GRID EIGEN TRUST

A FRAMEWORK FOR COMPUTING REPUTATION IN GRIDS

BY

BEULAH KURIAN ALUNKAL

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science
in the Graduate College of the
Illinois Institute of Technology

Approved _____
Adviser

Chicago, Illinois
December 2003

ACKNOWLEDGMENT

I would like to thank my project supervisor Dr.Gregor von Laszewski from Argonne National Laboratories for his supervision, mentoring, patience and guidance all through my work. His innovative and critical comments have boosted me to think more profoundly and critically. The idea of investigating the use of Reputation has been initiated by him. Part of the work summarized in this thesis has been presented by him in an oral presentation at the Workshop on Adaptive Grid Middleware, St. Louis, Sep 28, 2003. The corresponding preprint of the partial contents of that presentation is available as Argonne preprint ANL/MCS-P1109-090 and was done in collaboration with Ivana Veljkovic and Kaizar Amin. I express my deep gratitude for all their help and support.

This work was in part supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38. DARPA, DOE, and NSF support Globus Project research and development. The Java CoG Kit Project is supported by DOE SciDAC and NSF Alliance.

I would like to thank my academic supervisor, Dr. Xian-He Sun and committee member, Dr. Cynthia Hood for their time, effort and constructive feedback on my work. I attribute my special thanks to Professor James Dabbert, for his valuable suggestions regarding the language and grammar. I would like to attribute my special thanks to Ivana Veljkovic for her support in designing the mathematical model used in the framework.

Furthermore, I would like to specially thank my parents, Kurian Alunkal and Leela Kurian, siblings Hephsiba Alunkal and Ebenezer Alunkal for their support and their continual reassurances that I would be able to finish my thesis successfully.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

Abbreviation	Term
API	Application Programming Interface
DNS	Domain Name Service
GET	Grid Eigen Trust
GSDL	Grid Services Description Language
GT2	Globus Toolkit 2.0
GT3	Globus Toolkit 3.0
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
MDS	Monitoring and Discovering Service
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TPL	Trust Policy Language
UDP	User Datagram Protocol
VO	Virtual Organization
WSDL	Web Services Description Language

ABSTRACT

The Grid approach provides the ability to access, utilize, and control a variety of heterogeneous resources distributed across multiple domains and institutions by forming virtual organizations. Choosing appropriate resources in such a distributed and heterogeneous environment brings up several challenges. Several attempts have been made to select resources based on user requirements and resources availability, but there has been no attempts made in Grids to provide selection of resources using the notion of trust and reputation. We introduce Grid Eigen Trust, a new framework that is based on the concept of dynamic trust and reputation adaptation based on community experiences to compute reputation of resources, services and users. It has harnessed the power of web services technologies to allow communication and flexibility in the framework. Our experimental evaluations have shown the framework to be less computation-intensive, interoperable, cross-platform, and extensible.

CHAPTER 1

INTRODUCTION

We are moving from the era of centralized supercomputing to high-performance computing, distributed computing and peer-to-peer networking systems. Grid approach [61] is the term that describes all of these various mechanisms that allow massive number of computers and immense computing power. Grid approach is evolving as an important trend in high-performance computing. It exploits the advantages of the power of distributed systems.

Since the conceiving of the World Wide Web as a tool for sharing scientific information in the year 1990 by Tim Bernes-Lee, the applications of the Web have outgrown the initial problem of sharing scientific data between scientists. But meanwhile the scientists at Cern have also outgrown the developments of the Web. Cern's major project, The Large Hadron Collider (LHC) [15][1], is set to outweigh the boundaries that the Web is currently providing. The LHC experiment will accelerate particles close to the speed of light and crash them into each other using a Large Electron-Positron (LEP) Collider. LEP data are so accurate that they are sensitive to phenomena that occur at energies beyond those of the machine itself. This gives us a 'preview' of exciting discoveries that may be made at higher energies and allow us to calculate the parameters of a machine that can make these discoveries. Even though we do have sophisticated systems that will remove uninteresting data from the huge set of collected data, the data suitable for analysis will still be in the order of petabytes per year [48]. The computer power required to solve this problem does not seem to be feasible at the Cern site using only Cern's resources. Grid approach seems to be the appropriate solution to such problems.

The name Grid suggests an analogy with the electrical power grids. For in-

[1] numbers in brackets correspond to entries in the Bibliography.

stance, if we wish to use electricity, we do not worry about where it comes from or how it is fed into the wall socket. As long as we are able to plug into the electricity socket, we are able to use the power. Similarly, in order to use the huge computing resources, the user does not need to know where the application ran, or which storage was used for running his applications. The vision for the Grid approach is that it should be consistent, inexpensive, pervasive and dependable. As the Web browsers and the Web services provide a standard interface to distributed data, the idea of the Grid is to provide a consistent framework for distributed computing resources in the same way.

The progress in Grid technology is promising. On the software technology side, Globus Alliance [26] is on its way to develop frameworks for consistent, pervasive, inexpensive and dependable services. Hardware resources can be exploited from thousands of supercomputing centers, educational institutions and other organizations across the globe. As for network technology, the broadband access to the internet is becoming ubiquitous. In order to exploit these resources for high-end scientific solutions, applications that need to span over different organizations and institutions need to be engineered. The major requirement for design of such collaborative Grid applications is a robust framework that supports security and trust. Grid Security Infrastructure [31] handles issues with authentication and authorization within security. Grid Eigen Trust seeks to lay a foundation for a robust trust framework for Grids.

1.1 Trust

Trust is a vital component in every service transaction. For instance, consumers must trust that the producers will provide the service they advertise and producer must trust the customer is able to pay for the services used. Otherwise transactions cannot take place. Consider an example of a research scientist at Argonne

National laboratory requiring to access services of a big cluster located a research lab in California. There are two ways this cluster can be used. The cluster could be hosting a service that does sophisticated scientific calculations and the scientist wishes to use this service. On the other hand, the scientist might want to transfer an executable application to run it on the cluster. In the first case, an authentication mechanism which ensures the identity of the user is sufficient to allow access to the service. But in the latter case, even though the identity assures that the scientist is authorized to use the service, it does not guarantee that the application does not perform illegal operation on the cluster. In this case, information regarding the user's prior behavior can avoid suspicion and give access to the cluster. This illustrates that notion of trust and reputation can greatly enhance development of highly collaborative Grid applications.

Most of our daily work takes trust into consideration even though we are not aware of it. For instance, we board buses or trains trusting the driver will not incur accidents, we buy groceries from stores trusting they are not poisoned and so on. The notion of trust has several meanings associated with it. It could mean authentication information to any conceivable system, or level of trust put on the system. Azzedin et al. [8] has classified trust into two categories: identity trust and behavior trust. Identity trust is concerned with verifying authenticity of an entity, whereas behavior trust deals with trustworthiness. To understand this concept, let us consider an example of a Grid user running an executable code on remote supercomputer using a digitally signed certificate. The certificate only conveys his identity trust that is concerned with authentication, verification and the authorization that entity can access. It is handled in Grid using the Grid Security Infrastructure [31]. But it does not convey the trust in the executable code. It could have been written by a competent programmer. If, on the other hand, the user is previously known, then his code may be trusted. Or if it is a new user, then as long as the creator of the code

does not employ elaborate methods to evade his accountability, the user can be taken granted for risks that comes to the remote supercomputer by his usage.

Until now the trust management in Grid has been studied from the perspective of establishing security policies and credentials and determining whether credentials match policies. We will focus on the second type of trust the authors have specifically referred to as "behavior trust" in the rest of the chapters. Words such as "trust" or "reputation" refer to behavior trust.

A recent study [10] has demonstrated that appropriate feedback mechanisms can induce calculus-based credibility trust without repeated interactions between two transacting parties. Feedback mechanism used here is to evaluate the behavior trust. Researchers have used data from both an on line experiment and an on line auction market to verify it. They have confirmed that buyers develop trust in sellers' credibility partly as a result of feedback mechanisms and that trust has a substantial effect on the transaction by generating price premiums. There is also an on line Reputation Research Network system [54], where people decide who to trust, which encourages people to be more trustworthy and discourages those who are not trustworthy from participating. The concept of trust has been addressed within many disciplines including philosophy, psychology and sociology and environments such as Information systems, E-Commerce, On line systems, Peer-to-Peer computing and Ubiquitous Computing. Currently, it is right time and quite appropriate to apply concept of trust to Grids.

Moving from centralized system to distributed computing, distributed computing to Internet based applications, means that transaction span a range of organizations and domains. Coming to Grids, it not only involves information exchange and sharing but also allows resources and services spanning a large number of organizations and domains to share and co-ordinate. It is quite obvious that not all organizations or domains can be trusted to the same extent. Even though the security

infrastructure requirements between them may be similar, they may support different types of security policy based on different trust relationship required by them. This makes the trust problem complex in Grids.

1.2 Grid Eigen Trust Framework

As Laszewski points out in [5], "Community based adaptive metrics like trust and reputation serve as building blocks to support quality of service requirements in Grids. It is important to recognize that the self-evaluation of a service must be an integral part of the Grid architecture in order to increase reliability and predictability. Consider the case in which a service claims it will provide a particular level of quality and engages in a service level agreement with another service. Assume, this service fails to deliver the promised agreement, and the request is not fulfilled. Choosing a more reliable service can avoid this problem." In order to select a more reliable service, we need information regarding the history of the prior service usage. We have developed a new framework called Grid Eigen Trust to address this issue and provide trust and reputation for services based on their past history. Our algorithm uses EigenVector mathematical model to manage reputation in Grid environment. That is why we have named it as "Grid Eigen Trust".

To emphasize the important role that Grid Eigen Trust plays, Laszewski points out an interesting example in [5]. "Consider the example to design a Grid environment that agglomerates expensive and specialized resources including high-performance servers, storage databases, advanced scientific instruments, and sophisticated services to visualize macromolecules [65] or nano-material [62] structures. In these usage scenarios, we require the availability of reliable ad hoc Grid services to fulfill the necessary quality of service requirements posed by the secured real-time use. Furthermore, the sporadic and time limited nature of the services and resources used may result in a lack of historical data posing severe limitations on prediction services. Grid Eigen

Trust seems the appropriate framework to handle such situations."

1.3 Overview of Work

Our work includes aspects such as how can reputation be maintained among a user, a group, a community, the Grid. It supports reputation for users, resources as well as services belonging to an organization. Chapter 2 gives an overview of existing trust and reputation based systems that are prevalent in Grid approach, P2P Networks, Internet applications, ubiquitous collaborations, pervasive computing environments, mobile computing and in general on line systems and search engines. It provides comparison of various technologies and trust models. Chapter 3 describes the background, the goals of the system, and provides some brief analysis of the system. Chapter 4 discusses the system architecture, the various components of the system and the required functions that each component must perform. Chapter 5 describes the algorithm used by the framework. Chapter 6 discusses the prototype implementation of the system as a Grid service. Chapter 7 provides experimental evaluations of our system using simulations. Chapter 8 provides an example of a meteorological application using our reputation service and other Grid services along with Information Services. Chapter 9 concludes the document and provides information of how Grid Eigen Trust System may be worked into a future system for Grid computing research.

CHAPTER 2
LITERATURE REVIEW ON TRUST-BASED SYSTEMS

There are a variety of systems addressing the problem of trust in various disciplines. In this Chapter we give a review of work done into five main categories. We first discuss work already done in trust in the area of Grid computing (Section 2.1). We then discuss few of the models of trust developed for peer-peer systems (Section 2.2), internet applications (Section 2.3), ubiquitous computing (Section 2.4) and mobile computing (Section 2.6).

2.1 Grid Computing

The traditional Grid Security Infrastructure (GSI) [31] from the Globus Alliance, uses the X.509 certificates as its authentication mechanism for security. It has provided necessary mechanisms needed for authentication, but does not handle all the security management issues. For instance, there is limited support for policy-based management, dynamic authorization management and trust management in situations where the collaborating services do not have any prior knowledge of each other or their certifying authorities. Although GSI uses the public key infrastructure to reliably establish the identity of other collaborators, this identity does not convey information about the likely behavior of the principal. Identity alone therefore cannot be used for access control decisions. For example, a digitally signed code does not convey if it was written by competent programmers or if the certificate issuer is an industrial spy. Because of the sensitivity and vitality of data or information, the entities prefer to use the services only within closed box resources.

2.1.1 ConCert Software. Chang et al. [16] present ConCert software framework in which the notion of certified code uphold safety, security, and privacy policies.

The notion of certifying code attributes trust to the code. It does so by examining intrinsic properties of native code that is to be run on a remote machine in the Grid. The examining of code determines if it does try to attempt to perform illegal operation either intentionally or non-intentionally. Once the testing of the code is done, it is assumed to be trustworthy. The authors claim that there is no need of any additional trust mechanism to ensure safety of running code on a remote resource. Their system provides trust by using the concept of certified code. They assert that the certified code can be run even in trust less environment.

2.1.2 Managing Trust in Grid Networks. Azzedin et al. [7] have have studied the importance of trust in Grid environments and have shown how the computing performance in Grid can be improved by using the concept of trust and avoiding the large computational overhead incurred by the security infrastructure. Since our model exploits few of the advantages of Managing Trust approach, we elaborate the strategy used in more detail. In [8, 9] several aspects of trust values are considered as part of the global reputation model. First, the trust values decay with time. Second, trust relationships are based on a weighted combination of the *direct* relationship between domains as well as on the *global* reputation of the domains. Finally, the trust model should stimulate organizations to sanction entities who are not behaving consistently in the Grid environment and who break trust relations.

The following notations are introduced in [8, 9].

- Let D_i and D_j denote two domains.
- Let $\Gamma(D_i, D_j, t, c)$ denote a trust relationship based on a specific context c at a given time t of D_i towards D_j.
- Let $\Theta(D_i, D_j, t, c)$ denote a direct relationship for the context c at time t of D_i towards D_j.

- Let $\Omega(D_j, t, c)$ denote the global reputation of D_j for the context c at time t.
- Let $DTT(D_i, D_j, c)$ denote a direct trust table entry of D_i for D_j for context c. It is a table that records the trust value from the last transaction between D_i and D_j.
- Let $\Upsilon(t - t_{ij}, c)$ denote the decay function for specific context c where t is current time and t_{ij} is the time of the last update of DTT or the time of the last transaction between D_i and D_j.

Contexts in Grids can be numerous, varying from executing jobs, storing information, downloading data, and using the network. The main issue in trust management is computing $\Gamma(D_i, D_j, t, c)$. In [8,9], $\Gamma(D_i, D_j, t, c)$ is computed as the weighted sum of direct relationship between domain and global reputation of the domain.

$$\Gamma(D_i, D_j, t, c) = \alpha \cdot \Theta(D_i, D_j, t, c) + \beta \cdot \Omega(D_j, t, c) \tag{2.1}$$

where $\alpha, \beta \geq 0$, $\alpha + \beta = 1$.

The direct relationship is affected by the time elapsed between inter-domain contacts, hence

$$\Theta(D_i, D_j, t, c) = DTT(D_i, D_j, c) \cdot \Upsilon(t - t_{ij}, c) \tag{2.2}$$

The global trust for domain D_j is computed as

$$\Omega(D_j, t, c) = \frac{\sum_{k=1}^{n} DTT(D_k, D_j, c) \cdot R(D_k, D_j) \cdot \Upsilon(t - t_{kj}, c)}{\sum_{k=1}^{n} (D_k)} \tag{2.3}$$

where $R(D_k, D_j)$ is the recommender's trust level.

Since reputation is primarily based on what domains say about another domain, the recommender's trust factor $R(D_k, D_j)$ is introduced to prevent cheating through collusions among a group of domains. Hence, $R(D_k, D_j)$ is a value between 0 and 1 and will have a higher value if D_k and D_j are unknown or have no prior relationship among each other and a lower value if D_k and D_j are allies or business partners.

