that take place after each auction. Manager actors are not mobile in comparison to the representing actors. There are two types of representing actors in the model: actors that represent the sellers and actors that represent the buyers.

4.1.2 Marketplace Example

Actors join the marketplace every time by connecting the CMM or SM. When a new buyer or seller wants to join the marketplace, the system generates a new actor, which delegates the representing actor in the auctions that take place in the marketplace. Depending on whether the representing actor is a seller or buyer, respectively CMM or SM extends its trust map with trust information of the representing actor, which is 1. When a new actor is created, it inherits the extended trust map of its parent, and sets the trust value of the parent to 1. The new actors are trustworthy, since they are created by a manager actor and the manager actor which is the parent is trustworthy. Depending on the goal of a representing actor, CMM or SM may generate only one actor and send it to a series of auctions or creates siblings and sends them to different auctions.

After an actor migrates to a store it starts interacting with the StM. Since the actor is new the to the StM, StM does not have the actor in his trust map, but gets its trust information from CMM or SM. If the actors that join an auction, do not have each other's trust information in their trust maps and want to interact with each other, then they ask the actors they trust according to their trust maps for the trust information of that new actor.

At the end of every auction, before migrating to a new location, actors recalculate the trust values of their transaction partners and refreshes their trust maps with the new information. The participating actors furthermore send feedback information to the store manager and the StM calculates a trust value for the participating actors and overwrite the old entries. After that StM sends these trust information of representing actors to the SM and CMM.

The goal of having partially centralized information storage is to prevent SM or CCM sending malicious actors to Stores and in the same time to prevent StM send these malicious actors to auctions. Since there is not only one trust center, the system is protected from single point attacks, which may easily end up with denial of service.

