Collaborative Ad-hoc Applications

Paul O Connell BA BAI

A dissertation submitted to the University of Dublin,
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science

September 2000

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____

Paul O Connell

September 2000

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____

Paul O Connell

September 2000

Abstract

The proliferation of mobile hand-held devises and the development of ad-hoc networking technologies has opened the possibility for users of these devises to partake in collaborative applications with other mobile users. The sharing of resources and information within these groups will maximise the experience for all group members. This will require a user of the system to grant access to local resources to other users, some of whom may be semi or completely unknown. The major issue that affects the user of the collaborative ad-hoc application is how can they grant access to a user who is completely unknown to them.

People are able interact with others who are semi or completely unknown to them in many every day situations, the extent to which they interact is governed by the concept of trust. Trust is a complicated model that allows people to balance the risk of the unknown against the perceived benefits of interaction. The aim of this dissertation is to investigate if a model for trust production can be integrated into a computer access control mechanism to manage interaction with anonymous users.

A mechanism will be developed that will produce a trust value for any user based on the details of previous interactions between the two or recommendations. An dynamic trust-based access control system, which binds minimum trust levels to access right will then be used to verify if an unknown user is trusted enough to gain access to the users resources. A blackjack card game that uses a trust-based access control system to assign roles to users was implemented in java, JINI technology was used to distribute the application.

Acknowledgements

I would like to thank my supervisor Dr Christian Jensen for his advice and help during this year. I would also like to express special thanks to all the members of the MSc class for their help and support during the year.

Table of Contents

Table of Figures

1 Introduction

1.1 Introduction

The information revolution has redefined the way we communicate, conduct business and disseminate knowledge. The proliferation of mobile devises and networking technologies over this period has made it easier for users of these devises to interact and share information. The technology now exists that will allow users of these mobile devices, to form into collaborative ad-hoc groups [Jensen]. Group working allows users to share knowledge, pool resources and achieve results quicker than by individual processing. The synergy achieved by group applications is possible because users of the group are prepared to share and contribute resources and information.

In traditional groupware applications access to group is control by a predefined list of members managed by an administrator. The members of the group are prepared to share their resources because they trust that the administrator of the group has already established a level of trustworthiness in the users in places in the groups access list. The key requirement for the success of the groupware application is that all the users “trust” the administrator to verify members for access. This access control system is based on the assumption that a member trusts the administrator to verify that all existing group members will behave correctly when granted access to the member’s local resources.

This dissertation rejects this basic assumption because the transfer of control to the administrator means the user now has no control over who accesses his own resources. In a modern computer system this complete lack of control over one’s own resources is unacceptable. A mechanism that will allows a user to maintain access over their own resources and still interact in group applications is the focus of this dissertation.

1.2 Proposed Solution

People use trust relationships to determine how to interact with known and unknown user every day. It enables to cooperation between people, provides a mechanism for lowering access barriers and enables complex transactions between groups of users. Without trust there would be a loss of efficiency and dynamism. Marsh quotes Golembiewski and McConkie in his thesis “Perhaps there is no single variable which so thoroughly influences interpersonal and group behaviour as does trust ” [Marsh].

Collaborative ad-hoc applications extend on groupware applications, but do not require an administrator to manage who can access the group. This new application type allows members to join groups in an ad-hoc dynamic manner. The lack of any administrator means the members of the group must manage access to the group by themselves. The members cannot rely on any third party to verify the trustworthiness of a user. The user of the collaborative ad-hoc application needs to take control of the trust formation process. The users must be provided with a mechanism that will allow them to establish a trust value for users. In order to establish a strategy for interaction with a user who’s intent is unknown, the user needs to able to answer the question “Why do I trust this user to perform this action? ”.

1.3 Aims

The aim of this dissertation is to develop, design and implement a trust-based access control system for use within a collaborative ad-hoc application environment. The dissertation will first focus on the development of a generic framework that will allow users to form, join and exit collaborative ad-hoc groups. The second step will focus on the design and implementation of a new security mechanism for use in the collaborative ad-hoc application environment. This will involve the creation of a dynamic access control system and a system that can calculate the trustworthiness of an individual. When

these two steps have been completed, a sample application will be implemented to test the behaviour of the trust model developed.

