Choosing Reputable Resources in Unstructured Peer-to-Peer Networks using Trust Overlays

PhD Thesis

by

Georgios K. Pitsilis

Supervisor: Dr. Lindsay F. Marshall

School of Computing Science - Newcastle University

June 2007

Thesis submitted to Newcastle University in partial fulfilment of the requirements for the doctor of Philosophy in Computing Science

Abstract

In recent years Peer-to-Peer Systems have gained popularity, and are best known as a convenient way of sharing content. However, even though they have existed for a considerable length of time, no method has yet been developed to measure the quality of the service they provide nor to identify cases of misbehaviour by individual peers. This thesis attempts to give to P2P systems some quality measures with the potential of giving querying peers criteria by which to judge and make predictions about the behaviour of their counterparts. The work includes the design of a reputation system from which querying peers can seek guidance before they commit to transaction with another peer.

Reputation and Recommender systems have existed for years but usually as centralized services. Our innovation is the use of a distributed recommendation system which will be supported by the peers themselves. The system operates in the same manner as "word-of-mouth" in human societies does. In contrast to other reputation systems the word-of-mouth technique is itself decentralized since there is no need for central entities to exist as long as there are participants willing to be involved in the recommendation process.

In order for a society to exist it is necessary that members have some way of knowing each other so that they can form relationships. The main element used to link members in an online community together is a virtual trust relationship that can be identified from the evidence that exists about their virtual partnerships. In our work we approximate the level of trust that could exist between any two parties by exploiting their similarity, constructing a network that is known as "web of trust". Using the transitivity property of trust, we make it possible for more peers to come in to contact through virtual trust relationships and thus get better results than in an ordinary system.

Acknowledgments

This work would not have been possible without the encouragement and help of many other people. I am specially grateful to my supervisor Dr. Lindsay Marshall for spending significant amounts of time helping me and the numerous discussions we had. His support was very significant, especially in providing ideas and help to find the way forward. I would also like to thank him for his support in the preparation of the conference papers we submitted during the preparation of my thesis and for the great amount of patience he showed in proofreading and revising my texts.

I am also thankful to staff and students of the School of Computing Science at Newcastle University for making my study such a joy. I would like also to thank the technical staff of the school for the high level of professionalism they showed and for their support they provide me for carrying out my experiments

Finally, I would also like to thank the Greek State Scholarships foundation for sponsoring my studies abroad for the period of three and a half years. Also a big thank to Dr. John Lloyd, the head of the School of Computing Science, for his generous support for my travel to Versailles, France where I presented my work.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Problem Statement

Peer-to-Peer information sharing environments have gained recognition and popularity in recent years. In spite of the excellent characteristics they provide for the ways that participants can collaborate, the issue of quality preservation in the shared services has not yet been considered seriously.

The reasons why users cannot always obtain services of the quality they expect are either because of deliberate action (e.g. provision of corrupted resources) or are dependent on the current state of the infrastructure (e.g. network congestion, software failures). Systems where survivability is based on self-organization into communities could be a solution to the quality problem that is implemented by the peers themselves. In this thesis we propose that the deployment of an assessment schema based on a localized view of reputation where peers act by themselves, could help towards finding an optimum resource discovery policy. The deployment of an assessment schema is also expected to improve the general provision of services.

Even though there are various Peer-to-Peer models, in this thesis we consider only the unstructured one known as the "atomistic type" of P2P network because it is closest to the model of virtual communities which approximates best the way that real communities work. We chose this because, knowing the benefits and the pitfalls of networking architectures (centralized/distributed), our aim is to offer a solution that is closer to the fundamental principles of Peer-to-Peer technologies. These aim to offer a kind of communication as independent as possible from the use of centralized entities, approaching the way that human entities naturally communicate with each other.

In this work we intend to apply a reputation propagation mechanism to the system we propose. Reputation in general can be thought of as "*what we expect about an entity's behaviour based on observations and collected information from*

the environment regarding the past actions of the agent" [1]. In our computational model we attempt to apply a word of mouth scheme over the virtual community that is formed through the Trust relationships that we establish hypothetically between the members of that community.

In order to do this we examine the Trust relationships that may have been developed between members and derive from them a measure of reliability as seen from the point of view of the querying entities.

Similar systems have been running for years on the web as centralized services but they cannot give accurate answers to a satisfactory number of queries. What we aim to do with our work is to make predictable how satisfied users will be with the answers to queries so that they can be assisted in making the best choice. We leave any security issues that may arise for future research.

Our research covers several areas of technology:

- Recommender Systems
- Peer-to-Peer technologies
- Trust systems and Reputation

Peer-to-Peer systems exist fundamentally as decentralized services because they were built to work in this way. Trust systems exist as theoretical subjects in the computing literature and at the time of writing there is no solution that involves Trust applied to a real application. On the other hand, Recommender systems are well known and run as applications in centralized environments and so far they have had remarkable success.

The challenge is to combine Peer-to-Peer technologies which run decentralized, with the Recommender systems which currently run as centralized services on the web. As will be seen from the results of this work, the decentralization and the use of Trust we attempt in the recommender systems has positive effects on the P2P service itself because it produces better results than before.

1.2 Related Work

Many researchers have in the past worked towards the objective of improving search results for queries by organizing peers into groups of common interests [2]. This is based on the assumption that if a peer has a particular interest in a piece of content then it is likely to have items of interest to others as well. This concept is also the basis of Recommender Systems which are based mainly on Collaborative Filtering techniques. The main idea behind Collaborative filtering is that, if two entities agreed in the past about choices of items, then it is likely that they will agree again in the future.

In the existing solutions neither knowledge nor experience of a peer's behaviour that could be taken into consideration in any future choices is used. Observations of the past could shape future expectations and thus help in making successful decisions.

The need to exploit this information which otherwise goes wasted and unused leads us to the introduction of the notion of *Reputation*. The concept of reputation and its associated structures is not new and it has been formed by the way that people trade in human communities. In computing science reputation is defined as the common belief of a group of entities about another entity with reference to some certain activity.

This sense of common belief makes reputation seem to be a more objective characteristic than a subjective one. In centralized systems there are no difficulties as to how this objective characteristic will be assessed, but in decentralized environments like unstructured Peer-to-Peer Systems, reputation is difficult to maintain due to a lack of central entities where common beliefs can be held and managed. As it happens in real communities, reputation as common belief is shaped by the individual opinions that the members of a community hold. Those measures are mainly subjective and in the literature are known as *Trust relationships*.

1.3 Reputation in Computing

Most of the work in computing concerning reputation has been tailored to problems similar to those found in the static web. Even though the applicability of such methods is limited in the Peer-to-Peer area it is worthwhile to examine the reasons why.

The first approach to provision of evaluated search results on the web was using ordinary search engines, where ranking and relevance indicators in XML format were automatically attached to search responses [3]. This direction lead to a new way of providing data through the web that is now called "The Semantic Web". The novelty of the Semantic web was the extra semantic information together with the data which were used as an aid for the users in making their choices.

The work of assessment, and later the production of metadata was performed exclusively in centralized search engines (e.g. Google) which from then on were considered as trusted entities for the provision of that kind of information. In Google the weighting is determined by a number of factors such as the internal back-link index. This shows how many links from other web pages point to a page and every link is considered as a recommendation for someone to visit that particular page. Apart from the danger that this leads to ambiguous results since a highly referenced page does not necessarily mean that it is of good quality, it also has the danger that the central entity might provide biased recommendations. For example, if the central entity gets some benefit from giving a high rank to some web content it is difficult to find out if this is happening.

1.4 Trust and Reputation

Trust is a complex concept and is therefore not easy to define. Also the Trust literature can be quite confusing because the term is used with a variety of meanings [6].

A working definition inspired by McKnight & Chervany [6] is: "*Trust is the extent to which one party is willing to depend on something or somebody in a*

given situation with the feeling of relative security, even when negative consequences are possible".

As can be seen from this definition, Trust includes some basic concepts which are dependence, risk and uncertainty. Dependence is explicitly expressed whereas risk and uncertainty are expressed through the possibility of negative consequences. Uncertainty is an important characteristic which comes from the fact that there is always a lack of knowledge when making a decision.

The above definition includes the fact that non-living material or abstract things can also be trusted. The fact that different entities can have different kinds of Trust for the same target entity (trustee) indicates that Trust is subjective. Unlike Reputation which is objective, Trust is what each different participant in a community subjectively beliefs about the trustee entity and therefore we can say that individually expressed Trust is what shapes the Reputation of that entity. This is important because non-living material or anything abstract can be trusted although an item does not have the free will to behave honestly or dishonestly in the same way that living persons would. It is not always possible to distinguish between items and agents because the distinction between agents and non-living material can be fuzzy as happens in the case of automated systems. In general they can be seen as extensions of the humans they work for, who do have free will.

Stephen Marsh [20] defines Reputation as follows: "*Reputation is the amount of trust inspired by the particular person in a specific setting or domain of interest*". Reputation is conceived as a multidimensional value. Individuals may enjoy a high Reputation in one domain while they have a low Reputation in another. As with Trust, Reputation is context specific.

Li.Ding et.al. in [29] give another definition for reputation and they present Reputation as public Trust. As they say, "*Public Trust is based on the reported social experiences throughout agent society and it reflects the general opinion about individuals*" derived from secondhand evidence and, according to them, *Personal Trust* is derived from the agent's own social experiences (first hand evidence).

1.5 From centralization to Peer-to-Peer

In the early stages of the Internet the services and the connectivity was much like a Peer-to-Peer infrastructure. Universities in the United States were connected together for the purpose of exchanging scientific information. The Peer-to-Peer type of organization was at the machine connectivity level. Services from the early days such as Email, which still runs as a Peer-to-Peer service, demonstrates the architectural ideas on which it was based.

The increasing load on what became the Internet, the explosive growth of its user base and the emergence of its new face, the Web, were all factors that drove development away from the early peer network to a more hierarchical structure.

The resulting trend was towards the client-server model where the great majority of users were seen as passive recipients of static, server-stored information.

Parallel to this divisive trend, a trend emerged in Internet connectivity for direct data transfers over high bandwidth connections between top-level servers, further distancing the Internet from its Peer-to-Peer roots. The recent development of Internet2, a separate higher-capacity backbone network has added a further dimension of separation to the basic connection structure.

Much of the motivation behind various Internet peer technologies can be seen as a reaction away from server centric content and passive clients, back to free exchange between individually combined client-server nodes in the network. The fact that data and resources are controlled by the users and not an external authority is characteristic of, and motivation for, many of the current Peer-to-Peer technologies. The success of Email, which was the killer application for the early Internet, confirms the assumption that what people always want to do, whatever the technology, is to communicate with each other. It is an open question now if e-mail should be considered as peer-to-peer technology. The original Peer-to-Peer functionality moved on into another technology that is now called IM which is used for chatting. Today's chat and IM are perceived as Peer-to-Peer in the same way that conversations on the telephone are. Chat technology resolves the individual addressing issue by maintaining a centralized directory to correlate a registered user with an on line Internet address. As can be seen from this example

there is either a total or partial dependence on central entities for Peer-to-Peer services that are running over the Internet infrastructure.

1.6 Architectural Models of Peer-to-Peer infrastructures

A good characterization of Peer-to-Peer models is based on the amount of dependence on centralized entities they require. The models differ from each other in the ways that searching for information takes place. [11] contains the following categorization.

1.6.1 Unstructured model

In the *Unstructured* or *Atomistic* model of P2P networks there is no central administration, all nodes are both server and client and each is completely autonomous, managing its own resources and connectivity itself. This offers maximum availability, stability, robustness and persistence of data but it suffers from difficulties in node discovery. A good example of this type of network is the Gnutella P2P network. [75]

1.6.2 User Centric model

The *User Centric* model is quite similar to the previous one but there is central server mediation. In the simplest form a directory server is used to simplify the way that nodes find each other. The server can be thought of as a centralized registry. Nodes are required to register with this directory before they and their contents become available to the rest of the community. However the existence of the central entity raises privacy and vulnerability issues.

1.6.3 Data Centric model

The *Data Centric* model is identical to User Centric with the difference that the central entity maintains an index of available resources and not users. An example of this type of network was Napster.

The original vision of the World Wide Web by its "creator" Tim Berners-Lee and others was a data centric Peer-to-Peer network of globally hyperlinked content space where no single server had precedence over any other. However much to the disappointment of the first visionaries the Web instead evolved to have mostly static content and be server-centric. With users constrained to the role of passive consumers, and little peer communication between them, functional development of Web browsers focused mainly on fancy presentation features. Nevertheless, the importance and use of open Peer-to-Peer models has returned on a new level, user-to-user rather than machine-to-machine, making the developmental chronology implied by the previous table reasonably accurate.

In our study we will be dealing with the first model, Unstructured or Atomistic as it is closest to the way that human entities communicate with each other in the contemporary communities.

1.7 The challenges

The basic problem related to reputation management in Peer-to-Peer networks is that information about transactions performed between peers is dispersed throughout the network so that a peer can only have an approximate view of the global state of the network.

Another problem that makes the situation even more complicated is that the peers that hold and process the trust-related information cannot be considered as totally trustworthy themselves and nodes must take into account the possibility that those peers might be behaving maliciously.

Due to the dynamic nature of Peer-to-Peer communities new rules have been put in place and they obey the application of new regulations for the handling of any kind of such information that is to be collected. In the case of reputation ranking information, the problem is that such information becomes stale far more quickly than in ordinary client-server systems whose operation and services are long-lived.

Furthermore the storage of large amounts of critical information in the peers themselves, might not be a suitable solution because:

- With the databases running on ordinary peers without using a replication scheme, the availability of the data cannot be guaranteed since peers join and leave at any time and without prior warning.
- It is likely that the nodes and the replication mechanism will not be able to cope with the high levels of traffic that the database requests will create.
- The idea does not conform with the self-structured philosophy of the Peer-to-Peer networks.

1.8 Word of mouth

The schema we intended to apply in our work for the dispersion of the information has an analogy in the physical world: "*word-of-mouth*". Wikipedia [42] defines "*word- of-mouth*" as:

> "*The passing of information by verbal means, especially recommendations, but also general information, in an informal, person-to-person manner, rather than by mass media, advertising, organized publication, or traditional marketing. Word of mouth is typically considered a spoken communication, although web dialogue, such as blogs, message boards and emails are often now included in the definition*".

Word-of-mouth promotion is highly valued by marketeers. It is felt that this form of communication gives valuable source credibility. People are more inclined to believe word-of-mouth promotion than more formal forms of promotion because the communicator is unlikely to have an ulterior motive, i.e. they are not out to sell you something (for evidence of the conditions under which word-of-mouth communication is effective, see R.Grewal et al. [43]). Moreover, people tend to believe people that they know. In order to manufacture word-of-mouth communications, marketers use a wide variety of publicity techniques.

There is some overlap in meaning between word-of-mouth and the following: rumour, gossip, innuendo, and hearsay; however the negative connotations of these words are not usually part of the meaning of word-of-mouth.

A useful definition we can give for word-of-mouth is:
"*A social reputation dispersion mechanism which has its basis in the human way of communicating data and distributing opinions between people. Using word of mouth techniques in computing for the diffusion of the information in the same way as in the human communities can be more effective and less costly in resources*".

Both this definition and that of Wikipedia exclude the use of mass media and traditional marketing as a way of promoting the services to be published. The analogy with the computer information world is that mass media symbolize the central entities or web sites of high publicity, and traditional marketing the client server protocols.

1.9 Recommender Systems

There are many Recommender systems [11] used in on-line systems today. They help people make choices when they do not have sufficient personal experience of the options from which they have to choose. In these systems we recognize two types of users: the recommenders and those who seek recommendations. Users can either provide anonymous or pseudo-anonymous recommendations and also encounter the problem of "*free riding*" which comes from content sharing in Peer-to-Peer applications, as well as the incentive and privacy problems as found in the peer preview system used in academia. "Free Riders" or "leeches" are those peers which take part in a Peer-to-Peer scheme without sharing any files or information.

The best known types of Recommender systems have as their basis Collaborative Filtering [12] which helps humans make choices based on the opinion of other people. The simplest form of this is "*word-of-mouth*". As we saw, word-of-mouth is also known as "*a system for propagating reputations*" [1]. Ways of automating the word-of-mouth technique can be found in [13]. This work attempted to build a personalized music Recommender system using Social Information filtering, (i.e the word-of-mouth technique), by organizing users in

communities of similar interests. The system was called "Ringo" and compared user profiles to find which users had similar tastes.

In simple terms, in the word of mouth model a query about some other entity *B* is sent by a human *A* to her neighbours and it propagates (up to a predefined distance *h)* until a neighbour that knows *B* is reached. Then *B*'s opinion is sent back to *A* to help in decision making.

This way of forming decisions requires that opinions can be propagated from people to their neighbours to a distance *h* or at least that the effects of propagation on the transferred message are known. (e.g. unsuccessful attempts)

We can distinguish two challenges for Recommender systems: the time the algorithms take to respond and the quality of the received results. In some ways these are in conflict since the less time an algorithm spends searching for neighbours the more scalable it will be and the worse its quality. For this reason it is better to attack these two challenges simultaneously so that the system can be both useful and practical.

Another less important challenge is to build Recommender systems which are resistant to attacks. This vulnerability comes from the centralized nature of existing solutions where all ratings are available to the participants and thus making it possible to know everyone's tastes and the way that people express their opinions. We intend to move towards Recommender systems which will be less vulnerable to attack by making them distributed with correlations performed by third parties and not by the central system itself.

In the next chapter we present a more detailed analysis of the Collaborative Filtering technique which is the best known and most important of the other methods.

1.10 Trust in computing

There is a whole range of challenges not met by traditional security approaches which will typically protect resources from malicious users by restricting access only to authorized users through such devices as encryption, authentication and access control. However in many situations we have to protect ourselves from those who offer resources, so the problem is reversed. Information providers can

act dishonestly by providing false or misleading information and traditional security mechanisms are unable to provide protection against this type of threat. In the case where information providers are anonymous peers the situation becomes even worse because the peers' identity is hidden by the protocol. This happens because peers have no incentive to act for the benefit of the community.

On the other hand, Trust and Reputation can provide protection again such threats. The problem faced by potential collaboration partners is basically that of making decisions involving risk and uncertainty. Trust is a catalyst for cooperation because it allows entities to interact spontaneously and efficiently.

The ability to gain the Trust of others is an important criterion for the success and survival of an entity because it makes others willing to collaborate with it. The safest and most often used strategy is simply acting in a responsible and trustworthy manner. The ability to assess correctly the trustworthiness of a target is therefore an equally important criterion for the performance, success or survival of an entity.

Computer networks are increasingly removing us from direct interaction. We may now collaborate in Peer-to-Peer networks with people and organizations we have never met and perhaps have never heard of before. Many of the traditional strategies e.g. access control systems, for representing and accessing trustworthiness can no longer be used in such situations. It can therefore be difficult to assess whether the services and information provided by remote parties are reliable or even whether they are correctly represented.

Thus there is a need for mechanisms that enable parties to determine the trustworthiness of remote entities through computer mediated communication and collaboration. These mechanisms should recognize trustworthy entities as such. The idea behind this is that such Trust and Reputation systems will enable highly trustworthy entities to attract collaboration from others and discourage fraudulent users from participating in the community.

1.11 Uncertain probabilities theory

There is a theory called Dempster-Shaffer theory [7], otherwise known as the theory of uncertain probabilities, which allows us to allocate probability like

weights to a set of events in a way that allows statements of ignorance about the likelihood of some of events. In our approach we have made use of the basic principles of this theory and its successor, Subjective Logic. More information about this theory can be found in chapter 3.

One of the weaknesses of the Dempster – Shaffer theory is that it cannot produce good results in situations like ours which are characterized by high uncertainty and the demand for short response times that a real time system requires. Therefore we use Subjective Logic to avoid this problem. Subjective Logic can be thought as extension to Shafferian Theory which is based more on classical probability theory.

Another problem is the construction of the opinions that will be used in the Subjective Logic algebra. This is because the values have to be given in a special form expressed as a triplet of *belief*, *disbelief* and *uncertainty*. In this thesis we have used our own method of forming opinions so that they can be used by the algebra of Subjective Logic. Another way of producing triplets of data from evidence is to use the Beta distribution function [26] but such a method requires evidence to be in a predefined format which is not suitable for our case scenario.

1.12 Assumptions

We assume that the peers involved in the infrastructure being examined are not behaving maliciously and that the Trust values they provide in response to the queries are those derived from the application of the formulae we use. We also assume that there is no bias in the expressed opinions.

1.13 Hypothesis

In this thesis we hypothesize that there is a connection between Trust and Similarity. In other words, users which seem to have a similarity in the choices they have made are likely to Trust each other rather more than if they were dissimilar.

In other words, the logical hypothesis we make is that if we can assume that two entities are likely to Trust each other's ratings, in the case where they have

previously provided similar ratings for similar services, then we can use this Trust to allow recommendations to be propagated transitively through the network. This will allow entities to obtain recommendations from greater distances in a Trust network.

1.14 Thesis Contributions

This thesis introduces a set of contributions that addresses the problems of Recommender systems and intends to improve the quality of the services provided.

More analytically it provides:

- An overview of a Trust model that uses the operators of Subjective Logic and how this could be used in a Recommender system.
- A model for deriving Primary Trust from evidence or from qualitative and quantitative information about users' preferences. The novelty in this model is that Trust is distinguished from Similarity.
- Validation of a distributed, Trust-enabled Recommender system which can run in existing Peer-to-Peer communities as a consultation service.
- A performance analysis which shows the sizes of communities in which such a solution can be applied.

1.15 Dissertation Outline

The work in this thesis is structured in two parts: In the first part we present a study of what can be achieved in terms of the benefits we can have if exploiting the Trust relationships in a transitive way. In this study we have used a set of formulae to encode Trust relationships in a form that is convenient for applying the theory of uncertain probabilities and then we used Subjective Logic operators to perform the analysis of the resulting Trust graphs. This logic was found useful because it provides a framework for computing Trust properties in long chains.

The second part of the thesis studies the performance of such systems and focuses on scalability issues. In this study we examine systems of various sizes and we make conclusions about the applicability of the method..

Chapter 2 gives a brief overview of some models of Reputation as well as to other existing solutions that run as consultation services on the web today. We focus more on the problems of these Reputation systems but also describe their Trust models, what problems they try to solve, and the circumstances under which they operate in today's info-sphere.

We also describe the fundamental operation of Recommender systems, presenting the mathematical formulae behind them and the functions they provide in the environments they are applied to.

Chapter 3 discusses Trust systems for computational environments as well as focussing on Trust models and calculus. Specifically, it presents the essential knowledge that is required to understand the Trust model we use. We also refer to the other Trust properties as they can be found in the bibliography. Finally there is an introduction to Subjective Logic and its algebra that we use in our Trust calculations experiments.

In *Chapter 4* there is an outline of our architecture presenting the steps that have to be followed in order to build a distributed Recommender system. Each individual phase that has to be carried out to have a workable solution is analyzed. There is also a comparison with the centralized architecture focusing on the benefits of the distribution.

In *Chapter 5* we present the Trust model we use in our project and analyze the way that pure evidence is converted to Trust values. We also evaluate our method by comparing its results against an existing mapping model based on the Beta distribution function. For the evaluation we use an existing sample dataset of evidence which we convert to opinions using both methods. The creation of a new mapping was necessary because no behavioural data were available in the format that Subjective Logic can use.

In *Chapter 6* we apply our Trust model to an existing data set of evidence in a hypothetical Recommender system and in this way we build a web-of-trust. We apply two Subjective Logic operators to calculate the Trust in transitive chains. We finally measure the benefits of our trust-enabled model against a typical Recommender system that makes no use of Trust and where the decisions are made intuitively.

Chapter 7 contains a performance analysis which shows the applicability of our trust-enabled community in a real Peer-to-Peer file exchange application. In this chapter we test our protocol's behaviour in terms of traffic that the protocol produces in the environment on which it runs. We have included tests for various sizes of agent communities and we intend to find the community sizes for which the Recommender system can provide responses within an acceptable time limit.

In conclusion, *Chapter 8* provides a general discussion of the success of our approach in meeting the requirements we set above and also discusses improvements of the model that have been left as future work.

Chapter 2

REVIEW OF RELATED WORK

2.1 Introduction

In this chapter we define and describe reputation systems and look at some of the problems that such systems have. We also analyze related work that has been done in our areas of interest, including Reputation systems in general, particularly in the Peer-to-Peer world. We briefly examine the best known problems of Reputation systems and present relevant, older solutions such as UDDI. In this chapter Recommender systems are distinguished from Reputation Systems. In general Reputation Systems are used to assist agents in choosing reliable participants to interact with when some information about the Trustworthiness of the agent is offered. On the other hand Recommender Systems provide assistance to people in making choices when their personal experience is inadequate. Therefore Recommender Systems can be considered as a special class of Reputation Systems because they provide Recommendations by employing Reputation mechanisms in their internal operations. The main focus of our work lies in how Recommender systems work. This review is no by means exhaustive, for a more complete survey see [45].

2.2 Reputation Systems

Systems established in electronic environments for building Trust between members are known as "Reputation Systems". In general a Reputation system is used to assist agents in choosing reliable peers to interact with, when some information about the trustworthiness of an agent, a peer or a resource in general is offered.

In Reputation systems feedback is collected from members of a community regarding past transactions with other members of that same community. This feedback is then analyzed and made publicly available to the whole community in the form of feedback profiles of its members. If some member who is seeking assistance accepts the past behaviour as a relatively reliable predictor of future behaviour then

such profiles can act as a control mechanism essentially used as the digital equivalent of a person's reputation.

One of the problems of reputation systems, is that negative feedback is rarely given by participants because they fear retaliation. So participants tend to skip the feedback phase although the effort required to complete it is small. Furthermore it is difficult to preserve privacy and anonymity and at the same time provide enough information so that the entity in question can be successfully identified by other peers [55].

Rahman and Hailes [1] proposed a Trust model that can be applied in a Peer-to-Peer network based on Marsh's work [20]. Rahman's model is somewhat simplified compared with Marsh's proposed model and Trust can have only 4 possible values. The goals of this work were to assist users in finding trustworthy entities and providing artificial autonomous entities to reason about Trust. The model is inspired by work that has been done in the area of the social sciences and it is based on real-world characteristics of Trust. This model identifies three types of Trust: Interpersonal, System and Dispositional. Interpersonal Trust is context specific and is the Trust that one agent has directly for another. The second type, System Trust, is not based on the state of the trustee but on the reliability of the system to which the examined entity belongs. The third type of Trust is called Dispositional Trust or 'basic Trust' and describes the general attitude toward oneself and the world. This type of Trust is independent of any context. McKnight et. al. [39] define further subtypes of Dispositional Trust.

The vulnerability of this approach is in the fact that every agent in the community has to be quite complex and include large data structures that represent global knowledge about the whole network. In real situations this might be a time-consuming task. It is also unclear if the model scales for hundred of thousands of nodes. Another important weakness that must be considered if the system is used in Peer-to-Peer systems is the fact that during bootstrapping a new agent faces a high degree of uncertainty about other agents it may find. As a result, it will be unable to distinguish between trustworthy and untrustworthy agents and that makes it vulnerable to manipulation, as happens to any newcomer to a community. To reduce uncertainty and risk it is recommended that new agents are equipped with a number of trusted entities so that initial interactions can be made with trusted parties only, or with those recommended by already trusted recommenders.

Another model that has its basis in gossiping techniques is that of Yu and Singh. [38]. Gossiping is the action of spreading news from entity to entity in a network especially by rumours or private information. The term also carries implications that the news so transmitted has a personal nature. In Computing Science *gossiping* is defined as "*a mechanism for scheduling communication in a network in which individuals exchange information periodically according to a fixed schedule*". For more, see [64][67]. In Yu's and Singh's model, a Social network amongst agents is built that supports participants' Reputation. Every agent keeps a list of its neighbours which is updated and computes the trustworthiness of other agents by testimonies received from reliable referral chains. After a bad experience with some other agent, a bad rating is propagated to the rest of the neighbours, so that the other agents update their ratings accordingly. As in the previous model it is unknown whether these communities can scale to large numbers of users or not.

N.Mezzeti presents a Socially Inspired Reputation model in [44] aiming to develop a model that allows an entity to predict whether another entity will exhibit dependable behaviour or not, based on the behaviour that the same entity exhibited in the past. It defines Trust as a measure of how much reliance an entity can place on the dependability of another's behaviour within a specific context. This model provides amongst others things a property called transitivity which is used to represent trustworthiness and competence in recommending in a specific context. It also refers to the measure of strength as "*Trust degree*" with values between [0,1].

With regard to transitivity, the model defines a rule which says that the trustor should be prevented from trusting a given trustee more than either the Trust he places on the recommender or the Trust the recommender places on the trustee. This leads to a formula in which the Trust to the trustee (Cecilia) is the product between the Trust in the intermediate entities:

E.g. $T(Alice, Cecilia, o) = T(Alice, Bob, j(o)) \cdot T(Bob, Cecilia, o)$ where,

o is the context and T symbolises the Trust relation between two entities. $j(o)$ is the mathematical representation of the jurisdiction sub-context associated with o, given a context o and it is represented in mathematical notation by $j(o)$. An entity controlling

the context *o* is trusted to provide reliable Trust information about trustees within context *o*. The jurisdiction sub-context was introduced in order to allow transitivity in the Trust relationships.

This model also introduces time into the Trust relationship which decays with some rate *n* as time passes. Wrong or unexpected interactions have an effect on the Trust value and thus the Trust degree.

Even though the model takes into consideration time and Trust transitivity, it does not discuss how a parallel combination of Trust relations would affect derived Trust. For example what would be the consequences on Alice's Trust of Cecilia if there were more than one recommender (e.g. Bob and David) who could offer some experience about trusting Cecilia? This characteristic makes the model unsuitable for building and deriving Trust in a "web-of-trust" scheme where, in principle, when an entity is well trusted by more than one trustor then the derived Trust is stronger.

Stakhanova et al. [46] present a Reputation system which is specially designed for the Gnutella Peer-to-Peer network. The proposed solution does not employ centralized storage and what is presented in this paper is a policy for managing traffic in a Peer-to-Peer network based on peers' Reputation. In this solution each peer monitors the activity of its connected peers and makes Trust decisions based on the individual thresholds they set for good and bad actions. Traffic from a node with a bad Reputation is not accepted by those who have characterized it as a bad node. Such a system enables each node to have its own personal view of every other neighbouring node and is based on good and bad actions as characterised by the users themselves.

Even though the figures show that Reputations follow the behaviour of each node according to how it behaves, the system requires human intervention via a GUI for people to rate and characterize the peers as good or bad. Note that, a node that has been characterized as bad might be useful for forwarding the queries and pings of the Gnutella protocol to other peers no matter how it has been rated. The policy has not been tested in large networks to know for sure what the consequences are of applying these policies at a large scale.

NICE [47] is a distributed, reputation-based, approach for Trust management designed as a platform for implementing co-operative applications over the Internet.

NICE is based on a distributed scheme where user Reputation information is stored in the form of cookies (signed statement) expressing peer satisfaction about transactions. Before initiating a transaction a peer checks a local database to ensure that a targeted peer can be trusted. One of the aims of this work is to attack the problem of *"free riders"*, who only use services from the system without offering any resources in return, by revealing the free riders in the Peer-to-Peer community. NICE uses a signalling system which communicates messages using a multicast protocol. The goal of the default policies in NICE is to limit the resources that can be consumed by cliques of malicious users and, via the Trust computation, to identify misbehaving nodes. The main idea is to build a web-of-trust based on the cookies that are communicated between the nodes-entities. For each transaction in the system each user involved produces a signed statement (cookie) about the quality of the transaction. For example, Alice signs a cookie stating that she has successfully completed a transaction with Bob. Bob may store the cookie and use it later to prove his trustworthiness to other users including Alice. We can describe the existence of Trust in terms of a directed graph called a trust-graph, with vertices representing the users in the system and the edges denoting that a cookie is held on one vertex (Alice) about another user-vertex (Bob). The set of Alice's cookies that Bob holds denotes how much Alice trusts Bob. In the case that prior to a transaction there is no direct link between the entities who wish to interact; an indirect link through others can give an estimate of the target's trustworthiness.

However, if no cookie is available for the peer in question, co-operation with other peers can gather the necessary information to go on. The method does not provide a clear solution to the case where there are multiple paths that lead from the origin node of the Trust graph to the destination, where in the best case it uses a weighted sum of the strongest paths. The suggested scheme has to be tested to see if it scales for infrastructures that include a large number of nodes. As regards the assignment of values to cookies, in many cases it is unclear how to assign real-valued quality metrics to transactions. The NICE approach also requires the cooperation of peers in Reputation calculation, but such cooperation might not always be available, for example in cases where there is a conspiracy between malicious users.

In the approach presented by Gupta et al. [48] there is a debit-credit mechanism which is used for the computation of reputations which credits peer Reputation scores

for serving content and debits for downloading. *"Behaviour"* and *"Capabilities"* are the factors used in the calculation of the Reputation score of each node in the system. This solution is an example of a centralized solution with respect to where data are stored and where decisions are made. The proposed model tracks positive peers' contribution to the system using a credit–debit mechanism. In the system each peer computes and stores its Reputation as derived by its own formula locally. For ensuring secure and distributed access to Reputation scores a *"Reputation Computation Agent"* periodically collects the Reputations from the peers using a key pair. Unfortunately, this approach does not provide mechanisms for decreasing the Reputation of users behaving maliciously.

PGP (Pretty Good Privacy) is another distributed Trust model used for proving the identity of key holders. It makes use of user defined thresholds to decide whether a given key should be trusted or not. Unlike NICE described above, PGP allows only one level of interaction. For example if entity A is trying to decide the trustworthiness of an entity B then there can be at most only one entity between A and B since there is no transitivity in PGP. In [25], however, there is a method for assessing Trust in certification chains.

2.3 UDDI: an Existing Consultation Service

UDDI is a primitive form of consultation system that exists on the Web running as a meta-service for locating Web Services by enabling robust queries against reach metadata. It was used as an industry specification standard for building flexible, interoperable, XML Web Service registries useful in private as well as public deployments. UDDI stands for "Universal Description Discovery and Integration" protocol and creates a standard interoperable platform that enables companies and applications quickly, easily, and dynamically to find and use Web Services over the Internet. UDDI also allows operational registries to be maintained for many purposes in different contexts. UDDI is a cross-industry effort by software as well as marketplace operators and e-business leaders within the OASIS standards consortium [37]. It was built for Business-to-Business collaborations and operates as a yellow pages service where queries about services can be answered. UDDI is a good example

of a discovery service but it operates as a protocol in which the requirements of a query will be described in some pre-defined format in such a way that will be understood by other counterparts that have adopted the UDDI protocol.

A business may deploy one or more private and/or public UDDI registries. A private registry permits access only to authorized users. A public registry does not restrict access to its registry. A business may choose to deploy multiple registries in order to segregate internal and external service information. An internal registry supports intranet applications, while an external registry supports extranet applications. Industry groups may deploy a UDDI registry to support public or private exchanges.

In terms of Reputation, in the UDDI scheme the reputation management responsibility is given to distinct centralized entities which act as agencies that receive advertisements from providers. Potential partners can locate them directly by querying using the UDDI language. This scheme even though it is a working solution that looks like a Peer-to-Peer scheme, does not have the flexibility and robustness of pure unstructured Peer-to-Peer systems where the registries can be any node in the community.

In table 1 there is a summary of the Reputation systems we examined with their advantages and disadvantages.

Method	In favour	Against	Scalability
Rahman-Heiles	Inspired by work that has been done in the area of social sciences and based on real world characteristics of Trust	Every agent in the Community needs to be complex and include large data structures	Unknown
Yu Sigh	Based on Gossiping techniques and propagates bad experiences which does not consume many resources		Unknown
Mezzati	Supports transitivity in Trust	It does not describe how parallel combination of Trust relations would affect the derived Trust therefore is unsuitable for building webs-of-trust	Unknown
Stackhanova	Does not employ a centralized storage' a pure Distributed Solution.	Requires human intervention via a GUI for people to characterize peers as good or bad.	Unknown
NICE	Aims to attack the problem of free-riders. Supports web-of-trust.	No clear solution in the case where there are multiple paths between the origin and the target.	Multicast Yes
Gupta	Secure via the use of a key pairs for the calculation of Reputations	Centralized approach in regard to where data are saved & decisions are made.	Unknown
UDDI	Standard adopted by the Industry	It does not have the flexibility & the robustness of pure unstructured Peer-to-Peer systems where the registries can be at any node in the community.	Yes

Table 1. Summary of Reputation Systems in General

2.4 Reputation systems in the Peer-to-Peer world.

These can be considered as a special category of Recommender systems (see chapter 1). They are Reputation systems because they provide Recommendations by employing Reputation mechanisms in their internal operations. In the bibliography they can be found under both descriptions. To avoid confusion, in the following chapters the term Recommender systems will be used instead of Reputation systems.

