Incremental Trust in Grid Systems

Michael Brinkløv

LYNGBY 2005
M.Sc. THESIS
NR. 54

IMM

Printed at IMM, DTU

Preface

This report documents the M.Sc. project of Michael Hvalsøe Brinkløv. The project was carried out in the period from February 1st to August 1st, 2005 at the Technical University of Denmark, Department of Informatics and Mathematical Modelling. The project was supervised by Robin Sharp.

I would like to thank my supervisor for his great interest in my work and his constructive criticism and inspiration during the project period. I would also like to thank my family for their support and encouragement.

Kgs. Lyngby, August 1st, 2005

Michael Hvalsøe Brinkløv, s991094

Abstract

Access control is an important prerequisite for computer security in a distributed system. An important aspect of access control is the management of trust. Trust describes to which extent a system believes that the credentials for an access request complies with a given security policy. In a grid system this is a complex problem since it can be expected that new users with new credentials will try to access the system. It can also be expected that new credentials will be issued by new authorities. These possibilities provided by grid computing call for a flexible notion of trust.

In this M.Sc. project the aim is to investigate the use of a trust management system that relies on a notion of incremental trust. This means that the system's trust in credentials can increase or decrease. This change in trust depends on whether or not the activities attempted are acceptable or unacceptable from a security point of view. The project involves a study of relevant literature on trust and trust management. The subsequent formulation of heuristics leads to the implementation of a simulation program. The simulation program investigates fuzzy logic and Bayesian statistics when managing incremental trust in a grid system. The flexible design of the simulation program provides the possibility of exploring the advantages and disadvantages for different trust management models.

Keywords Grid systems, incremental trust, fuzzy logic, Bayesian statistics.

Resumé

Adgangskontrol er en vigtig forudsætning for at opnå computersikkerhed i et distribueret system. Et vigtigt aspekt af denne kontrol er styringen af tillid. Tillid beskriver i hvor høj grad et system er overbevist om at akkreditiverne for en handling følger en given sikkerhedspolitik. Dette er et komplekst problem i et grid-system eftersom det kan forventes at nye brugere med nye akkreditiver vil kræve adgang til systemet. Det kan ligeledes forventes at nye autoriteter vil udstede nye akkreditiver. Disse muligheder i et grid-system betyder at der er behov for et fleksibelt udtryk for tillid.

I dette M.Sc. projekt er målet at undersøge styringen af tillid i et system der beror på inkremental tillid (incremental trust). Ved incremental trust kan tilliden til akkreditiver stige eller falde. Denne ændring i tillid afhænger af om aktiviteterne er acceptable eller uacceptable fra et sikkerhedssynspunkt. Projektet involverer en undersøgelse af relevant litteratur der omhandler tillid og tillidsstyring. De efterfølgende definerede heuristikker leder til implementeringen af et simulationsprogram. Dette program undersøger anvendelsen af fuzzy logik og Bayesiansk statistik til styring af incremental trust i et grid-system. Den fleksible opbygning af simulationsprogrammet gør det muligt at undersøge fordele samt ulemper ved forskellige tillidstyringsmodeller.

Nøgleord Grid systemer, incremental trust, fuzzy logik, Bayesiansk statistik

Contents

List of Figures

Chapter 1

Introduction

The use of distributed systems for scientific computing has been subject to a growing interest in recent years. The term grid system describes the various mechanisms that allow massive number of distributed computers to collaborate. This results in immense computing power. The name grid suggests similarities with the electrical power grids and when using electricity we do not worry about where the electricity comes from or how it arrives to the socket. As long as we are able to plug appliances into the socket, we are able to use the power. Similarly, in order to use the distributed computing resources, the user does not need to know where the application ran, or which storage was used for running the applications.
The progress of grid technology is promising. The paper *Grids and grid technologies for wide-area distributed computing*[1] provides an overview of the efforts in grid software. For example the software technology created by the Globus Alliance (www.globus.org) provides a very popular grid development framework. The grid software combined with the high speed of Internet access becoming increasingly available enables more and more hardware resources from supercomputing centres, educational institutions and other organisations across the globe to collaborate as a grid system. However the distributed nature of grid systems dictate the need for a robust and flexible security scheme. With the possibility that hardware resources continuously join and leaves the grid system, the users who needs to run applications can't be sure of the quality of service provided by a hardware resource. Likewise the hardware resources might not previously have had contact with a given user and consequently can't be sure that running the user's application will not harm the resource. In such a scenario trust can be used as a factor for deciding whether or not communication should occur. How trust should be established in a distributed computing environment is not an easy task. In this thesis I will investigate the use of trust in a decentralised grid environment.

1.1 Motivation

The concept of trust is deceptively simple. Intuitively everyone has a general understanding of what constitutes trust. However as the chapter detailing background on trust and trust management will show, agreeing on a definition of trust and how to manage it is not straight forward.
The amount of literature involving trust indirectly reflects how desirable the use of trust in a computer security context is. In a grid system where resources are expected to join and eventually leave the system, a resource will often have to decide if communication should occur between itself and another resource whom it had little or no previous contact with. A notion of trust would prove extremely helpful for the resource in making such a decision. When the notion of trust has an incremental nature, a trust management model will further enable resources in making decisions in a dynamic environment such as a grid system.
In any computing environment the measurement of behaviour for the purpose of calculating trust is not simple. What constitutes a behaviour has to be investigated, with the intent of using real world quantifiable parameters for the measuring of behaviour. An objective of my project it therefore to use a quantifiable notion in order to describe behaviour leading to a notion of trust that can be managed.
Most trust management models investigate the efficiency of their model with the use of a simulation program. However, without the comparison of the different models in similar scenarios, advantages and disadvantages between the models remain hidden. Subjecting different models to similar scenarios is therefore an important part of my project.

1.2 Structure of the report

The report is structured as follows:

Chapter 1 Introduction

Chapter 2 contains descriptions of the various notions of trust and trust management models presented in related literature. Included is also introductions to fuzzy logic and Bayesian statistics.

Chapter 3 presents the security considerations for a grid system.

Chapter 4 presents the analysis, design, implementation and testing of the grid simulation program.

Chapter 5 presents the scenarios and results from the experiments with the simulation.

Chapter 6 Discussion and concluding remarks.

1.3 Contents of the appendices

The appendices contains the following:

Appendix A contains the user guide for the simulation program.

Appendix B explains the contents of the enclosed CD-ROM.

Appendix C contains a flow of control diagram of the simulation.

Appendix D contains the class diagrams for the simulation program.

Appendix E contains the API for the simulation program.

Appendix F lists the default fcl file.

Appendix G contains the configuration files for the experiments.

Appendix H contains additional plots from experiments.

Chapter 2

Background

The interest for trust in computer security has grown substantially. A google search on the subject returns millions of hits involving discussion forums, research models and commercial security systems. The focus areas are diverse, which has lead to several disagreeing definitions on trust, what is considered relevant when using the notion trust, and how such a notion should be evaluated.

The purpose of this chapter is to give an overview of the efforts. Initially the different trust concepts including reputation will be discussed. This is followed by a description of the computerised models and finally introductions to fuzzy logic and Bayesian statistics are given.

2.1 Trust

Intuitively the concept of trust seems straight forward. When somebody behaves badly they are not to be trusted and vice versa when they show good behaviour. In a real world scenario however, the distinction between good and bad is not always obvious. It is seldom the case that a given behaviour falls into the extremes "good" or "bad". Far more often an action or sequence of actions will be interpreted as a mere degree of the extremes. In this section trust is discussed and some of the considerations that are needed to evaluate trust is presented. The purpose is to underline some of the complexities that without careful considerations could be easily missed.

2.1.1 Analyzing trust

Agreeing on a definition on trust is not an easy task. Many people have tried but none have presented a definition that claimed wide spread acceptance. The most apparent reason lies in the fact that defining the elements of what constitutes trust is hard. Psychologists, sociologists and philosophers all contribute to the discussion of what consitute trust. In any

given scenario, the action of an entity[1] will result in an wast amount of information about its behaviour. What is the consequence of the action, how did other entities perceive this action and what actions did the entity perform in the past all these factors influence the trust in that particular entity.

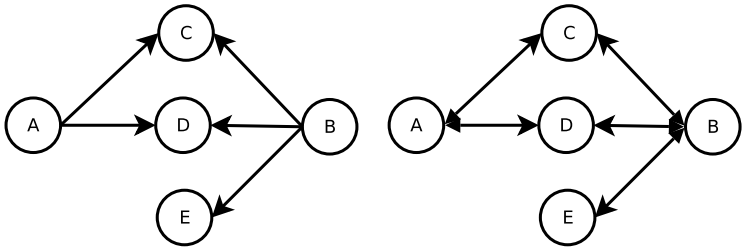
The initial consideration when discussing the use of trust in computer security must be whether or not trust is meaningful when it expresses a relationship between computers. Intuitively the requirements for an entity to be able to trust or distrust another entity requires human thinking capabilities. This is because trust, between humans, often is based upon an expectation of behaviour in a given situation. What is considered benevolent or malicious behaviour is decided by the person, *the trustor*, who is evaluating the trust. The person looking to be trusted, *the trustee*, might not even know what the trustor considers benevolent or malicious behaviour. This suggests that at least the trustor has to be a thinking entity[2]. Since computers do as they are told and can't be considered thinking entities that are able to distinguish between benevolent or malicious behaviour in every situation, a sceptic would say that trust between computers is meaningless. However in [3] Audun Jøsang argues that since computers are tools used by humans, and a distributed system can be considered an indirect social interaction, trust in this context is meaningful. A computer is neither benevolent nor malicious but the person controlling it might be. When discussing whether or not a computer is trustworthy what actually is implied is the belief that a computer will or will not resist malicious manipulation by a human. Similarly humans, often subconsciously, decide if another person is likely to be manipulated, or smart enough to resist manipulation, when the trustworthiness of that person needs to be assessed. When trying to define what constitutes benevolent and malicious behaviour the evaluation context is very important. Moral philosophers like Immanuel Kant(1724-1804) have tried offering a universal law. Kants universal law states that the difference between benevolent and malicious behaviour can be defined by the test:

> Act only on that maxim whereby thou canst at the time will that it should become a universal law[3].
>
> *Immanuel Kant(1724-1804)*

The test actually states that *we should not want to do something which we could not wish would be done by everyone*[3]. If an action fails this test then it is considered malicious. This approach is however to broad to be called upon in practise for defining what constitutes malicious behaviour in a computer security context. If it's assumed that Immanuel Kant is correct and humans subconsciously perform this test, it still requires that the entity is a thinking entity. Consequently the test is hardly applicable to computer security. Instead of considering the most general scenario, the selection of possible scenarios should be narrowed to those that frequently appear in distributed systems.

Imagine the situation depicted in figure 2.1(a). The letters A to E denote entities and the arrows express trust relationships. The arrow head denotes the direction of trust, for

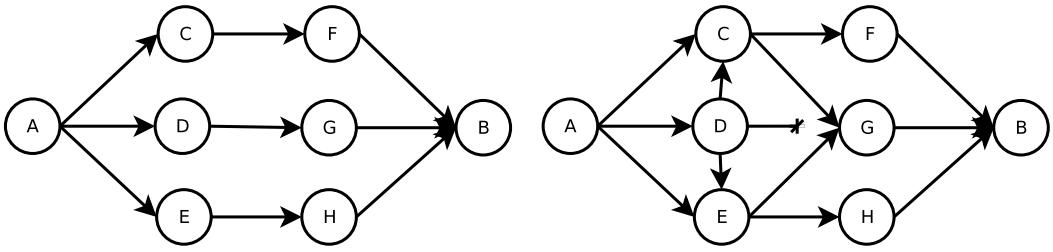
[1] The term entity will be used throughout the thesis to denote a human or a computer.



(a) One way (b) Two way

Figure 2.1: Trust situations

example entity A trusts C but the reverse might not be the case. In the scenario entity A trusts C and D, and entity B trusts C, D and E. When observing such a scenario some questions come to mind: Is there a trust relation from A to B or E? Would it make a difference if B or E trusted A? The answer must be no. Since A trusts C, D and E but C, D or E don't trust A there is no information that would support the notion that A has a trust relationship with B or E.
If the scenario is changed and more information is available, as depicted in figure 2.1(b), the situation is different. Here trust goes both ways. Since A trusts both C and D, and C and D trust B, A can now trust B to a certain degree and consequently also E. However now the *degree* of trust that A has in B and E must be considered. The foundation for the trust relationship, from A's point of view, lies in the *recommendations*, or *ratings*, of B given to A by C and D. The trust between A and B is also referred as a trust transitivity[4]. In order to determine the degree of trust the quality of these recommendations should be examined. This is illustrated in figure 2.2(a). Here, A has a degree of trust in B, through the



(a) (b)

Figure 2.2: Recommendation situations

recommendations from the three trusted parties. This however, does not mean that there is

a direct trust relationship to B, but rather that A is able to determine B's *reputation*[2]. The quality of the recommendations will vary depending on the context. If C, D and E are close business partners of A (and likewise for the next links) then it would be sound judgement for A to trust B. If on the other hand C, D and E are business competitors of A then their recommendations could be unreliable and the reputation of B has to be established via another link. The situation can be further complicated as illustrated in figure 2.2(b). Here D specifically distrusts G but C and E trust G. Additionally D trusts both C and E. Since there now is a difference of opinion about the reputation of B, A's point of view is affected. Entity A now needs to examine the reason for D's distrust of G, and whether or not the reason is good enough to discredit the recommendations given by C and E. As with behaviour there is no universal way of determining this. The situations are merely pointed out to illustrate that serious considerations are needed when designing a system that employs trust relationships.
In [5] by Audun Jøsang and Stéphane Lo Presti as well as in the SECURE[6] project the relationship between risk and trust is investigated. In both projects they argue that trust is *inherently linked to risk*[6]. The transaction context defines what level of risk that is associated with a possible transaction. Consequently the amount of trust needed should vary depending on the context in order to reflect the amount of risk involved for the trustor.

In a real world scenario thousands of entities would be communicating in different contexts, calling for even closer attention to details in each possible scenario.

2.1.2 Notions of trust

When trust is discussed in the context of computer security, it is often divided into the two categories identity trust and incremental trust[3]. Identity trust explores the trustworthiness of the identity credentials of an entity. A real world example of identity trust would be the passport verification in an airport. Here the passport is verified, meaning that the passport holder is the person stated in that passport and that the passport is genuine. In computer security the identity credentials can be a certificate or/and a set of public/private keys. The actual verification process uses cryptographic encryption/decryption and hash functions. Two commonly known identity trust schemes are X.509 certificates and Pretty Good Privacy (PGP). Both schemes will be described in section 2.2. Identity trust schemes can be considered as a binary mode of trust, in the sense that an entity will either pass or fail an identity verification.

Incremental trust focuses on expected behaviour of an entity. A real world example could be a case where a person asks permission to borrow a pen. If the person borrowed two pens last week without returning them it might be reasonable to assume that the pen in question wouldn't be returned either. If the person in question is a good friend or an infinite amount of pens are available, the pen might be given to the person anyway. If however the item in

[2] A reputation is the result of the combined recommendations. The reputation notion is treated in greater detail in section 2.3

[3] Incremental trust is also often referred to as adaptive trust or behavioural trust.

question was a 1000 $ or more, the same might not be the case. Most likely some form of assurance that the money would be returned would be needed. By considering the person' credit history and other available information it could be determined if the person is eligible for credit.
In computer security entities don't borrow pens or money but apply for access to computer systems. The situation however share some similarities. Given an entity's history the expectation of what would happen, and whether access should be granted, is calculated. The entity's history consists of a number of factors that describe the previous behaviour. Did the previous actions of the entity cause the computer system to crash repeatedly, any access, if given, should be fairly limited. The behavioural history of an entity is however seldom a list of actions that fall into extremes. More frequently it consists of a sequence of factors that show a degree of benevolent or malicious behaviour. Incremental trust views all the factors and try to calculate the expected future behaviour of the entity. It could be argued that the goal of incremental trust is to mimic the trust relationships that exist between humans. The sections 2.3 and 2.4 describe some of the models that focuses on incremental trust. It should be noted that incremental trust unlike identity trust avoids the use of encryption, hash functions and signatures and unless directly stated the identity of the entity is undisputed in incremental trust schemes.

2.2 Identity trust

Identity trust schemes detail solutions for mapping a name to a certificate or a public key. The goal is to enable the users of the system to have a high degree of confidence in the identity of others. This is made possible through the use of encryption, cryptographic hash function and digital signatures. I assume the reader has a fundamental knowledge of these concepts otherwise *Security in Computing*[7, Chap. 2] contains an excellent introduction.

2.2.1 X.509

X.509 is a widely used standard for defining certificates originally published as a part of the X.500 directory recommendations. This was designed to be a global and distributed directory service.
An X.509 certificate contains information about the person or organisation to which it belongs, the public key, information about the certificate issuer, the time limits where this certificate can be used and the operations for which it can be used. An X.509 certificate consists of the fields shown in table 2.1. To issue certificates a Certification Authority (CA) is needed. Every CA has a public key certificate to identify it self and it is signed by a CA with a higher level of authority. CA's are organised into a hierarchical tree of certification. The original intent was to reflect the design of the X.500 directory service. This means that there exists a single, global tree of authority, whose root is the most trusted authority of all. Processing the validity of a certificate basically consists of validating the path from the subject issuer until a trusted CA or the root CA is found. This is illustrated in figure

Version
Serial number
Signature algorithm
Issuer name
Validity period
Subject (user) name
Subject public key information
Issuer unique identifier
Subject unique identifier
Extensions
Signature algorithm
Signature of the above fields

Table 2.1: X.509 certificate

2.3. Each dot represents a CA and each arrow represents a path between the CAs. Apart

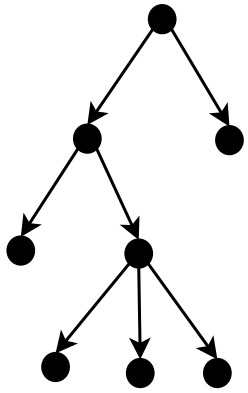


Figure 2.3: CA hierarchy.

from issuing certificates and verifying certificate paths, CAs also maintain revocation lists. A revocation list contain certificates that are not valid anymore. A certificate gets invalid if it is compromised.
With the use of various certificate extensions and trust models between CAs efficient and robust verification of names to public key binding can be achieved.

2.2.2 PGP

Pretty Good Privacy (PGP) is a software application designed by Philip R. Zimmermann[4] to allow secure exchange of files and messages with guaranteed confidentiality, integrity,

[4]www.philzimmermann.com

authentication and to some extent, non-repudiation. PGP is based on both public-key and symmetric-key cryptography. For confidentiality, PGP randomly generates a session key to use for symmetric encryption of a message. The session key is encrypted using the recipient' public key. The encrypted message and the encrypted session key is sent to the recipient. For authentication, a hash of the message is computed and digitally signed using the sender' private key. This is sent to the recipient who can verify the origin of the message using the sender' public key. It is possible to achieve both confidentiality and authentication when the two methods are combined.
Each user is identified by a unique name that usually is the e-mail address. PGP supports a web of trust model, where no central authority or hierarchy of authorities for certifying public keys is needed. Instead, a name to public key binding is done by trusted introducers who vouch for the binding by digitally signing it. In practise a user may make any trusted user an introducer. The underlying theory of this model is that everyone builds up their social network of trust by judging recommendations from people that they know. Each PGP user maintains multiple public and private keyrings. A private keyring stores the key pairs owned by the user, while the public key ring stores the known public key bindings. Multiple public key rings could be maintained to divide up friends/business partners etc. Additionally each public key binding is associated with a level of trust. In PGP there are four levels: "unknown", "untrusted", "marginally trusted" and "fully trusted". These levels are intended to reflect the trustworthiness of a public key owner as an introducer of other public key bindings. In PGP, a public key binding is permanent unless the keyring maintainer explicitly removes it. It should be noted that the concept of trustworthiness of public key bindings is not a part of identity trust, but should be viewed as a way of managing trust. The concept of managing trust is further described in section 2.4.

2.3 Reputation

The ability to learn from other peoples' experiences is important when considering security schemes based on trust. Trust and reputation are however not the same thing. Audun Jøsang defines reputation as *what is generally said or believed about a person's or thing's character or standings*[8]. Intuitively the distinction between reputation and trust seems clear as it's possible to trust somebody because of their good reputation or despite their bad reputation.
Reputation systems concentrate on the propagation of one's own experience to others and the gathering of information. By asking one's surroundings for recommendations or ratings an entity's chances for not being fooled by a potentially misbehaving entity increases. At the same time the entity must ensure that recommendations are accurate, meaning that the surroundings aren't colluding. In this section I will describe two reputation models.

2.3.1 TRUMMAR

TRUMMAR(TRUst Model for Mobile Agents based on Reputation), is presented in the paper *TRUMMAR - A Trust Model for Mobile Agent Systems Based on Reputation*[9] by Ghada Derbas. It is a discrete reputation based trust model that should enable agents to protect them selves against malicious hosts. The model includes notions for concepts such as prior-derived reputation, first impression, reputation decay with time, hierarchy of host systems(neighbours, friends and strangers) and the inclusion of interaction results in reputation calculations. The computational model is defined by the following equation, where x, y, i denotes entities:

$$\Omega(x,y) = \alpha\ \Omega(x,y)_{previous} + \underbrace{\beta\frac{\sum_i \Omega(x,i)\Omega(i,y)}{\sum_i \Omega(x,i)}}_{\text{neighbours}} + \dots$$

$\Omega(x,y)$ denotes the reputation value of y from x's point of view. When calculating a new reputation value, x's previous reputation value of y has the weight α. This is added to the weighted average of a given hierarchy level, here the neighbours, reputation rating of y. The neighbours have the weight β. $\Omega(x,i)$ expresses the reputation of i from x's point of view and $\Omega(i,y)$ expresses the reputation of y from i's point of view. The effect is that x's trust in i influences x's opinion on the rating of y given by i. Other hierarchy levels could be included with weights of γ for friends and δ for strangers. The model documentation suggests defining the weighting factors such that $\alpha > \beta > \gamma > \delta$ and $\alpha + \beta + \gamma + \delta = 1$
The TRUMMAR model also supports the decay of reputation information with time. The change is exponential according to $e^{(t-t_0)/\tau}$ where τ is defined a an empirical constant and t denotes time. This means that a bad host is not considered bad forever and a good host is not considered good forever. Based on reputation thresholds an agent can determine whether or not a given reputation value for an agents is sufficient to deem that agent trustworthy.
The model has been simulated and results are presented that support the claims from the paper. The loss of reputation information over time is particularly noticeable in the simulation.

2.3.2 Eigentrust

Eigentrust was presented by Sepandar D. Kamvar, Mario T. Schlosser and Hector Garcia-Molina in the paper *The Eigentrust algorithm for reputation management in P2P networks*[10]. As the title of the paper indicates, Eigentrust is an algorithm for reputation propagation and management in a peer-to-peer system.
The authors argue that a transaction between peers can be viewed as satisfactory or unsatisfactory. A peer i will for example rate a transaction with another peer j as satisfactory or unsatisfactory and s_{ij} denotes the number of satisfactory minus the number of unsatisfactory transactions. Eigentrust provides a distributed algorithm for propagating peer i's local trust information s_{ij} to other peers, and the pseudo code for the algorithm is written

in the paper[10].
In Eigentrust the local trust value has to be normalised to c_{ij}. The authors stipulate that there are some potential drawbacks when normalising the local trust values, but when simulating the algorithm none of the potential drawbacks revealed problems. As with other reputation models, trust values are obtained by asking acquaintances and weighing their rating by the trustworthiness of the rater. For example peer i's trust value for peer k would be computed by asking the acquaintances j about peer k. Formally written as:

$$t_{ik} = \sum_j c_{ij} c_{jk}$$

In a matrix notation where $C = [c_{ij}]$ and $\vec{t_i}$ is a vector containing t_{ik} values, the vector containing all of peer i's trust values is $\vec{t_i} = C^T \vec{c_i}$. When peer i also asks peer k's acquaintances then $\vec{t_i} = (C^T)^2 \vec{c_i}$. Continuing in this manner defines the trust vector of peer i as $\vec{t_i} = (C^T)^n \vec{c_i}$ which would enable peer i to have a complete trust view of the system. When n is large many iterations are needed, but it will converge to a single global trust vector. The Eigentrust algorithm exploits this by dividing the network into groups. The groups consist of a number of peers, called score managers in charge of computing the trust values for a selection of peers. Which peers that belong to which groups of score managers is computed with a secure hash function. This ensures that malicious peers can't join a particular group of score managers and report false trust values. Of course a peer can't belong to the group of score managers that calculate it's own trust value. When a peer needs the trust value for another peer it will query all of the score managers. The peer-to-peer system has to contain a collection of pre-trusted peers. For example the creators of the network. These peers function as a "seed" for the trust system.
Using the Eigentrust algorithm it is possible to allow peer anonymity and it is very robust when handling malicious peers, or collectives of peers. The authors' simulation results also reveal that the trust information converges very fast.

2.3.3 Beta reputation system

Audun Jøsang and Raslan Ismail have presented an interesting approach for a reputation system with *the beta reputation system*[11]. It is targeted towards e-commerce, but is applicable in other areas as well. It is based on the beta probability density function and even though the paper describes reputation propagation in a centralised approach the system can be used in a decentralised environment as well. The model is based on theory from Bayesian statistics which is described in greater detail in section 2.6.
The beta probability density function can be used to represent probability distributions of binary events. It is a continuous function indexed by two parameters α and β. Using the gamma function ($\Gamma(x) = (x-1)!$) it has the following expression:

$$f(p \mid \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1} \quad \text{where } 0 \leq p \leq 1 \text{ and } \alpha, \beta > 0$$

When restriction $p \neq 0$ if $\alpha < 1$, and $p \neq 1$ if $\beta < 1$ the probability expectation value will be:

$$E(p) = \alpha/(\alpha + \beta)$$

The probability expectation value is the mean of the distribution and can serve as measurement for the reputation of an entity from a given beta probability density function.
An example application of the beta reputation system could be as follows: If there are two possible outcomes $(x, \bar{x})$ for a process and r is the number of outcomes of type x and s is number of outcomes type $\bar{x}$, the probability density function of observing outcome x in the future can be expressed by setting $\alpha = r + 1$ and $\beta = s + 1$ where $r, s \leq 0$. If for example x was observed 7 times and $\bar{x}$ was observed once, the beta function expression is $f(p \mid 8, 2)$. A plot of this function is shown in figure 2.4: The probability expectation is

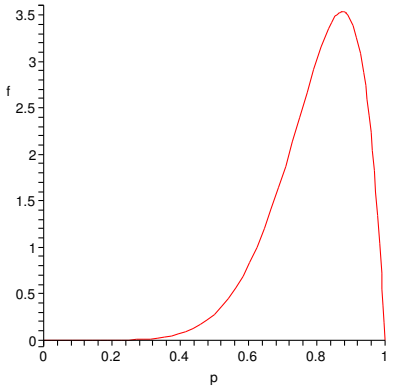


Figure 2.4: The beta function

given by $E(p) = 0.8$. This indicates that the relative frequency of outcome x is somewhat uncertain, but the most likely value is 0.8. This means that on a [0,1] scale the reputation of that particular entity would have a 0.8 value. Or in other words there is an 80% probability next outcome being of type x.
Instead of using a reputation value in the [0,1] interval it might be more suitable to have a reputation rating be in the interval of [-1,1] where 0 is a neutral rating. The probability expectation value can easily be scaled. If entity X provided positive (r) and negative (s) feedback about entity T, the reputation rating would be:

$$Rep(r_T^X, s_T^X) = \frac{r_T^X - s_T^X}{r_T^X + s_T^X + 2}$$

This formula gives entity Z the ability to ask entity X what it thinks about entity T. If Z decides to ask another entity, say Y about T the feedback can be combined by:

$$r_T^{XY} = r_T^X + r_T^Y \quad \text{and} \quad s_T^{XY} = s_T^X + s_T^Y$$

By simply adding the positive and negative feedback it is possible to combine feedback from multiple entities.
Time often plays an important role when judging an entities reputation. Old feedback may not always be relevant for the actual reputation rating, because an entity may change its behaviour over time. A way of giving less(or no) weight to old feedback is by a forgetting factor λ. If a number of entities has provided a sequence Q containing n feedback tuples $r^Q_{T,i}, s^Q_{T,i}$ indexed i about an entity T the combined feedback is:

$$r^Q_T = \sum_{i=1}^{n} r^Q_{T,i} \quad \text{and} \quad s^Q_T = \sum_{i=1}^{n} s^Q_{T,i}$$

With the forgetting factor λ being $0 \leq \lambda \leq 1$:

$$r^Q_T = \sum_{i=1}^{n} r^Q_{T,i}\lambda^{n-i} \quad \text{and} \quad s^Q_T = \sum_{i=1}^{n} s^Q_{T,i}\lambda^{n-i}$$

If λ has the value 1 it is the same as no forgetting factor. If the value is 0, all but the most recent value is forgotten. To avoid keeping all feedback forever, a recursive more space efficient algorithm is derived.
Most systems that need a reputation system are not focused on binary events. Getting entities to provide feedback saying "entity T delivered what it promised 7 times but lied once" is not going to suitable in a real world system. Instead it would be preferred that entities are able to provide feedback as a single value that can be interpreted as the degree of satisfaction when dealing with the entity in question. A single value in the interval [0,100] or [-1,1] would be desirable. Additionally when seeking reputation information about an entity the option of weighting the feedback differently is preferred. If a friend provided feedback this should have a higher weight then say a complete stranger. Both of these things can be accomplished by:

$$r = w(1+v)/2 \quad \text{and} \quad s = w(1-v)/2$$

The above formula gives the ability to calculate r and s values when feedback is provided with a single value v in the interval [-1,1]. w is the weighing factor that can be used to weight feedback differently.
Simulation of the beta reputation system in a e-Market has been done in the paper *Simulating the Effect of Reputation Systems on E-markets*[12]. This showed promising results for fostering good behaviour in a e-Market. The follow-up paper *Filtering out unfair ratings in bayesian reputation systems*[13] from 2004 extended the simulation with an algorithm for filtering out unfair ratings in the system.

2.4 Trust management

With the assumption that trust is measurable and quantifiable, the term trust management is assigned to describe methods for controlling trust and the ability to make decisions based

on trust. The trust management application area, be it peer-to-peer, e-commerce, mobile agents or grid computing systems, influences the requirement to a great extend. This has lead to different interpretations of what constitutes trust management. In this section I will give an overview of some of the trust management schemes available. Even though the schemes are targeted towards different application areas, all of them use a decentralised approach to trust management.

2.4.1 Logic based systems

Matt Blaze, Joan Feigenbaum and Jack Lacy presented the first trust management solution. Their paper *Decentralized trust management*[14] presented a novel approach to computer security. Traditionally system security was viewed as processing a signed request that consisted of a task of authentication and access control. This meant that the receiving system first had to determine who signed the request, then decide if that person was allowed access to the required resource so the requested action could be performed. The authors considered this approach to be flawed in a distributed system.
Instead they proposed to evaluate whether of not the key that signed a request is authorised to perform the action stated in the request. Or as stated in the paper: *Does the set C of credentials prove that the request r complies with the local security policy P?*The answer will then be given by the trust management engine that is a separated system which processes (r, C, P) inputs. The trust management engine, named PolicyMaker, is in charge of the trust management. The idea is illustrated in figure 2.5, where PKI denotes a public key infrastructure [7, p. 436]. It's important to notice that the trust management engine appears

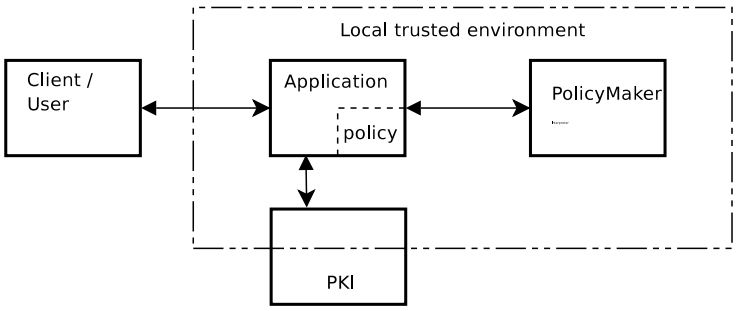


Figure 2.5: The PolicyMaker model.

very much like a database query engine. It can be considered a general-purpose, application-independent algorithm for checking proofs relevant to the calling application. The credential signature verification is not part of the trust management engine. The program contacting PolicyMaker is supposed to verify the signatures. Additionally PolicyMaker is not expected to discover that for example the credential set C only needs one more credential to complete the proof. It only answers the question with the given information in the form of a proof (or lack of a proof). The questions to the trust management engine is formulated as queries of the form:

$key_1, key_2, .., key_n$ **REQUESTS** *ActionString*

The action strings are only interpreted by the calling application. The semantics of the action strings does not have to be known by PolicyMaker. The queries are processed by PolicyMaker based on trust information contained in assertions. An assertion is a policy or a credential and have the form:

Source **ASSERTS** *AuthorityStruct* **Where** *Filter*

The *Source* indicates the source of the assertion, the *AuthorityStruct* specifies the public key(s) to whom the assertion applies and the *Filter* is the predicate that the action string must satisfy on order for the assertion to hold. With a given query PolicyMaker can, based on the defined assertions, provide a proof that the request complies with the specified policies (or fail to produce a proof). The proofs will be presented in acceptance records of the form: assertion number i, issued by source s_i, approves action string R_{ij}
The language AWK was developed for assertions but PolicyMaker does not dictate that this particular language must be used. PolicyMaker is flexible when it comes to choosing the assertion language. However, assertions are monotonic which means certain types of policies used in practise will be excluded. An effect of this is for example that the use of negative credentials such as revocation lists is prohibited. The authors argue that the monotonic restriction encourages a conservative and prudent approach to security since it in practise means that no potentially dangerous action is allowed by default. Instead of using a revocation list an entity would be required to present a certificate of non-revocation.

Lessons learned from PolicyMaker inspired the same authors to develop the trust management system KeyNote[15]. It was designed to include the wishes for standardisation and easier integration into applications. KeyNote uses a similar approach as PolicyMaker but assigns more responsibility to the trust management engine and the assertion language has been fixed. The main differences between KeyNote and PolicyMaker is listed below:

1. Credential signature verification is done by KeyNote
2. The assertion language is a human-readable notation based on C-like expressions and regular expressions.
3. Assertions in processed by KeyNote always returns a boolean (authorized or not) answer.

The KeyNote model only differs in a single place from the PolicyMaker model illustrated in figure 2.5. In KeyNote the communication with the PKI is not done by the application. Instead KeyNote will communicate with the PKI and check the signatures.
PolicyMaker uses, but is not fixed to, the language AWK. This language was criticised for being too heavyweight for most applications[16]. One of the complaints was that in a language that permits loops and recursion, like AWK, it is hard to enforce resource-usage restrictions. The purposed C-like notation for KeyNote is human-readable, without loops

and dynamic memory allocation and the format also resembles that of e-mail headers. The restrictions of no negative credentials used in PolicyMaker still applies in KeyNote and the trust management engine still does not enforce the policy. It only advises the calling application. The KeyNote system has been implemented in software packages like Apache-SSL *(www.apache-ssl.org)* and OpenBSD *(www.openbsd.org)*.
PolicyMaker and KeyNote have influenced the development of many security models that are based on trust.

2.4.2 Discrete based systems

Farag Azzedin and Muthucumaru Maheswaran have made extensive research in trust management in public resource grids. In public resource grids the resources does not have a prior trust relationship. They target their research towards trust driven public resource grid where the membership of the grid is as open as a peer-to-peer file sharing system. They propose using a discrete trust model for evaluating trustworthiness. In their papers *Towards Trust-Aware Resource Management in Grid Computing Systems*[17], *Integrating Trust into Grid Resource Management Systems*[18] and *Evolving and Managing Trust in Grid Computing Systems*[19] they define the notion of trust by a trust level (Γ) that is composed of past direct experiences, referred to as direct trust, and reputation in a given context[5]. The trust level between entity x and y at time t in a context c is defined by the following function:

$$\Gamma(x, y, t, c) = \alpha * \Theta(x, y, t, c) + \beta * \Omega(y, t, c)$$

The function Θ expresses direct trust and the function Ω expresses reputation. α and β are used to weigh the two functions. The direct trust function Θ is defined by:

$$\Theta(x, y, t, c) = DTT(x, y, c) * \Upsilon(t - t_{xy}, c)$$

DTT refers to a table of direct trust. This contains the previous trust levels. The value is multiplied by the decay function Υ that determines how fast trust will deteriorate. The argument is that if entity x trusted y a certain degree 5 years ago then the trust level would not be the same today. The reputation Ω is defined by:

$$\Omega(y, t, c) = \frac{\sum_{k=1}^{n} RTT(z, y, c) * R(z, y) * \Upsilon(t - t_{xy}, c)}{\sum_{k=1}^{n}(n)} \qquad \forall z \neq x$$

RTT is a reputation trust table. The reputation is the average of the product gathered from the n surrounding entities. It is multiplied with a recommender trust factor R. This will weigh the recommendations differently so collusions among groups of entities can be avoided. R is in the interval $[0, 1]$ and a higher value indicate that the recommender does not have a alliance with the target entity(y). Again the recommendations are multiplied with the decay function Υ. Naturally the entity looking for recommendations (x) is not included in the gathering of reputation values. It is worth noticing that DTT and RTT are essentially the same. This means that entities submit the direct trust value when an entity

[5] The notion of different contexts was added in [18, 19].

asks for a recommendation.
When the communication between two entities is being established the two entities will each compute the trust level for the other. This trust level is the offered trust level (OTL) and it falls into a category between A and E. A denotes *very low trust level* and E denotes *very high trust level*. Both entities should have a required trust level (RTL) for other entities that is between A and F, where F denotes *extremely high trust level*. F is not provided in any existing trust relationships. The reason for having a trust level F is that entities can enforce enhanced security by increasing their RTL to F. The expected trust supplement (ETS) is RTS-OTS. If ETS>0 for both entities the activity can proceed with no additional overhead. Otherwise there will be additional security overhead involved in supplementing the offered trust level to meet the requirements. Table 2.2 lists the supplement values for different RTL and OTL values. The table shows how much an entity should increase is

Requested Γ	Offered Γ				
	A	B	C	D	E
A	0	0	0	0	0
B	B-A	0	0	0	0
C	C-A	C-B	0	0	0
D	D-A	D-B	D-C	0	0
E	E-A	E-B	E-C	E-D	0
F	F	F	F	F	F

Table 2.2: Expected trust supplement values

offered trust level to meet the requirements.
To increase the scalability of the trust model Farag Azzedin and Muthucumaru Maheswaran divide the grid system into grid domains. A grid domain is further divided into resource domains and client domains. Each domain's trust agent will maintain a trust table similar to the table in 2.3. An entity can compute the offered trust level for a given entity for a

Client Domains	Resource Domains			
	...	RD_j		
	...	TL_j		
	...	A_1	...	A_k
CD_1	...	TL_{1j}^1	...	TL_{1j}^k
$\vdots$	$\vdots$	$\vdots$		
CD_i	...	TL_{ij}^1	...	TL_{ij}^k

Table 2.3: Example trust level table

given activity A_k by querying the table.
Using the model described here trust-aware resource management algorithms is presented and performance simulations are done.

With the paper *A Trust Brokering System and Its Application to Resource Management in Public-Resource Grids*[20] from 2004 Farag Azzedin and Muthucumaru Maheswaran expanded the model with the use of brokers. Furthermore they included notions of honesty and accuracy when calculating the reputation value and excluded the decay function Υ. The honesty of recommender z as observed by entity x is denoted by $H(x, z)$. If the difference between the highest and lowest received recommendation value is larger than constant ϵ, the honesty value is 0. If not the honesty value is 1. The accuracy of a recommender z as observed by entity x for a specific context c at a given time t is denoted as $A(x, z, t, c)$. The accuracy is a value in the interval $[0, 1]$. If an entity continually sends recommendations that differ significantly from what is otherwise observed the accuracy of the recommender decreases. The recommendation of an entity y given by z to an entity k for context c at time t is denoted by $RE_k(z, y, t, c)$. Before an entity x can use entity z's recommendation of y ($RE_x(z, y, t, c)$) it most be adjusted to reflect the accuracy of z. This adjustment is done with a shift function S. When $|R|$ is the number of recommenders, the reputation value calculated is by:

$$\Omega(y, t, c) = \frac{\sum_{k \in R}^{n} S(A(x, z, t, c), RE_x(z, y, t, c))}{|R|} \qquad \forall z \neq y$$

With the use of the accuracy concept it is possible to punish the entities that continually give bad recommendations. The direct trust function Θ is unchanged, except for the omission of the decay function Υ. As previously the direct trust and reputation values are weighed by the factors α and β to determine the trust value Γ. Farag Azzedin and Muthucumaru Maheswaran have decided to use the notion of brokers. These brokers represent a portion of the grid. A broker is responsible for managing the trust of the entities that are within its domain. The entities within a single domain are grouped into different classes by their expected reputation by the broker. A broker's reputation will depend on how accurate and honest it's representing the reputations of its entities. If the broker represents the entities badly the honesty and accuracy of that broker will decrease hence punishing the entire domain that it manages. This model has also been subjected to performance simulations.

2.4.3 Bayesian based systems

Since the properties of Bayesian statistics can be used for expressing subjective probabilities Bayesian statistics is very well suited for trust management. An introduction to Bayesian statistics is given in section 2.6.