This approach has several limitations. First, under the assumption that we have several domains, it is costly to compute the global trust (Equation 2.3). Even though scalability in terms of storage of trust values is improved by using the concept of domain, it does not solve the scalability issue related to computation of global trust value. Since the formula constitutes direct summation, the computational overhead increases as the number of domains increase in the network. According to the formula (Equation 2.3), we need to to consider all domains in the network for increased accuracy. This limits its computational scalability. Second, the authors suggest limiting the number of contexts, to reduce the fragmentation of the trust management space, in their study. Specifically, the authors reduced the number of contexts in the study to only three: printing, storage, and computing. However, in Grid environments we deal with many more contexts than just printing, storage, and computing. An example would be the evaluation of trust and reputation for network characteristics which is an essential part of any Grid infrastructure. So the problems existing due to space fragmentation in implementation, when the number of context increases does not seem to be feasible for Grids.

We conclude that, a model that scales well for large computation should be adopted to compute global reputation, and the problem rising out of the consideration of very large number of contexts be solved using new models or techniques.

2.2 Peer-to-Peer Networks

Peer-to-Peer provide an infrastructure to locate information and trade products. Gnutella [29] and Kazaa [42] are few of the extremely popular decentralized peer-to-peer systems used by millions of users worldwide. The anonymous, open nature of these systems, which is one of the most attractive features, offers an almost ideal environment for malicious users to infect the network. Various trust models being developed to reduce the risk of malicious users and inauthentic files in the network. Few of them are discussed in the following subsections.

2.2.1 EigenTrust Algorithm for P2P Networks. A reputation management algorithm for P2P networks, called EigenTrust, is introduced in [41]. Every peer i rates other peers based on the quality of service they provide. Therefore, every peer j with whom i had business, will be rated with a grade s_{ij}. To globalize this algorithm the individual grading scheme is normalized as described in [41]. Hence, for each peer j, the normalized local trust value c_{ij} is defined as follows:

$$c_{ij} = \frac{max(s_{ij}, 0)}{\sum_{j} max(s_{ij}, 0)} \tag{2.4}$$

The normalized local trust values throughout the P2P domain needs to be aggregated. This procedure can be done by means of a transitive trust mechanism: peer i asks its acquaintances for their opinions about other peers and weighs the opinion by the trust it places in his friends:

$$t_{ij} = \sum_{k} c_{ik} c_{kj} \tag{2.5}$$

where t_{ij} represents the trust that peer i puts in peer j based on the opinion of his friends $\{k\}$. The coefficients are assembled into a matrix, $C = [c_{ik}]$, hence the equation (2.5) is written in matrix notation as shown in equation (2.6).

$$\vec{T_i} = C^T \vec{c_i} \tag{2.6}$$

These normalized local trust values are aggregated using transitive trust (i.e., $T = (C^T)^2 c_i$ would mean that peer i is asking for opinion of his friends' friends, and $T = (C^T)^3 c_i$ for the opinions of their friends). Therefore, after n iterations, where n is the rank of the matrix, the transitive trust is obtained. Hence, T should converge to the same vector for every peer i. Since C is a row stochastic matrix, its largest eigenvalue is 1. Hence, the principal eigenvector of C^T is computed (i.e. the left eigenvector of C). The derivation of computation of trust shows that if the trust ratings between different entities is written in the form of a matrix, then trust of each entity is given by the principal eigenvector of the matrix. This algorithm converges very fast because of the size of the second eigenvalue as shown in [32].

The eigenvalue approach chosen in 2.2.1 is explicitly designed for P2P networks. It basically involves only a single context, which is file downloads. It cannot be directly applied to the underlying architecture of Grids that introduce virtual organizations providing an explicit classification of resources, users, and their multiple contexts.

2.2.2 Reputation-Based Approach for Choosing Reliable Resources for P2P Systems. Damiani et al. [19] have proposed a self-regulating system using a distributed polling algorithm by which resource requesters can assess the reliability of a resource offered by a participant before initiating the download. The algorithm uses a specific protocol, which consists of six phases. The Resource searching phrase consists of a Query message similar to a standard Gnutella interchange [29]. The second phase constitutes the Resource selection and trust vote polling where the poll responses includes a public key with which poll responses will need to be encrypted. This ensures the confidentiality of votes and association with those who have expressed

them. The third phase is vote evaluation, wherein the validation of votes is done. The fourth phase which is the Best servent check, decides on the best reliable source for download and the final phase is resource downloading.

2.2.3 PeerTrust. PeerTrust [50] aims to develop a trust mechanism for system in which peers can quantify and compare the trustworthiness of other peers and perform trusted interactions based on their past interaction histories without trusted third parties. Work includes a trust model and a decentralized and secure trust manager. A reputation-based trust model for P2P electronic communities has [67,68] has been developed. The model incorporates total number of transaction a peer performs and the credibility of the feedback sources in addition to the feedback a peer receives through its transactions with other peers.

2.2.4 Managing Trust in Decentralized Applications. This project [46] aims to provide solutions for decentralized trust management. The main focus is on turning current decentralized information systems into trusted environments in which participants can accurately assess the trustworthiness of their eventual partners in electronic exchanges [2]. Their major goal is to help P2P systems become trust-aware environments and develop many P2P applications, ranging from file exchange or playing chess on-line to P2P auctioning and other e-commerce oriented applications.

2.2.5 Free Haven System. The Free Haven Project (www.freehaven.net) aims to deploy a system for distributed, anonymous, persistent data storage which is robust against attempts by powerful adversaries. It describes a design for a publishing system that can resist the attempts of powerful adversaries to find or destroy any stored data. Reputation in P2P Anonymity Systems [20] focuses on anonymous publishing and explain why the systems can benefit from reputation. They describe designing reputation systems while still ensuring anonymity.

2.3 Internet Applications

Several trust management solutions are discussed for trust within internet applications [1]. Few of them are classified in the following subsections.

2.3.1 Public Key Infrastructure(PKI). The PKI [3] certificate systems mainly deal with authentication. They authenticate the owner's identity using a digital certificate. A digital certificate is issued by a certification authority and it verifies only the authentication. It does not handle the policies regarding what resources or services are permitted to be accessed by a user. The two best known certificate systems are the PGP and X.509. In PGP system [51] a user generates a (PublicKey, SecretKey) pair that is associated with his unique ID that is of the form(Name, EmailAddress). A user may specify the degree of trust that may be designated unknown, untrusted, marginally trusted or completely trusted. X.509 [55] contains more information than PGP. It contains the names of the signature schemes used to create them and the time interval in which they are valid. It differs from PGP in its level of centralization of information. Everyone will obtain certificates from an official certifying authority(CA) in X.509 framework.

2.3.2 Web of Trust. In web of trust [28], trust relationships and reputations are distributed and locally managed by each participant. The common ancestor of this approach is probably PGP, that allows users to certify other users' public keys without need for a Certification Authority. A PGP public key infrastructure consisting of a global database with public keys of participants together with claimed identities and signatures can be treated as web of trust. As a member of this infrastructure, one can decide whom to trust as an introducer of new keys to a lesser or stronger degree. Communication can take place between two identities if a link can be established between them. It employs certain heuristic to do such computations and provide

trust.

2.3.3 Platform for Content Selection (PICS). PICS [38] was developed by the World Wide Web Consortium to protect children from pornography on the Internet. The basic purpose of PICS is to provide filter between the potential viewer and web documents. Few of the filtering softwares use PICS rating to determine appropriateness of a given web page. The W3C has also published PICS Rules that can filter web pages based on policies.

2.3.4 ATT PolicyMaker. PolicyMaker is a trust management application that specifies what a public key is authorized to do. Blaze et al. [12] were one of the pioneers to study on trust management in decentralized systems. Their main goal was to provide a solution to the trust management problem that is independent of any particular application or service. They developed a trust management system called Policy-Maker, which binds public keys to access control without authentication. PolicyMaker binds public keys to predicates in policy. Certificates and policies are responsible for describing who are to be trusted and what are the actions they can perform. The PolicyMaker is responsible for ensuring that described actions actually conform to policies and certificates. Though PolicyMaker is a powerful analytical tool, the non-programmers who are likely to develop policies may have difficulty expressing policies using PolicyMaker.

2.3.5 Rule-controlled Environment For Evaluation of Rules and Everything Else (REFEREE). REFEREE [18] is a trust management system for making access decisions relating to Web documents. It needs to be integrated into a host application in order to provide trust management. It uses the PICS rating and PolicyMaker theoretical framework for interpreting trust policies and administering the trust protocols represented as software modules. It consists of two phases. One is

the bootstrap phase and the other is the query phase. In the bootstrap phase the host application gives a set of trusted assertions. In the query phase, the host application provides with action and required arguments such as credentials. REFREE runs the interpreter with the policy and list of arguments and returns an answer to the host application.

2.3.6 IBM Trust Establishment Framework. IBM Trust Establishment Framework [34] views trust management as enabling factor for e-business. It uses concept of certificates for managing trust. They have developed a role-based control model that uses certificates, a Java-based Trust Establishment module and a Trust Policy Language(TPL). The Trust Establishment module validates client certificate and maps a role to the owner of the certificate. TPL is used to specify local policy which defines what a role is permitted to do. This framework is used in web applications such as kids communities wherein only children can participate, access to large medical databases for research while limiting access to authorized people only, a business policy to select a reliable supplier or a reliable transportation. Trust management framework can be used to define policies for all such applications and enforce the policies to achieve the goal.

2.3.7 Logic-Based Formalisms of Trust. Relational formalisms have been used to model trust [36] The Authorization Specification Language(ASL) is used to specify authorization rules. They also support role-based access control. It can be used as a tool for specification and analysis of resource access trust.

2.4 Ubiquitous Computing

Study in Ubiquitous Trust [60] show that the trust is likely be an important component in the successful deployment of ubiquitous computing environments. It

examines cognitive, social, institutional, societal and technological factors to be examined within the context of ubiquitous computing.

2.5 Pervasive Computing

Trust-Based Security in Pervasive Computing Environments [39] proposes a security architecture applicable to distributed systems but geared towards pervasive computing. The trust management involves developing security policy, assigning credentials to entities, verifying that the credentials fulfill the policy, delegating trust to third parties and reasoning about users' access rights. The work is similar to a role-based access control - an approach in which access decisions are based on the roles that individual users have as a part of an organization such as doctor, nurse, manager or student. The user's access rights are computed from its properties. It allows delegation chain in which users only delegate rights to other users that they trust. Once users are given certain rights, they are responsible for the actions of the users to whom they subsequently delegate those rights and privileges.

2.6 Mobile Computing

In mobile computing, a mobile agent will not be restricted to a single domain at any point of time. It will be expected to be able to navigate between different security domains and handle them. Another issue concerned in mobile computing is agent-to-agent interaction. Work has been done highlighting trust management mechanisms, policies and mobility protocols but no trust framework has been designed or implemented.

Wilhelm et al. [66] have analyzed the problem with trust in mobile agent system. A mobile agent for e-commerce might be required to hold data related to the service provider such as the maximum price or lowest QoS. The agent might require information regarding multiple credit cards of user or even need personal

information in some situations. Finally, the agent might want to convey some very sensitive financial information information depending on the owner of the agent. To ensure security in such situations, they have identified four foundations as they refer to them. The foundations constitute blind trust, trust based on (a good) reputation, trust based on control and punishment and trust based on policy enforcement. Blind trust indicates that there is no particular motivation or belief or trust in a principal. Trust based on (a good) reputation, stems from the fact that the principal in question is well known. Trust based on control and punishment means that the trust comes from the underlying technical and legal framework to ensure the principal's proper behavior. Finally, trust based on policy enforcement is supposed to be negated by appropriate punishment.

At times it is very difficult to reliably discover a policy violation. A policy might be composed of many different rules. They establish the fact that it is not possible to enforce rules within a policy without relying on some piece of trusted hardware (TPE). But in order for the user to trust in these guarantees, it is necessary that he also trust in the TPE manufacturer. So this approach only replaces the trust with arbitrary service provider with trust in a TPE manufacturer.

2.7 Others

Reputations are also effectively used in electronic marketplaces [6, 21] as a measure of the reliability of participants. For instance, with eBay [21], buyers and sellers can express their votes(-1, 0, or 1) for each other after each transaction. Votes so collected are used by eBay to provide cumulative ratings of users that are made known to all participants. In systems like eBay, reputations are associated with physical identities and are managed at the eBay server.

Google, which is one of the well-known applications, uses the notion of trust to display the relevant information from internet web pages. It retrieves and displays the

relevant documents by sorting the web pages according to the PageRank of each web page. PageRank is one of the methods Google uses to determine a page's relevance or importance. Google employs the principal eigenvector of the matrix to compute the PageRank [40].

Reputation Systems such as CNET.com, EXP.com and expertcentral.com compute reputations by taking the feedback from experts and reviewers. OpenPrivacy (www.openprivacy.org) introduces a set of reputation services that can be used to create, use, and calculate results from accumulated opinions and reputations. Sierra, Talon, and Reptile [49] are OpenPrivacy projects that incorporate reputations to enhance searching as well as to discard unwanted information. There are also computational models [44], using social factors such as reciprocity, constituting social dictum "Be nice to others who are nice to you". The ReferralWeb system [53] helps to explore social networks.

Our Grid Eigen Trust framework does not provide any monitoring or brokering services for resources in Grid. It only provides a trust based framework. There are several monitoring based systems such as Ganglia [22] used for monitoring high performance computing systems. There are also few resource management frameworks suggested as part of traditional Grid approaches such as Condor/G [27], Nimrod/G [14] and AppLeS [11]. They do not consider the notion of trust and reputation in their frameworks.

2.8 Comparative Analysis

We provide comparative analysis to differentiate various technologies such as Grid, Peer-to-Peer, Web and Distributed Computing before comparing the models developed for these technologies. We will then compare various trust models based on the features and characteristics they support.

2.8.1 Technologies. Grid approach is not considered to be a revolution but the latest and most complete evolution of familiar developments such as the Web, peer-to-peer, distributed computing, computing and virtualization technologies according to [24, 25, 33]. "Like the Web, grid computing keeps complexity hidden: multiple users enjoy a single, unified experience, but unlike the Web, which mainly enables communication, grid computing enables full collaboration toward common business goals. Like peer-to-peer, grid computing allows users to share files, but unlike peer-to-peer, grid computing allows many-to-many sharing not only files but other resources as well. Like clusters and distributed computing, grids bring computing resources together, but unlike clusters and distributed computing, which need physical proximity and operating homogeneity, grids can be geographically distributed and heterogeneous. Like virtualization technologies, grid computing enables the virtualization of IT resources, but unlike virtualization technologies, which virtualize a single system, grid computing enables the virtualization of vast and disparate IT resources." The differentiation is shown in table 2.1. We use the features such as allowing to share information, files, and resources to distinguish these technologies. For instance, all the technologies enable sharing of inforamtion and files in one way or the other. But coming to sharing of resources, the web does not support sharing of resources and the Distributed computing requires physical proximity to allow sharing of resources.

2.8.2 Trust Models. The problem with many of these trust management systems are that they are used to identify a static form of trust (Sections 2.6, 2.3.4, 2.3.5 and 2.3.6). But it is quite important to address trust that dynamically changes with respect to time. In most of the models the trust and reputation are taken to be the same across multiple contexts. Few of the systems such as PGP and X.509 focus on authentication and data-integrity, but do not relate to actual need or requirement of trust. Work done in trust management in Grids (Section 2.1) emphasizes on

Table 2.1: Technology Comparison

Features	Grid	P2P	Distributed Computing	Web
Shares Information	Yes	Yes	Yes	Yes
Shares Files	Yes	Yes	Yes	Yes
Shares Resources	Yes	No	Yes	No
Allows Collaboration	Yes	Yes	Yes	No
Requires Physical Proximity	No	No	Yes	No
Protocol Independent	Yes	No	No	Yes
Platform Independent	Yes	No	No	Yes

avoiding the overhead that comes from the current security infrastructure by providing alternative solutions to ensure trust. But some of the features supported by security infrastructure, such as mutual authentication and single sign-on, which are the backbones that enable Grid computing, should not be sacrificed for performance issues.