One of the main goals of a Reputation system for a Peer-to-Peer network is to reduce the chances of a peer being cheated in a transaction. The general idea behind Peer-to-Peer Reputation systems is that Peers evaluate each other's Reputation information in a process based on mutual interaction. In some studies it has been suggested that decentralization is recommended in reputation systems because it significantly reduces the number of malicious transactions [56][57]. This happens for a number of reasons:

- The Reputation system itself rewards those peers that co-operate well with others
- It punishes those peers that cheat or try to do malicious things
- It motivates network peers to cooperate with each other

The cost of operation, which in the case of centralized systems has to be covered by the central entity where the service runs, is an important factor for a Reputation system. In the case of Peer-to-Peer Recommender systems the service is supported by the participants themselves and so there is no problem. Thus, theoretically, there is no significant cost for running such a service because it is shared amongst the participants.

It has been shown [61][62] that various problems and attacks can compromise a Reputation system. For example, systems in which easy change of identity is allowed have been shown to be prone to malicious behaviour of peers. Also, it is difficult to evaluate the integrity and reliability of the Reputation information in the case that it is stored on the peer's computer, because it can be easily falsified.

The weakness of the various proposed solutions is that they are specially tailored for particular protocols. E.Damiani et.al. [4] proposed a mechanism tailored to the Gnutella algorithm, which makes reliability checks on candidate participants prior to downloading by sending polls directly to all neighbouring peers via a Reputation exchange protocol called XRep. The purpose of this protocol is the collection of votes from those peers that have download material from the peer in question in the past. After all the information from the polls has been collected, it is validated by another poll and finally is used by the querying peer to reach a decision. Peers maintain local repositories of opinions about other peers they have interacted with in the past. The criteria for such updates are, however, subjective.

Even though there are no evaluation results available nor any performance analysis of the level of improvement that the algorithm offers in retrieval operations, the polling mechanism itself affects the scalability of the Gnutella protocol due to the extra messages that need to be sent. A rough estimate shows that the resulting traffic is a threefold increase.

Although this work addresses many security considerations for both Peer-to-Peer networks and Reputation systems, it offers no incentive to the peers to participate in the XRep scheme.

PRIDE [58] is another Reputation system also designed for the Gnutella network and requires small modifications to be made to the standard protocol. This protocol needs no central server to identify peers and store information. Peers create their own identities using self-certification and store any recommendations they receive from requesters locally. The key operations in the protocol are the creation of a Recommendation after a content download and the creation of a transaction number for the provider. The transaction number is signed and stored by the requester after a download. Both the signed transaction and the Recommendation are stored locally by the provider. The provider shows this information to the next requester as a proof of its Reputation to persuade him/her to proceed with a transaction. Thus, there is no need for a Reputation search by the requester as happens in the XRep protocol described above.

In the protocol every peer runs its own Certificate Authority (CA), which signs its identity certificate. This certificate specifies the IP address range within which the identity can be used. If the identity certificate is used by an address outside the range

then the transaction is aborted. IP addresses are not used as identities because they change and sometimes are not managed by the peers themselves.

The problem with self certification is that any peer can generate a large number of identities and maliciously increase their Reputation by giving false Recommendations; such a farm of identities is called a '*liar farm*'. In the case that self-certification is used, a peer's identities (Liar farm) cannot be mapped back to it without its consent. By the use of security zones that are subsets of the IP space it is assumed that only the information provider receives the Recommendation and the requester provides it. In this scheme each peer maintains a local database of verified identities. If the requester cannot find a valid identity in its local database then it verifies the identity by performing a cryptographic challenge-response to the IP address in the identity certificate of the previous requesters.

Once a requester has received a list of valid Recommendations it sorts them by IP address, determines a security distance d and divides the linear IP space into slices of length d. Then it calculates the Reputation of the information provider identity. The method is based on the fundamental assumption that it will be difficult for a malicious peer to generate identities that have totally non-contiguous IP addresses (identity farm). By increasing d a requester can reduce the probability of an information provider having an identity farm.

EigenRep [49] is an attempt to build a Reputation management system for Peer-to-Peer networks. In it each peer stores (locally) its own view of the peers with which it has committed transactions in the past. A global Reputation value is computed by using the local Reputation values assigned by other peers but weighted properly by the global Reputations of the assigned peers. The main source for the Reputation value is the peer's history of uploads.

This algorithm is intended to decrease the number of downloads of files which are inauthentic in a Peer-to-Peer file sharing network like Gnutella, and the method is based on Power iteration [79]. (Power iteration is the most straightforward technique for computing the principle eigenvector of a matrix). The global Reputation value is used by peers to choose from whom to download and the network effectively identifies malicious peers and isolates them.

The problem with this method is that it cannot protect the system from organised groups of malicious users who will give high marks to each other to acquire good Reputations.

PeerTrust [50] is another Trust management system for measuring and comparing the Reputation of peers in P2P networks, based on feedback from others. It has two main features:

First, it introduces three basic Trust parameters and two adaptive factors for the computation of the trustworthiness of peers. These are:

1) The amount of satisfaction received by the other peers in the community.
2) The total number of transactions a peer has committed to.
3) Some balancing factor to offset the impact from malicious peers misreporting other peers' service.

Second, it introduces a general Trust metric which combines the above parameters. The Trust value for a peer as it is computed in this system is subjective and dependent on the above three factors.

Each peer maintains a small database which stores a portion of the global Trust data. This solution is much like the storage system of the P-Grid database Reputation system, but PeerTrust requires co-operation from the peers for storing the Reputations. To avoid users acting maliciously there are multiple copies of data stored in the databases. Trust is computed on the fly by querying multiple databases across the network.

Micro-payments [51] can also be considered as elementary Reputation mechanisms. They comprise a number of digital payment mechanisms which are used to track the contribution of each of the participants in a network. The credits obtained are proportional to each one's contribution and in that way it works as an incentive to increase resource contribution. In such a model, Reputation profiles are developed for each of the peers depending on the number of credits each peer manages to collect during trading operations. Even though this policy can have positive effects in persuading users to exhibit good behaviour, it is solely based on subjective criteria. In spite of that, we mention it here as a good example of a basic distributed Reputation system.

The Histos system [10] approaches the challenge of sharing Reputation in highly connected online communities by using directed graphs for the representation of the relationships between the users. This characteristic gives a sense of personalization to the system. The system was inspired by the *Friend of a Friend Finder* scheme [54] where it was thought that everyone within a community could be known to each other through their relationships with their friends. Any relationships established between entities could be based on the commonalities found in their sets of interests. Histos is an attempt to develop methods by which we can automate the social mechanisms of reputation for the purposes of an electronic marketplace.

In a typical situation the nodes in the directed graph represent users and the weighted edges the Reputation ratings that one user gives another. The reason why a graph scheme has been used is because it gives higher value to inter-personal communication. This means, a node's idea of another, if there is no direct relationship between them, depends on the point of view. Thus, all the intermediates shape the reputation value of the examined node and therefore the calculated value is dependent on these intermediate nodes and especially on how well they know each other. Even though Histos is a promising solution, it does not make obvious how the relationships between nodes should be created or under what circumstances and semantic constraints. It is also not clear if transitive relationships can be supported over long chains of nodes. The formula that is used in Histos for the calculation of Reputation is based on the assumption that the Trust that is built from the Reputation relationships is absolutely transitive. As will be seen in the next chapter, there are certain restrictions derived from the properties of Trust under which transitivity can be considered. In other words, direct and indirect Trust relationships are treated in the same way by this algorithm.

Another interesting piece of work in the area of Peer-to-Peer Recommender systems is that of Kinatender and Rothermel [68] which presents the algorithms and the architecture of a Distributed Recommender system. The work uses a Trust over-net to establish Trust between individual entities but is preliminary and has not been applied in a real environment. Therefore no set of results is provided.

In table 2 there is a summary of the P2P Reputation Systems systems we examined with their advantages and disadvantages.

Method	In favour	Against	Scalability
Damiani et al.	Addresses many security considerations for both P2P networks and Reputation systems.	It is based on a polling mechanism which comes from a centralized idea.	Not scalable. Affects the scalability of Gnutella also.
PRIDE	Recommendations are created after a content download. So there is no reason for Reputation search by the requester which would be very consuming in resources.	The problem of self certification is that any peer can generate a large number of identities and maliciously increase their Reputations by giving false Recommendations.	Yes
EigenRep	Is intended to decrease the number of downloads of content that is inauthentic.	It cannot protect the system from organized groups of malicious users which will give high marks to each other to acquire good Reputations.	Unknown
PeerTrust	It regards the amount of satisfaction received by the other peers in the community.	Requires cooperation from peers for storing Reputations in the P-Grid database.	Unknown
Micropayments	A good example of a distributed Reputation system.	A solution solely based on subjective criteria. It comprises very basic operations.	Yes
Histos	The Reputation is dependent on the point of view which makes it more realistic since it uses theory from social networks.	It does not make clear how the relationships between the nodes should be created or under what circumstances and semantic constraints.	Unknown
Kinatender et al.	Pure distributed solution.	Not applied in a real environment so far.	Yes

Table 2. Peer-to-Peer Reputation Systems

2.5 Centralized Reputation Systems

This is the best known type of Reputation system and runs embedded in a centralized service. The Recommender systems we mentioned in the introduction have been studied from the perspective of the electronic commerce. Recommender systems often exist as services embedded into web sites therefore their use is completely transparent to the end user. They provide support for e-commerce activities. www.epinions.com[16], www.amazon.com[17] and www.ebay.com[18] are some of the most popular sites that use them.

In the systems we referred to in the first chapter, data analysis techniques are applied to solve the problem of helping customers to find the products they would like to purchase through E-Commerce sites. In a sense, Recommender systems are a type of application which belongs to the category of Knowledge Discovery in Databases [53] (KDD). For instance companies are using KDD to save money by discovering which products sell well in some periods of the year and in this way manage their inventories. KDD is also used by companies to reduce the cost of direct mail by discovering which customers will be more interested in particularly special offers.

How effective a Recommender system is can be measured in terms of its success in predicting consumer preferences.

One simple approach to building a Reputation mechanism is to have a central agency that keeps records of the recent activity of the users in the system in the same way that the scoring systems of credit history agencies work [27]. Then an algorithm is used to correlate the scorings and make predictions about future choices.

The best known technique is Collaborative Filtering [63] [5] [14]. This is used to detect patterns among the opinions of various users and utilize them in making recommendations based on the personal judgment of other members who happen to have similar tastes. It employs statistical techniques to find a set of customers which can be seen virtually as *neighbours* and also have a history of agreeing with the target user. The basic idea of Collaborative Filtering is to make predictions of scores based on the heuristic that two people who agreed (or disagreed) in the past will probably agree (disagree) again. Even though this heuristic can be sufficient for correlating numerous users, systems that have employed the method still appear to be highly sparse and thus ineffective in making accurate predictions at all times.

Once the neighbourhood of users is formed, an algorithm is used to produce recommendations. B.Sarwar et. al. [14] distinguish three phases in the recommendation production mechanism: *Representation*, *Neighbourhood Formation* and *Recommendation Generation*. The first one, "*Representation*" deals with the modelling of items that a customer has already purchased. The second phase, "*Neighbourhood Formation*" deals with the problem of identifying other neighbouring customers for which there are common items, and finally the third task focuses on the problem how to find the best N products or how a neighbouring user would rate some product unknown to her.

The Representation in a typical Collaborative Filtering system has the form of stored transactions of n customers against m products, an $n \times m$ matrix in which boxes either contain the ratings given by user n for product m or zero in the case that there is no experience by that user at all. Such a representation is not perfect and it has problems when associated with this type of Recommender system.

The commonest problems of centralized Reputation systems are:

- Sparsity: The problem is the reduced coverage due to having a sparse matrix and appears as the system being unable to produce recommendations for a user. By sparsity we mean the lack of data required for a Collaborative filtering system to work. Levels of 95% sparsity are common in contemporary Collaborative filtering systems. For instance, the sparsity of the matrices of "Eachmovie" and "Movielens", two of the publicly available datasets typically used in research, are respectively 97.6% and 95.8%.
- Scalability: These types of algorithms require computation that grows with both the number of users n and the number of products m. The increase in the numbers of users and items will require increasing amounts of computational and storage resources for the correlation of the data. Sparsity, however, helps those systems to scale up, since due to it the number of computations is kept within reasonable limits and makes it affordable for a centralized system to perform the required calculations in real time.
- Synonymy: Different item names can refer to similar products and thus correlation-based systems may see no match between products when computing the correlation. For example, suppose there are two customers of which the first

one rates 10 different "recycled letter pad" products with a high mark and the second one also rates with high mark 10 "recycled memo pads". Any correlation between the items will not be seen by a Correlation-based Recommender system, thus it will be unable to provide an association between the first and the second customer. Mainly, this is because these methods work with exact matches.

- Cold-start problem: Users who have experienced just a few ratings are unable to become members of potential neighbourhoods. So, they become isolated and thus they cannot receive good quality recommendations. This problem occurs in situations where a user enters the system and has expressed no ratings at all. Collaborative filtering techniques cannot provide any recommendations in such cases. Cold start users are usually identified as those who have expressed no more than 5 ratings. However, the users need to receive good quality recommendations as an incentive to keep using the system.
- Vulnerability to attacks: The attacker can simply create a fake user with similar preferences to those of the targeted user and thus he/she becomes highly influential to the victim. If the process of creating recommendations is known and the ratings of every user are publicly available, a simple but effective attack can be the following: A *malicious* user wants to recommend some product *f* to the *target* user. Then he can create a new user *fakeuser* who can rate all the items rated by the *target* user in the same way and also rate with the highest mark the product *f*. In this way the system looking for users similar to *target* user will see the high similarity of user *fakeuser* and *target* user and so it will weight his rating on *f* with the highest mark and therefore recommend *f* to the *target* user. This could occur in those centralized systems which make user ratings publicly available.

As regards the problem of sparsity, various techniques have been proposed for its reduction and the best known is Singular Value Decomposition (SVD)[30]. SVD is used in recommender systems to perform two different tasks: First, it is used to capture latent relationships between customers and products, which allow us to compute the predicted likeliness of a certain product by a customer that could not be seen before. Second, SVD is used to produce a *low-dimensional* representations of the original customer-product spaces and then compute the neighbourhood in the reduced

space. We can then use that to generate a list of the *top-N* product recommendations for customers. However, for extremely sparse data sets it has shown no improvement over standard techniques. In our approach we intend to reduce sparsity, as well as to provide users with recommendations of higher quality.

2.6 How a typical Recommender System works

Collaborative filtering is a technique used to detect patterns amongst the opinions of different users and to make recommendations to people automatically, based on others who have shown similar taste. It essentially automates the process of "word-of-mouth". In collaborative filtering systems the votes of a user are predicted by an algorithm which has its basis in some representative information about the user itself and a set of weights calculated from the database of user's experiences.

2.6.1 Memory-based algorithms

When asked for a recommendation by a user, a standard Collaborative Filtering algorithm includes the following 3 steps:

Step 1

It compares the user who asks for a recommendation against every other user in the community, computing a user similarity coefficient. For this task different techniques have been proposed: the Pearson Correlation Coefficient is the best performing and most frequently used [28]. Proposed alternatives are the Constrained Pearson Correlation Coefficient, the Spearman Correlation Coefficient and Cosine Similarity. In the following formula, the Pearson Correlation Coefficient $w_{a,i}$ represents the Similarity of user a to some user i with regard to their ratings on items and it is defined as:

$$w_{a,i} = \frac{\sum_j (u_{a,j} - \bar{u}_a)(u_{i,j} - \bar{u}_i)}{\sqrt{\sum_j (u_{a,j} - \bar{u}_a)^2 \sum_j (u_{i,j} - \bar{u}_i)^2}}$$

Where j is the number of items rated by both entities a and i, $u_{a,j}$ is the rating given by user a to item j, u_a is the mean of a's ratings. It is important to emphasize

that the coefficient can be computed only if there are items rated by both users. The Pearson correlation coefficient gives valid results in the case where it is calculated for more than five common experiences; otherwise it has unstable behaviour, is not meaningful and gives unsatisfactory results. The coefficient can be computed for overlapping items. For similar users it takes values close to 1 and for dissimilar users it tends to -1. A zero value would mean that there is no correlation between these two users or else, there is no linear relationship between the rated items and thus the experiences cannot be used for doing predictions.

Step 2

It predicts the recommendee's rating for every item she has not yet rated. So a predicted rating $p_{a,j}$ of an active user for item j is a mean rating plus a weighted sum of deviation from the mean rating for every user where the weight is the user similarity to user a:

$$p_{a,j} = \bar{u}_a + k\sum w(a,i)(u_{i,j} - \bar{u}_i)$$

If a user A is similar to user B, much importance is given to the opinions of user A when creating a recommendation for user B.

Step 3

The system suggests to the user the items with the highest predicted ratings.

Apart from Pearson's, the other method most often used for correlation is called Cosine-based similarity. Here, two items are thought of as two vectors in the m dimensional user-space. The similarity between them is measured by computing the cosine of the angle between the two vectors. Formally in the $m \times n$ ratings matrix similarity between items i and j is denoted by sim(i,j) and is given by:

$$similarity(i,j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\left\|\vec{i}\right\| * \left\|\vec{j}\right\|}$$

where the symbol "·" denotes the dot-product of the two vectors. The denominator contains the product of the actual length of the two vectors.

The memory based method is quite well known and was first used for rating articles. The best known developed scheme is Grouplens [5]. The Pearson Correlation Coefficient was established as the basis for the weight factor mainly from Grouplens. Hence, the correlation between users *a* and *i* can be calculated by knowing the sums of their relative distances over the j items for which both users have expressed ratings.

A detailed survey that discusses this type of algorithm as well as to the Amazon Recommender system can be found at [81].

2.6.2 Model-based methods

The type of collaborative filtering model we described above is called Memory-based and is the most common type. For reference we mention Model-based methods which are also known as *Probabilistic methods*. The best known types of probabilistic model are the Cluster model [76], the Bayesian Network [15] and the Horting model [33]. Despite their speed in providing recommendations they are not practical for environments in which user preferences are updated regularly [14].

Bayesian networks create a model based on a training set with a decision tree at each node and edges representing user information. The model can be built off-line over a matter of hours or days. The resulting model is small, fast, and essentially as accurate as nearest neighbour methods [32]. Bayesian networks may prove practical for environments in which knowledge of user preferences changes slowly with respect to the time needed to build the model but are not suitable for environments like ours in which user preference models must be updated rapidly and frequently.

In Cluster models users are grouped by their similarity in preferences and predictions are made regarding the participation of the user in some cluster. In the case of participation in various clusters the prediction is a weighted average.

Once the clusters have been created, predictions for an individual user can be made by averaging the opinions of the other users in that cluster. Some clustering techniques represent each user with partial participation in several clusters. The prediction is then an average across all clusters, weighted by the degree of participation.

As shown in [19], algorithms based on Clustering have in some cases worse accuracy than Nearest-Neighbourhood, therefore pre-clustering, which is a time

consuming operation, is recommended. Once the clustering is complete, however, performance can be good since the size of the group that must be analysed is much smaller.

Another type of Memory-based algorithm is called *Vector similarity*. In information retrieval, the similarity between two documents can be measured by supposing that each document is a vector of word frequencies and computing the cosine of the angle between the two frequency vectors [52]. This formalism can be adopted in collaborative filtering systems where Users and titles have the role of words and votes take the role of frequencies.

2.6.3 Extensions to Memory-based algorithms

One extension method for collaborative filtering is *default voting* and works where there are few common votes between two entities. The standard correlation algorithm in that case would not perform well because it uses votes in the intersection of the items that both individuals have voted upon $I_a \cap I_j$. If some default value is assumed for the cases that common votes do not exist then we can form the match over the union of their voted items $I_a \cup I_j$. Moreover we can assume some default vote value for some number of additional items that neither has voted on.

2.6.4 Other methods

Horting [33] is a graph-based technique in which nodes are users and edges between the nodes indicate the degree of similarity between users. Predictions are produced by walking the graph to nearby nodes and combining the opinions of the nearby users. Horting differs from nearest neighbour algorithms as the graph may be walked through other nodes which have not rated the item in question, thus it explores transitive relationships that nearest neighbour algorithms do not consider.

In other words, this method claims that it solves the problem of the quality of Recommendations by using transitive similarities. The Horting method has so far been tested with synthetic rather than real data and it seems that it produces better predictions than a nearest neighbour algorithm. Among the advantages of the Horting

method are speed, scalability, accuracy of results and the fact that it requires only a modest learning curve.

In our system we make use of a memory-based model since these are best known and are used in Reputation mechanisms in on-line bidding web sites such as eBay, Amazon, etc. Another reason for this choice was the nature of the dataset itself we used for testing which was more convenient (because of high availability of votes) for that method. The fact that memory-based models, in contrast to Bayesian models, do not have a learning phase makes them suitable for yet another reason. Such systems provide a suggestion service in the form of top selections or they simply provide some rating about a product that is asked for.

In all these memory-based systems the rating is provided by the users themselves but all the weighting calculations are done in a centralized manner finally producing a global scale value.

Here we need to mention the advantages that a decentralized system could offer in comparison to all the systems we have described. Decentralized system would provide resistance to attacks and can work unbiased since there is no central place where somebody could maliciously provide biased results. In addition to the scalability problems that centralized solutions have we should also mention that they provide a single point of failure.

It is worth pointing out that in the bibliography, memory-based algorithms are divided into Item-based Collaborative Filtering algorithms and User-based algorithms [31]. Experiments suggest that Item-based algorithms provide dramatically better performance than User-based algorithms while at the same time providing better quality than the best available User-based algorithms. This is also because User-based systems' widespread use has revealed potential challenges such as: *Sparsity* which is responsible for poor recommendations and *Scalability* because they require computation that grows with the number of items and the number of users.

2.7 Summary and Conclusion

As can be seen from the systems and algorithms reviewed, almost none of them work as a complete solution for providing recommendations in an unstructured Peer-to-Peer infrastructure, as they are almost all based on ideas inspired by the centralized world.

Many of them also require considerable user intervention to operate. None of the technologies presented can be adopted without change for a Recommender system.

The algorithms of Recommender systems we present in this review have been designed to work mainly in centralized services and require that a large amount of computation is done by a central entity and therefore they do not scale well for large numbers of users. Also, in the form they are currently provided they cannot be applied in decentralized environments.

Clustering users in groups of common interests is not suitable for our use-case because new recommendations are produced very frequently. To do this would require frequent restructuring of the clusters and as a result an increased number of computations with serious impact on performance.

In most advertised solutions there are no figures for performance and therefore the applicability of each method can not be fully guaranteed due to the lack of relevant evidence. Furthermore, many of the proposed solutions were tailored specially for existing infrastructures and Peer-to-Peer protocols and therefore those particularities were taken into consideration in the design of the proposed solutions. In all cases they use infrastructures from unstructured Peer-to-Peer networks.

As regards the use of graph topologies for propagating recommendations the algebras examined do not provide support for long chains nor for combining recommendations in serial or parallel when there are multiple paths to explore.

In the next chapter we present Trust systems for computational environments, refer to some fundamental issues and examine in more detail how they cope with network infrastructures.

Chapter 3

ESSENTIAL KNOWLEDGE ABOUT TRUST SYSTEMS

3.1 Introduction

In this chapter we explore Trust in computer systems and analyse its properties. Out of all the existing solutions for building Trust systems we focus on Subjective Logic, its operations, the algebra that it offers, and its benefits and limitations. First, however, we discuss some other models of Trust that are worth mentioning. We also discuss the Dempster-Shafer model which formed the basis for Subjective Logic and we give a basic description of the notion of Uncertainty as it is useful for expressing opinions that are based on observations that cannot be infinite. In this chapter there is more focus on Subjective Logic because it is the model we actually chose to use in our Peer-to-Peer Reputation system.

3.2 Trust in Computing Systems

Trust and Reputation have always been a concern for computer scientists and much work has been done to formalize it in computing environments [20]. In computing, Trust has been the subject of investigation in distributed applications in order to allow service providers and consumers to know how much reliance to place on each other. As we mentioned above, the relation between Trust and Reputation is that Reputation is a commonly held belief about an agent's trustworthiness.

Yahalom et.al in [21] distinguish between directly trusting an entity about a particular subject and trusting an entity to express the trustworthiness of a third party with respect to a subject. These two types of Trust are known as *Direct* and *Indirect* (or derived) Trust.

Given the above distinction, the obvious concern is how does one traverse a network or the web-of-trust (a trust recommendation path) that develops in an environment in which one Trusts an agent, who can also express beliefs about the trustworthiness of others, and the other who may do the same?

A variety of definitions of Trust have been given and in many of them Trust is declared to be dependent on the context in which an interaction occurs or otherwise on the observer's subjective point of view. One of the definitions by Deutsch [24] says:

> *"If an individual is confronted with an ambiguous path, a path that can lead to an event perceived to be beneficial (Va+) or to an event perceived to be harmful (Va-); he perceives that the occurrence of (Va+) or (Va-) is contingent on the behaviour of another person; and he perceives that the strength of (Va-) to be greater than the strength of (Va+). If he chooses to take an ambiguous path with such properties, I shall say he makes a trusting choice; if he chooses not to take the path he makes a distrustful choice."*

From this definition we notice that harmful events are those to which the individual pays more attention than beneficial ones. In other words, this explains why trustful paths are hard to build and easy to be destroy.

3.3 Properties of Trust

In this section we describe some essential properties of Trust which are found to be important for the modelling of Trust we perform in our proposed system. First of all we need to distinguish the right terms to use for the entities that take part in a Trust relationship. We call the entity which expresses Trust about something else the *trustor* and the passive entity for which the Trust is expressed about, the *trustee*.

3.3.1 Subjective

As we mention in the previous paragraph, Trust can be thought as a subjective measure, because different entities can have different kinds of Trust for the same target entity. If we imagine a community of agents, the Trust for a specific agent is dependent on point of view, since every agent will have a different set of experiences with the trustee and consequently different perception of their Trust on the trustee.

The repeated use of the word "*perceive*" in Deutsch's definition shows Trust as a subjective quality that two individuals place on each other. The fact that different entities can have different kinds of Trust in the same target entity means that Trust is subjective.

3.3.2 Context Specific

As with Reputation, Trust is also context specific since it is related to the objective and the nature of the Trust relationship, otherwise called *purpose*. From Deutsch's definition we also notice that Trust is related to the purpose and nature of the relationship.

For example, if the context is finding a good car mechanic, then the Trust for those entities that could be trusted must be to do with suggesting a car mechanic. As we will see in the next paragraph the existence of common purpose is a requirement for having transitive Trust.

3.3.3 Transitive

As seen in [7] Trust is not implicitly transitive. There are arguments about whether or not Trust is transitive. However, under specific circumstances transitivity can be allowed when covered by explicit conditions.

Trust transitivity means if, for example, Alice trusts Bob who trusts Clark then Alice will also trust Clark. This is what happens in a typical recommendation. In a classic scenario if Bob trusts Clark as a good car mechanic then the Trust of Alice for Bob must be in suggesting somebody to be a good car mechanic. Under this common purpose (Alice trusts Bob to provide that recommendation) we can say that Alice can trust Clark. So Alice trusts Bob as a recommender of a good car mechanic. If the level of Trust of Alice for Bob is known then it makes it possible to find out how much trust Alice can place on Clark to be a good car mechanic. The Trust purpose of the final leg must somehow be part of every leg of the Trust chain. Trust in the ability to recommend represents recommendation Trust and is precisely what makes Trust become transitive.

As we saw in section 3.2 Trust can be split into *Direct* and *Indirect* (derived). This raises an issue of how one can traverse a whole chain of intermediate entities to find a Trust value for a distant trustee.

Even though it has been shown that Trust is not necessarily transitive [7] there are various requirements such as the context, otherwise known the Trust purpose, that need to be specified and which indicates the ability to recommend [22]. This ability, if it exists, makes *Indirect* Trust possible. Assuming that this ability is present in a long chain then a recommendation can be made since indirect Trust can be calculated along the chain.

Sometimes in the literature, *Indirect* Trust is referred to as *Recommendation* Trust and *Direct* Trust is referred to as *Functional* Trust. In reality, Trust is never absolute and many researchers have proposed expressing Trust as discrete verbal statements, as probabilities or continuous measures. One observation which can be made, from a human perspective, is that Trust is weakened or diluted through transitivity.

The idea of constructing chains of transitive Trust, based on a single Trust purpose with *Functional* and *Recommendation* variants, is captured by the following definition found at [34]:

"A valid transitive Trust chain requires that the last leg in the chain represents functional Trust and that all other legs in the chain represent recommendation Trust where the functional and the recommendation Trust legs all have the same Trust purpose".

This observation suggests that the Trust purpose of the final leg must somehow be part of every leg in the Trust path.

In the case that there are both functional and recommendation Trust in a purpose, these should be expressed as two separate Trust legs. In the above example, the existence of both functional and recommendation Trust legs e.g. from Clark to David should be interpreted as Clark having Trust in David not only to be a good car mechanic but also to recommend somebody else for that job. In that case that expression can be pictorially represented as in the following figure.

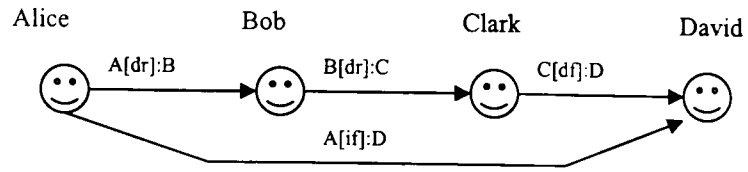

figure 1.

The above chain of entities in figure 1 can be written as:

Alice : David = Alice : Bob : Clair : David

The transitive path stops where the first non-recommendation Trust leg is encountered.

A distinction can also be made between *Initial Direct* Trust and *Derived Indirect* Trust, where *i* indicates Indirect Trust, *d Direct* Trust , *r Recommender* Trust and *f Functional* Trust. What we aim to do is to find out what would be the Functional Trust of Alice for David given that we know the Trust of the intermediate entities as recommenders:

Alice[ifP1]:David = Alice[drP1]:Bob[drP1]:Clair[dfP1]:David

In other words in order to know the *Functional* Trust of a distant entity, we require the *Recommender* Trust of the intermediate entities to be known. In the case that an entity has both functional and recommendation Trust for another entity then it should be expressed as two separate Trust legs in the expression.

As can be seen, if *Functional* Trusts are known or can be derived by some method or formula then the problem that arises is how to calculate the *Recommender* trusts.

3.4 Combination of Trusts – Topologies

In the previous examples we used serial combination of trusting entities from which we derived the indirect *Functional* Trust of Alice to David. There are cases in which recommendations are received from several sources in order to provide plentiful information in making decisions. We refer to that case as parallel Trust combination. Assume that there is a second opinion in the previous example where Alice can be told how good David is:

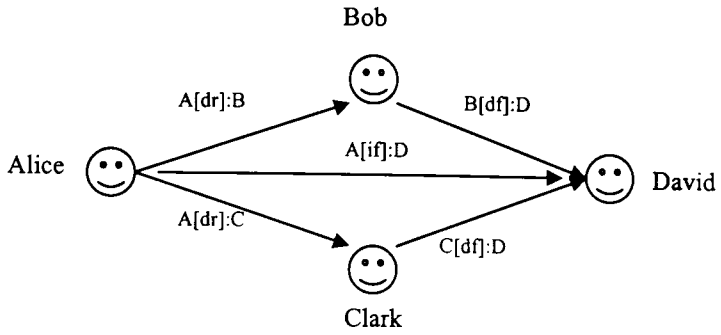

Figure 2.

In this scenario Alice needs to get her car serviced and she asks Bob and Clark who both have some personal experience with David. In other words Alice asks for a second opinion about David. In the case that she receives positive recommendations about David, the Functional Trust of Alice to David is stronger than if she had asked for one recommendation only. In the case of a parallel combination of Trust the expression would be:

$$\text{Alice[ifP1]} : \text{David} = (\text{Alice[drP1]} : \text{Bob[dfP1]} : \text{David}), (\text{Alice[drP1]}:\text{Clair[dfP1]}:\text{David}).$$

3.5 The problem of Dependence

Recommendations should be passed in their original form and not as indirect derived Trust, otherwise there is the danger that the calculation of derived Trust is done incorrectly. The rule is that only direct Trust should be recommended. This is because the relaxation of this rule may lead to situations where the originating entity will not get the real picture of the Trust network. This way makes impossible the existence of hidden topologies that the trustor cannot see. We require that all the information regarding the topology from the trustor to the trustee is forwarded back to the originating entity.

In a recommendation topology similar to that of figure 3 there are various ways that the graph can be traversed to find how much Trust A would place to E.

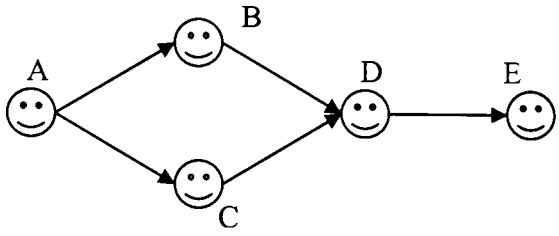

Figure 3.

A perceived topology (A:E) might be interpreted as (A:B:E),(A:C:E) or (A:B:D:E),(A:C:D:E) both of which are wrong. In the first scenario the existence of the D entity is kept hidden while in the second one the DE relationship appears twice in the expression. The correct one can be considered to be A:E = ((A:B:D),(A:C:D)):E which corresponds to the real topology. There is a crucial difference between recommending Trust resulting from our own experience and recommending Trust which has been derived as a result of recommendations from others. (see 3.2). By recommending Direct Trust the querying entity is able to know the real topology of the Trust network between A and E and therefore the calculations of the simplifications of the resulting Trust graph must be done by the originator A itself after it has received all the topological data.

In a parsing algorithm, which would be constructed to perform the simplification task, such simplification sub-operations should not be done by the intermediate entities themselves. In the example in figure 3 the originator A must wait until all 5 Trust vectors are received $A \to B, A \to C, B \to D, C \to D, D \to E$ during a Trust query and then apply some Trust algebra in order to calculate the derived Trust $A \to E$. Assuming that serial and parallel operations can be applied to a series of Trust vectors, A has first to combine serially the two chains $A \to B$ and $B \to D$ and also $A \to C$ and $C \to D$. Then it should combine in parallel the two chains $A \to B \to D$ and $A \to C \to D$ in order to get A $\to$ D. Finally, it should again apply the suggestion operator to (A $\to$ D and D $\to$ E) to get the requested opinion A $\to$ E.

In order to apply this rule we need to assume that the Trust vectors that are forwarded back from a transitive chain are passed unaltered to the query originator. In other words, we assume that intermediates act with honesty during this phase. This requirement disallows the deployment of simple cacheing mechanisms in the

intermediate nodes which could offer much to the pre-calculation of Trust that could save time and computational effort.

Josang et al. suggests a more practical form of expressing transitive chains. Especially when the topology that is to be analyzed (or some parts of it) are unknown, each isolated Trust relationship can be expressed individually and then an automatic parser can be used to establish valid topologies depending on what needs calculating. In [34] he presents a simple parser algorithm that can be useful for determining whether at least one Trust path exists between two principals. However, further analysis is required to derive the measure of Trust resulting from the topology.

3.6 Time

Josang in [22] mentions *time* as an important element in a Trust relationship. Obviously the Trust of the trustor in the trustee regarding a certain purpose at a certain point in time may be different from the level of Trust after several transactions have taken place between the two entities. However, even if no transactions take place, Trust should gradually change with the passing of the time. The rule about time that should be followed is that when more than one instance of a Trust value exists, the one which has the newest timestamp must be used.

It is also worth mentioning the paper of N. Mezzeti [44] which introduces time to the calculation of Reputation.

3.7 Asymmetry

Trust is an asymmetric binary relation between a trustor and a trustee. It is asymmetric since "A trusts B" does not necessarily imply that "B trusts A". Moreover both the trustor and the trustee could be one or many agents.

3.8 Measures of Trust

Trust has a value. A Trust relation is associated with a value that represents its strength or degree of truth. Trust can have various measure forms and depends on the approach and use. For example it can have the form of binary Trust (trusted, non-

trusted), it can be discrete (strong Trust and weak Trust, strong distrust and weak distrust) or it can even have contiguous forms such as probability or a percentage of belief property in the Trust relationship. From a probabilistic point of view there would be both a certain amount of belief and disbelief (the complement of belief) which can be used to express the level of trustworthiness with absolute certainty.

3.9 Temporal Dimension

Trust has a temporal dimension. Since Trust is learned from past observations, Trust values evolve with new observations and experiences. Moreover, to account for changes in a trustee's behaviour, recent observations carry more weight in deriving Trust than past observations.

3.10 Dempster-Shafer Theory

Dempster-Shaffer theory allows the allocation of probability-like weights to a set of events in a way that permits statements of ignorance about the likelihood of some of the events. The allocation of weights leads to the derivation of two numbers: the *Belief*, the degree to which an event is supported by the evidence, and the *Plausibility*, the degree to which there is a lack of evidence to the contrary. See figure 4. These two numbers are the basis on which any belief-based decision is made.