Yao Wang and Julita Vassileva

Yao Wang and Julita Vassileva have presented in the paper *Bayesian Network-Based Trust Model*[21] a decentralised Bayesian network based trust model for a peer-to-peer system. In their scenario the peers act as file providers and users of shared files. Peers will develop two different notions of trust, the trust in file providers' competence in providing files and the trust in other peers' reliability in making recommendations. This means their notion of trust is based upon a a trust relationship from direct trust and reputation information. They have an important model assumption that states that all peers are truthful when they give

recommendations. It's however stressed that peers have different ways of evaluating other peers since evaluation of performance is based on subjective preferences. The consequence is that what is viewed as satisfactory performance by one peer, might not be considered satisfactory by another and vice versa.
Direct trust is founded on the three factors: download speed, file quality and file type. Download speed and file quality are divided into the categories high, medium and low and the possible file types are music, movie, document, image and software. To represent the trust between peers a naive Bayesian network is used. Such a network is composed of a root node and several leaf nodes that are assumed to have a strong probabilistic independence [22]. Each peer has a root node associated with every other known peer. A root node is associated with two values, satisfying and unsatisfying, denoted by 1 and 0, respectively. For a root node T, $p(T = 1)$ represents the overall trust in that peer. This is measured as the total number of satisfying interactions divided by the total number of interactions. $p(T = 0)$ is the percentage of unsatisfying interactions. This means that $p(T = 1) + p(T = 0) = 1$. Each leaf under that root node represents the peers' different capabilities as shown in figure 2.6. Associated with each leaf is a conditional probability table. For example the file

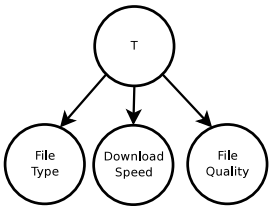


Figure 2.6: Naive Bayesian network.

type (FT) leaf for node T contains a table as shown in table 2.4. These probabilities are

	T=1	T=0
Music	$p(FT = Music\|T = 1)$	$p(FT = Music\|T = 0)$
...	...	...

Table 2.4: Conditional probability table for file type

calculated similarly to calculation of the overall satisfaction. For example the probability of satisfying interactions involving music files is calculated by the following formula where n denotes total number of interactions involving music files and $m1$ denotes the number that was satisfying:

$$p(FT = Music|T = 1) = \frac{p(FT=Music,T=1)}{p(T=1)}$$
$$p(FT = Music, T = 1) = \frac{m1}{n}$$

By combining the various tables, the probability of a peer being trustworthy in for example providing high speed, high quality music files can be computed. This of course requires that

peers update their Bayesian networks after each interaction.
An interaction is evaluated approximately like the discrete based models. For example using download speed s_{ds} and file quality s_{fq}, the level of satisfaction s is calculated by $s = w_{ds} * s_{ds} + w_{fg} * s_{fq}$ where w denotes a weight. If s is below a satisfaction threshold, the interaction is concluded to be unsatisfactory. Calculation of reputation values are also like the discrete models since it consists of summations of weighed averages.
The model also include notion of "gossip". When peers are idle they can exchange and compare Bayesian networks with surrounding peers. The authors have made a simulation of a peer-to-peer network that implemented the model which showed only slight improvements compared to a non Bayesian network system. The authors claim that the disappointing results are due to the simplistic simulation and suggest that better results might be obtained with a more complex scenario.

Lik Mui, Mojdeh Mohtashemi and Ari Halberstadt
With the focus on a network of agents in e-businesses, Lik Mui, Mojdeh Mohtashemi and Ari Halberstadt have introduced a model that uses a notion of reciprocity in the paper *A Computational Model of Trust and Reputation for E-businesse*[23]. Formally reciprocity is defined as the mutual exchange of deeds (such as favour or revenge). They also use the notions trust and reputation where reputation is defined as the perception an agent creates through past actions about its, intentions and norms, and trust is the subjective expectation an agent has about another's future behaviour based on the history of their encounters. Figure 2.7 illustrates the relationship between these notions. The arguments

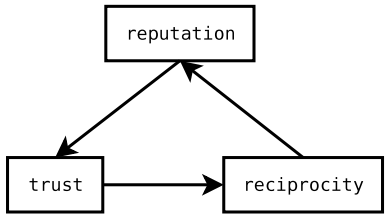


Figure 2.7: Trust, reputation and reciprocity relationship.

for this relationship is:

- An increase in an agent x's reputation in a social network should also increase the trust from other agents for x.
- An increase in an agent y's trust of x should also increase the likelihood that y will reciprocate positively to x's actions.
- An increase in an agent x's reciprocating actions to other agents in his embedded social network should also increase agent x's reputation in the social network.

The model has some theoretical similarities with the beta reputation system presented in 2.3.3, since it uses a beta distribution and the probability expectation value to measure the

probability that an agent will cooperate in a given context. The authors do however make two simplifications. First it is assumed that no agents will join or leave the network. Secondly possible actions are restricted to be binary, meaning that an agent can only cooperate or not cooperate.
For two agents x and y the reputation of y in the eyes of x in a context c is denoted $\Omega_{xy}(c)$. It is assumed that x always performs cooperate actions and x is assessing y's tendency to reciprocate cooperative actions. When it is assumed that both agents communicate within the same context, c is omitted, and the ith encounter between x and y is denoted $q_{xy}(i)$. Subsequently the history of encounters is defined by $H_{xy} = \{q_{xy}(1), .., q_{xy}(n)\}$. With n encounters and p as the number of cooperations, Ω_{xy} can then be estimated by $\hat{\Omega}_{xy} = p/n$. Using a beta distribution and omitting subscripts the posterior estimate becomes $p(\hat{\Omega}|H) = Beta(\alpha + p, \beta + n - p)$. Trust Γ is then described by :

$$\Gamma = E[\hat{\Omega}|H] = \frac{\alpha + p}{\alpha + \beta + n}$$

The formula expresses an agent x's estimate of the probability that agent y will cooperate given the history H_{xy} of encounters.

A theorem provides bounds on the parameter estimate $\hat{\Omega}$, and α and β are set to be 1 if the agents are complete strangers with no mutually known friends. Furthermore a simple scheme for propagation of reputation is presented. Some might argue that a separate notion for reciprocity is not needed since the general understanding of trust already contains such a notion.

2.4.4 Fuzzy based systems

Fuzzy logic has for a long time been used in control systems where a certain amount of uncertainty was present. A very interesting approach to trust management has been developed to exploit the decision making capability on input that has some degree of uncertainty. An introduction to fuzzy logic is given in section 2.5.

Daniel W. Manchala introduced the use of fuzzy verification in the paper *Trust Metrics, Models and Protocols for Electronic Commerce Transactions*[24]. It describes a model that use trust metrics and a notion of quantifiable trust to determine risks on transactions in a e-commerce environment. Since the focus of the system is on issues in e-commerce the trust variables include amongst others purchase cost, transaction history and purchase frequency. The model provides a way for determining what level of transaction verification is needed in each particular case. It is intended to be used by the online shops to protect them selves against customer fraud. This involves defining a trust zone in a trust matrix as illustrated in figure 2.8. This matrix could also be extended to include a third dimension denoting the transaction frequency of the customer. If a transaction lies outside the trust zone then every transaction has to be verified. This is denoted by V. If the transaction falls within the trusted zone then none or a smaller number of transactions have to be verified. The unique feature of the model is that instead of making a static scheme where for example every 30th

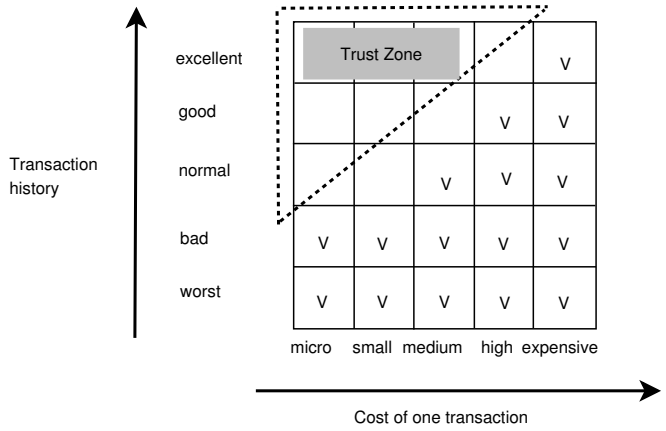


Figure 2.8: Trust matrix.

micro transaction of a customer with excellent history is verified, a more versatile scheme is presented. This involves the use of fuzzy logic. The static definitions within the trusted zone can be replaced by fuzzy membership functions that through the use of linguistic terms allows a more easy interpretation. In practise it means that instead of verifying every 30th micro transaction of a customer with an excellent history the system allows a "small" amount of transactions to be verified. Since the linguistic term “small” can be used in fuzzy logic a more flexible verification scheme is obtained.
This scheme can of course be extended to include the ability for customers to make similar trust matrices about the vendors. Furthermore a propagation mechanism can be used to merge several trust matrices to form an overall trust relationship.

Kai Hwang and Shanshan Song
There are many differences between e-commerce and grid systems. The papers *Trusted Grid Computing with Security Assurance and Resource Optimization*[25] and *Fuzzy Trust Integration for Security Enforcement in Grid Computing*[26] by Kai Hwang and Shanshan Song present a trust model for securing grid resources where grid security is enforced through trust update, propagation, and integration across sites. Their scheme is part of the GridSec (*Grid Security*) project at University of Southern California. Figure 2.9 illustrates the security infrastructure in GridSec. The security at each site is monitored and coordinated by a security manager. All the security managers of the grid work together to enforce a central security. The trust managers communicate through encrypted channels in a Virtual Private Network (VPN). This means that there is no need for a PKI or a certificate authority in the GridSec project.
The authors argue that since trust often is expressed by linguistics terms rather then numerically, fuzzy logic is very suitable in the field of trust management. Fuzzy inference is capable of quantifying the imprecision or uncertainty involved in trust measurement. Additionally it is very flexible because different membership functions and inference rules could be developed for different grid applications without changing the underlying structure of

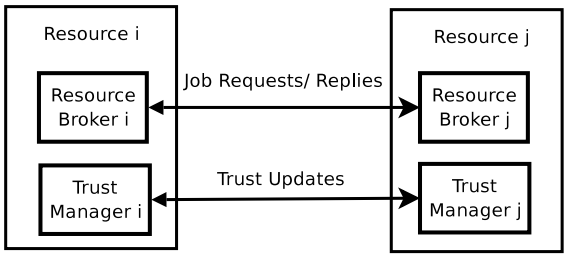


Figure 2.9: The GridSec project.

the system. Kai Hwang and Shanshan Song have chosen to focus the trust assessment on dependability, security, reliability and performance which means that the trust index (Γ) is determined by two factors: the job success rate (Φ) and self-defence capability (Δ). It should be noted that one can quantify the factors in practise. Job success rate is simply a matter of observing how many jobs succeeded and the self-defence capability can for example be supplied by an intrusion detection system. Various membership functions and fuzzy inference rules can then be defined to induce the trust index. The fuzzy inference is a process composed of the following five simple steps:

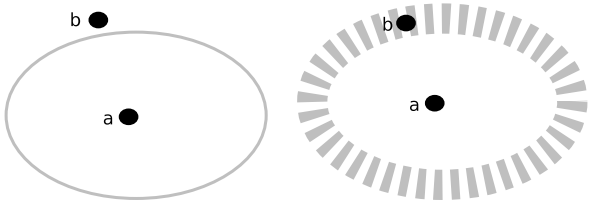
1. The values of Φ and Δ are determined.
2. The membership functions are used to determine the membership degrees for Φ and Δ.
3. The fuzzy rules are used to map the input space (Φ - Δ) onto the output space (Γ).
4. The output from each rule is aggregated.
5. The trust index is derived through a defuzzification process.

Each resource then maintains a trust vector that contains the trust indexes of the other known resources. The trust index is updated periodically with the site operations and propagated to the other sites. The paper [26] defines a trust update and a trust propagation algorithm. Furthermore a scheduler algorithm is introduced to optimise the resource allocation. The simulation results of this scheduler with the fuzzy-logic model showed promising results in Kai Hwang and Shanshan Songs tests.

2.5 Fuzzy logic

In the previous section two models were described that used fuzzy logic to determine a notion of trust. This approach shows various advantages and for this reason I will describe fuzzy logic in greater detail. I assume that the reader is familiar with conventional boolean (true/false) logic but not multivalued logic.

Fuzzy logic was initiated in 1965 by Lotfi A. Zadeh, professor in computer science at the University of California in Berkeley. It can be interpreted as a superset of conventional logic in the sense that it can handle concepts of partial truth. This allows a more human-like way of thinking and notions like "rather tall" or "very fast" to be formulated mathematically processed by computers. This ability is provided with the use of fuzzy sets differs from the conventional crisp sets. In a crisp set the distinction between being in the set or not being



(a) Crisp set (b) Fuzzy set

Figure 2.10: A crisp and a fuzzy set

in the set is clear. In figure 2.10(a) the value a is clearly in the set while the value b is clearly not in the set. In contrast, a fuzzy set does not make the same distinction. Figure 2.10(b) shows a fuzzy set where the value a surely is in the set but the value b is in the area between being in the set and not being in the set. This means that the value has some form of membership to the set. This can be further illustrated in figure 2.11. This is the normal way of illustrating fuzzy sets and is called a membership function. If b has the value 0.4 it

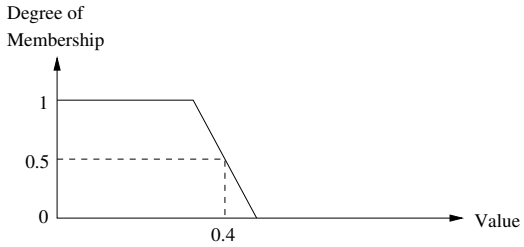
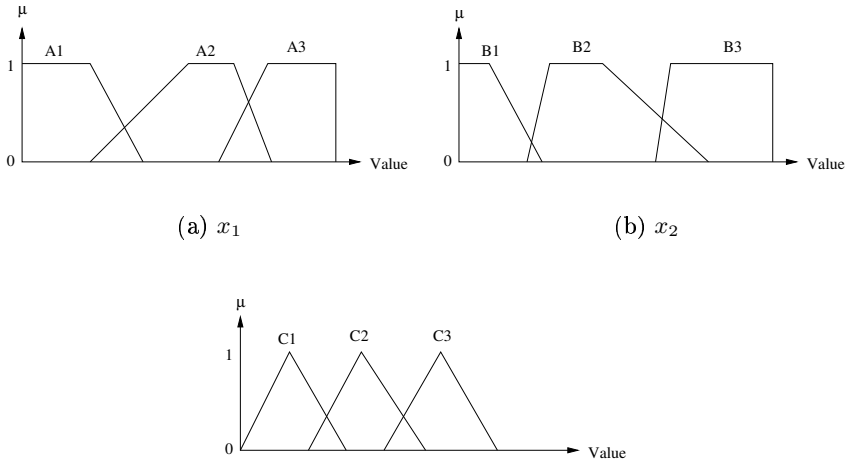


Figure 2.11: Fuzzy membership

can be said to have a 0.5 membership of the set. The degree of membership equal to one is full membership, in this case the value a, and the degree of membership zero is assigned to values that are not in the set or in the fuzzy border area. The degree of membership is often denoted by the symbol μ. The real power of fuzzy logic comes through the use of overlapping fuzzy sets. This provides a unique and robust way of making decisions on information that has a certain degree of vagueness. Consider a process where two variables x_1 and x_2 are measured and the needed output is y. The fuzzy sets can be defined and illustrated as the membership functions shown in figure 2.12. The process could be a chemical process



(a) x_1 (b) x_2

(c) y

Figure 2.12: Fuzzy membership functions

where x_1 is a temperature measurement, x_2 is the reaction speed of the process and y is the decision whether or not to add a specific component to the process. To make decisions based on x_1 and x_2 measurements a rule base is needed. This is a collection of *if-then* rules. For example:

IF x_1 is A1 **AND** x_2 is B1 **THEN** y is C1
IF x_1 is A2 **AND** x_2 is B2 **THEN** y is C2
...

The process of deciding what to do in a specific case where x_1 and x_2 are measured is called fuzzy inference. The first step is to map the crisp measurements to their respective fuzzy sets. According to the rule base a fuzzy output set is obtained. Several methods have been defined for obtaining such a fuzzy set. A common approach is to use the min-operation to compute the fuzzy output sets and the max-operation to combine the fuzzy output sets into one set. This approach can be summarised in the following formula where r is the number of rules.

$$\mu_{Ck}(y) = max[min[\mu_{Ak}(x_1), \mu_{Bk}(x_2)]] \quad k = 1, 2.., r$$

The formula states that the intersection (min) of the membership functions are computed. Subsequently the union (max) of the resulting set are computed to obtain the final fuzzy set. This method is called the max-min inference method or Mamdani method[6]. Other methods are described in [27, Chap. 8]. The inference process is illustrated in figure 2.13. It should be noted that x_1^* and x_2^* have partial membership of two fuzzy sets A1,A2 and

[6]After the British professor E.H. Mamdani.

B1,B2 respectively. If the measurements of x_1^* and x_2^* yielded a result of full membership in only A1 and B1 respectively the resulting fuzzy output would in this case be the fuzzy set C1. The grey figure shows the resulting fuzzy output set. In order to obtain a crisp value y^*

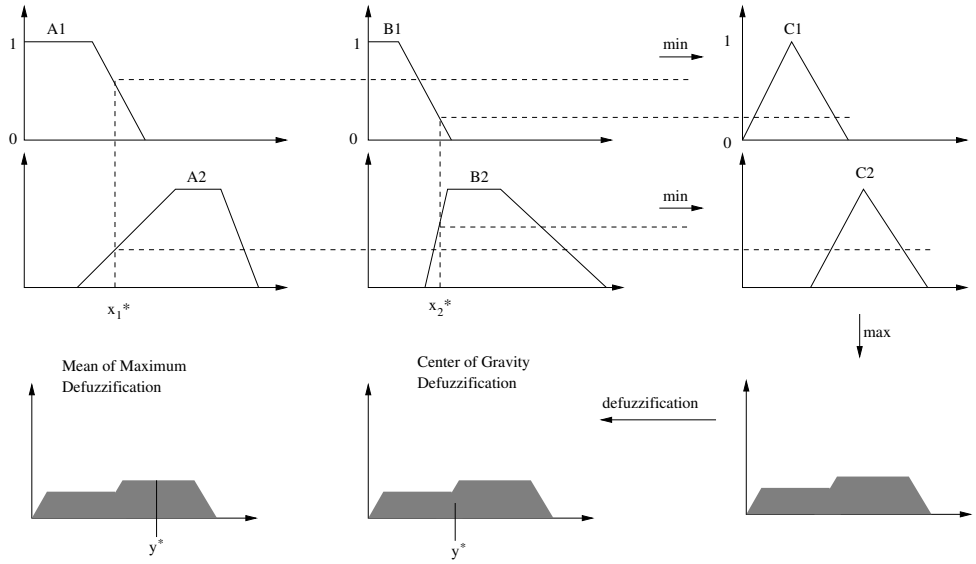


Figure 2.13: Fuzzy inference and defuzzification example

(or action that should be performed) the defuzzification process must be performed. Again several defuzzification methods are available and a detailed description can be found in the book *Fuzzy Logic with Engineering Applications*[27, Chap. 5] by Timothy J. Ross. Some of the most widely used methods are Center of Gravity (CoG) and Mean of Maximum (MoM). CoG is, essentially, the weighed average of the output membership function, while MoM is the mean of the highest points (maximum) of the output membership function. The difference is shown in figure 2.13. The defuzzification methods are often divided into the categories "Centroid" and "Maxima" methods and are variation of either the CoG or the MoM method. Which method to prefer is highly dependent on the specific application.
Several chapters could be written about the different aspects of fuzzy logic. This description of fuzzy logic is in no way meant as complete description of the possibilities of fuzzy logic, but merely points out some of the strengths that fuzzy logic has when it comes to decision making on uncertain variables.

2.6 Bayesian statistics

Several of the models presented earlier in this chapter rely on concepts from Bayesian statistics. Particular the beta reputation system from 2.3.3 and the trust management models from section 2.4.3 are based on knowledge obtained from Bayesian statistics. For a better understanding of the models I will give a general introduction to Bayesian statistics.

Bayesian statistics are not a new concept. It relies on the work done by Thomas Bayes (1702-1761) and what is known as Bayes theorem. Thomas Bayes was a priest who never published a mathematical paper in his life. The paper in which the theorem appears was read before the Royal Society by Richard Price in 1764. The theorem provides a fundamentally different approach to data analysis, and because of its difference from regular frequency analysis, the Bayes concepts are often misunderstood or misused. To avoid misunderstandings I will list the notation with a short description:

- $P(A|H)$ denotes the probability of A given the conditions H. This is also called the conditional probability.
- $P(B|A,H)$ denotes the probability of B when the conditions A and H are true.
- $P(A,B|H)$ denotes the probability of both A and B happening given conditions H. This is also called the joint probability.
- $P(A \text{ or } B|H)$ denotes the compound probability and states the probability of A or B happening under conditions H. A and B must be mutually exclusive, meaning that both can't be true at the same time.
- If $A_1, A_2, ..., A_i$ covers all possibilities it is said to be exhaustive.
- X denotes an observation.
- $P_0(A_i|H)$ denotes the prior probability of A_i. This is the probability of A_i under the conditions H before the observation X.
- The posterior probability of A_i is denoted by $P(A_i|X,H)$ and states the probability of A_i after the observation X.

Given the above notations Bayes' theorem can be defined. The condition H is assumed be true both before and after the observation X, and H is therefore not directly stated in the formula. This is the most general form of the theorem:

$$P(A_i|X) = \frac{P_0(A_i)P(X|A)}{\sum_j P_0(A_j)P(X|A)}$$

The proof of the theorem is simple and can be seen in the book *Measuring Uncertainty: An elementary introduction to Bayesian Statistics*[28, p.65] by Samuel A. Schmitt. The application of the theorem can be understood as a way for a new observation to influence the previous observations. Meaning that the prior probability, or prior probability distribution, is used in connection with the new observation to obtain a posterior (newest) probability or probability distribution. When subsequent observation is obtained the posterior probability becomes the prior probability used to calculate the new posterior probability. The impact of this is best illustrated with an example. I use a classic example from [28]:

Suppose a person participates in the mass screenings for a relatively rare disease. It is known that the disease strikes about 1 in 5000, and the screening test has about 5% false positives and about 2% false negatives. This give a prior probability of:

Not diseased	Diseased
4999/5000	1/5000

The conditional probabilities are :

$$P(\text{positive}|\text{not diseased}) = 0.05 \quad P(\text{positive}|\text{diseased}) = 0.98$$
$$P(\text{negative}|\text{not diseased}) = 0.95 \quad P(\text{negative}|\text{diseased}) = 0.02$$

Two outcomes of taking the test is possible.
1) The test is positive:

Alternative	Prior	P(positive|alt)	Joint	Posterior
Not diseased	.9998 x	.05	= .049990 →	.9961
Diseased	.0002 x	.98	= .000196 →	.0039

Note that the posterior probability for not diseased is calculated by $0.49990/(0.49990 + 0.000196) = 0.9961$ and likewise for posterior probability for diseased. These calculations lead to the following result when the test showed a positive result. Before the test there was a $1/5000 = 0.02\%$ risk of having the disease. After the test the risk of having the disease rose to $0.39\% \approx 1/256$. This of course does not mean that the person has the disease but further testing should naturally be done.

2) The test is negative:

Alternative	Prior	P(negative|alt)	Joint	Posterior
Not diseased	.9998 x	.95	= .949810 →	.999996
Diseased	.0002 x	.02	= .000004 →	.000004

Because the test comes back negative it does not mean that the person can not have the disease. The risk of having the disease drops however from $1/5000 = 0.02\%$ to $0.0004\% \approx 1/250000$. This indicate that it relatively unlikely that the person has the disease, however not impossible.

The example illustrates the Bayesian approach to data analysis. Intuitively this approach also seems probable, however in regular statistics, commonly called frequency statistics, a given experiment should be repeated a number of time in order to assess the probability of the observation. In other words, frequency statistics examine the probability of the data given the model (hypothesis), while Bayesian statistics examine the probability of the model given the data.
The terms likelihood and odds are useful when talking about or arriving at posterior probability distributions. Likelihood or likelihood function is the name given to the probability conditional on a particular alternative. In a sense, likelihood works backwards from probability. For example: given B, the conditional probability $P(A|B)$ can be used to reason about A. Given A, we use the likelihood function $P(A|B)$ to reason about B. It can be interpreted as $b \to P(A|B = b)$. Odds can be interpreted as the ratio of the prior or posterior probabilities. For example prior odds=$\text{prior}_a/\text{prior}_b$ and posterior odds=$\text{post}_a/\text{post}_b$.

The literature about Bayesian statistics is vast, and this section has only scratched the surface of the possibilities of Bayesian statistics. Further information about Bayesian statistics found in the book *Measuring Uncertainty: An elementary introduction to Bayesian Statistics*[28].

2.7 Summary

In this chapter I have given examples of the considerations needed when using the notion trust. The common perception of trust relies on a series of subconscious considerations that in the end conclude whether or not an entity should be trusted. In a computer security context such subconscious considerations need to be performed by a computer. Two different notions of trust were described. The first notion, identity trust, involves the belief in entity's identity. The two popular identity trust schemes PGP and X.509 certificates were described. The second notion of trust, incremental trust which will be further investigated in this thesis, involves the calculation of the expected behaviour of an entity. The notion of reputation connected with incremental trust was also described as well as reputation models such as TRUMMAR, Eigentrust and the beta reputation system. Further I showed how computer systems are to manage trust with examples of the wide range for the trust management models. Models using discrete methods, Bayesian methods and fuzzy logic methods were also introduced. Finally introductions to fuzzy logic and Bayesian statistics were given in order to enable a better understanding of the trust management models.

Chapter 3

The notion of trust in grid systems

In the previous chapter I described some of the research that has been done in the field of trust and trust management. While all of the consideration and the models present interesting and unique solutions to individual problems, some specialise on general issues with identity, trust or reputation by focusing on agents, e-commerce, peer-to-peer or grid systems. All of the models can contribute when researching incremental trust in grid systems, but some are more relevant than others. In this chapter I will explore the models that I believe to be relevant for this particular project.

3.1 Security considerations for grid systems

Grid systems by nature rely on distribution. Centralised management with lookup tables and global policies goes against this concept. This means that a centralised management scheme would violate the conceptual simple purpose of a grid system, which is to allow easy access to remote and distributed computing resources[1]. Consequently no security scheme for a grid system should impose on this principal by significantly deteriorating the performance of a grid environment through centralised security enforcements. With that in mind, no feature in a grid system should rely on getting information from every grid participant. This means that all resources connected to the grid infrastructure should be able to handle the security themselves. Additionally this should be done without the need of contacting every resource in the grid for every security related decision. This in no way means that every resource has to store the history of everything it ever experienced forever. Several gigabytes of history would require an continually increasing amount of storage space but more importantly it would also mean longer and longer computing time for calculating security values.
A trust management system in a grid system environment would have to fulfil the requirements of being a decentralised system imposing only an acceptable amount of overhead

[1]The term *resource* denotes either a resource that provides a computational resource to the grid system i.e. a site, or a resource that consumes the computational resource made available in the grid i.e. a user.

in the system without the need for excessive communication and storage. In such a security scheme it would not be expected that at a given time each and every resource has the same security related opinion on a given matter. Over time more and more resources would tend to agree on a security matter but instantaneous spreading of information is not a possibility. This also indicates that trust should be a slow varying factor. A potential implication of this could be that a given resource is perceived as having a low trust value by all but a few resources who in turn lay open to the bad behaviour of the resource. A resource could continue to exercise its bad behaviour longer than if the security followed a centralised or full propagation of information model. A way of combatting this would be to enable the grid resources to form networks and alliances within the grid system. These alliances or networks would exist to allow resources to notice significant changes in trust information quicker. This mirrors the social networks between humans, if a friend experiences another person's very good or bad behaviour the friend could in turn give recommendations or warnings about that person to his friends. Evaluation of course depends on how much faith you put into your friends' advice. If a security scheme included a notion for friends/alliances/networks changes in trust information, would spread quicker without significantly imposing on the grid principles hereby limiting the time frame for resources to exploit the distributed security scheme.
None of the models presented in the previous chapter put significant emphasis on how to measure the behaviour of other resources. The models for peer-to-peer system, for example Eigentrust from section 2.3.2 and Yao Wang and Julita Vassileva's model from section 2.4.3, both use a threshold value. The download speed of a peer can for example be satisfactory or unsatisfactory depending on whether or not it was below an empirical threshold value. In a grid system a binary notion to describe the interaction result is not suitable. Since the interactions in a grid system mainly consist of submitting and processing jobs, the definition of threshold values to describe the wide variety of possible jobs would be quite cumbersome. Instead the result of an interaction should denote a degree of satisfaction. The parameters for measuring behaviour with the purpose of a subsequent calculation of a trust should be based on the result of the job processing.

3.2 Trust parameters

In reality one can only make decisions based on a fixed number of parameters. These parameters are used to measure the behaviour of a resource in order to enable a subsequent trust value to be calculated. The accuracy of the calculation of a resource's behaviour can be increased by basing it on a larger amount of parameters. However, more parameters mean more calculations and in turn higher overall security overhead. If the inclusion of a parameter only increases the accuracy by a small degree, but imposes a significant overhead in calculations, the justification of including the parameter is questionable. If the decision is the same with or without the parameter, there is no justification for imposing overhead by including it. Additionally if a security system for grid system is to have any practical application the parameters must relate to the quantifiable possibilities of the grid system. If

grid computing is simply viewed as a way for executing computing jobs on remote resources, the actual job would contain several easily quantifiable values. An example set of job values

CPU time requirement
Memory space needed
Storage disk space needed

Table 3.1: Job values

are listed in table 3.1. A resource would be able to measure the behaviour of another resource and in turn make trust decisions based on how much the actual processing of a job deviated from what was believed. From a site's point of view it would be determining how much the informed requirements deviated from the actual requirements. From a user's point of view it would be determining how much the actual allocation of time, memory etc. deviated from what was informed. Deviation of quantifiable values can serve as parameter for trust calculation.
Other parameters to use as trust parameters would be the recommendations from friends and allies (described in greater detail in section 3.3) and possibly some special values. Special values could for example be from an IDS (intrusion detection system). An IDS has the ability to calculate how probable it is that the system has been compromised.

3.3 Grid alliances

Considerable emphasis should be put into defining alliances or special relationships between resources in a grid security scheme, since this provides faster adaption to security changes within the system. The alliances should form the basis of getting relevant and accurate reputation information. The formation of alliances is not a trivial problem since a certain amount of automation should be required. If every resource connected to the grid manually had to define each and every resource whom it had an alliance with the result would be a very inefficient alliance structure. Simply put, the amount of work in identifying and adding to a alliance list, would become tiresome and overwhelming in a dynamic setting such as a grid. Too few alliances would result in a slow propagation of trust information. Too many alliances would result in a large security communication overhead.
One possibility for overcoming the initial creation of alliances could be to form alliances based on geographical distance. Meaning that if one had the option of forming an alliance with one of two strangers, where one is your neighbour and the other resides on the other side of the world, the most probable choice should be the neighbour. If your neighbour misbehaved or betrayed the alliance, confronting the neighbour would be possible. One could obtain a reason for the betrayal and consequently remove or let the alliance remain. Investigating bad behaviour of a stranger that resides on the other side of the world would not be that easy. The reason might be just as good, but confrontation would be more cumbersome. If, over time, a stranger on the other side of the world proved to be a very trustworthy individual an alliance could naturally be formed.

Most current grid technologies rely on a certificate in order to form the basis for access-control. A geographically based initial alliance structure could be initiated by the information in the certificate.
Suppose the resources of a grid was geographically located as shown in figure 3.1. Here

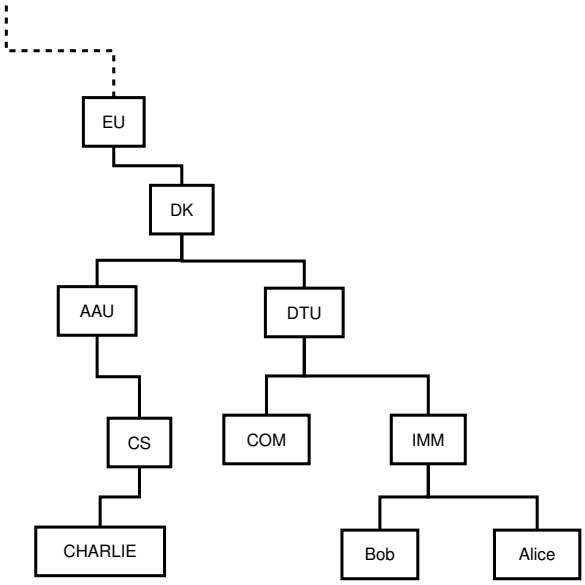


Figure 3.1: Initial alliance structure.

Alice and Bob are both located in the same region (EU), country (DK), university (DTU) and department(IMM). Charlie, on the other hand, is located at a different university. If Alice was presented with the certificates from Bob and Charlie, alliances could be formed. Alice would compare the number of shared fields in the certificates. In the case of Bob there would be 4 and in the case of Charlie it would be 2. The alliance between Alice and Bob would consequently be stronger than a possible alliance between Alice and Charlie. Simply put, one could say that Alice and Bob are in the same four groups yielding an alliance strength of four, whereas Alice and Charlie are only in the same two groups yielding a strength that equals two.
The comparison would be easy if the certificates included region, country, university and institute but that might not be a practical solution in every grid system. Certificate policies could dictate that grid-certificates only contained the resource name and issuer's name. To avoid changing the policy and issuing new certificates, an X.500 directory service or LDAP service could be associated with the grid. By looking up the issuer's name and resource name the complete path could be extracted and alliances could be formed similarly to when the certificate contained the information.
It should of course be noted that alliances formed in this way don't dictate that an alliance with a resource should remain forever, regardless of the resource's behaviour. The initial alliance formation should only serve as a starting point.

3.4 Evaluating trust

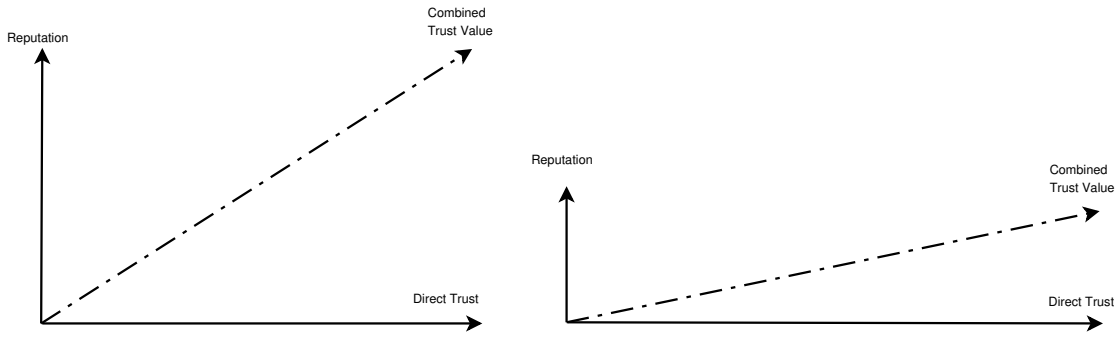
The motivation for a notion of trust in a grid system would be to allow resources to make decisions that with a high degree of certainty are correct. This involves learning from past mistakes and predicting the most likely outcome of a particular event. Many of the models reviewed in the previous chapter calculated trust as a discrete value with a variation of the formula:

$$\alpha * \text{Trust}_{old} * e^{-(t-t_0)/\tau} + \beta * x_1 + \ \ldots = \text{Trust}_{new}$$

By defining suitable values for the weighing factor α, β and the constant τ for the decay of trust, promising results can be obtained from simulations. By including more and more factors (x) an increasing amount of special cases where resources misbehave would be correctly reflected in the simulation experiments. Other schemes base the trust evaluation on a combination of own experience (called direct trust) and reputation trust:

$$\alpha * \text{Direct trust} + \beta * \text{Reputation trust} = \text{Trust}$$

Again promising results can be obtained with suitable α and β values through simulation. A problem with both approaches are finding reasonable values that can cope with all possible scenarios in a real life grid environment. The values are a heuristic which means that there is no proof that the values are the best possible solution. The best one can say is that with a gives set of values the outcome is satisfactory.
It is questionable that a notion of trust simply can be perceived as a scalar value. It will, to say the least, be difficult to find values that always weigh parameters optimally. Instead one could view trust as being a vector. If for example direct trust and reputation trust are the only trust notions, the combined trust value could be illustrated in 2D as shown in figure 3.2(a) where the length of the Reputation and Direct Trust lines are defined by their values. This could naturally be expanded to as many dimensions as needed according to



(a) Scenario 1 (b) Scenario 2

Figure 3.2: Trust as a vector

the number of trust parameters. But the question as to how one can compare two combined trust values remains. The length of the lines in figure 3.2(a) and 3.2(b) are the same but this doesn't necessarily mean that the trust levels are equal.
As described in section 2.4.4 Song and Hwang combine the trust parameters Intrusion Detection System(IDS) and job success rate with fuzzy logic. Since fuzzy logic provides an easy way of doing computing on terms that are linguistically defined, it seems reasonable to use it as a way to make decisions on trust parameters. The parameters would fit into fuzzy sets, for example "high" reputation trust and "low" direct trust. For this reason the use of fuzzy logic as a way of evaluation trust parameters is a topic I am going to investigate further through simulation.
As described in section 3.2 two interesting trust parameters are the personal experience (or direct trust) and reputation trust. Many of the reputation systems described in the previous chapter also follow a discrete calculation method similiar to:

$$R_x = \alpha * \sum_{i=0}^{n} R_x^i \ldots$$

By summarising the reputation trust values given from friends, neighbours and strangers an opinion about x could be formed. The problem is that reputation information is instantly forgotten after it has been calculated. It does not provide any substantial ability to learn from past mistakes. Every time it is calculated, the entity looking for a reputation value is at the mercy of the recommenders. The beta reputation scheme on the other hand, presented in section 2.3.3, involves a interesting scheme that through previous observations continually improves a reputation trust value. Intuitively the Bayesian approach of subjective observations is a fitting term when it comes to calculating reputation values. For this reason I find the use of the beta reputation system worth investigating further.

3.5 Ratings

Several of the trust management models presented in the previous chapter used a notion of reputation trust. The reputation of an resource is based on the ratings given by recommenders. The recommenders that submit ratings can have different motivations for giving ratings. None of the models however, discussed the possible motivations for a recommender. This is the purpose of this section.
In a grid environment there are basically two different groups of resources: resource consumers i.e. the users and resource providers i.e. the sites. The users all have a similar expectation for participating in the grid system, and likewise for the sites. If a user experiences that the jobs that it submits are quickly processed and a correct result is returned, the user's expectations for the site are met. The user will happily submit more jobs to the site in the future. If a site receives a job from a user with an accurate description of the jobs requirements, the site can perform an efficient scheduling of jobs. Subsequently the site's expectation for the user is met if the job did not indicate any malicious act. In the future the site will happily accept more jobs from the user.

When a resource, be it a user or a site, is asked to give a recommendation it will base such a recommendation on the experienced behaviour of the resource in question. Since a user only has experiences with the behaviour of sites, the user would find it hard to give ratings of another user. If a user were to give a rating of another user, the rating could only be based on the quality of the ratings received from the user. In order to determine the quality of the received ratings, the actions leading up to the received rating had to be examined. Since in a decentralised environment there is no central facility that lists all actions by all resources, such an examination would be complex and time consuming. The only way it could be done, would be to contact the resources that the resource that is being examined has been in contact with and enquire for the evidence of the interaction. It is also possible that one of the resources that provided the evidence had to be examined, which would lead to recursive examination of resources. It would quickly evolve into a problem bearing strong resemblance to the Byzantine generals' problem[2]. If users only give ratings of sites and sites only give ratings of users, this can in large parts be avoided. Such an investigation can be avoided because the motivation for giving inaccurate ratings changes. The resources would have less to gain by giving inaccurate ratings. This can be illustrated by comparing a grid system to a commerce scenario:
Imagine the scenario of banks and customers. The banks can be compared to the sites and the customers can be compared to the users. The purpose of the banks is to extend credit to customers and the goal of the customers is to accumulate the highest amount of credit. Credit would be the equivalent of processing time. A bank would be satisfied with a customer if the customer followed the credit agreement. A customer would be satisfied with the bank if the bank presented a proper credit agreements and if the bank fully honoured the credit agreement. Suppose a customer approached another customer and asked for a rating of a bank. Assuming that the banks have unlimited credit to loan, the customer would have little incentive to lie. He would not gain anything by keeping the customer away from a fair bank and directing him towards a unfair bank. If a bank asked another bank for a rating of a customer the same arguments apply. It can be assumed that banks are part of a business federation who's goal is to assure that banks appear loyal and trustworthy toward the customers. Assuming that a customer has an unlimited need for credit the bank that gives the rating knows that the customer will return. However, if a bank asked a customer to rate another bank the customer would have much to gain by an overstated rating of the bank. The purpose would naturally be to acquire a significantly better credit agreement with bank that asked. If a costumer asked a bank to give a rating of another costumer the same would apply. By convincing the customer that the customer in question was much better than the customer asking, the bank could convince the customer to sign a new and more favourable from the banks point of view, credit agreement.
A model where users only are allowed to rate sites and sites only are allowed to rate users is a monotonic rating system. The purpose of using such a scheme is simply that the incentive for giving false and inaccurate ratings is greatly reduced.

[2]The classic problem: Two separated generals will win a battle if both attack at the same time but lose if either attacks alone. They send messengers but they may be captured. If one decides to attack, how can he be sure that the message has reached the other general and that the other general will attack too?

3.6 Summary

This chapter has described some of the security considerations needed for trust management in a grid system. This included arguments as to why such management should be decentralised with the purpose of allowing individual resources to measure and calculate trust themselves. Some of the possible parameters that can be used to measure the behaviour was listed. The concept of alliances and how an initial alliance structure could be set up were also described. A brief overview of the commonly used methods for calculating trust values and reputation trust values was also given. This also included arguments explaining why the fuzzy logic and the Bayesian approaches are particularly interesting. Finally the often overlooked aspect of resources rating other resources were presented, and I argumented why a monotonic rating scheme reduces the incentives for rating inaccurately.

Chapter 4

Simulation

Like the models presented in chapter 2 I will examine the progression of trust in a grid system by creating a simulation program. The purpose of this chapter is to document the simulation. First I will analyse the simulation requirements. This is followed by a design description and subsequently the implementation will be briefly be described. Finally I will describe the testing efforts that I have made in order to ensure the accuracy of the simulation output.