Trust evaluation should be dynamic having a notion of learning and adaptation. We have compared our approach with other approaches. The table 2.2 and table 2.3 provides a detailed comparative analysis of the various trust models. The distinguishing features in table 2.2 include the support of multiple or single contexts, the amount of total computational overhead, the scalability of the model and the computation methodology employed within various models. The table 2.3 distinguishes them based on the form of trust, whether static or dynamic, the type of model, its language dependency, dependence on Certificate Authority and specific hardware requirements to support the trust model. Our work exploits advantages from two of the existing models. One is The EigenTrust for P2P systems discussed in Section 2.2.1 and the other is Integrating Trust in Grid discussed in Section 2.1.2

Table 2.2: Trust Models Comparison 1

Trust Model	Section No	Context	Computation Overhead	Scalability	Computation Methodology
Grid Eigen Trust	N/A	multiple	Negligible	Yes	Distributed
Trust in Grid	2.1.2	multiple	Reasonable	No	Centralized
P2P EigenTrust	2.2.1	single	Negligible	Yes	Distributed
PeerTrust	2.2.3	multiple	Reasonable	Partially	Distributed
Mobile Computing	2.6	multiple	Reasonable	Partially	Centralized
ATT PolicyMaker	2.3.4	N/A	Reasonable	Partially	Centralized
REFREE	2.3.5	multiple	Reasonable	Partially	Centralized
IBM Trust Mgmt	2.3.6	N/A	Reasonable	Yes	Distributed

Table 2.3: Trust Models Comparison 2

Trust Model	Form	Type	Language Dependent	Certificate Authority	Hardware Required
Grid Eigen Trust	dynamic	global	No	No	No
Trust in Grid	dynamic	global	No	No	No
P2P EigenTrust	dynamic	global	No	No	No
PeerTrust	static	global	No	No	No
Mobile Computing	static	not global	No	No	Yes
ATT PolicyMaker	static	not global	Yes	No	No
REFREE	static	not global	No	No	No
IBM Trust Mgmt	static	not global	Yes	No	No

CHAPTER 3

REQUIREMENTS OF GRID EIGEN TRUST

A number of technical and behavioral factors interplay to influence success of any design and deployment. Trust has become one such factor in computing field. Ample trust is needed for successful deployment of various computing environments. When facing social dilemmas or uncertainties, individuals always try to seek opinions and trust on subjects of concern.

To apply trust frameworks to community Grids, [64], Laszewski points out that it is important to revisit in more detail the role of virtual organizations(VOs) and institutions participation in creating them to illustrate its complexity. In discussions with Laszewski, we have found that as shared resources in a virtual organization are contributed by various institutions, it is important to recognize the need of an elaborate reputation service network that deals with the fact that resources can be part of multiple domains and VOs. The different cases are depicted in Figure 3.1. Here, the institutions I_1, I_7 and I_2 are a part of virtual organizations A, B and C respectively, whereas one part of I_3 belongs to VO A and the other part belongs to VO C. Institution I_6 does not belong to any of these virtual organizations. Considering these various possibilities, the management of reputation in Grids becomes quite complex.

In such complex Grid settings, any given reputation framework for the Grid must adhere to a basic set of minimal requirements. Grid Eigen Trust focuses on the following smaller issues which are highly related to the more general issues of Grid computing research.

- Scalability
- Robustness

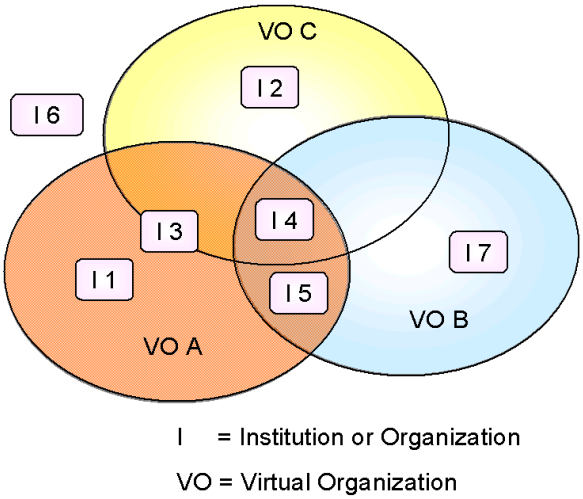

Figure 3.1: Institutions Forming Virtual Organizations

- Extensibility
- Cross-platform compatibility

Each of these features will be described in brief in this chapter, with additional information available with the discussion of the implementation of the system.

3.1 Scalability

There are two different aspects of scalability that Grid Eigen Trust focuses on; one is scalability in terms of number of services, users or resources using the system, the other is scalability in terms of computation and storage.

Grid computing enables seamless integration of computing systems, clusters, data storage, specialized networks as well as sophisticated and scientific instruments and softwares. Each of these provide different services and different contexts. Maintaining and evaluating trust and reputation for each of these services for the multiple contexts they provide is a daunting task. The system should be able to support any new context or service that might be dynamically added to the system at any point of

time. The organization of these services and resources as shared domains and virtual organizations make the problem more complex. A hierarchical based computational model is designed to tackle this issue. Reputations are computed at various levels. At the lowest level the reputation of each entity either a resource or service or user is computed. The next higher layers compute the reputation of overall reputation of the organization or institutions to which the entities belong. The highest layers handle the reputation of virtual organizations formed.

The other aspect of scalability is related to the computational overhead that increases as the size of the network increases. This issue is addressed using standard mathematical model to perform computations. EigenVector mathematically model, which is used by well known reputation systems [32,41], shows to converge fast as the systems scales along large networks. Distributed storage mechanisms are employed to ensure scalability in storage systems.

The system should adhere to a simple design that enables minimal overhead in terms of computational, infrastructure, storage requirements. As the system scales to larger networks, the computation should be distributed so that the computational overhead is minimal.

3.2 Robustness

Loss of failure of any services should not imply loss of any reputation information. The systems need to be fault tolerant. The system should not enable advantages for malicious entities with poor reputations to continuously change their identities to obtain new status. To avoid false reporting a mechanism must be provided to evaluate the accuracy of the reported reputation. The framework should be fair while calculating the reputation without giving any preference to newcomers. Ideally, the reputation of an institution should not calculated within the institution; rather it must be computed by combining independent evaluations from external ser-

vices reusing the institutions entities. We incorporate notion of automated feedback mechanism to handle this issue by preventing human interaction and delegating the task to specialized services.

3.3 Extensibility

It would be impractical to think that Grid Eigen Trust could, in its current form, satisfy all of the possible requirements for a reputation system in a Grid computing environment. It is for this reason that Grid Eigen Trust System has extensibility at the very core of the system.

Much of this extensibility comes from the use of web services technologies such as SOAP [13] and WSDL [17] for object access and object description. Even though HTTP [23] is currently used as transfer protocol, it is possible to use to use other transport and invocation methods such as FTP [52], BEEP [56], SMTP [43] or another yet undeveloped protocol to transfer the requests from one host to another. It also allows multiple programming languages to be used although the use of SOAP causes additional overhead over straight XML.

3.4 Cross-Platform Compatibility

The cross-platform compatibility is achieved by designing our system within The Open Grid Services Architecture (OGSA) [26] framework. OGSA builds on the Web services technology mechanisms to uniformly expose Globus Grid services semantics to support integration with various underlying native platforms. OGSA includes the use of the Web Services Description Language (WSDL) [17] to describe the methods of a service and the Simple Object Access Protocol (SOAP) [13] to actually utilize the system.

CHAPTER 4
GRID EIGEN TRUST ARCHITECTURE

Grid Eigen Trust has a layered architecture and adopts a hierarchical model for computation of reputation. Its methodology and specifications are discussed in detail in this chapter. First, we explain few of the terms we often use in our framework before we present our new architecture.

4.1 Terminology

In this section we define the basic terminology that will be used throughout the rest of the chapters.

4.1.1 Definition: Trust. Trust is an ambiguous concept that defies exact definition. However, a notion of trust can be established with sufficient detail for specific operational purpose.

T. Grandison and M. Sloman [1] have defined Trust in the following way. "Trust is a complex subject relating to belief in honesty, truthfulness, competence, reliability etc. of the trusted person or service." It summaries to say that "trust is really a composition of many different attributes such as reliability, dependability, honesty, truthfulness, security, competence, and timeliness, which may have to be considered depending on the environment in which trust is being specified." They define Trust in simple terms as "the firm belief in the competence of an entity to act dependably, securely and reliably within a specified context" and distrust as "the lack of firm belief in the competence of an entity to act dependably, securely and reliably within a specified context."

For our framework, we define trust as the underlying principle for a security mechanism applicable in a global context. As such, trust is a mechanism for reducing

risk in unknown situations. Hence, trust has an important role as a commodity that enables interactions in an unfamiliar environment while weighing the risks associated with actions performed in that environment.

4.1.2 Definition: Reputation. We use the definition of Reputation given in [5]. According to Laszewski, "Reputation refers to the value we attribute to a specific entity, including agents, services, and persons in the Grid, based on the trust exhibited by it in the past. It reflects the perception that one has of another's intentions and norms. Resource reputation provides a way of assigning quality or value in regards to a resource." If a resource is known to provide certain qualities over a period of time, then it is assumed to have good reputation.

4.1.3 Entity. We do not provide a definition for an Entity. We refer to an entity as a resource, service, or a user. We use the term entity in place of a resource or service or a user throughout the remaining chapters.

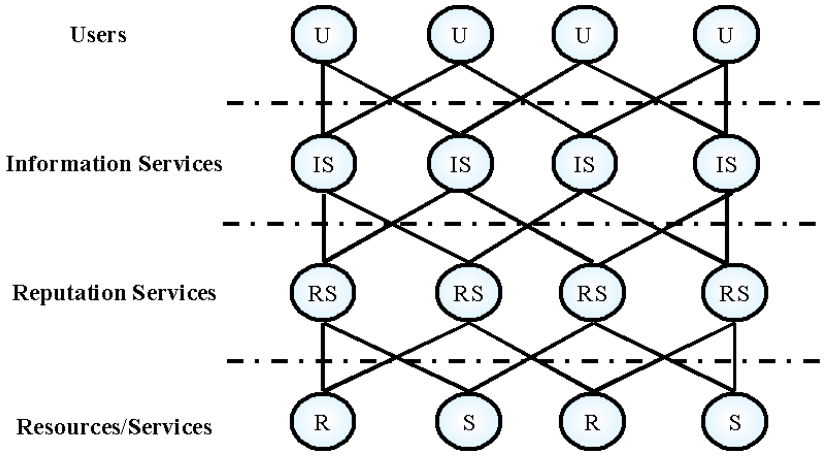

Figure 4.1: Layered Architecture of Grid Eigen Trust

4.2 Layered Architecture

The layered architecture of the Grid Eigen Trust is depicted in Figure 4.1. At the highest level in the hierarchy is the users who wish to use Grid Services. This could be a software or a human user interacting with the system. A user interacts with underlying application through any of the operating systems prevalent such as Windows, Unix or MacOS. At the next level is the Information Services. The users contact the information services for the details about reputation of services or resources required. The Reputation Services form the next layer. These services provide reputation information to the Information Services so that they can be published and available to the users. Furthermore, the Reputation Services can contact the Information Services for information regarding the resources or services that were not registered to the reputation service.

The communication between Reputation Services and Information Services takes place through an InformationService interface. The interface is shown in Figure 4.2 and Figure 4.3. The authentication methods retrieve the essential authentication information to access the information service. The binding templates and description provides the technical details of services. The contact information and discoveryURL provides the information about the owner of the services being published in the registry. There is also a set and get property method that enables the Information service to store a property that is specific to a service. It could be the reputation value that the service holds or could be any other qos related property.

At the bottom level in the layered architecture is the set of actual services or resources. Reputation Services compute the reputation of these entities and publishes them in the Information Registries. The communication takes place from higher to lower layers. There is also a possibility of users directly contacting the reputation services for reputation information.

At the core of each Reputation Service is the Grid Eigen Trust computation

```
public interface InformationService{
/**
 * get Authentication Information
 * @return authentication value
 */
public String getAuthenticate() ;

/**
 * Get Binding template information
 * @return binding Template info.
 */
public String getBindingTemplate() ;

/**
 *
 * @return contact of the service
 */
public String getContact() ;

/**
 * @return description of the service
 */
public String getDescription() ;

/**
 * @return the URL of service
 */
public String getDiscoveryURL() ;

/**
 * @return user defined property
 * such as reputation
 */
public String getProperty() ;
```

Figure 4.2: Operations to be Supported by Information Service-Part 1

```
/**
 * Set the authentication information.
 * @param string
 */
public void setAuthenticate(String string) ;

/**
 * Set binding template information
 * @param string
 */
public void setBindingTemplate(String string) ;

/**
 * Set contact information
 * @param string
 */
public void setContact(String string) ;

/**
 * Set description about service.
 * @param string
 */
public void setDescription(String string) ;

/**
 * Set the discovery URL.
 * @param string
 */
public void setDiscoveryURL(String string) ;

/**
 * Set the property that is associated with
 * service. It could be reputation data or any
 * other qos data.
 * @param string
 */
public void setProperty(String string);

 }
```

Figure 4.3: Operations to be Supported by Information Service-Part 2

engine. The computation engine uses an hierarchical reputation model for computing the reputation.

4.3 Hierarchical Reputation Model

The hierarchical reputation model used in our computation engine is shown in Figure 4.4. This model was designed with suggestions from Laszewski. At the lowest level in the hierarchy are the entities. Entities support various contexts. The trust associated at this level with the entities is named as Entity Trust. The next higher level constitutes the institutions to which entities belong. Institution Trust refers to trust attributed to institutions. The top level constitutes Virtual Organizations and the trust attributed to them is referred to as VO rust.

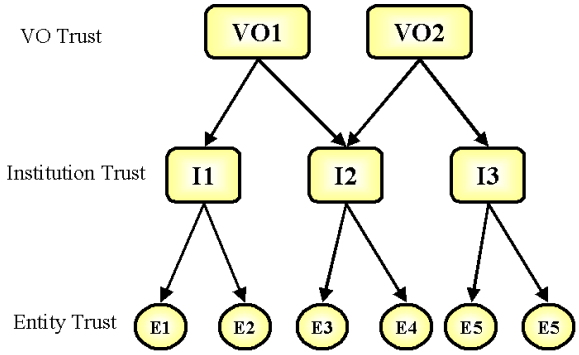

Figure 4.4: Hierarchical Model Used in Grid Eigen Trust

4.3.1 Entity Trust. We establish a trust value for each entity based on various contexts it supports within an institution. This trust is termed as Entity Trust.

4.3.2 Institution Trust. We use the term *Institution Trust* for referring to a trust value for each institution. *Institution Trust* differs from other context trust due to the fact that it agglomerates several context trust values to a single one. *It reflects*

a general opinion of the reliability of an institution to provide accurate information on what resources this institution supplies. Due to this simplification a institution trust between institutions can be calculated quickly to obtain the global trust.

4.3.3 VO Trust. We attribute trust value to virtual organizations based on the institution trust of the institutions that constitute the virtual organization. We term this trust as VO Trust.

Details about the computation are discussed in the algorithm explained in Chapter 5. We have addressed the complexity of computing reputation for each context by using this notion of hierarchy in computing reputation. To illustrate this concept, let us consider the scenario shown in Figure 4.5. In this scenario, two VOs are depicted containing two institutions each. Each institution has a set of entities, specifically physical resources, services, and users. Each of these entities support multiple contexts. The total number of entities belonging to these virtual organizations along with the context they support might be large. The problem becomes more complex as the number of institutions increase. Our implicit hierarchical model handles this issue and breaks the problem of reputation computation to three levels as already mentioned. By using such a kind of approach, we are able to break the unmanageable complex problem into simple manageable trust computations. Our reputation services uses this hierarchical model.