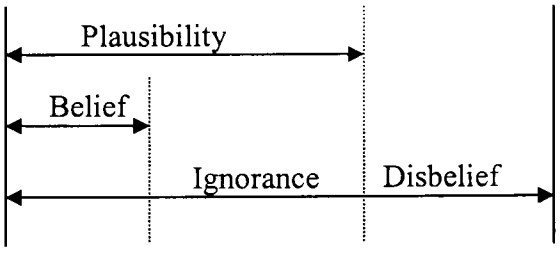

Figure 4.

This theory owes its name to work by A.Dempster (1968) and G.Shafer (1976). The Dempster-Shafer Theory, also called *Frame of Discernment*, has two aspects: degrees of belief and one question utilizing subjective probabilities for a related question. Dempster's rules for combining degrees of belief which are based on independent items of evidence, gives a third basic probability from two given probabilities. This

framework consists of belief – ignorance – disbelief; the noteworthy point is that ignorance is not always the same as disbelief. In the Bayesian framework of discrimination there is only belief and disbelief. Because the Dempster-Shaffer theory introduces a new property, that of ignorance, there is no correspondence with Bayesian logic.

Example:

Imagine that there is a traffic light and two witnesses who report its state. The traffic light can be in any of the following states {*Red, Amber, Green, Red+Amber*}. In the case that the first witness would say that he saw an Amber light and the second one a Red light it would be easy to find states which are compatible with those observations which in this case is the *Red+Amber* combination. In an environment with uncertainty the reliability of the witness (in that case how sure they were about what they saw) should be taken into consideration in the analysis.

In the case that the second witness had said that he was sure that he saw a *Green* light, conventional probabilistic logic would give no solution to the problem. Saying nothing about the Amberness of the light, is similar to excluding the possibility that the light was *Amber*. In Dempster-Shaffer theory by introducing ignorance the result might be either that the light was *Green* or *Amber* or *Amber+Red* depending of how sure the first witness was in comparison with the second witness. In that case there is no definite decision about what the light's state was but a decision will be taken when new evidence is provided to resolve the situation.

3.11 Uncertainty

Because opinions are based on observations it is always impossible to know for certain the *real* or *objective* behaviour of the examined entity. [23] introduces the notion of uncertainty to describe this gap in knowledge or else the absence of belief and disbelief. Uncertainty is important in Trust modelling as it is always present in human beliefs and thus is suitable for expressing these kinds of beliefs.

Trust models for calculating Trust transitivity in long chains which also make use of uncertainty have been proposed in the past [8] and are better known as an "*Algebra for Artificial Reasoning*" or "*Subjective Logic*". This model has its basis in uncertain

probability theory and provides the appropriate logical operators for combining beliefs and deriving conclusions in cases where there is insufficient evidence.

As absolute certainty can never exist, the uncertainty property (u) has been introduced to fill in the gap that deals with the absence of both belief and disbelief. A probabilistic approach would treat trustworthiness by observing the pattern of some entities' behaviour and using only two properties belief (b) and disbelief (d) where b+d=1 , $b.d \in [0,1]$. As we mentioned earlier in this chapter, traditional binary calculus assumes statements of Trust as dual valued either *true* of *false*. As such, Subjective logic can be seen as an extension of both binary calculus and probability calculus. The relationship between b,d and u is expressed as b + d + u = 1 which is known as the *Belief Function Additivity Theorem* [8]. As can be seen from this theorem, uncertainty is interpreted as something that fills the void in the absence of belief and disbelief.

3.12 Trust modelling with Subjective Logic

In uncertain probability theory [23], Trust can be thought of as the level of belief established between two entities in relation to a certain context. In this theory *Belief* is expressed in a metric called *Opinion*. Trust is a fuzzy concept and is usually expressed as a binary statement (true or false) but our imperfect knowledge is translated into degrees of belief, disbelief and uncertainty.

Subjective logic is an extension of the Dempster-Shafer logic of the theory of evidence and as we have mentioned, it produces better results for environments with high uncertainty and in cases where real time results are required.

The triplets *(b,d,u)* obeying the rules of the Belief Function Additivity theorem are called *Opinions*. Each of the *b,d,* and *u* parameters has a value between 0 and 1. In Subjective Logic there is also a fourth redundant parameter which is known as *Relative Atomicity a* and shows the bias of the system. An opinion can be converted to a plain probabilistic value by calculating the notion of Probability Expectation *E(x)*:

$$E_x = b + a \cdot u$$

Subjective logic also provides the traditional logical operators for combining opinions (e.g. $\wedge, \vee$) as well as some non-traditional ones. In total, in order to reason about uncertainty six operators have been defined: *Conjunction*, *Disjunction*, *Negation*, *Consensus*, *Discounting* and *Conditional inference*. In our research we use

only the *Recommendation* and *Consensus* operators which are useful for combining series of opinions serially or in parallel and thus analyzing graphs that have been built up from numerous Opinions and thus can be used for simplifying graphs of opinions into a single Opinion. A complete reference to the algebra of Subjective Logic and on how the algebra is applied to *Belief*, *Disbelief* and *Uncertainty* properties can be found in [25].

Even though opinions in the form of (b,d,u) are more manageable due to the flexible calculus that opinion space provides, evidence is usually available in other forms easier for humans to understand.

Despite the fact that *Trust* and *Reputation* express different things, they both have common characteristics such as being context specific and dynamic (they change over time and follow experience growth). Since there are no formal ways of absolutely measuring the Reputation of an object or an entity in general, we attempt to give a solution to approaching trustworthiness by measuring and analysing the derived properties of Trust. Comparing Opinions can be done in relation to a basis. Josang in [8] suggests a priority-ordering scheme for opinions. According to that scheme when comparing opinions the next 3 criteria with the following order must be used:

1. The opinion with the greatest Probability Expectation *Ex*.
2. The opinion with the least Uncertainty.
3. The opinion with the least Relative Atomicity *a*.

Even though such criteria are suitable for ordering Opinions, no tests have been carried out in real environments where Opinions represent physical values. Also the sense of order as given by the above rules is quite fuzzy and nothing explains whether this order should be followed strictly in some evaluation experiment compared with any other alternative.

3.13 The algebra of Subjective Logic

Open networks allow users to communicate without any prior arrangements such as contractual agreement or organization membership. However the nature of open networks makes authenticity difficult to verify. Therefore, an algebra has been proposed as *"logic for uncertain probabilities"* [8] which can be used to calculate the trustworthiness of a target entity in a big network of interacting agents.

As we discussed in paragraph 3.4 there are cases where Trust must be derived by using serial and parallel combinations of Trust vectors starting from the originating entity and ending at the target entity. A full set of logical operators has been proposed in this algebra but we will focus only on the operators used for combining serial and parallel Trust. This algebra has also been tested [25] for assessing Trust in certification chains in the calculation of the authenticity of keys in the chain.

In the algebra of *Subjective Logic* the Opinions are defined as: Let ω={b,d,u} be a triplet where the first, second and third component correspond to belief, disbelief and uncertainty respectively. Opinions defined in this way have 2 dimensional measures consisting of a probability dimension and an uncertainty dimension. Since an opinion can be interpreted as an uncertainty probability measure, this logic can be called a *"calculus for uncertain probabilities"*.

The symbol ω will be used to denote Trust. We use superscripts to indicate the subject and subscripts to indicate the belief statements, so that:

$$\omega_p^{\mathrm{A}} = \{b_p^{A}, d_p^{A}, u_p^{A}\}$$

represents Agent's A's belief about p, where p is the Trust purpose or else the context of belief. For example, A believes that some key is authentic to the degree that is expressed by the belief disbelief and uncertainty components respectively. Next we focus on the Recommendation and Consensus operators.

3.13.1 Recommendation

Let A and B be two agents where $\omega_B^A = \{b_B^A, d_B^A, u_B^A\}$ is A's opinion about B's recommendations and let p be a binary statement where $\omega_p^B = \{b_p^B, d_p^B, u_p^B\}$ is B's opinion about p expressed in a recommendation to A. Then A's opinion about p as a result of the recommendation from B is defined by

$$\omega_p^{AB} = \omega_B^A \otimes \omega_p^B = \{b_p^{AB}, d_p^{AB}, u_p^{AB}\}$$

Where: $b_p^{AB} = b_B^A b_p^B$, $d_p^{AB} = b_B^A d_p^B$, $u_p^{AB} = d_B^A + u_B^A + b_B^A u_p^B$,

The recommendation operator can only be justified when it can be assumed that recommendation is transitive, or, more precisely, that the agents in a recommendation chain do not change their behaviour.(i.e. what they recommend)

3.13.2 Consensus

Let A and B be agents and the Opinions respectively held by them about the same binary statement p be respectively: $\omega_p^A = \{b_p^A, d_p^A, u_p^A\}$ and $\omega_p^B = \{b_p^B, d_p^B, u_p^B\}$. Then the Consensus Opinion held by an imaginary agent [A, B] representing both A and B is defined by:

$$\omega_p^{A,B} = \omega_p^A \oplus \omega_p^B = \{b_p^{A,B}, d_p^{A,B}, u_p^{A,B}\} \text{ where,}$$

$$b_p^{A,B} = (b_p^A u_p^B + b_p^B u_p^A)/(u_p^A + u_p^B - u_p^A u_p^B)$$

$$d_p^{A,B} = (d_p^A u_p^B + d_p^B u_p^A)/(u_p^A + u_p^B - u_p^A u_p^B)$$

$$u_p^{A,B} = (u_p^A u_p^B)/(u_p^A + u_p^B - u_p^A u_p^B)$$

The effect of the Consensus operator is that it reduces uncertainty. Moreover, Opinions which contain zero uncertainty cannot be combined. The requirement for Consensus is that the same relationship should not appear twice in an expression. e.g the Consensus of an Opinion with itself.

3.14 The problem of Dependence

As we mentioned before, the Consensus and Recommendation operations must be applied in the right order in an expression to avoid having hidden topologies. That happens in cases where several recommendation chains produce Opinions for the same statement. It is therefore important to know in which order Opinions are combined so that the Opinion independence criterion is not violated.

3.15 Pitfalls of Trust graph analysis

Special care must be taken in the cases where graphs of Opinions cannot be analyzed and simplified down to a single Opinion because of the existence of topologies that seem unable to be analyzed using the provided algebra. In these cases neither Consensus nor Recommendation operations can be applied in an analysis. A special arrangement must be made in order to exclude from the graph analysis those Opinions that can lead the operation of analysis to deadlock. Ideally, all possible paths between any pair of entities should be taken into account when deriving the Trust value. However, a directed graph may contain loops and dependences. Excluding certain paths can avoid this, but can also lead to information loss. Therefore, some specific selection criteria are needed in order to find the optimal subset of paths to include. The following figure (5) illustrates an example of a graph that is called a Non-optimal Directed Series-Parallel Graph.

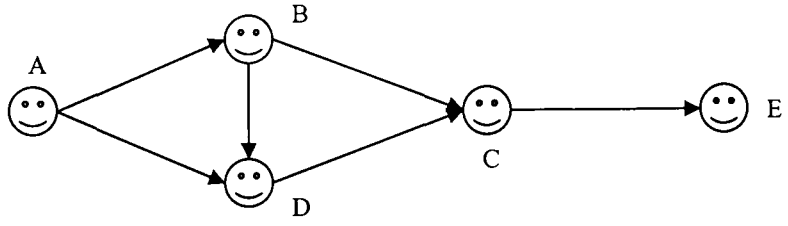

Figure 5.

It is assumed that A is the source and E is the target for which Trust is to be calculated.

Josang et. al. in [41] presents some criteria for preserving *Directed Series Parallel Graphs*. Among those criteria, there is a way of characterising some paths as non-canonical ones and thus excluding them from the simplification process. In the above

example $B \rightarrow D$ is a non-canonical and therefore it should not be considered in the calculation of Trust between A and E.

There is a criterion for preserving a *Directed Series Parallel Graph* when adding new Sub-paths which is given in three rules. More information about the criteria can be found in Chapter 6 where we examine some special cases. The question that arises though is what happens in the cases where the infrastructure is pre-specified and the links between the entities already exist? In the above example a decision must be made regarding which one of $B \rightarrow D$ or $B \rightarrow C$ will be considered as Canonical.

3.16 The beta distribution function

Another problem that has not been discussed so far is how Reputations are created. In the physical world evidence exists in various formats and various kinds. This creates the need for some engine that will produce Reputations from the data gathered in a form that can be recognizable by the Trust algebra we presented above. Such systems are called Reputation engines because they are used to produce data from physical sources and involve activities that convert them from their primitive form to first hand evidence.

The *beta* Reputation engine is so called because it is based on the beta probability distribution function. In contrast to most other Reputation systems [65][66] which are intuitive and ad-hoc, the beta distribution system has its basis in the theory of statistics. Although we describe a centralized approach, the beta Reputation system can also be used in a distributed environment.

The system is based on the beta probability density function which is used to represent probability distributions of binary events. In our model (see Chapter 5) we have used our own Reputation system due to the nature of the existing data which made it impossible for them to be expressed as binary events.

Beta provides a sound mathematical basis for combining feedback and for expressing Reputation ratings. The mathematical analysis leading to the expression for posteriori probability estimates of binary events can be found in many text books and papers [77][78] on probability theory.

The whole idea behind the beta engine is the existence of a process which creates events with two possible outcomes $\{x, \bar{x}\}$. From those events we call r the observed

number of outcome x and s the observed number of $\bar{x}$. Then the probability density function of observing outcome x in the future can be expressed as a function of past observations by setting:

$$\alpha = r + 1 \text{ and } \beta = s + 1 \text{ where } r,s \geq 0$$

For example a process with two possible outcomes $\{x, \bar{x}\}$ which has produced outcome x seven times and $\bar{x}$ only once can be expressed with a beta distribution function as f(p|8,2). The probability expectation value given by E(p)=0.8 can be interpreted as saying that the relative frequency of outcome x in the future is somewhat uncertain and that the most likely value is 0.8. There is a whole family of continuous beta density functions indexed by two parameters α and β.

The probability density function for beta is a 3-dimensional representation of uncertain probabilities and there is a way to perform a mapping between the two representations (evidence and opinions) which lead to equivalent interpretations.

In the mapping done by Josang [8] there can be opinions (b,d,u) provided by the r and s parameters we introduced above that can be expressed as functions. Josang in [8] has set empirically the following requirements in order to map the evidence space to opinion space :

1. b (belief) to be an increasing function of good experiences r,
2. d to be an increasing function of bad experiences s,
3. u to be a decreasing function of (r,s),
4. e[f(p)]=e[ω]

where 1 and 2 are set so that there is an affinity between b and r and between d and s. Requirement 4 means that there is equality between the probability expectation values of the derived opinions and the beta probability density function of the evidence. From those requirements we get:

$$b = \frac{r}{r+s+2}, \qquad d = \frac{s}{r+s+2}, \qquad u = \frac{2}{r+s+2}$$

As can be seen, uncertainty reaches its maximum value 1.0 when there is no evidence at all and thus good and bad experiences reach zero $r = s = 0$.

3.17 Other Trust models

It is worth mentioning the existence of other Trust models that have been proposed so far, such as that of Li Ding et.al. [29] which distinguishes Trust into *Domain* Trust and *Referral* Trust. The first one refers to an agent's beliefs about the trustworthiness (or usefulness) of other agents' knowledge in a certain domain. Referral Trust relates to an agents' beliefs about the trustworthiness of other agents' referral knowledge. This concept is quite similar to that of Josang's ideas about Primary and Recommender Trust.

In the same paper besides the *right, wrong* and *unknown* properties of Trust there is a new property introduced which is called *untouched* and corresponds to experiences where an agent responds with "*not known*". In other words it distinguishes ignorance from experiences in which the participant answered with "*not known*".

Their method uses the consensus of selected trusted agents to derive Trust about an unfamiliar agent. They also provide a formula which shows that an agent A derives the reliability of *Domain* Trust about agent B from the weighted consensus of a set of other agents NA.

$$R_{domain}(A,B,domain) = \frac{\sum_{N \in NA}[R_{domain\_ref}(A,N,domain) * R_{domain}(N,B,domain)]}{|NA|}$$

3.18 Other systems

It's worth mentioning the work done by P.Massa in building a Trust Aware Collaborative Filtering system [35]. Here there is an attempt to build a web-of-trust between the users who express their Trust values to their neighbours by themselves. In this system there is no need for central entities to work. We can visualize the derived Trust network with nodes being the users and edges Trust statements.

The derived or predictive Trust to a distant user in the graph is calculated simply by multiplying the intermediates' Trust values from the origin to the destination. After some interaction has taken place, users are required to express their level of Trust about the user they have interacted with. Once the web-of-trust has been formed, a graph-walking algorithm is used to predict the importance of a certain node in the

network. The webs-of-trust of all users can be aggregated into one global Trust network that would look like this.

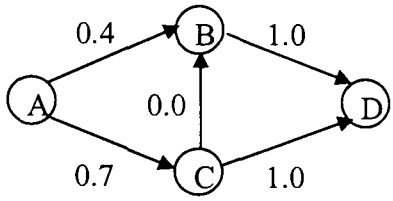

Figure 6.

From the Trust values it is possible to calculate the similarity measures between two distant users where the similarity metric used is that of Pearson.

In this work, different experiments were conducted with different maximum propagation distances from 1 to 4. The policy was based on the heuristic that the further away the user for which Trust is to be calculated is from the current user the less reliable is the inferred Trust value. There is also a comparison between the classic pure Collaborative Filtering method and the proposed one, called Trust-aware Collaborative Filtering.

The results show that user similarity tends to perform well with users who have rated many items and poorly with users that have rated few items. However, even though the method benefits from coverage (because more users become potential neighbours through the Trust network) this method generates substantially high mean error which increases as the depth of Trust propagation increases. Such high mean absolute error (MAE) makes any results that the system can provide unreliable. Especially for cold start users the error appears to be quite high and accurate prediction is difficult for the system.

Among the weak points of this model we should mention the way that neighbourhoods of users are formed which requires the users themselves to provide Trust measures about those they consider as neighbours. We believe that it is unreasonable to expect users to rate large number of items and at the same time to involve themselves in a process of rating their counterparts' trustworthiness since such an operation is both boring and risky for them. This is because users use their own personal taste, which might be different from some other people's, to rate material. That requires users to have some experience in expressing Trust rates about other counterparts as well as being able in rating items.

Another weak point of this approach is the way that Trust is propagated in the resulting graph. The propagation, in the sense that it happens in the proposed system, is done in a simplistic and arbitrary way by multiplying the Trust values along a path. The paper does not state how a complex graph is traversed and what rules are applied in the calculation of Trust.

Finally, the calculation of the correlation coefficient is treated positively only in the case that there is positive similarity ($w>0$) and what happens in the case where the users for which the similarity is to be calculated are completely dissimilar ($w = -1$) is not explained in that paper.

3.19 Trust management

Aberer and Despotovic in [36] have proposed a solution for how to manage Trust in Peer-to-Peer communities. It is based on the idea of analyzing the earlier transactions of agents and deriving from those the Reputation of an agent. It defines as Reputation the assessment of the probability that an agent will cheat and the method is based on statistical data analysis of former transactions. They propose a method that scales well for a large number of participants because it does not make use of a central database and due to its distributed nature it can be applied to a Peer-to-Peer system. In this model the data are stored in a distributed database called P-Grid [40].

The problem related to Reputation based management is that information about transactions performed is dispersed around the network and thus peers can have only an approximation of the global situation. As behavioural data the model uses complaints and the form of Trust that it uses can have a binary value indicating that an agent is trustworthy or not. It assumes that Trust always exists and that malicious behaviour is the exception, so Reputation is based on global knowledge regarding complaints that have been received about some entity. The issue of Reputation is seen from the perspective of data management, and more specifically is concerned with where the data about the complaints will be kept. It mainly focuses on the operations that take place when a complaint is to be stored or retrieved from the distributed database and not on the formation of Trust and the procedures that must be followed within the distributed community. Even though the access method is organized in a Peer-to-Peer fashion and the data are dispersed over various hops in the network, the

decisions and the assessment of Trust are not made as it would be in a Peer-to-Peer network. Moreover, the system does not have any preventative measures against inserting arbitrary complaints about users.

Furthermore, P-Grid is advertised as a distributed solution for data storage with all the benefits that decentralization offers. Undoubtedly, having the Reputation data distributed in the peers is beneficial, but on the other hand, the application of a strict Reputation based assessment model in a global scale comes in contrast with the freedom of expression of the personal opinions that all peer-participants in a dynamic Peer-to-Peer community should be allowed to have. In addition, performing updates with high regularity in P-Grid whenever a transaction occurs raises an extra cost in resources and likely translates into deterioration of the existing traffic. In spite of the decentralized characteristics that P-Grid provides, the traffic problems caused by the large number of messages still remains unsolved.

This model also deals with the storage and dissemination of Trust data and not with deriving how much Trust to place in a distant entity.

Using word of mouth techniques for the diffusion of information in the same way as in human communities can be more effective and less costly in resources.

3.20 Summary and Conclusion

Subjective Logic, which has its basis in Saferian theory, has some interesting characteristics that are worth discussing. It offers a set of operators sufficient for deriving the level of Trust in long transitive chains in complex graphs of user agents. There are, however, some special cases of graphs in which the standard operators that the algebra provides can not resolve, and that appears to be a future research issue. The useful (for our case) characteristic of using uncertainty for describing lack of evidence and also the fact that it provides a calculus by which Trust can be calculated between entities in a graph, steered us in the direction of using it in the Peer-to-Peer model we propose. The requirement for the evidence to be in a format suitable for the method requires that the users' agents prepare and deliver them in the appropriate form so that they can be used. More specifically it requires that the data be in a binary form and be referred as good and bad behaviours. For the reason that this is not always convenient we intent to build our own model for converting evidence to

opinions. In essence, it looks to be a very promising solution for solving the problem of quality of the services in Peer-to-Peer systems.

In the toolkits referred to in paragraphs 3.18, 2.4 that make use of Trust in networking environments, there is no distinction between Primary and Secondary Trust. Other issues examined in solutions described as 'Trust Systems', were to do with the storage of data where almost all of the systems discussed (see paragraph 3.19) provide distributed storage of data. In general, there was not much work done in Trust derivation from evidence and in most solutions the rating evidence is provided by the users themselves expressing how happy they were after a transaction has been completed.

For the above reasons we intend to use Subjective Logic for the modelling and management of Trust for our distributed Recommender system. In the following chapter we deal with the encoding and translation of evidence to a form that is recognizable by Subjective Logic, so that they can be converted into Trust measures. We also take care that the main properties of Trust and the principles in which the operations are carried out will not be violated.

In the work that is presented next we propose, build and study a Recommender system which will make use of Trust in relating users together. We first show what can be achieved by the use of Trust in a Reputation system and second we attempt to apply it in a distributed infrastructure.

Chapter 4

THE PROPOSED ARCHITECTURE

4.1 Introduction

In this chapter we describe the design of a distributed architecture for a Recommender system and present its benefits as compared to a centralized architecture. The main characteristic of this architecture is that, for reasons that will be explained later in this chapter, it has its basis on Trust infrastructures. Trust is derived from evidence that the users themselves provide to the system in the form of opinions.

Our design has been done in such a way that the derived system will operate as an integrated collaborative filtering system; therefore, in addition to elementary operations, we have included other functionality found in typical collaborative filtering systems.

Using Trust in a Reputation system can have positive effects on usability and can also provide better results as regards the prediction of user ratings. Chapter 6 contains a study which shows the improved results of a Trust-enabled Reputation system in comparison to a non-Trust-oriented system.

The strong points of such an architecture are its dynamic view of Trust and the robustness that it provides, since there is no single point of failure. The main advantage is that such a design is suitable for Peer-to-Peer systems which, as is well known, are self structured, making them stay functional under conditions which may require dynamic restructuring.

We also examine a simple scenario of the operation of a Recommender system by tracking the steps that would be followed in a typical search operation, and we analyse the sub-operations carried out.

In the following two chapters we analyse each individual step of recommendation production separately. We also discuss the main requirements for such an architecture to become applicable at every individual step and thus give an idea of the context of their operation.

There are many similarities between the structures found on a Peer-to-Peer network and a Trust network. In our work we will examine if the existing

infrastructure that establishes a Peer-to-Peer network is itself enough to support a Trust-enabled Recommender system.

4.2 Motivation

Contemporary Recommender systems operate essentially as centralized services [16,17,18]. The main idea behind them is to correlate users based on the opinions they have expressed in the past and to provide them with suggestions in the form of predictions of ratings about items they want to know about.

Trust-enabled Recommender systems can provide an improvement in the quality of recommendations. This is achieved by decreasing sparsity in the datasets, but if this is not done correctly then the extra effort required may cause more side effects than benefits as we will see later in chapter 6. By sparsity we mean a lack of users' shared experiences that Collaborative filtering systems need to work. The extra effort is necessary because such a technique requires first and foremost a large amount of computation to support the Trust infrastructure, and this increases with the number of users. Given the requirement that such systems need to operate in a live, interactive fashion, response times must be kept within strict limits and any additional computational load will almost certainly have negative consequences for usability.

If the system uses a flooding algorithm to propagate Trust values to neighbouring users this would require $O(n^t)$ unicast operations since a query running to depth d would need roughly $L = n\sum_{t=1}^{d} n^{t-1}$ operations for each search operation to propagate to n trusted neighbours. In reality the number of messages will not be as many as the formula shows due to the sparsity of the datasets.

High sparsity is also responsible for obscuring the scalability problem since it is never possible for all users to be correlated with each other. This means that predictions cannot be made about every prospective choice for any user within the community.

The two challenges, namely reduction of sparsity and scalability are in conflict since the less time spent on a search query, the worse the quality of the results the system will give. The increasing number of computations is the main reason for the reduced scalability of Collaborative Filtering systems and this is why, theoretically, a

centralized Recommender system of this type would not scale to a massive number of users and products.

In the next chapter we deal with the operations of Trust derivation that are required in the initial step of Primary Trust establishment, so first we build a model that derives Trust from evidence using an empirical method.

As regards the calculation of, so-called, secondary Trust, we apply two of the rules of evidential reasoning to the Trust calculation as they are described in the framework of *Subjective Logic*. As we described above, that framework, under some specific constraints, provides transitivity of Trust relationships which translates to a capability for deriving how much Trust to place on a distant entity not directly known to the one making a query.

The main idea behind our work on using Trust in such infrastructures is to use it in conjunction with a similarity metric and thus make it possible to work out predictions about measures that are formed using similarity. As we will see in the next two chapters the novelty in our approach is that we apply a scheme through which Trust metrics can be transformed to similarity and vice versa.

The purpose of the proposed scheme is to provide correct Reputation metrics about resources that can be found within a community. These Reputation metrics appear in such system in the form of ratings of items which can be used to guide the choices of a potential user of the system.

Recommender systems [11] generally run as centralized services and often exist as services embedded into web sites which provide support for e-commerce activities. Proposals for interesting designs for Distributed Peer-to-Peer Recommender systems have also appeared in the past. Nevertheless, in the existing works no performance measures have been presented to show the limitations (or advantages) of every particular solution. Also, as we saw in chapter 2, the proposed solutions do not deal with and do not solve the problem of sparsity that Recommender systems have when there are insufficient data to support a recommendation. In general there is no decentralised solution running today, and the level of sparsity in centralized solutions is high and obscures their problems.

4.3 The Lifecycle of the proposed system

In this section we provide a high level view of the system focusing particularly on each entity involved as well as the way in which they interact and collaborate with each other. Figure 1 shows a user-centric view of the collaborating entities.

The aim of the proposed scheme is to provide accurate predictions of ratings for items that a user has as yet not experienced. Achieving this requires finding how similar the user is to others that have had that same experience in the past. In standard Collaborative filtering techniques Similarity between users can only be derived if a considerable number of common experiences exists. Our contribution is a way to link Similarity with Trust in such a way that the latter can be derived directly from former and vice versa. In this way derived similarities can be converted into Trust estimates and then be shared with the user community. In the same way the community can provide Trust estimates (in the form of direct Trust) which users can analyse by using Subjective Logic algebra and derive how much they would Trust some other user for which similarity can not be calculated directly. Finally the Trust estimates are converted into similarity expressions and along with the common experiences of the local neighbours provide the predicted rating.

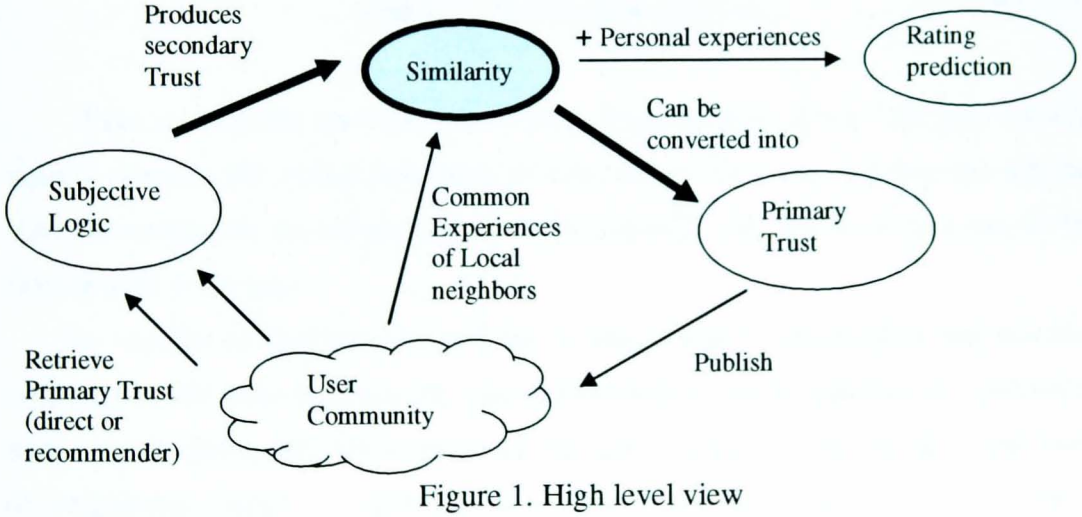

Figure 1. High level view

4.3.1 The System Components

We can describe the system in more detail as follows. In all we distinguish five collaborative entities that can be identified as the main mechanisms of our proposed architecture. These are:

- The user profile
- The registry of local knowledge about the neighbours' rating behaviour
- The similarity calculator
- The Trust estimator
- Trust repository
- The rating prediction engine

We can identify a potential user's personal rating behaviour as the main input along with the description of an item that the user is interested in experiencing. The output can be a prediction of the level of the user's satisfaction if using the item. The composition of the above elements can be represented as in figure 2.

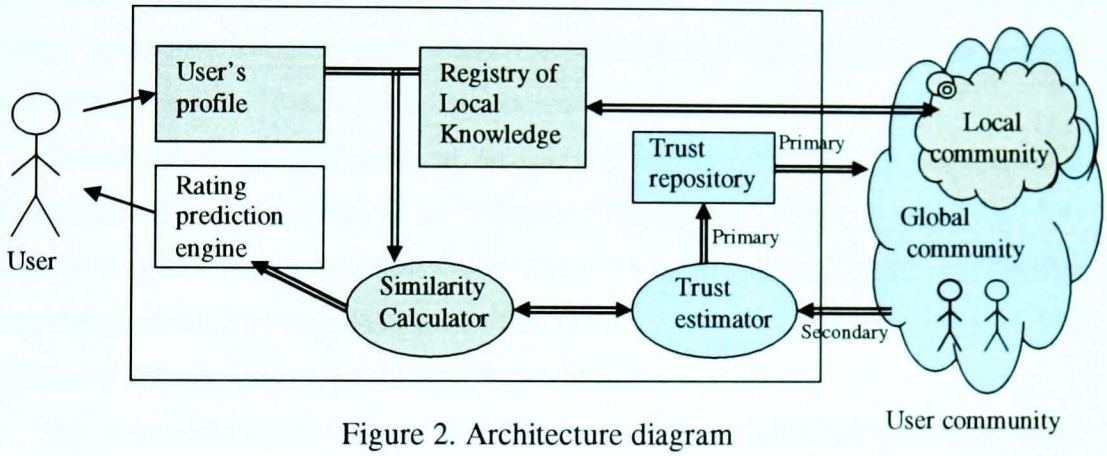

Figure 2. Architecture diagram

Next we describe the functions of each entity in more detail. The user's profile mainly contains the rating behaviour of the current user who is joing the scheme. Such information is the ratings that the user in question has given to items he/she has experienced in the past.

The registry of local knowledge refers to the volume of information that has been collected by the user regarding the rating behaviour of his neighbours. Neighbouring nodes can be any nodes whose existence the user is aware of. In regard to discovery of neighbours within a small logical distance from the user, there are various mechanisms that can be used to achieve this and these are discussed in more detail in the next chapter.

The main purpose of the similarity calculator is to provide measures of how similar various neighbouring users appear to be with each other. The Trust estimator is mainly used for converting similarity measures to Primary Trust estimates and vice

versa. The Trust estimator can also provide estimates of Secondary Trust using as input Primary Trust estimates for other users in the community. The latter information is held in the Trust repository from which the Trust estimator retrieves information to build secondary Trust.

Finally, the rating predictor calculator has as its main task to provide a prediction in a comprehensible form to the user, taking as input rating behaviour as well as similarities with other users.

4.3.2 How do these components collaborate together?

As previously mentioned, the purpose of our proposed system is the provision of good quality recommendations to users. Recommendations may be concerned with a single item or may take the form of a suggestions list.

Almost all components used in the architecture diagram (except for the Trust estimator) can also be found in an ordinary collaborative filtering mechanism. The idea is to allow users to derive their own rating predictions based on existing knowledge about the rating behaviour of others within the community for whom their similarity with the querying users can be calculated.

The main difference between our proposed scheme and standard collaborative filtering is the use of the Trust estimator entity itself. The latter operates as an aiding mechanism which can express Trust estimates for community members that are willing to contribute in the Trust derivation. Next, using the Trust estimator this information is converted into Secondary Trust which can then be passed to the Similarity Calculator for further processing.

We need to distinguish Primary from Secondary Trust because the former is established between any user and his local neighbours in a pro-active way. As such Primary Trust has a local scope whereas Secondary Trust is built upon Primary Trust whenever the system needs to know the trustworthiness of non-neighbouring users. Therefore Secondary Trust is built in a Reactive and dynamic way whenever the circumstances require. For example a rating prediction request sent by a user requires that a number of Secondary Trust queries be executed so that non-neighbouring recommenders can be assessed appropriately.

We can distinguish the following five entities that take part in the process of composition of Primary Trust: User's Profile, Registry of Knowledge, Local Community, Trust Estimator and Trust Repository. Secondary Trust, however, involves only three entities: the Trust Repository, the Trust Estimator and the contribution of the Global community since this Trust is built upon the Primary Trust that is shared by other community members.

Due to the dynamic nature of the system in which each prediction request involves many different entities of a dynamically changed topology such as a P2P system, it is necessary that Secondary Trust is recalculated on a frequent basis. In contrast, Primary Trust requires recalculation whenever the set of personal experiences of the local neighbours is enriched with new experiences.

4.3.3 Example of the use of the system

When a user signs up to the service he first registers his own experiences so that a profile is created for him. When joining the community a sharing of profiles is performed between the local user and his close neighbours and finally this information is stored in the Registry of local knowledge. By comparing the profiles it is possible to find out how similar and how trustworthy each neighbour is for the local user (performed by the Similarity calculator and the Trust Estimator respectively). The Trust that is built is based on first hand evidence which in this case is shared experiences. Therefore Primary Trust is built proactively, is stored in the users' local Trust Repository and can be provided to other users upon request as recommendations.

Whenever the user wishes to know how much he would like a new product, he first checks his local repository for similar users that have experienced the same product in the past and then the search is broadened to the global community. Since it is impossible to know the similarity (and thus the primary Trust) for each of the distant users due to the lack of first hand evidence, the system tries to approximate the trustworthiness of these users by sending relevant queries to the Global community. Upon receiving the Primary Trust estimates of all the intermediate users which are in the path between himself and the remote one the Trust estimator builds, using Josang's Subjective Logic operators, the transitive path and calculates the derived

Trust. It is worth mentioning that this Trust is created in a reactive way and is thus built upon recommendations identified as Secondary Trust. The received Trust value is saved in the Local repository for future use by the user (but should be never provided as a recommendation to other users to preserve opinion independence) and finally passed to the Similarity calculator. The process of calculating the similarity is repeated for all the remote users who have had experiences with the product in the past. Finally all similarities along with the product ratings for all involved users in the query (local and remote) are sent to the rating prediction engine to generate the user's predicted rating for the product.

4.4 Our proposed architecture

In this section we outline the operations that our proposed architecture requires and associate them with relevant functions that can be found in a typical Collaborative Filtering system.