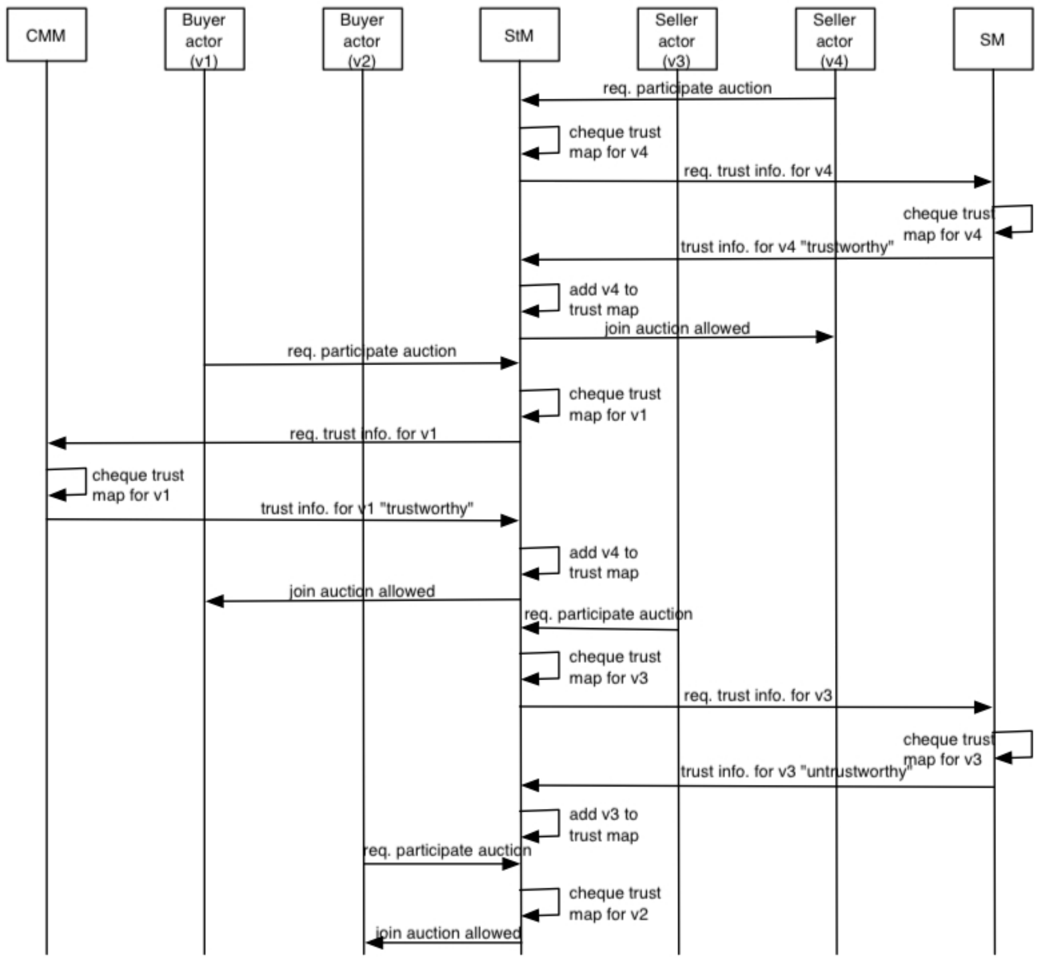

Figure 4.2: Message flow before an auction begins.

The message flow from the moment an actor is sent to a store until the beginning of an auction is shown in Figure 4.2. We assume that StM has in the beginning of the interaction only the actor v_2 in his trust map. We furthermore assume that actor v_3 is tagged as untrustworthy in the trust map of SM and the rest of the actors are tagged in the trust maps of SM and CMM as trustworthy. In this figure we can see how the trust propagation between the actors that does not know each other is realized in AcTrust.

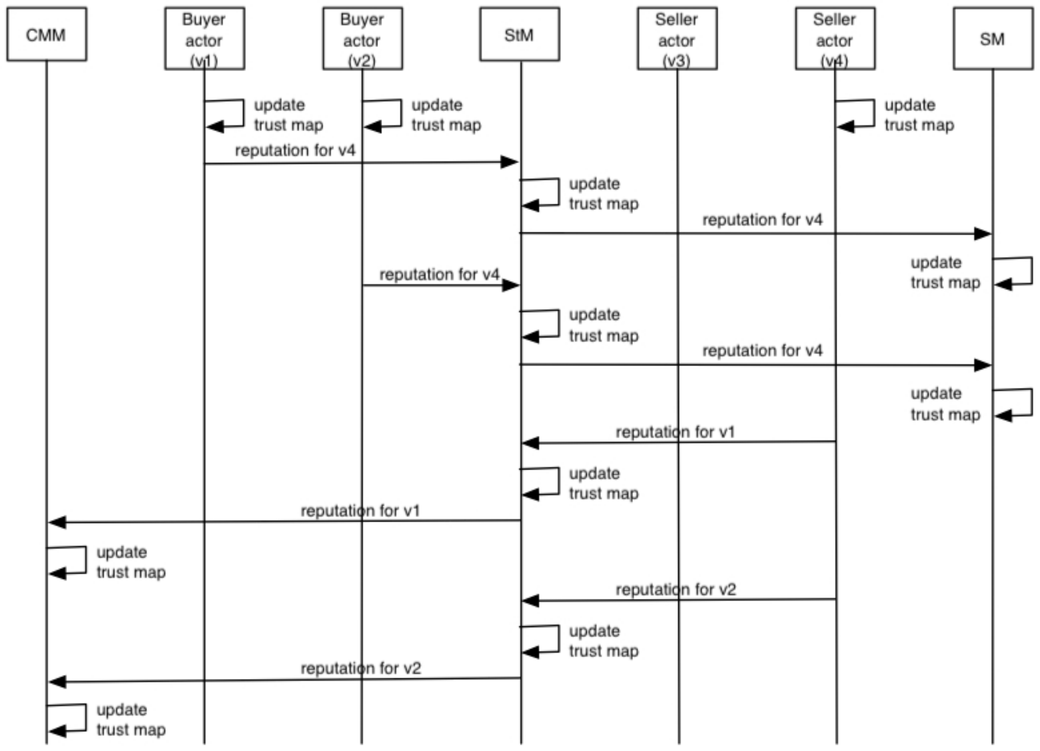

Figure 4.3: Message flow after an auction finishes.