There can be a number of reputation services running within a virtual organization or institution. Each reputation service is responsible for a subset of entities within the hierarchy. The reputation services compute the reputation in a collaborative, but distributed fashion. Reputation values are distributed among various reputation services in the network in order to increase lookup speeds. In order to calculate and maintain the reputation, each reputation service uses the Grid Eigen Trust algorithm discussed in Chapter 5.

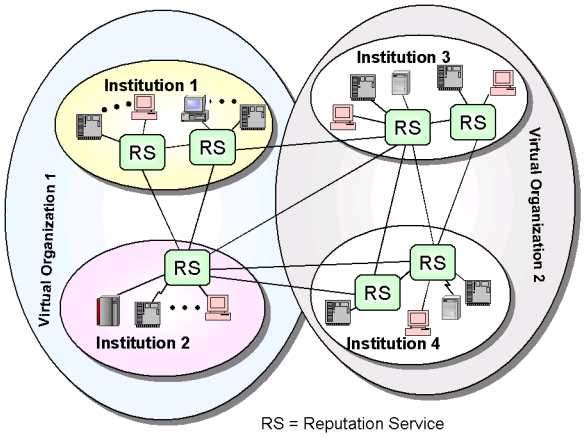

Figure 4.5: Overview of Reputation Services Distribution in Grid

4.4 Components of Reputation Service

Apart from the computation, the reputation service is also responsible for several other functions such as data gathering, information storing and retrieval. The various components of the reputation service is shown in Figure 4.6. It consists a collection manager, calculation engine, data collection manager, and reporter.

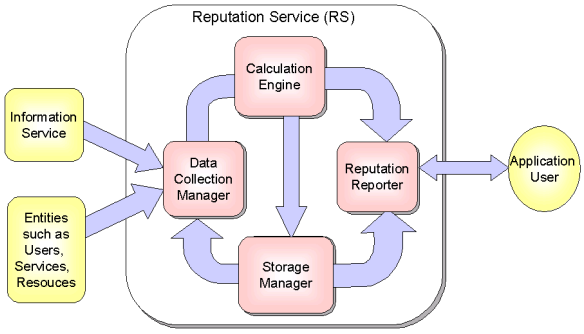

Figure 4.6: Components of a Reputation Service

When an application submits a request for a service, cast in a qualitative statement, to a reputation service, the reputation service evaluates the statement and computes the reputation for all the entities providing the required service using the heuristics explained in Chapter 5. It contacts other reputation services, if required,

and returns the information regarding the services and their reputations back to the requester. The requester can decide to select the most appropriate service by looking at the reputation values. This procedure can be easily modified for enabling and enhancing automated resource selection decisions in the Grid. We describe each of the components of a reputation service and the functionalities they support in the following subsections.

4.4.1 Data Collection Manager. The collection manager is responsible for evaluating the quality statement describing the requested reputation, and collecting relevant data from the Information Services and entities such as resources, services and users. It gives the collected data to the computation engine. There are two different types of data that needs to be collected. One is the feedback data that is required for updation after using the services. The other one is the data related to registration information of new services. The important feature of the this component is that the data that is collected needs to be authentic. Automated data collection mechanisms can be deployed to ensure the authenticity.

4.4.2 Computation Engine. The computation engine is the focal point in the architecture. It computes the Entity Trust, Institution Trust and VO Trust. It uses the heuristics explained in Chapter 5 to compute reputation values.

4.4.3 Storage Manager. The storage manager is responsible for the storage of reputation values. There are two kinds of data which we wish to store. One is the data required by computation engine which we call Computational data. The second one is the Reputation data, which is the final reputation values computed at any given time. We handle the Computational data by storing the trust values in a distributed fashion within each reputation service. Reputation data requires type of storage that ensure reliability and avoid loss of data in case of occurance of a system

or service failure. The Information Registry services or file based systems is used for such storages.

4.4.4 Reporter. The reporter is responsible for retrieving the required reputation information. It contacts the storage manager to report the reputation values whenever queried by some entity in the Grid.

CHAPTER 5
GRID EIGEN TRUST ALGORITHM

Grid Eigen Trust algorithm is designed considering few main characteristics of trust. The definition of trust is given in Chapter 4. Here we give the characteristics of trust that we considered in designing our algorithm. We use the most often used names John, Mary and Peter to explain the characteristics. They can be given as follows:

- Trust is limited to a specific context. John trusts Mary to use his computer does not mean he trusts her to handle his finances.

- Trust changes with respect to time. The trust John acquired five years ago in a specific context might not be the same as the trust attributed to him in the same context, a year ago.

- Trust is transitive. If John trusts Mary, and Mary trusts Peter, then we can conclude that John trusts Peter.

- Social trust affect the trust factor. A person is more likely to be trusted if he is trusted by other people. The trust of other people provide a basis of one's trust.

- Trust is dynamic. Managing trust involves collecting information, monitoring the current relations and evaluating the trust.

- The degree of trust that is how much we trust is important. Quantification of trust is important and should be evaluated on mathematically sound basis.

Now we need to convert these formulations to mathematical formulas as reported in [5]. An hierarchical model which divides the computation of reputation at

various levels is proposed as discussed earlier. By combining Institution Trust of the institution, which is not present in the existing trust models, and Entity Trust within institution (for specific context c at time t), we can derive a reliable trust value for the given entity for a specific context at a given point of time. We apply the eigenvector mathematical model to compute the Institution Trust of an institution. Currently, we compute the reputation of a virtual organization as weighted sum of the reputations of all institutions that belong to the virtual organization.

To describe our GridEigenTrust algorithm, we use the notations described in the next section.

5.1 Notations

- Let E_i denote an entity and I_i and I_j denote two institutions.
- Let $\Gamma(E_i, t, c)$ denote total trust based on a specific context c at a given time t of entity E_i.
- Let $\Theta(E_i, t, c)$ denote direct trust for the context c at time t of E_i.
- Let $\kappa(I_i, I_j, t)$ denote direct trust at time t of I_i towards I_j.
- Let $\Omega(E_i, t, c)$ denote the global trust of E_i for the context c at time t.
- Let $\Upsilon(t-t_{ij}, c)$ denote the decay function for specific context c where t is current time and t_{ij} is the time of the last update of reputation value.

There are two main computations involved. First is the calculation of Entity Trust which is at the lowest level as described in Section 4.3. The second one is the computation of Institution Trust between institutions. The higher levels of trust are computed

based on the Institution Trust and as per the policies defined within virtual organizations. These values are computed at the application level. Here we discuss the computation of the Entity Trust and Institution Trust.

5.2 Calculating the Entity Trust

All entities that use resources or collaborate with users within another institution, grade the quality and reliability of the requested entity. When the entity represents a certain resource or service, we usually call this trust value *service reputation*. When entities represent users, this value represents a *user reputation*, trust, or reliability parameter associated with the user. The overall grade of the entity is established as the weighted sum of the previous grade (which decays with time) and the new grade. It is also important to consider how much we trust the institution from which the remote entity (i.e. entity that gives the grade) originates its requests.

If $\Theta_p(E_i, t_i, c)$ is the previous cumulative grade established at time t_i for entity E_i within context c, $G_j(t, c)$ is a new grade given by entity from institution I_j and $T(I_j)$ institution trust of institution I_j, overall new cumulative grade $\Theta(E_i, t, c)$ can be calculated as

$$\Theta(E_i, t, c) = \frac{\alpha(c) \cdot \Theta_p(E_i, t_i, c) \cdot \Upsilon(t - t_i) + \beta(c) \cdot T(I_j) \cdot G_j(t, c)}{\alpha(c) + \beta(c)} \tag{5.1}$$

where $\alpha(c), \beta(c) \geq 0$.

The parameters $\alpha(c)$ and $\beta(c)$ reflect the context importance of the latest grade the entity received.

If an institution just joined the Grid, the initial trust values will be set to a low initial value since the trust must be earned first. However, if the entity for which we assign the trust is sufficiently similar to others in the already existing Grid, an

initial value can be obtained from these already integrated entities. We chose the lowest trust value. However, it will be penalized with a linear correction function.

Let $\Theta_0(E_i, t_0, c)$ denote the initial trust value for an entity E_i within our institution for a context c. Let $\Theta(E_i, t_i, c)$ denote the cumulative reputation value gathered from other entities (defined by equation (5.1). Then the initial trust of the entity is the weighted sum between these two values:

$$\Gamma(E_i, t, c) = \frac{\gamma(c) \cdot \Theta_0(E_i, t_0, c) + \delta(c) \cdot \Theta(E_i, t_i, c)}{\gamma(c) + \delta(c)} \tag{5.2}$$

where $\gamma(c), \delta(c) \geq 0$.

5.3 Calculating the Institution Trust

As the number of organizations increase, it becomes very difficult if we use weighted summation formulas to compute reputation. So we take take the advantage that the trust ratings can be normalized and considered as a matrix. Because the matrix is stochastic, the right eigenvector associated with the eigenvalue of 1 is the stationary distribution associated with the stochastic matrix. The values in the eigenvector represent the reputations of the institutions. The derivation of computation of trust given in EigenTrust algorithm 2.2.1, shows that if the trust ratings between different entities is written in the form of a matrix, then trust of each entity is given by the principal eigenvector of the matrix. So we could directly compute eigenvectors to get the reputation values. This is the reason we use EigenVector Mathematical Model to compute institution trust. Unfortunately, the values thus obtained are often global in nature, and lack context dependence. We solve this issue by using the Entity Trust in conjunction with the Institution Trust.

The institution trust of institution I_i toward institution I_j reflects *the opinion of institution I_i about the quality and trustworthiness of information institution I_j*

supplies. Therefore, we introduce global context, besides maintaining individual contexts (compare Section 2.1.2). In case we do have a priori knowledge about the initial trust information, we assign this value at initialization time of our algorithm.

Let the initial value of trust be represented as $C(I_j)$. *Institution* trust should be obtained through the weighted sum of direct experience and global trust value of institution I_j.

Direct experience can be calculated in the same way as in equation 5.1. It is a normalized weighted sum between $C(I_j)$, the cumulative grade from the previous period $\kappa_p(I_i, I_j, t_{ij})$ and the new grade $G(t)$.

Users within institution I_i grade the reputation of a certain entity E_j within institution I_j with grade $\Phi(E_j)$. Also, institution I_j advertises the quality of service of this entity with grade $\Delta(E_j)$. Then, institution I_i will grade reliability of information given by institution I_j with grade $G(t)$. For determining grade $G(t)$ we have three cases:

- If $\Phi \in [\Delta - \epsilon, \Delta - \zeta]$, new grade $G(t)$ is 1.
- If $\Phi > \Delta - \zeta$, new grade $G(t)$ is bigger than 1.
- If $\Phi < \Delta - \epsilon$, new grade $G(t)$ is less than 1, depending on how much the Φ differs from Δ

Direct experience that institution I_i has with I_j at some time t, $\kappa(I_i, I_j, t)$ can be calculated in the same way as in equation 5.1. It is a normalized weighted sum between $C(I_j)$, cumulative grade from the previous period $\kappa_p(I_i, I_j, t_{ij})$ and the new grade $G(t)$.

$$\kappa(I_i, I_j, t) = \frac{\alpha \cdot C(I_j) + \beta \cdot \kappa_p(I_i, I_j, t_{ij}) \cdot \Upsilon(t - t_{ij}) + \gamma \cdot G(t)}{\alpha + \beta + \gamma} \tag{5.3}$$

where $\alpha, \beta, \gamma \geq 0$.

Institution trust of institution I_j, can now be calculated using the EigenTrust methodology explained in the Section 2.2.1. If we replace s_{ij} with $\kappa(I_i, I_j, t)$ in Section 2.2.1, we obtain c_{ij} as follows:

$$c_{ij} = \frac{max(\kappa(I_i, I_j, t), 0)}{\sum_j max(\kappa(I_i, I_j, t), 0)} \tag{5.4}$$

The initial vector also needs to be replaced as $\vec{T_0} = t_0(i)$, $t_0(i) = C(I_i)$. Now we have all the ingredients to apply a power iteration for computing the principal eigenvector of C^T, which represents global institution trust values for institutions in Grids.

Similar to the approach in 2.1.2, the overall computation of the reputation can be given using the formula (5.5).

$$\Gamma(E_i, t, c) = \alpha \cdot \Theta(E_i, t, c) + \beta \cdot \Omega(E_i, t, c) \tag{5.5}$$

We can summarize the basic steps of the algorithm as follows:

Entity E_i within institution I_1 wants to use entity E_j within institution I_2 in the context c at time t.

- Consider the institution trust of I_2 computed using the EigenTrust algorithm.
- Ask I_2 about $\Theta(E_j, t, c)$, the trust value of entity E_j within institution I_2.
- In calculating the overall trust value for entity E_j, in formula (5.5) replace $\Omega(E_j, t, c)$ with institution trust of I_2 times $\Theta(E_j, t, c)$.
- Compute the overall trust for the entity $\Gamma(E_j, t, c)$ with formula (5.5).

After computing the trust values, we can compare them to suggest the resource with highest reputation. Various modifications, such as the introduction of a

statistical selection algorithm based on random variables, are obviously possible.

5.3.1 Advantages. This combined approach has several advantages. First, the EigenVector mathematical model used for computing Institution Trust converges rapidly. The proof is given in [32]. Secondly, it introduces less computational overhead than directly computing global trust values for individual entities within every context. The reason is that the number of values for computation is not too large since we are computing global trust values of institutions through hierarchies, not on overall pool of individual entities for specific contexts. The global reputation of the institutions computed affects the global reputation of each of the entities for each of the context they support, thus ensuring that global trust is computed for individual entities. We provide the experimental evaluation of our strategy with centralized summation strategy using simulations in Chapter 7.

CHAPTER 6

IMPLEMENTATION

We have provided a prototype implementation of the system along with the specification of Grid Eigen Trust. This chapter documents the design of the system and implementation decision of that design. The implementation for Grid Eigen Trust is built on the following tool sets:

- Globus Toolkit 3.0
- Java CoG Kit 1.1
- Java (JSDK 1.4.2)

The Globus Toolkit is a software toolkit that allows us to program and develop Grid-based applications. Globus Toolkit 3.0 (GT3) [25] is a usable implementation of the formal and technical specification OGSI [57] of the concepts described in OGSA. The Open Grid Services Architecture (OGSA) [26] defines common and standard architecture for Grid-based applications using concept of Grid Service as its core. Grid Services are an extension of Web Services. Web services is a distributed computing technology that allows to create client/server applications. They are platform-independent and language-independent. Web services are rapidly maturing based upon key technologies of SOAP, WSDL and UDDI. The table 6.1 provides a summary of important services provided by each of them in Web Services. Similar technologies used in Grid Services are also summarized in the table 6.1.

There are features lacking in Web services, such as persistence, notification and life-cycle management, which make them less versatile. Grid Services provides functionalities supporting these features. The features supported by Grid Services apart from the ones supported by Web Services are summarized in the table 6.2

Table 6.1: Web and Grid Services Functionalities

Name	Web Services	Grid Service	Functionality
Service Discovery	UDDI	MDS3	Find Web Services
Service Description	WSDL	GSDL	Describe Web Services
Service Invocation	SOAP	SOAP	Invoke and Pass messages.
Transfer Protocol	HTTP,FTP	HTTP,FTP	Transmit messages
	BEEP,SMTP	BEEP,SMTP	

Table 6.2: Grid Services Additional Functionalities

Services	Functionality
Factory	Create transient services
Life-Cycle management	Maintain service lifetime (creation, destruction)
Notifications	Subscribe and Notify.
Service Data	Index capabilities and characteristics of services

We have implemented Grid Eigen Trust as a Grid Service which supports the above mentioned functionality. Due to its OGSA service orientation, it can be easily integrated with other services such as a registry service or a brokering service. A registry could be a distributed service that integrates information from several sources. A brokering service could be a QoS service that negotiates with a variety of service providers to identify resources that meet user requirements. Chapter 8 shows an example application that involves integration of Grid Eigen Trust with other Grid and information services.