In the scheme we propose there is not only decentralization in the operations involved in the provision of Recommendation but it also has the characteristic that the entities involved can exist and operate autonomously, and also appear to have equal roles within the community. Each entity in the scheme has its own knowledge about its neighbours and can operate similarly to the other entities in the community (e.g. can initiate and reply to queries). They also can voluntarily offer some fundamental support for the existence of the community. For example, the propagation of Trust query messages according to commonly agreed rules that we will describe below is essential for the recommendation service. Such autonomous schemes can be found in unstructured Peer-to-Peer networks and in the architecture we propose the autonomous entities meet the requirements that a peer should have in an unstructured network.

As mentioned in Section 2.5 above, Sarwar et.al.[14] distinguish 3 phases in recommendation production in centralized systems and suggest *Knowledge Representation*, *Neighbourhood formation* and *Recommendation generation* as the key tasks. Our Peer-to-Peer based concept fits this categorization almost perfectly since Knowledge Representation and Neighbourhood Formation overlap. So we

distinguish *Trust Discovery*, *Recommendation Search* and *Recommendation Generation* respectively as the main operations of our distributed proposal.

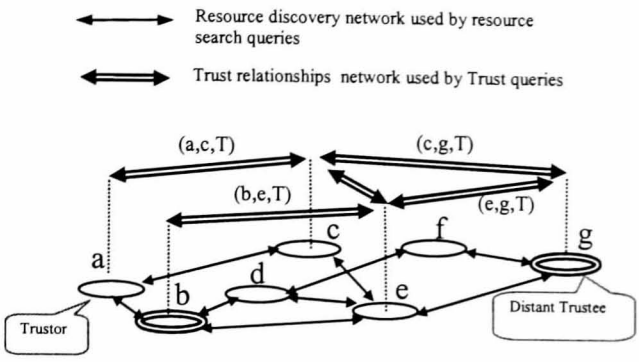

Figure 3. A typical Trust Over-net. (a,c,T) denotes the Primary Trust of a for c.

In order to explain these operations more easily we first present a common searching scenario in a hypothetical Peer-to-Peer system that provides Trust-based recommendations. The scenario consists of the following four steps:

1. User U initiates a simple resource-searching query seeking for some product A she is interested in.
2. The system detects, through the resource discovery operations, the subset S of entities that can provide their own experiences with product A.
3. The system tries to estimate the trustworthiness (*Secondary Trust*) of each of the S entities through the Trust graph, and after that attempts to reach each one of them by initiating Trust queries via its neighbours in the Trust overlay.
4. Once the secondary Trust information has been received back by the trustor (query initiator) and the Trust for every entity S has been established, U estimates what she believes about the trustworthiness of each one, from which she finally derives the similarity to each one from the set of S. Then, by using a predictability formula derives the expected rating of product A.

Figure 3 shows a typical example of two entities **g** and **b** which share some experience with product A. In the scenario a third entity **a** is also interested in product A. This entity is first informed which other entities have experienced A before and by sending a search query finds out that **g** and **b** are those that make up the subset S mentioned in step 2. After it has received this information, it then tries to derive the

trustworthiness of each of the counterparts **g** and **b** for which there is no direct Trust relationship through the Trust graph even though **a** is a physical neighbour of one of them (**b**).

In order to run a Trust query of the type described in step 3 Primary Trust establishment must have been done prior to the search operation. This phase is described below as the *Trust Discovery Phase* and is executed asynchronously with respect to the *Recommendation Search. Trust discovery* is performed by the execution of algorithms that do correlation of the first-hand evidence that any two potential counterparts have collected from their experiences. In the next chapter we will explain in more detail how to implement these operations. The experiences here refer to ratings that the users have given to items. From such evidence we derive the similarity of the entities involved and consequently how much Primary Trust they should place on each other. The Trust overlay that connects the trusted nodes, for which there is sufficient first hand evidence, is based upon those relationships. Special criteria and policies to restrict the overlay to selected nodes can also be applied. This justifies the fact that there is no absolute match in the above scheme between the physical resource sharing Peer-to-Peer network and the Trust overlay.

When this phase finishes, the peers have formed a mesh of overlay links connecting each other and are able to relay Trust queries that help in the derivation of secondary Trust. The secondary Trust queries are performed using a broadcast mechanism based on flooding messages through the overlay. The combination of overlay links produced in the Trust formation phase together with the way that Trust queries are dispersed within the mesh looks very much like an application layer multicast scheme.

In application layer multicast there are various ways of restricting communication to a subset of peers that want to communicate as part of a multicast group. One way to achieve this is by getting peers that wish to form a multicast group to create their individual Peer-to-Peer overlay and then use the new overlay to communicate with each other. Another way is to make the peers that are to take part in the multicast group filter and forward the messages to those only that belong to the group or discard them. Our approach combines both ways of achieving restricted communication with the difference that the first is put into operation during the *Neighbourhood Formation* phase and the second during the secondary Trust querying where Trust-based filtering

is also applied in the propagation of Trust queries. (See Chapter 6 for the description of the algorithm).

Going back to the first phase of our design, we see Trust establishment as the operation of estimating the levels of belief, disbelief and uncertainty between users using their behavioural data as evidence.

Secondary Trust can be calculated if knowing the primary Trust of entities in the overlay which can be achieved by collecting all the Trust vectors that exist between the querying and the examined entity in the overlay mesh. Note the case where there might not be any vectors collected. There might be cases where secondary Trust cannot be calculated due to a lack of Trust relationships in the overlay, which translates to a lack of first hand evidence or common experiences between the relevant peers. In the following sections we analyze each of the individual steps separately.

Figures 4 and 5 show a representation of a centralized architecture for a Recommender system and the proposed distributed one respectively.

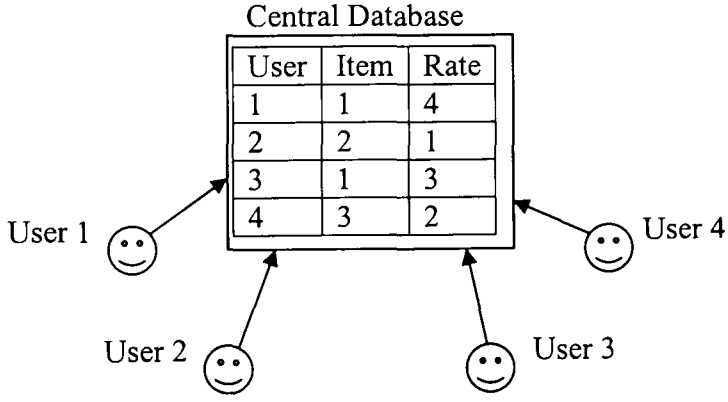

Figure 4. A centralized architecture for a Recommender system

For the sake of simplicity in figure 4 we depict the whole Peer-to-Peer network as a bus topology meaning that there is a mesh type of network connecting the users together.

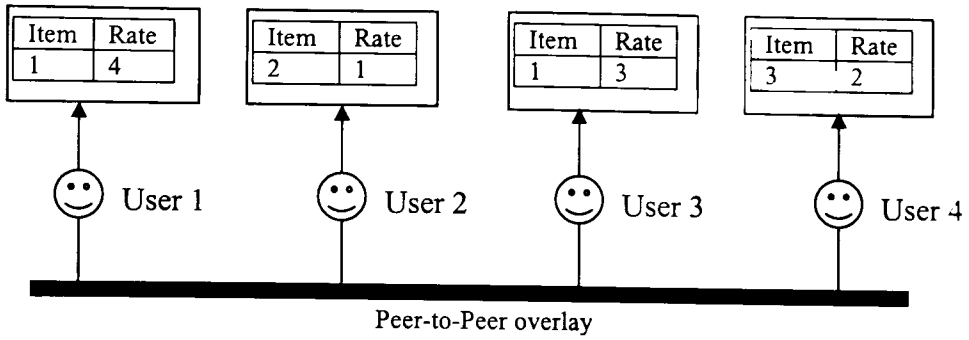

Figure 5. A distributed architecture of a Recommender System

The main difference between the two architectures is that in the distributed system each user keeps their own local database with their own experiences instead of using a common central place for maintaining the data. Such a distributed approach is much more robust than the centralized equivalent but it makes it more complicated to implement search operations.

Next, we present in more detail each operation in the graph structure which is used for the recommendation production.

4.5 Trust Discovery phase.

The Trust discovery phase we mentioned is concerned with the representation of Trust relationships and the formation of neighbourhoods of users with respect to the level of Trust they place on each other. This kind of Trust is known as *Primary Trust*. This phase must take place before a search operation commences and every entity that wishes to participate in the Trust scheme must implement this phase. In contrast to centralized Recommender systems, in our design there is no central customer-product matrix since users maintain their own tables for selecting their neighbours and they keep such information for every product they have used in the past just for the needs of this phase. In the next chapter we present an empirical model for Trust derivation using evidence in which Trust values are derived from the common experiences that every pair of users has had in the past. We can imagine individual cells of the hypothetical user-product matrix as experiences.

In this model the existence of common experiences is a requirement for the establishment of Trust. Additional filters, for example on the minimum number of

common experiences, are sometimes essential for setting up a good Trust graph especially in cases where there is unstable behaviour. In the proposed model there is a mapping that uses both quantitative and qualitative measures to transform observations into Opinions. The use of qualitative measures is what makes our proposed method different from others which simply use the quantity and the kind of experiences as metrics for Trust establishment. The basic idea of our scheme is, the easier the two parties can predict the ratings of their common choices, the more trusted they are. An empirical formula taken from Collaborative Filtering systems is used for the predictions

The operations of the *Trust Derivation* phase can be carried out jointly off-line by the parties involved, which can periodically broadcast messages indicating they are looking for entities with which they wish to establish primary Trust. In the implementation level such operations can be carried out adequately using IP multicasting. Members can periodically send multicast messages to their sub-networks in order to find new neighbours and those who respond can proceed to the phase of exchanging experiences.

Once the *Primary Trust* relationships have been established, the Trust overlay is set and ready to resolve Trust requests. A typical Trust request should be directed to a single node and have the format of (Source node, Target node). That call should expect a response that may look like as a single Trust triplet (b,d,u).

To keep the protocol as simple as possible we rely solely upon the honesty of each counterpart and allow the calculations to be done by the entities involved themselves. A dishonest counterpart could answer either by providing inaccurate data or could even use the data received from the other counterpart to perform an attack. If we consider such attacks from malicious users who might want to influence their counterparts by providing fraudulent evidence, the Trust calculation and the transmission of the results could be done confidentially by the parties involved.

We consider the detection and prevention of such attacks as an issue for further research. For example, Trust calculation could be done by a trusted third party from whom both counterparts expect to get the right results and be assured that the third party will not use the data for performing an attack against any of the counterparts.

4.5.1 The Security Issue

A simple form of the protocol that could be used for computing Trust values more securely and prevent the kind of attack we mentioned is the following, which makes use of a *Third Trusted Party* C:

	Command	Info transmitted	Comment
1	$\{A\} \rightarrow \{B\}$		Requesting B to choose a Third Trusted Entity
2	$C \rightarrow \{B\}$		B finds a Third Trusted Entity which responds positively
3	$\{B\} \rightarrow \{A\}$	C	{B} announces to A which Third Trusted Entity has been chosen.
4	$\{B\} \rightarrow C$	$(Items)_k$	B masquerades and submits its own ratings set to C.
5	$\{B\} \rightarrow \{A\}$	k	B submits the key k also to A.
6	$\{A\} \rightarrow C$	$(Items)_k$	A masquerades and submits also its own choices to C.
7	$C \rightarrow \{A\}$	$Trust_{A \rightarrow B}$	C works out the ratings and announces to each party the Trust values they should place on their relationships
8	$C \rightarrow \{B\}$	$Trust_{B \rightarrow A}$	

We use braces to indicate that an entity communicates anonymously with the other party. Parentheses indicate encrypted content using a key. Anonymity in the IP layer can be provided by using, for example, MIX cascades [69]. In order to allow communication between two peers without revealing their identities to each other, techniques such as anonymous web-posting can be used where messages are posted in anonymous letter boxes associated with keys that are set up for a specific purpose [70].

Given that the Trust Computation application will be running on a unstructured Peer-to-Peer system, we assume that the task of being a *Third Trusted Party* will be carried out by one of the available network peers and not by some external entity. As

a *Third Trusted Party* we can imagine a peer randomly chosen or according to some criteria from the entities that take part in the schema.

That simple form of the protocol does not deal with the fact that the Trusted party C might be one of the counterparts A or B. Even though we assume anonymous communication, it would be easy for either of the A or B to understand if they have been selected as *Third Trusted Parties* by carefully examining the messages exchanged. In case this happens, the selected *Third Trusted Parties* could be able to recover the other's profile and use it for malicious purposes.

What we suggest is grouping items in the ratings list before it gets transmitted and the allocation of more than one *Third Trusted Party* for the calculation of the Trust value by sending a separate group of ratings from the list to each one. The optimum number of *Third Trusted Parties* as well as the size of the group of items that is required to provide a certain level of security is a subject for further research.

4.6 Recommendation Search

Once the Trust graph has been shaped by the primary Trust relationships, *Recommendation Search* queries can be serviced. Such queries can be initiated by any peer that is member of the Trust overlay or, in other words fulfils the requirements needed for establishing relationships with other counterparts. Such requirements include the minimum number of experiences that each node possesses as well as the common experiences between the two counterparts. Chapter 5 describes in more detail. The purpose of this kind of query is to make those entities that are known to have useful experiences for the query originator reachable via the Trust overlay, in order to derive their trustworthiness. Special rules can be applied as regards reachability, for example the distance in hops from the originator to the candidate entities that will contribute with their own experiences in the rating prediction.

We assume that a flooding scheme, through which Trust queries propagate in the Trust graph up to a pre-defined hop distance, will be used. As we will show in chapter 7 there is an optimum hop distance to which it is advisable to propagate queries. It is worth mentioning the requirement for the existence of common purpose [22] in the relationships in order to make it possible for users to employ transitive Trust using the graph. In our analysis we have made the assumption that this requirement is fulfilled

along a path and some entity which has been recommended for some context is also good at providing recommendations for the same context.

Valid Trust paths starting from the query originator and ending at the target node, if that is reached, will be sent back to the originator directly. The process must be done this way and not otherwise in order to let the originator understand the real topology that has to be analysed as the final structure and thus avoid cases where hidden topologies might exist.

Various quality restrictions can be set when initiating queries, for instance to disclose those entities with a low number of experiences or those which do not appear to be trustworthy. In the latter case the queries stop propagating further when a participant with controversial trustworthiness is encountered during path exploration.

This policy can also reduce the number of unimportant links in the resulting graph and thus help in keeping the derived Trust calculation times low. Special care must be taken so that there will be a balance between the number of vectors returned and the restrictions set, in order to avoid cases where no results at all are produced. The last case we mention is related to a notion we call coverage ratio (see Chapter 6), which is the number of services that can be reached by a particular user divided by the total number of services that the system can provide ratings about.

As we will see in Chapter 6, analysis shows that using filters for Trust query propagation has no serious impact on the error in the predicted recommendations. Even though the range of filters used in the tests is not wide, the result suggests the application of stronger filters since in this way traffic is kept low due to the simpler structures communicated as query results.

As search queries go deeper, so does the coverage ratio – the number of entities reached – but the resulting exponential increase in the number of messages due to using a flooding protocol, impacts scalability. As we will see in Chapter 7, the application of filters influences the user's satisfaction due to the high impact on the response times.

Upon receipt of the Trust vector replies, the originator should from then on maintain a collection of Trust vectors which constitute a graph leading to the distant node. Then the derived Trust of the target node can be easily calculated by parsing and analyzing the collected graph. This is precisely where the *Recommendation* and *Consensus* operators of Subjective Logic must be applied in sequence to the graph. A loop simplifying one vector at a time until the remaining structure becomes simplified

down-to a single vector may be the proper algorithmic approach. The reason the resource greedy task of parsing the entire graph is done by the querying entity itself, even though it could be done by the intermediate entities in the form of cacheing, is that in this way it preserves the dependence avoidance requirement in the calculation of Trust. This is known in the literature as the *Dependency Problem* in Trust derivation [34]. We also referred to the dependency problem in Chapter 3.

4.7 Recommendation Generation

Finally, Recommendation Generation comprises the collection of all Derived Trust query results from the previous step. The results are in the form of Trust statements expressed in triples of belief, disbelief and uncertainty. Using a transformation formula that we present in the next chapter, each of the secondary Trust measures can then be turned into similarity measures, expressing in this way how similar the querying entity appears to be with each of the entities that have some experience with the product in question and that were found through the previous step.

Knowing how similar each pair of entities is, makes it easy to predict the rating that the querying entity would give to a particular product. The following steps are the same as those that can be found in a plain Collaborative Filtering system. The actual prediction of the rate that a querying user would give to a particular product can be approximated using a suitable formula. For example, the Grouplens Recommender System uses Resnick's [11] prediction formula. In its general form is:

$$p_{a,j} = \bar{u}_a + k\sum w(a,i)(u_{i,j} - \overline{u_i})$$

Such a formula requires the average rating of the recommendee be known as well as the weighting factor w, which in our case is the similarity measure between the recommendee entity *a* and the series of entities *i* which have experience of the product in question. Also needed are the ratings of other products that the *i* entities have tested, as well as the average rating of every other entity *i*. We assume that such information is provided and processed by the system with a mechanism similar to that of the *Trust Discovery* phase.

The prediction formula returns a value *p* which, if the algorithm is applied to a series of products, can give results in the form of the top X products for the user *a*.

4.8 The network protocol explained

In this section we will explain the protocol used in the *Recommendation Search* phase for building the dynamic Trust graph. In particular we explain how the requests and the replies are communicated between the nodes involved in the Trust overlay. By doing this, we try to give a rough idea of how the graph is built by explaining schematically the messages that are communicated between the parties involved.

We Call A the trustor entity which is searching for information about a trustee B. We assume that the trustor has decided that B is the right place to ask for product ratings by running an information searching query prior to this phase and thus being informed about the existence of B. A initiates a query to its neighbours which replicate the message and forward it to their neighbours, up to a predefined search depth k. Once the message has reached its destination (trustee B) a reply is sent back to the originator A directly. Each reply message should contain all the path associated information which in this case would be the Trust vectors from all the intermediate nodes encountered from the source A to the destination B. We can imagine each vector of information as a triple of (*Source, Destination, Trust value*) which symbolises the Trust that the Source places on the Destination entity. In the case where we use Subjective Logic in the processing of opinions we can see the Trust value as a triple of numbers.

The time taken from Trust query initiation to the time that the propagated query reaches the trustee B through the intermediate nodes is called *Propagation Time.*

The time taken for all replies to be collected by all the various nodes is called *Collection Time.* The reason why more than one reply might be returned to the trustor A is that there can be several different paths to B. The fact that replies are sent asynchronously to the trustee suggests that the trustor should wait for some time for a sufficient number of replies, if not all, to be received. For this reason in a real implementation of the protocol an appropriate threshold must be set in order to avoid trustor A waiting forever before going to the next step. The collection time varies significantly from query to query depending on how many different paths exist between A and B and also from the capacity of the network links as well as the available bandwidth of the intermediate nodes. Given that the system runs over a

Peer-to-Peer infrastructure the utilization and the bandwidth of the lines used by the users that take part in the scheme have to be considered.

In a performance analysis we did of the model, which can be found in Chapter 7, we consider the capabilities of the network and we study the system performance for users using lines of low capacity (Modems) to connect to the Internet and also the case where nodes of higher connection capacity are used (DSL connections). This comparison was made specifically to show how the protocol is affected by current and future technological developments.

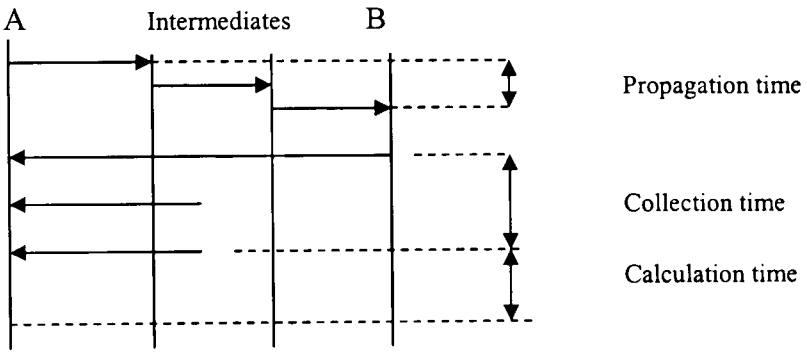

Figure 6. The Request-Reply protocol for the collection of the opinions

Even though the network infrastructure affects the Propagation and the Collection time, the Computation time for the graph analysis is dependent solely on the available computing power of the trustor node A and how busy the trustor is or, in other words, the number of search queries running at the same time in the node.

Figure 6 explains roughly the communication operations that take place in a typical Trust query scenario.

In order to test the scalability of the protocol which implements the Trust transitivity we performed the simulation experiment presented in chapter 7. The purpose of this experiment was to validate the mechanism of derivation of secondary Trust and especially the performance implications that it might have on usability. In this simulation experiment it was assumed that Primary Trust has already been built when the simulation began. In the study the cost in terms of resources due to the Primary Trust formation is consider negligible.

In addition, Chapter 6 presents a statistical evaluation that demonstrates the benefits that can be achieved when knowledge from both the similar and the trusted

participants is used for predicting user preferences and is compared to using similarity alone. In that experiment the arithmetical calculation of secondary Trust was performed in the same way as in a real implementation. The time taken for Trust to be computed was also measured in this experiment and used as an input parameter to the scalability evaluation in chapter 7.

4.9 Summary and Conclusion

In this chapter we gave an outline of a proposed distributed architecture for Peer-to-Peer recommender systems, which can be seen as having the form of a virtual Trust over-net. The formation of the Peer-to-Peer overlay is the key to recommendation production and this is where the Subjective Logic algebra rules we presented in the previous chapter can be applied.

One of the major differences of such an architecture in contrast to the conventional one is the use of distributed data ratings instead of keeping them in a central place, which makes it more secure and resilient to attacks as well as leading to a more robust scheme. The basic idea explained in simple terms is that each node keeps its own personal experiences which it shares with a selected group of other nodes with which it has some similarities. From those similarities is possible to calculate Trust measures between the "*similar*" members. The application of such Trust measures in a whole community forms a network known as "*web-of-trust*". By calculating Trust relations that meet the criteria of transitivity we can transform them back to similarity metrics and thus get able to answer queries for which there was insufficient evidence for a local user to work out.

We also presented a simple protocol that implements this architecture and described the steps that must be followed in order to run this protocol to a real environment. In the next two chapters we get into more detail about the way that primary Trust relationships are formed and how beneficial the new query mechanism is for the whole community.

Chapter 5

THE TRUST DISCOVERY PHASE

5.1 Introduction

In this chapter we describe the first set of operations that occur in a Recommender system, during the *Trust Discovery Phase.* This phase is concerned with the representation of Primary Trust relationships and the formation of neighbourhoods of users with respect to the level of Trust they place in each other. This phase must be implemented by every entity that wishes to participate in the Trust scheme. The challenge for those entities that have decided to participate in the schema is how to turn first hand evidence into Trust measures. This requires a transformation formula which, in our case, is adapted to a Recommender System where first hand evidence is in the form of ratings that users have given to items.

In this chapter we present our technique for modelling Trust relationships by using first hand evidence produced in recommender systems' environments. In such an environment agents collaborate with each other with the common aim of providing accurate recommendations to each other. First-hand evidence is any data concerning primary Trust that an agent has derived from its own experiences with the examined entities. Second-hand evidence consists of Recommendations from other agents. It is so called because the derived Trust (secondary Trust) from these Recommendations has not been derived from the agent's own experiences with the entities in. In our model we use techniques taken from Collaborative Filtering to express Trust properties as beliefs. We also validate our model by comparing the results with those from existing models for mapping types of behaviours to Trust values based on the Beta distribution. Finally, we describe the requirements of a protocol that could be used for the deployment of the proposed model in a real distributed environment.

5.2 Trust derivation from evidence

As shown in the previous chapter, evidence can exist in various forms. In a system that makes use of Trust, the form of the evidence must be in a form that the Trust model requires.

A Subjective Logic based model requires that Trust be expressed in the form of opinions, triples of belief, disbelief and uncertainty. In Recommender systems instead, evidence usually exists in the form of ratings that users have given to items and this makes a transformation formula necessary. In Collaborative Filtering systems statistical techniques are employed to develop virtual relationships between users. In this way, neighbourhoods of users consisting of those who have a history of agreeing in the past, and thus are considered to be similar, can be formed. Although a similarity relationship can easily be expressed for any pair of entities, this type of relationship is not transitive whereas Trust relationships can be [22], under appropriate circumstances with regard to a common Trust purpose.

The necessity for approaching Trust modelling in a new way leads to the idea of expanding the neighbouring base of the users by using Trust relationships that could have been developed between them so that it is possible for other members of the community to be reached through them. Sparsity in the Recommender systems is the main reason for poor behaviour because not much evidence can be gathered to support a Recommendation. It appears mainly because users themselves are not willing to invest much time or effort in rating items.

For such an idea to become applicable, users must somehow be able to place Trust in their neighbours. In the usual centralized consumer opinion sites [16] a requirement is that this Trust measure should be provided by the users themselves. In other words, users should be able to set the Trust values by giving a measure of how much they Trust other entities. Moreover, this means that users need to have developed a good instinct for judging things and this cannot be guaranteed. Poor judging abilities lead to the danger of establishing relationships with the wrong counterparts. Trust on the other hand is not easy to estimate and it is not always possible to assure that the people that are behind the machines are always capable of doing this, even if the evidence they have would be sufficient for some other person. In other words it is a personal issue of how much evidence is sufficient for a certain person to become able to build adequate Trust with another.

Our contribution to this issue is to introduce a technique for mapping between similarity measures and Trust. In other words, by knowing how similar two entities are, they will be able to estimate how much Trust they should place on each other. Also in reverse, knowing how trusted two entities are, will be sufficient to derive a measure of similarity for them. The reason for trying to use both forms of correlation between users is because each type has its own advantages. For example, similarity measures are suitable for use in Collaborative Filtering systems since predictions can be made using them and Trust measures have transitive characteristics under certain circumstances.

5.3 The basis of our model

In our model we use ordinary measures of similarity taken from Collaborative Filtering in order to form the potential Trust between the correlated entities. This Trust would be propagated using a method identical to a "*word-of-mouth*" scheme in the graph built up by the relationships between the entities. The aim of our Trust model is to enable some entities in the graph to find out how similar they are even if there is no direct similarity relationship between them.

The Trust that an entity should place on a distant one will be derived through the Trust graph. Ultimately, the transformation of the value back into a similarity measure, could make it appropriate for use in Collaborative Filtering schemes.

W express Trust in the form of opinions as they are modelled in Subjective Logic. As we mentioned in the previous chapter, Trust is considered a subjective measure and this introduces the important idea that there is always imperfect knowledge when judging things. Imperfect knowledge is expressed with the notion of *Uncertainty*. Subjective logic [8] uses a simple intuitive representation of uncertain probabilities by using a three dimensional metric that comprises belief, disbelief and uncertainty. Even though Opinions in the form (b,d,u) are more manageable due to the quite flexible calculus that opinion space provides, evidence however is usually available in other forms that are more understandable by humans.

As mentioned in Chapter 3, Trust is context specific. It is also related to tasks in a sense that entities are trusted to perform a particular task. A simplistic approach would be to determine the levels of Trust and distrust that should be placed on some entity from its probabilistic behaviour as seen from the trustor's point of view.

The *Beta Distribution Probability Function* can offer an alternative representation of uncertain probabilities [26] making it possible to approximate Opinions from behavioural data. However, data in that evidence space are considered as sets of observations and therefore they must be provided strictly in binary form representing the possible two outcomes of a process x or $\bar{x}$. These can be thought of as a positive and a negative possible outcome. So, a behaviour is described by the value of x or $\bar{x}$ that derives from the set of observations. As we saw in Section 3.16, the Beta Reputation system is based on using beta probability density functions to combine feedback and derive reputation ratings.

As can be seen from the formula in section 3.16 that give the values of b,d and u, the Opinion properties (b,d,u) are solely dependent on the number of observations characterised as good or bad. These parameters though, are meaningless in a Collaborative filtering system where the available data are nothing but ratings that users have given to items.

The Beta reputation system requires that there are processes with two possible outcomes to estimate the probability of positive outcomes in the future. As can be seen it is oriented to a special type of problem where outcomes can be categorized as "good" and "bad" behaviours. For other kinds of evidence a different mapping is necessary. The existence of data in the form of "good" and "bad" behaviours is difficult to find in existing datasets or to expect people to provide them in that form. In that case, some kind of criterion is required for the transformation of data from some usable form to the appropriate beta distribution values. Opinions or Trust values, on the other hand, do not have any meaning for people if they are asked to provide them.

In contrast, other similarity based approaches such as that in [33] are based on the idea of getting the users linked together indirectly using predictability measures, but these have not been tested in real environments.

The principle idea of a Reputation system is that past experience with remote transaction partners can be projected into the future, giving a measure of their trustworthiness. This effect has been called the "*shadow of the future*" by the political scientist Robert Axelrod [59].

5.4 Our Trust model

In general, Trust models are used to enable the parties involved in a Trust relationship to know how much reliance to place on each other. Our model aims to provide a method for estimating how much Trust two entities can place on each other, given the similarities between them.

As we have seen, the problem with Recommender systems is that the entities involved provide their views about other entities in the form of ratings about items and not in the form of Trust estimates about other entities. This means, making the model Trust-enabled requires that all this info, which so far has been expressed in the form of ratings, should be transformed into Trust values.

In order to do this, we consider the ratings that users have given to items as the behavioural data required for the composition of their opinions.

In our model we assume that the level of Trust that develops between every pair of entities is based on how similar each other's choices seem to be as they see it. In other words, the Trust values derived are based on similarity of choices. We use the Pearson coefficient to express the similarity measure, as this is the best known and most suitable coefficient for this type of application. It takes values between –1 and 1 and two entities are considered to have high similarity when their Pearson values are close to 1 and are completely dissimilar when the value is –1. A value of 0 would mean that there is no relationship between the two entities at all.

Unlike the Beta distribution mapping to Opinions which we described above, in our model we describe Uncertainty by using both *Quantitative* and *Qualitative* criteria to extract information from the evidence. This is because we believe that, unlike Beta function based modelling, in the physical world the number of experiences required to define uncertainty is variable. The reason is that some people need different numbers of experiences from others in order to reach the same level of certainty in a Trust relationship they develop with a counterpart.

5.5 Modelling Uncertainty

As with the Beta distribution, the way of applying quantitative criteria obeys the rule that Uncertainty should be inversely proportional to quantity of the evidence. As regards the quality of the data we re-define the perception of Uncertainty as the inability of some entity to make accurate predictions about the choices of the other party in the Trust relationship. A low ability value would be the result of the existence of conflicting data making the observer unable to fill in its own Uncertainty gap. Such a combination of low quantity of data and inability to make accurate predictions would lead to a high Uncertainty value. In the case that there are not enough observations to distinguish rating trends, data might appear to be highly conflicting. Uncertainty would also be high in the case where there are not sufficient data to support an opinion.

Bearing in mind the idea that those entities whose ratings can be accurately predicted should be considered as trustworthy sources of information, the Uncertainty in such relationships should be lower. Therefore we propose the following formula to model Uncertainty from prediction error:

$$u = \frac{1}{k}\sum_{x=1}^{k}\frac{|p_x - r_x|}{m}$$

where k is the number of common experiences (ratings) of the two entities that take part in a relation, p_x is the predicted rating of item x calculated using a prediction calculation formula and r_x is the real rate that the entity in question has given to item x. The quantity m represents the maximum value that a rating can take and is used here as a measure of rating.

The logical reasoning for deriving the above formula for Uncertainty is the following: Uncertainty is proportional to the prediction error for every user's single experience; therefore the nominator represents the absolute error between the predicted value (using a rating prediction formula) and the real (rated) value. The dominator m has been used for normalization of the error in the range 0-1. The sum symbol has been used to include all the experiences (k in number) of the particular user. Finally, the result is divided by the total number of experiences (k) to get the average normalized error.

In the sum we take every pair of common ratings and try to predict what the rate p would be. Every time we perform the prediction by assuming that all but the real rating of the value that is to be predicted exists.

A suitable prediction formula that can be used for calculating p_x is that of Resnick's [11]. In our approach, for every pair of common ratings we calculate the Pearson correlation coefficient and then we use it for the prediction of the rating that we have kept hidden and that we are trying to predict it.

Resnick's prediction formula requires that the Pearson correlation coefficient w(a,i) (which expresses a measure of similarity between a and i) , and the average rating of the users involved should be known. The correlation value is used as a weight for the user ratings, according to the intuition that, if another user rates in a way similar to the current user, then her ratings are useful for predicting the ratings of the current user. $p_{a,j}$ is the predicted rating of user a for product j, $u_{i,j}$ is the rating of every other user who has experienced j in the past and $\overline{u_i}$ is the average rating of all products that the u_i user has rated.

$$p_{a,j} = \overline{u}_a + k\sum w(a,i)(u_{i,j} - \overline{u_i})$$

The pairs of common ratings in this case play the role of the evidence, which are k in number. This means that, a calculation of uncertainty requires k times the calculation of prediction which requires k^2 times the calculation of similarity value. We can say that an order of magnitude of 2 for k common items for a calculation of a single opinion is a weak point of the algorithm. This is because normally the value of the item that is to be predicted must be excluded from the calculation of similarity and only the other pairs of values should be used. Nevertheless, the predicted values $p_{a,j}$ can be accurately calculated by using the similarity value that has been derived by the whole set of pairs. In the case that the number of pairs of items involved in the calculation is high, the deviation by using the same similarity value is negligible.

Whenever a new pair of ratings is added into the collections of two related users their similarity must be recalculated since new evidence may have turned up. A formula that will avoid the recalculation of the whole set from the beginning whenever a new experience is added to the set of experiences of each counterpart is left as a future research problem.

As can be seen from our formula, Uncertainty is inversely proportional to the number of experiences k. This agrees with the targets we set as requirement for Uncertainty regarding the quantitative criteria we discussed in the previous paragraph. Unlike the mapping based on the Beta distribution function where uncertainty exclusively tends to 0 as the number of experiences grows, in our model this tension remains quite vague because uncertainty is now also dependent on the average prediction error. In the extreme case where there is high controversy in the data, u will reach a value close to 1, leaving space for the other elements of the opinion, (belief and disbelief), to develop their values. In some extreme case the predicted and the actual rated values match each other: this happens when both counterparts have given the same ratings for all the same items. This is what makes Uncertainty equal to 0 and belief equal to 1. In the interesting case where the correlated entities have given opposite ratings, for example, what A has rated with high value, B has rated with low value and vice versa, thanks to the prediction formula, this will also give values equal to the rated ones and the uncertainty will get its lowest value of 0 again. But in this specific case the rest of the opinion gap will be filled with disbelief which will have the value of 1.

Another interesting characteristic of our model is the asymmetry in the Trust relationships produced which adheres to the natural form of relationships since the levels of Trust that a pair of entities place on each other may not necessarily be the same. This comes from the fact that Trust properties according to our model are derived from ratings, which have not necessarily taken the same values in both sides of a relationship.

5.6 Modelling belief – disbelief

As regards the other two properties b (belief) and d (disbelief), we set them up in such a way that they are dependent on the value of the Similarity metric $w_{a,u}$. In our model we have used Pearson's Correlation Coefficient as a similarity metric, which introduces the idea of computing the similarity between the two users, which is one of the standard steps in Collaborative Filtering techniques. The Pearson Correlation Coefficient has the form:

$$w_{a,u} = \frac{\sum_{i=1}^{m}(r_{a,i} - \overline{r_a})(r_{u,i} - \overline{r_u})}{\sqrt{\sum_{i=1}^{m}(r_{a,i} - \overline{r_a})^2 \sum_{i=1}^{m}(r_{u,i} - \overline{r_u})^2}}$$

where: $\overline{r_a}$ and $\overline{r_u}$ are the average ratings of the users a and u respectively, $r_{a,i}$ is the rating of user a for item i and $r_{u,i}$ is the rating of user u for item i.

The correlation coefficient is computed only for items common to both users.

For the two properties belief and disbelief we use the following formulae:

for belief

$$b = \frac{(1-u)}{2}(1 + w_{a,u})$$

for disbelief

$$d = \frac{(1-u)}{2}(1 - w_{a,u})$$

As can be seen from the derived formulae b + d = (1 – u), which conforms to the *Belief Function Additivity Theorem* of Subjective Logic. The ratio of belief and disbelief is shaped by the Correlation Coefficient (or Similarity) value. In this way, a positive Correlation Coefficient would be expected to strengthen the belief property at the expense of disbelief. In the same way disbelief appears to be stronger than belief between entities that are negatively Correlated ($w_{a,u}<0$). According to this model two entities which have given similar ratings to the same items and thus behave similarly in their rating behaviour, will have stronger belief than disbelief in the Trust relationship between them and therefore they will Trust each other more.

The two formulae can also be used in the opposite way, so as to estimate how similar (expressed by w) the two entities should consider each other given their Trust properties. As can be seen from the previous expressions, any two of the Trust properties would be enough to derive the similarity between the two counterparts.