4.1 Analysis

A grid system can be viewed as a system that facilitates easy communication between resources. The term resource can describe either a resource that provides a computational resource or a resource that consumes the computational resource, in the following referred to as a site or a user. A user can submit a computing job to one of several sites, which in turn processes the job. When the job has been completed the user can retrieve the job result. If the job failed the user must be notified hereof. This simplified view of a grid system is illustrated in figure 4.1. Since the objective of the simulation is to experiment

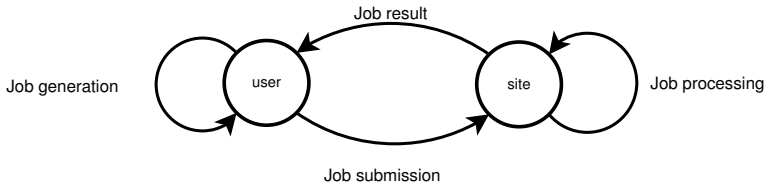


Figure 4.1: Simplified grid system.

with trust in a grid system the simulation must include a notion of trust. More specifically the simulation must include the following abilities:

- A resource can associate trust values to other resources
- A resource can update trust values according to events
- A resource can make decisions based on trust values

For a user of the grid system this means that the user must be able to decide if a site meets a trust value requirement. Afterwards, the user can submit a job to a site that meets the trust requirements and later retrieve the result of the job from the site. Based on the processing of the job the trust value for the site should be updated. Similarly the site should decide if a user meets the trust requirement and consequently if the user's job should be processed. If the user meets the trust requirement, the job should be inserted into the job queue. Otherwise the user should be notified that the job was rejected. After a job has been processed the trust value of the user should be updated to reflect the event. This sequence of actions is illustrated in figure 4.2. As described in section 3.2, trust can be based on

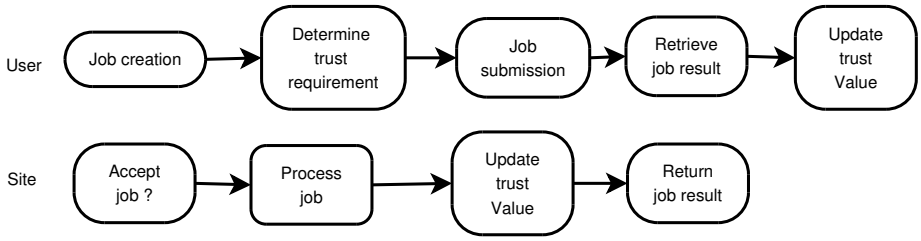


Figure 4.2: Sequence of actions.

a great variety of parameters. I propose that both users and sites need the information from the job processing in order to determine a degree of satisfaction from the interaction which leads to the calculation of a trust value. How to determine a degree of satisfaction is further analysed in section 4.1.1. By combining the trust value calculated from the interaction with the old trust value, a notion of direct trust can be calculated. As several of the models presented in chapter 2 suggested the combination of direct and reputation trust can determine an overall or combined trust value. This process for the user and site is illustrated in figure 4.3. There are two significantly different methods for combining direct

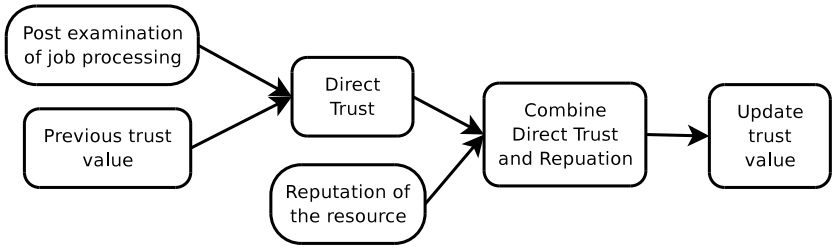


Figure 4.3: Trust update.

and reputation trust. The most commonly used is the discrete approach as used by Farag Azzedin and Muthucumaru Maheswaran, described in section 2.4.2. The other method is to use fuzzy logic as described by Kai Hwang and Shanshan Song in section 2.4.4. Their model does however, not combine direct and reputation trust but the arguments for using fuzzy logic in their context still applies when combining direct and reputation trust. It should be possible to use both combination methods in the simulation.
As described in section 2.3 detailing reputation systems, there are two very different methods for calculating the reputation of a resource. The first is by using a discrete reputation system as for example the TRUMMAR model from section 2.3.1. The second is to use Bayesian approach as for example the beta reputation system from section 2.3.3. The simulation should make it possible to use both of the methods, and it would be interesting to compare the quality of the two methods.

The simulation must also meet some flexibility requirements, which will enable the simulation of various scenarios:

- The number of resources will not be fixed.
- The actions of each resource should reflect a behaviour.
- The behaviour of a resource is independent of other resources.
- The behaviour of a specific resource can change at a specific time or several times.
- A resource must be able to give ratings of other resources.
- Ratings can't be assumed to be accurate.
- A new resources could be introduced in a running simulation.
- The time frame for the simulation time can be specified.
- The frequency of jobs must variate.

In order to obtain results that can be analysed, a suitable export method of results is also needed. This means an output path must be specified where data detailing the simulation run and simulation results will be saved for further investigation. The actual investigation of results is however, not a part of the simulation program.

4.1.1 Resource behaviour

The ability to assess the behaviour of a resource is essential in the simulation. As described in section 2.1.1 a universal distinction between good and bad behaviour is difficult. This means that the task of distinguishing between different behaviours in a grid system is in no way a trivial task. With respect to the trust parameters pointed out in section 3.2 certain assumptions can be made when determining degrees of good and bad behaviour. One of the most essential factors is whether or not a job is completed. If a site does not process

the jobs of a user even though the site informed the user that it would, the user's trust in the site must suffer. Similarly if the user misinforms about the requirements of the job, the site's trust in the user will deteriorate. A simple way of expressing this could be via the difference between allocated/informed CPU time and the actual/allocated CPU time needed for the job.
For example if a site allocated 20 seconds of CPU processing time to a job that required 70 seconds of processing time it would be considered worse than if a site allocated 50 seconds of CPU processing time for the same job. In both cases none of the sites allowed the job to finish but the site allocating 50 seconds of processing time should still be considered to have a better behaviour. The argument is that since there does not exist a formula or a program that in every single case can accurately predict the required CPU time of a given job, the probability of the job succeeding when its allowed to run for 50 seconds is higher than when the processing time is limited to 20 seconds. The difference between allocated processing time and needed time can serve as a measurement for behaviour. If on the other hand a job required only 20 seconds of processing time to complete, a site that completed the job in 30 seconds should be considered to have a better behaviour than a site that completed the job after 100 seconds of processing time. The site that completed the job in 30 seconds is closer to meeting that particular job's requirements. The user does not have to wait as long before obtaining the result of the job. If a site takes longer to complete the job it might indicate that the site swapped the job out of memory a couple of times or that it is scheduling jobs inefficiently. In most cases a user would prefer to send jobs to sites that processed jobs as efficiently as possible, as it is desired from a performance point of view.
If the grid system had a payment scheme that required users to pay for each CPU hour spent, it further validates the preference of the site with the lowest completion time, since it also in the long run would be the cheapest. To summarise, a user could asses the behaviour of a site using the numeric difference between the allocated processing time of a job and the informed processing time needed. This will be denoted Δ CPU time.
Similar arguments can be made when a site assesses the behaviour of a user. In most cases a user is required to give an estimation of how long a job will take. If the user informed a CPU time requirement that was less than was actually needed, it would mean that the site had to drop the job or extend the the processing time allocated. If the job has to be dropped, this basically means the time already allocated to the job was a waste, and if more time is allocated to the job to allow it to be completed it would push back other jobs. Similarly if the user informed the site that the required processing time to be significantly more than what actually was needed, the sites scheduling of jobs would become more inefficient. If there was a payment scheme for the grid it would result in a loss of income for the site. To summarise the numeric difference between the actual and the informed CPU time can also serve as a way of measuring the behaviour of a user. From the site's point of view this will also be denoted Δ CPU time.
It could of course be argued that in the real world a site would never know for certain how much CPU time would actually be required from a job without completing it. However, with the assumption that a site occasionally will decide to complete a user's job regardless of how much time was specified by the user, the actual CPU time requirements could be established. A site policy that for example states that at least every 10th job submitted

by a user should be completed, could allow the site to make trust calculations of every 10th received job. From the user's point of view the calculation of trust is more simple. By comparing what was asked for to what was received, in terms of CPU time, a good measurement of a site's behaviour can be calculated. However, without the assumption that the user is competent when specifying the required amount of CPU time, a user could exhibit bad behaviour without even knowing it. If the user for example genuinely believed that a job required significantly less CPU time than what actually was the case, the trust calculation for the job would, from the site's point of view, result in a low degree of satisfaction and consequently also a low trust value for the user. This however, merely illustrates the important distinction between intent and behaviour. It is the behaviour that is being evaluated in the trust calculation, not the intent of the resource.
Assuming that a trust value is in the interval [-1:1] the calculation of trust given to a specific job could be determined according to the figure 4.4. When Δ CPU time is small or close to

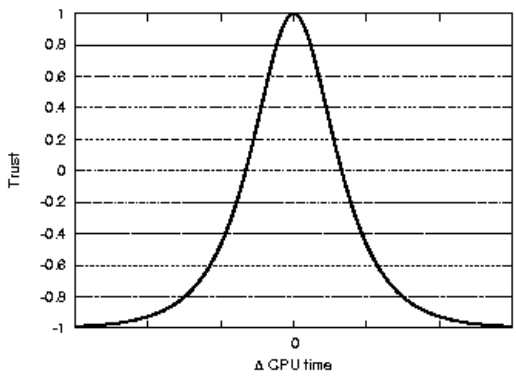


Figure 4.4: Calculating a trust value based on Δ CPU time

zero this will result in a high degree of satisfaction from the interaction and consequently a high trust value. From this site's point of view it did not waste time on a deceiving user, and from the user's point of view it did not waste time on a site that did not deliver what was promised. When the Δ CPU time increases the degree of satisfaction falls. From the user's point of view either the site did not complete the job or overcharged the job processing. From the site's point of view, the user either understated the processing time of the job or overstated the processing time of the job.
The question remains as to how to evaluate the behaviour of a resource if a job is rejected. From a user point of view it might seem very inconvenient that the site declined to process the job. But compared to the case where a site accepts the job, places it in the job queue and after a long waiting period processes the job for one second and afterwards drops it, an initial rejection is not that bad. With a clear rejection the user can re-submit the job to a different site without a long waiting period. As a result a rejection can be interpreted as, if nothing else, the site is somewhat honest. This does of course not mean that sites that reject jobs should be trusted more, since the reasons for rejecting a job could be that the site does not trust the user sufficiently. A suitable action taken by a user would be to

compare its trust in the site with the reputation of the site. Since the site did not process the job, the user can not evaluate how well the job processing went. This means that no direct trust value can be calculated. Instead the user could rely on the reputation of the site for updating the trust in the site. If for some reason the user has a higher trust value for the site than the reputation of site the trust value naturally should not be updated to reflect this. Instead the trust in that particular site should decrease a little. From a site point of view the rejection of a job is somewhat different. The primary reason for refusing to process a user's job would be that the user is not sufficiently trusted. It should not influence the site's trust in the user that a job was rejected. Since the reason the job is rejected is insufficient trust in the user, the user should not be blamed for trying again. Instead a suitable period of time should elapse before accepting another job from the user as a way of examining whether or not the user has bettered its behaviour. If the user has bettered itself, trust in the user should naturally increase to a level that indicates that it has improved its trustworthiness, but if this is not the case there should be an even longer period of time before the user's jobs is retried. What is important is the number of times the user's jobs has been rejected. This means that one rejection does not change the site's trust in the user but if the job of a user is retried without noticeable improvement the trust in the user should suffer. Likewise for the user. If a user decides not to send a job to a particular site because of insufficient trust, the site could then be retried after a period of time. If no improvement is detected when retried, it should be even longer before the next retry.
When a resource misbehaves and is rejected over a period of time, but then improves it behaviour, the record of the previous bad behaviour should be forgotten. The term "once a crook always a crook" should not apply. It is expected that occasionally a resource will misbehave. For example a site might be misconfigured or experience a hardware failure, or a user makes a mistake about the required job completion time. What is important is not the small glitches a resource can make, but the overall long term behaviour of the resource. If a resource, in the long run, exhibits good behaviour, it is to be trusted. If it exhibits bad behaviour in the long run, it is not to be trusted. This intentionally reflects a "positive" approach to assessing the behaviour of others.
The parameters described in section 3.2 listed several parameters that could be used for assessing the behaviour of resource. For example the values describing storage and memory requirements could also be used. Calculating a behaviour from these two values would be very similar to calculating behaviour from the CPU time. Because of the similarity I find it useful for simulation purposes to limit the calculation of behaviour to rely on merely the CPU time. Furthermore the introduction of storage and memory as parameters to calculate behaviour, would need different contexts, since scenarios would arise to reflect situations where one parameter should have a higher significance than the remaining parameters. Although different contexts still could be defined when using just the CPU time, a single context of evaluation can be considered more reasonable. Similar considerations can be made when contemplating including an IDS value for calculation of trust.

4.1.2 Progression of trust

A high level of trust in another resource should not instantaneously be brought about. As previously described, a high trust value should be obtained over time based on a significant number of encounters, where each encounter resulted in a positive outcome. Likewise a resource should be deemed untrustworthy after a series of encounters that resulted in a negative outcome. This does however, not mean that a positive experience with a previously unknown resource only should have a small impact on the trust value in that resource. Intuitively a positive encounter would result in a high trust value in that particular resource. It would however require a long series of positive experiences to give the resource a very high trust value. Likewise if a previously unknown resource misbehaves is would result in a low trust value. But it should require a long series of negative experiences to give the resource a very low trust value. To summarise a high trust value should be brought about quickly if the resource is behaving properly but it should take a long time to get a very high trust value. Likewise a low trust value should be given quickly with bad behaviour but a very low trust value should be based on many bad experiences. With a trust value being in the interval [-1:1] this can be illustrated in figure 4.5. The x-axis denotes the experience and

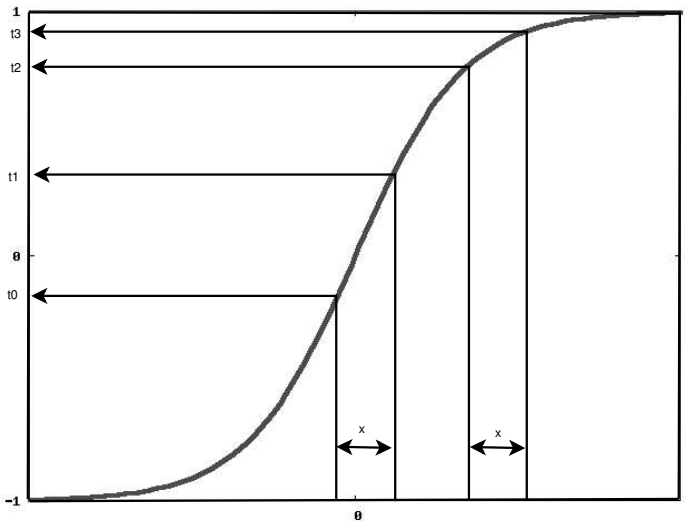


Figure 4.5: Trust progression.

the y-axis denotes trust. If a resource has the trust t_0 and a given interaction has resulted in a change x, the new trust would be t_1 . If the resource however, had the trust t_2 but the same experience x was observed, it should only result in a relatively small increase in trust to t_3.
The figure also implies that a maximum or minimum trust should not be reached by resources. If for example the highest theoretically possible direct trust value is 1 then a direct trust value of 1 would make the notion of direct trust meaningless. Having a direct

trust value of 1 would mean that the resource is trusted beyond any reasonable doubt. Or put in another way, no matter what the resource does, it would be trusted. Although the borderlines of trust is an interesting topic, it is not the immediate focus of my project.

4.1.3 Rating other resources

A resource should also be able to serve as a recommender by rating other resources. These ratings are the foundations for a reputation system, which enables resources to learn from other resources experiences. It can however not always be assumed that a received rating is accurate. In the worst case it could be misleading. To help avoid misleading ratings a notion of alliances is helpful. An alliance between resources could also be interpreted as a "friendship" between the resources, and if in doubt most would trust the ratings given from friends higher than ratings given from strangers. However alliances can also inspire resources to give inaccurate ratings about their alliance partners. A resource could be motivated for lying about an alliance partner. If a stranger asks a resource for a rating on one of the resources friends then the rating could be a more positive rating than what it actually should be. The term "you help your friends" should apply in this simulation. However if a stranger asks a resource for a rating of a resource that is not a "friend", meaning that there is no alliance, the resource could answer depending on its honesty. Since there is no special connection between the resources there is not a particular strong incentive for the resource to lie. If it does not care particularly about either of the resources it could lie a bit, should the resource feel the need to confuse other resources, or it could give a quite accurate rating, should the resource an honest resource seeking to help its surrounding resources. The following can be summarised about a resource's behaviour when it comes to giving ratings:

- If an alliance partner asks for a rating of another resource, the recommender's rating will be accurate.
- If a resource asks for a rating on an alliance partner, the recommender's rating will be higher than how the resource actually should be rated.
- If a resource asks for a rating where there is no alliances with either of the resources the given rating will depend on the honesty of the recommender.

Various notions exist to denote honesty. It would however seem suitable to use to behavioural values, used to determine informed/allocated CPU time, to determine how honest a rating will be. Higher values will result in more inaccurate ratings, lower value will result in more accurate ratings. Consequently a highly dishonest and misbehaving resource exaggerates a lot when giving ratings of alliance partners. A somewhat honest and behaving resource will only exaggerate a little when giving ratings of alliance partners. Likewise the misinformation with ratings of non-alliance partners to non-alliance partners will be greater with increasing behavioural values.
It should be noted that it would be possible to have an alliance partner that is not trusted

when it comes to processing jobs or accepting jobs, but that same alliance partner could be trusted when receiving ratings. In the real world it would also be possible to have a friend whose recommendations of a merchant was trusted, but whom you would not allow to for example borrow your car. The trust needed to handle jobs is not comparable with the trust in a rating. As described in section 3.5 users should only seek ratings of sites from other users, and sites should only seek ratings of user from other sites. The monotonic approach simplifies and reduces the motivation for misleading ratings.

4.1.4 Event types

During the simulation a number of events could occur. The most common event that should happen would be the event where a user submits a job to a site. A subsequent event should be used to allow the site to notify the job creator that the job processing has finished, or in case the job was rejected, the job creator should be notified hereof. Since the behaviour of a resource must be changeable at a given time there should also be an event type used for such an occurrence. Likewise when a new resource is introduced into the running simulation it requires an event. This is summarised below:

- Job submission
- Job result retrieval
- Job rejection notification
- Resource behaviour change
- Addition of a new resource

The list contains the five different events that can occur in the simulation. It might seem simplistic to describe the possibilities of a grid system only to allow five different occurrences. However I consider the five listed event types to be the most essential events when considering trust management in a grid system. In a real world grid system, sites could have scheduled down times or be taken out of the grid. The event used for job rejection notification could simply be used if a site is offline. Additionally in a real world grid system a site would spend time on saving or swapping out job results, and a user would spend time on generating jobs and interpreting the job results, both of which to a large part can be ignored when it comes to using trust management.

4.2 Design

To meet the requirements I have designed a discrete event simulation to examine the progression of trust and managing hereof in a grid system. I attend to implement this design in a object-oriented programming language which means that the design will make use of for example inheriting.

A discrete event simulator models a system by a representation of state variables that changes instantaneously at separate points in time. Or put in another way the system can change at only a countable number of points in time, where the change is reflected in the updating of state variables. The points in time where the state variables can be updated are at the times of events. After an event has occurred and the state variables may have been updated, the simulation time is advanced to the next event. The time between events is skipped. Some discrete event simulations use a fixed increment in time, meaning that the time increment a fixed interval, and all the events within that time segment are processed, and then the time is incremented again with the same fixed value. I have however chosen the next-event time advance approach.
An object is needed that can handle the advance of the simulation time to the next event and handle the processing of that event by updating the state variables. This object is denoted the event handler. Separate objects are needed to hold the state variables. The simplified view of a grid system illustrated in figure 4.1 suggest that objects for users and sites should be used to hold the state variables. These should pass job objects between each other to simulate the submission and retrieval of jobs in a grid environment. The basic layout of the simulation is illustrated in figure 4.6. The number 1 next to the line denotes

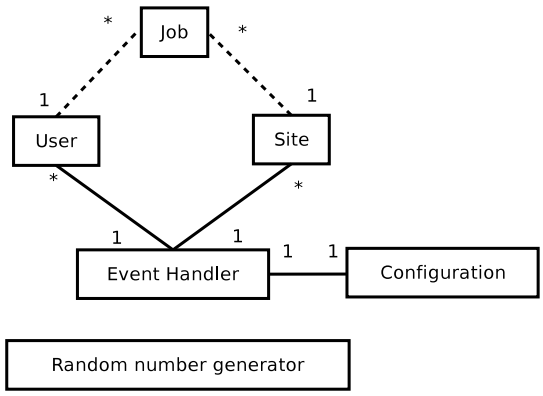


Figure 4.6: Outline of the simulation program

that for example a user can only be associated with one event handler and the * denotes that for example an event handler can have multiple users associated with it. Likewise a user can have multiple jobs associated with it, but a job can only be associated with one user (or site if it is being processed). The requirements of each of these objects are further described in the next sections.

4.2.1 The event handler

The event handler has a significant amount of responsibility. It controls the simulation by advancing the simulation time and by instructing the various objects to update their state variables. The methods that it should provide are described here. The event handler

should:

- Read the configuration simulation file.

Flexibility requirements dictate that a configuration file should be used to specify the simulation parameters. By creating an object that can read the configuration file and return the parameters when they are needed, this is made possible.

- Set up the simulation by creating the simulation objects specified in the configuration file.

The configuration file will include a number of users and sites that should be present in the given simulation. This should be done by instantiating all of the resource objects. These objects will hold the state variables and they are further described in section 4.2.2

- Inform the resource object of each others' existence.

All of the resources have a full and complete picture of the grid. Every resource will have a knowledge of every other resource that is connected to the grid system.

- Have a queue where events could be added.

Before a simulation can be started there must be a finite sequence of events that are to occur. The event handler should have a priority queue where the events are stored. The priority of events are determined by the events' time of execution. The actual generation of events will however not be done by the event handler. This will be handled by an external method. This external method should also model an arrival of events. A Poisson distribution is commonly used for generation of event times.

- Retrieve the next event from the queue and execute that event.

The event handler will retrieve the next event from the event queue and proceed to process the event by advancing the simulation time and instructing the affected objects to preform the actions dictated by the event. The actions of particular events are further described in section 4.2.4. This will result in the update of the state variables.

- When the last event has been processed output the results of the simulation to the location specified in the configuration file.

When the final event from the event queue has been executed, the results from the simulations should be saved. By means of an output method further described in section 4.2.7 this will be made possible. The results can then be subjected to further investigation.

4.2.2 Resources

The requirements for sites and user have a number of similarities. This can be exploited with the use of class inheriting. By defining the methods that both users and sites need in a resource class, we avoid having to define a method twice. The user and the site simply inherit the resource class which makes the methods available. The concept is illustrated in figure 4.7.

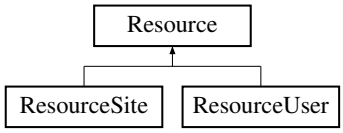


Figure 4.7: Resource inheriting

Data elements
The data elements common for both types of resources are:

- Resource id.

Each resource is identified by a unique id. This resembles a certificate that identifies the resource. It is assumed that the identity of resource is not in question, meaning that nobody can assume the identity of others or acquire a new identity.

- Behavioural values min and max.

The two values min and max will determine the behaviour of a resource. The reason for using two values when determining the behaviour of a resource is further described in section 4.2.8.

- Trust threshold.

The users will have a lower trust threshold for sites. This threshold will determine if they want to submit jobs to a site. Likewise the sites have a lower trust threshold. This threshold will determine if they want to accept jobs from a user.

- List of groups.

A resource, be it user or site, can be the member of a number of groups. The groups will identify the alliances the resource has with other resources.

- Queue of jobs.

A user will have a queue of jobs that it wants to submit to sites. If a site rejects a job it will be inserted into the queue for later submission to (another) site. Likewise the site has a job queue that hold the jobs that it will process. The sites scheduling algorithm will follow the first-in-first-out approach. A more efficient algorithm that prioritised highly trusted users before marginally trusted users could also have been used, but as the focus of the simulation is trust management and not performance such an scheduling algorithm would not be in line with the purpose of the simulation.

- Container for known resources.

Each resource should know the unique id of all of the other resources. Additionally it should have a trust value, and the history of trust values associated with that resource. There should also be a time stating when the trust value was updated. This history of trust values will be outputted after the simulation. The calculation of trust value should not rely on the trust history values.

Methods
Both types of resources update the data values very similarly. The methods for doing this can be defined in the resource class. The methods must:

- Determine direct trust from an interaction.

A user will determine a trust value from a processed job based on how much time it was informed that the job needed and how much time the site allocated. The difference is denoted ΔCPU time. A site will determine a trust value from a processed job based on how much time the user informed that the job would need and the time that the job actually needs to complete. This difference is also denoted ΔCPU time. Based on the resource object type, the direct trust method will extract the needed data from the job object and calculate the ΔCPU time. ΔCPU time will be used to calculate a trust value. When this value is combined with the previous trust value, the result is a direct trust value.

- Calculate the reputation a resource.

A site will need to calculate the reputation of a user by combining the ratings received by sites. Likewise a user will need to calculate the reputation of a site by combining the ratings received by other users. Based on the object type the appropriate resources will be asked to give a rating. These rating will then be combined using either a discrete reputation system, or via the beta reputation system, depending on what is specified to be used. Some details of the reputation methods are described in section 4.2.9.

- Combine the direct trust value with the reputation value.

When a direct trust value and a reputation trust value has been calculated for a given resource, be it user or site, the values need to be combined to a single combined trust value.

As described earlier, the simulation should provide two ways for doing this combination. The first way is by a discrete approach. Here each value has a weight and the sum of both weights is equal to one. This is similar to the approach presented by Farag Azzedin and Muthucumaru Maheswaran in section 2.4.2. The other way is by combining the values using fuzzy logic as proposed by Kai Hwang and Shanshan Song presented in section 2.4.4. They do however, not combine direct trust with reputation trust, but the concept is the same. The fuzzy logic approach requires the use of fuzzy membership functions and rule base, which will be further described in section 4.2.10.

- Add a resource to the container of known resources.

When the simulation is set up, the resource will have a complete knowledge of all the resources that participate in the grid system. I assume that every resource has a full knowledge of the resources that participate in the grid system. This means that the id, the type and the group membership of each resource is known by each and every resource.
In a real world grid system this would of course not be the case but I have chosen this simplification for simulation purposes. Alternately a plan and subsequent events would have to be made where the presence of resources were revealed. The revealing of a resource does however occur when a new resource is added to the simulation. At the time where a new resource is introduced into a running simulation its presence has to be announced to the other resources. Again the id, the type and group memberships of that particular resource has to be known by each and every resource connected to the grid system.

- Find the most trusted resource of all the known resources of a specified type.

Occasionally a resource might need to determine which resource is the most trusted of all. For example if all sites are below a user's trust threshold, the user will have to chose the most trusted site when submitting a job. It has to chose the lesser evil. The most trusted resource of a given type is determined by :

$$\text{Trust level} > \text{Alliance strength} > \text{update time}$$

When a resource has to find the most trusted resource of a given type it will compare the trust values of its known resources. If multiple resources have an equal high trust level, then the resource with highest number of shared group memberships is chosen. This signifies that the alliance with that resource is the strongest. If multiple resources have the same alliance strength the resource with the most recent updated trust value is considered the most trusted. It might however, still be the case that several of the resources have been updated at the same time, for example in the beginning of the simulation, and in this case a random resource between the resources with highest trust value, group memberships and update time, is chosen.

- Give a rating of a resource.

A resource can at any time in the simulation be asked to give a rating of another resource. The rating will be based on the behaviour of the resource and the alliance with that resource that is seeking a rating value or the alliance with the resource being rated. If there is any alliance with the resource that is seeking a rating, the given rating will be honest. If there is an alliance with the resource to be rated the rating value will be boosted depending on the behaviour of the rater. Likewise if there is no alliances at all the rating will depend on the behaviour of the rater. This is further described in section 4.2.8.

4.2.2.1 Specific for a user

The users have some specific needs that are not required for the sites. These needs will be described here. They have have to:

- Submit jobs to a site

The job event, described in section 4.2.4, will pick a random user and propose a site. The user will then generate a job, if no jobs are in its job queue. This includes setting up the data values of the job. I have chosen the actual CPU time for a job to be a random number between 20 and 40. If the proposed site is above the trust threshold then it will be chosen. If the proposed site does not meet the trust requirements, another more trusted site might be chosen. If the proposed site is below the user's trust threshold it has to be decided if the site should be retried. By checking if the current time is above the retry time for the site, this will be decided. If the current time is below the retry time for the site then it will not be retried and a more trusted site for the job will be chosen. If it is time to retry the site this will be noted, the site will be retried and the user will proceed with submitting the job to the proposed site. In the end the user will have a job and have decided where that job should be submitted. The job will then be returned to the event who in turn submits it to the chosen site.

- Get the results from a job.

The job done event will handle the retrieval of finished jobs and the return of the job results. This event is described in section 4.2.4. When the user receives the processed job it will need to calculate a new trust value for the site based on the success of the interaction. First the direct trust value is calculated and saved in the sequence of direct trust values. Afterwards the reputation value is calculated by seeking ratings from other users. This reputation value is also saved in a sequence. These two values are then combined using either the discrete approach or the fuzzy logic approach, depending on the simulation configuration. Finally the trust value for the site is updated to the new combined trust value. The combined trust value is saved in a sequence of values. The sequences for direct, reputation and combined trust are used when the simulation results are outputted to files. This is described in section 4.2.7. It should be noted that neither the direct or the reputation trust sequences are used when the trust values are calculated or updated. Their purposes are to allow an

examination of the results of the simulation results when the simulation has ended. Likewise only the most resent combined trust value is used when calculating the direct trust value. The sequence is merely used for examination of the progression of trust.

- Receive a rejected job.

When a site decides to reject a job, the job will be returned to the job creator. This is done via the job rejected event described in section 4.2.4. When the user receives a rejected job it will calculate the reputation of site that rejected the job. If the user has a higher level of trust in the site than the reputation states, the trust in that site will be updated to to the reputation trust value. Since no job processing took place the user can't evaluate the site on that basis. Instead the surroundings will be trusted to supply a trust value. If the reputation trust value is higher than the user's trust in that site, the trust for the site will not be updated to the reputation trust value. Instead the trust that the user has in the site will decrease a little. How much will be determined by experimenting.

4.2.2.2 Specific for a site

Similar to the users the sites also have some specific needs. They:

- Have a queue for rejected jobs.

The site will store the rejected jobs before returning them to the job creator. Since it might reject several jobs these will be stored in a queue.

- Receive a job that might be processed.

When a site receives a job that it might process, the site will examine who created the job. First the trust level of the job creator is determined. If the trust level is sufficient, meaning that it is above the trust threshold, the job will be scheduled for processing. If the user falls below the threshold it will be examined if it is time to retry the user's job. If it is time for a retry, the job will be scheduled, otherwise the job will be rejected and inserted into the rejected jobs' queue. If it has been decided that a job is to be scheduled for processing the site examines its job queue. The sites use a first-in-first-out(FIFO) scheduling where jobs are processed in order of their arrival. The site determines when the job in question will be able to start. Afterwards the time that is allocated to the job processing is decided. This depends on the behaviour of the site and will be described in section 4.2.8. Then the site will specify in the job that the site will process it, and the job is inserted into the job queue. The finishing time of the job is returned to allow a job done event(described in section 4.2.4) to be generated.

- Return a processed job.

The job done event will specify that the site has completed a job. The site will then extract the job from the job queue and examine the degree of satisfaction of the job processing. The direct trust value will be calculated (according to the description in section 4.2.8) and saved to the sequence of direct values for the job creator. Afterwards the reputation trust (as described in section 4.2.9) will be calculated by asking the surrounding sites for a rating of the job creator. This value will also be saved as a sequence of reputation trust values associated with the job creator. The two values, direct and reputation trust, will be combined using either the discrete or the fuzzy logic approach. Finally the combined trust value is used to update the trust value for the given user. Again the trust value is inserted into a sequence denoting the trust progression for the user. Similar to the users, the sequences of trust value are merely used to enable an easy output of simulation results.

- Return a rejected job.

The job rejected event (described in section 4.2.4) will inform a specified site that it can return a rejected job to the job creator. The site will retrieve the job from the rejected job queue and return it.

4.2.3 Job

The job object serves as a container for data values. Within the simulation, what the job actually does is irrelevant. Since the trust in a user is based on how accurately users state the requirements, and the trust in a site is based on how accurate it fulfils the stated requirements, the jobs merely has to contain these values. Additionally it will also be helpful if a job lists the creator of the job and preferred site. Since it can not always be assumed that the preferred site will process the job, the actual site that processes the job should be noted. The table 4.1 lists the data values that should be present within a job object.

Actual CPU time	Informed CPU time	Given CPU time
Job creator	Preferred site	Actual site

Table 4.1: Job data values

Additionally it would also be convenient that the start time and finishing time of the job is listed.
I assume that resources only change the job values that they are allowed to change. However no encryption or cryptographic hash functions will be used to ensure this. Encrypting and decrypting values would course the simulation to be slower. By assuming resources only change the job values that they are allowed to change no benefits would come of using encryption.

4.2.4 Events

Although the events that can occur in the simulation are very different they do however, share the property that they must happen at a given time and the effect of the event must influence the parameters of one or several of the resources. By using inheritance this can be exploited. The figure 4.8 shows the inheritance relationship. Common to all types of events

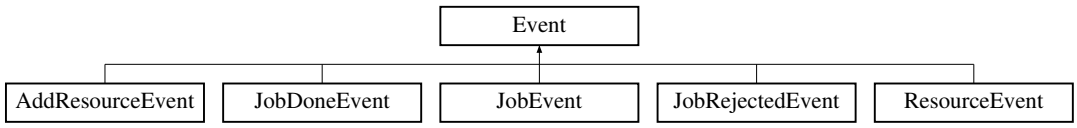


Figure 4.8: Designing events

are a value that states the time of the event. Additionally each method will have an action to perform. By declaring the event class to be an abstract class, making the action method a pure virtual function, all classes that inherit the abstract class will be forced to implement the action method. The specific actions of each type of event will now be described:

4.2.4.1 Addition of new resources

The addition of a new resource to a running simulation will be handled by an add resource event. This event will create the resource with the parameters specified in the configuration file. Afterwards the existing resources in the grid will be made aware of the new resource's presence. This means that the type of resource, the resource id and the groups that it belongs to is presented to the other resources.

4.2.4.2 Returning the results from a job

After a job has been submitted for processing the job event will generate a job done event. This event will handle the returning of the job result. The appropriate site will be contacted and instructed to return the result of the job. The job will contain the result of the processing. The job is afterwards returned to the job creator. Both the site and user will update their trust value for each other during this event.

4.2.4.3 Sending jobs to sites

It will not be predefined which user sends which jobs to which sites. A series of job events will be created and when a job event happens, a random user and a random proposed site will be chosen. This involves extracting a job from the user and submitting it to the designated site. The site will then return a point in time where the job will be finished.

This will result in the creation of a job done event. If the site rejects the job, a job rejection event will be created.

4.2.4.4 Returning a rejected job to the job creator

When a site decides to reject a user's job it will result in a job rejected event. This event will obtain the rejected job from the site and return it to the user. The user can then update its trust in the site accordingly.

4.2.4.5 Changing the behaviour of a resource

The configuration file can contain specific points in time where the behaviour of a resource is to change. This will be done by a resource event. This will find the specified resource and change its behavioural values.

4.2.5 Configuration

The specifics of a simulation should be determined by a simulation file. The purpose of the configuration class is to read the configuration file and make the values available. The configuration will be a simple scanner and parser of a text file.

4.2.6 Random number generation

Many of the other classes will have the need to obtain random numbers. The random number generator class will supply a stream of random numbers. There will also be the need to obtain random numbers from a Laplace (also called a double exponential) distribution, as will be described in section 4.2.8.

4.2.7 Gathering of results

When the simulation has ended the outcome of the simulation will be outputted to a specified location. The event handler will handle this by an output method that will contact each and every resource in the grid and ask for their sequences of combined, direct and reputation trust values. This is the history of trust all of which will be written. To simplify the subsequent analysis of the results, a directory should be created where all of the data is written. I have chosen that the name format for the directory should reflect the time of the simulation. Additionally the combination method, fuzzy or discrete, should also be reflected in the name and the reputation system, beta or discrete likewise. A suitable format would be *xy-month-date-time* where x denotes the combination method and y is the reputation system.

4.2.8 Behaviour

Simulating good and bad behaviour is not easy. It is however needed when a resource has to decide how much time should be allocated/informed in connection with a job. Additionally the behaviour of a resource will also determined by how accurately it gives ratings of other resources. It should be noted that the proposed techniques for simulating and determining behaviour are heuristics.

Informing and allocating CPU time

The two behavioural values min and max will be used to determine a resource's behaviour. When a user has to determine how much CPU time should be informed that a job will require, it will be based on a random number. The same will be the case when a site has do decide on how much time to allocate based on the informed CPU time requirement. The random number will be drawn from a Laplace distribution. This follows the equation $e^{-|x/width|}$ where the centre of the distribution is zero. The two behavioural values will be used to determine the width of the distribution. High behavioural values produces a wide distribution and subsequently a higher probability that the number drawn will be further from the centre of the distribution. A slimmer distribution will result in a higher probability that the number drawn is closer to zero. The absolute sum of the absolute behavioural values, min and mix, will be used to determine how much time will be allocated by the site or how accurate the informed CPU time by the user us. When Laplace(x) denotes a random number drawn from a Laplace distribution with the width x the informed and allocated time can be calculated by:

- Informed CPU time = actual CPU time + Laplace(|min|+|max|) - min + max
- Allocated CPU time = informed CPU time + Laplace(|min|+|max|) - min + max

The values min and max can be used to shift the time in a desired direction. For example a large positive max value and a very small positive min value could be used to simulate that a user always overstates the requirements for a job. For a site it would however mean that it always allocated more time than was needed. When the absolute values of min and max are equal no shift will be performed.

Measuring behaviour

A user will measure the behaviour of a site by the difference (ΔCPU time) between informed CPU time and allocated CPU time. A site will measure the behaviour of user by the difference(ΔCPU time) between informed CPU time and actually needed CPU time. When using a trust interval of [-1:1] and using a smoothing factor s, a trust value based on interaction i can be calculated by:

$$\text{Trust}_i = 4/(e^{\Delta\text{CPU time}_{i-s}} + e^{-\Delta\text{CPU time}_{i-s}}) - 1$$

This is the function described in section 4.1.1. The smoothing factor should be defined in the configuration file and the purpose is to limit or enhance the effect of the difference. For example if the smoothing factor was 1 and the difference was 2, the trust value calculated would be approximately 0.45. If however the smoothing factor was 0.5 but the difference still was 2, the resulting trust value would be -0.12. By setting an small smoothing factor is becomes possible to simulate scenarios where deviations are harshly judged. With a larger smoothing factor a relaxed scenario can be simulated.

Rating resources

The simulation must implement the ability to give inaccurate ratings. This means that the resources must be able to lie. This again is not an easy task. However, by limiting the possibility that resources can only rate the opposite type of resources, a large part of the motivation for lying can be disregarded as discussed in section 3.5. A resource's motivation for lying would simply be the need to help the alliance partners. Or in the case where the resource seeking a rating is an alliance partner, an honest rating should be given. When dealing with complete strangers the given rating should however be directed by the rater's behaviour. When the allocated/informed CPU time was determined a Laplace distribution of random numbers was used. Similarly when a resource has to supply a rating a Laplace distribution will also be used. When there is an alliance with the resource to be rated, the actual rating will be boosted by random positive number from Laplace(|min|-|max|) divided by a constant c_1. When there are no alliances the rating might be boosted or lowered. This can be done via a random number from Laplace(|min|-|max|) divided by a constant c_2. The behaviour can can be summarise to the following:

- Alliance with the resource asking for a rating : honest answer
- Alliance with the resource to be rated : Actual trust value + |Laplace(|min|-|max|)|/c_1
- No alliances: Actual trust value + Laplace(|min|-|max|)/ c_2

Since a resource would want to boost the trust of its alliance partner but not damage the reputation of itself and the alliance partner, the boosting of the rating should not be a huge overstatement. When there are no alliances the resource giving the rating has no real motivation for lying. The random number from the Laplace distribution will simply denote an inaccuracy when rating another resource. It can be interpreted as an "I don't really care" answer. Because of this it could be argued that $c_1 > c_2$ since it would mean that a resource would lie more when giving ratings to strangers about strangers than when giving ratings to strangers about an alliance partner. By lying too much about an alliance partner it could damage the reputation itself and the alliance partner. When dealing with two complete strangers this will not be a major concern of the rater. It does not really care about either of the resources. The actual values of the constants have to be determined via experiments.

4.2.9 Reputation methods

The simulation should provide a discrete model and the beta reputation model for combining the ratings to a reputation trust value. The discrete approach presented by Farag Azzedin and Muthucumaru Maheswaran in section 2.4.2 weighs the different rating by the trust that the resource has in the rater. Since a user asks other users for ratings on sites, and users don't have a trust value for other users, ratings can't be weighed according to the trust in the rater. The same naturally applies for sites. Instead I propose weighing the rating based on the alliance strength. For example if a resource x asks two resources y, z for a rating of another resource. If resource x has an alliance strength of three with resource y, but only an alliance strength of one with resource z then the y rating would be weighed higher. In the given example the combination of the ratings would be:

$$\text{Reputation value} = \tfrac{3}{4} \text{ rating from } y + \tfrac{1}{4} \text{ rating from } z$$

The alliance strength of a resource divided with the total alliance strength in the resources that are giving ratings will function as a way of weighing the raters according to trustworthiness. I have also chosen that every resource has at least one alliance with every resource. Meaning that the minimum alliance strength is one. This basis alliance is an alliance that exists between all the resources and it symbolises a grid system bond.
A similar approach to weighing the raters will work with the beta reputation system proposed by Audun Jøsang. However a base weight will be used for ratings. In section 2.3.3, where the beta reputation system is described, the base weight is referred to as w. A rater will have the weight: alliance strength * w. The base weight will be found through experiments.