Java CoG Kits [63] provide good support for developing client side interfaces to the services provided in the Globus Toolkit. It can be used for formulating tasks that are generic in regards to GT2 and GT3 technologies. We have used this tool to generate tasks in our simulator. More details of how we use it are given in Section 7.2.

Java [37] is a cross-platform object oriented programming language that was first initiated by Patrick Naughton, Mike Sheridan, and James Gosling of Sun in 1991. The decision to implement Grid Eigen Trust in Java was straight forward since there

are only two languages which provide appropriate bindings required for Grid services. One is Java and the other is Microsoft's .NET, which is not mature or well supported as Java. In addition, the object oriented programming features of Java helps in writing code that is re-usable, easy to read and maintain. Modularity and information hiding can be achieved by using encapsulation. It has built in exception handling procedures that helps to produce robust applications and components. In short Java is known for producing portable, architecturally neutral, robust, and dynamic code. Another great advantage of Java is that it is an open source language. The limiting factor of using Java is with its execution speed. Since Grid Eigen Trust framework does not demand high speed, Java was chosen for its implementation.

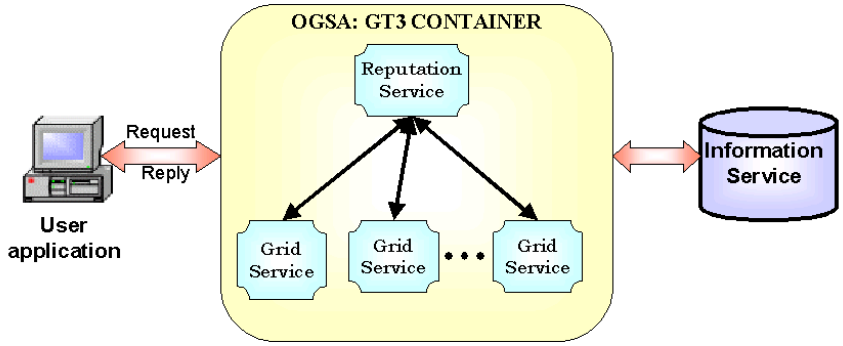

Figure 6.1: Runtime Execution Model

6.1 Runtime Execution Model

Figure 6.1 shows the runtime execution model supporting interaction with other specialized services. It interacts with users to retrieve, update and acquire reputation values. On the other hand, it also communicates with the Information Services to publish the reputation values so that other services can directly retrieve from the information services. Since it runs within the OGSA framework, it can interact with any other OGSA-based Grid Service either for feedback or for reporting reputation information. The runtime model constitutes different phases of a reputation service.

The various phases describe the life cycle of a reputation service. The four different phases in a reputation service are described in the following subsections.

6.1.1 Creation Phase. The system begins by starting up an instance of a Reputation Service. The instance fetches the initial reputation values of entities belonging to an institution or organizations directly from Information Services or from an external user. This instance tries to find out if there are any other instances of the reputation services running in its domain or vicinity and tries to reference them as neighbors.

6.1.2 Registration Phase. An entity can register to the Reputation Service at any point of time. It needs to provide information regarding its name, the contexts it supports and the various initial reputation values it attributes to each of these contexts. It can be registered as part of an existing institution, which is registered under the reputation service, or be registered under a new institution. If the registering entity does not specify the institution ID, then the service creates a new ID for the institution and registers the entity under it.

6.1.3 Updation Phase. Reputation values are updated in this phase. Reputation Service get the feedback either from other services which are OGSI based or directly from users. Whenever an entity being registered to Reputation Service is being used, it has to get a feedback either from brokering services which manage scheduling or directly from users who use the services. This can be designed based on the application requirement. We can exploit the notification mechanisms supported by OGSA framework to automatically fetch feedback to Reputation Service.

Reputation Service supports a parameter for frequency of update.This parameter is used to decide how often the reputation values needs to be updated. Currently this parameter can be set by the user directly. More sophisticated or automated

mechanisms given in Performance Analysis [35] can be used to automate this process. Our current implementation does not use this methodology. It allows the user to set the time frame as to how often it is to be updated.

6.1.4 Termination Phase. In this phase, the service updates all the reputation values to the information services before terminating.

6.2 Computation Engine

The major part of the implementation constitutes the computation engine, which is at the core of Grid Eigen Trust framework. There are two main computations performed within this engine. One is Entity Trust that is reputation of each entity for specific context. The other is Institution Trust that is the reputation of institution which is independent of context.

6.2.1 Entity Trust. Entity Trust is updated every time a service is being used by another entity. Its updation does not require reputation services to interact with other reputation service instances.

6.2.2 Institution Trust. Institution trust is updated only based on the time frame set within the Computation Engine. It requires that the reputation service compute the value in a distributed manner. Our implementation adopts the Distributed EigenTrust algorithm developed for P2P systems [41] to calculate the global reputation value using EigenVector Mathematical model. Once we do the basic computations as required by our algorithm, we perform the computation in a distributed manner. The steps can be given as follows:

- Each institution stores its own institution trust and a set of trust values belonging to its neighboring institutions which uses its resources or services.

- Using the algorithm explained in Chapter 5 the computation engine calculates the weighted trust value of each institution using the previous value, the feedback values collected and importance given to initial reputation values if present.

- Each institution sends this information to all the other institutions about which they have opinion.

- The power iteration continues till the values converge.

- The resultant values computed constitute institution trust values of each institution.

The Distributed EigenTrust algorithm [41] has proved to have several advantages. Since there are computations to be done are distributed among various reputation services, it is not computational intensive. The algorithm has shown to converge very fast and the reason for the fast convergence is discussed in [32].

6.2.3 Virtual Organization Trust. This trust is computed based on the policies written within virtual organization. It is computed as the weighted sum of institution trust of institutions multiplied by the proportion to which they contribute to the virtual organization. This trust is rarely used because the virtual organizations are dynamic in nature.

6.3 Reputation Grid Service

We have implemented our system within the OGSA framework as discussed in Section 6.1. It uses the Grid Eigen Trust algorithm to manage reputations. As the reputation service operates in a Open Grid Service Infrastructure (OGSI), the service has a number of 'operations' can be used by other components.

6.3.1 Operations. The operations supported by reputation services are implemented as an API with a set of primitives, briefly described as:

- *register:* is invoked when a new entity requires to be registered to reputation service for its reputation to be computed.
- *request:* is invoked when other services require reputation information.
- *update:* to update an existing trust value using the feedback.
- *unregister:* is invoked when a service does not want to be considered for reputation calculation.

With this set of methods any Grid service can interact with the Reputation Service, either for using reputations or for registering or updating the trust values.

6.3.2 Design. The design of reputation service is based on object-oriented methodology. The advantage is that it promote reuse of classes for further extensions of the service in future. The operations described in Section 6.3.1 are exposed in the service interfaces as Grid service operations as as shown in the WSDL Listing 6.2 and 6.3. The WSDL document gives the details about the signatures of the methods supported by the service. The namespace for the Grid Service is defined in the beginning of the document. OGSI namespaces are included next, which represents the OGSI bindings. The next important step is to import all the OGSI-specific types, messages and portTypes. There are four operations defined: register, unregister, update and request. By looking at the message types, any other service can invoke these methods.

6.3.3 Input/Output XML Strings. Requests to the reputation service are made in the form of XML string. We use simple xml notations to collect and

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://reputation.cog.globus.org/Reputation"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:gridservicesoapbinding="http://www.gridforum.org/namespaces/2003/03/OGSI/bindings"
xmlns:impl="http://reputation.cog.globus.org/Reputation"
xmlns:intf="http://reputation.cog.globus.org/Reputation"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:import location="../../ogsi/ogsi_bindings.wsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI/bindings"/>
 <wsdl:types>
  <schema targetNamespace="http://reputation.cog.globus.org/Reputation"
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="requestReputation">
    <complexType>
     <sequence>
      <element name="in0" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="requestReputationResponse">
    <complexType>
     <sequence>
      <element name="requestReputationReturn" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="updateReputation">
    <complexType>
     <sequence>
      <element name="in0" type="xsd:string"/>
     </sequence>
    </complexType>
     </sequence>
    </complexType>
   </element>
   <element name="updateReputationResponse">
    <complexType>
     <sequence>
      <element name="updateReputationReturn" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="registerReputation">
    <complexType>
     <sequence>
      <element name="in0" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="registerReputationResponse">
    <complexType>
     <sequence>
      <element name="registerReputationReturn" type="xsd:string"/>
   </element>
```

Figure 6.2: WSDL Document by the Reputation Service-Part 1

```
   <element name="unregisterReputation">
    <complexType>
     <sequence>
      <element name="in0" type="xsd:string"/>
     </sequence>
    </complexType>
   </sequence>
    </complexType>
   </element>
   <element name="unregisterReputationResponse">
    <complexType>
     <sequence>
      <element name="unregisterReputationReturn" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
  </schema>
 </wsdl:types>
   <wsdl:message name="unregisterReputationRequest">
      <wsdl:part element="impl:unregisterReputation" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="registerReputationResponse">
      <wsdl:part element="impl:registerReputationResponse" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="unregisterReputationResponse">
      <wsdl:part element="impl:unregisterReputationResponse" name="parameters"/>
   </wsdl:message>
   </wsdl:message>
   <wsdl:message name="requestReputationRequest">
      <wsdl:part element="impl:requestReputation" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="updateReputationRequest">
      <wsdl:part element="impl:updateReputation" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="requestReputationResponse">
      <wsdl:part element="impl:requestReputationResponse" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="registerReputationRequest">
      <wsdl:part element="impl:registerReputation" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="updateReputationResponse">
      <wsdl:part element="impl:updateReputationResponse" name="parameters"/>
      <wsdl:part element="impl:updateReputationResponse" name="parameters"/>
   </wsdl:message>

  <wsdl:portType name="ReputationPortType">
      <wsdl:operation name="requestReputation">
 <wsdl:input message="impl:requestReputationRequest"/>
 <wsdl:output message="impl:requestReputationResponse" />
      </wsdl:operation>
      <wsdl:operation name="updateReputation">
 <wsdl:input message="impl:updateReputationRequest"/>
 <wsdl:output message="impl:updateReputationResponse" />
      </wsdl:operation>
      <wsdl:operation name="addReputation">
 <wsdl:input message="impl:registerReputationRequest"/>
 <wsdl:output message="impl:registerReputationResponse"/>
      </wsdl:operation>
      <wsdl:operation name="removeReputation">
 <wsdl:input message="impl:unregisterReputationRequest"/>
 <wsdl:output message="impl:unregisterReputationResponse"/>
      </wsdl:operation>
   </wsdl:portType>
</wsdl:definitions>
```

Figure 6.3: WSDL Document by the Reputation Service-Part 2

```
<service_request>
<service_name>Math service</service_name>
<service_context>3</service_context>
<service_reputation>50</service_reputation>
<service_cost>8</service_cost>
</service_request>
```

Figure 6.4: Reputation Request String

```
<service_reply>
<institution_id>1</institution_id>
<service_reputation>0.94</service_reputation>
<service_context>3</service_context>
<service_id>1068586241634</service_id>
</service_reply>
```

Figure 6.5: Operations Supported by the Reputation Service