This asymmetry in the relationships is mainly responsible for unequal similarities in the normal and the reverse relationship. Also responsible for this difference are the differing points of view, and the formula included in the calculation of uncertainty, which is used to work out the predictions p_x in the proposed technique. Formulae identical to Resnick's empirical formula for the Grouplens Collaborative filtering system can be used for the purpose of predicting values.

5.7 Evaluating the model

As explained above the modelling we performed for the properties of Trust was done on an empirical basis, so we considered it appropriate to do a comparison of our modelling technique with other potential methods of modelling Trust Opinions using evidence.

In this section we present experimental results in the form of a comparative study that shows the accuracy of our modelling technique. For the evaluation of our Evidence to Opinion mapping method, we tested it against a model that uses the Beta probability distribution function. Even though a model of evidence based on the Beta probability density function would be inflexible and, as we explained, would not always feasible, we attempt a comparison between this and our model and demonstrate how close the results for both schemes appear to be and also under what circumstances. Note that modelling in Beta is not always possible because it is dependent on a subjective view of the data, which may lead to different interpretations by various people. As we will see next, the choice of the statistical measure we used as a criterion in evaluating an elementary piece of data in the Beta function was strictly our own.

For the testing we used a dataset taken from a real Collaborative Filtering system known as *MovieLens*. The main reason why we chose this system was because the dataset was publicly available and thus our method can be evaluated by anyone who wishes. MovieLens [60] is a movie recommender system based on Collaborative Filtering established at the University of Minnesota. The whole dataset is publicly available and contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 users who joined the service in the year 2000. As stated before, the Beta distribution function requires that behavioural data should be expressed in a strictly binary form referring to the two possible outputs of a process: satisfactory or

non-satisfactory. This characteristic makes this type of coding very restrictive when applied to a small set of applications that can give binary output exclusively unless there is a way of converting available data to their binary equivalents. Given that we have to deal with Recommender systems in which ratings are expressed by continuous values or in numerical discrete alternatives, this makes the model inflexible.

In our case, in the dataset used for our experiments the ratings were available in discrete values ranging from 1 to 5. For the sake of our evaluation we restricted the test to a subset of the MovieLens database based on 100 users only. In total the testing dataset comprised 12,976 ratings provided by this subset of users. The analysis we performed on the dataset showed an asymmetric distribution of ratings with mean=3.61, standard deviation squared=1.24, median M=4 and a skew to the left. The value 4 for the median can be explained by the fact that people tend to be kind when they rate things they have experienced themselves and therefore almost always give a higher rating compared with what they actually think.

We faced two challenges when carrying out this experiment. First how to make the experimental dataset suitable for representing evidence for the Beta distribution and second what measures to use for the comparison.

The fact that no data supplied by users, showing how much they Trust each other, were given in an appropriate form, led us to generate artificially the weights that should be placed on the relationships. In Beta modelling the evidence should be provided in binary form x or $\bar{x}$, to represent how an entity would perceive the behaviour of the other party in the relationship.

To judge the rating of every single item with respect to its behaviour, we defined our own criterion. The basic idea of this is the use of the relative distance between a pair of ratings given by two users as a condition to decide whether some behaviour is characterised as good or bad.

Let us call $R_{A,k}$ the rate that the first user A gave to item k and $R_{B,k}$ the rate of the second user B to the same item. A relatively long distance between the $R_{A,k}$ and the $R_{B,k}$ should be considered – subjectively judged – by A or B as unsatisfactory behaviour of the other counterpart. As it can be seen, such a rule requires a criterion that would define when a behaviour would correspond to x or $\bar{x}$.

In our experiment we used the Median value as the barrier for characterising a behaviour as bad if the two ratings have been placed on different sides. For the same purpose a different metric from Median could be used to separate one area form

another could be used, but we believe that the Median is the most appropriate because it divides the sample into two parts with an equal number of item ratings.

For example, a case where $R_{B,k}$=3 and $R_{A,k}$=5 should be considered as $\bar{x}$. The median reflects the way that users rate items. Finally, we choose one of the four possible scenarios of the table in Figure 1., that characterise a behaviour.

	$R_{A,k}>M$	$R_{A,k}<M$
$R_{B,k}>M$	x	$\bar{x}$
$R_{B,k}<M$	$\bar{x}$	x

Figure 1. Truth table of Evidence

Once all the pairs of common ratings have been examined, we transform the evidence to opinions (b,d,u) by using the Beta transformation equations. (See section 2.21).

The kind of data representation, which we used in the Beta model in the comparison test, in order to be successful and produce correct results, requires that the sample dataset be static, with known statistical values, and not dynamic as in a real world scenario. Therefore its use is not recommended for real environments.

For the sake of the evaluation tests we built a graph of the 100 randomly chosen users from the MovieLens database, which constituted 8782 Trust relationships and the results are given in Figure 2 below.

In order to be able to compare the Opinions created by each model, we converted them into a plain probabilistic value, which by convention is called the Probability Expectation. This can be interpreted as saying that the relative frequency of both counterparts in the relationships agreeing in taste is somewhat uncertain and the most likely value is $E_x = b + a \cdot u$. a represents a measure called *Relative Atomicity* which is used to describe if, and how much bias, there is in the system towards belief or disbelief. A formal definition of Probability Expectation can be found in [8]. In the experiment we measured how close the two derived opinions from the two methods are by comparing their probability expectations. Therefore, the values shown in our results are in terms of this measure.

In figure 2 we present the divergence between our modelling and the Beta probability distribution function, the measurements being derived from the relative

difference (%) between the two probability expectations. In this table, the results have been grouped for various classes of common experiences that constitute an opinion, to show how the number of experiences affects the divergence between the two types of modeling. The second column indicates how many relationships from the dataset belong to each class, based on the number of common experiences.

5.8 Discussion of the results

Figure 2 shows increased divergence between the two modelling techniques for small numbers of common experiences. This can be explained as the result of the noisy behaviour of the Correlation Coefficient. In these categories the quality of predictions is quite uncertain.

Class	Num of Common experiences	Sample size	Mean divergence (%)	Standard deviation (%)
1	[2-3)	624	17.50	8.64
2	[3-5)	1326	16.67	8.17
3	[5-10)	2277	13.77	7.67
4	[10-20)	2191	11.29	7.03
5	[20-40)	1417	10.10	6.88
6	[40-60)	471	11.22	7.33
7	[60-80)	195	11.98	7.75
8	[80-100)	101	12.90	7.68
9	[100-150)	115	10.64	7.71
10	[150-200)	46	10.52	8.49
11	[200-250]	10	11.26	7.81

Figure 2.a. Mean and Standard deviation of divergence for various numbers of common experiences

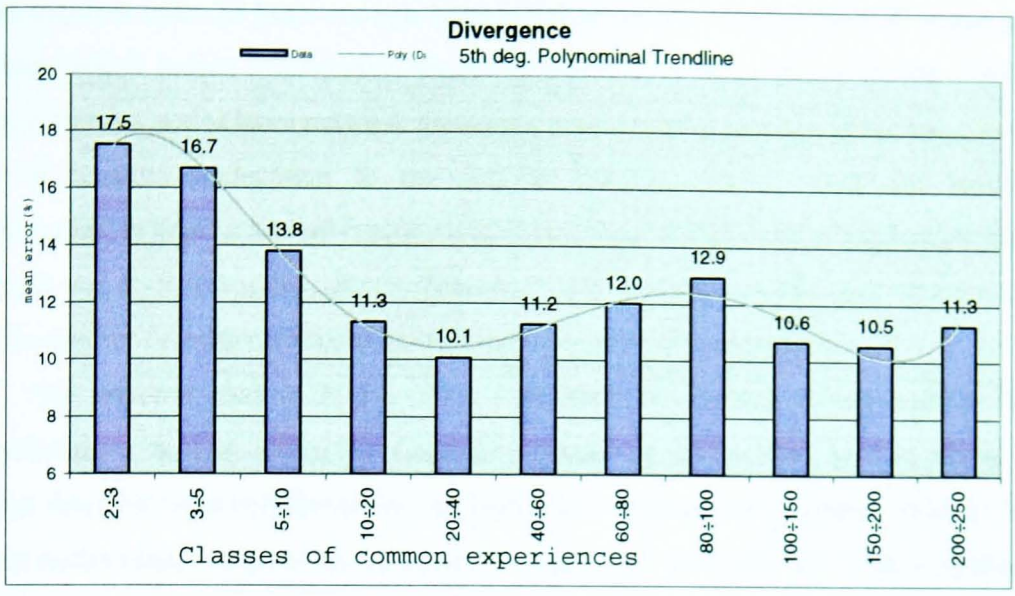

Figure 2.b. Divergence . Mean and Standard deviation visual representation

The small increasing trend in divergence that is noticeable as the number of common experiences grows (2-10) is due to the poor data set used in those categories, since only a very small sample size existed in this class of common experiences.

From the results it can be seen that our coding method converges to Beta modelling when the agents have at least 20 common experiences, at which point the divergence stabilizes to a relatively low value.

Even though both methods give slightly different results (differing by around 10%), there is no real situation which could be used as a point of reference to evaluate how accurate, in absolute measures, each coding method is.

Having this evaluation experiment as a guide we can decide not to use primary Trust relationships between users whose number of common experiences is less than 20, because in this range the proposed method's behaviour diverges significantly from the Beta modelling which we use as a point of reference.

5.9 Application to a P2P network

In this section we discuss how the Trust Discovery Phase can be established in an existing user community. Primary Trust establishment requires that users exchange information with each other regarding their past experiences of items.

This requires users to have enough knowledge about the communities to which they belong, which means information about the topology is somehow available. Peer-to-Peer networks nodes have network discovery implemented as a low level operation so that it remains transparent to the services running above. Thus the topology information is kept in a local registry which the Trust establishment mechanism could consult and so find appropriate neighbours. Various criteria can be used to determine the suitability of each neighbour such as hop distance, longevity etc.

We assume that a Peer-to-Peer infrastructure already exists before Trust establishment begins. Using the Gnutella network as an example we can show that nodes discover their neighbours by analysing the messages they receive (pongs) from those nodes which respond to the network exploration polls (pings). This information is kept in their local registry to be used in other operations such as resource searching.

In this way Trust discovery, which can run on top of network exploration, may be restricted only to near neighbours which are within a radius of a few hops of the querying node. This results in reaching and establishing Trust with, usually, a few hundred nodes, which is sufficient. Having in mind that the establishment of the relationships would be done using a request-reply scheme, congestion problems can be partially alleviated if the receiving side ignores any Trust establishment requests for which the relationship has already been built and thus send no reply at all.

In case the above scenario has not led to a large number of trusted participants the requesting node could then retry the discovery process using an increased hop count.

The calculation of Primary Trust can be done in the way explained in the previous chapter bearing in mind the security issues. Thus both the requesting and the replying entity could then supply their ratings from which the Trust value can be calculated.

5.10 Future Work

We intend to apply our modelling to a real system with the expectation of improving the quality of the derived recommendations. Another idea, which we will see applied in the next chapter, is to make use of the *"web-of-trust"* that would evolve from the establishment of direct Trust relationships between users. Our aim is to improve the recommendations provided by exploiting the experiences of any entities not

neighbouring the originator of the query but reached via the *"web-of-trust"*. This requires the calculation of derived Trust relationships from the primary Trust values. The question that arises from this is how accurate the predictions can be. Obviously there will be some increase in the coverage, which translates into reduced sparsity in the dataset of opinions that can be expressed.

The ad-hoc way we chose to code the positive and negative evidence for the Beta distribution necessitates more tests against other alternative coding techniques and the use of different statistical measures. (For example using the Mode instead of Median value as a criterion for separating good from bad experiences).

No matter how good recommendations such an architecture can provide, there are security weaknesses for the Recommender Systems, which must also be tolerated.

5.11 Summary and Conclusion

We presented an empirical technique for modelling the trustworthiness of entities using evidence that describe their rating behaviour. The novelty comes from the shaping of the derived Uncertainty which is dependent on a predictability measure and thus on the value of the evidence. We coded our derived Trust Opinions into metrics taken from Shafferian belief theory and we attempted an evaluation of our model against an identical one based on the Beta probability distribution function for mapping evidence to Opinions.

The evaluation was done experimentally by taking data from an existing database and processing the data using a formula which could transform them from their simple form to a compatible type that could be identified by both methods.

The use of the Median as a criterion for the identification of the distance between the two ratings and thus the characterisation of good and bad behaviours was mainly a personal choice. This seems to be a weak point of the evaluation method but it could not be avoided due to a lack of other transformation methods.

From the evaluations it appears that both methods produce very similar results. The strong points of the proposed technique can be summarized as its ability to incorporate similarity measures in its properties, its use of qualitative as well as quantitative measures to derive Opinions and its flexibility in accepting datasets of continuous values rather than just binary, which makes the method suitable for the Collaborative Filtering type of Recommender Systems. The weak point is the lack of

security in the transmission of Opinions which creates a security threat in the system. We leave this as a future research problem.

Chapter 6.

THE RECOMMENDATION SEARCH MECHANISM

6.1 Introduction

In this chapter we describe a method that can be used for carrying out the Recommendation Search referred to in Chapter 4 as one of the phases that must be implemented in a distributed Recommender system. The main idea is based on the use of Trust relationships to support prediction of user preferences. The method is presented here in its simplest form and can also be run in a centralized environment. It can, however, easily be extended to support distributed environments. It also has another interesting characteristic, which is the avoidance of the sparsity problem and for this reason helps in providing improved quality Recommendations.

This chapter deals only with the derivation of secondary Trust between the trustor and the trustee entities, which are both members of a given graph. After the completion of this phase the system is ready to provide Recommendations in the usual way. The exploitation of the recommendation search phase is worthwhile when there is a considerable hop distance between the trustor and the trustee and thus they can form a network through which secondary Trust is calculated.

Trust derivation is done using the *Consensus* and the *Recommendation* operators of Subjective Logic algebra and is especially used for the simplification of the Trust graphs formed. The type of Trust that is calculated through a Recommendation search operation is the type we called Indirect. For more details on types of Trust see Chapter 3.

As we discussed in Chapter 4 a typical neighbourhood formation scheme uses *Correlation* and *Similarity* as measures of proximity. With this approach, relationships between members of the community can be found only in the case that common experiences and common purpose exist.

We intend to exploit information, which at first glance may seem to be extraneous, in such a way that might be beneficial for the community. In a Recommender system this benefit appears as improved accuracy as well as improved

capability in providing Recommendations. In the previous chapter we made use of the common experiences that two entities might have, to establish hypothetical Trust relationships between the entities. Through such relations the entities will be able to find how much they should Trust other entities that are logically positioned a significant hop distance away.

This chapter also contains a validation of the proposed idea. An experiment is performed in which evaluation results are presented using statistical measures. More specifically, we measure the performance of a system that incorporates this idea and especially:

- The mechanism that converts Similarity to Trust and vice versa, and
- The concept of building a Trust infrastructure for deriving the Trustworthiness of non-neighbouring users.

Our approach is compared to the standard one which uses Similarity alone. In the evaluated system it is assumed that the derivation of primary Trust has been done in such a way that conforms to the limits set regarding the number of common experiences that two potentially trusted users must have and were presented in chapter 5. Assumptions have also been made regarding Trust transitivity.

6.2 Claims about performance

There are two challenges for recommender systems which appear to be in conflict with each other: *Scalability* and *Accuracy*. An approach to expressing a measure of performance in such system should take into consideration a combination of both challenges.

Accuracy has to do with how close the values predicted by the Recommender system, are to their real values. The calculation of accuracy involves an estimation method called *"leave-one-out"*. In this method a known value is kept hidden and the testing system tries to predict it. Although ratings already exist as personal assessments on things, we assume here that prior to a calculation these ratings do not exist and the Recommender system is trying to calculate their values using the algorithms. So, real values are considered as unknown throughout the evaluation.

Scalability has to do with how many different sources of data are considered in order to make a prediction. The more sources supplied, the worse the Scalability since

this demands a higher number of calculations to produce a result. In principle, sparse datasets require lower number of calculations and therefore such configurations scale better.

Accuracy is proportional to the amount of data that is available but appears to work at the expense of *Scalability*, since accurate predictions require more time for searching for sources of data. In this chapter we only deal with the first challenge, leaving the scalability issues as future work.

Sparsity is a characteristic of the Recommender Systems based on Collaborative Filtering. Such systems work using algorithms that are called *Nearest-neighbour* algorithms. Other technologies used include *Bayesian networks* and *Clustering*. In the latter, users are grouped by their similarity in preferences, and predictions are made regarding the participation of a user in a cluster. In the case of participation in multiple clusters the prediction is a weighted average. As shown in [19] algorithms based on clustering have worse accuracy than *Nearest-neighbour*, therefore pre-clustering is recommended. Pre-clustering is a graph partitioning method that is used to group objects of a sparse graph into sets of smaller clusters. It belongs to a category of clustering methods that are called *"Hierarchical Methods"*. For categorization of clustering methods see [85].

The basic idea behind Collaborative Filtering as described in Chapter 2, is to make predictions of scores based on the heuristic that people who agreed (or disagreed) in the past will probably agree (disagree) again. Even though such a heuristic can be sufficient to correlate numerous users with each other, systems which have employed this method still appear to be highly sparse and thus are ineffective in making accurate predictions all the time. This ineffectiveness is proportional to how sparse the datasets are.

By Sparsity we mean a lack of data required for a Collaborative Filtering System to work. In this specific case the data are in the form of experiences which users share with each other through the system. In a conventional centralized Recommender system, sparsity appears as a number of empty cells. In a system represented as a *"web-of-Trust"* graph, sparsity appears as lack of relationships between the nodes.

Our claim is that discovering Trust relationships and thereby linking users of the Recommender system together can have a positive impact on the performance of such a system. As discussed in previous chapters such linking of users is theoretically

possible and one can imagine that they compose a virtual network on top of the existing one: a Trust Overlay.

In a conventional Recommender System the main type of relation exploited between users is Similarity. Recommendations and the formation of groups of users are based on the similarity property of the entities that take part in the scheme. Our idea is that developing other types of relations in addition to Similarity between the users could help their connectivity base increase and thus their contribution to the system. Also, any other type of relation that can be mapped to similarity would be useful in increasing the connectivity base. Such an idea would also be beneficial to new users whose relationship group is poor and Similarity alone is not enough to help them exploit information from the collaborative environment. This is known as the *"cold start problem"* in collaborative filtering systems and we expect that our method will ease this.

In standard Collaborative Filtering, the correlation of ratings is done on a "*nearest-neighbour*" basis, which means that only users who have common experiences will be correlated. If we are only using Similarity metrics for doing correlations then a single node can benefit solely from knowledge and experiences which are within one hop distance. In Collaborative Filtering systems, whose datasets are characterised by high sparsity, this has as a result poor performance.

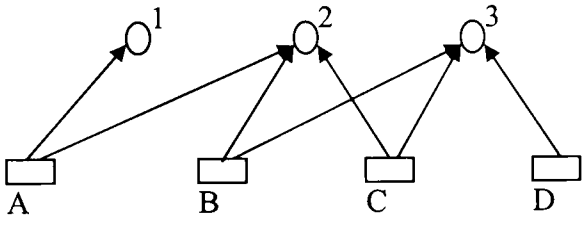

Figure 1.a

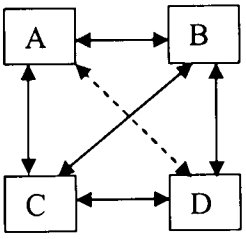

Figure 1.b

For example in the simple scenario in figure 1.a where 3 services are experienced by 4 entities when using the standard method there is no way that knowledge from entity D can reach entity A since no Similarity measures can be computed between them. We use the term knowledge for expressing the ratings that those entities have given about items tested in the past.

In Figure 1.a we call 1,2 and 3 the items for which experiences have been provided and A,B,C and D the user entities that take part in the example schema. As opposed to users A and D, users A and C can be correlated due to the existence of

common ratings between them and particularly with their personal experience of item 2. We can better show the relationships by using a graph such as that of figure 1.b. The denser the graph, the easier for an entity to collect the appropriate information and thus the higher the possibility of making a prediction. In addition, as shown in our experiments, the predictions due to the extra data collected are even more accurate.

Our idea is to exploit information from any experiences that can be reached beyond the barriers of the local neighbourhood for the benefit of the querying entities. We deal with this issue by utilizing any Trust relationships that could exist between the entities involved and in this way build a hypothetical Trust network within the system. In the above example by using Trust relationships in a transitive manner we can bridge the gap between the distant entities A and D of our example and find how trustworthy D looks to A and vice versa. From the derived Trust we can estimate the similarity of the distant entities and thus be able to predict their ratings of each other.

For long Trust graphs there is a problem with the length of the Trust chain and this raises the issue of how one can traverse a whole chain of intermediate nodes to find a Trust value for a distant one. In fact, there is a debate as to whether it is valid or not to consider transitive Trust relationships. Even though it has been said that Trust is not necessarily transitive [7], *Indirect Trust* is possible when various requirements are met. The existence of the same context along a chain, also known as Trust purpose, is crucial to exist and indicates the ability to recommend [22]. This ability comes from the way that Trust relationships have been formed.

Assuming that this ability is present in a long chain, then a Recommendation can be made, as Indirect Trust can be calculated along the chain. Hereafter, in this experiment we assume that the entities, in fact, do have this ability to provide Recommendations as soon as they appear to have a common taste for things. The basis of our hypothesis is that someone is considered a good recommender given when he/she knows the subject of Recommendation well. For example, someone who is known to be a good car engineer is also able (always assuming that he answers enquiries honestly) to judge and recommend someone else who is good at repairing cars. Or else, he can provide Recommendations about a service that are as good as how he himself provides the service.

Finally, from the Trust relationship we can express how similar the two entities in question might be and give one of them the opportunity to use the information from

which they derived their hypothetical Similarity. In Figure 1.b the Indirect Trust relationship would appear as an additional link between A and D (dashed line). In a system represented as a matrix of users by items it would affect its sparsity and in this case would decrease it.

6.3 Sparsity and Performance

Even though there are various techniques to help reduce sparsity, such as the Singular Value Decomposition technique [30], this is not always desirable. This is mostly due to the centralized architecture of most Collaborative Filtering and also because the number of calculations required for a prediction depends on the available data.

When the ratings are represented as a matrix of users and items, the number of calculations needed to make a decision is dependent on the product of users by items. As a result, if the prediction has to consider a significant number of ratings then a large amount of time will be needed.

Even though a query almost never involves more than a small number of users who have expressed a series of common ratings we must take into consideration the fact that at every single moment a quite significant number of queries may be running, which in a centralized system may be catastrophic. Given that, due to the Trust computation, there will be extra demand for resources because of the extra load for the handling of the new relationships that will arise, we believe that things might get even worse as regards computational load.

We conclude that such a requirement could make the Recommender system respond outside the time limits that the average user would accept. The systems we know about (e.g. Epinions , Movielens) appear to have sparsity levels that approach 97% which means that those services have almost empty matrices. Therefore conventional Recommender systems should not be considered as scalable.

As such the reduction of sparsity in the standard systems by using the normal techniques of Collaborative Filtering is not a panacea. In our proposed design though, there is the advantage of distributing the computing effort to that peer that initiated the query. This effort includes the following operations:

- The collection of the data about cooperation with the other users/peers,

- The computation of the derived Trust and Similarity,
- And finally, the production of the output to the user interface.

As will be seen in subsequent chapters the response time for queries in such a system is highly dependent on the Trust filters used for the propagation of queries and on the number of peers that are involved in the process.

6.4 The outline of our experiment

To support our hypothesis we ran experiments on a small community of 100 nodes and compared the Recommendations of our Trust-enabled system against those results that a plain Collaborative Filtering method would give. In our experiment we used the same community as used in the previous chapter for the evaluation of the neighbourhood formation protocol.

We also performed a comparison against the output that we would get if the choices were solely based on intuition. Even though an intuition based comparison would not have any practical result we present it here as a point of reference so as to emphasise the benefits of our design. Note that some intuition based value cannot be calculated in decentralized environments because they require the existence of global knowledge, which a single peer cannot have. We assume that an intuition-based choice follows the statistical distribution of the past choices of ratings over the items. We call this 'Intuition' in our terminology. In our particular case this distribution has a median value of 4.

The aim of performing this series of tests was to examine how efficient our design might be if applied to a real Recommender System. Efficiency is measured as how successfully the system can predict a consumer's preferences. Given that in a conventional Collaborative Filtering system the efficiency has a known value, our aim in using the Trust-enabled Collaborative Filtering system is to improve this situation and get as higher value as possible. As we mentioned above the data used were taken from a publicly available dataset of the MovieLens project.

We applied some filtering to the existing relationships to avoid poor performance due to the noisy behaviour of the Pearson Correlation Coefficient that we used for the prediction of values. Under this scheme, those relationships which were built upon 15

or less common experiences were not considered in our calculations. This value (15) was chosen as the most appropriate according to the evaluation experiment we performed in the previous chapter and was applied as a filter in the Trust queries used for the propagation of Trust. From the evaluation experiment it was shown that for this number of common experiences and beyond our Trust modelling converges to the modelling done using the Beta pdf.

The dataset we used also contained timestamps for every rating indicating the time when the rating took place, but this information was not considered at all in our correlations since at this stage of our research, our intention was to perform a study on the static behaviour of the model. The use of time stamps might be useful in some future experiment as a secondary criterion for choosing ratings that have been issued by both counterparts within a certain amount of time and accordingly be considered in a Trust relationship. In this variation, where more than one rating has been given by a user for some item, we would only take into consideration the newest for the calculations.

Next, we provide some data and the format of the experimental database to give an idea of what the data we used are like. More about the format of the database we used can be found in Appendix A.

The RATINGS were provided in the form of a stream of data contained in a single file. The rest of the information, such as USERS and MOVIES, was kept in separate files. The table called TRUST was derived by appropriate processing of the RATINGS table. More specifically this table contains triples of (b,d,u) values for all related users and were derived using our Trust modelling as described in Chapter 5.

In our analysis, we demonstrate how such a system would perform in comparison to a standard Collaborative Filtering scheme and also against a system that involves no use of recommendations at all. In the latter the users would make the choices by themselves by using their Intuition alone. For this comparison, every user's predictions were guided alone by their personal past experiences of every user separately by using the local knowledge acquired from the way that the user has rated its own experiences. Such a scheme would be somewhat impractical in cases where insufficient knowledge has been acquired by an individual user-node and thus the user would be unable to make predictions about its own future rating behaviour. The main reason that it is used here is for comparison purposes.

6.5 Definitions

For the sake of our experiments we introduced two notions that will be used in our measurements.

I. Computability: We define this as the total number of services for which a user can find Opinions using our system through the established Trust graph, divided by the total number of services that have been rated by all counterparts in the sample dataset.

Each user has its own Computability value because the number of Opinions that can be collected through the Trust graph varies from person to person and is dependent on the point of view as well as on the number of existing experiences. The performance limit reached when considering all counterparts should be seen as the normalization value as then no more services can be reached by any of the participants in the system.

II Recommendation Error. We define this as the average error that users make when trying to predict their own impressions of those services they can reach when using the Reputation system. It is defined as the predicted rating divided by the rating that is given after the experience. Similar to *Computability*, this measure is also specific to a particular user. That is because individual users can reach different numbers and groups of users from their points of view.

In order to provide a unique metric of effectiveness, we also introduce the notion of *Normalized Coverage Factor F*. This measure combines *Recommendation Error* and *Computability* into a single value and is expressed as:

$$F = (1 - \overline{E}) \cdot C$$

where:

$\overline{E}$ is the average *Recommendation Error* for a particular user and C is the *Computability* value for that user. We take the average value of E because each user has more than one experience for which the error can be calculated. In other words, F

represents how much a user benefits from its participation in the community. High values of F, which translates to low computability Error and good Coverage, would mean that the participation of the user in the group is beneficial for him/her.

6.6 Testing method

To perform the evaluation of our techniques we used two algorithms, one for the calculation of *Computability* and another one for the *Recommendation Error*. Due to the static nature of the available dataset we used in the experiment there was no way to simulate a real environment of users experiencing services in real time. The dataset that was available for the experiments was representing a single snapshot of the system at a single moment. Therefore, what we show in the results is what is observed at one particular instant.

We used the technique called *"leave-one-out"* as our metric for the reason that we should measure the difference between a prediction that is done using our Trust-enabled system, and the actual experience of each user.

```
Let S the set of all services
Let t the filter used in the trust propagation
Let hop the number of hops the trust can be propagated
Let U the set of all users in the group
For each user U_A in U {
  Let S_A ⊆ S the set of services that U_A has experienced
  For each service s in S_A {
    Let r = R(U_A, s)                    /* The rate of U_A on S */
    Let B_S ⊆ U the set of users that have also experienced s
     For each user U_B in B_S {
        Trust UA,B = f(path A>B, t, distance A>B<hop)
        CC A,B = f(Trust UA,UB)
     }
     Let Sp = f(CC_A,i), ∀ i ∈ B,
     Let Er = f(r,Sp)
   }
   RecomError = Average(Err)
}
```

Figure 2. Pseudocode for evaluating the Recommendation error

The pseudocode for the algorithm we used to evaluate the *Recommendation Error* is given in Figure 2. We have used some calls to some specific functions that worth explaining:

In the nested *For* loop, we call the Trust $_{UA,B}$ function with parameters:

- The filter t used in Trust propagation,
- the maximum hop count that the request can be propagated further away,
- the sub-graph that connects the source of the query A with the destination node B.

In the next part of the pseudocode where the calculation of similarity is performed, we use the Trust that was calculated in the previous step. Immediately after this loop is the calculation of the predicted rating *Sp* using the formulae we presented in the previous chapter. This is done by using all available information from the group of users that have used that service before and with whom Similarity can be calculated. Finally the Error (Err) for a single experience is calculated from the real rating and the predicted rating Sp. Setting *hop*=1 in the algorithm returns the prediction error for the plain Collaborative Filtering method, which is based on examining the nearest neighbours only. In the same way *hop*=0 can give the error if users were doing the choices guided by their intuition alone since there is no participation of the graph. In our experiments we ran tests for *hop* ranging from 0 to 3.

As regards *Computability* (or else Coverage), we also ran evaluations for hop distances with values from 0 to 3. The pseudocode of the algorithm we used in the experiment is shown in figure 3.

```
Let S the set of all services
Let t the filter used in the trust propagation
Let hop the number of hops the trust can be propagated
Let U the set of all users in the group
For each user U_A in U {
 Let S_A ⊆ S the set of services that U_A has experienced
 For each user U_B in U {
   Trust_UA,B = f(path A>B, t, distance_A>B<hop)
   If Trust_UA,B > 0 {       /*  B is reachable by A  */
     Let S_B ⊆ S the set of services that U_B has experienced
     S_A = S_A + S_B  /* Add it to A's potential experiences */
    }
  }
  Coverage = S_A / S
}
```

Figure 3. Pseudocode for calculating coverage

The function used in figure 3 for the calculation of the Trust between any two entities within the Trust graph is the same as that used in the pseudocode in figure 2.

By Coverage we mean the ratio between the experiences of an entity divided by all the experiences that this entity could have experienced. The If $Trust_{UA,B} > 0$ statement checks whether there can be a Trust relationship between the two users A and B.

The calculation of the relationship between the two entities was implemented by two operations, the first carries out the graph collection phase and the second the simplification of the resulting graph.

For the process dealing with discovering the Trust paths between any two entities in the Trust graph we used a parser to collect the paths in the graph that lead from the trustor entity to the trustee. The graph collection parsing was done in such a way that all the vectors between the trustor and the trustee were collected back to the entity from which the query originated (trustor). The requirement for opinion independence [25] obeyed that this should be done for each entity separately in order to avoid the existence of hidden topologies. Therefore, the calculation of the resulting Trust was left to each trustor individually after it received all the Trust vectors.

As regards the second phase in the calculation, this is a network simplification technique and is based on the application of two Subjective Logic rules (expressed by the *Consensus* and *Recommendation* operators) to the collected graph. This is to simplify the resulting graphs into a single Opinion that the trustor would hold about the distant trustee.

6.7 Problems in the graph analysis

In our model for the cases where Trust paths couldn't be analyzed and simplified further by just using these two operators, we applied a simple pruning technique to remove those Opinion vectors that were found to cause problems in the simplification process. In the case studies we used there were many models of different topologies which were found to be problematic and which could not be simplified further by just using the classic *Consensus* and *Recommendation* operations. In the literature these topologies are referred to as *non-canonical.*

Even though that there are guidelines [41] for how to construct DSPG or *Directed Series-Parallel Graphs* by sequences of serial and parallel composition operations, in our case scenario they could not be applied. The problem in our infrastructure is that

the graphs that are to be analyzed and checked for non-canonical topologies have already been formed when the graph analysis is to be done, since they are built upon the virtual relationships that exist between users. As we explained in the previous chapter these neighborhood formation schemes are derived from the first hand evidence (expressed ratings) that are available from each peer in the graph. However these graphs might include loops and dependencies which may lead to information loss if some graph pruning technique is applied unwisely. Therefore, any such operation must be applied carefully so as to minimize information loss. The same paper [41] refers to a fourth property which was introduced for each vector which is called *Confidence* and has to do with the significance of information that the vector carries. The criterion for defining *Confidence* is the maximization of *Certainty* (the complement of *Uncertainty*) in the produced sub-graph.

6.8 Phase 1 - The path discovery algorithm

In our model we used a simpler technique in which the vector that is to be pruned is picked by its position in the topology. The criterion we use is to apply the pruning operations with the aim of achieving the minimum loss with regard to the number of vectors that have to be removed from the original Trust graph, in order to bring the graph into a *Canonical* form.

In the testing infrastructures we used for our experiments we discovered various topologies which during the simplification lead to *non-Canonical* graphs and for reference we present them in our work indicating the vector which has to be pruned.

First we must describe the algorithm that does the construction of the Trust graph by finding all the available paths between the originator and the target entity in question. The algorithm we propose is executed recursively starting from the initial pair of trustor and trustee. The constructed graph is saved in a temporary set with the format (*trustor,trustee,TrustValue*) and is used by the algorithm that is applied afterwards for the simplification of graphs.

The *Saverecord* commands are called in case a recursive search attempt is successful and when the target has been reached. The first *Saverecord* that appears in the listing is initiated when the target is reached and then the vector that ends to the target is saved in the temporary set. The second *Saverecord* was put there to store

recursively the entire path from the query origin to the destination in the set. This is performed whenever control is returned back from a recursive call. The *Indicator* variable is used to show a successful execution of a *Saverecord* command which translates to a successful discovery of a path towards the destination node.

```
DEFINE Origin = The Entity where the search will start from
DEFINE Target = The target Entity where the search to be stopped
DEFINE Depth = The depth in the recursive search
DEFINE TeeMinExp = Filter used as the min number of experiences of trustee
DEFINE MinComExp = Minimum number of common experienced between the trustor and trustee
DEFINE TrustFilt = The minimum trust value in a trust relation to be  called the algorithm recursively

Find_path(Origin,Target,Depth,TeeMinExp,MinComExp,TrustFilt)

    If Depth=0 then Return(false)

    For all Trust Vectors with  origin=Origin and
              Commmonexperience>=MinComExp and
                                  bdu>TrustFilt       do
       Read Vector from the Dataset
       Indicator=false
       If Vector.experiences>=TeeMinExp Then
         If user2=Target Then
               Saverecord
               Indicator=true
         Else
           If Find_path(user2, Target,  Depth-1, TeeMinExp, MinComExp, TrustFilt)=true Then
              Saverecord
              Indicator=true
          Endif
        Endif
       Endif
    End_for

    Return (Indicator)
End
```

6.9 Phase 2 - The graph simplification algorithm

Once the resulting graph of vectors has been formed by the application of the previous algorithm the next concern is how to carry out the simplification of the graph.

Assuming that the output of the previous phase is a set of vectors, we can apply upon this set the following algorithm which in every loop of execution produces a new structure equivalent with the original one but having one vector less. The loop is executed until the resulting structure has been converted to a single vector.

```
DEFINE Origin = The Entity where the search will start from
DEFINE Destination = The target Entity where the search is to bended

DELETE any vectors that point to the Origin
DELETE any vectors that originate from the Destination Ds
WHILE there are more than a single vector in the graph DO
    DELETE self-loops A->A
    FIND a pair of consecutive vectors (a,b)
    APPLY the SUGGESTION operator to (a,b)
    FIND a pair of parallel vectors (c,d)
    APPLY the CONSENSUS operator to (c,d)
    DELETE unlinked vectors
    TAKE SNAPSHOT of the graph
    IF graph has not changed THEN
        DELETE loops A->B,B->A
        IF graph has not changed THEN
            APPLY the SIMLPE pruning algorithm by REMOVING the middle vector
            IF graph has not changed THEN
                APPLY the COMPLEX criterion (Origin , 1)
                IF graph has not changed THEN
                    APPLY the COMPLEX criterion (Origin, 2)
                    IF the graph has not changed THEN RETURN panic code
                ELSE Return
                ENDIF
            ELSE Return
            ENDIF
        ELSE Return
        ENDIF
    ELSE Return
    ENDIF
END WHILE
```

```
COMPLEX criterion (from F, hop H)
    FOR ALL vectors (k.m)
        IF K is in H hop distance form F
            AND K should be the ending of one vector and the beginning of at least two
            AND M should be the beginning of at least one vector and the ending of at least two.
        THEN REMOVE (k,m)
        ENDIF
    END FOR
```

In order to make the simplification algorithm easier to understand we need to clarify the following terms used in the relevant processes.