1.4 Roadmap

Chapter 2 is a literature survey it examines the existing mechanism for computer access control and the concept of trust. Chapter 3 outlines the specification and design of the system framework. Chapter 4 outlines the implementation route and the technologies used in the system. Chapter 5 presents the evaluation of the framework system and discusses the results from the sample application. Chapter 6 gives an evaluation of the thesis and contains concluding remarks on the work carried out for this dissertation. It also discusses the direction of future work in this area. A bibliography and appendix then presented.

1.5 Summary

This chapter has introduced the traditional mechanism for access control used in group applications and identified the fundamental problem that users must rely on the system administrator to verify the trustworthiness of the users they interact with. The chapter identified the not being able to directly establish trust in a user is a major problem in modern open networks. An alternative solution has been proposed that will allow the user to establish direct trust in another user. The solution calls for the development and implementation of a trust-based access control system.

2 Literature Review

2.1 Introduction

This chapter will introduce the various issues and concepts that are relevant to the design and implementation of the dynamic trust-based access control system. The aim is to identify the shortcomings of existing systems used for access control, and justify why a new system should be developed. The chapter will then examine the concept of trust and examine a framework for the production of trust.

2.1.1 Overview

This literature review is organized as follows. The first section starts by over viewing the operation of authentication and access control mechanisms used in traditional security. The third section discusses the trust management concept, which has been proposed as a solution for access control in large distributed systems, the PolicyMaker and Keynote trust management systems will be introduced. The fourth section introduces the concept of trust and examines the properties and operations that can be applied to it, a framework for trust production is also discussed. The fifth section discusses the use reputation and recommendations. The final section gives an overview of the topics discussed in this review and summaries the key design issues and points that should be apply to the design and implementation of the collaborative ad-hoc application framework and trust-based access control system.

2.2 Traditional Security Approaches

All security systems must be able to authorise a user to perform certain actions. This involves the basic ability to correctly authenticate a user in the

system and then assign them some level of access control to resources within that system. The question that the security system must answer is " Does a user U have the rights to do action A on resource R? ". A high-level view of the authorisation process can be seen below.

1. The user presents an identifier to the authentication system to be verified.
2. The access control process then checks to see if the user has access rights to the requested resource.
3. The access control process determines if the users pre-approved actions allow the user to execute the current requested action.
4. Access to the resource is granted or denied.

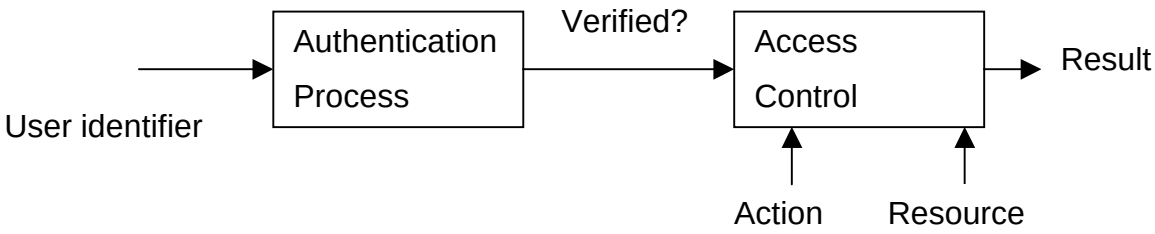

Figure 1: Authorisation Process Overview

This is the basic four-step process that all security systems use. The rest of this section will focus on how the authentication and access control processes have been developed and adapted to handle large-scale distributed systems.

2.2.1 Access Control

Access control systems enforce the policy that the manager of a resource applies to parties that requests access to that resource. The policy is relevant to a specific resource and normally refers to who can have access to the resource and what type of access they can have. The flexibility of the access control mechanism in terms of how it defines who can access the resource and secondarily what actions they can then take is the key to its success.