4.2.10 Fuzzy logic

Fuzzy membership functions and a rule base will be needed for the fuzzy logic combination of direct and reputation trust. Chapter 4 of [27] presents various design methods that can be used when designing fuzzy membership functions. However, none of the methods seem very applicable in this case. A suitable starting point for the membership functions is presented by Kai Hwang and Shanshan Song in their papers[25, 26]. I plan to base my functions on theirs.
I also plan to use the same membership functions for direct, reputation and combined output trust. A fuzzy rule base also has to be designed. I propose using a scheme where, when possible, direct and reputation trust should be weighed equally. When it is impossible to chose a fuzzy output set that is between the direct and reputation trust input sets, then the direct trust set will have a slight preference.
Kai Hwang and Shanshan Song use the min-max inference method and I plan to use the same. A defuzzification method also has to be chosen. As described in section 2.5 the two most common methods are the Center of Gravity (CoG) and Mean of Maximum (MoM). In this particular case CoG would preferred over MoM method. Using the MoM method

the mean of the maximum value of the fuzzy output set, where the membership was the strongest, would be returned from the defuzzification process. This is very desirable when the fuzzy logic is used in connection with an application that has to decide between performing a finite number of actions. With the CoG method, combined fuzzy sets would be used to give a value where the membership is the strongest. Hence the defuzzyfied crisp value from the aggregated output set would not be restricted to the number of membership functions. For this reason the CoG method is preferred.

4.3 Implementation

The design of the simulation has been implemented in C++ under the Linux environment and appendix C show the flow of control. The class diagrams in appendix D gives an overview of the implementation and the API in appendix E, generated by the program doxygen, give a more detailed description of the various methods and variables used by the classes.
The implementation relies on the data containers such as maps,vectors and dequeues and various algorithms provided by the standard template library. Random number generation is done with the use of the GNU scientific library(gsl - http://www.gnu.org/software/gsl). However since this is a C library a C++ wrapper class, called RandomGenerator, has been made. The gsl random number generator has been instantiated statically, meaning that it is shared, to ensure that random numbers come from the same generator. The default gsl random number generator type is used.
The configuration of the simulation will be loaded from file called "configuration.conf". This file must be present in the same directory as the grid simulator program.
For fuzzy logic operations the Free Fuzzy Logic Library (FFLL - http://ffll.sourceforge.net). FFLL uses a configuration file specified in the Fuzzy Control Language for the specifications of membership function, rule base, inference method and defuzzification method. The details of this language is specified by the International Electrotechnical Commission in the FCL draft 1.0 [29]. The program will load the file "resource.fcl" that must be present in the same directory as the program. FFLL use the parent-child principle. The following shows how the fuzzy logic model is setup and used.

```
#define DIRECT_TRUST    0 // Direct trust is the 1st variable
#define REPUTATION      1 // Reputation trust is the 2nd variable
//Create model
int model = ffll_new_model();
//Load fcl file
int ret_val = (int)ffll_load_fcl_file(model, "resource.fcl");
//Check if file was loaded properly
if (ret_val < 0)
    throw string("error opening resource.fcl file");
// create a child for the model...
int child = ffll_new_child(model);
```

All of the resources use the same model but the child is however unique to each resource. The fuzzy calculations has been done by:

```
/* A direct trust value (dt) and a reputation trust
 * value (rep) need to be combined using fuzzy logic. */
ffll_set_value(model, child, DIRECT_TRUST, dt);
ffll_set_value(model, child, REPUTATION, rep);
//Retrieve value
float combinedTrust = (float)ffll_get_output_value(model, child);
```

The use of FFLL did however present a number of difficulties. The program homepage stated that the program was written to abide with the POSIX standard. Unfortunately that was not the case, and the examination of the source code revealed the use of specific Microsoft compiler options and methods. Fortunately I found the web page http://wojci.dk/FFLL-linux where FFLL had been ported to Linux. The port was done of the 2.1.2 version of FFLL and the current version of FFLL is 2.2.1. Also the ported version was fixed to use an older version of the GCC compiler and deprecated header files. By editing the source code and the Makefile I was able to compile a working version. The next section will describe some additional difficulties involved with the use of FFLL.
The source code for the implementation can be found on the CD-ROM. The code contains comments that describe the implementation in greater detail.

4.4 Testing

Testing a simulation is quite hard, since it involves the use of random numbers. It would be difficult to verify that, given the input, the expected result was received as output from the simulation. Instead the testing has been done by investigating the individual methods of the classes. Especially the borderline cases for each method has been tested.
The first thing that has been done to eliminate the possibility of undetected errors is by a try-catch statement in the main method of the program. This is combined with an amount of abundant information being passed to the methods. If at any point an unexpected occurrence is observed an exception is thrown and the program terminates. For example when a job done event is created, the user's and site's ids are included in the constructor of the job done event. The job also contains the id of the user and the id of the site. When the event occurs the ids of the user and of the site are matched to what is stated in the event. If for some reason the site calculated the finishing time of the job incorrectly, these ids will not match. If this is the case an exception is thrown.
In the early stages of development the simulation also outputted information about what was occurring. For example when a user submitted a job, it was displayed how much CPU time was required and informed. When the site received the job it was also displayed how much time was allocated and when it expected to finish the job. When the job was finished the actual calculations where displayed. For a series of simulation runs, this output was logged and examined for correctness.

As shown in the API, several of the classes also contain methods that output information. These were used in the testing period. When testing the borderline cases, events where manually created and the methods that outputted information was invoked. For example when testing the group memberships of the resources this was manually setup and by listing the grid it was directly visible that the correct group memberships where claimed by resources and that the received ratings were correctly weighed.

The Linux port of FFLL that I have used, is based on the version 2.1.2. I discovered that the FFLL parser, used to read the fuzzy membership functions, rule base, inference and defuzzification method from the fcl file, depended on the DOS newline character. A Unix formatted file would no be parsed. However with the use of the utility "unix2dos" a Unix text file can converted to a DOS formatted file.
Upon testing the borderline cases for the fuzzy logic method a slight error was observed. For example if the defuzzified crisp value was supposed to be 0.8 it was instead 0.797997. When I investigated the ChangeLog between the versions 2.1.2 and the current version I discovered that a small error in the defuzzification method Centre of Gravity had been fixed. I tried patching the ported version by editing the source, but the code base was changed significantly between the versions. I saw three alternatives to fix the error:

1. Compile the simulation in a Windows environment with the current version of FFLL, and perform all simulations there.
2. Port the current version of FFLL to linux.
3. Abandon the use of FFLL and make my own fuzzy logic library.

I did not find the first alternative particular attractive, because the subsequent data analysis of simulation results would need to be done by a number of scripts. I am not very familiar with doing scripts in the Windows environment, but it is my experience that there are better possibilities in the Linux environment. After viewing the number of changes needed to port the version to Linux I did not find the second alternative appealing for time considerations. Since FFLL included a very flexible setup via a standard notation fcl file the last alternative was also unattractive. Since neither of these alternatives were desirable and since the error was not significant I instead chose to ignore it. I do not consider a difference of $0.8 - 0.797997 = 0.002003$ to be large enough to change the outcome of a trust based decision. If for example a resource had a trust value 0.8 which was considered high enough to allow access, a value of 0.797998 should also be high enough for access to be granted.

4.5 Summary

This chapter has described analysis, design, implementation and test of a program for the simulation of trust progression in a grid system.
In the analysis I described what I consider the most relevant information and types of occurrences in a grid system when the focus is on trust progression. Also the features that

the simulation should include were listed. With the focus on a grid system I analysed how varying behaviours should be simulated and determined. Additionally the simulation of a resource rating other resources was described. A design section detailed the individual objects needed in the construction of a discrete event simulation. This also included quantifiable parameters for measuring behaviour leading to calculation of trust. The design section was followed by a brief description of the actual C++ implementation. I intentionally made the implementation description very brief, only focusing on the use of the Free Fuzzy Logic Library and made references to appendices containing class diagrams and an API. The specifics of the simulation can be found by viewing the accompanying source code on the CD-ROM. The testing section dealt with the efforts I have made to ensure the correctness of the simulation. The test mainly consisted of borderline cases being examined. This was combined with an implementation where abundant information is passed to methods used to ensure that miscalculation would be detected. An error in The Free Fuzzy Logic Library was discovered in the testing phase.

Chapter 5

Simulation experiments

The purpose of making a grid simulation is to examine a number of scenarios where trust is involved. With the specification of simulation parameters and a subsequent data analysis, the progress of trust has been documented. This chapter will list the efforts of my experiments. First the simulation parameters are given with a description of their effect in an experiment. This is followed by an outline of the simulation strategy that details the most interesting simulated scenarios. Afterwards the initial setup of the simulation with default simulation parameters is described. Finally the experiment results are presented and discussed.

5.1 Simulation parameters

The outcome of the simulation is determined by a number of parameters. The parameters are specified in the configuration file and each parameter can be used to influence an experiment to reflect a scenario. In this section I will list the possible parameters with a short description of their influence. The actual parameter format and configuration syntax is specified in the User Guide in appendix A. The simulation parameters are:

- The number of users and sites.

This specifies the initial number of resources present in the grid. Both can be specified independently, but the minimum number of each type of resource has to be one.

- The behavioural values (min and max) for each resource.

The behaviour of each resource is dictated by two integer values. If a set of values are directly specified for a resource these are used. Otherwise the default set of values are used. As described in the design section 4.2.8 the two values determine the width of the Laplace distribution where random numbers are drawn. Small positive values gives a "thin"

Laplace distribution which will result in a higher probability that the resource informs or allocate the correct amount of required CPU time. High positive values will result in a higher probability that the resource will inform or allocate an incorrect amount of CPU time. It does however not imply that the resource will always inform or allocate incorrectly. Having a negative max value that is numerically higher than a positive min value would result in a case where the resource always informs or allocates too much time. Having a negative min value that is numerically higher than a positive max value would result in a case where the resource always informs or allocates too little time.
The behavioural values will also influence the ratings given by the resource. Small values will result in a high accuracy when providing feedback. Higher values result in a higher accuracy. Put in another way with higher values the resource's lies will be greater.

- The group membership of each resource.

Each resource can claim membership to a number of groups. If two separate resources are members of the same group, they share the group, and that will indicate an alliance between the resources. The greater the number of shared groups, the stronger the alliance. When two resources have an alliance, it means they will send honest ratings of other resources to each other. Consequently the ratings given by alliance partners will have additional weight when reputation values are calculated. Also they will, if asked about the other resources, boost the rating with a degree that depends on their own behaviour. With high behavioural values they will boost the reputation of alliance partners more. The alliance partners of a resource might not have a high trust value, but alliance partners do however always rate resources truthfully and boost the rating of alliance partners.

- The trust threshold for each resource.

Each resource can have a trust threshold. Unless directly specified in the configuration file the default value will be used. If the trust value for a resource falls below the threshold, it does not mean that the resource is excluded forever. Instead it is retried after some time. If the resource hasn't improved its behaviour, even more time has to pass before it is retried again.

- The change(s) in behaviour at specified time(s).

For every resource it's possible to specify a given time where the two behavioural values will change. For example the behaviour of a resource can change from a high probability of good behaviour to a high probability of bad behaviour. This can be used to simulate that for example starts to malfunction or that it is compromised.

- Addition of a resource or multiple resources at specific time(s).

By adding a resource during the simulation, it can be examined how a running trust scenario perceives the arrival of a stranger. Naturally it is also possible to specify the behavioural values, trust threshold and group membership values for the arriving resource.

- The simulation time.

Since trust is a slow varying factor it is important that the duration of the simulation is sufficient. The configuration file allows for an arbitrary simulation length.

- The lambda for the Poisson distribution for generation of events.

A Poisson distribution is used for generation of job events. Lambda is equal to the expected number of occurrences that occur during the given interval. For example if the simulation time is 100000 and Lambda is 25 the approximately 4000 job events will be generated for the experiment. Lambda combined with the simulation time can be used to control the amount of activity in the grid.

- The default trust value assigned to strangers.

It is possible to set the initial trust value of strangers. A negative value will indicate a paranoid grid system, while a positive value will imply a grid system where resources are trusted until proved otherwise.

- The direct weight of an experience.

The interaction between resources will result in the calculation of a trust value based on the experience. This will be combined with the previous (combined) trust in the resource to calculate a direct trust value. It follows the formula where α is the direct weight:

Direct trust = Previous (combined) trust value + α * trust value from the interaction

A high α value will result in a higher influence of the outcome of each interaction.

- The smoothing factor when calculating the trust from an interaction.

The difference between informed/assigned CPU time and actual/informed CPU time respectively is used to calculate a trust value. Using the difference x and the smoothing factor s, the trust value from an interaction is calculated by the formula :

$$4/(e^{x-s} + e^{-x-s}) - 1$$

The smoothing factor s is used to limit or enhance the effect of the difference in the trust value calculation. Or put in another way a larger smoothing factor will result in greater leniency.

- The fuzzy logic or the discrete combination of direct and reputation trust.

The method used for combining direct and reputation trust can be directly specified in the configuration file. This can be used to compare the two methods.

- The ratio between direct and reputation trust when using the discrete method.

When the combination is done using the discrete method, the ratio between direct and reputation trust needs to be specified. It is possible to base the trust value solely on direct trust or solely on reputation information or somewhere in the middle using this factor.

- The fuzzy membership functions and rule base when using fuzzy logic combination of direct and reputation trust.

An fcl file specifies the fuzzy membership functions and the rule base. This file must be present in the simulation directory.

- The beta or the discrete reputation system

The reputation system can either be the beta reputation system, as described by Audun Jøsang in section 2.3.3 or a discrete reputation as described by Farag Azzedin and Muthucumaru Maheswaran in section 2.4.2.

- The forgetting factor when using the beta reputation system.

According to the description of the beta reputation system in section 2.3.3 it is possible to use a forgetting factor that gives higher weight to newer ratings. The forgetting factor is a floating point value in the [0,1] interval. A forgetting factor of zero means nothing is forgotten and a forgetting factor of one means all but the most recent ratings are forgotten.

- The random number seed.

The final simulation parameter is the seed value for the random number generator. This will be directly specified in the configuration file or as a command line argument when running the simulation. If a command line argument is present, it takes preference over what is stated in the configuration file. This makes it significantly easier to repeat simulations using a shell script combined with a sequence of seeds. When a simulation is repeated with the same seed all events are the same. By fixing the random number seed and varying a simulation parameter, it is possible to examine the effect of a parameter where the conditions are unchanged.

5.2 Strategy

As described in the previous section there is a number of parameters that can be varied. It is essential to keep many of the parameters constant while varying a single parameter. Otherwise interpreting the results will be very difficult. Consequently a strategy for which parameter to investigate is needed. This section will outline the strategy.

The main goal of simulating trust in a grid environment is to examine whether or not trust can be used as a factor for access control. In order for that to be the case the trust management system must exhibit a high degree of stability but at the same time be flexible enough to handle continuous changing conditions.
Several of the models presented in chapter 2 included simulation results. The authors of the models described a simulation scenario and given suitable parameters, results were presented that for most cases provided some evidence of the accuracy of the model. However none of the models, except the Bayesian network based model described in section 2.4.3, compared their model with other models through simulations. If the model used discrete calculation methods, only the discrete methods were examined in the simulation, and likewise if the model used Bayesian or fuzzy logic methods. However, in my simulation it is possible to simulate the same scenario using either the discrete or the fuzzy logic combination method. And in both cases either a discrete reputation system or the beta reputation system could be used. A goal of the simulation is therefore to examine the advantages and disadvantages of each method and reputation system.
For a trust system to be efficient in a grid system, it should be able to cope with a number of scenarios. If for example the trust system breaks down when a resource changes its behaviour it would be useless in a real world grid system. There are some scenarios that are almost certain to occur in a real world grid system. It would however, be almost impossible to simulate all of these scenarios. Instead, what I consider the more frequent occurring scenarios will be examined. This includes a grid system where :

- Every resource exhibits good behaviour.
- One or more resources exhibits potentially bad behaviour.
- The behaviour of a resource changes.
- A resource joins a grid systems.
- A resource protects itself by having a trust threshold.
- A resource protects itself by having trusted alliances.

These five scenarios are very general. It is for example not described when and how much the behaviour of a resource could vary. When presenting the results of the experiments I will give a more detailed description of what, when and to what degree the events occur in the scenario.

5.3 Simulation setup

As with any simulation it is necessary to define some initial values for the simulation parameters that will serve as default values. Not all of these defaults can be defined via a mathematical analysis. Some has to be based on experimenting, or put in another way

they must be heuristically defined. When the default values have been established, it is possible to vary each value independently and observe the effect. I have done this in the early stages of experimenting and in this section I will list, and when possible argue, these default parameter values.

5.3.1 Discrete models

Discrete combination of direct trust and reputation is the most commonly used method. Several authors have simulated models using this method and their experiments present a good starting point for choosing a default value. In my simulation there are one specific factor that to a high degree determines the outcome when using the discrete method for combining direct and reputation trust. This is the ratio between direct and reputation trust determined by a value between 0 and 1.
The model presented by Farag Azzedin and Muthucumaru Maheswaran in section 2.4.2 relies on a similar factor to weigh direct and reputation trust. Their simulation experiments described in [20] use the ratio $1, 0.5$ and 0. When the factor is 1, only the direct trust information is used and when the value is 0.5, direct and reputation trust are weighed equally. When the ratio is 0, direct trust is disregarded leading to a full reliance on the reputation trust. The TRUMMAR model from section 2.3.1 also uses a similar ratio between the agent's own information and the reputation information obtained from other agents. The simulation documented in [9] uses the ratio 0.55, meaning that an agent's own experience is weighed slightly higher than the reputation trust.
I have chosen 0.5 as the default value for the ratio, as it causes direct and reputation trust to be weighed equally. This default ratio does however not mean that it is the correct ratio in every scenario. In certain scenarios it might be more suitable with a different ratio, but for practical purposes 0.5 is used as default.

5.3.2 Fuzzy logic

The use of fuzzy logic depends on the definition of fuzzy sets, membership functions and a fuzzy rule base. Kai Hwang and Shanshan Song used fuzzy logic to combine a job success rate and an intrusion detection system value to a trust value. I will however combine direct and reputation trust and obviously the terms are not directly comparable. I have chosen to use a slight modification of their outlined membership functions. Their membership functions shown in [25] use five overlapping fuzzy sets called "very low", "low", "medium", "high" and "very high", and are all defined using sigmoid functions. Since defining sigmoid membership function for the Free Fuzzy Logic Library can only be done by listing the data points that make up the curve I have instead chosen a linear outline for the membership functions. Figure 5.1 depicts the membership functions and the direct, the reputation and the combined output trust all share the same outline of the membership functions. It is worth noticing that the size of the sets, represented by the area of the membership function, is the same. For example the area of "very low" is $15 * 1 + 10 * 1/2 = 20$ and the area of "high" is $10 * 1/2 + 10 * 1 + 10 * 1/2 = 20$. Kai Hwang and Shanshan Song chose that the

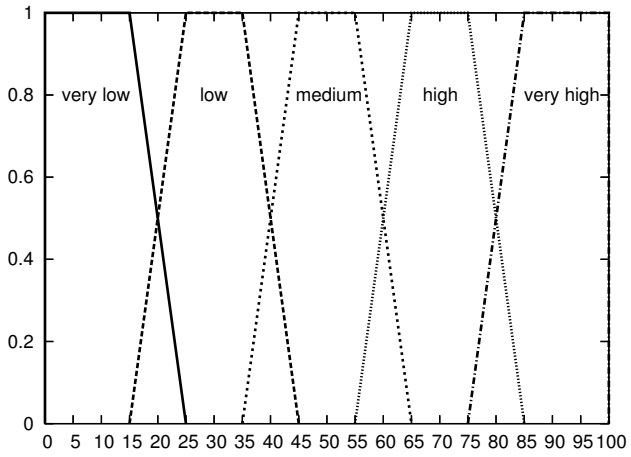


Figure 5.1: Fuzzy membership functions

fuzzy sets "very low" and "very high" were roughly half the size of the other sets. I, on the other hand can not find a compelling reason why this should be the case. For example I do not see a reason why the set of "very high" trust values should be smaller than the set of "high" trust values. Even if a very small amount of actions would lead to a rating of "very high", and a slightly bigger amount of actions would result in a "high" rating, this doesn't necessarily mean that the set of values that describes the sets should be different. Or put in another way: It should be harder for a resource to obtain a "very high" trust level than a "high" trust level, but that does not imply that there should be fewer values denoting "very high" than there should be denoting "high". The ramifications of having unequal set sizes would also give rise to questions like: how big should the size of the "medium" set be in contrast to the "low" or the "high" set? Should the size of the "low" set be equal to the size of the "high" set? I have intentionally avoided this by making the fuzzy set sizes equal. In figure 5.1 the x-axis is set to [0,100] however, the interval of possible values for direct and reputation trust is [-1:1]. The x-axis for the membership functions were set to [0,100] simply because the fuzzy logic library used did not allow for negative value. This means that before fuzzifing the crisp direct and reputation trust values, they have to be change to a [0,100] scale. This is easily done by multiplying the crisp value with 50 and afterwards adding 50. When the crisp output value is obtained it is divided with 50 and one is subtracted.
There is one very important aspect of using the centre of gravity defuzzification method in connection with the member functions I have specified. When combining the trust values the output interval is reduced to [-0.8:0.8][1]. The reason is best explained with an example. Suppose both direct and reputation trust are "very high". Because of the defined rule base and the used inference method, this would be aggregated to a fuzzy output set equal to the set "very high". When defuzzifying this set with the centre of gravity method, the crisp

[1]Actually to [-0.797997,0.797997] because of the calculation error in the Free Fuzzy Logic Library as described in section 4.4

value would be 90, since 90 is the centre of gravity. When changing the scale back this would result in the value: $90/50 - 1 = 0.8$. Likewise for the set "very low", where the centre of gravity is 10 and the change in scale would result in $10/50 - 1 = -0.8$. By changing how the [-1:1] interval was scaled to the [0:100] interval this could naturally be avoided. For example if direct and reputation trust are both 1 and when changing the interval by $1 * 40 + 50 = 90$. Again both values are "very high" and the centre of gravity of "very high" is 90. When changing it back it could be done by $(90 - 50)/40 = 1$. This would enable the [-1:1] interval to be used for the combined trust values. However doing it this way presented a conceptual problem, since it in effect would reduce the size of the fuzzy sets "very low" and "very high" to half. As described earlier having fuzzy sets of unequal sizes would conceptually present a problem. Consequently I decided that the interval [-0.8:0.8] would be used for the combined trust values. It can also be noted that it does not matter which intervals of trust are used. Conceptually the notion of direct, reputation and combined trust are three separate notions. It is merely practical that the intervals are similar when displaying the results. However, it is important to remember that a direct trust value of 0.5 is not the same as a reputation trust value of 0.5. The direct, reputation and combined trust value naturally depend on each other, but the intervals used are independent of one another.
Since I plan to compare the fuzzy logic with the discrete method for combining direct and reputation trust I have reduced the interval for combined trust using the discrete method to [-0.8:0.8]. If the value calculated by combining direct and reputation trust is above 0.8, the combined trust value will be 0.8. If it's below -0.8 then -0.8 is returned. This makes the discrete method comparable with the fuzzy logic method.
It should also be noted that since the direct trust value is based on the previous combined trust value it would make it nearly impossible for the direct trust value to reach 1 or -1. However, this was already the case when using the exponential function for calculating direct trust. It could also be noted that a direct trust value of 1 would in fact render a direct trust notion meaningless as described in the analysis section 4.1.2.

Kai Hwang and Shanshan Song did not list their rule base or the defuzzification method that they used. As described in the design, I will use the centre of gravity method for defuzzifying the aggregated fuzzy set to a crisp value. The default rule base shows a slight preference of direct over reputation trust. If for example direct trust is "low" and reputation trust is "medium", the output trust will be "low". However if direct trust is "very low" and reputation trust is "medium" then output set will be "low". Figure 5.2 shows examples of the rules. As previously described in the design section the min-max inference method is used. The fuzzy membership functions and rule base can be found in appendix F as a fcl file that is used as default.

5.3.3 Heuristic values

Other authors have used a discrete ratio between direct and reputation trust, and the use of fuzzy logic for trust calculations has also been investigated. When it was possible I have relied on other authors' knowledge. My simulation does however, contain additional parameters. These values have been defined via experimenting. In the early stages of

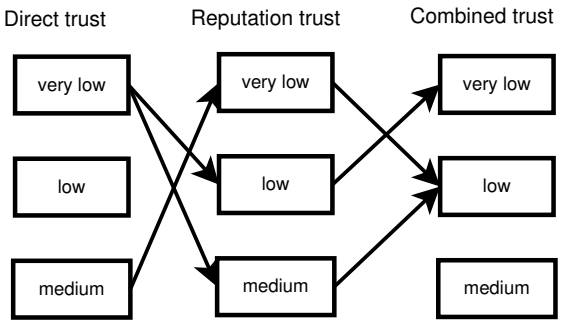


Figure 5.2: Example rules

Smoothing factor	1
Direct Weight	0.6
Stranger value	0.0
Forgetting factor	0.4
Poisson distribution mean	25

experimenting, various combinations of smoothing factors and direct weights was tried. The use of a smoothing factor of 1 and a direct weight of 0.6 seemed to give good results. For this reason the experiments documented here use these parameters. The initial value to give to strangers was also set to 0.0. This means that a neutral trust value is 0.0. By changing this value to a higher value a more paranoid grid system could be simulated. I however, chose a neutral value.
Audun Jøsang experimented with the forgetting 0,0.99 and 1 in the papers [12, 13]. The papers did however not describe at which rates events occurred, which meant that the conclusions of the experiments could not be applied in my simulation. A forgetting at 0.4 seemed to work well in my simulation. The experiments performed did also not include their base weight when combining ratings. The base value 40 was hard coded into the simulation since it also provided suitable results.
For the mean of the Poisson distribution, 25 is chosen. When it was desired that there should be more job events in the simulation I merely increased the simulation time. The two constants c_1, c_2 mentioned in section 4.2.8 denoting inaccuracy when rating resource have been set to 50 and 100 respectfully since they provided suitable results. The values indicate that the inaccuracy when rating strangers will be greater than the exaggeration when rating alliance partners. When a user receives a rejected job and its trust in the site is below the reputation of the site then the user's trust in the site will decrease by 0.1.
When investigating the results of an experiment, I found it useful in most cases to limit the number of users to four and the number of sites to four as well. Although a real world grid system naturally consists of more than four users and fours sites, I find a total of eight resources in the beginning of the simulation significant enough to obtain meaningful results.

When for example, a site seeks the reputation of a user the remaining three sites will be asked to rate the user in question. Had the grid consisted of 20 or more sites a strategy would be needed to determine which sites would be asked to rate the user (if no alliances existed). If the reputation was based on a significant amount of ratings the communication overhead would grow and consequently it would lead to an inefficient grid system. When the initial total number of resources is eight the process of rating other resources will not degrade the grid system severely. Additionally the examination of experiment results will not be overwhelming.

5.4 Interpreting the results

The simulator outputs data files that contain two columns of data. The first column contains the time of the event and the second contains the trust value calculated based on the event. For each user a data file for each site will be outputted and likewise for the sites. The combined, direct and reputation trust will also be outputted in three separate files. Consequently the number of output files can be calculated by :

(Number of users * number of sites * 2) * 3

A single simulation run of the scenario with 4 users and 4 sites will result in 96 separate data files. Manual investigation of this amount of data is hardly practical. Instead a series of scripts will be used to process the data into a form that allows for easier interpretation. I have chosen to primarily present the data using plots generated by the program gnuplot (www.gnuplot.info). The plots will depict the progression of trust, be it combined, direct or reputation trust, over the simulation time period.
Often I will repeat an experiment a number of times but with different random number seeds. The interpretation of this data will be done by combining the data files, sorting them and subsequently plotting a Bezier approximation curve. For example, user *u0*'s event times and direct trust values for site *s0* will be combined to a single file that is sorted and plotted with a Bezier approximation curve. The curve will depict *u0*'s perception of the direct trust progression for *s0*.
Unfortunately gnuplot will not list values that describe the how well the Bezier approximation curve fits the data. Using the program Maple, I tried to perform the approximations and various spline interpolations on moving averages. The goal was naturally to obtain values that described the quality of the curve fit and determine if an interpolation yielded a better fit. However, the use of Maple resulted in a significant increase in computational time. None of the interpolations or approximations depicted a significantly better fit than what was obtained when using the Bezier approximation in gnuplot. For practical purposes gnuplot has been used and the quality of Bezier curve approximation has been done by generating a plot depicting the data points and the curve approximation. A few examples of this will be shown.
The enclosed CD-ROM contains the various scripts I have used for the generating of plots

and examining of data files. Additionally the accompanying CD-ROM contains all the plots from all experiments.

5.5 Results

In this section I will describe the experiments that I have made. In all of the scenarios a number of users and sites will be present. All of the resources are identified by a unique id. The first letter of the id denotes the type of resource, *u* for user and *s* for site. When referring to a specific resource I will do so by its unique id. For example site *s2* will be referred to as simply *s2*, and user *u0* will be referred to as *u0*. The syntax of their id will denote the type of resource, hence the terms resource, site or user will often be omitted.

5.5.1 Comparing discrete with fuzzy logic combination

The first experiment will compare the two combination methods. Direct and reputation trust can be combined to a single trust value using either the discrete or the fuzzy logic method. In both cases the discrete reputation system will be used.

5.5.1.1 All resources exhibit good behaviour

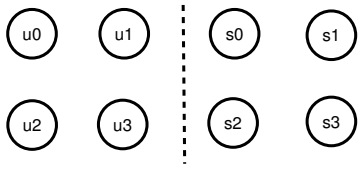
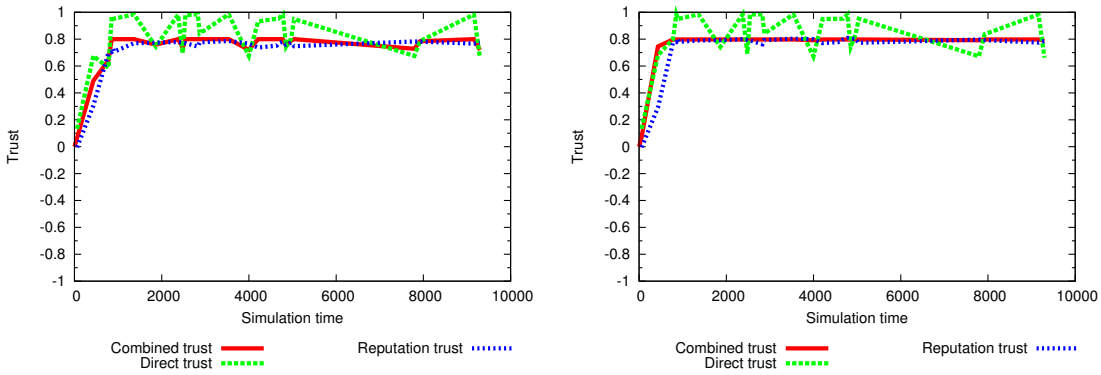


Figure 5.3: Scenario for experiment 1.1

Figure 5.3 illustrates the simulation scenario. A circle denotes a resource and the letters within the circle denotes the resource's unique id. The colour of the circle denotes the resource's behaviour, and the colour white expresses good behaviour. In this scenario all four users and all four sites exhibit good behaviour. Using the configuration file from appendix G.1 the scenario has been simulated. Figure 5.4 illustrates the results of the two experiments. In both cases the progression of direct, reputation and combined trust is plotted. The figure illustrate *u0*'s trust experience of *s0*. Appendix H.1 contains the plots for all resources when using the discrete combination method, and they all reflect the same experience as *u0*. The site's trust progression of users are also similar to what is depicted in the figure.
As specified in the configuration file, the discrete approach weighs direct trust and reputation trust equally (ratio is 0.5) and the fuzzy logic approach uses the default membership



(a) Discrete-Discrete

(b) Fuzzy-Discrete

Figure 5.4: Comparing combination methods 1

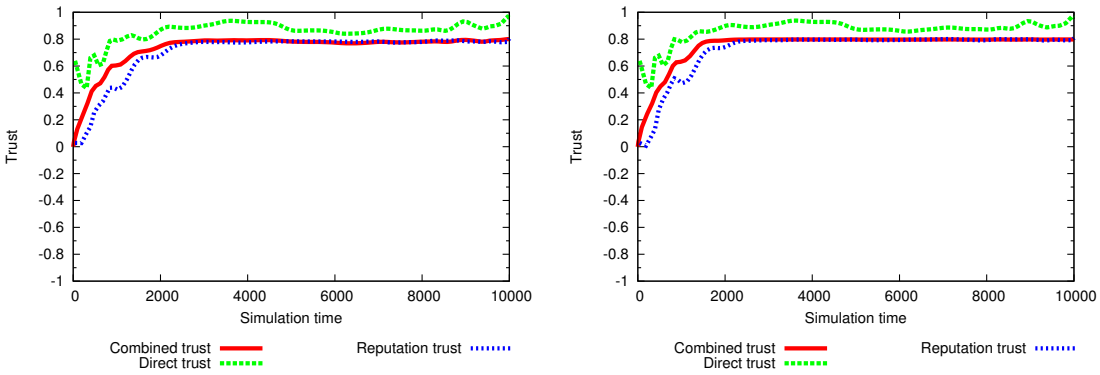
functions. Using either of the methods results in a similar progression of trust. The combined trust does not reach the value 1, but the highest value is 0.8 as expected. It should be noted that the same random number seed is used for both experiments (discrete and fuzzy logic). This means similar events occur and the resources behaviours are exactly the same for each simulation.
To exclude the possibility that the results from the experiment is an exception rather than an accurate depiction of events, the experiment has be repeated 20 times[2] for each combination method, with the random number seeds being [100,110,...,290]. Figure 5.5 contains Bezier approximation curves from the 20 repetitions. Again it is *u0*'s trust experience of *s0* that is depicted. The figures illustrates that the outcome of the single experiment was not an exception. The 20 repetitions reflects the same progression of combined, direct and reputation trust that was noted in the single experiment. It appears that when using the fuzzy logic method a higher trust value is obtained slightly faster than when using the discrete method. The "examine" program, that can be found on the enclosed CD-ROM, can calculate the average number of events needed for the trust value to be above (or below) a specified value. The table contains the average number of events needed for the trust value

Combination method	Combined trust	Direct trust	Reputation trust
Discrete	3.6	1.65	4.35
Fuzzy	2.85	1.65	3.55

Table 5.1: Average number of events before *u0*'s trust in *s0* is above 0.6

to increase above 0.6. It can be seen that when using the fuzzy logic method it courses the

[2]I also tried repeating the experiment 100 times. That did not change the conclusion. As a consequence I will, when repeating experiments, do it 20 times.



(a) Discrete-Discrete

(b) Fuzzy-Discrete

Figure 5.5: Bezier approximation of *u0*'s trust in *s0*

trust values to increase slightly faster compared to the discrete method. It is however, not significantly faster.

5.5.1.2 One resource exhibits potentially bad behaviour

It is also relevant to investigate how the two methods evaluate a resource that exhibits potentially bad behaviour. Figure 5.6 illustrates the scenario. The black circle denotes that

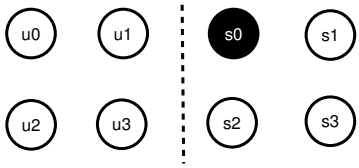
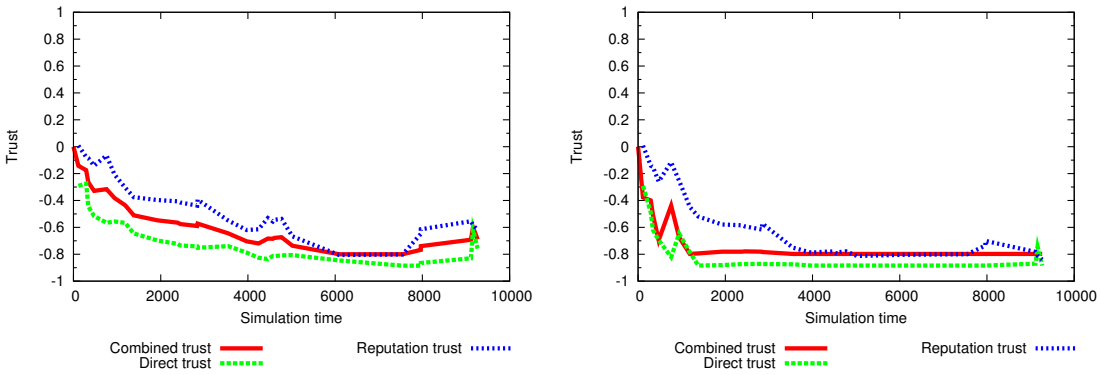


Figure 5.6: Scenario for experiment 1.2

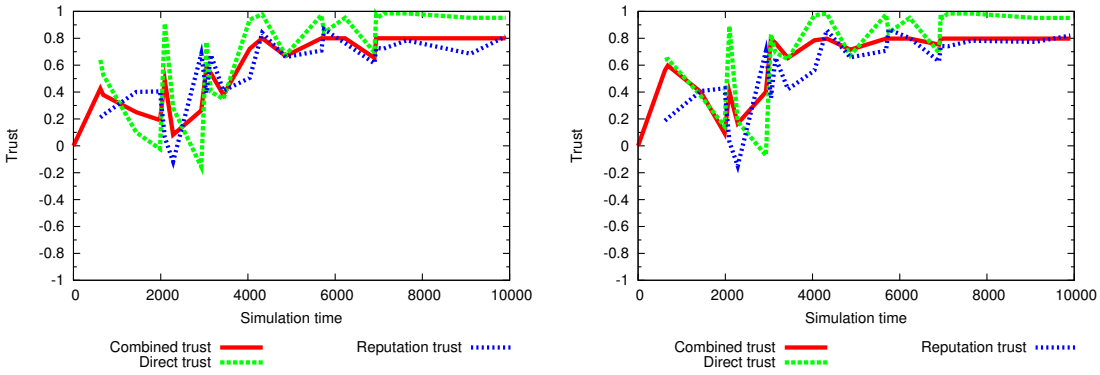
s0 exhibits potentially bad behaviour. Potentially bad behaviour should be interpreted as a high probability of bad behaviour. It is not the same as saying the resource always misbehaves. The behavioural values are both 20 which indicates a high probability of bad behaviour. Occasionally the resource will however, exhibit good behaviour. The experiment has been simulated once for each combination method, and figure 5.7 depicts *u0*'s trust experience of *s0*. It can be noticed that the combined trust value for the discrete method is exactly between the direct and reputation trust values until the value 0.8 is reached. This is not the case when using fuzzy logic. It appears the trust values decrease faster when using the fuzzy logic method.
It should be noted that all the other resources are still able to distinguish a well behaving re-



(a) Discrete-Discrete (b) Fuzzy-Discrete

Figure 5.7: Comparing combination methods 2

source from the potentially misbehaving resource. However, the site's calculation of a user's reputation values will be affected when a site misbehaves. This is because the misbehaving site sends very inaccurate ratings of the users. Without alliances the sites can't distinguish between ratings given by behaving and misbehaving resources. Consequently all the ratings will have the same weight. Figure 5.8 illustrates *s1*'s trust experience of *u1*. The previous

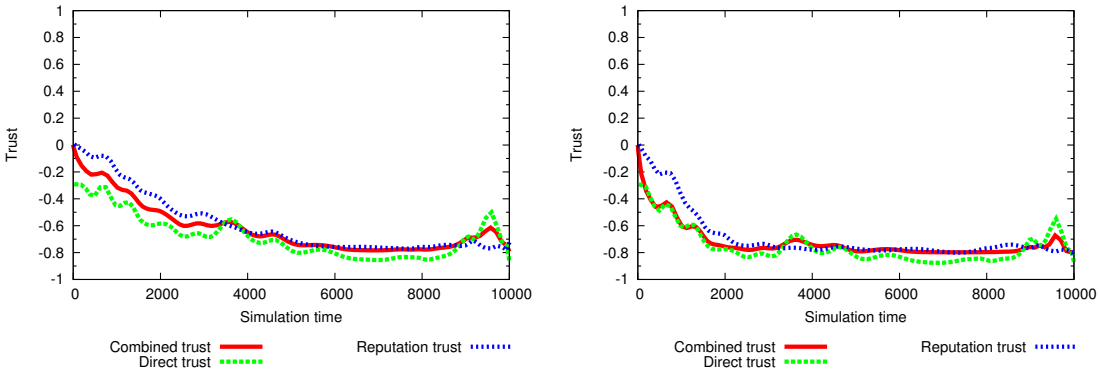


(a) Discrete-Discrete (b) Fuzzy-Discrete

Figure 5.8: Effect of inaccurate ratings

experiment illustrated how trust progressed in a scenario where every resource exhibited good behaviour. We can see that a misbehaving resource will influence the trust evaluation of other resources by sending inaccurate ratings. The resources will however still arrive at

a high trust value for the resources that exhibit good behaviour. The inaccurate ratings do however cause larger fluctuations. The enclosed CD-ROM contains the trust progression figures for all other resources.
The simulation is repeated 20 times for each combination method and figure 5.9 illustrates *u0*'s trust progression for *s0*. The 20 repetitions for each combination method illustrates



(a) Discrete-Discrete

(b) Fuzzy-Discrete

Figure 5.9: Bezier approximations of *u0*'s trust in *s0*.

the same progression of trust as the single experiment. The fluctuations that were noted when sites received inaccurate ratings are also present when repeating the experiment. The plots are however not shown here, but can be found on the CD-ROM. The average number

Combination method	Combined trust	Direct trust	Reputation trust
Discrete	7.8	5.4	9.55
Fuzzy	3.4	2.65	4.65

Table 5.2: Average number of events before *u0*'s trust in *s0* decreases below -0.6

of events needed for the trust value to decrease below 0.6 are listed in the table. It is obvious that the fuzzy logic approach reaches the -0.6 trust value significantly faster than the discrete approach does. The sequence of random seeds ([100,110,...,290]) are exactly the same for the fuzzy logic and discrete experiments. Consequently all the resources exhibit exactly the same behaviour, and the same sequence of events occurs. This means the data is equivalent and the different progression of trust values can be contributed to the difference between the two combination methods.
The experiment has been repeated for the discrete combination method with the noticeable change that direct trust has a significantly higher weight than reputation trust. The plots of this can be found on the CD-ROM. With direct trust having the weight 0.8 and reputation trust 0.2, the table below contains the number of events needed for the discrete

method. Although the fuzzy logic method still reaches a trust value below -0.6 faster than

Combination method	Combined trust	Direct trust	Reputation trust
Discrete	4.45	3.8	7.05

Table 5.3: Average number of events before *u0*'s trust in *s0* decreases below -0.6 when discrete ratio is 0.8

the discrete method does, the number of events needed are comparable when you put more emphasis on the direct trust value.