By *Self-loops* we mean vectors where the heads point back directly to their tails. By *Parallel vectors* we mean a pair of vectors that have the same starting point and the same ending point. *Unlinked vectors* refer to those vectors where their beginning points are not the ending points of any other vector nor are their ending points the beginning of another vector. The purpose of taking snapshots of the graph as it develops in the algorithm is to identify any successful simplification operations that may have been done in the database. There being no change monitored by the

snapshots is an indication of the existence of *non-Canonical* form in the resulting graph. The *simple* pruning algorithm that is called as a first option to spot a *non-Canonical* form is the one that is demonstrated in Figure 4.a. The *complex* pruning criterion that is applied as a second option is demonstrated in figure 4.b. As can be seen from the algorithm this criterion can run for distances of 1 and 2 hops from the origin node. In the graph we used for the evaluation of our algorithm we applied the complex (2^{nd} option) criterion for resolving complex cases at hop distances up to 2 from the origin. The main reason for not proceeding further to longer distances was the fact that in our experimental set up the Trust sub-graph was constructed using paths which were only 3 vectors long and resolving cases at that depth was quite sufficient.

The *panic code* we included in the pseudocode is returned in any case a non-canonical form has been spotted and it could not be resolved by applying either the *Simple* or the *Complex* criterion.

The simple pruning algorithm refers to cases like the one shown in Figure 4.a where O represents the origin and D the destination entities. The vector that has to be removed according to our pruning policy is shown as (A, B). Figure 4.b presents a more complicated scenario we identified which requires application of the complex criterion for depth 2, in order to be resolved.

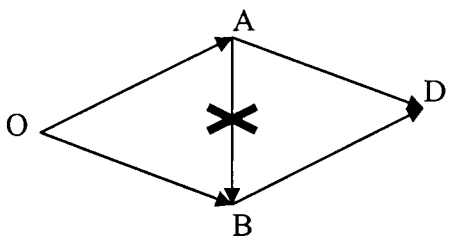

Figure 4.a

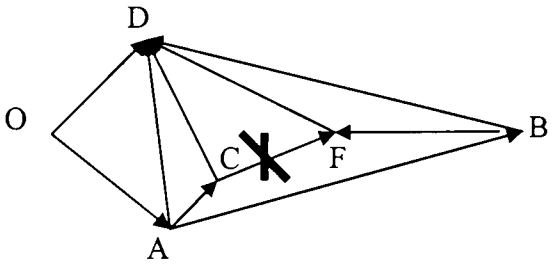

Figure 4.b

As can be seen, such a complex scenario can be resolved with the minimum loss if the indicated vector $C \rightarrow F$ is removed from the graph. In order to identify similar cases of vectors and successfully be removed from the structures we have described a set of criteria that a candidate vector must meet:

- The vector it is within two hops of the Origin (O),

- Its head (C) is the ending of at least one vector ($A \rightarrow C$) and the beginning of two others ($C \rightarrow F$ and $C \rightarrow Dest$),
- Its tail F is the beginning of at least one vector ($F \rightarrow Dest$) and the ending of at least two ($C \rightarrow F$ and $B \rightarrow F$).

In order to simplify the sub-graph by making the minimum possible alterations to it, the criteria are applied in such a way that on every single cycle only one vector will be removed from the processed graph and the algorithm through the *while-loop* will check if the complex case has been resolved and the derived Trust between the Origin and the Destination has become computable.

Next follows the parsing and simplification policy expressed in a formal way:

- Let V be a set of vertices of a directed graph and let E be a set of edges (arcs) $(e_1,e_2,e_3,\ldots,e_n)$ of that graph.
- Each Vertex represents an agent and each Edge the Trust relationship between the first and the second agent.
- We call *i,f* a pair of functions on E such that if $i(e)=x$ and $f(e)=y$ then e is an edge denoting the Trust of agent x for agent y.
- Assuming the existence of two vertices x and y such that $i(e)=x$ we call *degree(i(e))* the function that returns the number of edges that begin from x.
- We call the *outer degree (odegree)* of a vertex x the function that returns the number of vertices that originate from x and the *inner degree (idegree)* of the vertex y the function the returns the number of edges that end at y.
- Let *origin* and *dest* be the vertices where the graph parsing originates from and terminates, respectively.

Definitions:

Loop: A single edge path with the same initial and final points.

Digraph: A graph whose both directed edges exist. (e.g. $a \rightarrow b$ and $b \rightarrow a$).

Unlinked: A vertex having inner or outer degree, lower than 2.

Not Analyzed: A sub-graph that cannot be simplified using the standard algebra and is resolved by pruning one of its edges.

In order to get the graph simplified, the following six steps have to be applied in a loop up to the point where the whole graph is simplified into a single edge.

- Parallel composition: $\forall e_1, e_2 \in E : i(e_1) = i(e_2) \wedge f(e_1) = f(e_2), \Rightarrow e_p^{1,2} = e_p^1 \oplus e_p^2$
- Serial composition: $\forall e_1, e_2 \in E : i(e_2) = f(e_1), \Rightarrow e_p^{1,2} = e_2^1 \otimes e_p^2$
- Loops Removal: $\forall e_1 \in E : f(e_1) = i(e_1) \Rightarrow$ remove e_1
- Remove Unlinked Vertices: $\forall e_1 \in E : i(e_1) \neq origin \vee f(e_1) \neq dest \Rightarrow$ remove e_1
- Avoid Digraphs: $\forall e_1, e_2 \in E : i(e_1) = f(e_2) \wedge i(e_1) = f(e_2) \Rightarrow$ remove $e_1 \vee e_2$
- Simple Pruning Criterion: $\forall e_1, e_2, e_3, e_4, e_5 \in E :$ $i(e_1) = i(e_3) \wedge f(e_{21}) = f(e_4) \wedge i(e_2) = f(e_1) \wedge i(e_4) = f(e_3), i(e_5) = i(e_2) \wedge f(e_5) = f(e_3) \wedge degree(i(e_5)) = degree(f(e_5)) = 3 \Rightarrow$ remove e_5
- Complex Pruning Criterion: $\forall e_1, e_2, e_3, \in E :$ $i(e_1) = origin \wedge f(e_1) = i(e_2) \wedge i(e_3) = f(e_2) \wedge idegree(i(e_3)) \geq 1 \wedge odegree(i(e_3)) \geq 2 \wedge odegree(f(e_3)) \geq 1 \wedge idegree(f(e_3)) \geq 2. \Rightarrow$ remove e_3

6.10 Our Test-bed

Even though the system we designed is intended to be used as a distributed one, in our study we chose a centralized environment as our test-bed for evaluating the algorithms and for carrying out the processing of data. That was done for two reasons.

The principle reason was that the infrastructure was available which encouraged us to do it in the centralized way. Moreover, the tools that were available for the tests were tailored to run in a centralized rather than in a distributed manner. An equally important characteristic of this type of testing was that the type of evaluation applied would not affect the results produced. The alternative would be to build a real application for which the system was designed and run it in the same way as it would operate under the commands of real users. Since it was much more difficult to build and run the real application, we preferred to simulate the operations carried out in a centralized controlled system.

The other reason for choosing a centralized type of infrastructure for the evaluation was because in the case of applying our Trust enabled solution to an existing Recommender system, which would run as a centralized web site [16][17][18], we would have the opportunity to access the data in a way similar to a conventional system. Our contribution would be restricted in that case only to the provision of algorithms which would be used to calculate derived Trust and to interpretation of similarity. As stated previously about the datasets of contemporary Recommender systems, the high sparsity that they have is what makes such systems workable because the number of necessary computations is decreased to an affordable number. Deciding to run the proposed Recommendation service in a centralized manner makes it possible to investigate the application of some filtering of the relationships that will finally be used in the Trust derivation. More investigation is required into the applicability of the algorithm in extreme situations such as decreased sparsity.

Even in the case where the real system is to run in a centralized manner we could consider how this can be accomplished by using Web Services. In such a solution each graph analysis task could be run as a deployed service in a Grid architecture. As can be seen from the algorithms we presented, there is a huge processing requirement for the graph analysis where a single or a small number of CPUs would be unable to produce accurate results within a reasonable time without significant impact on the system's usability. Alternatively, doing the graph analysis in a decentralized manner the whole operation will be done simply via the exchange of messages between the nodes in the graph structure, which would not be very demanding in CPU power but instead would require network resources. This translates to the need for high speed connections between the nodes in the graph. In the performance analysis we present in Chapter 7 we simulate and analyse the performance of the distributed architecture which is the main focus of our work, rather than the centralized one. However, we do discuss the requirements, benefits and pitfalls of deploying it in a decentralized system. Next we describe in more detail the software itself as well as the infrastructure used to perform this task.

Our experiment was performed in such a way that a centralized database was the only entity used as a central repository of data. The process of Trust graph analysis was done as a service executed by a cluster of computers directed by a scheduler. We

can imagine each of the computers in the cluster to be running an infinite loop asking for graph analysis jobs to carry out. We were feeding the scheduler with combinations of users for which the *Recommendation error* and the *Coverage* were to be calculated as well as with other parameters such as the depth of search and the Trust propagation filter.

The central database was used as the place where the user ratings were stored together with the details of the users and the products. A separate table called TRUST in the database was used for storing data such as Trust vectors derived from the processing phase of first-hand evidence (ratings) which ran asynchronously as a separate task.

More specifically, there were two kinds of processes run by the scheduler.

- Processes that were calculating the primary Trust between any combination of nodes based on their common ratings (first hand evidence)
- Processes that were used for the calculation of the derived (secondary) Trust by analysing the graph.

The latter category of processes was doing the discovery of graph vectors that existed between the Origin and the Destination nodes as well as the calculation of secondary Trust. For that task the two algorithms we presented previously in pseudocode in figures 2 and 3 were implemented.

In the first category the tasks were designed to be executed asynchronously whenever a new rating was inserted into the system. In this case there is a possibility that a primary Trust relationship may change its value if a rating pattern for some entity is changed. More about the formulae used in the conversion of first hand evidence to primary Trust values can be found in Chapter 5.

The use of a centralized infrastructure was also the main reason why we used a subset of only 100 users from the MovieLens database even though the whole data set contains data from at least 3000 users. Even with the choice of using a decentralized network for the simulation of the experimental environment we would still need a large number of CPUs, for simulating each individual node. That was difficult to accomplish in the context of our research.

Nevertheless, to avoid bias in the results taken, the 100 user base that was actually used included users randomly chosen out of the nearly 3200 that were available.

6.11 Results

There were two parameters used in the measurements we performed on receiving the results: the Trust filter used for the further propagation of Trust from one hop to its neighbours (called the *"filtering policy"*), and the maximum distance that a message was allowed to be propagated during the calculation of derived Trust.

Figures 5, 6 and 7 show the results of the experiments. Figure 5 shows how Prediction Error changes with respect to hop distance and various belief filtering policies used in Trust propagation. In total, we performed tests for three filtering policies (b>0.5, b>0.6, b>0.7) where *b* refers to the belief property in a relationship which expresses primary Trust. According to this filtering policy, a Trust query is not allowed to propagate to entities that are not as trustworthy as the filter value. Under this scheme path exploration will proceed up to the point that the belief property of some neighbouring entity does not exceed the value set on the filter and also the hop distance does not exceed the preset value.

We assumed that the use of a filter which would block the propagation of queries to nodes in the discovery path that have b<0.5 should not be considered in the experiment because in that case the other two properties – uncertainty and disbelief – dominate. What we were willing to investigate were the cases where the entities involved were clearly seen to be trustworthy.

In all the diagrams (figures 5,6,7) we have also included the results when applying the plain Collaborative Filtering method. (propagation allowed up to 1 hop). We also show the results when the choices are made using intuition combined for all users together. This appears in the diagram as a dotted horizontal line. As we mentioned above, that case has been included only for comparative reasons because it corresponds to an imaginary situation where users predict their own choices by making use of the old ratings that they have given. Such a prediction scheme is based solely on the given ratings which follow the known distribution function referred to in Chapter 5. In that prediction scheme there is no categorization of Trust filters since there is no use of Trust at all. The results represent average values taken over the series of 100 entities.

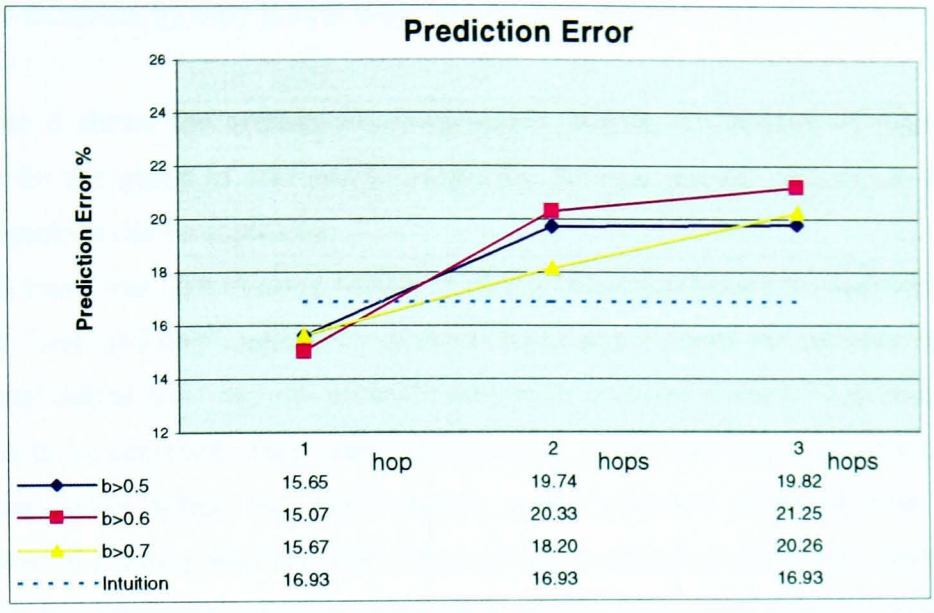

Figure 5. Accuracy of Recommendations

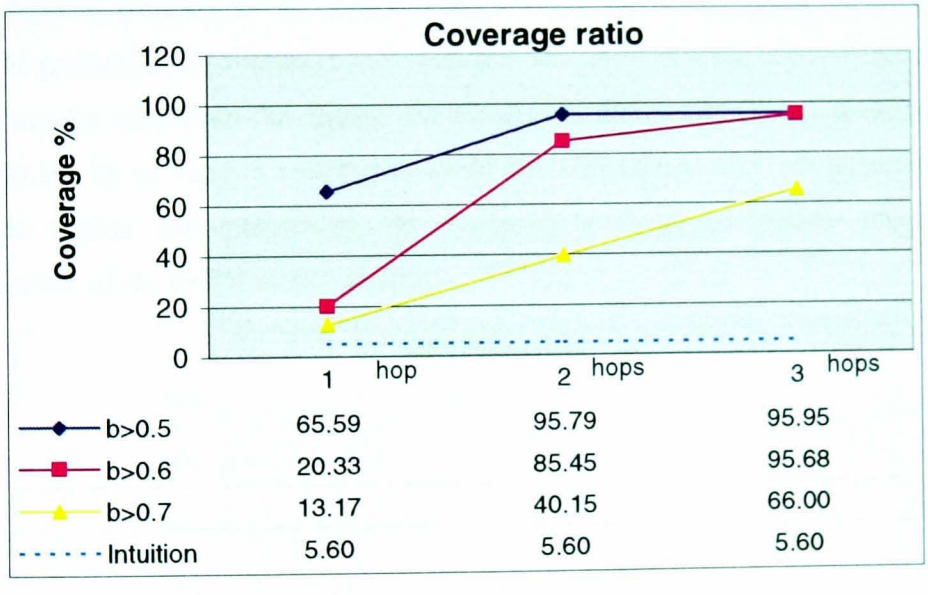

Figure 6. Computability of Recommendations

The results show that on average, a rating policy based solely on intuition appears to give lower error than our method, but as we will see, this criterion seems to be inadequate when used for making decisions. An equally interesting fact is that in our method (hop distance >1) the error is not affected significantly as the hop distance increases from 2 hops to 3, which means that there is no loss of precision in using the Trust graph. Especially in the case when the weak filtering policy (b>0.5) is applied,

the error increases by only 0.12% when the distance increases from the 2 hops to 3 hops.

Figure 6 shows the average *Coverage Ratio*, that is, the number of reachable services for the group of 100 nodes divided by the total number of services about which Opinions can be expressed.

In all cases, our idea of using Subjective Logic rules to calculate the derived Trust within a *"web-of-Trust"*, appears to perform better against both the intuitive choice (horizontal dotted line) and the plain Collaborative Filtering method (hop count 1). This was to be expected, since when traversing the graph there are more chances to find some entity which has some entirely new experience about an item. The application of a strong filtering policy has a negative impact on coverage, especially for short hop distances, whereas applying a middle-range filter (b>0.6) seems to improve the situation in the case of (hop=2). Another important characteristic is that strong filtering policies do not have a positive effect on Coverage Ratio (as expected) since the probability of finding a new node that has some new experiences declines.

As can be seen from the figures for *Prediction Error* and *Coverage Ratio* alone these do not let us reach a conclusion about the effectiveness of our algorithm. This was the reason for introducing the *Normalized Coverage Factor* which is a combination of the previous two factors.

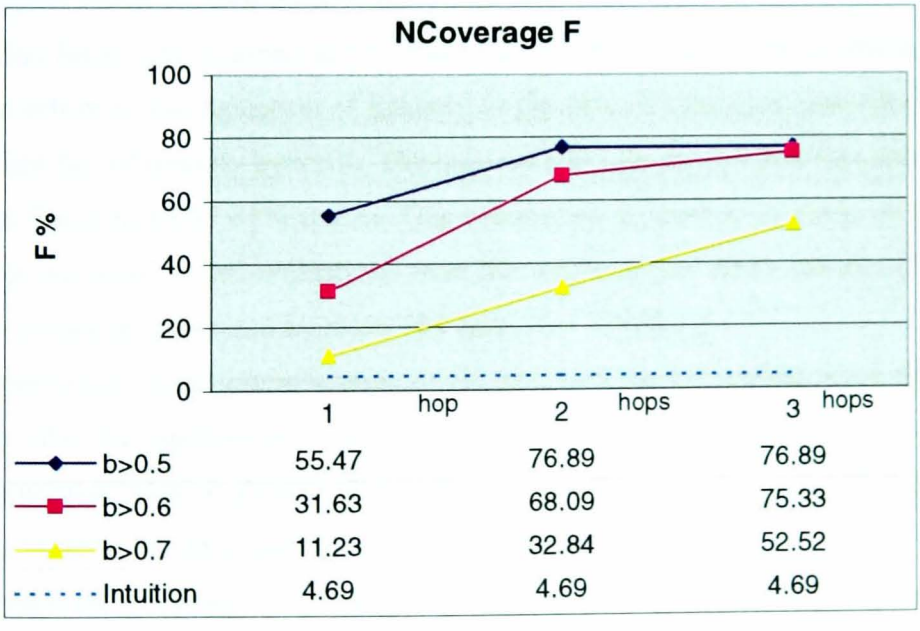

Figure 7. Normalized Coverage F

Figure 7 presents the *Normalized Coverage Factor* we introduced and which can be considered as the total gain from using some policy. From the graph it can be seen that the *NCoverage* factor has a similar trend as *Coverage* and the participants do not really get any benefit when strong filtering policies are applied. In almost all cases the curve of the filtering policy b>0.7 gives a lower *Normalized Coverage* than any other combination. The only exception is the case when a search of 3 hops is performed and a strong filtering policy is used. In this case it appears that a strong filtering policy (b>0.7) performs slightly better than the standard Collaborative Filtering method (max hop distance=1) for medium-filtering policy (b>0.6). Strong filtering, however, consumes less resources due to the simpler graphs that have to be explored and simplified. Therefore, such comparison without including the cost compared to the benefit is unfair. For such a purpose we need to define what the cost of a search operation is. In the next chapter there is a full performance analysis that identifies the best searching policy.

Finally, as seen from figure 7, as with Coverage Ratio, the intuition based approach performs worst of all the policies.

6.12 Discussion of the results

Our method seems to have positive effects on both Coverage (Computability) and the Prediction Error. The increase in Coverage that can be achieved also seems to have a positive effect on the reduction of Sparsity in the dataset. Our measurements show a significant fall of sparsity by 9.5%. The original 100 user dataset that was used in the test was found to be 97.85% sparse. This calculation of sparcity is based on the 100 users set we used in the evaluations over the whole set of items (6040). The total number of ratings expressed by those 100 users was 12976.

Nevertheless, such improvement might not look as interesting since the final sparsity after the application of the method is still high at 88.35%. Judging it from another point of view, it shows some improvement in the coverage of empty cells that reaches 18.45%, which in our opinion is a significant figure.

A more careful look at the results reveals that when using our method there is a 30% possibility that a user will get some benefit from using the Trust graph. The remaining 70% were those users who benefited by just using the plain Collaborative Filtering technique (as shown in 1 hop distance figures). This is because in the dataset

used it was likely for two users to have common experiences and this was dependent on how clustered the user communities in the examined dataset were. In our case, there was high clustering in the examined virtual topology since the users seemed to be connected with many others due to the high number of common experiences we set as a limit for a Trust relation to exist (15 experiences).

In the case of our experiment the 100 user dataset produced a single cluster of nodes. We filtered the users that constituted the virtual community and we selected from the MovieLens dataset only those who had expressed at least 30 experiences during their lifetime. In our case, if there were more than one clustered communities of users, then on average the benefit of using the graph would be higher because a small number of users would be enough to bridge the gap between those separate clusters and thus to increase the number of Recommendations that could be communicated to the other cluster. In this scenario we suspect that the percentage of users that benefited by making use of the graph would be higher.

The extra 30% benefit we mentioned characterizes the potential of the proposed system with respect to the dataset used in the experiment. In our opinion that potential cannot be characterised as high. Possibly modifying the rules by which the primary Trust was built (e.g to use stricter rules in the formation of primary Trust relationships) might give better results as regards the ratio of users benefited. As regards sparsity, a comparison between our method and others used for the purpose of reducing the sparsity would be interesting.

The results justify the explanation we gave previously saying that the plain method suffers from reduced Coverage due to the small number of ratings that close neighbours can provide and thus the possibility that a similarity relationship can exist between two entities is decreased. This is where Trust relationships can express some kind of correlation between users even if there are no common experiences between them. This happens because in the plain method, which is based on nearest-neighbour algorithms, the relationships rely upon exact matches.

As regards Prediction Error, a comparison against choosing ratings randomly instead of predicting them shows that our method performs better even for hop distances of 3. As is shown in figure 5, which represents the error, all figures except that of hop distance of 1 (plain method) appear to produce such high error in the predictions that exceeds that of the random choice. Using our dataset to generate random values based on the frequencies curve of ratings would give error rates as low

as 24.5%, but such a comparison would be unfair for two reasons. First, because such a rating requires users to have access to global knowledge which is unlikely to be possible. In a distributed model it is very unlikely that such information would be available globally and be disseminated everywhere due to the nature of distributed systems. The second reason is that the error is highly dependent on the distribution of ratings over the classes of rates and that makes a rating that is closer to the Median be more likely to be closer to the real value.

Although our method increases the system output by improving the quality of recommendations compared with the plain method, the algorithms do not scale to large amounts of data and thus performance degrades as the number of users increases. This is due to the large number of computations required for the calculation of *indirect* Trust through the Trust graphs, because of the execution of complex and CPU intensive algorithms. In other words, the design will not lead to a system capable of providing quality Recommendations in real-time. As we indicated above, the provision of a Recommendation should be seen as a synchronous operation which requires the results to be returned in a relatively small amount of time after the query has been initiated. In the next chapter, which includes a performance analysis of our model, we explore this issue by introducing a parameter called *"Patience Limit"*. Even in the case where the complex and expensive computations of *Direct* Trust vectors are done in the background there might still be a bottleneck due to the calculation of the *Indirect* Trust relationships and the discovery of Trust paths. This is because the method requires that a vector that represents *Direct* Trust must be recalculated whenever a new rating is introduced by any of the two participants in a Trust relationship. The main reason for this is the asymmetric property of Trust which requires recalculation whenever any side enriches its set of experiences. However, these recalculations could be done off-line, preserving the computing power for the graph analysis. This is quite feasible given that in such Recommender systems the user base and the item data do not change frequently. The same situation applies to the ratings of people about items.

For the above reasons and especially the high demand in CPU power, the method we proposed does not seem to be suitable for use in centralized systems. Cacheing techniques might improve the situation slightly, provided that changes in the virtual Trust infrastructure will not happen often.

Therefore, it is suggested that such a system should run in decentralized manner disseminating the cost of derived Trust calculation to the nodes that are involved in a query. More precisely, the graph discovery part of the algorithm should be done by a message-flooding scheme returning the Trust vectors involved back to the query originator. The difficult part though, relies on the graph analysis phase which has to be carried out only by the querying entity itself so as to avoid dependency problems. We consider the case where this operation is done jointly and in distributed manner by more than one different entities as a future research issue.

6.13 Future Issues

In the future we intend to perform a comparison of our method against other alternatives that are used for reducing the sparsity of large datasets such as Horting [33] or others based on Dimensionality Reduction such as Singular Value Decomposition [30]. As regards depth we chose when carrying out the graph analysis, we anticipate performing more analyses using depths greater than the 3 hops we used in the experiment. This case will need slight modifications to the algorithms that deal with the graph analysis and the search for *non-Canonical* infrastructures in the graphs. Searching beyond 3 hops will help us study how the performance increases with the depth of search and also find the optimum depth (if there is one) given the high computational load that searching demands.

Since the method is not particularly suitable for running in a centralized infrastructure, because of the high demand for computation resources, the outcome suggests a need to explain how a conversion to a Peer-to-Peer architecture accompanied with redefinition of the roles would improve the performance of the Recommender system. The benefit in this case is two fold. First, it provides distributed computational load as well as higher robustness and second it makes the system less vulnerable to the attacks we discussed earlier. A Peer-to-Peer architecture is also closer to the natural way that recommendations within groups of people take place due to the commonalities between human communities and Peer-to-Peer communities which appear to have the same structural characteristics and governing laws (equal communicating with equal).

6.14 Conclusions

In this chapter we proposed a method based on the idea of exploiting the potential Trust relationships between the members of a community so that they can extend their knowledge and as a result improve the quality of recommendations they receive. Our innovation is that, in addition to the similarity relationships used in classic Recommender systems to correlate users, in the proposed system Trust relationships are used as well. These additional relationships were found useful in cases where similarity relationships could not be expressed.

To measure quality we introduced 2 notions:

- *Recommendation Error,* which is the difference between a value that has been predicted by using our system, and the real one.
- *Coverage* which is the ratio of the items that can be reached by some entity using the derived Trust graph divided by the whole set of items.

By combining both *Recommendation Error* and *Coverage* together into one measure we introduced what we call *Normalized Coverage*, which shows the total benefit that a single entity receives by using the algorithm.

In the study we applied a model that uses quantitative and qualitative parameters to build primary Trust relationships between entities based on existing common choices.

We used two operators of Subjective Logic algebra for the calculation of Secondary Trust and to relate users through their transitive Trust relationships that could exist between them using a maximum search depth of 3 hops. In this way we extended the users' neighbour base.

We presented the algorithms used for the derivation of the Trust sub-graphs that link the Origin and the Target entities as well as for the simplification of those graphs.

Our series of experiments was run with the following variables: the number of hops that the Trust queries were allowed to propagate from one to another in the derived Trust graph, and the minimum value of a property known as *belief* used in Trust propagation. That value was used to express the strength of Trust of neighbouring entities and we used it in order to restrict propagation to the more trustworthy neighbours. For the evaluations we used real data taken from a publicly

available centralized Recommender system and we presented some preliminary results about the performance of our method.

We also discussed the benefits and the pitfalls of our method. Our first results showed that even if the method is not capable of providing recommendations in real time, it helped the system to improve its efficiency which translates into increased computability. The encouraging fact was that this improvement was achieved without significant impact on the accuracy of predictions. More precisely it was shown that the method improves the effectiveness of the proposed Trust-enabled Recommender system shown as decreased sparsity in the available dataset.

Given that the dataset was taken from another experimental system, we focused on the problems that appeared when applying the principles of transitive Trust onto existing environments. These were the existence of dependencies in the Trust graphs and the derivation of *non-Canonical* graphs by using the existing evidence.

We also pointed out the disadvantages (mainly concerned with performance) and how they could be overcome if the method is applied in a decentralized environment.

Chapter 7

PERFORMANCE ANALYSIS

7.1 Introduction - The need for Performance Analysis

From the experiments we performed in the previous chapter it became clear what challenges are met and what benefits are obtained by using a Trust-enabled Recommender system. In this chapter we approach the problem of deploying such a system in a real distributed infrastructure. Since centralized solutions seem to be inappropriate to meet the requirements that a "*web-of-trust*" infrastructure imposes such as no central control, equal communication between the participants and Trust dependent on point of view, we attempt an analysis of the distributed approach.

We chose to assess the performance of our approach by simulation. This could also have been done using analytic modelling but the complexity of the algorithm and the availability of a dataset based on real data steered us towards the simulation approach.

The evaluation experiment described in this chapter examines the performance implications that our proposed scheme for predicting the users ratings for items using Trust might have on usability. In the simulated Trust infrastructure it is assumed that Primary Trust has already been built when the simulation begins. Thus, the Trust network topology is assumed to be static throughout the experiment. In a real system, users are considered as individual nodes in the system and may join or disappear dynamically. As regards to the computational cost, in reality, user nodes may have different capabilities for processing requests but here are assumed as equal to each other. More specifically, the time needed for an elementary computation for processing a Trust response is taken from the experiment in Chapter 6. The behaviour of user-clients is also assumed to follow a certain pattern.

The fact that the simulation procedure requires an existing Trust infrastructure led us to prepare such an infrastructure offline and run the simulations using it. In other words, we planned to run a simulation based on a static rather a dynamic view of the system because we were more interested in studying its behaviour with regard to how

it would respond to Trust queries. However, we are aware of the fact that asynchronous maintenance of the Trust graph when recalculating primary Trust vectors introduces an additional cost in resources and we assumed that such a task would be done in the background and thus would not interfere with the task of calculating secondary Trust. We therefore assume that there is always spare capacity of resources such as network bandwidth and CPU cycles, which would otherwise be consumed exclusively for computing the primary Trust.

The main purpose of this evaluation is to discover the limits of the parameters under which a Peer-to-Peer system can support our Trust Recommender system.

To be in agreement with the tests we performed in the previous chapters (evaluation of the Trust derivation algorithm and the testing of the various Trust policies) we used the same 100 user community we chose from the MovieLens dataset. Even though the chosen sample community might appear small since it has been taken from a publicly available database which contained more than 3000 users, we chose to do so because the scope of our study was to see whether the system shows scalable behaviour. For other cases, we would need to know what limitations should be set in the parameters that describe the structural characteristics of the infrastructure: number of nodes, connection speeds etc.

7.2 The Simulation method

A simulation model describes the functions of a system expressed by individual events for its component elements. Such a model requires that the relationships between the elements in the system be described and so becomes able to capture the effects of the elements actions on each other as a dynamic process.

A simulation is driven either by generated input data or by feeding it with some representative input data. The first case is a probabilistic or Monte-Carlo Simulation, and in that case the pattern which the input data follows needs to be known. In the second case we have a deterministic or trace-driven simulation which requires that some typical input data should exist in order to simulate the input operations. This is especially useful in the case when it is unknown whether the input follows a known pattern which can be described with a function.

To specify the system we need to know three things:

- The arrival mechanism or the pattern of job arrivals at the server,
- The service mechanism or the description of the service time for a job,
- The queue discipline or the procedure by which jobs are selected from the service queue.

Also needed is the time that is taken for a typical service to be carried out.

Probability distributions provide the most convenient characterization of the varying intervals. Once these distributions are known we can write a program that generates sequences of service variants and drive the simulator with these random sequences. This is called a self-driven simulation. Instead, Trace-driven simulation is more suitable when the system workload is based on real data. More about Performance Evaluation Methodologies can be found at H.Kobayashi [9].

In our case we assume that the arrival pattern of new requests at the users' systems follows the Poisson distribution function, so we have used a self-driven simulation. In this probability distribution function the number of process calls in the period T follows the probability distribution:

$$P[n(T) = i] = \frac{(\lambda T)^i}{i!} e^{-\lambda T}$$

Where λ is the parameter of the Poisson distribution. That means, in every T period the physical user submits on average λ requests. We assume that queues use a FIFO discipline.

7.3 Time control in the simulator

We can summarize the operation of the simulation system thus: at every node, queries arrive initiated by node owners and enter a queue waiting for service. After the processes have received service, which in our case is the execution of a query, they leave the system. A query is a request to calculate the derived Trust between any two peers (trustee and trustor) in the system. Thus, the simulation model must describe and synchronize the arrival and the servicing of jobs.

The method we used for controlling time in our simulating environment is called Synchronous Timing. In this method there is a global clock in the simulated

environment and the environment status is advanced by a tick at an appropriately chosen time unit Δt. For every time unit Δt the system state is updated determining what events took place within the predefined time unit. We choose Δt to be relatively small in order to reduce the possibility that two events occur simultaneously. The value we used was $\Delta t = 1$msec.

Possible events that need to be observed during a time unit Δt are the initiation of a new query by the owner of a node, the propagation of a message to a neighbour, the initiation of a query reply etc.

In the simplest case scenario where the arrival pattern follows a Poisson distribution function the probability that an event will take place within a time unit of $\lambda\Delta t$ is:

$$F_x(t) = 1 - e^{-\lambda T}$$

regardless of when the last event took place. By using this equation we can create an event generator function and simulate the environment that is to be tested.

7.4 Related work

In the area of Peer-to-Peer system simulation we should mention the work done by K.Kant and R.Iyer [71] which is a performance evaluation for resource sharing networks, carried out using analytic modeling. B.Yang and H.Garcia-Molina [72] provide evaluations of a wide range of configurations of Peer-to-Peer networks based on existing file sharing protocols. Even though both the above papers are good references in the area of sizing Peer-to-Peer networks they are restricted to resource location operations rather than the establishment and discovery of Trust relationships. So, we found it inappropriate to base our evaluation upon the work that they produced and therefore implemented an evaluation environment from scratch that was best suited to our needs.

7.5 Assumptions

In order to perform testing we made a number of assumptions regarding the conditions under which rating prediction is performed. Some assumptions were found necessary due to the limitations that a simulated environment imposes:

- We assumed that the pattern of user demand follows a Poisson probability distribution function with parameter $\lambda=1(min^{-1})$. That means, every minute the physical user submits an average of 1 Trust request to the system. In all our experiments we kept this parameter fixed.
- Every vector of the Trust graph has the form of a triple of *(Primary Trust Value(b,d,u), Trustor, Trustee)* and has a fixed size of 50 bytes which breaks down to 15 bytes allocated to the Trust value that is being carried in the message, 10 bytes split into the node addresses of *trustor* and *trustee* respectively and the last 25 bytes occupied by TCP headers.
- The number of Trust estimation queries that follow a resource-searching query is equal in number to the entities that have been found to have some experience with the resource.
- The simplification process of the Trust graph is done by applying the *Consensus* and *Recommendation* operators of Subjective Logic [8], to simplify parallel and serial combinations of opinion vectors respectively. The time required for a Trust graph to be completely simplified is dependent on the number of vectors that make up the graph. From our measurements an elementary simplification operation implemented in java takes about 1 msec in a modern PC, thus we assume for the sake of our experiment that in all it takes: $t_s = \frac{1}{1000}\text{sec} \times Size\_of\_Graph$. For simplicity we assume that CPU power is always available for the Trust calculation to be performed.
- The time required for the collection of the graph is dependent on the available bandwidth and it is calculated as: $t_c = \frac{Size\_of\_\text{Re}plying\_Graph \times V}{Available\_Banwidth}$ where V is the size of the Trust vector in bytes. We assume that V equals 50bytes as described above.
- The calculation of Trust begins after the query originator has received all the reply vectors.
- Peers reply to Trust queries with absolute honesty and return triples of (b,d,u) values that correspond to real values.

7.6 Variables

We identified 3 variables used in the simulation, which generate 60 different testing scenarios. These are:

- The Trust filter. This represents the minimum level of belief that the trustor must have about some neighbouring trustee in order to allow a query to be propagated further away. In the experiment we used 3 different filters for the propagation b>0.5, b>0.6 and b>0.7
- The bandwidth capacity of the network connection that links together the nodes that constitute the testing environment. In total we performed tests for speeds of 7 kbytes/sec and 64 kbytes/sec, simulating a low speed (analog modem line) and a high speed (DSL) connection respectively.
- The size of the community ranging from 5 to 100 user nodes. It might not seem meaningful to perform tests for communities as small as 100 nodes when there is data available for communities of more than 3000 nodes, but our aim was simply to identify any trends in regard to scalability.