```
<service_update>
<service_name>Math service</service_name>
<service_context>3</service_context>
<service_reputation>78</service_reputation>
<service_id>1068586241637</service_id>
<institution_id>1</institution_id>
</service_update>
```

Figure 6.6: Operations Supported by the Reputation Service

send reputation requests and responses. Sample input, output and update strings are shown in Figure 6.4, Figure 6.5 and Figure 6.6 respectively.

In the output string, here only one listing is shown, but there could be multiple such replies concatenated together. The client can parse this simple xml string and display the output in any desired format. Optionally these values can be directly given to other services for automated resource selection.

6.4 Visualizer

A visualizer is also developed using to view the changing reputation values of various entities belonging to a reputation service. A simple sample screen-shot show-

ing 20 services with three context that each of them support is shown in Figure 6.7. There can be thousands of such entities. Visualization can be changed as per application requirements. User specific selections can be developed as part of graphical user interface to make viewing more flexible. As the simulation cycle progresses, the chart shows updated reputation values.

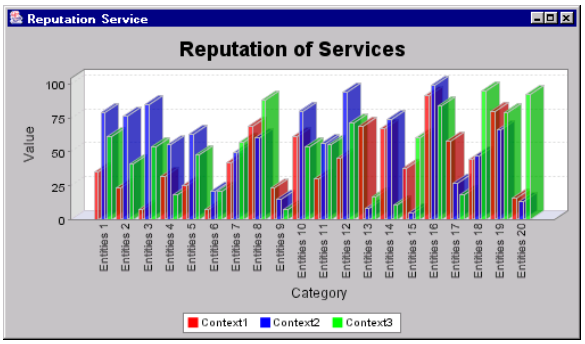

Figure 6.7: Reputations of Services for Different Contexts

CHAPTER 7

EXPERIMENTAL EVALUATIONS

In order to evaluate Grid Eigen Trust systems and understand its behavior when the number of users, resources and services increase within virtual organizations we have developed a simulator. There are several simulators present in Grid community. But none of them provides an environment to simulate our system. A survey of existing simulators are given in Section 7.1.

7.1 Grid Simulators

Existing Grid simulators include Bricks, MicroGrid, SimGrid and GridSim.

- *Bricks:* The Bricks [4] focuses on client/server interaction in global high performance computing systems. It allows for a centralized scheduling strategy, which does not scale well to large Grid systems. Our system is not a scheduler neither a replacement for scheduling. As such we could not use Bricks to simulate our system.

- *MicroGrid:* MicroGrid [47] is an emulator modeled after Globus. It allows for the execution of Globus enabled applications on a virtual Grid system. It imposes severe overhead in development as it requires the construction of an actual application to test the scheduling systems. The MicroGrid is also a scheduling simulator. Since it is application dependent it does not provide the flexibility to modify it to provide to our requirements, we could not extend it to run our simulations.

- *SimGrid:* SimGrid [59] is designed to simulate application scheduling. It is restricted to a centralized scheduler and time-shared systems. Simulating competing users, applications and schedulers each with their accompanying policies

can only be done by manually extending the simulator. Using SimGrid, we get the advantages of event-modeling required for schedulers. Since our system is not a scheduler we do not derive advantages using SimGrid.

- *GridSim:* GridSim [30] allows for the simulation of distributed schedulers, and is specifically aimed at simulating market-driven economic resource models. While its computational resource models are highly configurable, it only supports a basic notion of network connectivity. It does not simulate the underlying network's dynamics to a high degree of accuracy which is required when we wish to evaluate the performance. Since it is mostly used for simulating clusters, we do not find it appropriate to use for our framework.

- *GridG:* Recently research is being done in developing a tool called GridG [50], that is used for synthesizing realistic computational Grids. It produces structured network topologies that obey the power laws of Internet topology. This work is still in progress.

7.2 Grid Eigen Trust Simulator

Many of the simulators discussed in Section 7.1 are designed towards testing resource schedulers. Since we do not find advantages extending these simulators, we have developed a simulator that simulates feedback mechanisms for Grid Eigen Trust framework. We assume that these feedbacks currently generated by the simulator are to be produced in real-time using specialized services such as quality of service managers. The values can be set optionally using human interaction from applications. Simulations were basically done to test the behavior of Grid Eigen Trust system in Grid networks with multiple virtual organizations in various settings.

7.3 Network Model

The simulator simulates a number of reputation services running in several virtual organizations. Each reputation service has one or more institutions associated to it. These institutions are interconnected to each other by a power-law network. The reason we use the power-law networks for our simulation is due to the fact that the future Grids will be embedded in the Internet topology, thereby follow their power-law networks. The World Wide Web, Gnutella [29] and The Electrical Power Grid follow the power law network topology. In a power-law network, a new node always tries to connect to the node which has largest number of other nodes connected to it. This means that a new node tries to connect to a highly interconnected node.

For network related assumptions such as network bandwidth and latency we run simulations using values based on the observations made in a measurement study [58] in peer-to-peer systems for real networks. More specific details are given in Section 7.7.

7.4 Initialization

The simulator generates m_i number of reputation services and n institutions for each reputation service where m_i and n are specified by the user. If these values are not specified, then it generates the reputation services and institutions based on the number of entities that requires to be simulated. Each institution consists of users, resources and services. A user or resource or service is implemented as an entity. The simulator generates specified number of entities for each of the institutions. It generates a specified number of contexts for each entity. A context with respect to a resource refers to computing power, storage capacity, network bandwidth or application specific functionalities. A context with respect to a user refers to programmer, designer, or any such user-defined role. For entities representing users, it

also generates number of tasks as specified. These tasks are generated using the Java CoG Kit 1.1.

7.5 Execution

The simulation proceeds in simulation cycles. Each simulation cycle consists of retrieving all the tasks requests and simulating feedbacks. A simple brokering is performed to match the requests with available resources or services. Actual usage of resource is not simulated, only the feedbacks are being simulated. Every entity that is assigned to use a resource or service, sends a feedback to the system, which updates the reputation value of the resource. The process continues until all the tasks are completed. Entity Trust is updated whenever a feedback is received but Institution Trust does not get updated each time a feedback is received. There is a frequency of update parameter within the service that determines how often the Institution Trust needs to be updated when running the reputation service in real time. However when using the simulator, we update the Institution Trust for every simulation cycle.

7.6 Feedback Generator

The feedback in the system is simulated using a random number generator. The feedback values generated are always between the range of 0 and 100. If an entity is presumed to have a very good reputation, there is very less probability that an entity sends a very low feedback value to the same entity in a short span of time. Two extreme feedback values (one very high and one very low value) on same entity within a short span of time is unreal. For this reason, we have taken this possibility into consideration while generating a random feedback value.

7.7 Comparison Criteria

We are particularly interested in comparing our distributed EigenVector strategy with a centralized summation strategy as we refer to it. In centralized summation strategy which similar to the approach taken in Section 2.1.2, EigenVector mathematical model is not used for computing global reputation. Instead of performing distributed computation among various services, the values are accumulated to a single server where the computation is done. To illustrate our comparison criteria and the methodology used to measure time, we will describe how we have set up our simulation tests.

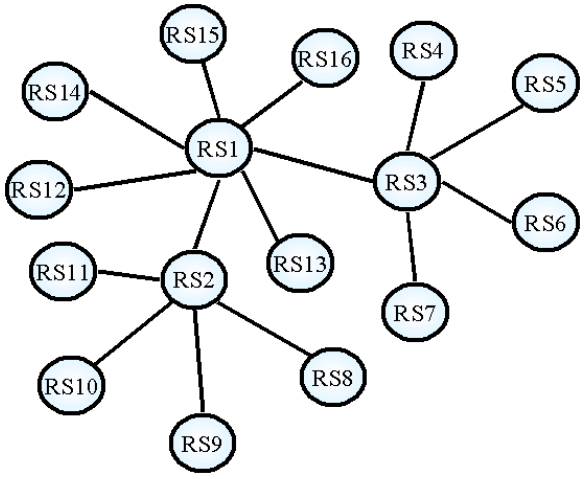

Figure 7.1: Reputation Services Network

Figure 7.1 shows a number of reputation services interconnected to each other being started up by the simulator. Details of how these services might be running within virtual organizations or institutions is elaborated in Section 4.3. We only focus on the interconnectivity and communication issues for evaluating the computational time using simulations. Each of the reputation services shown in Figure 7.1 compute VO Trust, Institution Trust and Entity trust as shown in Figure 7.2 and explained in Section 4.3.

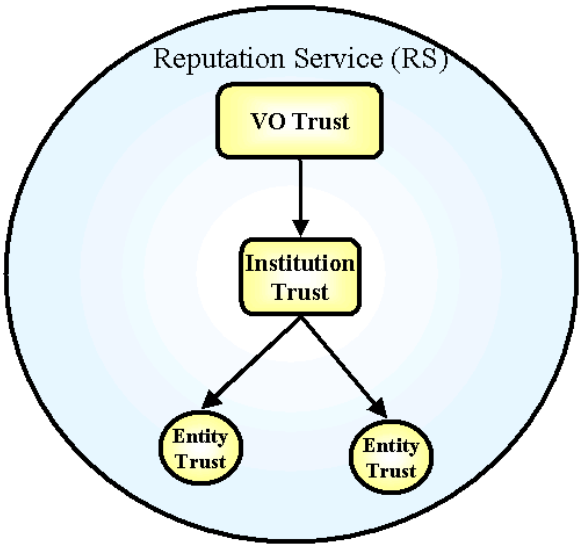

Figure 7.2: Trust Computations of a Reputation Service

Since the computational time for Entity Trust and VO Trust used in both the strategies is same, we do not include the time taken to compute them in our measurement. We specifically are interested in measuring the total time taken for computing Institution Trust using the distributed EigenTrust algorithm used in our model with a centralized summation algorithm. Before we differentiate the methodologies employed by both the algorithms, we present few statistical experimental measures used in our simulator.

We have implemented our reputation service as an OGSA Grid service. The important parameter to be considered while using Web or Grid services is the time taken by one service to send information to other services in the network. We use the measurement analysis test done for peer-to-peer systems for making assumptions about the latency time which affects the total time taken for sending information including the computational time. Studies in peer-to-peer systems [58], show that approximately 20 percentage of the peers have latencies of utmost 70ms for an average bandwidth of 1Mbps. For bandwidth larger than 1Mbps, this can be decreased further. Since we are only interested in the percentage improvement of the computa-

tional time using both strategies, we will assume that the latency at each node while communication is 30ms. In that case, every reputation service has a wait time of 30ms while sending information to other reputation services. This latency parameter can also be modified in the simulator if a different value is to be used.

7.7.1 Distributed EigenVector Algorithm. We have used the distributed algorithm given in [41] for implementing the EigenVector Model within the Grid Eigen Trust framework. The steps are listed in Section 6.2.2. Since many services compute the reputations in a distributed manner, the computation time is drastically reduced. One other factor for reduction in time is due to the fact that there needs to be less reputation services communicating with each other at any point of time.

We make this concept more clear by using Figure 7.1. It shows a network with reputation services numbered from RS1 to RS16. Consider the service, RS1 which is connected to RS2, RS3, RS11, RS12, RS13, RS14, RS15 and RS16. Assume the EigenTrust computation for Institutions is invoked at reputation service RS1. RS1 computes the reputations of all the institutions that belong to it and updates the trust values of all the institutions from which it has utilized services or resources. Once that is done, it notifies its neighboring services, RS2, RS3 RS11, RS12, RS13, RS14, RS15 and RS16 to perform similar computations. We find that all the services compute the calculations concurrently. That is within the same span of time all these 8 services finish their computation. We have handled this issue in our serialized simulator by dividing the wait time or the latency time by the total number of services working together concurrently at the same time. Several iterations of the distributed algorithm is performed similarly and the total time taken for all the iterations is measured.

7.7.2 Centralized Summation Algorithm. In Centralized Summation Algorithm, instead of performing the computations at various services in a distributed manner, the computations are done in a centralized service. One other major differ-

ence is that instead of using the EigenVector model to compute the rank a summation strategy similar to the one explained for evaluating the global reputation for domains as explained in Section 2.1.2 is employed. For uniformity in computing time, we use the same Virtual Organization, Institution and Entity setup even for the centralized methodology. Every reputation service in this case will not perform computations. They sent the trust values obtained from feedback to the centralized service. Here the computations cannot be done concurrently since single service is responsible for computations. The centralized service has to spent the specified latency time while getting values from each of the services RS1 to RS16. This increases the total computational overhead while using the strategy. Similar to the computational time measured for distributed strategy as described in Section 7.7.1, we measure the total time taken for updating the Institution Trust.

We specifically compute the computational time taken in both scenarios by varying the size of entities and contexts. Each experiment is run 100 times and we take the average of these experiments for evaluation. We compute mean, min, max and standard deviation values to compare our results.

7.8 Experimental Scenarios

We have considered four use case scenarios to test and analyze our system.

7.8.1 USE CASE 1: Varying Number of Entities. We have conducted experiments for evaluating the computational overhead incurred by our strategy as the number of entities increases. We started the simulation with an initial of 50 entities. The number of contexts for this simulation was kept constant to be five. After each simulation run, we measured the total time taken for updation of Institution trust for both our system and centralized strategy. We increased the number of entities by 50 each time and continued till we reached up to a total of 500 entities. We repeated the

Table 7.1: Increment Entities in Grid Eigen Trust for USE CASE 1

Entities	Min (sec)	Max (sec)	Mean (sec)	Standard deviation SD(sec)	Mean-SD (sec)	Mean+SD (sec)
50	1.81	2.01	1.91	0.049	1.864	1.963
100	1.82	1.99	1.9	0.044	1.867	1.955
150	2.75	2.97	2.87	0.066	2.804	2.938
200	3.67	3.94	3.83	0.077	3.756	3.911
250	4.58	4.94	4.79	0.096	4.693	4.887
300	5.52	5.91	5.78	0.109	5.677	5.896
350	6.41	6.88	6.6	0.125	6.496	6.747
400	7.3	7.89	7.65	0.147	7.503	7.798
450	8.31	8.87	8.59	0.175	8.415	8.766
500	9.23	10.21	9.59	0.181	9.418	9.780

experiment for 100 times and computed the mean, min, max, standard deviation and displacement of mean from standard deviation of the values retrieved. The values are shown in table 7.1 and table 7.2. All the values are measured as time in seconds.

The values show that as the number of entities increase the total time required for computing the reputation increases. We have compared the mean values obtained using our strategy with the mean values obtained for centralized strategy. The results shows that the percentage improvement achieved using our strategy is between 65 and 72 percentage over the centralized strategy. The comparison values are shown in table 7.3. The box and whisker plots for both the strategies are shown in Figure 7.4.The whiskers depicts the min and max values. The box shows the standard deviation from the mean values.

The major reason for the improvement in the computation time is due to the fact that in distributed computation method many agents compute the trust concurrently whereas in centralized the values have to reach to the centralized server or agent and then the computation has to be done.

Table 7.2: Increment Entities in Centralized Strategy for USE CASE 1

Entities	Min (sec)	Max (sec)	Mean (sec)	Standard deviation SD(sec)	Mean-SD (sec)	Mean+SD (sec)
50	5.28	7.0	5.58	0.250	5.331	5.832
100	5.39	5.80	5.58	0.106	5.481	5.693
150	7.68	10.51	8.59	0.671	7.922	9.265