Two different access control systems are presented in the following sections. The scalability and flexibility of the Access Control List (ACL) system and the Role Based Access Control (RBAC) system will be examined [Herzberg].

2.2.1.1 Access Control Lists

The most traditional mechanism for access control is the Access Control List. This system binds a user's identifier directly to the action that the user is approved to perform. The administrator by inserting the binding indicates that he trusts the user to behave correctly in performing the actions.

There are a number of issues associated with the operation of access control lists. The fundamental problem relates to the flexibility of the system. Every time a new user has to be added, or a privilege granted or revoked the administrator has to directly modify the list. The ACL system does not scale well. As the number of user accessing the resource grows the administrator has a greater problem managing the file. When the administrator inserts or modifies a binding into the ACL, a statement is being made about the trustworthiness of the user. The level of trust in the user may change over time, but unless the ACL binding is changed it represent a static trust statement. When a binding is inserted into the ACL, the administrator does not state the reasons why the user is trusted to correctly perform the actions granted.

Access control lists have been used in large-scale distributed systems because the concept is familiar but there are a number of fundamental reasons why they are unsuitable to this type of application.

The access control list framework relies on the absolute authentication the requester. In a distributed system authentication of the user is possible but not to the same extent as in an operating system.

Access control lists do not scale well. If applied in the ad-hoc application environment where there is potential to interact with a large number of users, maintaining an individual mapping of identities to access rights is impractical.

2.2.1.2 Role Based Access Control

In the role based access control framework users are granted access to roles based on the administrator's perception of their experience, competence, and the trustworthiness [Kuhn]. Each role in the framework has an associated set of access rights.

The system still directly maps names to roles and has the same limitation of the ACL system in this respect. The major improvement in flexibility for the administrator comes in the management of access rights of roles. When the administrator adds or removes an action right from a role the effect is immediate. He does not have to modify the privileges of every user on an individual basis as in the ACL system. In situation where there are a large number of users all having the same access rights, the role based system is highly efficient when modifications have to be made.

Another benefit of associating roles to access right is that roles tend to have overlapping responsibilities and privileges. This allows roles to inherit the rights of lower roles and create a role hierarchy. This removes the need to specify common actions repeatedly across different roles.

The main advantage of the role based framework is that once the role hierarchy has been developed and the correct actions have been associated with the role, the main administrative action is reduced to adding and revoking users between roles. The time that is needed to develop the initial role hierarchy and access rights might be longer than that of the ACL system.

2.2.2 Authentication Processes

The correct identification of the requesting user is a corner stone on which most authorisation systems operate. In a small-scale system it is possible for an administrator to assign a unique identifiers to each user of the system, but this solution does not scale well. As the number users in a system increase the administrator has to assign more identifiers and commits more resources to the management of the authentication system.

Public key cryptography is used in large-scale systems to create a unique key for every user. The system guarantees authentication of a message signed using the system. We will examine two public key encryption based systems that are used for authentication in distributed systems.

2.2.2.1 Distributed Key Management

The scalability problem of having administrators managing their own list of keys lead to the development of a global key management system. The fundamental change that allowed this system to develop was that system administrators accepted a trusted third parties statement that a key belonged to a user. The trust third parties, known as Certificate Authorities (CA) create X.509 certificates, which bind a users details to a public key and includes a reference to the CA. The X.509 key system works on a global scale because it is replicated. A Certification Authority exists in each country. A Master Certification Authority connects each Certification Authority at a higher level. A X.509 certificate works across country borders because a *trust chain* can be created via the Master CA.