In the simulation we performed it has been assumed that all nodes are connected with the underlying network using lines of the same capacity (e.g. modems) and also network congestion can only exist in the peers themselves and not in the underlying network.

7.7 Test plan

We ran a series of simulations for each combination of variables of *propagation filter*, *community size* and *bandwidth capacity*. In total, every combination was left to run for 10 sessions of about 2 hours of simulated time each and the results averaged. The length of simulation was chosen to ensure that the system came to a steady state before measurements were taken. The measures we used for the evaluation of the model were the *Response time* and the *Success Rate*.

Response time expresses the expected time taken between the initiation of the Trust query and the calculation of the Secondary Trust vector. In order to calculate

this for each query, we measured the time taken and estimated the distribution of the frequencies.

We introduced the notion of *Success Rate* because pure Response times, as will be shown, do not have a meaningful value. *Success Rate* expresses the probability that a query completes within a preset threshold of 10 sec. We chose this as a reasonable threshold value and we aimed in this experiment to measure how many times this value was exceeded thus rendering the system less usable. This threshold represents a measure called the *Patience Limit* and studies in interactive computer environments [73] have shown that for most users the *Patience Limit* has the above size. The same study mentions that a response that is given within 0.1 sec is considered by users as instantaneous. Furthermore, 1.0 sec is about the limit for the users' flow of thought to stay uninterrupted even though they will notice the delay. With that delay users have the feeling of operating directly on the data. The 10 sec limit is important for users for keeping their attention focused on the dialogue. Which is why we chose patience limit as the critical value for the decisions taken regarding the usability of the system.

This measure means that if no response has been received within this threshold the user may abandon the request or retry it. Abandonment and retry in distributed environments including Peer-to-Peer systems is quite expensive in resources because it adds more load to the system and makes the situation worse [74]. Therefore we treat both abandonment and retry as unsuccessful outcomes of queries.

```
while ( time has not elapsed ) do
   for each node
      if probability that a new query is issued > 1/min
         Initiate a new request with another randomly selected user
         S = trust query Initiated
         D = propagation delay of S
         R = time to collect the graph response of S
         C = processing time of S
         Response Time of S = D + R + C
         Success Rate of S = f(Response time of S,Coverage)
      end if
   end for
   display average Delay & Success Rate
 end do
```

Figure 1. The pseudocode of the algorithm we used for the testing

Figure 1 roughly explains the algorithm used in our testing plan. Appendix B contains the source code of the program that was used for the simulation experiment coded in Java. The reason why we moved to this solution and did not use an existing network simulation software was basically because the existing tools, e.g *ns* (aka network simulator) [80] are known to have a steep learning curve and they require a lot of experimentation to tailor them to the characteristics of each particular experiment. Besides, there was no need to describe in much detail the information communicated at the level of network packet exchange as is usually required by these tools. Another reason was the nature of the application itself: as with all other Peer-to-Peer software, our Trust-enabled system was running as a service above an existing network infrastructure and therefore, there was no need for high level operations to be described in much detail at the Transport and Network levels as a conventional network simulator would require. Hence, a relatively simple program written in a conventional programming language (e.g. java), which would simulate the basic operations of the application level protocol and of course would meet the requirements regarding *Time control* and the *Arrival mechanism* of the jobs we mentioned earlier, would suffice.

7.8 Results

Figure 2 shows the expected Response time of a typical Trust query for each of the 60 different configurations we included in the test and the various sizes of communities, and considers the case of using queries with a maximum distance of 3 hops. We ran no tests for the case of queries that were propagated for hop distances lower than 3 since these were found to be meaningless. The value of 3 hops was chosen as the most representative since it causes measurable traffic in the network. All the tests we performed for propagation distances of 2 hops produced Response times smaller than the 10 sec threshold we set as a limit. Therefore we concluded that there is no scalability problem for communities of up to 100 nodes in size using that propagation distance. At the same time a propagation distance of 3 hops is a good reference for comparisons if we use the concepts we introduced in the tests we performed in the previous chapter (*Coverage* and *Error*). The output in the vertical axis is provided in a logarithmic scale to make the figures easier to read.

The line which shows the *Patience limit* of 10 sec also appears in the diagram. For comparison, we also tested it by using three different Trust filters (b>0.5,b>0.6,b>0.7) in Trust propagation. We studied various sizes of communities ranging from 5 to 100 nodes and as can be seen from the diagrams, there are cases where the Response times are within the *Patience Limit*, especially when strong filtering policies e.g. (b>0.7) were applied.

As expected, with the increase in connection speed, the consequences of the Response time follow the inverse trend. The application of stronger filtering policies also decreases Response times. As can be seen in the figures, the curves are positioned in the graph as groups of two each having the same Trust filter. This can be explained because performance is influenced more by the Trust filter rather than the available bandwidth.

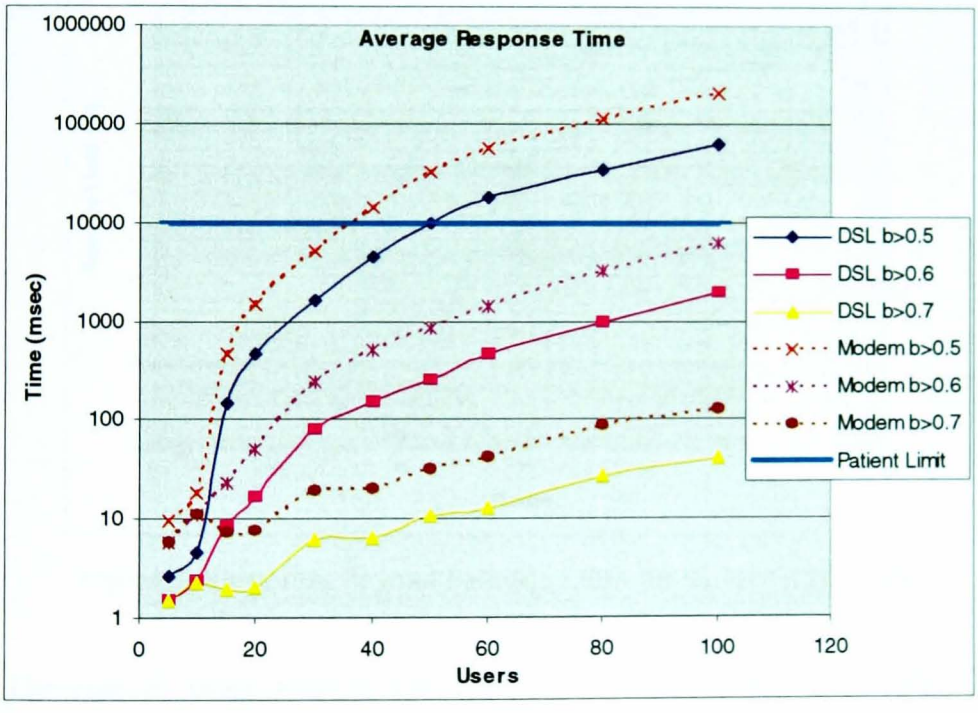

Figure 2. Expected Response times for various communities

Another interesting observation is the following: any combination of parameters which does not employ weak filtering policy in communities of up to 100 users, produces results within the 10 sec limit we set as an acceptable Response time.

Figure 3 shows the percentage of the queries whose response times exceeded the preset threshold of 10 sec and thus were considered unsuccessful. The rate of

uncompleted requests within the threshold can be regarded as the complement of the Success rate of the configuration being examined, so, from now on in the processing of the results we use this value. For comparison we present both results for the cases where a DSL connection bandwidth is used and for an analogue Modem line.

As can be seen in figure 3, no cases using strong Trust filters (b>0.7) have been included because there were no responses measured outside the *Patience limit*. Hence, for every figure from the b>0.7 filtering policy it should be assumed that all queries are within the *Patience limit*. As can also be seen from the diagram, the application of a weak filtering policy (b>0.5) affects the number of cases that go beyond the *Patient limit* significantly, even for low numbers of users (around 50).

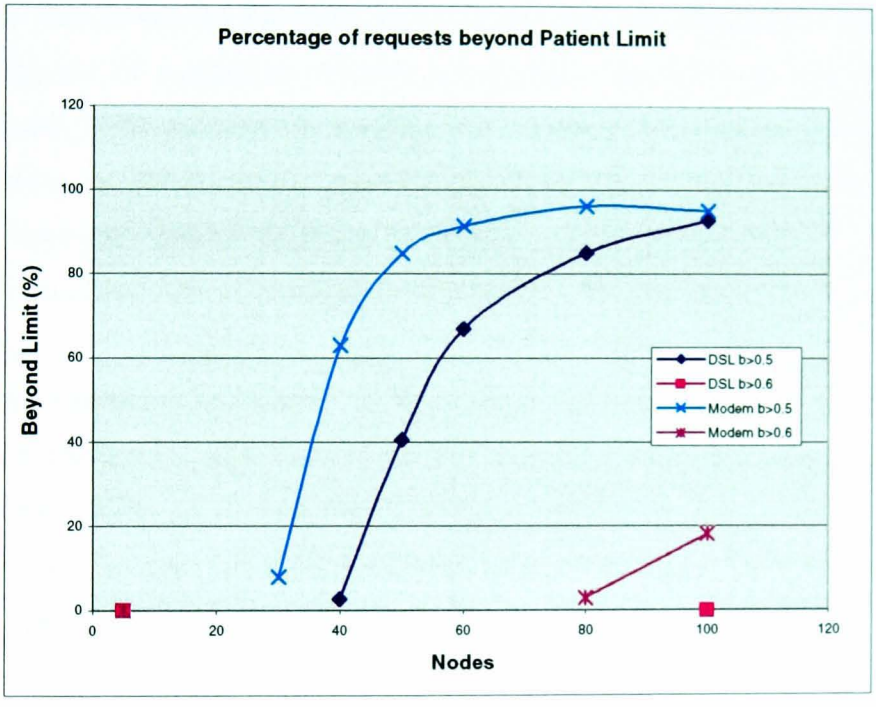

Figure 3. Responses beyond Patience Limit for various trust filters

The case of using Modem and applying the propagation filter b>0.5 seems interesting. In this scenario the maximum percentage of requests beyond the Patience Limit appears in the population of 80 users rather than that of 100 users. Another interesting characteristic is that the curves of Modem and DSL converge in the case of a 100-peer network.

The performance as presented in these figures does not seem adequate to indicate which is the best choice because it does not contain a factor that can be translated into

"*gain*" for each individual case along with some cost, which in the context of our experiment would be the Response time to the queries.

Therefore, we introduce the *Satisfaction factor* (SF), which combines a measure of performance based on the ratio of the successful responses (those which the Response time was within the *Patience limit*) and the Coverage that is achieved when applying a filtering policy. We define *Satisfaction Factor* as:

$$SF = Success\_Rate \cdot NCoverage$$

We can imagine the *Success Rate* as the complement of the percentage of responses that go beyond the Limit. In this way we assume that the rate of success of a combination of parameters (Connection speed, Propagation filter) is inversely proportional to the number of requests that exceeded the *Patience Limit* for that combination. In the previous figure we saw how the complement of *Satisfaction Factor* shapes for various depths of searching and for various Trust filters. In this way, *Success Rate* can be extracted indirectly from the data that were used to create Figure 3.

As we mentioned in Chapter 6, *Normalized Coverage* (NCoverage) is the total number of services for which a user can find opinions through the Trust graph divided by the total number of services that have been rated by all parties. NCoverage also takes care of the fact that there is always some error when trying to predict user ratings. The formula that gives NCoverage is:

$$NCoverage = (1 - \overline{E}) \cdot C$$

where, C is the Computability ratio and E is the average Recommendation error for a particular recommendation. By Computability ratio we mean the number of services reachable for the group of users divided by the total number of services about which opinions can be expressed. Figure 4 shows the NCoverage for three different filtering scenarios and for a maximum propagation distance of 3 hops. The full definition of NCoverage as well as what values it can take for various infrastructures can be found in the preceding chapter.

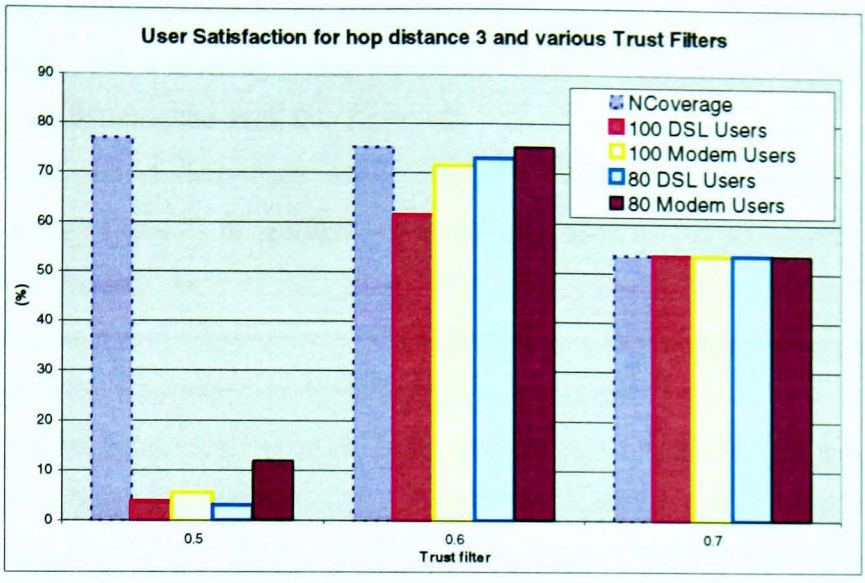

Figure.4 User Satisfaction for various trust filters

Trust Filter	b>0.5	b>0.6	b>0.7
N-Coverage	*76.89*	*75.33*	*53.53*
100 DSL	3.80	61.5	53.53
100 Modem	5.67	71.56	53.53
80 DSL	3.01	72.99	53.53
80 Modem	11.9	75.33	53.53

Figure 5. User Satisfaction

The figure for *Normalized Coverage* that is displayed in Fig. 4 considers only the case of propagation of up to 3 hops because the simulation we ran for the purpose of the performance analysis was done for this configuration only, and therefore excluded other cases. Also displayed is User Satisfaction (SF) for two individual configurations and for various Trust filters and two different sizes of communities (80 and 100 users). We did not perform tests for communities smaller than 80 users in size because in the small communities there is a very high *Success Rate* and, as expected, the *Satisfaction Factor* always reaches 100%.

From the table it can be seen that DSL users get the highest satisfaction for both sizes of communities when the middle (b>0.6) Trust policy is applied. Another interesting characteristic is that with both types of connections (DSL and analogue Modem) when a weak filtering policy for Trust (b>0.5) is used, users get almost no satisfaction. This happens mainly because this policy causes high congestion and as a

result query responses are being received outside the 10 sec *Patience Limit*. As a consequence, such a filtering policy should be avoided even though it provides good Coverage when using the available resources.

Both DSL and Modem users receive the same level of satisfaction when the strong filtering policy (b>0.7) is applied no matter the size of the community. That is because the *Success Rate* for this Trust filter approaches 100% and thus it does not affect the *Coverage* of the relevant combination (hop count=3 and Trust filter>0.7). In this combination *Coverage* and *Satisfaction Factor* match.

Also observe that in all combinations, the highest satisfaction is obtained when a middle-range Trust filtering policy is applied. It can be seen from the diagram that the peak for both the modem and the DSL curves emerges when the Trust filter has been set to (b>0.6).

With respect to the size of the communities, the diagrams show that satisfaction declines as communities grow. All 100 user curves appear to be below those of 80 users. This is expected since the larger the community the higher the number of messages communicated in the network links between the peers and thus the longer the delivery time for those messages to reach their destinations.

From figure 2 it can be seen that there is no significant difference in the Response time between using a slow (analogue modem) or a fast (DSL) connection since both figures have the same trend.

As regards scalability issues, from the diagrams it can be seen that as the number of users increases, the response times increase exponentially (fig. 2), which imposes a scalability problem. Due to a lack of computing resources we are unable to find out what trend the *User Satisfaction* follows for communities that are larger than 100 nodes.

7.9 Conclusion

Decentralized architectures, such as Peer-to-Peer, seem, both in theory and in practice, to be quite suitable for supporting services such as Recommender Systems which are traditionally centralized. In this chapter we have presented quantitative results for running Recommender Systems upon such an infrastructure. The simple analysis we performed was based on real data and it shows that, within the technical

limitations of the current technology, such an architecture can support a distributed Trust-enabled Recommender System effectively. There are some limitations with regard to the scalability of the algorithm we used, which shows that such algorithms do not scale well with the number of users when Trust query propagation is done in distances beyond 2 hops. This has serious effects on usability. Therefore, use of such algorithms is suggested in environments that have sizes no bigger than 100-150 nodes which mainly means medium size corporate networks. Given the drawbacks, one significant advantage of using such algorithm is the greater robustness it provides as opposed to a centralized solution.

We introduced a metric called *Satisfaction Factor* and the experiments show that for the sizes of Peer-to-Peer networks we tested and the variety of connection speeds, the best value for satisfaction is achieved when a medium Trust policy is applied.

As we mentioned, the scope of our study was to provide a macroscopic analysis of the behaviour of the protocol as the user community grows, considering the effects of its operation on the rest of the network infrastructure and its effects on the peers themselves. Our future plan is to build an analytic model that will help us to study these issues from a macroscopic view and would also give an understanding of the protocol performance in extreme situations which were impossible to test due to the lack of experimental data (e.g. propagating Trust beyond 3 hop distances). Comparison with an identical centralized solution with regard to the cost/value ratio as well as the applicability to an existing Peer-to-Peer protocol [75] is also an important future issue.

Even though we ran the tests for relatively small communities of population of 100 users max, in our opinion, there was no reason to employ more than this number of peers in order to derive a Trust recommendation. Even if a real Peer-to-Peer community contains a huge numbers of peers, the best 100 of them that could be involved in a Trust overlay could be selected by using the appropriate Trust propagation filter. In the analysis we did in the previous chapter, the *Prediction Error* of ratings is not affected significantly by the filters used in the Trust propagation. Therefore, we suggest filtering the propagated Trust in cases where the number of peers in the overlay is significantly large and the Response times cannot be kept within the acceptable time limits (Patience Limit). Alternatively some other criteria must be used in order to find the most suitable ones to support a query.

Chapter 8

CONCLUSIONS

8.1 Overview

A distributed Peer-to-Peer Recommender system has been designed and built and tested by simulation. Conventional Recommender Systems run in a centralized manner and our proposed prototype is distributed and is also enhanced with a Trust derivation model that is used to enrich the relationship base of the participants. As shown in Chapter 2, neither the idea of distributed Recommender systems nor Trust in computing environments are new. What we aimed to demonstrate with this study was the benefits that can be achieved by combining them. The value of this study has been two fold: to analyze the concept of a simple distributed Recommender system that uses Subjective Logic algebra for the calculation of Trust and also to provide an assessment of how, and whether, such a system could work in a real environment.

8.2 Assumptions

The nature of the Trust model and its similarity with a "*word-of-mouth*" scheme made it suitable for application in a Peer-to-Peer topology. In spite of the problems that were encountered with the existence of non-Canonical cases during the analysis of Trust graphs, the Subjective Logic enhanced Trust model we built was found to be a successful choice due to its special characteristics (which we had in mind when choosing it) such as support for transitive relationships. The main idea was to engage the entities involved in the Peer-to-Peer infrastructure with the task of carrying messages that encapsulate the trustworthiness of their neighbours.

It is worth emphasising that this idea was based on the assumption that the entities involved would respond with absolute honesty to the Trust queries they were asked to provide opinions for and also that they would not make any alterations to the contents of the query responses that they communicated back to the originators, either deliberately or

as a result of failure. In our judgment, such an assumption seems to be too restrictive for a model that works under the conditions of a real world community. Even though in all the experiments we performed the absolute honesty of the participants was assumed, it would be interesting to examine the consequences of having misbehaving parties in the chain of information delivery. Another important assumption we made is that users were not obliged to wait for unlimited time for their queries to be answered and therefore we introduced a *Patience Limit* which would determine whether queries would be considered successful or not. Furthermore, we assumed that in a real scenario primary Trust relationships would be calculated off-line.

Another implication that was considered seriously when we designed our system was the fact that users are usually reluctant to provide Trust measures about those with which they have interacted because such a process seems tedious when compared with the benefits received. Because of this it seems unreasonable to expect users to express judgments about their counterparts when a transaction they had been involved in has been finished. We therefore introduced a mechanism which derives the Trust measures between pairs of users from their rating behaviour. The latter was a major weakness of Subjective Logic which required the users to assign values of trustworthiness by themselves and which our method overcomes.

8.3 Analysis of the Conclusions of our experiments

In terms of performance of Recommender systems, a significant benefit of using our policy was the reduction of a metric called sparsity. Our experiments showed that using such a Trust model is beneficial for the community that adopts the scheme. Our system seemed to improve the total gain, which in our terminology we called the *NCoverage* factor, when compared with a standard Collaborative Filtering system that does not make use of Trust. In fact, Subjective Logic was helpful in the Trust derivation process in the cases where transitivity was necessary and was found effective for calculating the trustworthiness between two distant entities in a Trust graph and thus helped in making accurate predictions of ratings.

There were also some serious limitations imposed by our method and in the current simple form that we used in our experiments that seemed to cause scalability problems.

Performance analysis showed that the main reason for the scalability problems was the use of the simple flooding algorithm, even though various controls were applied in the forwarding of Trust messages. There is much room for experimentation and optimization in this area, which could lead to more scalable solutions than those we found in our experiments. For example, cacheing techniques could be applied to avoid queries being propagated further to areas of the graph whose topological characteristics and experience base do not change frequently.

A Recommender system obeying the rules we presented here would not scale to massive number of nodes whilst maintaining good performance as measured by *Normalized Coverage*.

The scalability problems were encountered in the case that search for trusted participants is done for a depth of 3 hops from the querying peer. As shown in the experiments any search queries that run for depths of 1 and 2 hops can provide results within reasonable time but a relatively poor set of participants is used, which means insufficient data and thus we did not focus on this case.

Given that a conventional Recommender system would contain the opinions of thousands of users, the limitation of our algorithm to supporting a few hundred of them seems significant. The proposed solution leads to a partitioned network in which no node will be able to calculate Trust with all of the nodes that take part in the system. One idea would be to apply the algorithm only to those datasets characterized by high sparsity, since the Trust enabled algorithm seems to be more helpful in those cases of datasets in which many users encounter the cold-start problem due to a lack of available data.

Nevertheless, the study was done with the assumption that the algorithm would be able to run on a contemporary network of conventional workstations connected together using a bus architecture which would have limited capacity and with uniformity in the connection speeds of the nodes.

Even though the tests we performed were for small communities of 100 nodes, due to the high computation effort that the simulation process required, the trend showed that the scalability barrier was very close to this value and any effort to go beyond the 100 nodes barrier would add significant cost to the performance of the system. The main problems for scalability come from the use of the Subjective Logic algebra itself and especially from the way that the derivation of secondary Trust is done. As we mentioned

in the description of the protocol, in order to avoid having dependencies in the opinion graphs analysis, whole groups of Trust vectors have to be transferred from the Trust destination (trustee) back to the query originator (trustor) where the derived Trust value can then be decided. The weak point of this design is that relatively lengthy messages have to be communicated via the network thus increasing congestion.

In the calculation of Response times we considered the time taken for all alternative paths, which were composed of primary Trust vectors, to be transmitted back from the query destinations to the origin. If we assume that a threshold is set, beyond which any message received will be discarded, we can assume that secondary Trust will be built using only the evidence collected within the threshold period. There is a danger in this that there could be cases where Trust might be calculated wrongly based on incomplete data. A future issue is to find the relationship between the percentage of Trust vectors collected and the accuracy of the calculated secondary Trust. The ideal value for secondary Trust is the value calculated considering all the Trust vectors.

The crucial point of our method is that the node where the query originates does not always have a complete picture of the network and thus is unable to know the real number of alternative paths between the trustor and the trustee. Therefore, in a real world case scenario it is impossible to know for certain the size of the expected query response and how long it will take to collect.

Given that every alternative path that links the destination (trustee) with the query initiator usually strengthens the derived trustworthiness, we understand how important it is for the property that expresses the *belief* in the resulting Trust vector to include as many paths as possible so that the actual value will be the least distorted. Conversely, the property that expresses the *uncertainty* of the resulting Trust vector decreases with the inclusion of more alternative paths.

As can be seen from the results of the Performance Analysis, response times are affected not by the networking infrastructure used but mostly by the Trust filtering policy itself. The Trust filtering policy affects the size of the returned graphs which translates to the number of calculations that have to be performed in the graph simplification phase. This leads to the conclusion that the algorithm is more demanding in computation than in network resources. This is encouraging because the bottleneck in CPU resources gives a further chance for improving the situation by reducing the time needed to carry out an

elementary graph analysis operation. In the computations we performed in the simulation we assumed that a simple simplification operation in Java would take about 1msec. Re-writing the part of the program which does the graph analysis using another language, such as C for example, could significantly improve the situation and achieve lower times. In this case, the overhead will be transferred to the network connections between the nodes. The improvement in both CPU utilization and networking infrastructure will help towards supporting larger Trust networks. In contrast to other Peer-to-Peer algorithms such as the Gnutella searching algorithm whose scalability is restricted by the networking infrastructure alone due to the small amount of computation it needs, our algorithm demands that both the computing and the network infrastructure be improved in order to expand the scalability barrier.

Finally, we should mention the advantages of a distributed Recommender system in comparison to a centralized solution, which come from the increased robustness of Peer-to-Peer networks together with the higher security in Trust derivation they can offer. In such an architecture the problem of having a single point of failure does not exist especially in the case when using the Trust derivation algorithm in a distributed topology. That is because queries are supported by many entities. If some node involved in a query fails then the system will still be able to answer queries and provide results. Even though the result will be distorted since not all the available sources will be considered in its computation, the system will continue working and providing recommendations. As regards the security issue, Chapter 4 describes an idea for how an attack might be intercepted.

8.4 The future infrastructure

The main idea on which the design of the whole system was based, was to build an unstructured network that works as a single tier system and that does not rely on central entities to work. To fulfill the requirement for having nodes totally independent from third parties and other centralized entities we designed our Reputation system like other Peer-to-Peer systems that are used for exchanging content on the web [75]. Most of those networks work almost independently of centralized entities.

As we saw in the performance diagrams, (see Chapter 7) the increase in the Response

Time is almost linear except in the cases where weak Trust filters are used. In all cases where medium and strong filtering were used, the threshold of 10 sec was not exceeded. If today with the current technology (average CPU power and average network capacity) a Trust network would efficiently support 150 well satisfied users, in the future when the infrastructure improves, this number may change to thousands or more, giving significant hope that Trust enabled Peer-to-Peer networks, like the one we propose, will be able to run on a wide scale. As a consequence the usability will also improve.

An example of an unstructured network that started with a low technological infrastructure is Gnutella. Searches in Gnutella were done using a flooding technique. Its designers restricted the number of nodes that could be seen by a single node (called a horizon) to around 3000, by setting a limitation on the number of hops that a message could be propagated before it expired (TTL). This led to a partitioned network where each user was only able to see and search the contents of around 3000 other nodes. In the same way, in our case where the Trust network is also partitioned, each node will be able to see and receive Trust influences from around 150 other counterparts. We believe that even if the technological infrastructure never reaches the expected point of maturity, this number is quite sufficient for entities to be involved in a Trust query. As with the Gnutella example, we could set our own restrictions in the propagation filters for Trust queries. A belief filter value of 0.6 looks quite reasonable since it does not significantly affect the Prediction Error of the Trust queries as was shown in Chapter 6.

8.5 Short term plans

The purpose of our study was to provide a macroscopic analysis of the behavior of our proposed protocol with the growth of the user community. In this study we considered the effects of the protocol's operation on the rest of the network infrastructure and also on the peers that adopt it. Our future plan is to build an analytic model that will help us to study these issues from a different perspective and also to give an understanding of the protocol performance in extreme situations that we have been unable to test due to lack of experimental data (e.g. propagating Trust beyond 3 hops).

Our future plans include: the alteration of the assumptions we have set and the study of the behaviour of the proposed system in cases where there may be dishonest Trust

query replies.

As regards the requirement for the existence of a common purpose in order for Trust relationships be used in a transitive way, we intend to alter the assumption we have made by saying that "*users are good recommenders since they know the context of recommendation well*" and re-run the experiment using pure Recommender Trust in the transitive chains. Since this kind of Trust is sometimes impossible to derive due to lack of data, it is easier to approximate its value. However, this requires that, using existing evidence, we somehow model the Trust that is placed on the recommender's abilities for a given purpose.

We also plan to investigate the model from a graph theoretical perspective and examine how the characteristic properties of the Trust graph such as the *Clustering Coefficient* might affect the quality of recommendations or else *Prediction error* and *Coverage*. We suspect that clustered communities benefit more by the deployment of such a system with regard to *Coverage* since more experiences can be reached through a bridging node in the graph. Moreover, a closer analysis of each user individually could show with greater clarity the characteristics of those user nodes that benefited most from their contribution to the system.

8.6 Future improvements

Going one step further, and inspired by the examples of Peer-to-Peer systems developed for content exchange, our system could be enhanced to a layered system with nodes running at different tiers of responsibilities. An example that adapts to our case would be to use the nodes at the top of the hierarchy as *Hubs* which communicate Trust messages about their members in their local neighbourhoods. This will reduce the demand for large communities of nodes. Each *Hub* could serve a larger number of second level nodes which will act as simple (2^{nd} layer) peers in the system. The *Hubs* could be connected directly with the nodes constructing a virtual sub-graph topology managed by the higher layer node. During the initialization of a 2^{nd} layer node a sign-up procedure to its local Reputation server could be performed, during which the new node would transfer all its ratings about the items it has experienced. The *Hub* then would be able to start calculating the Similarity and the Trust values between the new member and all the

others that belong to the same neighbourhood and in this way it would maintain virtually the local *"web-of-Trust"* in its memory. The Trust queries, instead of being broadcast from the simple nodes to everywhere possible, could be sent directly to their local *Hub* which would then manage the query using its own resources. In the case that the query could be resolved by the local community alone e.g. if there are lots of trusted parties and common experiences within the same community, the hub would perform the calculations for the derived Trust value using the experiences of the local neighbours and return the reply directly to the querying entity. Conversely, in the situation where the local results are poor the *Hub* could ask for the contribution of other *Hubs* which would reply with data from their local Trust graphs.

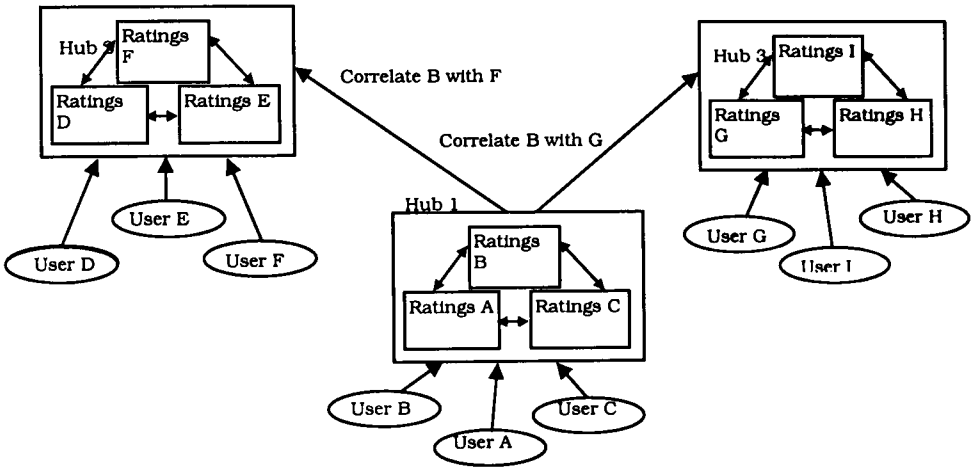

Figure 1.

The combination of the local Trust graph together with the Trust graphs from other *Hubs* as well as the relationships between them constitute a much wider graph for a querying entity to consult. The use of *Hubs* is also expected to reduce network traffic, which is the main bottleneck for graph analysis, but it will cause some extra overhead to the *Hubs* themselves which will have to carry out all the correlation computations for all users that have registered with them.

In Figure 1 we see the operation of *Hubs* as local Reputation engines where in the case of the existence of sufficient counterparts in a relationship the Trust query is resolved locally by the correlation of the local counterparts. E.g. A, B and C.

Another important advantage that *Hubs* could give is their use as cache engines. *Hubs* are aware of the changes in the topologies of the sub-graphs they manage and therefore

they could keep the contents of the graphs cached until they are asked to serve a Trust query about some entity that is under their supervision and then provide the reply instantly if no topological changes have happend.

Given that it is infeasible to know the contents of the graph that will be returned as an answer to a Trust query before the query is initiated, the use of a filter to decide how many responses are required for a satisfactory reply graph is necessary. Our study cannot give such an answer since it was entirely based on experimental. Also, as the two-tiered solution leads to a partitioned network, it would be very interesting to examine how partitioning affects the quality of the results.

8.7 Epilogue

A general question that arises from this study is how beneficial the use of distributed Recommender systems is, given that the existing solutions which run on the web today as centralized services are doing well in providing Recommendations. One crucial question that needs to be answered is the following: is it worthwhile to deploy such a distributed system since comparison with a centralized solution shows that the benefit it offers is negligible?

A second question that arises is: given that centralized solutions will not be able to give satisfactory results as the user communities grow, how successful might a distributed solution be that makes use of the Trust that can exist between the members of a community?

To sum up and give some answers to these two questions, our guess is that centralized solutions even though they might appear to be workable solutions today due to the high sparsity of the datasets, will not be able to provide the service with the same level of performance in a few years time when their datasets will be denser. This, soon or later, will be the subject of research for new methods of providing query responses within acceptable time limits. On the other hand, decentralized systems such as Peer-to-Peer which distribute the computation load already show great promise for solving the problem.

Nevertheless deployment must be done in a rational way. Trust enabled systems can provide solutions to the plain centralized systems that are only based on similarity. As

was seen, they can also help in the case of sparse datasets and help in tackling the cold-start problem for new users. The framework of Subjective Logic can help in the creation of Recommender systems of this kind and it seems that its features fit well with the requirements of a decentralized Trust-enabled Reputation system. (e.g. it supports transitivity and thus a graph derived by a "*word-of-mouth*" scheme can be represented). However, from the performance model we studied, it seems that the special requirement for opinion independence would lead to a less successful solution than we first thought. The scalability problems that we showed during the testing of the first solution necessitate a more careful adaptation of Subjective Logic theory to the problem of recommending opinions within a society. Our proposed improvement is the use of multi-tiered systems in which the duties within the community will be carefully allocated.

As a general conclusion, our experiments showed that a Trust-enabled Reputation system can be helpful for the participants in a Peer-to-Peer community in forming decisions about resources they might, or might not, like. As a result, the quality of the resources offered is preserved since the participants are able to make the best choices and, more importantly, aided by the community members themselves. The fundamental characteristic of such scheme, which is independence from central entities, can assure the provision of unbiased, robust and secure consultation services on the Internet. As regards the applicability of the proposed ideas, the scalability problems that are mentioned can be overcome by using existing schemes and techniques that have been used in Peer-to-Peer topologies in the past.