In Figure 4.3 we show the message flow that takes place after an auction ends and before the actors migrate to a different Store or join another auction. The actors in Figure 4.3 are the actors from the Figure 4.2, and we assume that this messaging takes place following the auction that starts after the trust propagation process in Figure 4.2.

4.2 P2P Communication

In this section, we are going to describe a P2P communication example using AcTrust in an environment, where the file transfer is realized using BitTorrent. Through AcTrust only the trusted peers are allowed to download and upload file pieces from each other, so that malicious peers will not overload the network resources.

4.2.1 BitTorrent Overview

BitTorrent is a second generation P2P file distribution application, which makes it possible to upload pieces of a file from different users, accounted in the recent years for a large and growing share of P2P Internet traffic [22]. This success of BitTorrent is established by distributing large files quickly and efficiently without overwhelming the capacity of single hosting machine [3]. BitTorrent can be used; if all users are downloading the same file at the same time and it redistributes the upload costs to all downloaders. As a result, BitTorrent makes hosting of a file with a potentially unlimited number of downloaders affordable [5].

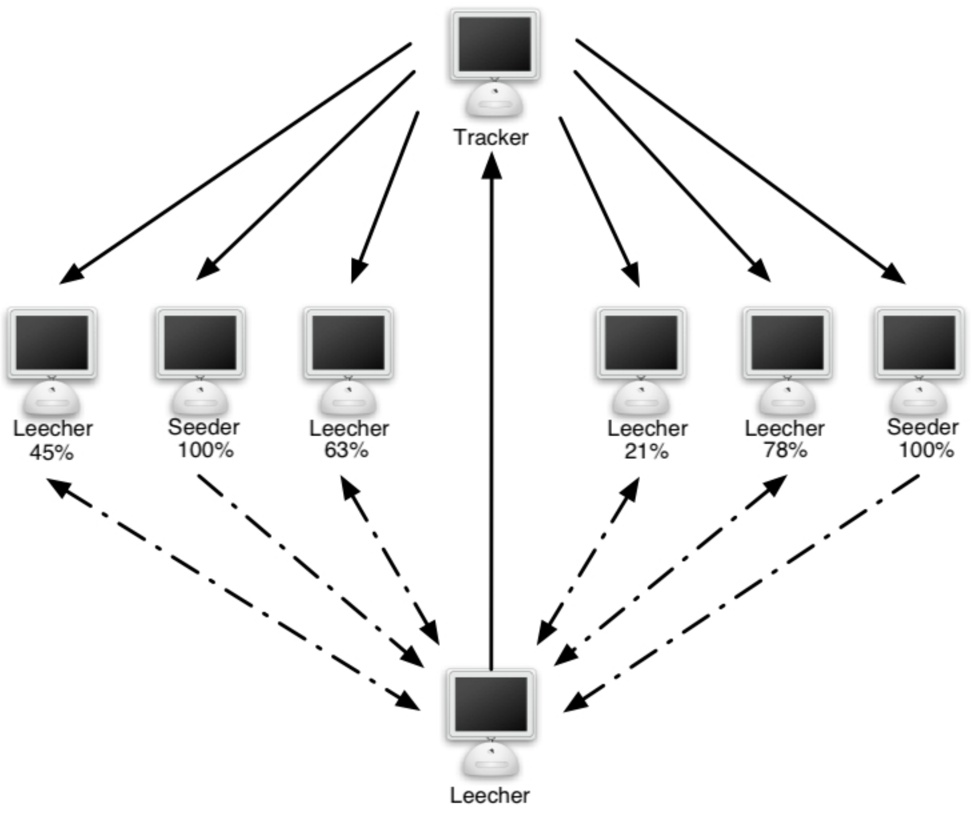

Figure 4.4: The architecture of BitTorrent.