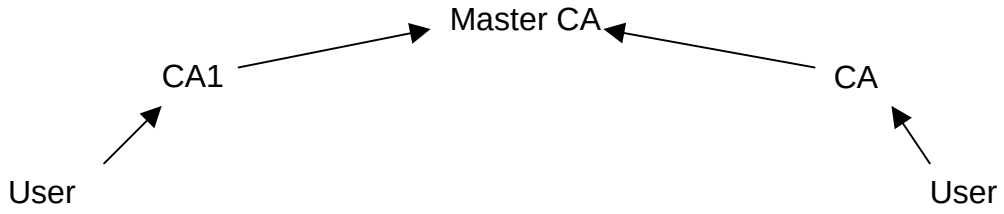

Figure 2: X.509 Certification Authority Hierarchy

The trust chain allows a user to authenticate a X.509 certificate because via the trust chain a path can be established to some Certification Authority that has already established a valid binding between the user and there public key. The major fault with the X.509 system is the amount of user detail included in the certificate at each CA is not uniform across the whole system. A low-level CA can bind very detailed information about a user in the X.509 certificate that they give a user, but at a higher-level CA will not capture as detailed information. The Certification Authority is used to form a trust relationship, but it can never be a good enough authority for every one in a large distributed system, its credibility depletes and its recommendations increase in uncertainty as the community grows. The major issue in this system for users is credibility, as a user climbs up the hierarchy pyramid in order to find a common Certification Authority, there is greater reach and less specificity [Khare]. It also assume that every one trusts the certification authority.

2.2.2.2 Pretty Good Privacy

Pretty Good Privacy system allows user to create a decentralised trust hierarchy by locally binding user identities to public keys. This approach is known as creating a “web of trust” [Abdul3]. The system works on the basis that as a user interacts with other in the distributed system, they will sign the certificates of individuals they believe to be authentic. This creates a trust relationship between these individuals. The system allows the user to designate trusted users to act as introducers, and define a level of trust that they associate with each introducer. In the case where a user interacts with an individual who is unknown to them but who’s certificate contains the signature of a number of known introducers, the user can be more confident about the authenticity of the new users public key, via the recommended trust relationship that was developed. The change from the X.509 system is that the perceived trustworthiness of an individual is based on local decision to trust. The problem of trust production is not addressed in this system. The PGP system cannot answer the question why a person is trusted.

2.2.3 Summary

In this section we have examined the operation of some authentication and access control mechanisms. We have seen that the authorisation process assumes there is some level of trust associated with a user once this process is complete. The problem is that on closer inspection of the two processes involved in authorisation, neither takes responsibility for stating why a user is trusted.

The access control process assumes incorrectly, that if a users identity is verified correctly, then that user is trusted. This is an overuse of the functionality provided by public key cryptography. The public key cryptography system will correctly authenticate a user from a public key, but it does not provide any information that can be used to derive a level of trustworthiness in the user.

The second issue is the inflexibility of existing access control systems. The existing access control systems rely completely on the administrator to configure all new user bindings. This approach is too restrictive in a situation where new bindings will be need regularly.

2.3 Trust Management Systems

The Trust Management approach to distributed system security was developed because of the perceived inadequacy of traditional authorisation mechanisms for distributed systems [Blaze2]. Traditionally every application implemented its own mechanisms for specifying access policy, checking compliance and binding user authentication to authorisation to perform security critical operations. The developers of the trust management system felt that this approach exposed the security of the application to high level of risk because the application developer might make simple but subtle mistakes the design and implementation of the system.

The main aim in the development of the trust management system was to produce an off-the-shelf security module that could be integrated into any application. The module allows each application to define application specific policies and credentials.

The notion of a “resource” for which the security system is designed to protect will vary widely across different distributed system. The developers recognised that the system should be able to define what a resource is within their own environment. The next step is to allow the system define access and restriction policies that apply to its resources. Policies and credentials can be modified to reflect changes in the use patterns of the system over time.

The solution that the Trust Management system proposes is to bind a set of specific keys directly to authorisation to perform a specific task, this mapping is known as a credential. The fundamental question that the trust management system now asks is “Does the set C of credentials prove that the request R complies with the local security policy P? ”. The request R, the local policy P and set of credentials C are passed to a trust management engine, which processes the request and outputs a decision whether the request and credentials provided is valid according to the local policy.

The trust management approach advocates the use of a general purpose, application independent algorithm for checking proof of compliance. Applications using the single general-purpose compliance checker in a distributed system can be confident that results from the checker are correct. This reduces the need to have each resource in a distributed system implement its own compliance checker.