5.5.1.3 Comments

The previous experiments have examined the fuzzy logic method and the discrete method for the combination of direct and reputation trust into a single combined trust value. It has been examined how the two perform in a scenario with well behaving resources as well as a scenario where there was one potentially misbehaving site. Both combination methods are able to distinguish between well behaving resources and the resource that potentially exhibited bad behaviour. It was however, noted that the trust values progressed slightly faster when using the fuzzy logic method. By changing the weight ratio for the discrete method a comparable number of events was observed. As described in section 5.3.2 the fuzzy logic rule base also shows a slight preference over direct trust. It should however, be noted that it is merely a slight preference, since the preference will only come in effect when there is doubt as to which output membership set should be chosen. In most fuzzy logic calculations direct and reputation trust are weighed equally. Or put in another way, when it is possible to select a fuzzy output set that is exactly between the direct and reputation fuzzy sets, that is done. When that is impossible, direct trust will be weighed the highest. When weighing direct and reputation trust differently with the discrete combination method then direct trust will have preference in all calculations. That might not be a reasonable decision in every scenario.
From the experiments it was also noted that a potentially misbehaving site will send inaccurate ratings that will influence the trust evaluation of users. This happens because the ratings will influence the reputation trust values. This will be investigated further in the next section.

5.5.2 Comparing reputation systems

In the following I will investigate the discrete reputation system and the beta reputation system. The direct and the reputation trust values will be combined using the discrete method.

5.5.2.1 All resources exhibit good behaviour

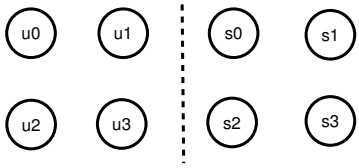
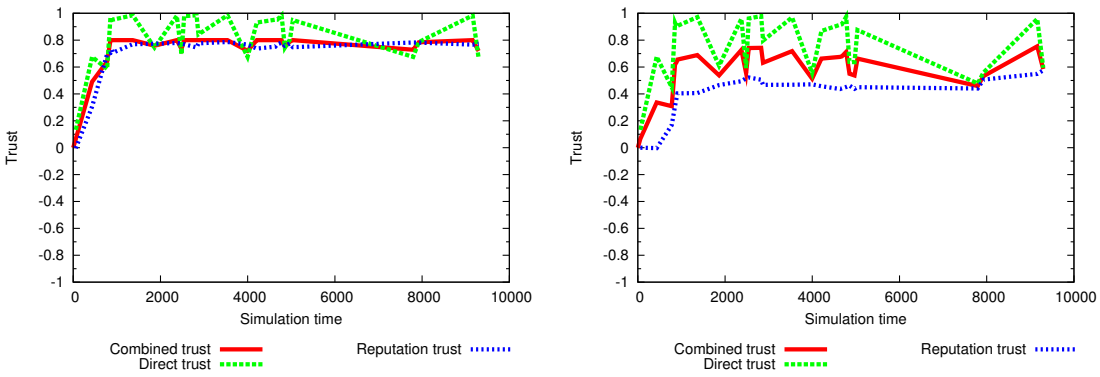


Figure 5.10: Scenario for experiment 2.1

The scenario, depicted in figure 5.10, is equivalent to the first scenario where the combination methods were examined. Using the configuration file from appendix G.2 the two reputations systems show the trust progression depicted in figure 5.11. As previously it is *u0*'s trust

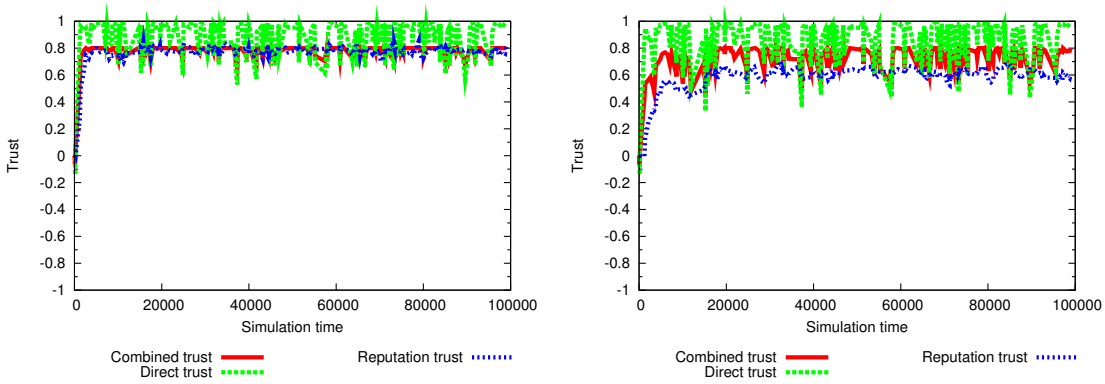


(a) Discrete-Discrete

(b) Discrete-Beta

Figure 5.11: Comparing the discrete with the beta reputation system

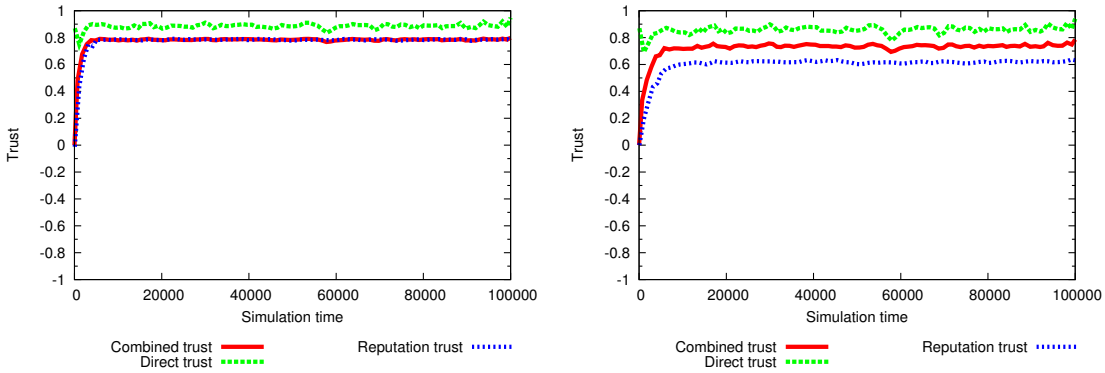
experience of *s0*. It can be seen from figure 5.11(b) that the reputation of *s0* increases slower when using the beta reputation system. This can be contributed to the fact that older values are not forgotten but included in the calculation of the new reputation value. The consequence is a slower increase in reputation trust. This is not the case when using the discrete system. In each reputation calculation it happens as if it was the first time. To investigate how the reputation will progress over a longer period of time, a second experiment is performed where the simulation time is extended to 100000. The result is depicted in figure 5.12. At the specified period of time the reputation of *s0* using the discrete reputation system increases to the maximum value. The beta reputation system however still increases slowly and after a period of time the increase in reputation trust seemes very small. To investigate if this is an exception the experiment is repeated 20 times for both reputation systems. The Bezier approximation curves in figure 5.13 illustrate the



(a) Discrete-Discrete (b) Discrete-Beta

Figure 5.12: Comparing reputation systems when simulation time is 100000

result. From the figures the effect is very visible. The discrete reputation system instantly



(a) Discrete-Discrete (b) Discrete-Beta

Figure 5.13: Bezier approximations of *u0*'s trust in *s0*.

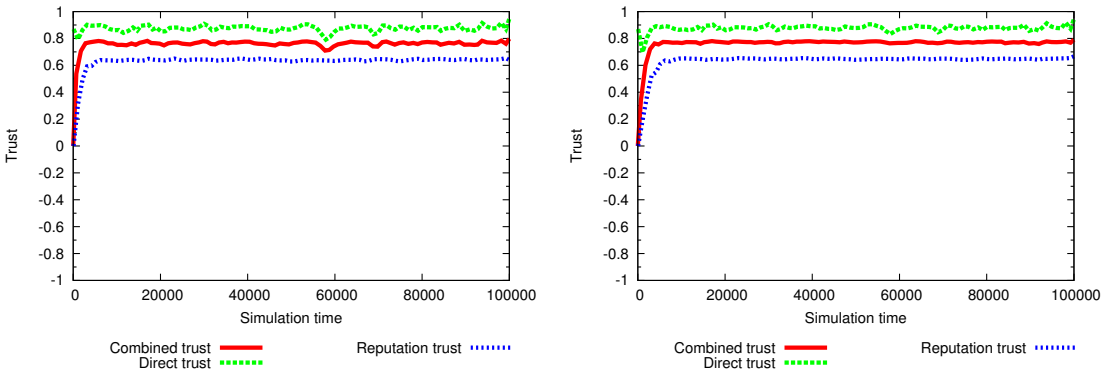
reaches the highest value. With the beta reputation system the reputation trust of *s0* does not increase significantly after reaching a value slightly above 0.6. As a result of this the combined trust value will also only increase slightly. The number events needed in order for a trust value to increase above 0.6 is listed in the table. It can be seen that the reputation value increases significantly slower when using the beta reputation system. Approximately 15 more events are needed.

When using the beta reputation system it can be noticed that the combined trust only

Reputation system	Combined trust	Direct trust	Reputation trust
Discrete	3.6	1.65	4.35
Beta	6.15	1.75	21.9

Table 5.4: Average number of events before *u0*'s trust in *s0* increases above 0.6

increases slightly after reaching a value 0.7. It could be worth investigating the influence of a change in weighing ratio between direct trust and reputation trust. Figure 5.14(a) illustrates the experiment repeated with the exception that direct trust has the weight 0.8 and the reputation trust only has the weight 0.2. It can be seen in figure 5.14(a) that



(a) Discrete-Beta

(b) Fuzzy-Beta

Figure 5.14: Bezier approximations of *u0*'s trust in *s0*. Discrete ratio is 0.8

with this change the combined trust value will rise faster since more emphasis is put on the direct trust value. This will in turn influence the ratings that resources submit to each other. The ratings will be higher and the combined trust value will increase faster. Figure 5.14(b) illustrates the same simulation but here the results are combined using the fuzzy logic method. The change in discrete ratio naturally has no effect when using fuzzy logic. The number of events needed for trust values to increase to 0.6 is listed in the table. It

Combination method	Combined trust	Direct trust	Reputation trust
Discrete	2.1	1.7	8.7
Fuzzy	5	1.7	10.4

Table 5.5: Average number of events before *u0*'s trust in *s0* reaches 0.6 when discrete ratio is 0.8

can be seen that the direct trust value approaches with the same speed for both of the

combination methods. However, the combined and the reputation trust values increase faster when using the discrete combination method. The fuzzy logic method is however faster when comparing it to the case where the discrete combination method weighed direct trust and reputation trust equally.

5.5.2.2 A resource exhibits potentially bad behaviour

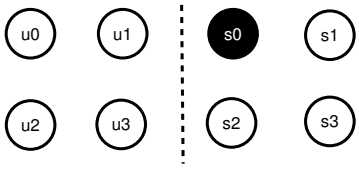
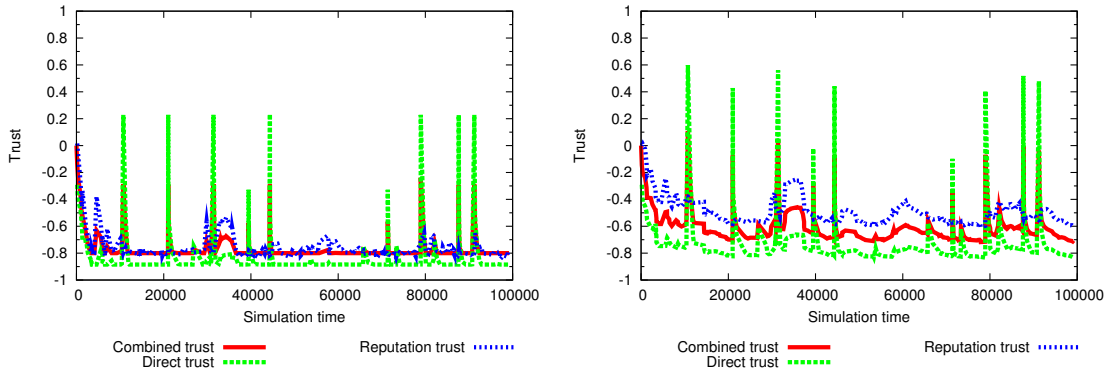


Figure 5.15: Scenario for experiment 2.2

The scenario is equivalent to the second scenario where the combination methods were examined. Knowing that the beta reputation progresses slowly the simulation time is initially set to 100000. Figure 5.16 illustrates *u0*'s trust experience of *s0*. It can be seen that the



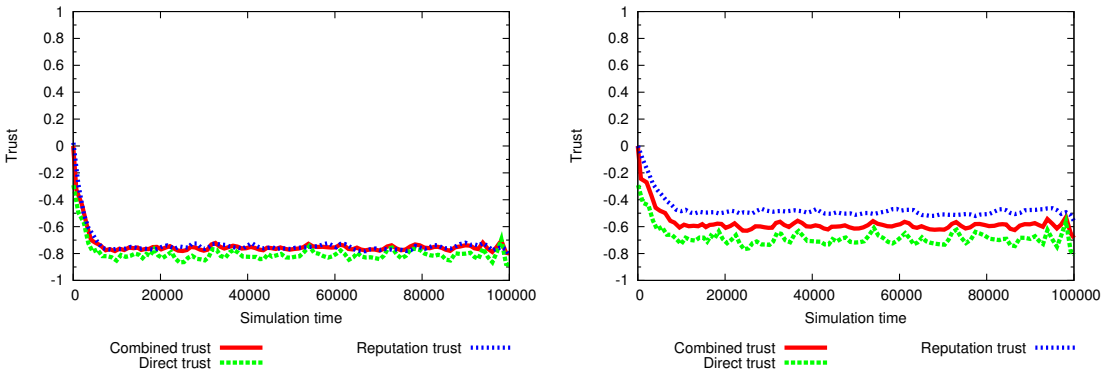
(a) Discrete-Discrete

(b) Discrete-Beta

Figure 5.16: Comparing the discrete with beta reputation system

trust values appears to progress slower when the beta reputation system is used. This could naturally have been expected from the previous experiment. It also appears that the reputation trust value does not decrease as much as with the discrete reputation system. This again is as expected from the previous experiment. The plots also shows significant fluctuations for the direct trust value. This reason for this lies in the fact that even though the two behavioural values for *s0* are high, it is not the same as saying it always will misbehave. It merely indicates that there is a higher probability that the site will allocate to much or

to little CPU time. Occasionally *s0* will allocate an accurate amount of CPU time and consequently receive a high direct trust value. The experiment has been repeated 20 times for each reputation system and the result is illustrated in figure 5.17. As expected with the



(a) Discrete-Discrete

(b) Discrete-Beta

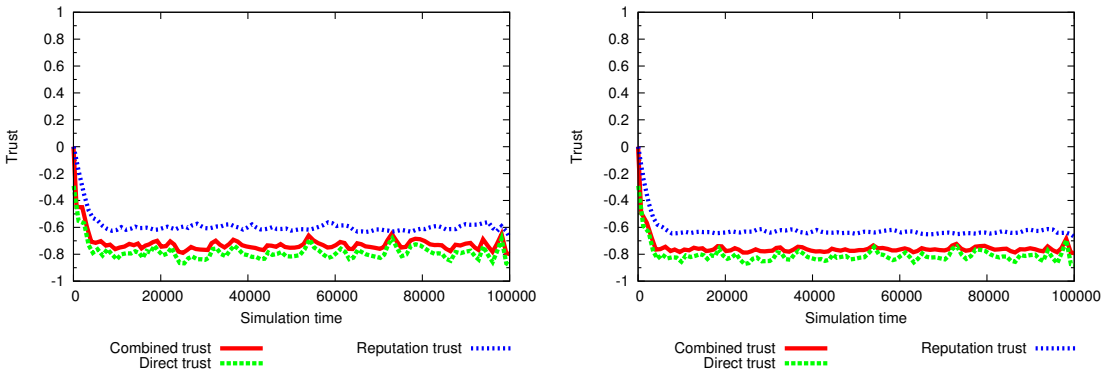
Figure 5.17: Bezier approximations of *u0*'s trust in *s0*

discrete reputation system in figure 5.17(a) all trust values almost instantly decrease to the minimum value. The beta reputation system in figure 5.17(b) does not reach the low values that the discrete reputation system does. In fact it appears that the reputation trust value does not decrease significantly below 0.5. Consequently the table lists the needed number of events to reach a trust value below 0.4. In the table the slower progression of reputation

Reputation system	Combined trust	Direct trust	Reputation trust
Discrete	3.65	2.35	5.4
Beta	7.2	2.6	15.05

Table 5.6: Average number of events before *u0*'s trust in *s0* decreases below -0.4 when discrete ratio is 0.5

trust when using the beta reputation system is very noticeable. A significant higher amount of events are needed.
In the previous experiments the trust values increased significantly more when direct trust was weighed higher than reputation trust. For this reason the experiment has been repeated 20 times where direct trust had the weight 0.8 and reputation trust the weight 0.2, exactly as the previous experiment. The purpose is to examine whether or not putting more emphasis on the direct trust value will cause a greater decrease of trust values. Figure 5.18 illustrates the result. The effect that was noted in the previous experiment is also apparent in this case. Figure 5.18(a) illustrates that when the weight for direct trust is increased the combined trust and the reputation trust decreases more. When using the fuzzy logic



(a) Discrete-Beta (direct trust w:0.8)

(b) Fuzzy-beta

Figure 5.18: Comparing discrete with fuzzy logic

combination this change is however not needed. Figure 5.18(b) shows the same experiment repeated when direct and reputation trust are combined using the fuzzy logic method. The

Combination method	Combined trust	Direct trust	Reputation trust
Discrete	2.75	2.5	7.15
Fuzzy	3.4	2.4	5.95

Table 5.7: Average number of events before *u0*'s trust in *s0* decreases below -0.4 when discrete ratio is 0.8

table contains the average number of events needed for the trust values to decrease below -0.4. When the discrete ratio between direct and reputation trust is 0.8 the discrete and fuzzy logic methods perform very similarly.

5.5.2.3 Comments

The beta reputation system showed a slower progression of reputation trust. This also results in a slower progression of combined trust. When using the discrete combination method, and the weight ratio between direct and reputation trust was 0.5, the trust values did not increase significantly beyond certain values. By shifting the ratio and putting more emphasis on the direct trust values it was possible to force the trust values to increase or decrease more. When using the fuzzy logic combination there was no need to change the fuzzy rule base.
When using a reputation system a slow progression of trust might be sought after. In most cases it would seem suspicious if the reputation of a resource increased significantly based on a very small number of events. This has to do with the scenarios. In the examined

scenarios none of the resources had any alliances which meant that they could not be sure of the accuracy of the ratings received. In these scenarios a slow progression of reputation trust should occur. However, the slow increase in reputation trust also coursed the combined trust to progress significantly slower. That might not be a sought after effect. In the scenarios the resources directly experience good or bad behaviour from the resource in question. This direct experience should naturally influence the overall combined trust in the resource. By putting more emphasis on the direct experience, meaning direct trust, the combined trust value would reflect this. But it should of course be noted that when putting significantly more emphasis on direct trust it will in a sense undermine the information from a reputation system.
The previous scenarios did however not show any strong indication of the beta reputation system being better or worse than the discrete reputation system.

5.5.3 Resources change their behaviour

I a real world scenario it can't be expected that a resource behaves either good or bad all the time. Here I will investigate the effect of a sudden behavioural change. When investigating the progression of trust only the combined trust will be plotted. The reason is that in a real world scenario only the combined trust value would be used to make a decision. In the following I will repeat the experiments 20 times for each combination method using either of the reputation systems, with the random number seeds being [100,110,..,290]. The plots will contain the Bezier approximation curves including the combined trust value data points.

5.5.3.1 A change to potentially bad behaviour

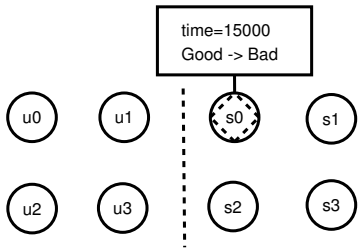
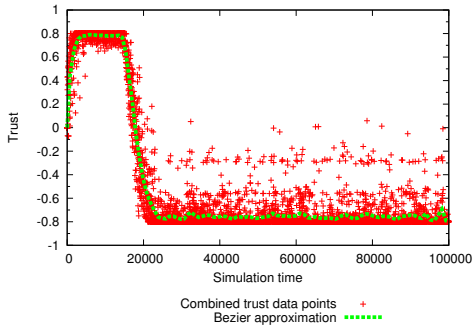
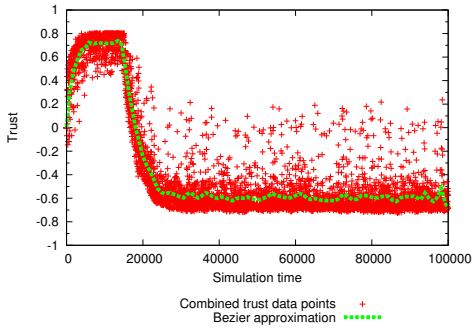


Figure 5.19: Scenario for experiment 3.1

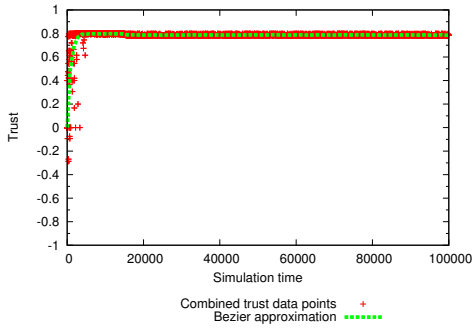
Using the configuration file from appendix G.3 the scenario from figure 5.19 has been simulated. The dashed lines within the circle for site *s0* indicate that it changes its behaviour. The change from good to potentially bad behaviour occurs when the simulation time is 15000. Figure 5.20 contains the data points and the Bezier approximation curve for the two combination methods using either of the two reputation systems. Only *u0*'s combined trust values for *s0* are plotted. The first that is noticeable when looking at the plots, is



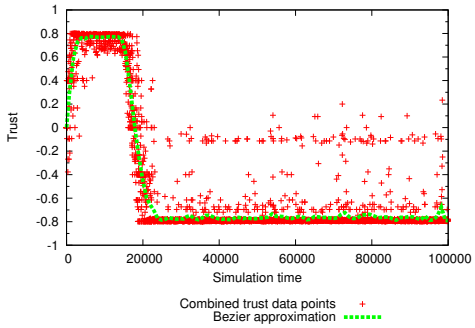
(a) Discrete-Discrete



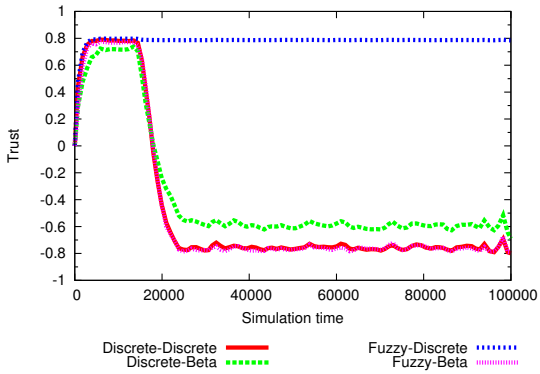
(b) Discrete-Beta



(c) Fuzzy-Discrete



(d) Fuzzy-Beta



(e) Bezier approximations

Figure 5.20: Change from good to bad behaviour at t=15000

that the fuzzy logic method using the discrete reputation does not pick up on the change in behaviour. It arrives at the wrong conclusion since, in the eyes of *u0*, *s0* exhibits good behaviour. The plots, present on the CD-ROM, show that all other users also conclude that site *s0* exhibits good behaviour when using fuzzy logic with the discrete reputation system. There is an explanation for this. When using the discrete reputation system all old reputation values are forgotten and hence the earlier behaviour of a resource will not be reflected in the reputation ratings. When a resource returns a rating, the rating will be based on the combined trust value. The combined trust value is based on the direct and the reputation trust value. If a resource continually receives high ratings, and therefore calculates a high reputation trust value, it will result in a relatively high combined trust value even if the calculated direct trust value is low. When using the fuzzy logic method, the trust values will be massed at certain values. These values are the centre of gravity for an output membership function. For example when the aggregated result only falls into the "very high" output membership function the centre of gravity is 0.8. If the subsequent combined trust value calculated does not have a higher membership in the "high" membership function, the combined trust value will not change. Then output membership function "high" has a centre of gravity around 0.4, and the values will not fall below this value. In short, when old reputation values are forgotten, the old reputation trust will be artificially high. This is caused since the values calculated with fuzzy logic will not decrease to the membership functions values below "very high". A later experiment will show an example that illustrates the mass of values at the centre of gravity for the output membership functions.

When observing the three remaining methods, it can be concluded that each of them arrive at a similar conclusion, namely that the behaviour of *s0* shifts from good to bad. A full page figure of the trust progression of *s0* from *u0*'s point of view using fuzzy logic and the beta reputation system is included as appendix H.2. This plot shows the progression of combined trust in each individual simulation run and it validates that the Bezier approximation curves are accurately depicting the progression of trust.

By observing the plots depicting the use of the beta reputation system more fluctuations can be noticed. This is the effect of the older reputation values. Or put in another way, $s0$ exhibited good behaviour in the past and even though is now exhibits bad behaviour the record of the good behaviour exists. When $s0$ occasionally allocates the informed amount of CPU time it receives a high trust rating. With the slower decreasing reputation value it will result in an higher combined trust value. This causes the fluctuations. With discrete method and the discrete reputation system the value instantly increases to the maximum. When the simulation time is above 15000 it also quickly decreases to the minimum value. Every trust calculation happens almost as if it was the first time it was calculated. The table contains the number of events that are needed for the trust values to decrease to -0.6 or below. It is particularly noticeable how much slower the progression of reputation trust is with the discrete method using the beta reputation system. The fuzzy method using the beta reputation system is however comparable with the discrete. Interestingly it can also be seen that fuzzy-logic combined with the beta reputation system is slightly faster than the discrete method with the discrete reputation system. Or put in another way the effect of the beta reputation system is not as apparent.

Method - reputation system	Combined trust	Direct trust	Reputation trust
Discrete-Discrete	17.3	18.4	18.8
Fuzzy-Discrete	∞	∞	∞
Discrete-Beta	30.75	19.8	116.875
Fuzzy-Beta	16.05	13.3	21.75

Table 5.8: Average number of events after time = 15000 before *u0*'s trust in *s0* is below -0.6

5.5.3.2 A change from potentially bad to good behaviour

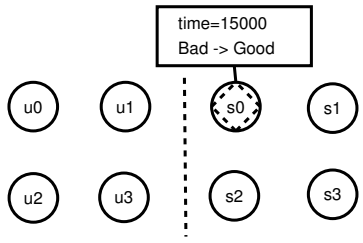
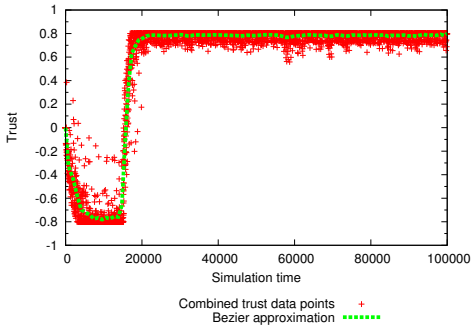
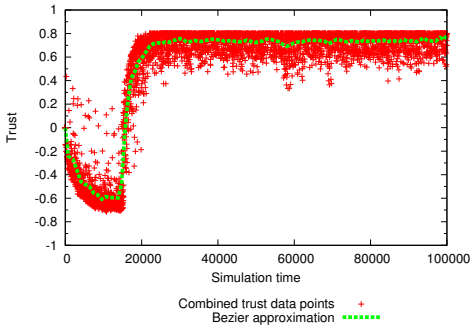


Figure 5.21: Scenario for experiment 3.2

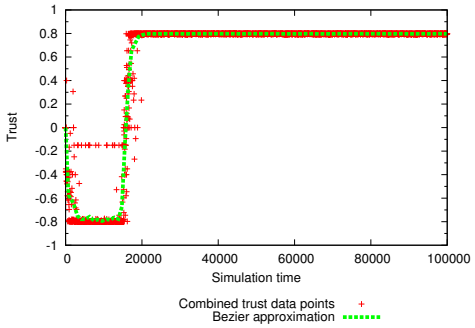
The scenario in figure 5.21 is very similar to the previous experiment. However instead of changing *s0*'s behaviour from good to bad, the behaviour will change from bad to good. The experiment has been simulated using the configuration file from appendix G.3. I have plotted the data points and the Bezier approximation curves for the two combination methods using the two reputation systems in figure 5.22. Only *u0*'s combined trust values for *s0* are plotted. The figures illustrates that each method using either of the reputation system arrives at the same conclusion. Namely that the behaviour of *u0*, in the eyes of *u0*, changes from bad to good. This is also the case for the other users as shown on the plot on the CD-ROM. The figure 5.22(e) that contains all the Bezier approximation curves in the same plot indicates that the trust progression is very similar in all cases.
From the previous experiment it might have been expected that fuzzy logic method using the discrete reputation system would arrive at a wrong conclusion again. This however, is not the case. As described in section 5.3.2 the fuzzy logic method slightly favours direct trust over reputation trust. This means that the direct trust values will have a slightly higher influence on the combined trust value. The results indicate that having the behavioural values [1,1] (for good behaviour) is better with a higher degree than the degree of behaviour obtained by having the behavioural values [20,20]. Put in another way the degree of good behaviour is higher than the degree of bad behaviour. The high degree of good behaviour of *s0* will result in higher direct trust values and consequently also a relatively high combined trust value. This will mean that the shift from a "very low" to a "low" combined trust level will be possible. The mass of data points is however noticeable in figure 5.22(c). Slightly



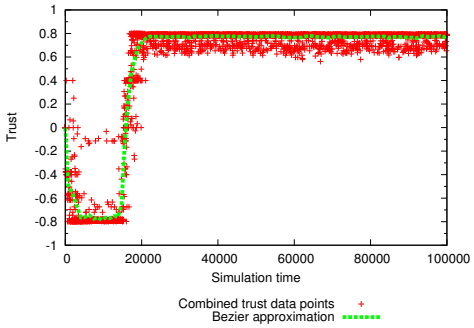
(a) Discrete-Discrete



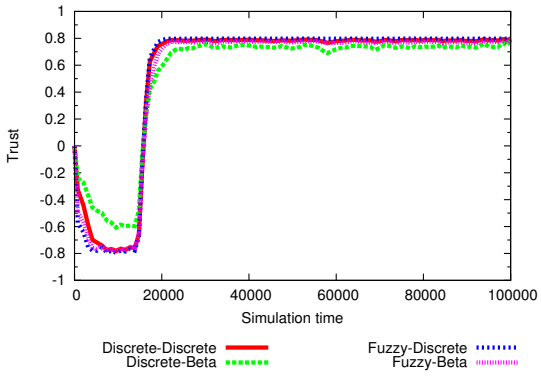
(b) Discrete-Beta



(c) Fuzzy-Discrete



(d) Fuzzy-Beta



(e) Bezier approximations

Figure 5.22: Change from bad to good behaviour at t=15000

below the combined trust value 0.1 there is a line of data points. The line indicates that the combined trust value will fall into the fuzzy output set "medium". In the previous experiment the membership function "high" was not reached, meaning that the combined trust value claimed stronger membership to the "very high" fuzzy set. In this experiment the "low" fuzzy set will be reached even though there is a strong membership to the fuzzy set "very low". The reason is that the degree of high direct trust values is larger than the degree of low direct trust in the previous experiment. It can be noted that the line also appears in figure 5.22(d) however not as clear. This can be contributed to the beta reputation system. Since the previous reputation values affect the current reputation value, the membership to the fuzzy set "medium" will not be as strong. Or put in another way, the continual progression of reputation trust will reduce the likelihood that a combined trust value will only claim strong membership to a single output fuzzy set. The table contains

Method - reputation system	Combined trust	Direct trust	Reputation trust
Discrete-Discrete	7.35	3.9	9.4
Fuzzy-Discrete	7.15	4.4	7.75
Discrete-Beta	10.55	3.6	24.55
Fuzzy-Beta	9.8	4.6	15.95

Table 5.9: Average number of events after time = 15000 for *u0*'s trust in *s0* is above 0.6

the number events needed for *u0*'s trust in *s0* to be above 0.6 after the time has reached 15000. In all four cases the events needed are almost similar. The only exception is when viewing the reputation trust when using the beta reputation system. It takes significantly more events to reach 0.6 compared to the discrete reputation system. Particularly when using the discrete method for combining direct and reputation trust. However as previous experiments showed, using an equal weight between direct and reputation trust causes all trust values to progress slower when using the discrete combination method.

5.5.3.3 Varying behaviour

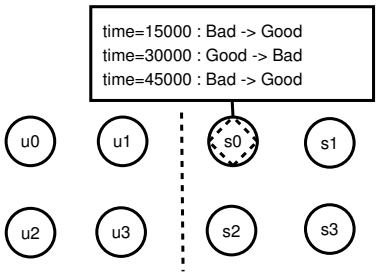


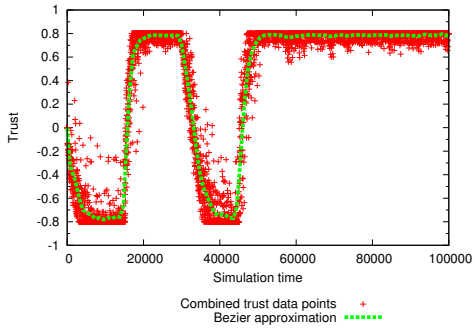
Figure 5.23: Scenario for experiment 4.1

The following experiment will reflect a more extreme turn of events. Here the behaviour of *s0* changes three times. In the beginning the behaviour will be potentially bad. After 15000 it will behave properly and 15000 later it will potentially misbehave again. After another 15000 it will behave properly again. The configuration file is shown in appendix G.4. The data points and the Bezier approximation curves for the two combination methods using the two reputation systems are illustrated in figure 5.24. As previously, only *u0*'s combined trust values for *s0* have been plotted. It can be noticed that the fuzzy logic method using the discrete reputation system does not pick up on the change in behaviour. This was expected and the reason for this is the same as previously stated. However, the plot in figure 5.25 illustrates that the previously stated reason is correct. When observing the direct trust value it can be seen that it drops significantly in the period where *s0* misbehaves. But the drop is not sufficient to cause a higher membership to a lower fuzzy set.
The remaining methods and reputation system in figure 5.24 illustrate the expected progression of trust. The table contains the number of events needed for the trust values to

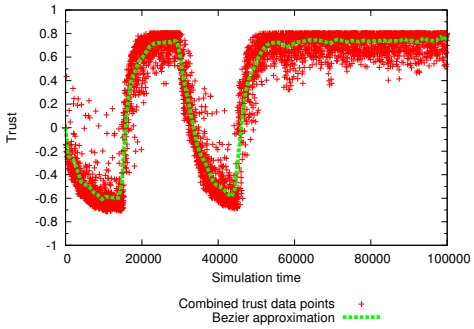
Method - reputation system	Combined trust	Direct trust	Reputation trust
Discrete-Discrete	7.1	3.15	8.3
Fuzzy-Discrete	0	0	0
Discrete-Beta	10.7	2.8	22.3
Fuzzy-Beta	9.2	4.15	15.2

Table 5.10: Average number of events after time = 45000 for *u0*'s trust in *s0* is above 0.6

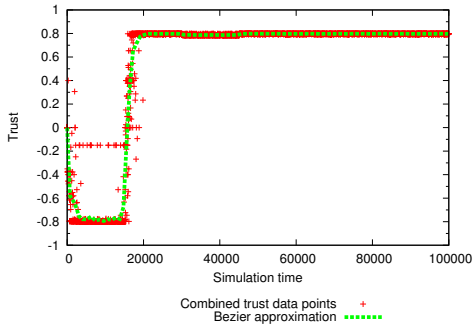
decrease below 0.6 after the simulation time has passed 45000. When using the beta reputation system there does seem to be a need for a larger number of events. I did however expect the beta reputation system to require a significantly larger amount of events before reaching the 0.6 value. This will be investigated in the next experiment.



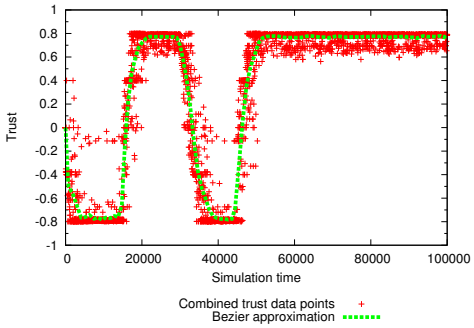
(a) Discrete-Discrete



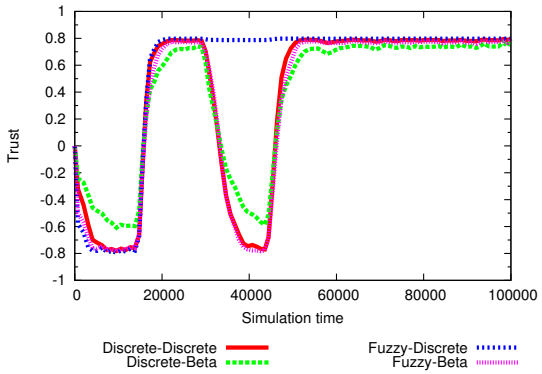
(b) Discrete-Beta



(c) Fuzzy-Discrete



(d) Fuzzy-Beta



(e) Bezier approximations

Figure 5.24: When behaviour varies

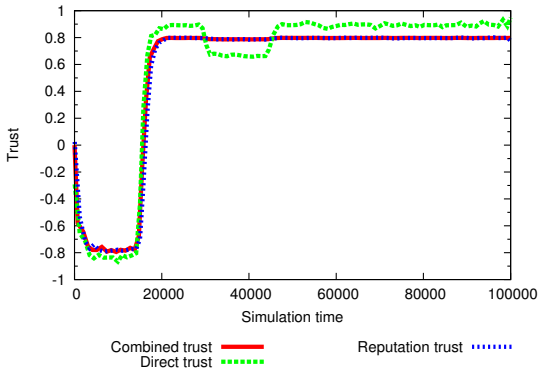
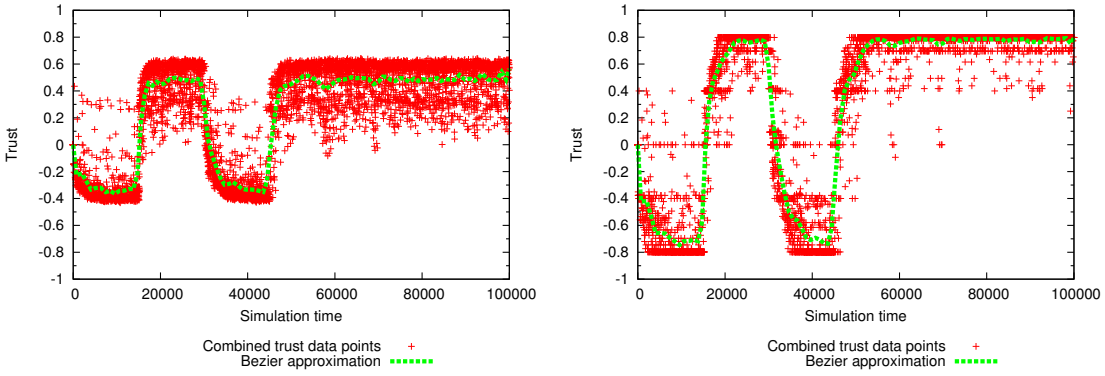


Figure 5.25: Fuzzy-Discrete. Beizer approximations of *u0*'s direct,reputation and combined trust for *s0*.