In Figure 4.4, the architecture of this distributed application is shown, where solid arrows show the information flow, and dashed arrows denote the file downloads and uploads. A peer that wants to download a file with using BitTorrent contacts the BitTorrent tracker. *Tracker* is a central component that keeps track of the peers in the system, in order to help downloaders find each other. *Seeds* are the peers that have the complete copy of the file and are willing to serve it to others, which are called *leechers*. In BitTorrent network, as a peer is downloading pieces of a file, other peers download the pieces it already has from him. The nodes that are connected to each other in order to download a file are *neighbors*. Nodes inform the tracker periodically as well as when they join or leave the torrent [3]. Each large file available for downloading is called *torrent*. Each node limits the number of current uploads to a small number, typically 5. The mechanism used to limit the number of current uploads is called *choking* and respectively uploading is called *unchoking*.

In order to publish a content in BitTorrent, a static file with the extension `.torrent` is put on a web server. This file contains information about the name, size, and the hashing information of the file and the URL of the tracker. In order to verify data integrity, the SHA1 hashes of all pieces are included in the `.torrent` file. Peers report that they have a piece of the file to the tracker only after checking its hash.

Since the peers can hash only after downloading the complete file, it is not possible to know prior whether the file in download is the desired file or not. Because of that BitTorrent is open to malicious attacks. We suggest using a reputation based trust mechanism in order to download files only from those peers that are trustworthy.

4.2.2 BitTorrent Example

Peers can join a BitTorrent network anytime they want to. After joining the network each node looks for opportunities to download and upload blocks from and to its neighbors, and in order to do this, they connect to the tracker.

We suggest here to extend the tracker with a trust map. So that the trustworthiness of each peer is checked before sending the list of peers that have the required pieces of the file. Furthermore, every node defines a threshold, when the tracker is checking the trust list in order to send a list of the trusted nodes, he compares the trust values of the seeds and leechers in the list with the threshold of the demander, and sends the information of only those nodes that have a higher trust value then the supplier. If none of the nodes have a lower trust value, then tracker sends an empty list.

Furthermore, after each finished download the downloader peer computes the hash to proof the integrity of the file, if they are the same then he sends ”success” as a positive feedback to the tracker, otherwise he sends ”failure” as a negative feedback.

5. Conclusion and Future Work

5.1 Contribution of this Thesis

The goal of this thesis was to generate a comprehensive trust model for distributed computing and to develop a trust language with its computational semantics. To achieve it's goal, we have first described different actor models that can constitute a base for AcTrust, a reputation based trust model for grid computing, and then gave a brief overview of trust based systems for secure distributed computing, like Sporas, PeerTrust, and TRUMAR as well as summarized the main problems and challenges in trusted distributed computing.

In Chapter 3, we have described AcTrust with its general properties, such as collection and storage of trust information and propagation of trust, and we gave the mathematical formula for the calculation of trust in AcTrust. Furthermore, we introduced the Trusted Actor Language, which we have developed for trusted actor communication and reputation of trust information between the trusted peers in a distributed actor system, as well as specified the language and it's operational semantics. We have defined the Trusted Actor Language as a layer above the basic actor language, which is itself a layer above the λ-calculus and concentrated on evaluation and propagation of trust. We have concluded this chapter with the proofs of the basic properties of AcTrust model.

Giving two examples in Chapter 4 we have discussed, how AcTrust can be applied to e-commerce and p2p file sharing.

5.2 Future Directions

AcTrust is a very basic but in the meanwhile powerful model, which is supported by TAL. It is a flexible model that can be improved with different features. For instance, it is interesting to treat statistic analysis for different metrics of the trust calculation formulas, e.g. the value of the transaction, or a dynamic community context factor and its relationship with that shown in this thesis. Furthermore, piggybacking trust information into messages constitutes a further direction. Since it makes the message traffic more transparent, while a peer is trying to calculate a trust value for a peer that is new to it.

Another extension of AcTrust is making a general update of the trust information periodically. The use of such an extension is to allow a more concurrent knowledge of trust, throughout the system. According to the AcTrust model, an actor updates it's trust map only if it receives a message from a new actor and after each transaction. Therefore, if a peer `y` in the trust map of an actor behaves maliciously with the other peers, after it's last transaction with a peer `x`, peer `x` notices this only during a new interaction with peer **y**. If the model has some kind of a periodic trust map update property, than the peer `x` can know about the maliciousness of peer `y`, before interaction with it. For a periodic update, peer `x` can use a rule similar to [**trustprop**], with an additional trust value, which it retrieves from this trust map.