2.3.1 PolicyMaker

PolicyMaker was the first tool for processing signed requests that embodied “trust management” principles, developed by Blaze. The PolicyMaker takes as input a set of local policy statements, a collection of credentials, and a string

describing a proposed trusted action. It can return a binary yes/no result or provide additional restrictions that would allow the proposed action possible.

Security policy and credentials are defines in terms of predicates, called filters that are associated with public keys. The PolicyMaker tool processes queries from applications, to determine if the public key is permitted to perform an action according to local policy. PolicyMaker recognises that complete security information may not be available locally, so allows trust to be deferred to trusted third parties which provide additional security information. The actions that the PolicyMaker tool is asked to consider are application specific.

Responsibility for authorisation is split between the PolicyMaker and the calling application, authentication by cryptographic verification of the requester is placed with the calling application. This means that the choice of signature scheme is independent of the choice to use PolicyMaker as the compliance checker. The application is also responsible for ensuring that all the credentials needed to ensure correct compliance of a request are provided in the original query to PolicyMaker [Blaze1, Blaze2].

2.3.2 KeyNote

The KeyNote tool can be considered a successor to PolicyMaker as the basic design principles were continued but extra design goals were specified, which included easier integration for applications and standardisation. KeyNote assigns more responsibility to the trust management engine, making it easier to integrate into applications. It also requires that assertions (policies and credentials) be written in a specific language, so that processing by the compliance checker proceeds more smoothly.

The application passes an “Action Environment” to the KeyNote tool; this is similar to the PolicyMaker “query” in that in contains a triplet of values, the security policy, a list of credentials and the request. The result of passing the action environment to the KeyNote tool is an application-defined string. In the simplest case, the result will just be “authorised”. KeyNote like other trust

management engines, does not enforce policy directly it only provides advice to applications that call it [Blaze1].

2.3.3 Summary

The Trust Management System attempts to improve on the traditional security system because it rejects the use of name-key binding and pre-complied access-control matrices. The trust management system can be considered more secure because of its ability to integrate into other system means that the development of the security system can be left to experts who can guarantee that the system achieves “proof of compliance”. The risk of simple security holes being left by applications developers is reduced.

The trust management system has addressed the problems of distributed application security, difficult to authenticate users and resource definition. But it still cannot answer the fundamental question “Why is the user trusted to execute this action?”.

2.4 Trust

Trust is an important feature in the decision-making process that humans use every day. The main issue with trust is that there are many different views of it, and this has lead to problems when trying to develop a general definition for trust. There are main reasons for the multiple views of trust but two reasons stand out. First, everybody is an *expert* on trust, that is to say that everybody has their own brand or idea of what trust is, and how to use it. The second is more straightforward, and simply states that there are many different types of trust, reflecting different situations and contexts.

We have seen in the two preceding sections how existing security systems cannot explain why a user is trusted to perform a certain action. The aim of this section is to show that trust can be formalised and produced within a framework. The goal of defining trust in this way is that if a security system is

asked, “Why do you trust this person? ” it will be able to answer stating the parameters and contexts on which it has built its trust assumption about that person.

The next section will present a number of trust definitions used in different fields in an attempt to create a large picture of trust as a concept. The properties and operations that apply to trust will then be presented. A framework for modelling trust will be explained and the discussion on how to quantify trust will then happen.

2.4.1 Introduction

Researchers in a wide range of academic fields have studied a concept of trust. The definitions of trust presented hereafter illustrate the different view of trust. The aim of presenting a number of different definitions of and statements about trust is that each explains a certain aspect of trust, examining them as a group it is hoped will provide us with a more general understanding of trust.

The first definition comes from Morton Deutsch (1962).

1. an individual is presented with an ambiguous path, a path that can lead to an event perceived to be beneficial (Va+) or an event perceived to be harmful (Va-);
2. he perceives that the occurrence of Va+ and Va- is contingent on the behaviour of another person; and
3. he perceives the strength of Va- to be greater than the strength of Va+.