5.5.3.4 Effect of the forgetting factor with varying behaviour

In the previous experiment almost identical progression of reputation trust was observed for the two reputation systems. I expected that the beta reputation system would progress noticeably slower than it did. In this experiment I will investigate the same scenario with the exception that older reputation values should be used for a longer period of time. This means the forgetting factor should be smaller. The experiment will only be done using the beta reputation system, since the forgetting factor is not used by the discrete reputation system. The forgetting factor has been decreased to 0.2 and figure 5.26 illustrates *u0*'s combined trust values for *s0*. When observing the plots it is apparent that the trust value calculated



(a) Discrete-Beta (b) Fuzzy-Beta

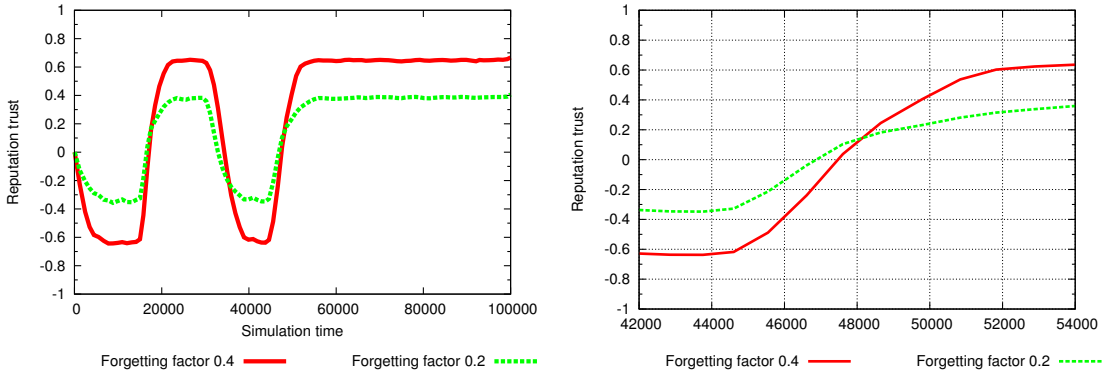
Figure 5.26: Forgetting factor when behaviour varies

by the discrete method using beta reputation system does not decrease or increase similarly to the previous experiment. In fact it appears that the spectrum of possible combined trust values can only be slightly below -0.4 and slightly above 0.6. From the previous experiments you could expect that higher values could probably be reached by putting more emphasis on the direct trust value. The fuzzy logic method still spans the [-0.8:0.8] interval as it did previously. When comparing the progression of trust with the progression from the previous experiment a slightly higher amount of trust values are below the maximum trust value. The random seeds are the same for this and the previous experiment, and the only change between the experiments is the forgetting factor for the beta reputation system. The reason for the smaller span of trust values must be a result of the forgetting factor. Conclusively the forgetting factor has the intended effect of slowing down the progression of trust. The

Method - reputation system	Combined trust	Direct trust	Reputation trust
Discrete-Beta	18.85	2.3	unknown
Fuzzy-Beta	12	3.65	unknown

Table 5.11: Average number of events after time = 45000 for *u0*'s trust in *s0* is above 0.6

table contains the number of events needed for the trust values to be above 0.6 after the time has passed 45000. In none of the cases did the reputation trust increase above 0.6. That happened in the previous experiment. The effect of the slow progression of reputation trust can however be seen in the relatively slower progression of combined trust. It has resulted in an increase in the number of events needed for the combined trust value to increase above 0.6. When observing only the progression of reputation trust values an effect of the forgetting factor can be seen. Figure 5.27(a) illustrates the Beizer approximation curves for the reputation trust values when using the fuzzy logic method and the beta reputation



(a) Different forgetting factors

(b) Closeup of time 42000-54000

Figure 5.27: Forgetting factor when behaviour varies

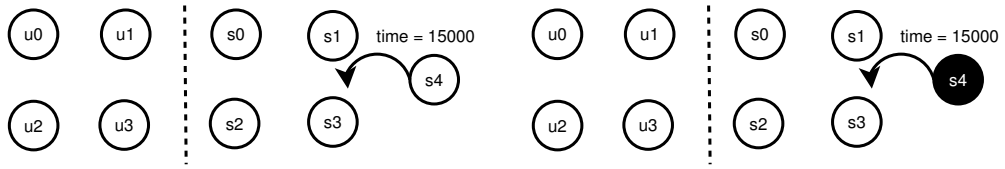
system. Both the previous experiment, where the forgetting factor was 0.4, and the current experiment, where the forgetting factor is 0.2, is shown. It can be noted, even though it is not shown here, that the discrete method using the beta reputation system exhibits a similar result. When observing the angle of the curves when the simulation time is around 48000 it can be seen that the lower forgetting factor causes a slightly slower increase in reputation trust. This can be verified by considering the closeup of the time frame 42000-54000 in figure 5.27(b). It is however only a slightly slower increase. The most noticeable fact from the plot is that the reputation trust values does not increase as much when the forgetting factor is lower. In fact it appears that the possible span for the reputation trust values are limited to[-0.4:0.4].
In the experimented scenarios a reduction of the forgetting factor caused a slower progression of reputation trust, but it happened at the cost of a lower span of possible reputation values.

5.5.3.5 Comments

In this section I have performed a number of experiments of different scenarios that involved changing behaviours of a resource. In all cases except one, when using fuzzy logic with a discrete reputation system, the trust variations where calculated, and reflected the behaviour in the scenario. The reason that the fuzzy logic method using the discrete reputation system did reflect the behaviour in the scenario was contributed to a combination of the design of the fuzzy logic membership functions and rule base, and the properties of the discrete reputation system. It was obvious from the outcome of the experiments that the reputation trust values calculated using discrete reputation system did not reflect earlier behaviour. Each time a reputation trust value was calculated, it was calculated as if it was the first time. This was illustrated in the plots as the trust values immediately changed when the behaviour of a resource changed. When using the beta reputation system this was not the case. However, the effect of older reputation values being used in newer reputation calculation was not as apparent as could have been expected. By decreasing the forgetting factor, a slow progression of reputation trust could be observed, but it came at the cost of a more compressed interval for trust values. When the beta reputation system was used the fuzzy logic method was able to span a wider interval of combined trust values than was possible when using the discrete method of combined trust. From the first experiments it was observed that putting more emphasis on the direct trust value when doing the combination would force the discrete method to use the same interval as was obtained using the fuzzy logic method. If a design goal of the beta reputation system was to reflect previous actions of a resource in current reputation calculations, this goal was achieved. However, the effect of the old reputation trust value was that reputation trust did not increase or decrease to very high and very low values. The effect was, that when using the discrete combination method the combined trust value also spanned a narrower interval of possible value. This was not the case when using fuzzy logic.
The reason that the reputation trust values did not increase or decrease significantly beyond certain values, most likely has to do with the base weight of the received ratings. As described earlier a base weight of 40 was chosen. I did experiment with many different base

weight values, but choosing one over the other was difficult. A lower base weight for ratings would cause the beta distribution to change more. Consequently it would mean a more rapid change in reputation trust and in a sense this undermined the design goal of the beta reputation system. Or put in another way a lower base weight would cause the posterior beta distribution to change more from the prior beta distribution. In some cases this might be sought after, in other case it might not be. It is however, evident that choosing a general base weight for ratings that works well in every scenario is to say the least a difficult task.

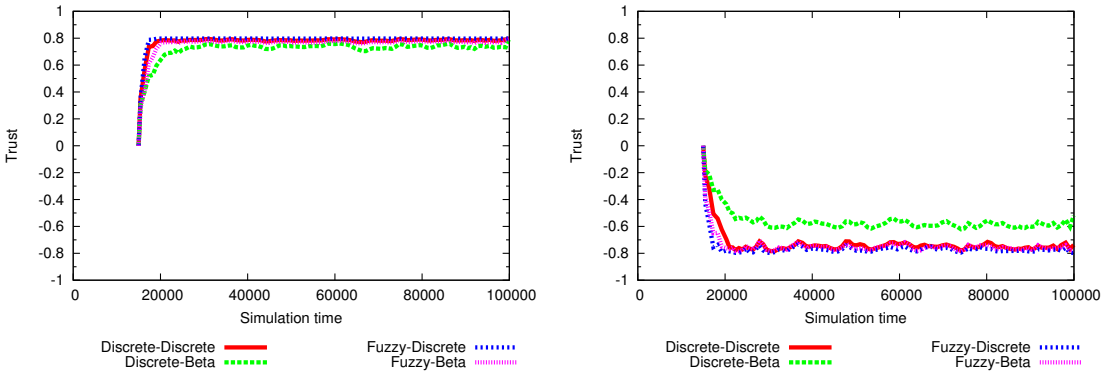
5.5.4 Arrival of a new resource



(a) Scenario for experiment 5.1 (b) Scenario for experiment 5.2

Figure 5.28: Arrival of a resource

This experiment will investigate how the grid system experiences the arrival of a new previously unknown resource. In the scenarios site *s4* joins the grid system when the simulation time is 15000. In both cases all of the resources exhibit good behaviour. Indicated in figure 5.28(a) experiment 5.1 simulates when the arriving resource exhibits good behaviour and experiment 5.2 illustrated in figure 5.28(b) simulates when the arriving site exhibits potentially bad behaviour. The configuration file G.5 is used and the experiment is repeated 20



(a) Discrete-Beta (b) Fuzzy-Beta

Figure 5.29: Arrival of a new resource.

times as previously. Figure 5.29(a) depicts the Beizer approximation curves for *u0* trust experience of *s4*. The figure clearly illustrates that the expected outcome occurs. When the arriving resource behaves the trust in the resource increases. When it misbehaves the trust decreases.
From the experiment it can be concluded that the measurement of trust will not exclude or intentionally give lower trust values to arriving resources. Both of the methods for combining trust using either of the reputation systems will give fair evaluations.

5.5.5 The effect of the threshold value

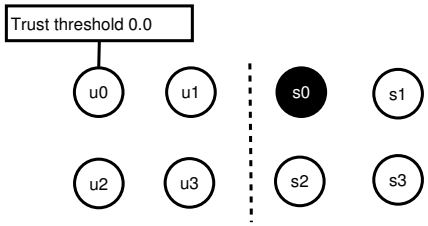
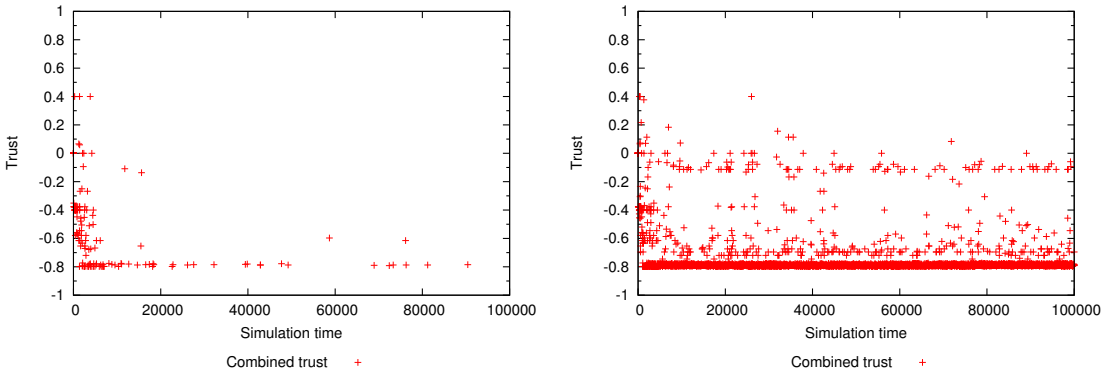


Figure 5.30: Scenario for experiment 6.1

The experiment scenario in figure 5.30 is very similar to the scenario from experiment 1.2. However the difference is that $u0$ has a trust threshold of 0.0 where the threshold previously was -1, equivalent to no threshold. In the experiment all resources besides $u0$ have no trust threshold. The configuration file from appendix G.6 is used and the experiment has been repeated 20 times exactly as in earlier experiments. The experiment has been done for both combination methods using either reputation system. All of them showed the same result. For this reason only the fuzzy logic method using the beta reputation system is shown. The remaining plots can be found on the CD-ROM. Figure 5.31 illustrates the data points for $u0$'s combined trust values in $s0$ and $u1$'s combined trust values in $s0$. The effect of the



(a) $u0$'s trust in $s0$

(b) $u1$'s trust in $s0$

Figure 5.31: Effect of the threshold.

threshold can be seen in the plots. In figure 5.31(a) $u0$'s trust threshold effects the number of events between $u0$ and $s0$. Because the trust value quickly falls below the 0.0 threshold, $u0$ will submit fewer jobs to site $s0$. Consequently fewer event occur between the resources. The events that occur when the trust values are below 0.0 are events where $s0$ is retried for the possibility that it has improved its behaviour. If the behaviour has not improved,

even longer time has to pass before $s0$ is retried again. This is obvious because there were several more trust value data points in the earlier states of the simulation. It should of course be noted that the data points are obtained from 20 experiments, and in a single run the site will not be retried as often as could be indicated by the plot. In figure 5.31(b) is can be seen that $u1$ does not have a threshold. $u1$ will continually submit jobs to site $s0$ even though it, in the eyes of $u1$, has a low trust value. From the table it is apparent that

user	Time $>$ 500	Time $>$ 2500	Time $>$ 7000
$u0$	6.15	3.75	1.8
$u1$	246.05	241.7	230.75

Table 5.12: Average number of events between a user and site $s0$

$u0$ will submit fewer and fewer jobs to site $s0$. As time progresses, longer periods of time has to pass before the site is retried. This is not the case when $u1$ is evaluating $s1$. It will continually submit jobs, and hence events between the resources will happen.
From the experiment it can be concluded that the trust threshold works as expected.

5.5.6 Importance of alliances

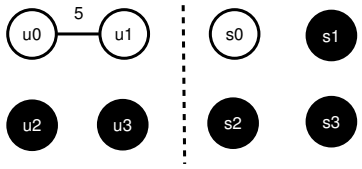
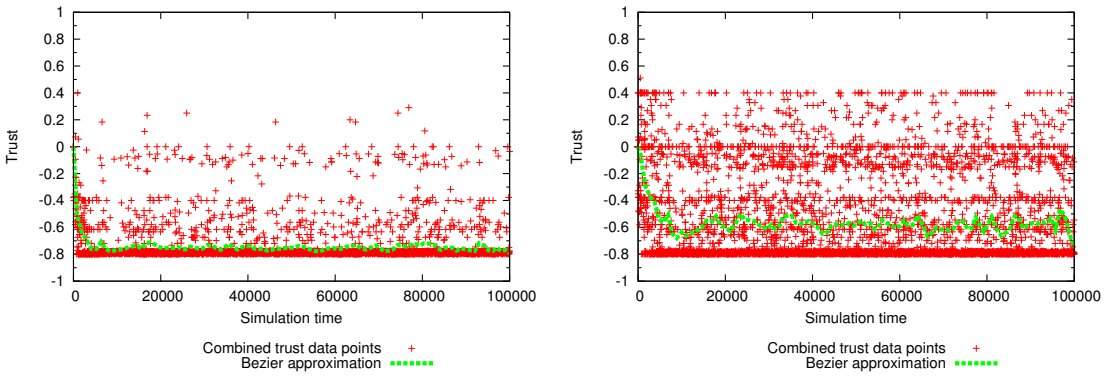


Figure 5.32: Scenario for experiment 7.1

This experiment depicted in figure 5.32 reflects a very hostile environment. In the scenario all of the resource besides $u0, u1$ and $s0$ potentially misbehave. However, there is a strong alliance between the users $u0$ and $u1$. The alliance strength is five as illustrated in the figure. The configuration file from appendix G.7 is used when the experiment was repeated 20 times for both combination methods using both reputation systems. The fuzzy logic and discrete methods with either of the reputation systems show similar results. With the use of alliances, resources are able to protect themselves better against resources with bad behaviour. The figure 5.33 shows an example of this. In both cases the fuzzy logic method is used with the beta reputation system. In figure 5.33(a) $u0$'s combined trust values for $s2$ is shown. In figure 5.33(b) $u2$'s combined trust values for $s2$ is shown. $u0$ has a strong alliance with $u1$ and the resource can rely on the ratings that it submits. Consequently it will also weigh those ratings higher when calculating the reputation value. This is not the case for user $u2$. It does not have any alliances and even though it receives ratings from two users with good behaviour ($u0$ and $u1$) it can not distinguish between the ratings.



(a) $u0$'s trust in $s2$ (b) $u2$'s trust in $s2$

Figure 5.33: Effect of alliances

This means that the ratings from user $u3$ will have a similar weight when calculating the reputation value. The Bezier approximation curve illustrates that $u2$ arrives to the same conclusion as $u1$. There is however significantly more fluctuations, and occasionally a wrong conclusion would probably be drawn should a trust value be used to make a decision based on the combined trust value. The large fluctuations have also convinced me that calculating average number of events needed for a trust value to be above or below a specified value would be misleading in this case.
The CD-ROM contains the remaining plots of user's evaluation of $s0$ who exhibits good behaviour as well as sites evaluating the users. In both cases outcome of the experiment is similar to what has been described above.

5.5.7 Breakdown of trust

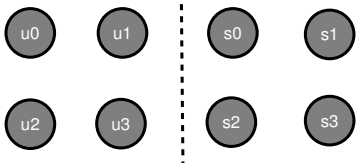


Figure 5.34: Scenario for experiment 8.1

The last experiment illustrated in figure 5.34 simulates a scenario where all of the resources show "medium" behaviour. This is indicated by the circles being grey. Medium behaviour means behaviour that will not always fall in to a good or bad category. Occasionally it will however. In the real world, if nobody is trusted or distrusted, the term trust becomes

meaningless. It would be impossible to distinguish between trustworthy and untrustworthy resources. It would result in a breakdown of trust. This is also the case in the simulation. The point of "medium" behaviour occurs when setting both behavioural values to five. The configuration file from appendix G.8 has been used in the 20 repeated experiments. Both the fuzzy logic and discrete method using either of the reputation systems arrive at the same conclusion. The trust relationship between users and sites, and sites and users are also equivalent. The figure 5.35 illustrates $u0$ trust evaluation of $s0$ using the fuzzy logic

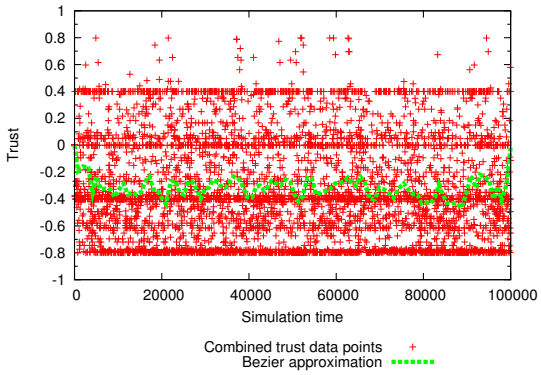


Figure 5.35: Breakdown of trust between $u0$ and $s0$

method and the beta reputation system. The trust values span almost the entire trust interval. In such a scenario it would be impossible to make decisions based on trust.
The remaining plots for the other users and sites and using the other method and reputation system are present on the CD-ROM. They also reflect a breakdown of trust.

5.6 Summary and comments

This chapter has documented the scenarios investigated using my simulator. The main emphasis was on comparing how a discrete and fuzzy logic combination of direct and reputation trust performed using either the discrete or the beta reputation system.

Summary
Initially the two methods for combining direct and reputation trust were examined. In both cases they used the discrete reputation system, and the first scenario depicted an environment where all of the resources exhibited good behaviour. This meant that all of the users correctly informed the needed CPU time for their jobs and the sites correctly allocated the informed amount of CPU time. Not surprisingly this lead to a general increase in the direct, the reputation and the combined trust values that resources had for each other. A quick increase in the various trust values was observed. Both the discrete and the fuzzy logic method arrived on average at trust values above 0.6 after at least 5 events. It was however, observed that when using fuzzy logic, slightly fewer events where needed. The sec-

ond scenario depicted an environment where one site potentially misbehaved. This meant that the site would often allocate an inaccurate amount of CPU time to the user's jobs, but occasionally it would allocate the correct amount. Using the discrete reputation system this lead to a decrease of the various trust values for both the discrete and the fuzzy logic combination method. This was of course expected but it was however also observed that significantly more events were needed for the discrete combination method for the trust value to decrease below -0.6. By changing the ratio between direct and reputation trust, it was possible to reduce the number of events needed. By changing the ratio the discrete and fuzzy logic combination methods revealed a similar progression of trust and similar number of events were needed for the trust values to decrease below -0.6. During the experiment the effect of inaccurate ratings from the site with potentially bad behaviour was also observed when other sites were calculating reputation trust. The effect of the inaccurate ratings were however not influential enough to break the trust system. The sites were still able to identify the behaviour of a user to be good behaviour.
The second part of the experimenting phase examined the two reputation systems under similar circumstances as the previous experiments. A significantly slower progression of reputation trust was observed when using the beta reputation system. This was the case when the all of the resources exhibited good behaviour as well as when one site exhibited potentially bad behaviour. This meant that a significantly larger number of events was needed for trust values to increase or decrease below a given trust value. As previously observed putting more emphasis on direct trust would result in comparable performance between the discrete and the fuzzy logic combination method.
The third part of the experiments examined several scenarios where the behaviour of the resources changed. It was interestingly noted that using the fuzzy logic method in combination with the discrete reputation system meant that a wrong conclusion was drawn. This had to do with the design of the fuzzy membership functions and rule base combined with the fact that old values were forgotten when using the discrete reputation system. The discrete method using either reputation system and the fuzzy logic method using the beta reputation system did however arrive at similar conclusions. It was noted that the discrete method using the discrete reputation system almost instantly revealed the change in trust value when the behaviour of a resource changed. When the discrete method was used in combination with the beta reputation system the change was slower as was the case when the fuzzy logic method was used with the beta reputation system. It was noticed that when the discrete method was used the combined trust values seemed to span a narrower interval of trust values. This was not the case when using fuzzy logic. This narrower interval of trust values clearly showed the effect of old reputation values being used when calculating reputation trust when using the beta reputation system. The earlier experiments indicated that putting more emphasis on direct trust would cause the combined trust value to span a wider interval when using the discrete method. The forgetting factor was also investigated for the beta reputation system. The effect was apparent as trust progressed even slower when the forgetting factor decreased. The effect was however an even narrower span of trust values. Using the fuzzy logic method the combined trust values were able to span a wide interval, even though the reputation trust changed very slowly.
The remaining experiments explored scenarios where a resource arrives to the running grid

system. For all cases the arriving resource was judged fairly and the experiment showed that arriving resources would not be kept out of the system. The effect of the trust threshold was also simulated and it revealed that resources can protect themselves with a threshold. Similarly the alliance simulation revealed the importance of having trusted alliances. With the use of a strong alliance a resource was able to significantly protect itself against inaccurate ratings. The final scenario explored a breakdown of trust. Like a real world scenario trust broke down and resources were unable to distinguish between good and bad behaviour.

Comments

Although the simulated scenarios only contain a relatively small number of users and sites, where a real world scenario would contain hundreds or even thousands of resources the simulated scenarios can still be considered relevant, since in a real world scenario the number of resources that a single resource would be in contact with would also be fairly limited. Naturally a site would have contact with more users than four. But it would however only gather reputation information from a limited number of sites. In real life it would be improbable for a site to gather reputation information from thousands of sites. The same would naturally be the case for users. The fact that the experiments described in this chapter also rely on approximately 1 gigabyte of simulation data would make the data analysis of more resources impractical.

One of the very interesting observations from the experiments was that when using the fuzzy logic method for combining direct and reputation trust, changes between scenarios did not have to be performed. When the beta reputation system was used it did not matter if resource behaved or misbehaved, it was detected in both cases. With the discrete combination method using the beta reputation system it was occasionally needed to shift the ratio between direct and reputation trust to allow the trust values to increase or decrease beyond a certain value. However, such a change meant that the parameters of the model was changed to fit the scenario. It most cases it would be more sought after that the model merely reflected the scenario. The flexibility when using fuzzy logic was reflected in the simulated scenarios.

It is easy to see why the discrete approach has attracted so much attention in the design of trust models. In all cases it measures the changes in trust very quickly and accurately. When setting the discrete parameters to suitable values it is easy to obtain good results. However, these need to be changed between scenarios. Consequently it is superior when measuring trust, but it does not exhibit the same flexibility as the use of fuzzy logic does.

As far as reputation systems, a similar observation was made. The discrete system instantly reflects the changes in behaviour but it does not reflect how the previous behaviour of a resource was. When using the beta reputation system it could be noted that the old reputation trust values influenced the calculations of new reputation trust values. However a major disadvantage when using the beta reputation system is the decision of what base weight to use when receiving ratings. This would probably be an area of the beta reputation system worth investigating further. As described in section 2.3.3 the beta reputation system uses the expected value when calculating the reputation trust value from a posterior beta distribution. It would be interesting to calculate the reputation trust value using for example the mode or the medium of the posterior distribution, however given the time restriction

for this project it was impossible to try so here. The beta reputation system did prove advantages over the discrete reputation system, but there are certainly areas that should receive further investigation.

Chapter 6

Discussion

In this chapter the results of the project will be summarised and set against the motivation for the project, which were described in the beginning. Furthermore possible future work for incremental trust in grid systems will be discussed and finally some concluding remarks are given.

6.1 Achieved results

In this thesis I have examined the notion of incremental trust in a grid system. Initially trust was analysed and the needed considerations to use such a notion where discussed. This was followed by a presentation of a wide array of models using discrete, Bayesian or fuzzy logic methods for establishing and modelling the progression of trust. A subsequent section narrowed the field of investigation by considering the notions particularly relevant in a grid system. Upon this I designed a grid simulator for the purpose of examining the progression of trust in a grid system. The main focus was comparing models that used a discrete method for the combination of direct and reputation trust and a fuzzy logic method for the combination of direct and reputation trust. Furthermore the discrete and the beta reputation system were to be examined. Several of the models presented by researchers used a discrete combination for direct and reputation trust, however none of the models used fuzzy logic for combining direct and reputation trust. My simulation investigated the use of fuzzy logic in this context. Similarly none of the presented models included a detailed description on how to measure the behaviour of resources. Using CPU time, a real world quantifiable parameter was used to measure behaviour leading to the calculation of a trust value. A notion of alliances was also described and investigated.
My experiments showed that using fuzzy logic to combine direct and reputation trust meant a flexible approach, as it was able to cope with the varying scenarios without the need to change fuzzy logic membership function or rule base. The beta reputation system also showed an intended slower progression of reputation trust.

A very interesting result of my experiments was the possibility of comparing the progression

of various trust notions using different models. Much of the literature written on trust management also relies on simulation result, but in most cases the simulation efforts only include investigation of one model. When reviewing the simulation results from other researchers there is no evidence that inconclusively proves that their model is superior to other model. With the comparison between the methods that I have done, it was apparent when the parameters of a model had to be changed to fit the scenario. In the earliest experiments the different models showed similar progression of trust. However, when the behaviour of resources started to change mid simulation, the beta reputation system revealed an intended slower progression of reputation trust. When the combination of direct and reputation trust was done using the discrete method, a change in the ratio between direct and reputation trust was needed for the simulation output to reflect the intended scenario. When using fuzzy logic no change was needed.

6.2 Future work

Future work would need to include the examination of incremental trust that is based on more parameters and different contexts. I my simulations I have used CPU time as the parameter to measure the behaviour of a resource. By including memory and storage requirements for jobs as well as an intrusion detection system value, more scenarios could be investigated. It would also be possible to use for example the library requirements of jobs to distinguish between jobs and the needed trust requirements for a job. Depending on these trust requirements and different contexts a more versatile trust threshold scheme could be developed. For example a resource could also define its trust thresholds based on the amount of trust it has in the various resources. Instead of using a fixed trust threshold, a resource could define what it perceives as trustworthy on the basis of what constitutes an average trust value.

A more detailed examination of the beta reputation system is also needed. For example the base weight of the rating would certainly be a field that should receive further investigation. As described earlier, using other factors than the expectation value to calculate a reputation value from the posterior beta distribution is also a topic that is of great interest.

The flexibility of fuzzy logic also sparked the idea of using fuzzy logic for combining the trust from an interaction with the previous trust value to form the notion of direct trust. Since these trust values are also different types of trust the same arguments that make fuzzy logic suitable might also make fuzzy logic suitable for calculation of direct trust. I did however choose not to perform these experiments as time did not permit it.
The design of the fuzzy logic memberships functions and rule base are heuristics merely based on the related work from Kai Hwang and Shanshan Song. Although these memberships functions and rule base work well in the simulated scenarios it is not certain that this would be the case in every scenario. Future work should also include an examination of different membership functions and rule bases.

The question remains whether or not a fuzzy logic combination of direct and reputation

trust calculated using the beta reputation system would work in a real world grid system scenario. Particularly whether or not it would be practical given the fact that the progression of trust is intended to be slow. When performing the experiments I noted the number of events needed for a trust value to increase or decrease above or below a given value. This was naturally dependent on the direct weight and the smoothing factor chosen. With the values that were used, a fast change of trust value was obtained as I found it desirable from a simulation point of view. With different values a slower trust progression would be obtained. The different tables in the result section 5.5 showed the number of events needed for trust to progress above or below a given trust value. These values indicate that the trust system is responsive enough for a real world grid system to measure the progression of trust.

It should naturally also be considered how a trust management scheme would fit into a real world grid system and how the information obtained from the trust system should be used. For example in the Globus framework[1], the security module could be extended to include methods for calculating trust using fuzzy logic and the beta reputation system. It would probably not be used as a form of authentication, but more as a convenient way for resource to calculate the expected outcome of an interaction. For example the sites could use the trust information to decide on a more efficient scheduling of jobs leading to a better allocation of resources. Likewise the users could use the trust information to help them decide where to send their jobs. When a site has a high trust rating the user would have a higher expectation of receiving a fair processing of its jobs.
Most of the models presented by the researchers rely on reputation information when trust has to be established. In a real world grid system the spread of information can however not be assumed to happen easily. Depending on the grid system framework possible solutions could be investigated. For example in the Globus framework the job manager in the grid system is in charge of directing jobs to sites and return results to users. This could be exploited for propagation of trust. For example when a user submits a job to a site it will first have to go through the job manager. Likewise when the job is returned to the user it passes through the job manager. The job manager could append relevant trust information to the job when passing the job to the resource. Practical experiments would reveal if it would be possible without seriously degrading the performance of the job manager.

6.3 Concluding remarks

This project has been an interesting experience for me. First of all I have had the opportunity to explore the notion of trust in computer security which has been both challenging and exciting. Furthermore the project gave me a better understanding of the requirements from a grid system's point of view. And last, but not least the simulation experiments gave me a lot of experience in the examination and interpretation of significant amounts of data. All experiences that I hope to utilise when entering the workplace environment.
It is my belief that the use of incremental trust particularly in grid systems will continue to receive significant attention in the years to come due to two facts. First of all the amount of

data processing requirements will continue to grow leading to an increase in the use of grid systems. Secondly the use of incremental trust will certainly provide advantages as more and more resources team up to exploit the powers of a grid systems.

Bibliography

[1] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and grid technologies for wide-area distributed computing. *Softw. Pract. Exper.*, 32(15):1437–1466, 2002.

[2] Audun Jøsang, Claudia Keser, and Theodosis Dimitrakos. Can we manage trust? In *iTrust*, pages 93–107, 2005.

[3] Audun Jøsang. The right type of trust for distributed systems. In *NSPW '96: Proceedings of the 1996 workshop on New security paradigms*, pages 119–131. ACM Press, 1996.

[4] Colin Boyd Audun Jøsang, Roslan Ismail. A survey of trust and reputation systems for online service provision. In *Decision Support Systems*, 2005.

[5] Audun Jøsang and Stéphane Lo Presti. Analysing the relationship between risk and trust. In *iTrust*, pages 135–145, 2004.

[6] V. Cahill, E. Gray, J.-M. Seigneur, C. Jensen, Y. Chen, B. Shand, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nixon, G. Serugendo, C. Bryce, M. Carbone, K. Krukow, and M. Nielsen. Using Trust for Secure Collaboration in Uncertain Environments. In *Pervasive Computing Magazine*, volume 2, pages 52–61. IEEE, 2003.

[7] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing.* Prentice Hall Professional Technical Reference, 2002.

[8] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. 2005.

[9] Ghada Derbas, Ayman I. Kayssi, Hassan Artail, and Ali Chehab. Trummar - a trust model for mobile agent systems based on reputation. In *ICPS*, pages 113–120, 2004.

[10] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 640–651, New York, NY, USA, 2003. ACM Press.

[11] Audun Jøsang and Roslan Ismail. The beta reputation system. In *Proceedings of the 15th Bled Conference on Electronic Commerce*, June 2002.

[12] Audun Jøsang, Shane Hird, and Eric Faccer. Simulating the effect of reputation systems on e-markets. In *iTrust*, pages 179–194, 2003.

[13] Audun Jøsang, Jadwiga Indulska, and Andrew Whitby. Filtering out unfair ratings in bayesian reputation systems. In *Proceedings of the 7th Int Workshop on Trust in Agent Societies*, 2004.

[14] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164+. IEEE Computer Society, 1996.

[15] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. Keynote: Trust management for public-key infrastructures. In *Proceedings of the 6th International Workshop on Security Protocols*, pages 59–63. Springer-Verlag, 1998.

[16] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The role of trust management in distributed systems security. pages 185–210, 1999.

[17] Farag Azzedin and Muthucumaru Maheswaran. Towards trust-aware resource management in grid computing systems. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 452. IEEE Computer Society, 2002.

[18] Farag Azzedin and Muthucumaru Maheswaran. Integrating trust into grid resource management systems. In *ICPP '02: Proceedings of the 2002 International Conference on Parallel Processing (ICPP'02)*, page 47. IEEE Computer Society, 2002.

[19] A. Farag and M. Muthucumaru. Evolving and managing trust in grid computing systems, 2002.

[20] Farag Azzedin and Muthucumaru Maheswaran. A trust brokering system and its application to resource management in public-resource grids. In *IPDPS*, 2004.

[21] Yao Wang and Julita Vassileva. Bayesian network-based trust model. In *Web Intelligence*, pages 372–378, 2003.

[22] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Mach. Learn.*, 29(2-3):131–163, 1997.

[23] Lik Mui, Mojdeh Mohtashemi, and Ari Halberstadt. A computational model of trust and reputation for e-businesses. In *HICSS*, page 188, 2002.

[24] Daniel W. Manchala. Trust metrics, models and protocols for electronic commerce transactions. In *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, page 312, Washington, DC, USA, 1998. IEEE Computer Society.

[25] Shanshan Song and Kai Hwang. Trusted grid computing with security assurance and resource optimization. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS-2004)*, sep 2004.

[26] Shanshan Song, Kai Hwang, and Mikin Macwan. Fuzzy trust integration for security enforcement in grid computing. In *NPC*, pages 9–21, 2004.

[27] Timothy J. Ross. *Fuzzy Logic with Engineering Applications*. McGraw-Hill, New York, 1995.

[28] Samuel A. Schmitt. *Measuring Uncertainty: An elementary introduction to Bayesian Statistics*. Addison-Wesley, Reading, Massachusetts, USA, 1969.

[29] International Electrotechnical Commission. Fcl syntax draft 1.0, 1997. http://www.fuzzytech.com/binaries/ieccd1.pdf.

[30] Robert Lafore. *Object-oriented programming in C++ (2nd ed.)*. Waite Group Press, Corte Madera, CA, USA, 1995.

Appendix A

User Guide

Compiling
The Free Fuzzy Logic Library(FFLL) has been compiled using version 3.3.5 of the GCC compiler. If another compiler is used, FFLL must be recompiled. Provided that the directory is writable, this can be achieved by changing to the FFLL directory and issuing the command:

```
#make static
```

The file libffll.a should be moved to the lib directory in the source_code directory. If the GNU scientific library (gsl) is installed at a standard location and the directory is writable, the simulation can be compiled by issuing the command:

```
#qmake grid-simulator.pro && make
```

If gsl is installed at a nonstandard location the file randomgenerator.h must be edited before compiling the simulation.

Running
The program can be executed by issuing the command:

```
#sh grid-simulator
```

Optionally a random number seed can be specified as an argument to the program. The program needs two files to be present in the program directory. The resource.fcl file containing the fuzzy logic membership functions and fuzzy rule base, and the file simulation.conf. The last file specifies the configuration of the simulation. If the file resource.fcl has been edited then the file has to be converted to DOS format by issuing the command:

```
#unix2dos resource.fcl
```

Simulation configuration

Here the syntax of the configuration file is described. The configuration file is segregated into sections. The first section, denoted [Config], describes the main options for the simulation. It consists of a list of mandatory statements. The following lists an example with a brief description:

```
[Config]
users=4                 # initial number of users
sites=4                 # initial number of sites
time=100000             # simulation time
mu=25.0                 # mean of Poisson distribution
trust_eval=discrete     # combination method "discrete" or "fuzzy"
rep_eval=beta           # reputation method "discrete" or "beta"
direct_weight=0.6       # direct weight of the experiences
smoothing=1.0           # smoothing factor
stranger=0.0            # initial trust value to strangers
discrete_ratio=0.5      # ratio between direct and reputation trust
forgetting=0.4          # forgetting factor
seed=time               # random number seed. "time" or some integer
path=/opt/experiments/ # path where the results are saved
```

Most of the options are self explaining. It should however be noted that even though only the discrete combination method uses the discrete_ratio statement it must be specified even when fuzzy logic is used. The same applies with the forgetting factor as it also needs to be specified when using the discrete reputation method. When a random number seed is specified at the command line it takes precedence over what is stated in the configuration file.

The second section, denoted [Default], lists the default values for the resources. This also consists for a set of mandatory statements. The following lists an example with a brief description:

```
[Default]
i=1             # the default min behavioural value
j=1             # the default max behavioural value
threshold=-1.0 # default trust threshold. Must be -1<=x<=1
```

It should be emphasised that the min and max values are labelled i and j respectively in the configuration file.

The remaining sections of the configuration file are optional. The first is the [Groups] section. This can be used to specify alliances between resource. The following lists an example :

```
[Groups]
u0= 1 2 3  # groups for user u0
u1= 1 2    # groups for user u1
s0= 1 3    # groups for user u3
```

In the example user $u0$ has an alliance with user $u1$ and site $s0$. In both case the alliance strength is 3 as group 0 is shared by all resources. User $u1$ has an alliance strength of 2 with site $s0$.
The second optional section is the [Add] section. This section specifies the addition of new resources to a running simulation. The following lists an example:

```
[Add]
time=1000 s4 i=1 j=1 g=1 threshold=-0.4
```

The section specifies the time the resource is going to be added to the simulation, the id of the resource (that also denotes its type). The i and j values denotes the min and max values, g denotes the group memberships and threshold denotes the resource's trust threshold. All of these values are mandatory when adding a resource. Several resources could naturally be added.
The final type of section denotes specific values for a resource. The following lists an example:

```
(s0)                # resource id
i=20                # s0's min behavioural value
j=20                # s0's max behavioural value
time=15000 i=1 j=1  # change in behaviour
threshold=0.0       # s0's trust threshold
```

These values can be specified for an optional number of the resources. The behavioural values can be excluded and the change in behaviour and threshold are naturally also optional. Multiple changes in behaviour can be specified.

The next page lists a full configuration file example.

```
[Config]
users=4
sites=4
time=100000
mu=25.0
trust_eval=fuzzy
rep_eval=beta
direct_weight=0.6
smoothing=1.0
stranger=0.0
discrete_ratio=0.5
forgetting=0.4
seed=500
path=/opt/experiments/

[Default]
i=1
j=1
threshold=-0.4

[Groups]
u1= 1 2 3
u2= 1 2

[Add]
time=1000 s4 i=1 j=1 g=1 threshold=-0.4
time=1500 u4 i=20 j=20 g= threshold=-1.0

(s1)
i=20
j=20
time=15000 i=1 j=1
time=30000 i=20 j=20
time=45000 i=5 j=5

(u0)
i=3
j=3
threshold=0.0
```

Appendix B

Contents of CD-ROM

The enclosed CD-ROM contains the following directories and files:

source_code contains the source code for the program. Included is also the default fcl file.

experiments contains the plots of the experiments. Subdirectories describes the experiment number. Additional subdirectories describe what type of plot it contains and the methods used. For example the directory name Fuzzy-Discrete denotes that the fuzzy logic combination is used with the discrete reputation system. The simulation configuration file is also included in the plot directories.

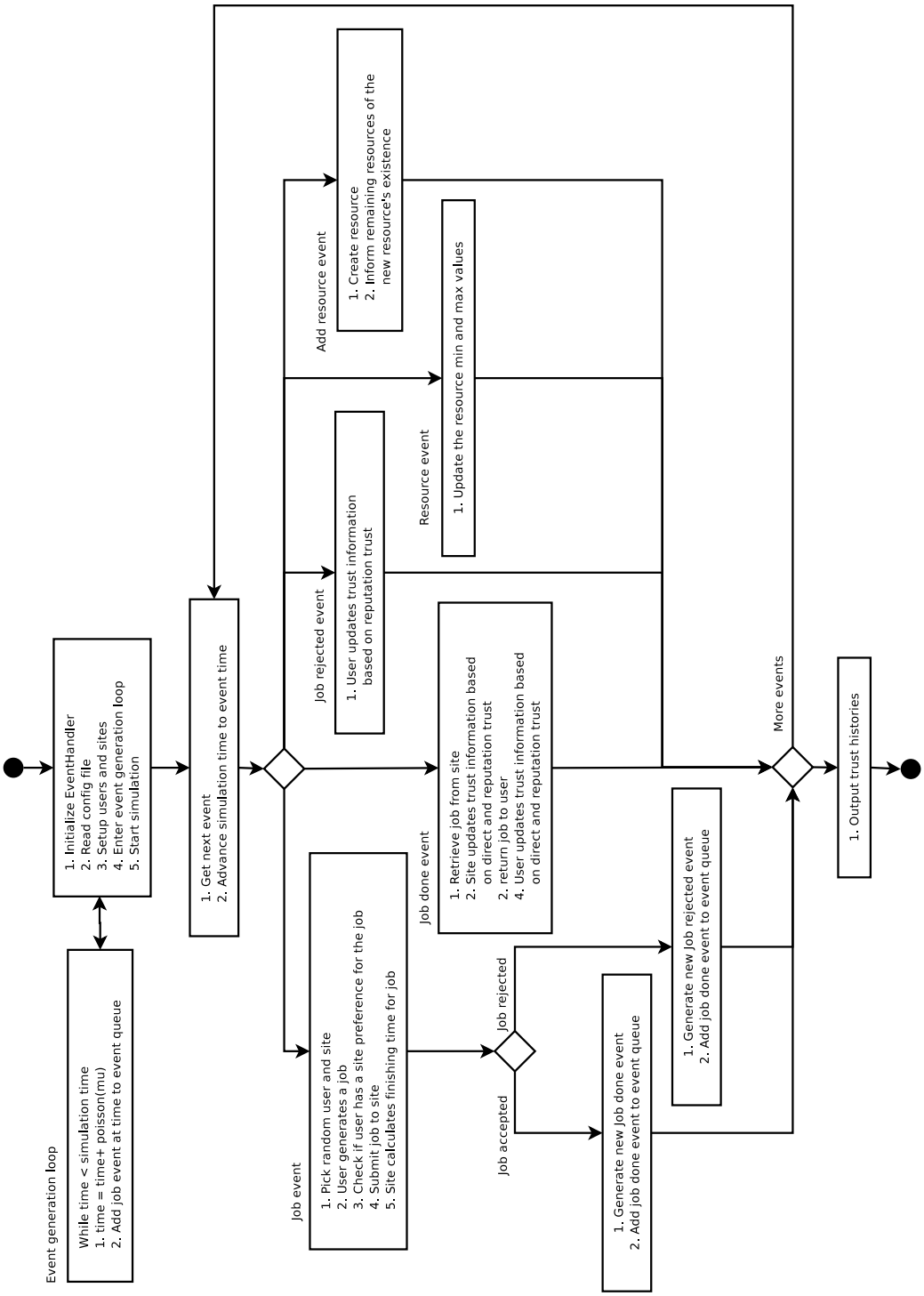
scripts contains the various scripts and programs used for investigating the results of the experiments. Each file contains a brief description on how to use the script and what the output is.