200	12.34	14.0	13.46	0.707	12.75	14.17
250	14.96	16.10	15.72	0.247	15.47	15.96
300	16.02	16.95	16.44	0.269	16.17	16.71
350	18.72	21.95	19.44	0.523	18.91	19.96
400	22.25	22.81	22.5	0.109	22.4	22.7
450	25.51	27.93	26.27	0.293	25.98	26.57
500	27.8	29.09	28.4	0.366	28.08	28.82

Table 7.3: Comparative Analysis Table for USE CASE 1

Entities	Grid Eigen Trust (sec)	Centralized Strategy (sec)	Improvement Percentage
50	1.91	5.58	65.71
100	1.9	5.58	65.78
150	2.87	8.59	66.58
200	3.83	13.46	71.52
250	4.79	15.72	69.52
300	5.78	16.44	64.81
350	6.6	19.44	65.93
400	7.65	22.5	66.13
450	8.59	26.27	67.3
500	9.59	28.4	66.26

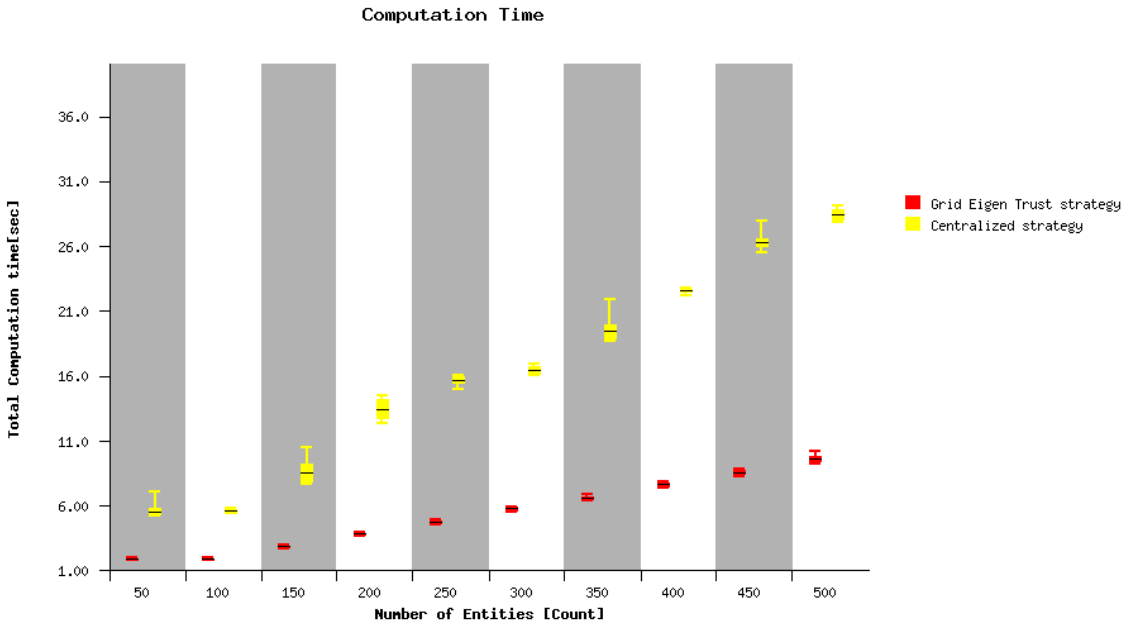

Figure 7.3: Total Computation Time for USE CASE 1

7.8.2 USE CASE 2: Varying Number of Contexts. In designing parallel and distributed applications in Grid, we are interested not only in using a compositions of multiple individual resources, but also in using appropriate networking and I/O technologies for interconnection between resources. Thus, we get multiple contexts, not only for resources and services, but also for interconnecting links.

For example, consider there is a cluster in a certain organization that has two types of network connection between the nodes: Ethernet and Myrinet. Bandwidth and latency are the two characteristics that distinguishes them. Ethernet connection is very reliable connection and processes that are lengthy have very good chance of completion. On the other hand, Ethernet is known to be very slow compared to Myrinet, much owing to the TCP protocol it uses. Myrinet is very fast and reliable for short jobs on this particular cluster but not for long ones. If one requires only the bandwidth and latency characteristics to decide on which connection to choose, then considering bandwidth and latency as two different contexts and attributing values to

Table 7.4: Increment Contexts in Grid Eigen Trust for USE CASE 2

Entities	Min (sec)	Max (sec)	Mean (sec)	Standard deviation SD(sec)	Mean-SD (sec)	Mean+SD (sec)
1	1.84	1.85	1.84	0.004	1.841	1.851
2	1.83	1.84	1.83	0.004	1.834	1.843
3	1.84	1.93	1.88	0.03	1.849	1.923
4	1.87	2.07	1.97	0.082	1.89	2.061
5	1.88	1.97	1.91	0.040	1.875	1.956
6	1.87	1.95	1.92	0.035	1.887	1.958
7	1.88	1.96	1.92	0.034	1.89	1.963
8	1.93	1.98	1.96	0.021	1.941	1.984
9	1.95	2.07	2.00	0.07	1.930	2.073
10	2.01	2.03	2.02	0.008	2.014	2.031

them solves the problem. The user can choose to use Ethernet or Myrinet by looking at their bandwidth and latency contexts.

In addition to bandwidth and latency, there could also be complex parameters such as jitter, routing and multi-cast support that might be required when trying to use the interconnection links for other applications. In that case the context cannot be limited to two but now becomes five. Continuing this way, it is not possible to generalize context to a small number. A Grid environment must support multiple contexts at any point of time.

We ran experiments on our Grid Eigen Trust system by varying the number of contexts. The major improvement of our strategy over the centralized strategy is with managing the storage space and avoiding fragmentation of space. Since our system does not use a centralized mechanism for storing the reputation values, the problem of fragmentation of space as discussed in [8, 9], is not applicable to our system. It is not reasonable to compare the amount of memory used for our system and a system defined in [8, 9] since the solutions take different approaches for reputation computation. In addition to overcoming the space fragmentation problem, the computational

Table 7.5: Increment Contexts in Centralized Strategy for USE CASE 2

Entities	Min (sec)	Max (sec)	Mean (sec)	Standard deviation SD(sec)	Mean-SD (sec)	Mean+SD (sec)
1	1.08	1.66	1.33	0.241	1.097	1.580
2	1.24	2.20	1.92	0.208	1.720	2.137
3	1.84	2.18	1.99	0.26	1.73	2.25
4	2.19	2.24	2.21	0.020	2.19	2.236
5	2.24	2.46	2.38	0.097	2.282	2.477
6	2.55	3.24	3.10	0.207	2.90	3.315
7	2.85	3.80	3.17	0.145	3.02	3.320
8	3.13	3.55	3.28	0.189	3.097	3.477
9	3.17	3.31	3.29	0.057	3.239	3.355
10	3.69	4.22	3.98	0.220	3.768	4.209

overhead is also improved using our strategy.

Similar to the use case discussed in Section 7.8.1, we computed the total time taken to calculate the Institution trust. We ran the experiment by incrementing the context by one each time for 10 simulations. Here the number of entities was kept constant to be only 50. Each of the experiment was run for 100 times and we computed the average of all the runs. The table 7.4 and table 7.5 shows the mean, min, max, standard deviation and displacement of mean from standard deviation (i.e. mean+standard deviation and mean-standard deviation) values. All the values are measured as time in seconds.

The mean values are compared in table 7.6. Initially when the number of context is only one, the centralized strategy takes less computational time in comparison to our strategy. The negative 27.49 percentage shows the improvement of computational time for centralized strategy over our strategy. But we find that the values are positive from the second row where the number of contexts is 2 and there is an improvement of around 4 to 40 percentage in the computational time. The box and whisker plots are shown in Figure 7.4. Each box and whisker shows min, max, mean

and standard deviation values.

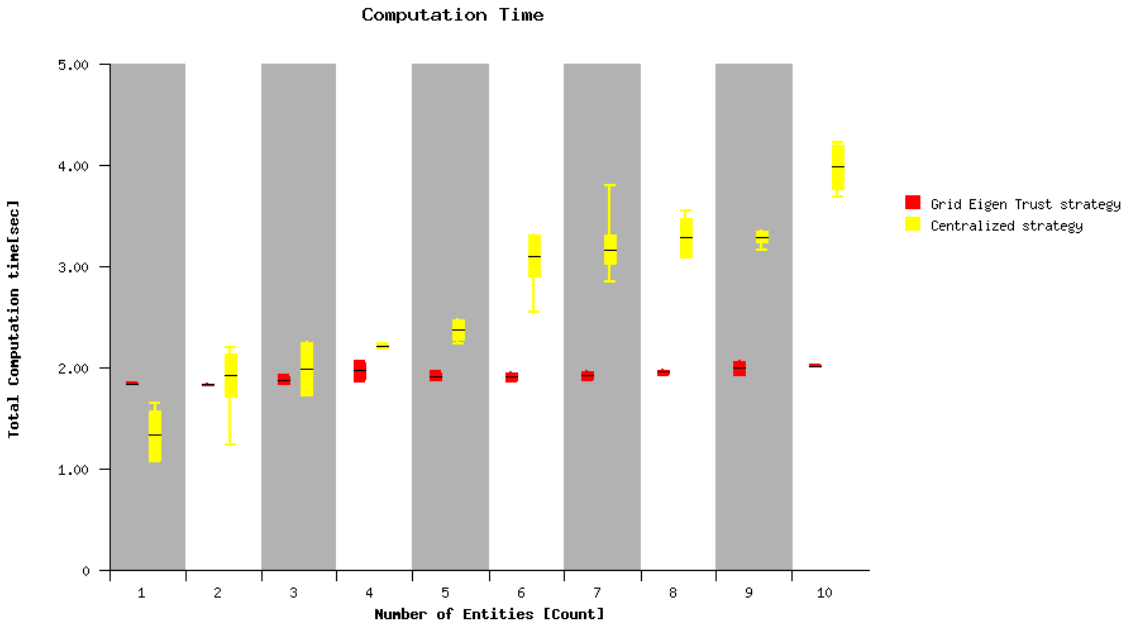

Figure 7.4: Total Computation Time for USE CASE 2

7.8.3 USE CASE 3: Reputation Service Failures Analysis. In Grid environment, where hundreds or even thousands of service requests are made to remote servers, failures are an inevitable event. There can be failures in power systems, operating systems, communication networks or even in middle-ware. Services must be made resilient to failures that can occur at every level and every interface.

Providing robustness in the time of intrinsic failures and crashes is one of the essential features supported by any reliable system. Even though OGSA [26] address issues relating to availability and robustness by introducing the concept of Factories, it does not deal with failing or unavailable instances of Grid service. It also does not deal issues such as what needs to be done if a service overloaded.

In order to overcome these limitations that occur due to inevitable failures, we have designed Grid Eigen Trust system to be able to manage these vulnerabilities. As

Table 7.6: Comparative Analysis Table for USE CASE 2

Contexts	Grid Eigen Trust (sec)	Centralized Strategy (sec)	Percentage Change
1	1.84	1.33	-27.49
2	1.83	1.92	4.66
3	1.88	1.99	5.543
4	1.97	2.21	10.69
5	1.91	2.38	19.49
5	1.92	3.10	38.11
7	1.92	3.17	39.22
8	1.96	3.28	40.29
9	2.00	3.29	39.29
10	2.02	3.89	49.28

explained in Section 4.3, the distributed nature of the hierarchical reputation Model used in Grid Eigen Trust obviates single point failures in the system. Failures occurring at the individual reputation services are effectively handled by using the services of information registries. As discussed in the Section 4.2, the reputation services contact the information services or registries to publish the reputation information. Other services or applications contact the information services to get the reputation values. The frequency rate at which the reputation services publishes the updated reputation values is dependent on the application requirements. Thus, at any point of time all the reputation values are stored in the information registry. If a reputation service instance fails at any point of time, then a new instance of the reputation service can be started in order to manage the reputation. The values from the information registry can be fed back to the instance so that the reputation services takes the values from information registry as initial reputation values. It continues updating subsequent reputation values using these values rather than starting to evaluate the reputation of a service from the scratch.

7.8.4 USE CASE 4: Extensibility Analysis. The extensibility support provided by the Grid Eigen Trust System comes from the fact that the system adopts OGSA architecture and defines a generic interface to connect to Information Services. Much of the extensibility feature within our framework comes from the use of web services technologies such as SOAP [13] and WSDL [17] for object access and object description. Any application implemented as a Web or Grid service can easily be integrated into our framework. Since the communication is done using SOAP and WSDL protocols, any applications that understand these protocols can communicate and interact with our reputation service. This makes our framework extensible.

7.9 Limitations

In this section, we present few of the limitations incurred in our approach as well as our implementation. Future research can be taken up to overcome these limitations.

- One of the limitations can be named as sociological limitation. Since reputation is intensely related with society and its relations, the notion of trust might discourage few of the resources or services from sharing their services. Most of the time services with bad reputation are seldom used and the services with high reputation are used very often. This might discourage the services which do not provide high quality of services and on the other hand encourage the services which provide a very good quality of service. Also few institutions or organizations might be reluctant to allow their entities to interact with reputation services from other institutions.

- We have not investigated on computing the optimal number of agents that has to be running in any organization at any point of time to assure less computational and time overhead. This can ensure that few of the reputation service are not

loaded with many entities and few of the services have only few entities whose reputations are to be managed.

- The accuracy of the distributed and centralized algorithm which were used in our simulation tests, were not presented using mathematical complexities formulas. Since the simulation involved power-law networks, a more complex mathematical model can be researched to compute such complexities theoretically. Instead, a set of experimental results have been presented for verification.

CHAPTER 8

POSSIBLE EXAMPLE APPLICATIONS

In this chapter, we will discuss how we could use the Grid Eigen Trust System to interact with the Information Services and other Grid Services. We will give an example to show how these systems can interact with each other to provide automated computing.

Information services are a vital part of any Grid software infrastructures. It provides fundamental mechanisms for discovering and monitoring the existence and characteristics of resources, services, computations and entities which could be part of a Grid. Grid Services as explained in Chapter 6 extends conventional Web service functionality. Since the Reputation Service is implemented as OGSA Grid service, it not only inherits several advantages of a Grid service, but also makes it interoperable with other Grid services. Web or Grid services are published to be used by software unlike websites which are directly used by humans. Software applications using various Grid services can be developed for facilitating easy use of the computational power provided by Grid infrastructure.

Resource allocation or scheduling Grid services can use reputation as one of its QoS parameters to make resource selection decisions in automated systems. Reputation to be used by such a service is computed by the Grid Eigen Trust Engine present in the Reputation Grid service and published in Information Services. Other Grid services then retrieves the reputation information thus published to take quality of service decisions.

Grid Services can also directly interact with Reputation Service by using the endpoint URL. We can also exploit notification mechanisms supported by Grid services to develop automated interactive applications. Notifications allow clients to be notified whenever changes occur in a Grid service. Notifications in Grid services is

related to service data. They allow both Pull and Push notifications. Pull approach is used when observers need to get different information whenever an event has occurred. In Push approach the changed data is send to all the observers along with the notification message.

For instance, the Reputation Service can be registered to a QoS Grid service to get automated feedbacks on usage of resources or services instead of allowing human intervention for feedback. The QoS Grid service uses the Notification mechanism to notify the Reputation service about the feedback once a job is completed. This avoids unnecessary polling by Reputation service for feedbacks and facilitates autonomic computing.

Thus the OGSA architecture makes our system to be platform-independent, language-independent, robust and easily interoperable with sophisticated services for automated dynamic resource management.

8.1 Application

To illustrate the how exactly the interaction takes between between the Information Services, Reputation Services and other Grid Services, we provide a hypothetical application example that integrates all of them.

The example is inspired by the infrastructure needed to obtain climate and weather forecasts [45, 64]. In climatology and weather forecast it is typical to develop systems that are used for identifying, accessing, preparing, assimilating, predicting, managing, analyzing, mining, and visualizing a broad array of meteorological data to predict mesoscale weather events such as floods, tornadoes, hail, strong winds, lighting and winter storms.

Let us consider an example of an application pertaining to assimilating, analyzing and visualizing meteorological information. The meteorological information is retrieved using various sensors and tools. Automated Services running at sites where

sensors are kept, capture the sensor data.

Let us assume that we use Grid services for each of the services required by the application as show in Figure 8.1.

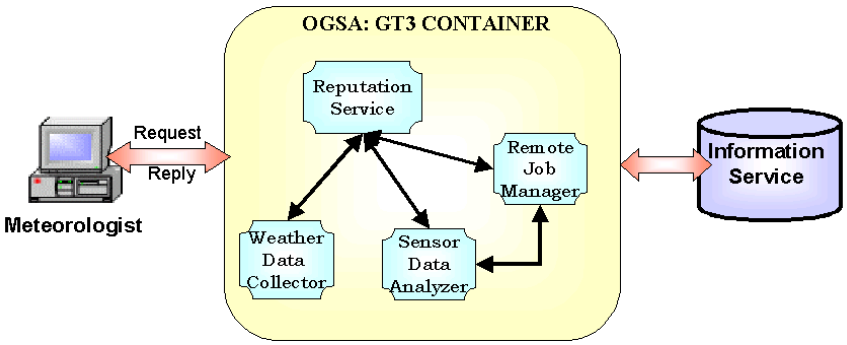

Figure 8.1: A Meteorological Example Application

A meteorologist will try to get the captured sensor data using the services running at the Data collection sites. Now he needs to send this data for analysis. He might want to contact Grid services that provide the analysis of the captured meteorological information. He contacts an Information Service to find about the services available. The information service provides a number of services that do the analysis. The meteorologist wants the service to be reliable otherwise, he might not be able to predict the weather conditions in the right time. If the service is not reliable then it might cause problems due to which the weather predictions cannot be done. So he contacts a Reputation Service regarding the reputation of the services providing analysis. The reputation service provides reputation information of all the services that are registered to it and support the required context, which is analyzing the meteorological data.

After having selected the service that performs analysis, the meteorologist wants computational power needed to perform huge computational analysis on the data collected. He needs to find out remote job manager services that enable the user to use remote computational resources. He optionally can use the reputation

service to select the most reliable job manager service and perform the computations required. Similarly, he might be interested in Grid services that provide visualization and similar other services.

In this example, we find that there is lot of interaction taking place between services. This interactions can be automated by writing appropriate software applications for the functionalities discussed. In that case, the interaction of the meteorologist with the application is simplified. He does not have to worry about any searching and selection of services. Since the services are designed to be interoperable they can communicate with each other and provide the final results to the meteorologist. Thus reputation services play significant role in enhancing automated selection services and aid in developing sophisticated scientific applications.

CHAPTER 9

SUMMARY AND CONCLUSIONS

We have presented Grid Eigen Trust system that provides a less computation-intensive, cross-platform, robust and extensible framework for managing reputations in Grid-based systems. We provided a brief overview of few of the existing trust models available for peer-to-peer systems, Grid computing systems, internet applications, pervasive computing, ubiquitous computing and mobile computing. We then presented the requirements for a reputation-based system and showed how Grid Eigen Trust system fulfills those requirements.