A trust management system that works with a dynamically changing desired lowest trust value depending on a risk analysis is another challenging future direction. This property will allow the peers to dynamically change their threshold, depending on how urgent the situation is. This property can especially support systems of online supply chain management, like Singh et al. has described in [23]. The success in his work creating information transparency through effective integration of information flows by intelligent agents across the multiple electronic marketplaces that comprise the internet enabled supply chain.

5.3 Conclusion

Communities are getting more and more connected in the digital world. As the physical distance loses its relevance, consumption of distributed resources gains more weight. We believe that in the close future communication and interaction in almost every field will be realized over digital networks. This evaluation will make many processes in our lives significantly faster, but will also involve various challenges, like security problems and power consumption in mobile peers. In this thesis, we have concentrated on the security problems of grid computing.

As the digital world mimics the real world from many different perspectives, we have suggested in this thesis a security model that mimics a sociological factor, "trust". In the real world, every person that communicates with another, builds a level of trust by analyzing the previous communications or the feedbacks received from other people, and depending on the level of trust adjusts his behavior. Furthermore, trust based security is easy to apply to the digital world.

In this thesis, we have tried to combine some properties of different actor communication models with reputation based trust. Although trust improves the security of such systems, there are still challenging security properties that can be applied to distributed computing for achieving secure communication. Further study on this field is critical to the future of secure distributed computing.

REFERENCES

[1] Gul Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[2] Gul Agha, Ian A. Mason, Scott F. Smith, and Carolyn L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7(1):1–72, 1997.

[3] Ashwin R. Bharambe andCormac Herley and Venkata N. Padmanabhan. Analyzing and improving bittorrent performance. Technical report, Microsoft Research, 2005.

[4] M. Chen and J.P. Singh. Computing and using reputations for internet ratings computing and using reputations for internet ratings. In *Proceedings of Third ACM Cnference on Electronic Commerce*, pages 154 – 162, Florida, USA, 2001.

[5] Bram Cohen. Incentives build robustness in bittorrent. In *Proceedings of 1st Workshop on Economics of Peer-to-Peer Systems*, SIMS Berkeley, 2003.

[6] B. Cubaleska and M. Schneider. Applying trust policies for protecting mobile agents against. In *Proceedings of Third international Workshop on Policies for Distributed Systems Systems and Networks*, pages 198–201, Moterey, California, 2002.

[7] C. Dellarocas. The digitalization of word-of-mouth: Promise and challenges of online reputation mechanisms. *Management Science*, vol. 49(no. 10):1407 – 1424, 2003.

[8] G. Derbas, A. Kayssi, H. Artail, and A. Chehab. Trummar - a trust model for mobile agent systems based on reputation. In *Proceedings of the IEEE/ACS Inernational Conference on Pervasive Services*, pages 113–120, 2004.

[9] M. Deutsch. Cooperation and trust: Some theoretical notes. In M. R. Jones, editor, *Nebraska Symposium on Motivation*, pages 275–320. Nebraska University Press, 1962.

[10] R. Falcone, K. S. Barber, L. Korba, and M. Singh. Challenges for trust, fraud, and deception research in multi-agent systems. In *Trust, Reputation, and Security: Theories and Practice*, Lecture Notes in Artificial Intelligence, pages 8–14. Springer Verlag, 2003.

[11] Rino Falcone, Giovanni Pezzulo, Cristiano Castelfranchi, and Gianguglielmo Calvi. Why a cognitive trustier performs better: Simulating trust-based contract nets. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, pages 1394–1395, 2004.

[12] J. Field and C. Varela. Transactors: A programming model for maintaining globally consistent distributed state in unreliable environments. In *Proceedings of ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages*, pages 195–208, 2005.

[13] D. Gambetta. Can we trust trust? In *Trust: Making and Breaking Cooperative Relations, Department of Sociology, University of Oxford (2000) 213-237*, pages 213–237. Department of Sociology, University of Oxford, 2000.

[14] R. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 403–412, New York, NY, USA, 2004. ACM Press.