FFLL contains a port of the Free Fuzzy Logic Library.

report contains a PDF file of the report.

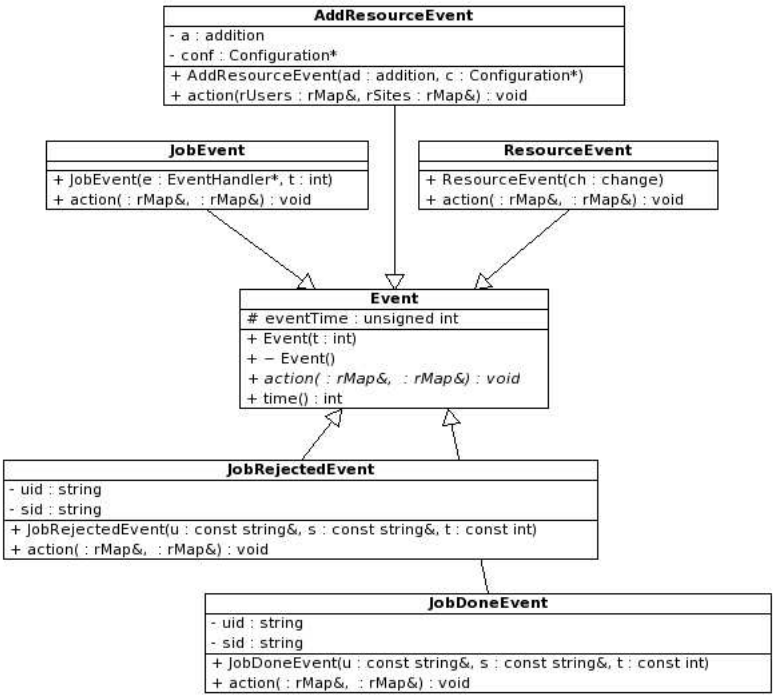
Appendix C

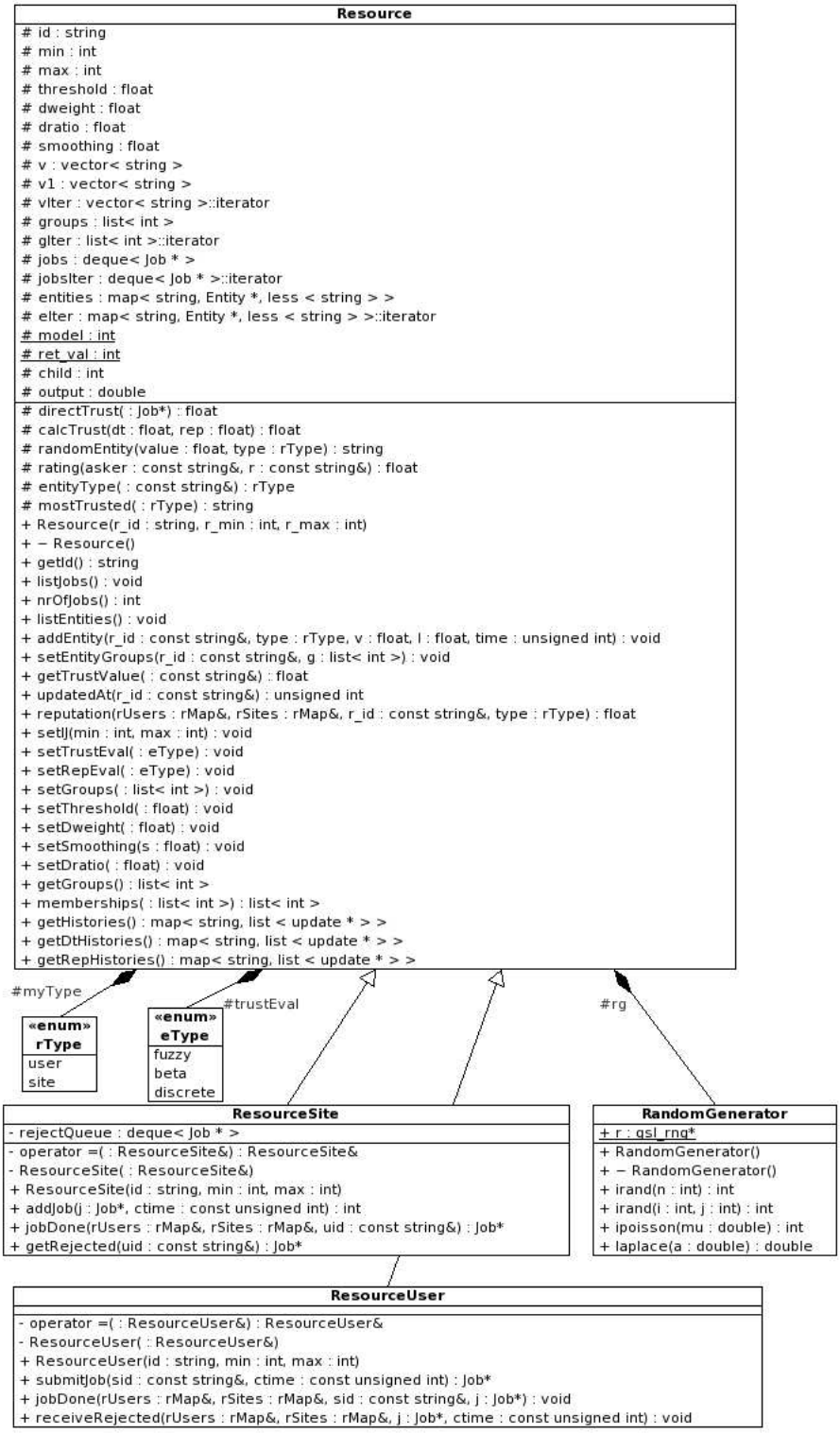
Simulation flow of control

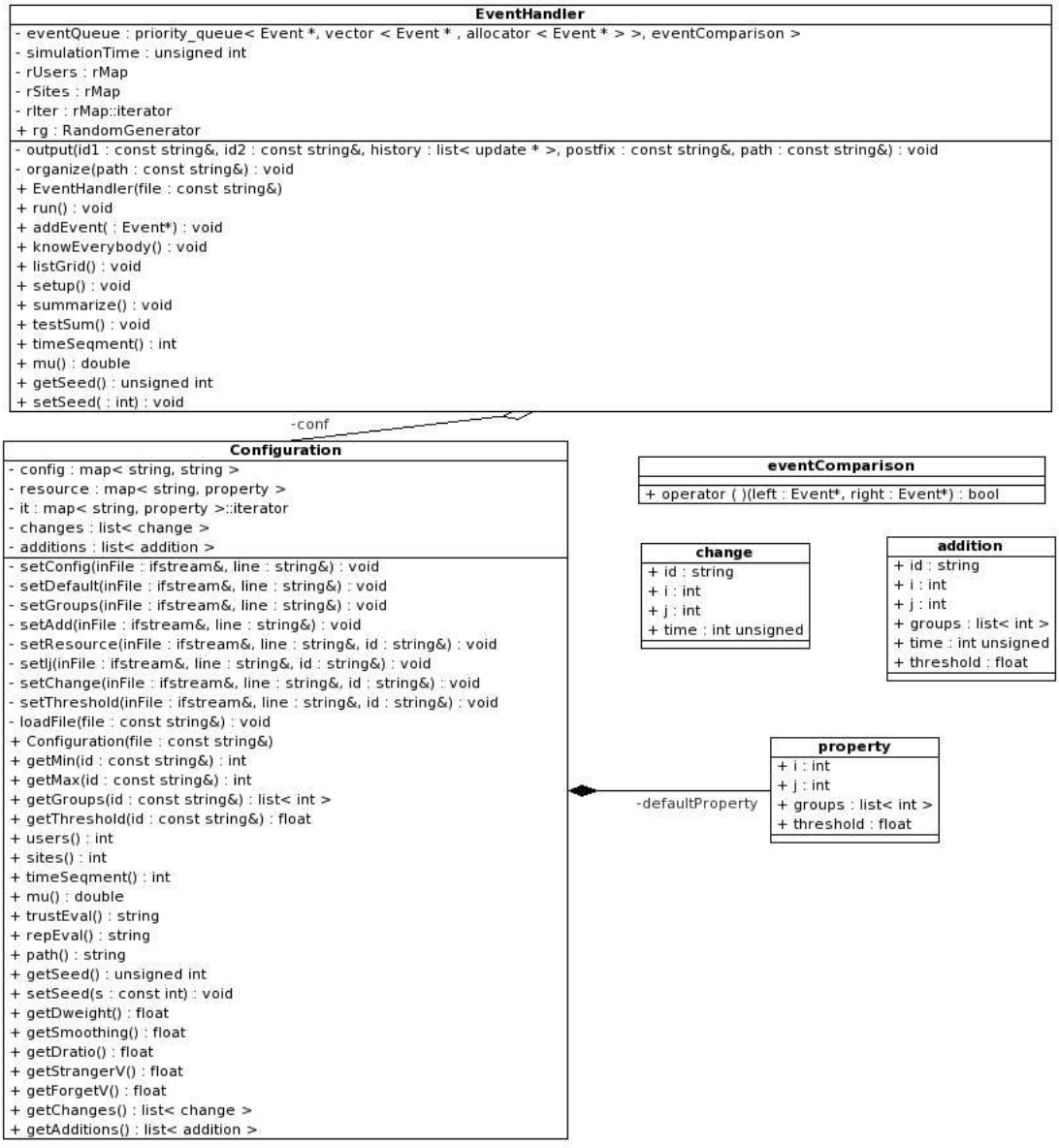

1. Initialize EventHandler
2. Read config file
3. Setup users and sites
4. Enter event generation loop
5. Start simulation
Event generation loop
While time < simulation time
1. time = time+ poisson(mu)
2. Add job event at time to event queue
1. Get next event
2. Advance simulation time to event time
Job event
1. Pick random user and site
2. User generates a job
3. Check if user has a site preference for the job
4. Submit job to site
5. Site calculates finishing time for job
Job rejected
Job accepted
1. Generate new Job done event
2. Add job done event to event queue
1. Generate new Job rejected event
2. Add job done event to event queue
Job done event
1. Retrieve job from site
2. Site updates trust information based on direct and reputation trust
2. return job to user
4. User updates trust information based on direct and reputation trust
Job rejected event
1. User updates trust information based on reputation trust
Add resource event
1. Create resource
2. Inform remaining resources of the new resource's existence
Resource event
1. Update the resource min and max values
More events
1. Output trust histories

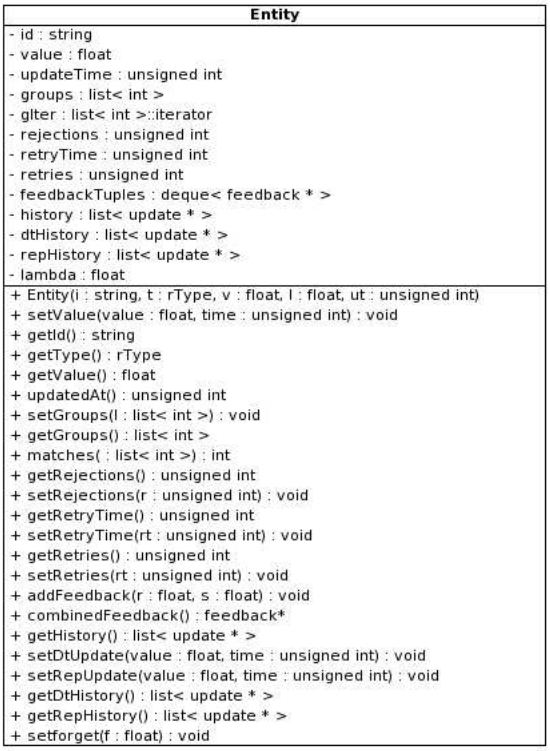
Appendix D

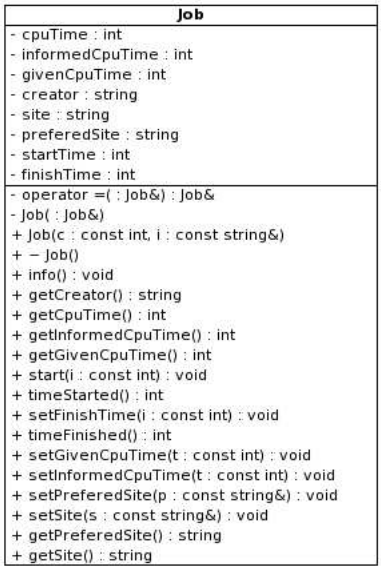
Class diagrams



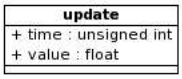

Resource
id : string
min : int
max : int
threshold : float
dweight : float
dratio : float
smoothing : float
v : vector< string >
v1 : vector< string >
vIter : vector< string >::iterator
groups : list< int >
gIter : list< int >::iterator
jobs : deque< Job * >
jobsIter : deque< Job * >::iterator
entities : map< string, Entity *, less < string > >
eIter : map< string, Entity *, less < string > >::iterator
model : int
ret_val : int
child : int
output : double
directTrust(: Job*) : float
calcTrust(dt : float, rep : float) : float
randomEntity(value : float, type : rType) : string
rating(asker : const string&, r : const string&) : float
entityType(: const string&) : rType
mostTrusted(: rType) : string
+ Resource(r_id : string, r_min : int, r_max : int)
+ ~ Resource()
+ getId() : string
+ listJobs() : void
+ nrOfJobs() : int
+ listEntities() : void
+ addEntity(r_id : const string&, type : rType, v : float, l : float, time : unsigned int) : void
+ setEntityGroups(r_id : const string&, g : list< int >) : void
+ getTrustValue(: const string&) : float
+ updatedAt(r_id : const string&) : unsigned int
+ reputation(rUsers : rMap&, rSites : rMap&, r_id : const string&, type : rType) : float
+ setIJ(min : int, max : int) : void
+ setTrustEval(: eType) : void
+ setRepEval(: eType) : void
+ setGroups(: list< int >) : void
+ setThreshold(: float) : void
+ setDweight(: float) : void
+ setSmoothing(s : float) : void
+ setDratio(: float) : void
+ getGroups() : list< int >
+ memberships(: list< int >) : list< int >
+ getHistories() : map< string, list < update * > >
+ getDtHistories() : map< string, list < update * > >
+ getRepHistories() : map< string, list < update * > >
#myType
#trustEval
#rg
«enum»
rType
user
site
«enum»
eType
fuzzy
beta
discrete
ResourceSite
- rejectQueue : deque< Job * >
- operator =(: ResourceSite&) : ResourceSite&
- ResourceSite(: ResourceSite&)
+ ResourceSite(id : string, min : int, max : int)
+ addJob(j : Job*, ctime : const unsigned int) : int
+ jobDone(rUsers : rMap&, rSites : rMap&, uid : const string&) : Job*
+ getRejected(uid : const string&) : Job*
RandomGenerator
+ r : gsl_rng*
+ RandomGenerator()
+ ~ RandomGenerator()
+ irand(n : int) : int
+ irand(i : int, j : int) : int
+ ipoisson(mu : double) : int
+ laplace(a : double) : double
ResourceUser
- operator =(: ResourceUser&) : ResourceUser&
- ResourceUser(: ResourceUser&)
+ ResourceUser(id : string, min : int, max : int)
+ submitJob(sid : const string&, ctime : const unsigned int) : Job*
+ jobDone(rUsers : rMap&, rSites : rMap&, sid : const string&, j : Job*) : void
+ receiveRejected(rUsers : rMap&, rSites : rMap&, j : Job*, ctime : const unsigned int) : void


EventHandler
- eventQueue : priority_queue< Event *, vector < Event * , allocator < Event * > >, eventComparison >
- simulationTime : unsigned int
- rUsers : rMap
- rSites : rMap
- rIter : rMap::iterator
+ rg : RandomGenerator
- output(id1 : const string&, id2 : const string&, history : list< update * >, postfix : const string&, path : const string&) : void
- organize(path : const string&) : void
+ EventHandler(file : const string&)
+ run() : void
+ addEvent(: Event*) : void
+ knowEverybody() : void
+ listGrid() : void
+ setup() : void
+ summarize() : void
+ testSum() : void
+ timeSeqment() : int
+ mu() : double
+ getSeed() : unsigned int
+ setSeed(: int) : void
-conf
Configuration
- config : map< string, string >
- resource : map< string, property >
- it : map< string, property >::iterator
- changes : list< change >
- additions : list< addition >
- setConfig(inFile : ifstream&, line : string&) : void
- setDefault(inFile : ifstream&, line : string&) : void
- setGroups(inFile : ifstream&, line : string&) : void
- setAdd(inFile : ifstream&, line : string&) : void
- setResource(inFile : ifstream&, line : string&, id : string&) : void
- setIj(inFile : ifstream&, line : string&, id : string&) : void
- setChange(inFile : ifstream&, line : string&, id : string&) : void
- setThreshold(inFile : ifstream&, line : string&, id : string&) : void
- loadFile(file : const string&) : void
+ Configuration(file : const string&)
+ getMin(id : const string&) : int
+ getMax(id : const string&) : int
+ getGroups(id : const string&) : list< int >
+ getThreshold(id : const string&) : float
+ users() : int
+ sites() : int
+ timeSeqment() : int
+ mu() : double
+ trustEval() : string
+ repEval() : string
+ path() : string
+ getSeed() : unsigned int
+ setSeed(s : const int) : void
+ getDweight() : float
+ getSmoothing() : float
+ getDratio() : float
+ getStrangerV() : float
+ getForgetV() : float
+ getChanges() : list< change >
+ getAdditions() : list< addition >
eventComparison
+ operator ( )(left : Event*, right : Event*) : bool
change
+ id : string
+ i : int
+ j : int
+ time : int unsigned
addition
+ id : string
+ i : int
+ j : int
+ groups : list< int >
+ time : int unsigned
+ threshold : float
property
+ i : int
+ j : int
+ groups : list< int >
+ threshold : float
-defaultProperty


Entity
- id : string
- value : float
- updateTime : unsigned int
- groups : list< int >
- glter : list< int >::iterator
- rejections : unsigned int
- retryTime : unsigned int
- retries : unsigned int
- feedbackTuples : deque< feedback * >
- history : list< update * >
- dtHistory : list< update * >
- repHistory : list< update * >
- lambda : float
+ Entity(i : string, t : rType, v : float, l : float, ut : unsigned int)
+ setValue(value : float, time : unsigned int) : void
+ getId() : string
+ getType() : rType
+ getValue() : float
+ updatedAt() : unsigned int
+ setGroups(l : list< int >) : void
+ getGroups() : list< int >
+ matches(: list< int >) : int
+ getRejections() : unsigned int
+ setRejections(r : unsigned int) : void
+ getRetryTime() : unsigned int
+ setRetryTime(rt : unsigned int) : void
+ getRetries() : unsigned int
+ setRetries(rt : unsigned int) : void
+ addFeedback(r : float, s : float) : void
+ combinedFeedback() : feedback*
+ getHistory() : list< update * >
+ setDtUpdate(value : float, time : unsigned int) : void
+ setRepUpdate(value : float, time : unsigned int) : void
+ getDtHistory() : list< update * >
+ getRepHistory() : list< update * >
+ setforget(f : float) : void


Job
- cpuTime : int
- informedCpuTime : int
- givenCpuTime : int
- creator : string
- site : string
- preferedSite : string
- startTime : int
- finishTime : int
- operator =(: Job&) : Job&
- Job(: Job&)
+ Job(c : const int, i : const string&)
+ ~ Job()
+ info() : void
+ getCreator() : string
+ getCpuTime() : int
+ getInformedCpuTime() : int
+ getGivenCpuTime() : int
+ start(i : const int) : void
+ timeStarted() : int
+ setFinishTime(i : const int) : void
+ timeFinished() : int
+ setGivenCpuTime(t : const int) : void
+ setInformedCpuTime(t : const int) : void
+ setPreferedSite(p : const string&) : void
+ setSite(s : const string&) : void
+ getPreferedSite() : string
+ getSite() : string


feedback
+ r : float
+ s : float


update
+ time : unsigned int
+ value : float

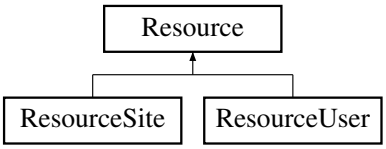
Appendix E

API

E.1 Resource Class Reference

The resource class contains methods and variables that are common for the users and sites.

Inheritance diagram for Resource::



Public Member Functions

- **Resource** (string r_id, int r_min, int r_max)
- string **getId** ()

  Returns the resource id.

- void **listJobs** ()

  Lists the jobs in the job queue (used for testing).

- int **nrOfJobs** ()

  Returns the number of jobs in the job queue (used for testing).

- void **listEntities** ()

  Lists the known resources (used for testing).

- void **addEntity** (const string &r_id, rType type, float **v**, float l, unsigned int time)

- void **setEntityGroups** (const string &r_id, list< int > g)
- float **getTrustValue** (const string &r_id)

  Returns the trust value for resource r_id.

- unsigned int **updatedAt** (const string &r_id)

  Returns the last time where resource r_id was updated.

- float **reputation** (rMap &rUsers, rMap &rSites, const string &r_id, rType type)
- void **setIJ** (int **min**, int **max**)

  Sets the i (min) and the j (max) values.

- void **setTrustEval** (eType)

  Sets the trust combination method(fuzzy or discrete).

- void **setRepEval** (eType)

  Sets the reputation method(discrete or beta).

- void **setGroups** (list< int >)

  Sets the resource's groups.

- void **setThreshold** (float)

  Sets the trust threshold.

- void **setDweight** (float)

  Sets the direct weight.

- void **setSmoothing** (float s)

  Sets the smoothing factor.

- void **setDratio** (float)

  Sets the discrete ratio (between direct and reputation trust).

- list< int > **getGroups** ()

  Returns the list of groups that the resource is a member of.

- list< int > **memberships** (list< int >)

  Returns a list of the shared groups.

- map< string, list< **update** * > > **getHistories** ()

  Returns all combined trust histories.

- map< string, list< **update** * > > **getDtHistories** ()

  Returns all direct trust histories.

- map< string, list< **update** * > > **getRepHistories** ()

  Returns all reputation trust histories.

Protected Member Functions

- float **directTrust** (**Job** *)
- float **calcTrust** (float dt, float rep)
- string **randomEntity** (float value, rType type)
- float **rating** (const string &asker, const string &r)
- rType **entityType** (const string &)

  Returns the type of a resource (entity).

Protected Attributes

- string **id**

  The unique id for the resource.

- rType **myType**

  The type of the resource.

- int **min**

  The min behavioural value.

- int **max**

  The max behavioural value.

- float **threshold**

  The trust threshold.

- float **dweight**

  The direct weight.

- float **dratio**

  The discrete ratio between direct and reputation trust.

- float **smoothing**

The smoothing factor.

- vector< string > **v**

  Temporary vector used for sorting.

- vector< string > **v1**

  Temporary vector used for sorting.

- list< int > **groups**

  The resource's group.

- deque< **Job** * > **jobs**

  The job queue.

- map< string, **Entity** *, less< string > > **entities**

  The map of known resources (entities).

- **RandomGenerator rg**

  The Random number generator.

Static Protected Attributes

- static int **model** = ffll_new_model()

  The fuzzy model.

Constructor & Destructor Documentation

Resource::Resource (string *r_id*, int *r_min*, int *r_max*) `[inline]`

Constructs the resource object. A child object for the fuzzy logic model will also be created.

Parameters:

r_id The resource id

r_min The min behavioural value

r_max The max hahavioural value

Member Function Documentation

void Resource::addEntity (const string & *r_id*, rType *type*, float *v*, float *l*, unsigned int *time*)

Adds a resource to the map of known resources

Parameters:

r_id The id of the resource

type The type of the resource

v The trust value for the resource

l The forgetting factor to use

time The time the resource is added to the map

float Resource::calcTrust (float *dt*, float *rep*) [protected]

Combines direct and reputation trust using either the fuzzy logic or the discrete method.

Parameters:

dt The direct trust value

rep The reputation trust value

Returns:

The combined trust value

float Resource::directTrust (Job *) [protected]

Calculates the direct trust value based on the job.

Returns:

The direct trust value

string Resource::randomEntity (float *value*, rType *type*) [protected]

Returns the id of a random resource with the specified type and with a trust level above the specified value.

Parameters:

value The minimum trust value

type The type of resource

Returns:
The random resource id

float Resource::rating (const string & *asker*, const string & *r*) `[protected]`

Returns the trust value that will be reported when asked to rate a resource.

Parameters:
asker Is the id of the asking entity

r The id of the entity being rated

float Resource::reputation (rMap & *rUsers*, rMap & *rSites*, const string & *r_id*, rType *type*)

Calculates the reputation trust value. Depending on the configuration either the discrete or the beta reputation system will be used.

Parameters:
rUsers A reference to the map containing pointer to the users

rSites A reference to the map containing pointer to the sites

r_id The id of the resource to be rated

type The type of resources being asked

void Resource::setEntityGroups (const string & *r_id*, list< int > *g*)

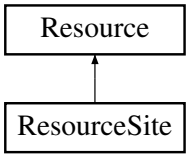
Sets the known groups for a resource (entity)

Parameters:
r_id The id of the resource

g The list of known groups

E.2 ResourceSite Class Reference

Inheritance diagram for ResourceSite::



Public Member Functions

- **ResourceSite** (string **id**, int **min**, int **max**)
- int **addJob** (**Job** *j, const unsigned int ctime)
- **Job** * **jobDone** (rMap &rUsers, rMap &rSites, const string &uid)
- **Job** * **getRejected** (const string &uid)

Private Member Functions

- **ResourceSite** & **operator=** (**ResourceSite** &)
- **ResourceSite** (**ResourceSite** &)

Private Attributes

- deque< **Job** * > **rejectQueue**

  The queue is used to store the rejected jobs before they are returned to the user.

Detailed Description

This class contains the specific methods needed by sites. This class inherits the **Resource**(p. 129) class.

Constructor & Destructor Documentation

ResourceSite::ResourceSite (ResourceSite &) `[private]`

Overwrites the default copy constructor. Eliminates the risk of accidently copying a ResourceSite object. The ResourceSite objects must be unique.

ResourceSite::ResourceSite (string *id*, int *min*, int *max*) `[inline]`

Constructs a ResourceSite object. The constructor will also specify that the resource is of type site.

Parameters:

id The unique id of the site

min The min behavioural value

max The max behavioural value

Member Function Documentation

int ResourceSite::addJob (Job * *j*, const unsigned int *ctime*)

Will maybe add a job to the site's job queue. This depends on job creators trust value. If the trust value is below the threshold then the job creator might get retried depending on the current time. If it is rejected then 0 is returned. Otherwise the finishing time for the job is returned.

Parameters:

j The job to be processed

ctime The current simulation time

Returns:

The time the job will finish.

Job * ResourceSite::getRejected (const string & *uid*)

Returns a rejected job. It is checked that the rejected job in the queue actually is created by the specified user before returning a pointer to the job.

Parameters:

uid Id of the job creator

Returns:

The rejected job

Job * ResourceSite::jobDone (rMap & *rUsers*, rMap & *rSites*, const string & *uid*)

Returns a processed job to the job creator. First the user id is checked to make sure it matches the creator of the job. Then the direct trust value for the job is calculated. Afterwards the reputation of the job creator is calculated. These are combined and the trust value for the user is updated. Finally a pointer to the finished job is returned.

Parameters:

rUsers A reference to the map containing pointers to the users

rSites A reference to the map containing pointers to the sites

uid The id of the user

Returns:

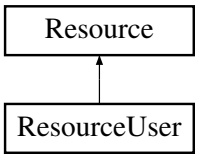
Pointer to the processed job

ResourceSite& ResourceSite::operator= (ResourceSite &) `[private]`

Overwrites the default assignment operator. Eliminates the risk of accidently assigning a ResourceSite object to another ResourceSite object

E.3 ResourceUser Class Reference

Inheritance diagram for ResourceUser::



Public Member Functions

- **ResourceUser** (string **id**, int **min**, int **max**)
- **Job** * **submitJob** (const string &sid, const unsigned int ctime)
- void **jobDone** (rMap &rUsers, rMap &rSites, const string &sid, **Job** *j)
- void **receiveRejected** (rMap &rUsers, rMap &rSites, **Job** *j, const unsigned int ctime)

Private Member Functions

- **ResourceUser** & **operator**= (**ResourceUser** &)
- **ResourceUser** (**ResourceUser** &)

Detailed Description

This class contains the specific methods needed by users. This class inherits the **Resource**(p. 129) class

Constructor & Destructor Documentation

ResourceUser::ResourceUser (ResourceUser &) [private]

Overwrites default copy constructor. Eliminates the risk of accidently copying a ResourceUser object. The ResourceUser objects must be unique.

ResourceUser::ResourceUser (string *id*, int *min*, int *max*) [inline]

Constructs a ResourceUser object. The constructor will also specify the resource is of type user.

Parameters:

id The unique id of the user

min The min behavioural value

max The max behavioural value

Member Function Documentation

void ResourceUser::jobDone (rMap & *rUsers*, rMap & *rSites*, const string & *sid*, Job * *j*)

Returns the job to the user. The job is examined and a trust calculation is performed. First it is checked that the site is equal to the site that processed the job. Then the direct trust value is calculated. Afterwards the users are asked to rate the site leading to the calculation of a reputation trust value. These trust values are combined and the trust value for the site is updated. All the trust values are saved in sequences.

Parameters:

rUsers A reference to the map containing pointers to users

rSites A reference to the map containing pointers to sites

sid The site that processed the job

j Pointer to the processed job

ResourceUser& ResourceUser::operator= (ResourceUser &) [private]

Overwrites default assignment operator. Eliminates the risk of accidently assigning a ResourceUser object to another ResourceUser object

void ResourceUser::receiveRejected (rMap & *rUsers*, rMap & *rSites*, Job * *j*, const unsigned int *ctime*)

Returns a rejected job to the user. The job can later be resubmittet to another site. First the other users are asked to rate the rejecting site in order for a reputation trust value to be calculated. If the user's combined trust value for the site is higher than the reputation trust value, then the combined trust value is set to the reputation value. Otherwise 0.1 is subtracted from the combined trust value (unless it results in a value below -1).

Parameters:

rUsers A reference to the map containing pointers to the users

rSites A reference to the map containing pointers to the sites

j Pointer to the rejected job

ctime The current simulation time

Job * ResourceUser::submitJob (const string & *sid*, const unsigned int *ctime*)

Creates a job and returns it. If there is a rejected job in the job queue then this is used. Otherwise a new job is created. If the proposed site is below the trust threshold then it is decided if the site should be retried. This depends on what the current simulation time is. The actual CPU time for the job will be a random number between 20 and 40. The informed CPU time will depend on the behavioural values.

Parameters:

sid The proposed site

ctime The current simulation time

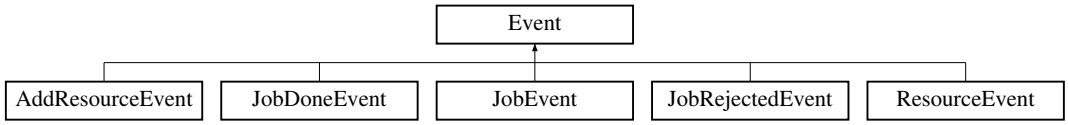
Returns:

Pointer to the job to process

E.4 Event Class Reference

Abstract class for events.

Inheritance diagram for Event::



Public Member Functions

- virtual void **action** (rMap &, rMap &)=0

  Pure virtual function for the event action.

- int **time** ()

  Returns the event time.

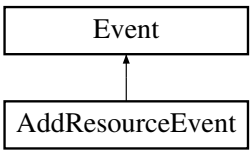
Protected Attributes

- unsigned int **eventTime**

  The time of the event.

E.5 AddResourceEvent Class Reference

Inheritance diagram for AddResourceEvent::



Public Member Functions

- **AddResourceEvent** (**addition** ad, **Configuration** *c)
- void **action** (rMap &rUsers, rMap &rSites)

Detailed Description

The purpose of this class is to provide the possibility of adding new resources to a running simulation. An AddResourceEvent object is created when the simulation is set up. When the simulation time is equal to the event time the resource is created, inserted into the resource map and the other resources are made aware that it is participating in the grid.

Constructor & Destructor Documentation

AddResourceEvent::AddResourceEvent (addition *ad*, Configuration * *c*) `[inline]`

Constructs a AddResourceEvent object

Parameters:

ad Parameters for the resource to be added.

c The configuration object for the simulation

Member Function Documentation

void AddResourceEvent::action (rMap & *rUsers*, rMap & *rSites*) `[virtual]`

The method will create the resource with the parameters specified in the addition structure and with the simulation options from the configuration object. Afterwards the resource is added to the resource map and other resources are made aware of its existens.

Parameters:

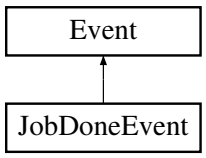
rUsers A reference to the map containing pointer to the users

rSites A reference to the map containing pointer to the sites

Implements **Event** (p. 141).

E.6 JobDoneEvent Class Reference

Inheritance diagram for JobDoneEvent::



Public Member Functions

- **JobDoneEvent** (const string &u, const string &s, const int t)
- void **action** (rMap &rUsers, rMap &rSites)

Detailed Description

The purpose of this class is to provide an event where a job can be returned from the site to the user.

Constructor & Destructor Documentation

JobDoneEvent::JobDoneEvent (const string & *u*, const string & *s*, const int *t*) `[inline]`

Constructs the JobDoneEvent.

Parameters:

u The id of the user (job creator)

s The id of the site (job processor)

t The time of the event

Member Function Documentation

void JobDoneEvent::action (rMap & *rUsers*, rMap & *rSites*) `[virtual]`

The method will extract the job from the site and return it to the job creator.

Parameters:

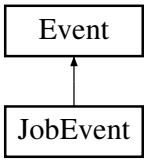
rUsers A reference to the map containing pointer to the users

rSites A reference to the map containing pointer to the sites

Implements **Event** (p. 141).

E.7 JobEvent Class Reference

Inheritance diagram for JobEvent::



Public Member Functions

- **JobEvent** (**EventHandler** *e, int t)
- void **action** (rMap &rUsers, rMap &rSites)

Private Attributes

- **EventHandler** * **eh**

  Pointer to the **EventHandler**(p. 156). *Needed to add a* **JobDoneEvent**(p. 144) *or* **JobRejectedEvent**(p. 147).

Detailed Description

The purpose of the job event is to provide an event where users can submit jobs to sites. The event will pick a random user and propose a randomly picked site. The user will create a job and the JobEvent will submit the job to the site specified by the user.

Constructor & Destructor Documentation

JobEvent::JobEvent (EventHandler * *e*, int *t*) `[inline]`

Constructs the JobEvent

Parameters:

e Pointer to the **EventHandler**(p. 156).

t Time of the JobEvent

Member Function Documentation

void JobEvent::action (rMap & *rUsers*, rMap & *rSites*) [virtual]

This method will pick a random user and propose a site. When a job is received from the user, the job will be submitted to the specified site. The site will return the time where the job will be finished. If the finishing time is 0 this means the job was rejected. A **JobRejectedEvent**(p. 147) will be created and inserted into the EventHandler's event queue. If the finishing time > 0 a **JobDoneEvent**(p. 144) will be created and inserted into the EventHandler's event queue.

Parameters:

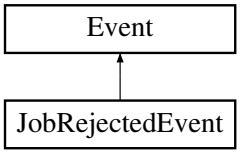
rUsers A reference to the map containing pointer to the users

rSites A reference to the map containing pointer to the sites

Implements **Event** (p. 141).

E.8 JobRejectedEvent Class Reference

Inheritance diagram for JobRejectedEvent::



Public Member Functions

- **JobRejectedEvent** (const string &u, const string &s, const int t)
- void **action** (rMap &rUsers, rMap &rSites)

Private Attributes

- string **uid**

  The id of the job creator.

- string **sid**

  The id of the site that rejected the job.

Detailed Description

The purpose of this class it to make an event where a rejected job can be returned to the user.

Constructor & Destructor Documentation

JobRejectedEvent::JobRejectedEvent (const string & *u*, const string & *s*, const int *t*) `[inline]`

Constructs the JobRejectedEvent

Parameters:

u The id of the user

s The id of the site

t The time of the event

Member Function Documentation

void JobRejectedEvent::action (rMap & *rUsers*, rMap & *rSites*) [virtual]

The method will extract the rejected job from the site and return it to the user.

Parameters:

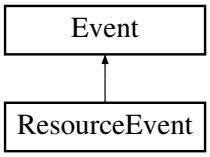
rUsers A reference to the map containing pointer to the users

rSites A reference to the map containing pointer to the sites

Implements **Event** (p. 141).

E.9 ResourceEvent Class Reference

Inheritance diagram for ResourceEvent::



Public Member Functions

- **ResourceEvent** (**change** ch)
- void **action** (rMap &rUsers, rMap &rSites)

Private Attributes

- **change c**

  Contains the change in behaviour.

Detailed Description

The purpose of the ResourceEvent is to provide the possibility of changing the behavioural values of a resource mid simulation.

Constructor & Destructor Documentation

ResourceEvent::ResourceEvent (change *ch*) `[inline]`

Constructs the ResourceEvent Object.

Parameters:

ch Contains the parameters for the change in behaviour

Member Function Documentation

void ResourceEvent::action (rMap & *rUsers*, rMap & *rSites*) `[virtual]`

The method will locate the resource and change the behavioural values.

Parameters:

rUsers A reference to the map containing pointers to the users

rSites A reference to the map containing pointers to the sites

Implements **Event** (p. 141).

E.10 Entity Class Reference

Public Member Functions

- **Entity** (string i, rType t, float v, float l, unsigned int ut)
- void **setValue** (float **value**, unsigned int time)
- string **getId** ()

  Returns the id of the entity.

- rType **getType** ()

  Returns the type of entity.

- float **getValue** ()

  Returns the (combined) trust value.

- unsigned int **updatedAt** ()

  Returns the last update time for the entity.

- void **setGroups** (list< int > l)

  Sets the entity's list of groups.

- list< int > **getGroups** ()

  Returns the entity's list of groups.

- int **matches** (list< int >)

  Find how many groups match the entity's groups.

- unsigned int **getRejections** ()

  Returns the number of rejections.

- void **setRejections** (unsigned int r)

  Sets the number of rejections.

- unsigned int **getRetryTime** ()

  Returns the earliest time the resource can get retried.

- void **setRetryTime** (unsigned int rt)

  Sets the earliest time the resource can get retried.

- unsigned int **getRetries** ()

  Returns the number of times the resource has been retried.

- void **setRetries** (unsigned int rt)

  Sets the number of times the resource has been retried.

- void **addFeedback** (float r, float s)
- **feedback** $*$ **combinedFeedback** ()
- list< **update** $*$ > **getHistory** ()

  Returns the combined trust history.

- void **setDtUpdate** (float **value**, unsigned int time)
- void **setRepUpdate** (float **value**, unsigned int time)
- list< **update** $*$ > **getDtHistory** ()

  Returns the direct trust history.

- list< **update** $*$ > **getRepHistory** ()

  Returns the reputation history.

- void **setforget** (float f)

  Sets the forgetting factor.

Private Attributes

- string **id**

  The entity's id (The resource id).

- float **value**

  Trust value for entity.

- unsigned int **updateTime**

  Last update time.

- rType **type**

  The resource type.

- list< int > **groups**

  The resource's known groups.

- unsigned int **rejections**

  Number of times the resource has been rejected.

- unsigned int **retryTime**

  The earliest time the resource can get retried.

- unsigned int **retries**

  The number of retries.

- deque< **feedback** * > **feedbackTuples**

  The sequence of feedback tuples(for the beta reputation system).

- list< **update** * > **history**

  The combined trust history.

- list< **update** * > **dtHistory**

  The direct trust history.

- list< **update** * > **repHistory**

  The reputation trust history.

- float **lambda**

  The forgetting factor.

Detailed Description

The purpose of the Entity class is to allow resources to have information on other resources. When another resource gets known, an Entity object will be created. This object will store the trust, rejections and group informations known by the resource. The object will also store the trust histories. A trust history contain the sequence of trust values with the associated time of calculation.

Constructor & Destructor Documentation

Entity::Entity (string *i*, rType *t*, float *v*, float *l*, unsigned int *ut*) `[inline]`

Constructs an Entity object. The object is used by a resource to store information on another resource.

Parameters:

i The id of the resource

t The type of the resource

v Initial trust value

l Initial forgetting factor

ut Time the resource is added

Member Function Documentation

void Entity::addFeedback (float *r*, float *s*)

Adds a feedback tuple to the sequence of feedbacks. The method is used by the beta reputation system.

Parameters:

r The r value of the feedback

s The s value of the feedback

feedback * Entity::combinedFeedback ()

Returns a combined feedback value for the beta reputation system. The method will iterate over the sequence of feedback tuples with the specified forgetting factor.

void Entity::setDtUpdate (float *value*, unsigned int *time*)

Adds an entry to the direct trust history.