We have presented the layered architecture including the hierarchical model of Grid Eigen Trust in Chapter 4. We also described the important functionalities supported by various components of Grid Eigen Trust and explained the importance of the Computation Engine. Chapter 5 described the new algorithm that is used by our framework. Chapter 6 provided the details about the implementation of the system including the syntax of service requests and the WSDL document required for Web Services interaction.

Our Grid Eigen Trust system promises to be less computational intensive based on our experimental evaluations given in Chapter 7. Our implementation harnessing the technology of web services gives multiple advantages of our system over the existing systems. Our system shows to be interoperable, platform-independent, protocol-independent and extensible due to the fact that it has been implemented as a Grid service within the OGSA framework. The possibility of integrating with specialized Grid services makes Grid Eigen Trust system suitable for autonomic computing.

9.1 Future Work

Even though Grid Eigen Trust provides basic functionalities required for a reputation framework, there are other areas of research that attracts attention. Robust security systems can be developed to ensure trust in feedback mechanisms in cases where human interaction is involved.

We have not used MDS3 as our Information Registry since MDS3 is in the initial stage of development. We have only defined an interface that needs to be implemented to interact with Grid Eigen Trust. As long as the Information Registry is able implement the interface we have specified, it can be easily used for publishing and retrieving the reputation values. The features supported for information services by GT3 itself can be exploited to support reputation publishing and retrieval.

Even though we have shown how our system handles various types of failures such as system failure or service failures using one of our use case scenarios, more fault tolerant mechanisms can be developed and deployed. Currently we use a parameter to set the frequency during which the institution trust needs to be updated. More advanced methodologies can be used to provide automated updation of these trust values.

Another important and interesting area of research as an extension to this work is designing an automated resource selection system based on reputation. Several challenges related to load imbalance needed to be addressed in such systems. If selection is merely based on reputation, then the services with high reputations will always suffer from heavy load. Robust algorithms can be developed to handle such issues in developing automated resource selection systems. Such work would greatly enhance autonomic computing.

BIBLIOGRAPHY

[1] A survey of trust in Internet application. *IEEE Communications Surveys and Tutorials*, 3(Fourth Quarter), 2000. http://www.comsoc.org/livepubs/surveys/public/2000/dec/grandison.html.

[2] Karl Aberer and Zoran Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Tenth International Conference on Information and Knowledge Management ACM CIKM 2001*, pages 310–317, Atlanta, Georgia, 5-10 November 2001. ACM Press. http://lsirpeople.epfl.ch/despotovic/CIKM2001-trust.pdf.

[3] Adams and Farrell. IETF RFC 2510: PKI Certificate Management Protocols. ftp://ftp.rfc-editor.org/in-notes/rfc2510.txt, March 1999. visited Oct 24th, 2003.

[4] Kento Aida, Atsuko Tekefusa, Hidemoto Nakada, Satoshi Matsuoka, Satoshi Sekiguchi, and Umpei Nagashima. Performance Evaluation Model for Scheduling in Global Computing Systems. *The International Journal of High Performance Computing Applications*, 14(3):268–279, Fall 2000. http://dlib.computer.org/conferen/hpdc/8579/pdf/85790352.pdf.

[5] Beulah Alunkal, Ivana Veljkovic, and Gregor von Laszewski. Reputation-based Grid Resource Selection. In *Workshop on Adaptive Grid Middleware*, number ANL/MCS-P1109-0903, New Orleans, Louisiana, 27 September 2003. AGridM 2003. http://www.iit.edu/∼alunbeu/publications/reputation.pdf.

[6] Amazon. http://www.amazon.com, Oct 2003. visited Oct 20th, 2003.

[7] Farag Azzedin and Muthucumaru Maheswaran. Towards Trust-Aware Resource Management in Grid Computing Systems. In *Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, pages 452–457, Berlin, Germany, May 21-24 2002. IEEE Computer Society Press. http://www.cs.mcgill.ca/∼anrl/PUBS/ccgrid2002_farag.pdf.

[8] Farag Azzedin and Muthucumaru Maheswaran. Evolving and Managing Trust in Grid Computing Systems. In *Canadian Conference on Electrical and Computer Engineering 2002*, pages 1424–1429, Hotel Fort Garry, Winnipeg, Manitoba, Canada, May 12-15 2002. IEEE Computer Society Press. http://www.cs.mcgill.ca/∼anrl/PUBS/ccece2002_farag.pdf.

[9] Farag Azzedin and Muthucumaru Maheswaran. Integrating Trust into Grid Resource Management Systems. In *International Conference on Parallel Processing 2002*, pages 47–54, Vancouver, B.C.,

Canada, August 18-21 2002. The International Association for Computers and Communications, IEEE Computer Society Press. http://www.cs.umanitoba.ca/∼anrl/PUBS/icpp2002_farag.pdf.

[10] S. Ba and P. A. Pavlou. Evidence of the Effect of Trust Building Technology in Electronic Markets: Price Premiums and Buyer Behavior. *MIS Quarterly*, 26(3):243–268, Sep 2002. http://www.sba.uconn.edu/users/sulin/BaPavlou.pdf.

[11] F. Berman and R. Wolski. The AppLeS Project: A Status Report. In *The 8th NEC Research Symposium*, May 21-22 1997. http://www.cs.ucsd.edu/groups/hpcl/apples/pubs/nec97.ps.

[12] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *IEEE Symposium on Security and Privacy,1996*, Oakland CA, May 6-8 1996. IEEE Press. ftp://dimacs.rutgers.edu/pub/dimacs/TechnicalReports/TechReports/1996/96-17.ps.gz.

[13] Don Box, Davin Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple Object Access Protocol. Technical Report NOTE-SOAP-20000508, May 2000. http://www.w3.org/TR/SOAP.

[14] Rajkumar Buyya, David Abramson, and Jonathan Giddy. Nimrod/G: An Architecture of a Resource Management and Scheduling System in a global Computational Grid. In *the 4th International Conference on High-Performance Computing in the Asia-Pacific Region*, pages 283–289. IEEE Press, May 2000. http://www-unix.gridforum.org/mail_archive/perf-wg/pdf00000.pdf.

[15] The Large Hadron Collider Project. http://lhc.web.cern.ch/lhc/general/gen_info.htm, Oct 2003. visited Oct 24th, 2003.

[16] B. Chang, K. Crary, M. DeLap, R. Harper, J. Liszka, T. Murphy VII, and F. Pfenning. Trustless grid computing in ConCert. In M. Parashar, editor, *Grid Computing – Grid 2002 Third International Workshop*, pages 112–125, Berlin, November 2002. Springer-Verlag. http://www-2.cs.cmu.edu/∼concert/papers/grid2002/ grid2002.pdf.

[17] Erik Christensen, Fancisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language. Technical Report NOTE-wsdl-20010315, March 2001. http://www.w3.org/TR/wsdl.

[18] Yang-Hua Chu, Joan Feigenbaum, Brian LaMacchia, Paul Resnick, and Martin Strauss. REFEREE: Trust management for Web applications. *Computer Networks and ISDN Systems*, 29(8–13):953–964, 1997. http://www.farcaster.com/papers/www6-referee/www6-referee.htm.

[19] Ernesto Damiani, De Capitani di Vimercati, Stefano Paraboschi, Pierangela Samarati, and Fabio Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *9th ACM conference on Computer and communications security*, pages 207–216. ACM Press, Nov 2002. http://seclab.dti.unimi.it/Papers/ccs02.ps.

[20] Roger Dingledine, Nick Mathewson, and Paul Syverson. Reputation in P2P Anonymity Systems. In *Proceedings of Workshop on Economics of Peer-to-Peer Systems*, June 2003. http://freehaven.net/doc/econp2p03/econp2p03.pdf.

[21] Ebay. http://www.ebay.com, Oct 2003. visited Oct 20th, 2003.

[22] Matthew L. Massie David E Culler Federico D. Sacerdoti, Mason J. Katz. Wide Area Cluster Monitoring with Ganglia. In *Cluster 2003-IEEE International Conference On Cluster Computing*, Hong Kong, 1-4 December 2003. IEEE Press. http://ganglia.sourceforge.net/papers/ Sacerdoti03Monitoring.pdf.

[23] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. IETF RFC 2616: Hypertext transfer protocol – HTTP/1.1. ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt, June 1999. visited June 12th, 2003.

[24] Ian Foster. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, August 2001. http://www.gl.iit.edu/database/frame/compendex.htm.

[25] Ian Foster, Carl Kesselman, Jeffrey Nick, and Steve Tuecke. Grid Services For Distributed System Integration. *Computer*, 35(6), June 2002. http://www.globus.org/research/papers/ieee-cs-2.pdf.

[26] Ian Foster, Carl Kesselman, Jeffrey Nick, and Steve Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Information. In *Global Grid Forum 5*, Edinburgh, Scotland, June 2002. Global Grid Forum. http://www.globus.org/research/papers/ogsa.pdf.

[27] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: a computation management agent for multi-institutional grids. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International*

Symposium, pages 55–63, San Francisco, CA, USA, August 2001. IEEE Computer Society Press.

[28] Caronni Germano. Walking the Web of Trust . In *Proceedings IEEE Ninth International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2000)*, pages 153–158, 100 Bureau Drive, Gaithersburg, MD, 14-16 June 2000. IEEE Computer Society Press. http://www.olymp.org/∼caronni/work/papers/wetice-web-final.pdf.

[29] Gnutella. http://www.gnutella.com/, Oct 2003. visited Oct 20th, 2003.

[30] GridSim: A Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing. http://www.cs.mu.oz.au/∼raj/grids/gridsim/, Oct 2003. visited Oct 20th, 2003.

[31] Grid Security Infrastructure. http://www.globus.org/security/, Oct 2003. visited Oct 20th, 2003.

[32] Taher H. Haveliwala and Sepandar D. Kamvar. The Second Eigenvalue of the Google Matrix. http://www.stanford.edu/∼sdkamvar/papers/secondeigenvalue.pdf, Oct 2003. visited Oct 20th, 2003.

[33] What is Grid Computing. http://www-1.ibm.com/grid/about_grid/what_is.shtml, Oct 2003. visited Oct 20th, 2003.

[34] Trust establishment. http://www.haifa.il.ibm.com/projects/software/e-Business/TrustManager/index.html, Oct 2003. visited Oct 24th, 2003.

[35] R. Jain, editor. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, 2001.

[36] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A Logical Language for Expressing Authorizations. In *IEEE Symposium on Security and Privacy*, pages 31–42, Oakland, CA, May 04-07 1997. IEEE Press. http://www.sis.uncc.edu/∼gahn/courses/ITIS6210/Fall03/ techpapers/jajodia97logical.pdf.

[37] The source for java technology. http://java.sun.com, Oct 2003. visited Oct 20th, 2003.

[38] David Singer Jim Miller, Paul Resnick. Rating Services and Rating Systems. Technical Report REC-PICS-services-961031, Oct 1996. http://www.w3.org/TR/REC-PICS-services.

[39] L. Kagal, T. Finin, and J. Anupam. Trust-Based Security in Pervasive Computing Environments. *IEEE Computer*, 34(12):154–157, 2001. http://www.cs.umbc.edu/∼lkagal1/papers/computer-article.pdf.

[40] Sepandar D. Kamvar, Taher H. Haveliwala, and Gene H. Golub. Adaptive Methods for the Computation of Page Rank. http://www.stanford.edu/∼sdkamvar/papers/adaptive.pdf, Oct 2003. visited Oct 20th, 2003.

[41] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Twelfth International World Wide Web Conference, 2003*, Budapest, Hungary, May 20-24 2003. ACM Press. http://www.stanford.edu/∼sdkamvar/papers/eigentrust.pdf.

[42] Gnutella. http://www.kazaa.com/us/index.htm, Oct 2003. visited Oct 20th, 2003.

[43] John Klensin et al. IETF RFC 2821: Simple mail transfer protocol. ftp://ftp.rfc-editor.org/in-notes/rfc2821.txt, April 2001. visited June 17th, 2003.

[44] A. Halberstadt L. Mui, M. Mohtashemi. A Computational Model of Trust and Reputation. In *35th Hawai'i International Conference on System Science*, Island of Hawaii, Big Island, 7-10 January 2002. http://www.cdm.lcs.mit.edu/people/lmui/docs/TrustReputationModel.ps.

[45] Linked Environments for Atmospheric Discovery(LEAD). http://lead.ou.edu/project_summary.html, Nov 2003. visited Nov 20th, 2003.

[46] Managing Trust in Decentralized Applications. http://lsirwww.epfl.ch/projects/swiss/trust-project.htm/, Oct 2003. visited Oct 20th, 2003.

[47] MicroGrid- Emulation Tools for Computational Grid Research . http://www-csag.ucsd.edu/projects/grid/MGridDownload.html, Oct 2003. visited Oct 20th, 2003.

[48] Keith Norman. Grid Computing. *Tesella Scientific software solutions*, 1(1), April 2003. issue V1.R1.M0 Tesella free technical supplements.

[49] OpenPrivacy. http://www.openprivacy.org/, Oct 2003. visited Oct 20th, 2003.

[50] PeerTrust Overview. http://www.cc.gatech.edu/projects/disl/PeerTrust/, Oct 2003. visited Oct 20th, 2003.

[51] The International PGP Home Page. http://http://www.pgpi.org/, Oct 2003. visited Oct 25th, 2003.

[52] Jon Postel and Joyce Reynolds. IETF RFC 959: File transfer protocol. ftp://ftp.rfc-editor.org/in-notes/rfc959.txt, October 1985. visited June 17th, 2003.

[53] ReferralWeb. http://www.cs.washington.edu/homes/kautz/referralweb/, Oct 2003. visited Oct 24th, 2003.

[54] Reputation Research Network. http://databases.si.umich.edu/reputations/index.html, Oct 2003. visited Oct 24th, 2003.

[55] W.Ford D.Solo R.Housley, W.Polk. IETF RFC 3280:internet x.509 public key infrastructure. ftp://ftp.rfc-editor.org/in-notes/rfc3280.txt, April 2002. visited Oct 24th, 2003.

[56] Marshall Rose. IETF RFC 3080: The blocks extensible exchange protocol core. ftp://ftp.rfc-editor.org/in-notes/rfc3080.txt, April 2001. visited June 12th, 2003.

[57] I. Foster J. Frey S. Graham C. Kesselman T. Maquire T. Sandholm D. Snelling P. Vanderbilt S. Tuecke, K. Czajkowski. Open Grid Services Infrastructure (OGSI). Version 1.0 (draft), 5 2003.

[58] S. Saroiu and S.D. Gribble P. Krishna Gummadi. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of the Multimedia Computing and Networking (MMCN)*, January 2002. http://www.cs.washington.edu/homes/tzoompy/publications/mmcn/2002/mmcn.ps.

[59] Simgrid. http://gcl.ucsd.edu/simgrid/, Oct 2003. visited Oct 20th, 2003.

[60] Joseph S. Valacich. Ubiquitous Trust: Evolving Trust into Ubiquitous Computing Environments. In *Workshop on Ubiquitous Computing Envionment*, Cleveland, OH USA, 24-26 October 2003. http://weatherhead.cwru.edu/pervasive/Paper/ UBE

[61] Gregor von Laszewski and Kaizar Amin. *Grid Middleware*, chapter Middleware for Commnications. Wiley, 2004. http://www.mcs.anl.gov/∼gregor/papers/vonLaszewski–grid-middleware.pdf.

[62] Gregor von Laszewski, Kaizar Amin, Mihael Hategan, and Nestor J. Zaluzec. GridAnt: A Client-Controllable Grid Workflow System. In *37th Hawai'i International Conference on System Science*, Island of Hawaii, Big Island, 5-8 January 2004. http://www.mcs.anl.gov/∼gregor/papers/vonLaszewski–gridant-hics.pdf.

[63] Gregor von Laszewski, Jarek Gawor, Peter Lane, Nell Rehn, Mike Russell, and Keith Jackson. Features of the Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 14:1045–1055, 2002.

[64] Gregor von Laszewski, Gail Pieper, and Patrick Wagstrom. Gestalt of the Grid. In *Performance Evaluation and Characterization of Parallel and Distributed Computing Tools*, Series on Parallel and Distributed Computing. Wiley, 2003. http://www.mcs.anl.gov/ gregor/papers/vonLaszewski–gestalt.pdf.

[65] Gregor von Laszewski, Mary Westbrook, Ian Foster, Edwin Westbrook, and Craig Barnes. Using Computational Grid Capabilities to Enhance the Ability of an X-Ray Source for Structural Biology. *Cluster Computing*, 3(3):187–199, 2000. ftp://info.mcs.anl.gov/pub/tech_reports/P785.ps.Z.

[66] U. G. Wilhelm, L. Buttyán, and S. Staamann. On the Problem of Trust in Mobile Agent Systems. In *Symposium on Network and Distributed System Security*, pages 114–124, San Diego, CA, March 11-13 1998. Internet Society. http://www.isoc.org/isoc/conferences/ndss/98/wilhelm.pdf.

[67] Li Xiong and Ling Liu. Building Trust in Decentralized Peer-to-Peer Electronic Communities. In *Fifth International Conference on Electronic Commerce Research (ICECR-5)*, Montreal, Canada, 23-27 October 2002. ACM Press. http://www.cc.gatech.edu/projects/disl/PeerTrust/pub/xiong02building.pdf.

[68] Li Xiong and Ling Liu. A Reputation-Based Trust Model For Peer-To-Peer Ecommerce Communities. In *IEEE Conference on E-Commerce (CEC'03)*, Newport Beach,California USA, 24-27 2003. IEEE Press. http://www.cc.gatech.edu/projects/disl/PeerTrust/pub/xiong03reputation.pdf.