APPENDIX A

The structure of *MOVIELENS* Database

```
================================================================================
USERS FILE DESCRIPTION
================================================================================

User information is in the file "users.dat" and is in the following
format:

UserID::Gender::Age::Occupation::Zip-code

All demographic information is provided voluntarily by the users and is
not checked for accuracy.  Only users who have provided some demographic
information are included in this data set.

- Gender is denoted by a "M" for male and "F" for female
- Age is chosen from the following ranges:

        *  1:  "Under 18"
        * 18:  "18-24"
        * 25:  "25-34"
        * 35:  "35-44"
        * 45:  "45-49"
        * 50:  "50-55"
        * 56:  "56+"

- Occupation is chosen from the following choices:

        *  0:  "other" or not specified
        *  1:  "academic/educator"
        *  2:  "artist"
        *  3:  "clerical/admin"
        *  4:  "college/grad student"
        *  5:  "customer service"
        *  6:  "doctor/health care"
        *  7:  "executive/managerial"
        *  8:  "farmer"
        *  9:  "homemaker"
        * 10:  "K-12 student"
        * 11:  "lawyer"
        * 12:  "programmer"
        * 13:  "retired"
        * 14:  "sales/marketing"
        * 15:  "scientist"
        * 16:  "self-employed"
        * 17:  "technician/engineer"
        * 18:  "tradesman/craftsman"
        * 19:  "unemployed"
        * 20:  "writer"
```

Sample Data:

ID	Genre	Age	Occupation	Experiences	ZIP
11	F	25	1	137	04093
12	M	25	12	23	32793
13	M	45	1	108	93304
14	M	35	0	25	60126
15	M	25	7	201	22903
16	F	35	0	35	20670
17	M	50	1	211	95350
18	F	18	3	305	95825
19	M	1	10	255	48073
20	M	25	14	24	55113

==========
MOVIES FILE DESCRIPTION
==========

Movie information is in the file "movies.dat" and is in the following format:

MovieID::Title::Genres

- Titles are identical to titles provided by the IMDB (including year of release)
- Genres are pipe-separated and are selected from the following genres:

  * Action
  * Adventure
  * Animation
  * Children's
  * Comedy
  * Crime
  * Documentary
  * Drama
  * Fantasy
  * Film-Noir
  * Horror
  * Musical
  * Mystery
  * Romance
  * Sci-Fi
  * Thriller
  * War
  * Western

- Some MovieIDs do not correspond to a movie due to accidental duplicate entries and/or test entries
- Movies are mostly entered by hand, so errors and inconsistencies may exist

Sample data:

ID	TITLE	GENRE
1	Toy Story (1995)	Animation|Children's|Comedy
2	Jumanji (1995)	Adventure|Children's|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama
5	Father of the Bride Part II (1	Comedy
6	Heat (1995)	Action|Crime|Thriller
7	Sabrina (1995)	Comedy|Romance
8	Tom and Huck (1995)	Adventure|Children's
9	Sudden Death (1995)	Action
10	GoldenEye (1995)	Action|Adventure|Thriller

==========

RATINGS FILE DESCRIPTION

==========

All ratings are contained in the file "ratings.dat" and are in the following format:

UserID::MovieID::Rating::Timestamp

- UserIDs range between 1 and 6040
- MovieIDs range between 0 and 3592
- Ratings are made on a 5-star scale (whole-star ratings only)
- Timestamp is represented in seconds since the epoch as returned by time(2)
- Each user has at least 20 ratings

Sample data:

User	Movie	Rate	Timestamp
1	1193	5	978300760
1	661	3	978302109
1	914	3	978301968
1	3408	4	978300275
1	2355	5	978824291
1	1197	3	978302268
1	1287	5	978302039
1	2804	5	978300719
1	594	4	978302268
1	919	4	978301368

APPENDIX B

In this appendix we present the source code of the program that was used to carry out the performance analysis. The reason why we present it here is because we did not use a conventional emulation tool in the experiment. The code is mainly written in java and uses calls to a database object where the MovieLens dataset is stored.

```
import java.net.*;
import java.text.DateFormat;
import java.io.*;
import java.util.*;
import java.util.EventListener;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

class ReportRec {          // For every request there is a record of performance
    int ReqID;             // The unique ID of the request.
    String request;        // The request as Origin->Target
    int timeStart;         // when the request initiated
    int timeEnd;           // when the request serviced
    int numOfPaths;        // Sum up all the vectors of the PayLoad
}

class StatisticRec {
    int StartTime;
    int Unique_req_ID;
    int payload;
}

class KnowledgeRec {   // This is used to know in a case of a reply has to be sent back, through which node
it came from
     int ID;           //  request ID, After it is used then it should be cleared.
     int From;         //  where it came from
}

class TCPreq {
    int TTL;            // how far can be propagated to.
    int Waiting;        // how long has been wating in the stack
    int Destination;    // which node it tries to reach if it is a query only
    int ID;             // useful to know if request tracking is required.
    int Type;           // Type of the request ( 0: query / 1: reply )
    int From;           // the node ID of where it came from
    int Origin;         // The node where it started from
    int TimeStamp;      // When the request was born.
    int Payload;     // The hops followed to reach destination
}

class Peer {
    int ID;
    int [] Link = new int[201];          // Table of links.
                                         // Every peer can be linked with up to 100 other peers
    ArrayList Stack = new ArrayList();   // request stack (TCP/IP) contains unexecuted requests.
                                         // Every pair contains
                                         //         i) type of request
                                         //        ii) time it takes to expire.
    ArrayList KnowledgeDatabase = new ArrayList();
                                         // Database of knowledge of which ID came from which node
                                         // The elements are KnowledgeRec records.
}

// ---------------------------------------------------------------------------------------------

public class P2Sim_cl_new {

     static ArrayList ReportDataBase = new ArrayList();
     static Peer [] Nodes = new Peer[201];              // The static community of 100 nodes
     static int L = 3;                                  // Frequency at which the requests are issued in
                                                        // the system by the user (5 per min)

     static int MaxNumOfUsers = 50;
     static int time = 0;                       // Time counter starts form:
     static int StopTime = 1200000;             // miliseconds to run for
     static int Bandwidth = 7168;               // bandwidth in bytes/sec (MODEM)     - (for512DSL is 65535)
     static int initial_TTL = 3;
     static int PatientLimit = 10000;           // Patiemt Limit is set to 10 sec

     static int MsgLen = 50;                    // size of a single message in bytes.
```

```
     static int MaxID=0;                                  // define and fill registry with node information
     static mySQL_class SQL= new mySQL_class();
     static ArrayList Statistic = new ArrayList(); // database which is used to store the results

     // ------------------------------------------------------------------------

     static void report() {

         for (int i=1;i<MaxNumOfUsers;i++) {
             System.out.println();
             System.out.print("("+i+") ");
              for (int j=0;j<MaxNumOfUsers;j++) {
                if (Nodes[i].Link[j]!=0) {
                     System.out.print(i+"->"+Nodes[i].Link[j]+" ");
                } // if
              } // for
              System.out.println();
              System.out.print("Stacksize:"+Nodes[i].Stack.size()+" contents:");
              for (int k=0;k<Nodes[i].Stack.size();k++) {
                  TCPreq StackElement = new TCPreq();
                  StackElement = (TCPreq)Nodes[i].Stack.get(k);
                  // ID,DESTINATION,TTL,TYPE,WAITING.
                  System.out.print("
"+StackElement.ID+":"+StackElement.Destination+":"+StackElement.TTL+":"+StackElement.Type+":"+StackElement.
Waiting);

              } // for
              System.out.println();

              //System.out.print("Knowledge:"+Nodes[i].KnowledgeDatabase.size());
              for (int j=0;j<Nodes[i].KnowledgeDatabase.size();j++) {
                  KnowledgeRec KR = new KnowledgeRec();
                  KR=(KnowledgeRec)Nodes[i].KnowledgeDatabase.get(j);
                  System.out.print(" "+KR.ID+":From:"+KR.From+" ");
              }
              System.out.println();
         } // for

     } // report

// ------------------------------------------------------------------------
     static void build_network() {

          String Query="";
          for (int i=1;i<=MaxNumOfUsers;i++) {
               Query = "Select distinct User2 from Trust where bdu>=0.500 and User2<="+MaxNumOfUsers+" and
User2>0 and User1="+i+" and common>=5";

               Peer N = new Peer();
               N.ID = i;
               Nodes[i] = N;

               String Output = SQL.ExecuteSQL(Query,0);
               StringTokenizer DataLine =  new StringTokenizer(Output.trim(),"\n");

               int j=0;
               while (DataLine.hasMoreTokens()) {
                      String nextNumber = (DataLine.nextToken()).trim();
                      int counterpart=Integer.valueOf(nextNumber).intValue();
                      j++;
                      Nodes[i].Link[j]=counterpart;  // Linking the nodes togeather and build the graph.

               } // while
        } // for
     } // build network

// ------------------------------------------------------------------------
     public static void main(String [] args) {

         System.out.println("P2P Simulator. G.Pitsilis");

         if (args.length<1) {System.out.println("Usage: java P2Sim_cl_new <users>"); System.exit(1); }
         MaxNumOfUsers = Integer.valueOf(args[0]).intValue();

         System.out.println(MaxNumOfUsers);
         System.out.println("*"+SQL.connect("localhost","root","","MovieLens"));

         build_network();          // build the network using the vectors from the database
                                   // At this point every node knowns every other noded that
                                   // is connected to.
         while (time<StopTime) {
                    time++;
                    if (time%1000==0) { System.out.println("("+time+")"); }

                    // For all nodes check if there is some query to run.
                    for (int i=1;i<=MaxNumOfUsers;i++) {
                             double Probability = (double)(L) / 60000.0;  // probability that a request is
```

```
                                                                    // being issued within a single
                                                                    // msec
                        double P = Math.random();
                        if (P<=Probability) {    // User decides to initiate a query
                                                 // about some other user X
                                                 // e.g. find the trustworthiness of x

                                                 // Do not initiate any request between the
                                                 // (time limit-patient limit)
                            int f=(int)(Math.random()* MaxNumOfUsers);
                            int unique_request_ID = initiate_test_request(i,f);

                                                 // initiate the statistic procedures
                            ReportRec NR = new ReportRec();
                            NR.ReqID = unique_request_ID;
                            NR.timeStart = time;
                            NR.request = String.valueOf(i)+"->"+String.valueOf(f);

                                                 // A new record in the report database is created
                            ReportDataBase.add(NR);
                        } // if
                        update_nodes_requests(i);      // refresh all nodes TCP stacks with requests
                                                       // replies have to be initiated.
                } // for
    } // while

    print_report();
    System.out.println("FIN:"+time);

} // main

// ------------------------------------------------------------
static void update_nodes_replies(int node) {
    // The replies
    for (int k=0;k<Nodes[node].Stack.size();k++) {
         TCPreq cTCP = new TCPreq();
         cTCP = (TCPreq)Nodes[node].Stack.get(k);

         cTCP.Waiting = cTCP.Waiting - 1;
    } // for
}
// ------------------------------------------------------------
static void update_nodes_requests(int node) {

    // This method is used to update the status of each request in the stack of node (node)
    // Is called every one millisecond.

     for (int k=0;k<Nodes[node].Stack.size();k++) {   // for all elements in the TCP stack of the node

            TCPreq cTCP = new TCPreq();
            cTCP = (TCPreq)Nodes[node].Stack.get(k);  // Get the TCP request
                                                     // decrease expiry time by one msec

            // The delay time should be analogous to the stack size or how many
            // requests are waiting in the queue.

            // 100 is the maximum size of the stack.

            cTCP.Waiting = cTCP.Waiting - 1;
                                                     // and put it back by removing the element
                                                     // and put back the new one
            if (cTCP.Waiting<=0) {
                                                     // if the time has expired, propagate the
                                                     // request to all neighboring nodes except the
                                                     // one where it came from
                                                     // find first which the neighbors are.
                for (int linkID=0;linkID<200;linkID++) {

                    int s = (Nodes[node].Link[linkID]);
                    if (s > 0) {                     // if there is a link from this position in
                                                     // the link table then decrease TTL-1
                                                     // s: other node that the current one is linked
                                                     // to
                                                     // req : request that is to be propagated.
                        TCPreq req = (TCPreq)Nodes[node].Stack.get(k);

                        if (s!=req.From) {               // If it is not the one where it came from
                            propagate_request(s,req,node); // node is the id of the node where the
                                                           // request is propagated from
                        } // if
                    } // if

                } // for
                // removing it first from the old node's TCPstack.

                TCPreq T = new TCPreq();
                for (int w=0; w<Nodes[node].Stack.size(); w++) { T=(TCPreq)Nodes[node].Stack.get(w); }
                Nodes[node].Stack.remove(k);             // remove it from the stack after
```

```
                                                        // propagation or expiration
                for (int w=0; w<Nodes[node].Stack.size(); w++) { T=(TCPreq)Nodes[node].Stack.get(w); }
            } else {
                    Nodes[node].Stack.set(k,cTCP);      // replace the old TCP req with the
                                                        // refreshed one.
            } // if cTCP waiting
        } // for

} // redfresh_nodes_requests
// ------------------------------------------------------------
static void propagate_request(int node,TCPreq req,int FromNode) {
                                                        // node: which node is looked for
                                                        // TCPreq: request
                                                        // FromNode: where the request came from.
    if (node == req.Destination) {                      // here goes the code that will have to
                                                        // be executed if the destination is found
                                                        // In that case it does not propagate any
                                                        // further.
       req.Payload = initial_TTL-req.TTL+1;
       Sumas(req.ID, req.Payload, time, node);

       return;
    } // if

    req.Payload = initial_TTL-req.TTL;

    if (req.TTL>0) { // it must be broadcast further

         int endsat=0;
         for (int f=0;f<200;f++) {
             if ( Nodes[node].Link[f]!=0 ) { endsat = f; }
         }
         req.Waiting = (int)(MsgLen * endsat / Bandwidth)*1000;

                                                        // 28 is the message length waiting time
                                                        // before propagation calculated msec

                                                        // Search if the record exists already in
                                                        // the knowledge database
         boolean exists=false;
         for (int j=0;j<Nodes[node].KnowledgeDatabase.size();j++) {
             KnowledgeRec KR = new KnowledgeRec();
             KR=(KnowledgeRec)Nodes[node].KnowledgeDatabase.get(j);
             if ((KR.ID==req.ID)&&(KR.From==FromNode)) {
                 exists=true;
             } // if
         } // for

                                                        // if it does not exist then and insert it
         if (exists==false) {                           // Add it to the knowledge base
              KnowledgeRec KR = new KnowledgeRec();

              KR.ID = req.ID;                           // the ID of the
              KR.From = FromNode;                       // The node where the request has been
                                                        // forwarded from
              Nodes[node].KnowledgeDatabase.add(KR);
         }

         TCPreq newReq = new TCPreq();

         newReq.Destination = req.Destination;
         newReq.ID = req.ID;
         newReq.Waiting = (int)(MsgLen * endsat / Bandwidth)*1000;

         newReq.TTL = req.TTL-1;

         newReq.Origin = req.Origin;
         newReq.Destination= req.Destination;
         newReq.TimeStamp = req.TimeStamp;
         newReq.Type = 0;
         newReq.From = FromNode;

                                        // Here the request is added to the stack of the new node.
         Nodes[node].Stack.add(newReq);
    } // if
} // propagate_request

// ------------------------------------------------------------
// this is called by a target node when a reply has to be generated
static void initiate_reply(TCPreq request) {
                                         // The requestID is the unique code number used for
                                         // distinguishing the origin of a request
         TCPreq k = new TCPreq();
                                         // setup to propagate to a hop distance of 3
         k.Waiting = 1000;               // waiting time before propagation 100 msec
         k.Destination = request.Origin; // the destination of the reply which must be the query
```

```
            k.ID = request.ID;                          // originator
                                                        // The ID is set the same with the request in order to
            k.Type = 1;                                 // follow the same path back to the originator
            k.Payload = request.Payload;                // Replies are of type 1
                                                        // The payload is cleared here and is filled with nodes as
            k.Origin = request.Destination;             // to traverses back to the origin.
            k.From = k.Origin;
            System.out.println("("+time+") node "+k.ID+" initiates a reply to "+request.Origin+" with
ID:"+k.ID+" and it has payload "+k.Payload);

                                                        // Search for the Statistic record using the unique ID.
                                                        // find and replace the record which has the ID
            int existsAt = 0;                           // with a new one which has the
            int j;
            for (j=0;j<Statistic.size();j++) {
                StatisticRec SR1 = new StatisticRec();
                SR1 = (StatisticRec)Statistic.get(j);
                if (SR1.Unique_req_ID==k.ID) {
                   existsAt = j;
                   StatisticRec SR = new StatisticRec();
                   SR =(StatisticRec)SR1;
                   SR.payload = k.Payload;
                   Sumas(SR.Unique_req_ID,SR.payload,(time-SR.StartTime),1);
                   Statistic.remove(j);
                   Statistic.add(SR);
                } // if
            } // for

    }
    // ---------------------------------------------------------------------------
    static void Sumas(int RID,int payload, int time,int node) {

            // Add it to the statistic report
            boolean found=false;
            for (int j=0;j<ReportDataBase.size();j++) {
                ReportRec RR = (ReportRec)ReportDataBase.get(j);
                if (RR.ReqID == RID) {
                   found=true;
                                                        // Create a temporary new record to replace the
                                                        // existing one
                   ReportRec NewTempRec = new ReportRec();
                   if (Nodes[node].Stack.size()!=0) {
                       NewTempRec.numOfPaths = RR.numOfPaths + payload *
Nodes[node].Stack.size()/MaxNumOfUsers;
                   } else {
                       NewTempRec.numOfPaths = RR.numOfPaths + payload;
                   }
                   NewTempRec.ReqID = RID;
                   NewTempRec.timeEnd = time;
                   NewTempRec.timeStart = RR.timeStart;
                   NewTempRec.request = RR.request;
                   ReportDataBase.remove(j);                // replacing the old with the new record.
                   ReportDataBase.add(NewTempRec);
                   return;
                } // if

            } // for
            if (found=false) {                          // create a new account ID
               ReportRec NewReportRec = new ReportRec();
               NewReportRec.ReqID = RID;
               NewReportRec.numOfPaths = payload * Nodes[node].Stack.size();
               NewReportRec.timeEnd = time;  // time elapsed
               ReportDataBase.add(NewReportRec);
            }
    } // Sumas

    // ---------------------------------------------------------------------------

    static void print_report() {
          int countNumOfPaths=0;
          int countBeyondPatientLimit=0;
          double AvgWaitingTime=0.0; int Elements=0;
          System.out.println("---------------- REPORT --------------------------");
          System.out.println(" ID, Time Start, Time End, Vectors, return graph time,avg waiting time ");
          for (int j=0;j<ReportDataBase.size();j++) {
                ReportRec RR = (ReportRec)ReportDataBase.get(j);
                double ReturnGraphTime = (RR.numOfPaths*25.0+10)/Bandwidth*1000;
                double CalcGraphTime = (RR.numOfPaths*1); // 100 msec each simplification eperation
                System.out.print(RR.ReqID+" "+RR.request+" "+RR.timeStart+" "+RR.timeEnd+"
"+RR.numOfPaths+" "+ReturnGraphTime);
                if (RR.numOfPaths>0) { countNumOfPaths++; Elements++;
                   AvgWaitingTime=AvgWaitingTime+(RR.timeEnd-
RR.timeStart)+ReturnGraphTime+CalcGraphTime;
                   System.out.println("     "+AvgWaitingTime/Elements+"      "+((RR.timeEnd-
RR.timeStart)+ReturnGraphTime+CalcGraphTime)+"    #"+AvgWaitingTime+" "+Elements);
```

```
                    if ((RR.timeEnd-RR.timeStart+ReturnGraphTime+CalcGraphTime)>10000) {
countBeyondPatientLimit++; }
                } else { System.out.println(); }
            } // for
            System.out.println("Only the "+((double)countNumOfPaths/(double)ReportDataBase.size()*100.0)+"%
has replied");
            System.out.println("The
"+((double)countBeyondPatientLimit/(double)ReportDataBase.size()*100.0)+"% is beyond the patient limit");

            System.out.println("Number of users:"+MaxNumOfUsers);
            System.out.println("Bandwidth:"+Bandwidth);
            System.out.println("TTL:"+initial_TTL);
        }

        // ------------------------------------------------------------------------

        static int initiate_test_request(int nodeID,int to) {

                    TCPreq k = new TCPreq();
                    k.TTL = initial_TTL;                          // setup to propagate to hop distance of 3
                    k.Type = 0;                                   // Is of type request

                    int endsat=0;
                    for (int f=0;f<200;f++) {
                        if ( Nodes[nodeID].Link[f]!=0 ) { endsat = f; }
                    }

                    k.Waiting = (int)(MsgLen * endsat / Bandwidth)*1000;
                                                                  // 28 is the message length waiting time
                                                                  // before propagation calculated msec
                    k.Origin = nodeID;                            // The originator
                    k.From = k.Origin;
                    k.TimeStamp = time;                           // When it was born
                    k.Destination = to;                           // where it goes to
                    Nodes[nodeID].Stack.add(k);
                    MaxID++;
                    k.ID = MaxID;

                    StatisticRec SR = new StatisticRec();         // here a record in the database is created
                    SR.StartTime=k.TimeStamp;
                    SR.Unique_req_ID=k.ID;
                    Statistic.add(SR);
                    return(k.ID);
        }

        // ------------------------------------------------------------------

}
```

The next piece of code in order to run requires connection with the MovieLens database which will be provided in the format we mention in Appendix A. Next follows the classes required to achieve connectivity with the database via a JDBC interface. The following code requires that the appropriate JDBC drivers have been installed in the system which will be used for the analysis. The source code that is following has been prepared to run with MySQL Connector/J set of JDBC drivers. It can be ignored in the case that some other driver or other type of access is used in the data.

```
class mySQL_class extends Object {

  static Connection con;

  static String connect(String Server,String User,String Password,String Database) {
    try {
          // Load the driver
          Class.forName("com.mysql.jdbc.Driver").newInstance();
          String url = "jdbc:mysql://"+Server+"/"+Database+"?user="+User+"&password="+Password;
          con =  DriverManager.getConnection(url);
          return "Connection established";
        }
    catch ( Exception e ) { return e+" :Error while connecting to the Database"; }
  }

  // --------------------------------

  static String Disconnect() {
     try {
           con.close();
           return "Disconnected";
         }
     catch ( Exception e ) { return "Error while Disconnecting from the Database"; }
   } // disconnect

  // ------------------------

  static void ExecuteSQLupdate(String query) {

    try {
           java.sql.Statement stmt = con.createStatement();
           int output = stmt.executeUpdate(query);
           stmt.close();
    } catch (Exception e) { e.printStackTrace(); }

  } // ExecuteSQLUpadte

  // -------------------------------------

  static String ExecuteSQL(String query,int head) {   // head should take value 1:YES or 0:NO

     String Otpt="";

     try {
           java.sql.Statement stmt = con.createStatement();
           java.sql.ResultSet rs = stmt.executeQuery(query);

           // Getting Metadata
           java.sql.ResultSetMetaData meta = rs.getMetaData();
           int columns = meta.getColumnCount();

           for (int i=1;i<=columns;i++) {
                 Otpt = Otpt + meta.getColumnLabel(i) + " ";
           }
           Otpt = Otpt + "\n";

           if (head==0) { Otpt = ""; }

           String ReportLine;

           while( rs.next() ) {
              ReportLine = "";
              for (int i=1; i<=columns; i++) {
                    ReportLine = ReportLine + rs.getString(i) + " ";
              }
              Otpt = Otpt + ReportLine + "\n";
           }
           rs.close();
           stmt.close();
         }
      catch (Exception e) { e.printStackTrace(); }
```

```
 return Otpt;
}

// ---------------------------------------

void getDataInfo() {

 try {
       java.sql.DatabaseMetaData md = con.getMetaData();

       if (md==null) {
                 System.out.println("No Database Meta Data");
               } else {
                 System.out.println("Database Product Name    :      "+ md.getDatabaseProductName());
                 System.out.println("Allowable active connections: "+ md.getMaxConnections());
               }
   }
   catch (Exception e) { e.printStackTrace(); }

 }
 // ---------------------------------------
}
```

REFERENCES

[1] Abdul-Rahman, A., Heiles, S. (2000). Supporting trust in Virtual Communities, in Proc. of International Conference On Systems Sciences, Hawaii.

[2] Sripanidkulchai, K., Maggs, B., Zhang, H. (2003). Efficient Content Location Using Interest Based Locality in Peer-to-Peer Systems, in Proc Infocom 2003

[3] Carvin, A. (2004). Berners-Lee: Weaving a Semantic Web, Retrieved February 1st, 2005, from the Digital Divide Network, Web site: http://www.digitaldivide.net/articles/view.php?ArticleID=20

[4] E.Damiani, E., DeCapitani, S., Paraboschi, S., Samarati, P. (2002). A Reputation-Based approach for finding Reliable Resources in P2P Networks, CCS'02, Nov 18-22, Washinghton DC, USA.

[5] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. Riedl, J. (1994). Grouplens: An Open Architecture for Collaborative filtering on Netnews, In Proc. of ACM Conf. on Computer Supported Cooperative Work, New York, pp175-186.

[6] McKnight, D.H., Chervany, N.L. (1996), The meanings of Trust, Technical report MISRC Working Paper Series 96-04, University of Minnesota, Management Information Systems research Centre.

[7] Christianson, B., Harbinson, W.S. (1996). Why Isn't Trust Transitive, In Proc of the Security Protocols Workshop, pp171-176.

[8] Josang, A. (2001). A Logic for Uncertain Probabilities, International Journal of Uncertainty Fuzziness and Knowledge-based Systems, 9(3), pp279-311.

[9] Kobayashi, H. (1978). Modelling and Analysis: An Introduction to System Performance Evaluation Methodology, Addison-Wesley Publishing Company Inc. ISBN: 0201144573

[10] Zacharia, G., Moukas A., Maes, P.(1999). Collaborative Reputation Mechanisms in Electronic Marketplaces, In Proc. 32th Hawaii International Conference on Systems Sciences, 7pp.

[11] Resnick, P., Varian, R. (1997). Recommender Systems, Communications of ACM, 40(3), pp56-58.

[12] Goldberg, D., Nichols, D. Oki-D.Terry, B.M. (1992), Using collaborative filtering to weave an information tapestry, Communications of the ACM, 35(12) pp61-70.

[13] Shardanand, U., Maes, P. (1995). Social Information Filtering: Algorithms for automating Word of Mouth, in Proc CHI'95, ACM Conference on Human Factors in Computing Systems, pp210-217.

[14] Sarwar, B., Karypis, G., Konstan, J., Riedl, J. (2000). Analysis of Recommendation Algorithms for E-Commerce, In Proc of the Second ACM Conference on Electronic Commerce pp158-167, ACM Press.

[15] Jensen, F.V. (2001). Bayesian Networks and Decision Graphs, Springer, ISBN: 0387952594

[16] Epinions Web site, http://www.epinions.com

[17] Amazon Website, http://www.amazon.com

[18] Ebay Web site, http://www.ebay.com

[19] Breese, J.S., Heckerman, D. Kadie,C. (1998). "Empirical Analysis of Predictive Algorithms for Collaborative Filtering". In Proc. of the 14th Conference on Uncertainty in Artificial Intelligence, pp 43-52.

[20] Marsh, S. (1994). Formalizing Trust as Computational Concept, PhD Thesis, University of Stirling, Scotland.

[21] Yahalom, R., Klein, B. Beth, T. (1995). Trust relationships in Secure systems. A Distributed authentication perspective, In Proc. of the 1993 IEEE Symposium on Research in Security and Privacy, Denver, Colorado, USA, pp202-209.

[22] Jøsang, A., Gray, E., Kinateder, M. (2003). Analyzing topologies of Transitive Trust, In Proc. of the Workshop of Formal Aspects of Security and Trust, FAST, Pisa.

[23] Shafer, G. (1976). A Mathematical Theory of Evidence, Princeton University Press, ISBN: 069110042X.

[24] Deutsch, M. (1962). Cooperation and Trust: Some theoretical notes, in M.R. Jones (ed.) Nebraska Symposium on Motivation, Nebraska University Press.

[25] Jøsang, A. (1999). An Algebra for Assessing Trust in Certification Chains, In Proc. of NDSS'99, Network and Distributed Systems Security Symposium, The Internet Society, San Diego.

[26] Josang, A., Ismail, R. (2002). The beta Reputation System, In Proc of the 15th Conference on electronic Commerce, Bled, Slovenia.

[27] Fair Isaak Co, http://fairisaac.com

[28] Herlocker, J., Konstan, J., Borchers, A., Riedl, J. (1999). An algorithmic framework for performing Collaborative filtering, In Proc of the Conference on Research and development in Information Retrieval.

[29] Ding, L., Zhou, L., Finin, T. (2003). Trust based Knowledge Outsourcing for Semantic Web Agents, In Proc. Of the IEEE/WIC International Conference on the Web.

[30] Sarwar, B., Karypis, G., Konstan, J., Riedl, J. (2000). Application of Dimensionality Reduction in Recommender Systems-A case study, In Proc Workshop at the ACM-SIGKDD Conference on Knowledge Discovery in Databases.

[31] Sarwar, B., Karypis, G., Konstan, J., Riedl, J. (2001). Item based Collaborative filtering Recommendation Algorithms, In Proc 10th international conference on World Wide Web, Hong Kong, pp285–295.

[32] Breese, J.S., Heckerman, D., Kadie, C. (1998). Empirical Analysis of Predicted algorithms for collaborative filtering, In Proc of the 14th conference on Uncertainty in Artificial Intelligence, Madison,WI, USA, pp.43-52.

[33] Aggarwal, C.C, Wolf, J.L., Wu K. and Yu, P.S. (1999). Horting Hatches an Egg: A New Graph-theoretic Approach to Collaborative Filtering. In Proc. of the ACM Conference on Knowledge Discovery in Data, San Diego, CA, USA, pp.201-212.

[34] Jøsang, A., Pope, S. (2005). Semantic Constraints for Transitive Trust, Second Asia-Pacific Conference on Conceptual Modelling, Newcastle, Australia, pp59-68.

[35] Massa, P., Avesani, P. (2004). Trust-Aware Collaborative filtering for Recommender Systems, in Proc. of International Conference on Cooperative Information Systems (CoopIS), Cyprus, pp492-508.

[36] Aberer, K., Despotovic, Z. (2001). Managing Trust in a Peer-to-Peer Information System, In Proc. of the 10th ACM International Conference on Information and Knowledge management, Atlanta, Georgia, pp310-317.

[37] Official web site of UDDI: http://www.uddi.org

[38] B.Yu-M.P.Singh, "A Social Mechanism of Reputation Management in Electronic Communities", In Proc of the 4th International Workshop on Cooperative Information Agents, Lecture notes in Computer science Vol 1860, Springer 2000.

[39] McKnight, D., Cummings, L., Chervany, N., (1995). Trust formation in New Organizational Relationships, In Proc of Information and Decision Sciences Workshop, University of Minnesota, pp23.

[40] Aberer, K. (2001). P-Grid: A self-organizing access structure for P2P information systems, In Proc. of the 9th International Conference, CoopIS, Trento, Italy, pp179-194.

[41] Jøsang, A., Gray, E., Kinateder, M. (2006). Simplification and Analysis of Transitive Trust Networks, Web Intelligence and Agent Systems Journal, 4(2), pp139-161.

[42] Wikipedia Web site: http://en.wikipedia.org/

[43] Grewal, R., Cline, T., Davies, A. (2003). Early-Entrant Advantage. Word-of-Mouth Communications. Brand Similarity and the Consumer Decision-Making Process, Journal of Consumer Psychology 13(3),pp187-197.

[44] Mezzetti, N. (2004). A Socially Inspired Reputation Model, 1st European PKI Workshop, Samos Island, Greece, pp191-204.

[45] Petricek, V. (2004). Recommender systems. An Annotated bibliography, Technical report, University College London, Last modified February 2.

[46] Stakhanova, N., Ferrero, S., Wong-Y.Cai, J. (2004). "A Reputation-based Trust management in Peer-to-Peer Network systems" In Proceedings of 17th International Conference on Parallel and Distributed Computing Systems, PDCS, pp510-515.

[47] Lee, S., Sherwood, R., Bhattacharjee, B. (2003). Cooperative Peer Groups in NICE In Proc Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies, pp1272-1282.

[48] Gupta, M., Judge, P., Ammar, M. (2003). A Reputation system for Peer-to-Peer Networks, In Proc of International Workshop on Network and Operating System Support for Digital Audio and Video, pp144-152.

[49] Kamvar, S., Schlosser, M., Garcia.Molina, H. (2003). The Eigentrust algorithm for reputation management in P2P networks, In 12th International World Wide Web Conference, pp640–651.

[50] Xiong, L., Liu, L. (2002). Building Trust in decentralised peer-to-peer communities, in International Conference on Electronic Commerce Research, (ICECR-5).

[51] Micropayments Web site http://www.w3.org/ECommerce/Micropayments/

[52] Salton, G., McGill. M.(1983).Introduction to Modern information Retrieval, McGraw-Hill, New York. ISBN:0070544840.

[53] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (1996). Advances in knowledge discovery and Data Mining", AAAI/MIT press, ISBN 0-262-56097-6.

[54] Minar, N., Moukas, A. (1998). Friend of a Friend Finder, Software agents Group. MIT Media Lab, Working paper, Cambridge, MA,USA.

[55] Marti, S., Garcia-Molina, H. (2003). Identity crisis: Anonymity vs. Reputation in P2P Systems, In Proc. of Third IEEE International Conference on Peer-to-Peer Computing, Linkoping, Sweden, pp134.

[56] Dewan, P., Dasgupta, P. (2005). Securing P2P Networks using Peer Reputations: Is there a silver bullet?, IEEE Consumer Communications and Networking Conference CCNC, Las Vegas, Nevada, pp30-36.

[57] Delacoras, C., Resnick, P.(2003). On Line Reputation Mechanisms A Roadmap for future Research, Summary report of the First Interdisciplinary Symposium on Online Reputation Mechanisms, Cambridge, Massachusetts.

[58] Dewan, P.,Dasgupta, P. (2004). PRIDE:Peer-to-Peer Reputation Infrastructure for Decentralized Environments, The Thirteenth International WWW Conference, pp 480-481.

[59] Axelrod, R. (1984). The evolution of Cooperation, Basic Books, New York, ISBN:0465021212.

[60] Miller, B.N., Albert, I., Lam, S.,K., Konstan, Riedl, J., (2003), MovieLens Unplugged: Experiences with an Occasionally Connected Recommender System, In Proc. of ACM 2003 International Conference on Intelligent User Interfaces (IUI'03)(Accepted Poster), pp263 - 266.

[61] Clausen, A. (2004). The Cost of Attack of PageRank, International conference on Intelligent Agents Web Technology and Internet Commerce, Gold Coast, Australia. pp77-90.

[62] Buchegger, S., Boudec, J.L. (2004). A Robust Reputation System for P2P and Mobile Ad-hoc Networks, In Proc. of the Second Workshop on the Economics of Peer-to-Peer Systems, Harvard, Cambridge, USA.

[63] Keeter, K.H. (2000) Collaborative filtering: Community values, Technical report, *IBM,* http://www.ibm.com/services/innovation/etrcollaborative_filtering.pdf

[64] Farley, A., Proskurowski, A. (1980). Gossiping in grid graphs, Journal of Combinatorics, Information and System Science, 5(2), pp161-172.

[65] Morinaga, S., Yamanishi, K., Tateishi, K., and Fukushima, T. (2002). Mining Product Reputations on the Web, In Proc. of the Eighth ACM SIGKDD International Conference on Knowledge Discover and Data Mining, Alberta, Canada, pp341-349.

[66] Keast, G., Toms, E.G., Cherry, J. (2001). Measuring the Reputation of Web Sites: A Preliminary Exploration, In Proc. of the 1st ACM/IEEE-CS joint conference on Digital libraries, Roanoke, Virginia, USA, pp77-78.

[67] Liestman, A.L., Richards, D. (1993). Network Communication in Edge-Colored Graphs: Gossiping, IEEE Transactions on Parallel and Distributed Systems, 4(4), pp438-445.

[68] Kinatender, M., Rothermel, K. (2003). Architecture and Algorithms for a Distributed Reputation System, In Proc. of the First International Conference on Trust Management: iTrust'03; Heraklion, Greece, pp01-16.

[69] Chaum, D., (1981), Untraceable electronic email, return addresses and digital pseudonyms, Communications of the ACM 24(2), pp84-90.

[70] Huberman, B., Hogg, T. (2002). Protecting Privacy while revealing data, Natural Biotech, vol. 20, p332.

[71] Kant, K., Iyer, R. (2003). Modeling and simulation of Adhoc/P2P Resource Sharing Networks, Enterprise Architecture Lab, Intel Corporation, Technical Report, http://kkant.ccwebhost.com/papers/simpra.pdf

[72] Yang, B., Garcia-Molina, H. (2003) Designing a Super-Peer network, In Proc. of 19th International Conference on Data Engineering (ICDE), pp49-60.

[73] Miller R.B (1968). Response time in man computer conversational transactions, In Proc of AFIPS, Fall point Computer Conference, Vol. 33, pp267-277.

[74] Kant, K., Iyer, R. (2001). A performance model for Peer-to-Peer File sharing Service, Enterprise Architecture Lab, Intel Corporation, Technical Report.

[75] The Gnutella protocol specification, http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf

[76] Ungar L., Foster D. (1998). Clustering Methods for Collaborative filtering, In Proc of Workshop on Recommendation Systems at the 15th National Conference on Artificial Intelligence, Madison, Wisconsin, USA.

[77] Drake, A. W. (1967). Fundamentals of Applied Probability theory, McGraw-Hill, ISBN: 0070178151.

[78] Orwant, J. (1995). Heterogeneous learning in the doppelganger User-modding system, User Modeling and User-Adapted Interaction, 4(2), pp107-130.

[79] Golub, G.H., Van-Loan,C.F. (1996). Matrix Computations, 3rd edition, Johns Hopkins University Press, Baltimore and London, ISBN:0801854148.

[80] Zheng, Y., Akhtar, S. (2001). Networks for Computer Scientists and Engineers, Oxford University Press, ISBN: 0195113985.

[81] Linden, G., Smith, B., York, J. (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering, IEEE Internet Computing 7(1): pp76-80.