Parameters:

value The direct trust value

time The time of calculation

void Entity::setRepUpdate (float *value*, unsigned int *time*)

Adds an entry to the reputation history.

Parameters:

value The reputation trust value

time The time of calculation

void Entity::setValue (float *value*, unsigned int *time*)

Updates the combined trust value.

Parameters:

value The new combined trust value

time The time that the update occured

E.11 EventHandler Class Reference

Public Member Functions

- **EventHandler** (const string &file)
- void **run** ()

  Runs the simulation. Performs all the events until the event queue is empty.

- void **addEvent** (**Event** *)

  Adds an event to the event queue.

- void **knowEverybody** ()

  Makes all resources known to each other.

- void **listGrid** ()

  List the grid (used for testing).

- void **setup** ()

  Creates the users and sites and sets up the simulation.

- void **summarize** ()

  Calls the output methods.

- void **testSum** ()

  Outputs results (Used for testing).

- int **timeSeqment** ()

  Returns the simulations time specified in the configuration file.

- double **mu** ()

  Returns the mean of the Poisson distribution specified in the configuration file.

- unsigned int **getSeed** ()

  Returns the random number seed.

- void **setSeed** (int)

  Sets the random number seed. The method will be used if the command line overwrites what is specified in the configuration file.

Public Attributes

- **RandomGenerator rg**

  The random number generator.

Private Member Functions

- void **output** (const string &id1, const string &id2, list< **update** * > history, const string &postfix, const string &path)
- void **organize** (const string &path)

Private Attributes

- priority_queue< **Event** *, vector< **Event** *, allocator< **Event** * > >, **eventComparison** > **eventQueue**

  The priority queue for events.

- unsigned int **simulationTime**

  The simulation time.

- rMap **rUsers**

  The map containing pointer to the users.

- rMap **rSites**

  The map containing pointer to the sites.

- **Configuration** * **conf**

  The main configuration object.

Detailed Description

The purpose of the EventHandler class is to control the simulation. An EventHandler object will load the configuration file. When instructed it will set up the simulation. A subsequent instruction will cause the queue of events to be executed. Afterward it can output the results of the simulation.

Constructor & Destructor Documentation

EventHandler::EventHandler (const string & *file*) `[inline]`

Constructs an EventHandler object

Parameters:

file The file name of the configuration file

Member Function Documentation

void EventHandler::organize (const string & *path*) `[private]`

The method will gather the various trust histories from the resources and send them to the output method

Parameters:

path The path to where the histories are to be written

void EventHandler::output (const string & *id1*, const string & *id2*, list< update * > *history*, const string & *postfix*, const string & *path*) `[private]`

The method to write a history of trust values to a specific location

Parameters:

id1 The id of the resource's trust values that are to be written

id2 The id of the resource that id1 has the trust experince with

history The history to write

postfix The ending of the filename

path The path where to write the file

E.12 Job Class Reference

The purpose of the Job is merely to function as a data value container.

Public Member Functions

- **Job** (const int c, const string &i)
- void **info** ()

  Method to view the data for the job (used for testing).

- string **getCreator** ()

  Returns the id of the job creator.

- int **getCpuTime** ()

  Returns the actual CPU time.

- int **getInformedCpuTime** ()

  Returns the informed CPU time.

- int **getGivenCpuTime** ()

  Returns the CPU time allocated by the site.

- void **start** (const int i)

  Sets the start time for the job.

- int **timeStarted** ()

  Returns the time where the job was started.

- void **setFinishTime** (const int i)

  Sets the time where the job will finish.

- int **timeFinished** ()

  Returns the time where the job will finish.

- void **setGivenCpuTime** (const int t)

  Sets the CPU time allocated by the site.

- void **setInformedCpuTime** (const int t)

  Sets the CPU time informed by the user.

- void **setPreferedSite** (const string &p)

Specifies the preferred site id for the job.

- void **setSite** (const string &s)

  Sets the site id specifying the site that processes the job.

- string **getPreferedSite** ()

  Returns the id of the preferred site.

- string **getSite** ()

  Returns the id the site that processed the job.

Private Member Functions

- **Job** & **operator**= (**Job** &)

  Overwrites assignment operator. The jobs must be unique.

- **Job** (**Job** &)

  Overwrites default copy constructor. The jobs must be unique.

Private Attributes

- int **cpuTime**

  The actual CPU time needed to complete the job.

- int **informedCpuTime**

  The CPU time the job creator informs the job will need.

- int **givenCpuTime**

  The CPU time the site allocates to the job.

- string **creator**

  The creator of the job.

- string **site**

  The site that process the job.

- string **preferedSite**

  The job creators prefered site.

- int **startTime**

  The start time of the job.

- int **finishTime**

  The finishing time of the job.

Constructor & Destructor Documentation

Job::Job (const int *c*, const string & *i*) [inline]

Constructs a job object.

Parameters:

c The actual CPU time needed to complete the job

i The id of the user creating the job

E.13 RandomGenerator Class Reference

This is a C++ wrapper class for the gnu scientific library.

Public Member Functions

- **RandomGenerator** ()
  Constructs a random number generator.
- **RandomGenerator** (int s)
  Creates a random number generator with seed s. (This method is only used once).
- ~**RandomGenerator** ()
  Deletes the random number generator.
- int **irand** (int n)
  Returns a random number below the value n.
- int **irand** (int i, int j)
  Returns a random number between i and j.
- int **ipoisson** (double mu)
  Returns a random number from a Poisson distribution with mean mu.
- double **laplace** (double a)
  Returns a randum number from a Laplace distribution with mean a.

Static Public Attributes

- static gsl_rng * **r** = gsl_rng_alloc (gsl_rng_taus)
  The shared random number generator.

E.14 Configuration Class Reference

Public Member Functions

- **Configuration** (const string &file)
- int **getMin** (const string &id)
- int **getMax** (const string &id)
- list< int > **getGroups** (const string &id)
- float **getThreshold** (const string &id)
- int **users** ()

  Returns the number users in the simulation.

- int **sites** ()

  Returns the number sites in the simulation.

- int **timeSeqment** ()

  Returns the time length of the simulation.

- double **mu** ()

  Returns the mean of the Poisson distribution.

- string **trustEval** ()

  Returns the method used for combining trust.

- string **repEval** ()

  Returns the type of the reputation system.

- string **path** ()

  Returns the output path for simulation results.

- unsigned int **getSeed** ()

  Returns the random number seed.

- void **setSeed** (const int s)

  Sets the random number seed should it be specified at the commandline.

- float **getDweight** ()

  Returns the direct weight.

- float **getSmoothing** ()

  Returns the smoothing factor.

- float **getDratio** ()

  Returns the discrete ratio between direct and reputation trust.

- float **getStrangerV** ()

  Returns the default trust value to assign to strangers.

- float **getForgetV** ()

  Returns the forgetting factor.

- list< **change** > **getChanges** ()

  Returns the list of changes that will occur in the simulation.

- list< **addition** > **getAdditions** ()

  Returns the list of resources will be added to the simulation.

Private Member Functions

- void **setConfig** (ifstream &inFile, string &line)
- void **setDefault** (ifstream &inFile, string &line)
- void **setGroups** (ifstream &inFile, string &line)
- void **setAdd** (ifstream &inFile, string &line)
- void **setResource** (ifstream &inFile, string &line, string &id)
- void **setIj** (ifstream &inFile, string &line, string &id)
- void **setChange** (ifstream &inFile, string &line, string &id)
- void **setThreshold** (ifstream &inFile, string &line, string &id)
- void **loadFile** (const string &file)

Private Attributes

- map< string, string > **config**

  Map used to store the main configuration values.

- **property defaultProperty**

  Contains the default resource parameters.

- map< string, **property** > **resource**

  Contains parameters for specific resources.

- list< **change** > **changes**

  List of changes that will occur during the simulation.

- list< **addition** > **additions**

  List of resources that will be added to the simulation.

Detailed Description

The purpose of this class is to read the .conf file and set up the simulation. A set of methods will scan and parse the configuration file and store each value. The objects can query the configuration object for the specifics of the simulation. This class is quite simple. It expects that the simulation user is capable of writing correct .conf files. Should a mistake appear in the .conf file an exception will be thrown.

Constructor & Destructor Documentation

Configuration::Configuration (const string & *file*) `[inline]`

Constructs and automatically loads the configuration file specified.

Parameters:

file The configuration filename.

Member Function Documentation

list< int > Configuration::getGroups (const string & *id*)

Returns the list of groups that a resource is a member of.

Parameters:

id The resource id

Returns:

The list of groups

int Configuration::getMax (const string & *id*)

Returns the max value for a specified resource

Parameters:

id The resource id

Returns:
The max value for the resource

int Configuration::getMin (const string & *id*)

Returns the min value for a specified resource

Parameters:
id The resource id

Returns:
The min value for the resource

float Configuration::getThreshold (const string & *id*)

Returns the trust threshold for a resource

Parameters:
id The resource id

Returns:
The trust threshold

void Configuration::loadFile (const string & *file*) [private]

Main file loader. The method will open the file and invoke the various methods used for parsing

Parameters:
file The name of the configuration file

void Configuration::setAdd (ifstream & *inFile*, string & *line*) [private]

Sets [Add] resource part of configuration. The method will expect to find a time, resource id, behavioural values, group memberships and a trust threshold value. If any of the parameters are missing an exception is thrown.

Parameters:
inFile Used to read from the configuration file

line Used to store the read line.

void Configuration::setChange (ifstream & *inFile*, string & *line*, string & *id*) [private]

Sets the change for a resource. The method will expect to find a time followed by i(for min) and j(for max) values. If any of the parameters are missing an exception is thrown.

Parameters:

inFile Used to read from the configuration file

line Used to store the read line.

id The resource id

void Configuration::setConfig (ifstream & *inFile*, string & *line*) [private]

Sets the [Config] part of configuration. The method will parse the number of users, sites, simulation time, mean of Poisson distribution, trust combination and reputation system, direct weight, smoothing factor, trust value to assign to strangers, ratio between direct and reputation trust (for the discrete method), forgetting factor (for the beta reputation system), random number seed and the path where the simulation results should be saved. If any of the parameters are missing or not appear as the expected order an exception will be thrown. This is also the case even if the parameter is not needed.

Parameters:

inFile Used to read from the configuration file

line Used to store the read line.

void Configuration::setDefault (ifstream & *inFile*, string & *line*) [private]

Sets the [Default] part of configuration. The method will read the default behavioural values of a resource (i for min and j for max) and the default trust threshold. The property defaultProperty will be created. If any of the expected parameters are missing or appear in unexpected order an exception will be thrown.

Parameters:

inFile Used to read from the configuration file

line Used to store the read line.

void Configuration::setGroups (ifstream & *inFile*, string & *line*) [private]

Sets the [Groups] part of configuration. The method will expect to find a resource id followed by a list of integers. A property struct is created and inserted into the map called resource.

Parameters:

inFile Used to read from the configuration file

line Used to store the read line.

void Configuration::setIj (ifstream & *inFile*, string & *line*, string & *id*) `[private]`

Sets the behavioural values i(for min) and j(for max) for a resource

Parameters:

inFile Used to read from the configuration file

line Used to store the read line.

id The resource id

void Configuration::setResource (ifstream & *inFile*, string & *line*, string & *id*) `[private]`

Sets (resource id) part. The method will direct the various parsing jobs to other methods

Parameters:

inFile Used to read from the configuration file

line Used to store the read line.

id The resource id

void Configuration::setThreshold (ifstream & *inFile*, string & *line*, string & *id*) `[private]`

Sets the trust threshold for a resource. Will expect to find a threshold= followed by a value between -1 and 1. An error will result in the throwing of an exception.

Parameters:

inFile Used to read from the configuration file

line Used to store the read line.

id The resource id

E.15 feedback Struct Reference

Data container that contain feedback to the beta reputation system.

E.16 update Struct Reference

Data container that contains the time and value of a trust update.

E.17 addition Struct Reference

Data structure that contains the id of a resource that is added to the simulation with the i (min), j (max), groups, trust threshold values at a specified point in time.

E.18 property Struct Reference

Data structure that contains the i (min), j(max), groups and trust threshold values for a resource.

E.19 change Struct Reference

Data structure that contains the id of a resource whose behavioural values i (min), j(max) change at a specified point in time.

E.20 eventComparison Struct Reference

Structure used to order events according to event time.

Appendix F

Fuzzy logic membership functions and rule base

```
FUNCTION_BLOCK

VAR_INPUT
        Direct_Trust            REAL; (* RANGE(0 .. 100) *)
        Reputation              REAL; (* RANGE(0 .. 100) *)
END_VAR

VAR_OUTPUT
        Trust                   REAL; (* RANGE(0 .. 100) *)
END_VAR

FUZZIFY Direct_Trust
        TERM very_low := (0,0)(0,1)(15,1)(25,0) ;
        TERM low := (15,0)(25,1)(35,1)(45,0) ;
        TERM medium := (35,0)(45,1)(55,1)(65,0);
        TERM high := (55,0)(65,1)(75,1)(85,0);
        TERM very_high := (75,0)(85,1)(100,1)(100,0);
END_FUZZIFY

FUZZIFY Reputation
        TERM very_low := (0,0)(0,1)(15,1)(25,0) ;
        TERM low := (15,0)(25,1)(35,1)(45,0) ;
        TERM medium := (35,0)(45,1)(55,1)(65,0);
        TERM high := (55,0)(65,1)(75,1)(85,0);
        TERM very_high := (75,0)(85,1)(100,1)(100,0);
END_FUZZIFY
```

```
FUZZIFY Trust
        TERM very_low := (0,0)(0,1)(15,1)(25,0) ;
        TERM low := (15,0)(25,1)(35,1)(45,0) ;
        TERM medium := (35,0)(45,1)(55,1)(65,0);
        TERM high := (55,0)(65,1)(75,1)(85,0);
        TERM very_high := (75,0)(85,1)(100,1)(100,0);
END_FUZZIFY

DEFUZZIFY Trust
        METHOD: COG;
END_DEFUZZIFY

RULEBLOCK first
        AND:MIN;
        ACCU:MAX;
        RULE 0: IF (Direct_Trust IS very_low) AND (Reputation IS very_low)
                THEN (Trust IS very_low);
        RULE 1: IF (Direct_Trust IS very_low) AND (Reputation IS low)
                THEN (Trust IS very_low);
        RULE 2: IF (Direct_Trust IS very_low) AND (Reputation IS medium)
                THEN (Trust IS low);
        RULE 3: IF (Direct_Trust IS very_low) AND (Reputation IS high)
                THEN (Trust IS medium);
        RULE 4: IF (Direct_Trust IS very_low) AND (Reputation IS very_high)
                THEN (Trust IS medium);
        RULE 5: IF (Direct_Trust IS low) AND (Reputation IS very_low)
                THEN (Trust IS very_low);
        RULE 6: IF (Direct_Trust IS low) AND (Reputation IS low)
                THEN (Trust IS low);
        RULE 7: IF (Direct_Trust IS low) AND (Reputation IS medium)
                THEN (Trust IS low);
        RULE 8: IF (Direct_Trust IS low) AND (Reputation IS high)
                THEN (Trust IS medium);
        RULE 9: IF (Direct_Trust IS low) AND (Reputation IS very_high)
                 THEN (Trust IS medium);
        RULE 10: IF (Direct_Trust IS medium) AND (Reputation IS very_low)
                 THEN (Trust IS low);
        RULE 11: IF (Direct_Trust IS medium) AND (Reputation IS low)
                 THEN (Trust IS medium);
        RULE 12: IF (Direct_Trust IS medium) AND (Reputation IS medium)
                 THEN (Trust IS medium);
        RULE 13: IF (Direct_Trust IS medium) AND (Reputation IS high)
                 THEN (Trust IS medium);
        RULE 14: IF (Direct_Trust IS medium) AND (Reputation IS very_high)
```

```
                THEN (Trust IS high);
        RULE 15: IF (Direct_Trust IS high) AND (Reputation IS very_low)
                THEN (Trust IS medium);
        RULE 16: IF (Direct_Trust IS high) AND (Reputation IS low)
                THEN (Trust IS medium);
        RULE 17: IF (Direct_Trust IS high) AND (Reputation IS medium)
                THEN (Trust IS high);
        RULE 18: IF (Direct_Trust IS high) AND (Reputation IS high)
                THEN (Trust IS high);
        RULE 19: IF (Direct_Trust IS high) AND (Reputation IS very_high)
                THEN (Trust IS very_high);
        RULE 20: IF (Direct_Trust IS very_high) AND (Reputation IS very_low)
                THEN (Trust IS medium);
        RULE 21: IF (Direct_Trust IS very_high) AND (Reputation IS low)
                THEN (Trust IS high);
        RULE 22: IF (Direct_Trust IS very_high) AND (Reputation IS medium)
                THEN (Trust IS high);
        RULE 23: IF (Direct_Trust IS very_high) AND (Reputation IS high)
                THEN (Trust IS very_high);
        RULE 24: IF (Direct_Trust IS very_high) AND (Reputation IS very_high)
                THEN (Trust IS very_high);
END_RULEBLOCK

END_FUNCTION_BLOCK
```

Appendix G

Configuration files

G.1 Experiment 1

```
[Config]
users=4
sites=4
time=10000
mu=25.0
trust_eval=discrete    # or fuzzy
rep_eval=discrete
direct_weight=0.6
smoothing=1.0
stranger=0.0
discrete_ratio=0.5     # or 0.8
forgetting=0.4
seed=100
path=/tmp/experiments/

[Default]
i=1
j=1
threshold=-1.0

# In experiment 1.2
# (s0)
# i=20
# j=20
```

G.2 Experiment 2

```
[Config]
users=4
sites=4
time=10000           # or 100000
mu=25.0
trust_eval=discrete
rep_eval=discrete    # or beta
direct_weight=0.4
smoothing=1.0
stranger=0.0
discrete_ratio=0.5
forgetting=0.4
seed=100
path=/tmp/experiments/

[Default]
i=1
j=1
threshold=-1.0

# In experiment 2.2
# (s0)
# i=20
# j=20
```

G.3 Experiment 3

```
[Config]
users=4
sites=4
time=100000
mu=25.0
trust_eval=discrete    # or fuzzy
rep_eval=discret       # or beta
direct_weight=0.6
smoothing=1.0
stranger=0.0
discrete_ratio=0.5
forgetting=0.4
seed=time
```

```
path=/tmp/experiments/

[Default]
i=1
j=1
threshold=-1.0

(s0)
i=1                    # or i=20
j=1                    # or j=20
time=15000 i=20 j=20   # or time=15000 i=1 j=1
```

G.4 Experiment 4

```
[Config]
users=4
sites=4
time=100000
mu=25.0
trust_eval=discrete  # or fuzzy
rep_eval=discrete    # or beta
direct_weight=0.6
smoothing=1.0
stranger=0.0
discrete_ratio=0.5
forgetting=0.4
seed=time
path=/tmp/experiments/

[Default]
i=1
j=1
threshold=-1.0

(s0)
i=20
j=20
time=15000 i=1 j=1
time=30000 i=20 j=20
time=45000 i=1 j=1
```

G.5 Experiment 5

```
[Config]
users=4
sites=4
time=100000
mu=25.0
trust_eval=discrete       # or fuzzy
rep_eval=discrete         # or beta
direct_weight=0.6
smoothing=1.0
stranger=0.0
discrete_ratio=0.5
forgetting=0.4
seed=time
path=/tmp/experiments/

[Default]
i=1
j=1
threshold=-1.0

[Add]
time=15000 s4 i=1 j=1 g= threshold=-1.0

# Or
# [Add]
# time=15000 s4 i=20 j=20 g= threshold=-1.0
```

G.6 Experiment 6

```
[Config]
users=4
sites=4
time=100000
mu=25.0
trust_eval=discrete    # or fuzzy
rep_eval=discrete      # or beta
direct_weight=0.6
smoothing=1.0
stranger=0.0
discrete_ratio=0.5
```

```
forgetting=0.4
seed=100
path=/tmp/experiments/

[Default]
i=1
j=1
threshold=-1.0

(u0)
i=1
j=1
threshold=0.0

(s0)
i=20
j=20
```

G.7 Experiment 7

```
[Config]
users=4
sites=4
time=100000
mu=25.0
trust_eval=discrete  # or fuzzy
rep_eval=discrete    # or beta
direct_weight=0.6
smoothing=1.0
stranger=0.0
discrete_ratio=0.5
forgetting=0.4
seed=100
path=/tmp/experiments/

[Default]
i=20
j=20
threshold=-1.0

[Groups]
u0= 1 2 3 4 5
```

```
u1= 1 2 3 4 5

(u0)
i=1
j=1

(u1)
i=1
j=1

(s0)
i=1
j=1
```

G.8 Experiment 8

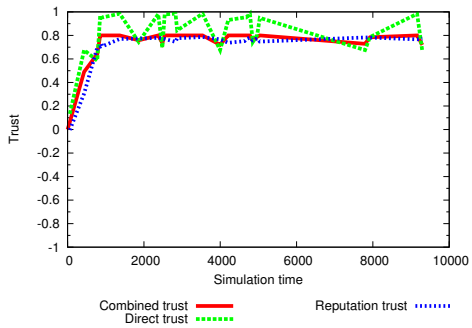
```
[Config]
users=4
sites=4
time=100000
mu=25.0
trust_eval=discrete  # or fuzzy
rep_eval=discrete    # or beta
direct_weight=0.6
smoothing=1.0
stranger=0.0
discrete_ratio=0.5
forgetting=0.4
seed=100
path=/tmp/experiments/

[Default]
i=5
j=5
threshold=-1.0
```

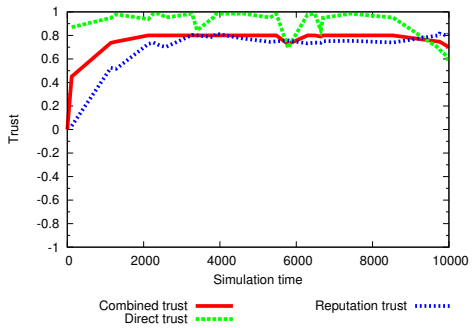
Appendix H

Plots

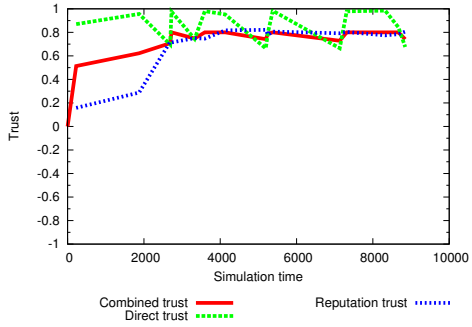
H.1 Example 1



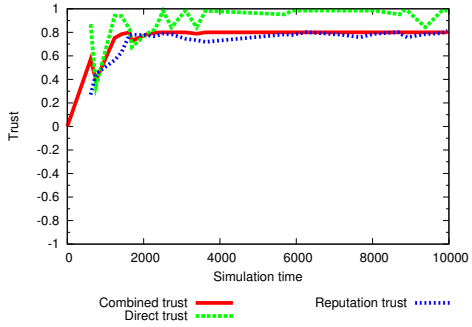
(a) *u0*'s trust in *s0*



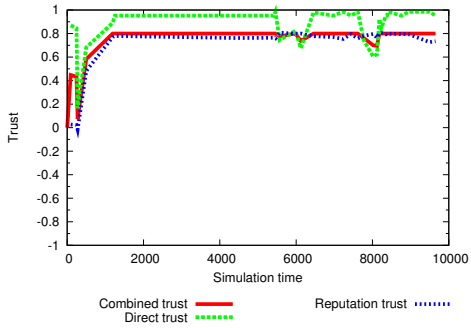
(b) *u0*'s trust in *s1*



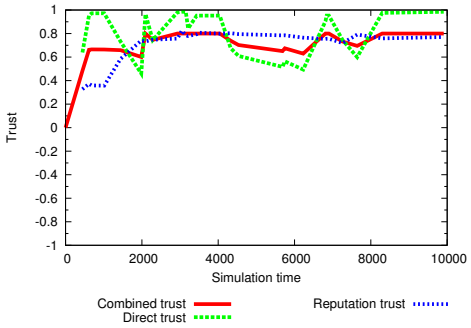
(c) *u0*'s trust in *s2*



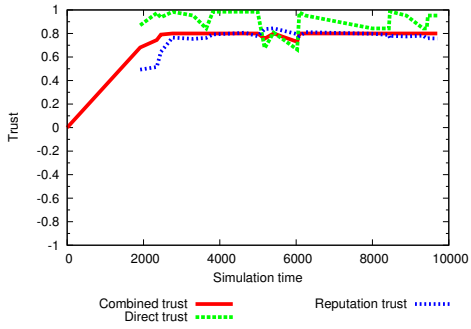
(d) *u0*'s trust in *s3*



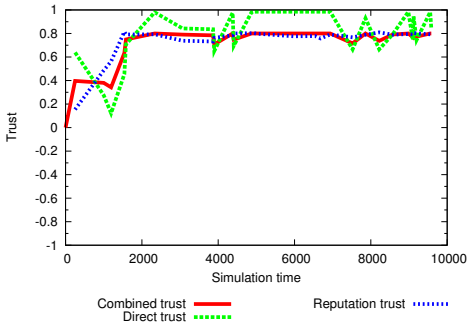
(a) *u1*'s trust in *s0*



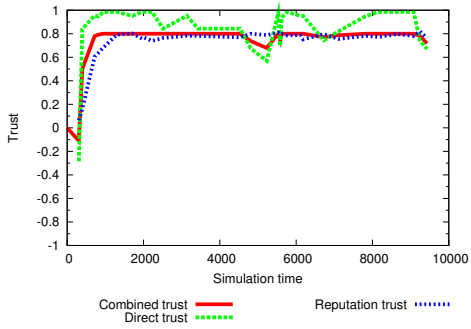
(b) *u1*'s trust in *s1*



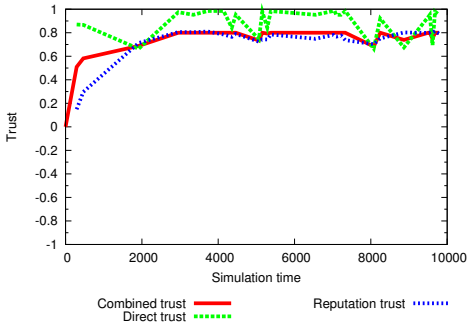
(c) *u1*'s trust in *s2*



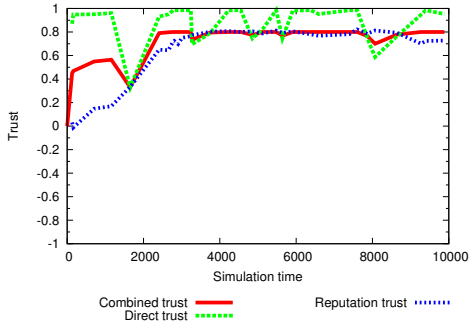
(d) *u1*'s trust in *s3*



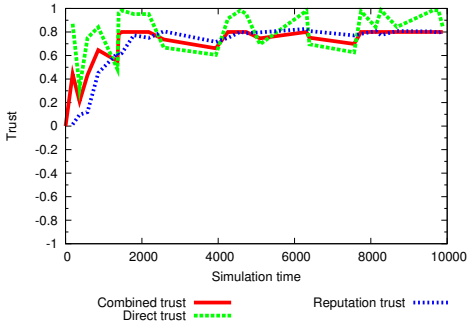
(a) *u2*'s trust in *s0*



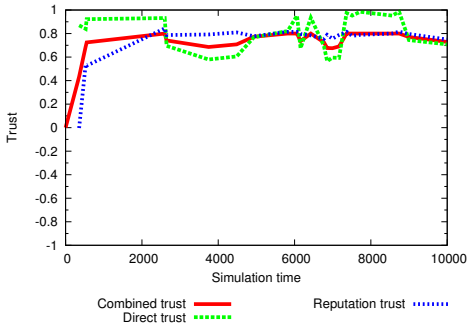
(b) *u2*'s trust in *s1*



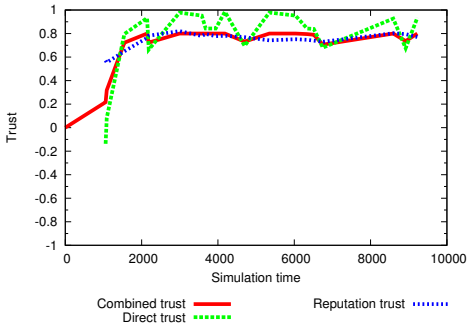
(c) *u2*'s trust in *s2*



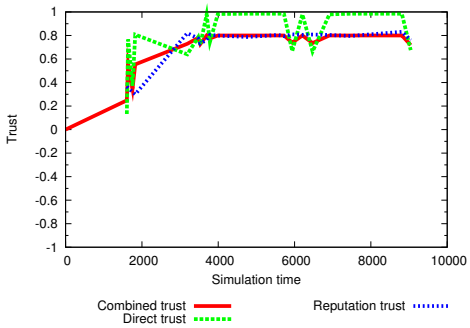
(d) *u2*'s trust in *s3*



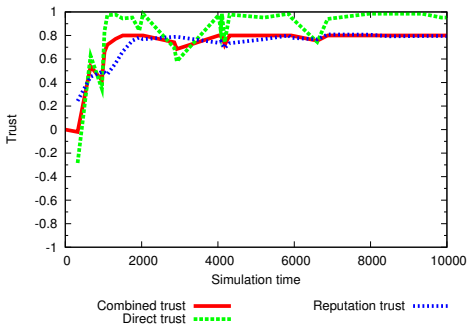
(a) *u3*'s trust in *s0*



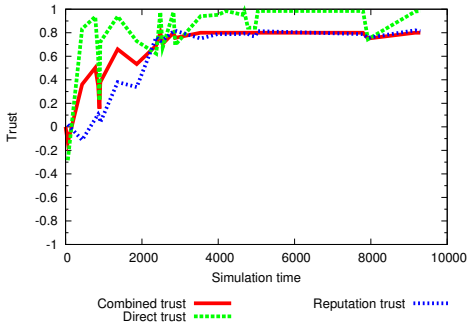
(b) *u3*'s trust in *s1*



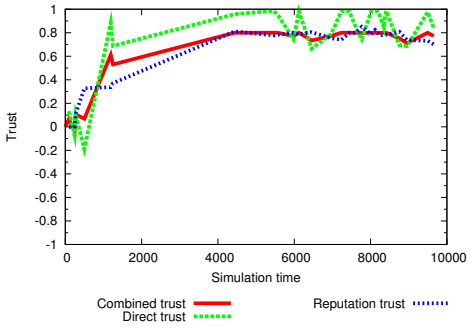
(c) *u3*'s trust in *s2*



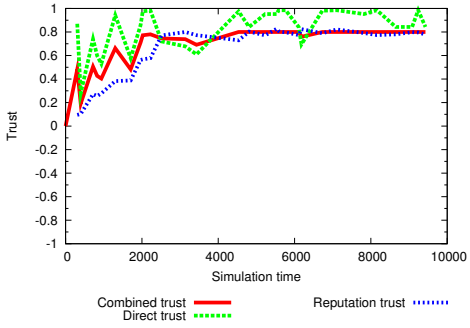
(d) *u3*'s trust in *s3*



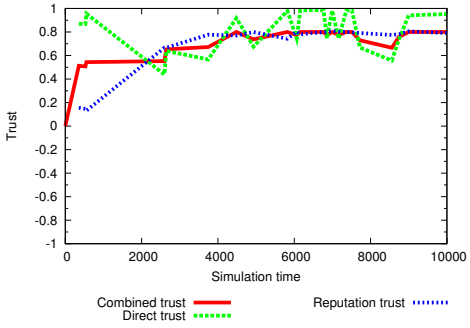
(a) *s0*'s trust in *u0*



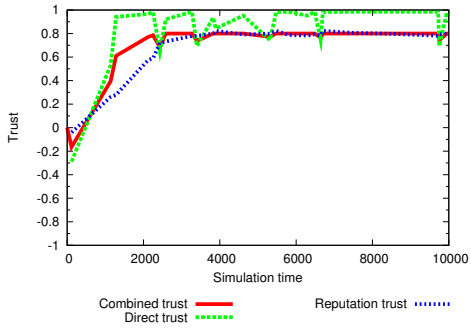
(b) *s0*'s trust in *u1*



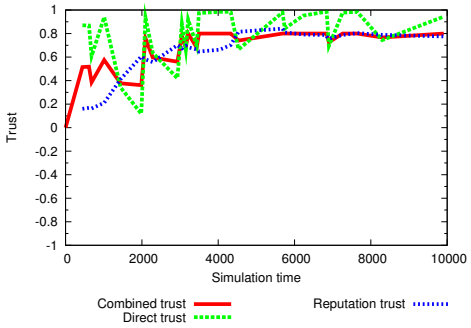
(c) *s0*'s trust in *u2*



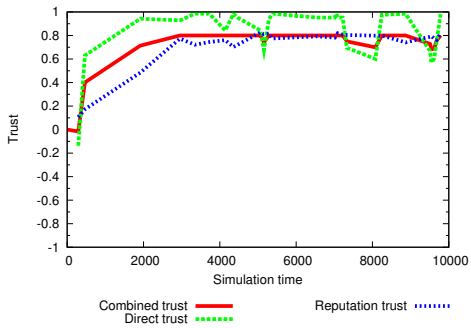
(d) *s0*'s trust in *u3*



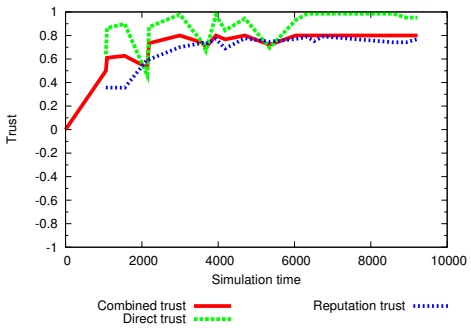
(a) *s1*'s trust in *u0*



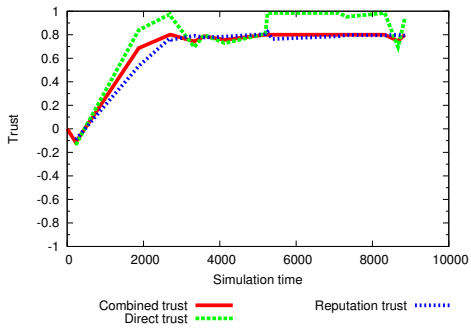
(b) *s1*'s trust in *u1*



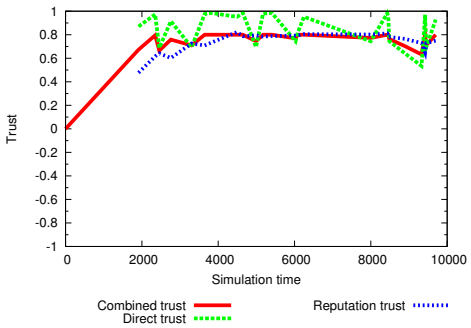
(c) *s1*'s trust in *u2*



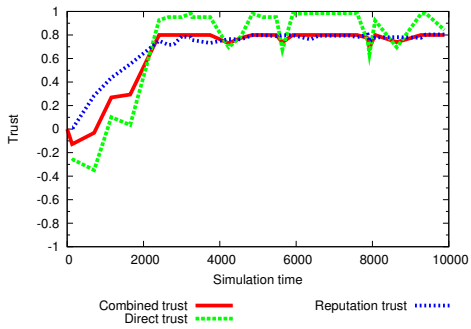
(d) *s1*'s trust in *u3*



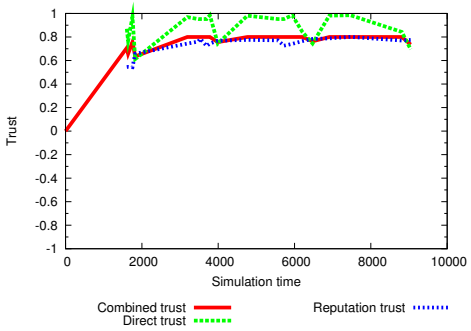
(a) *s2*'s trust in *u0*



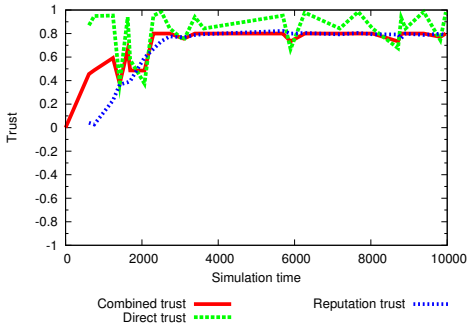
(b) *s2*'s trust in *u1*



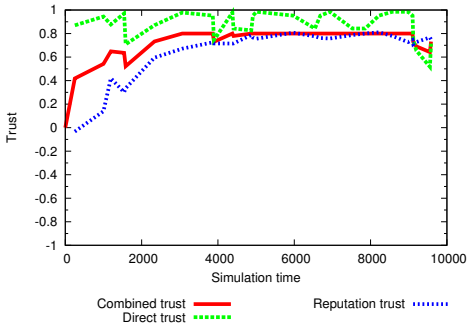
(c) *s2*'s trust in *u2*



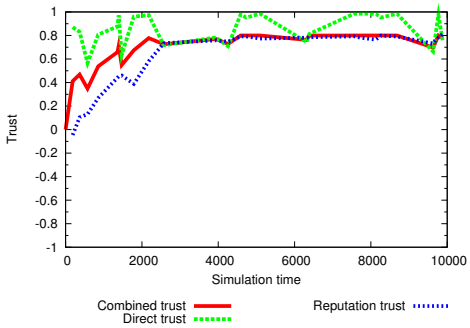
(d) *s2*'s trust in *u3*



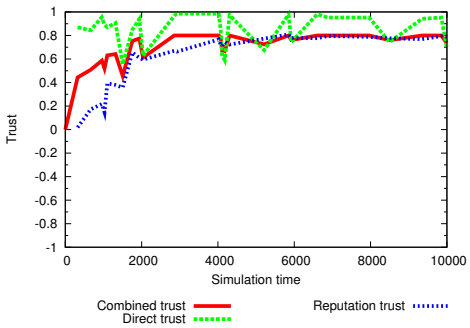
(a) *s3*'s trust in *u0*



(b) *s3*'s trust in *u1*

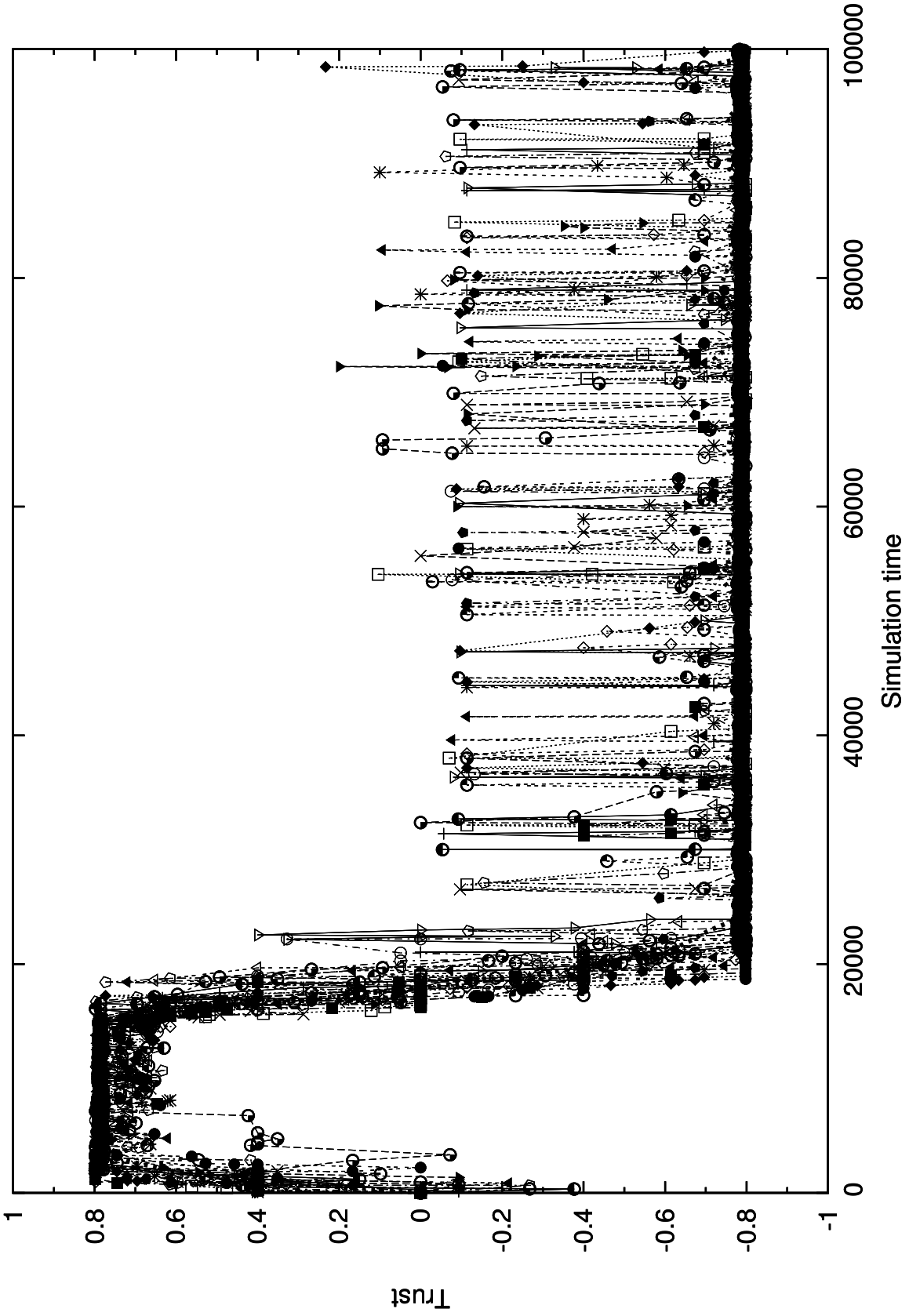


(c) *s3*'s trust in *u2*



(d) *s3*'s trust in *u3*

H.2 Example 2


Trust
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
Simulation time
0
20000
40000
60000
80000
100000