[15] Audun Jsang and Simon Pope. Semantic constraints for trust transitivity. In *CRPIT '43: Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling*, volume 43, pages 59–68, Newcastle, New South Wales, Australia, 2005. Australian Computer Society, Inc.

[16] R. Jurca and B. Faltings. An incentive compatible reputation mechanism. In

Proceedings of IEEE Conference on E-Commerce, Newport Beach, CA, pages 285–292, 2003.

[17] M. Kinateder and K. Rothermel. Architecture and algorithms for a distributed reputation system. In P. Nixon and S. Terzis, editors, *Proceedings of the First International Conference on Trust Management*, volume 2692 of *LNCS*, pages 1–16, Crete, Greece, May 2003. Springer-Verlag.

[18] K.-J. Lin, H.Lu, T. Yu, and C.Tai. A repuataion and trust management broker framework for web application. In *Proceedings of the IEEE International Conference on e-Technology and e-Services*, pages 262–296, 2005.

[19] A. Morali. Security aspects of digital actors. Bachelor Thesis, June 2005.

[20] L. Mui, M. Mohtasshemi, and A. Halberstadt. A computational model of trust and reputation. In *Proceedings of the 35th Hawaii International Conference on System Sciences*, page 188, Big Island, Hawaii, 2002. IEEE.

[21] J. M. Pujol, R. Sangüesa, and J. Delgado. Extracting reputation in multi agent systems by means of social network topology. In *Proceedings of the first international Joint Conference on Autonomious Agens and MultiAgeny Systems*, pages 467–474, Bologna, 2002.

[22] D. Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *Proceedings of ACM SIGCOMM*, pages 367–378, Portland, Oregon, USA, August – September 2004.

[23] Rahul Singh, A. F. Salam, and Lakshmi Iyer. Agents in e-supply chains. *j-CACM*, 48(6):108–115, 2005.

[24] A. Tajeddine, A. Kayssi, A. Chehab, and H. Artail. A comprehensive reputation-based trust model for distributed systems. In *Security and Privacy for Emerging Areas in Communication Networks*, pages 118–127. Workshop of the 1st International Conference on Publication Date, September 2005.

[25] Robin D. Toll and Carlos Varela. Mobility and security in worldwide computing. In *Proceedings of the 9th ECOOP Workshop on Mobile Object Systems*, Darmstadt, Germany, 2003.

[26] R. Turlapani and M. N. Huhns. Multiagent reputation management to achieve robust software using redundancy. In *Proceedings of he 2005 IEEE/ACM Internetional Conference on Intelligent Agent Technologies*, pages 386–392, 2005.

[27] C. Varela. *Worldwide Computing with Universal Actors: Linguistic Abstractions for Naming, Migration, and Coordination.* PhD thesis, University of Illinois at Urbana-Champaign, 2001.

[28] M. Venkatraman, B. Yu, and M. P. Singh. Trust and reputation management in a small-worls network. In *Proceedings of Fourth international Conference on MultiAgent Systems*, Boston, Massachusetts, July 2000.

[29] Y. Wang and J.Vassileva. Trust and reputation model in peer-to-peer networks. In *Proceedings of the International Conference on P2P Communication*, 2003.

[30] L. Xiong and L. Liu. A reputation-based trust model for peer-to-peer ecommerce communities. In *Proceedings of the IEEE International Conference on E-Commerce*, pages 228–229, 2003.

[31] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge And Data Engineering*, 16(7):843–857, July 2004.

[32] Bing Yu and Munindar P. Singh. An evidential model of distributed reputation management. In *Proceedings of First International Conference on Autonomous Agents and MAS*, pages 294–301, Bologna, Italy, 2002.

[33] Giorgos Zacharia. *Collaborative Reputation Mechanisms in Electronic Marketplaces.* PhD thesis, Massachusetts Institute of Technology. Dept. of Architecture. Program in Media Arts and Sciences., 1999.

PREFACE

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation is not substantially the same as any that I have submitted for a degree or any other qualification at any other university. No part of this dissertation has already been, or is being currently submitted for any such degree, diploma or other qualification.

Ayşe Moralı
Troy-USA, den 19.04.2006