The choice the individual makes in this situation reflects his level of trust in the other person; the decision to take the ambiguous path reflects the distrustful choice. The repeated uses of the word perceive in the definition implies that trust is a subjective quality an individual places in another. The second point about this definition is that trusting decisions are based on a form of

cost/benefit analysis. Different individuals decision to trust will differ with each individuals perception of the estimated cost (Va-) and benefit (Va+) [Marsh].

A second definition is taken from Diego Gambetta (1990).

”Trust is a particular level of the subjective probability with which an agent will perform a particular action, both before [we] can monitor such action and in a context in which it affects [our] own action”.

This definition again re-enforces the subjective nature of trust. The definition also states that trust is affected by actions that we cannot monitor. This means that in a situation where full information is not available, the quality and meaning of the information we can monitor will have a major effect on our level of trust.

Another aspect of trust relates it to the amount of information and risk that situation creates. Reagle states “Trust itself represents an evaluation of information, an analysis that requires decisions about the value of specific information in terms of several factors” [Reagle]. In a situation where full information is not available the likely hood of making an incorrect choice is increased, this leads to risk. Trust has been proposed as a mechanism for the reduction of complexity (risk) in situations. Trust is used to derive further information from the surroundings.

2.4.2 Properties and Operations

The subjective nature of trust is evident in most of the definitions presented in the last section. This property creates the single biggest problem to creating a trust production model for computer models because to parameters used for calculating trust by different people varies widely.

Trust is a non-transitive property. The following statement gives a good example of this “If Ann trusts Bob, and Bob trusts Cathy, therefore Ann trusts

Cathy" is in general, not true. The statement can only be considered true if the following conditions hold.

1. Bob explicitly tells Ann that he trusts Cathy. This is an example of a recommendation.
2. Ann must trust Bob as a recommender. If Ann does not trust Bob as a recommender then she should ignore any information he passes her.
3. Ann must be allowed make a judgement on the quality of Bob's recommendation.

A trust statement is a context specific. On order to give a recommendation on the trustworthiness for someone the context in which the recommended trust value will be used is need. The following statement is an example of this "I trust my brother to drive a car, but not to fly a plane".

2.4.3 A Framework for Trust

The definitions of trust that have been presented above have failed to converge and provide us with a single generic definition. Although the concept and operations of trust are well understood a framework for quantifying trust is need so that unambiguous conversation can occur. McKnight and Chervany in their working paper "The meaning of trust" [McKnight] presents a framework that classifies different aspects of trust and provides a system which shows how trust can influence behaviour.

System Trust

The extent to which, one believes that the proper impersonal structures are in place to enable one to anticipate a successful future endeavour. System trust reflects that safeguards are in place to reduce the amount of risk that user has to expose themselves too. These safeguards may be in the form of regulations, guarantees or stabilising intermediaries.

Dispositional Trust

The extent to which, a person consistently trusts across a broad spectrum of situations and parties. This is an example of a cross-situational cross-personal construct. It is a general intention by a person to believe that people will behave in a certain manner and is valid over a broad spectrum of situations. This value reflects whether an individual is optimistic or pessimistic in their approach to new situations [Swarup].

Situational Trust

The extent to which, one intends to depend on another in a non-specific party in a given situation. It means that one has formed an intention to trust every time a particular situation arises, irrespective of one's beliefs about the attributes of the other party in a situation. It

Trusting Belief

The extent to which, one party believes that another party is willing and able to act in the trusting party's best interest. The attributes that affect the trusting belief in another include benevolence, honesty, competence and predictability. Assessing these attributes to form a belief is inherently subjective process.

Belief Formation Process

Is the process by which information and experience gathered from the environment is processed to from new trusting beliefs about parties.

Trusting Intention

The extent to which one party is willing to depend on another party in a given situation with the feeling of relative security even though negative consequences are possible. A trusting intention implies the entity has made a decision about the various risk and benefits of allow this trust

Trusting Behaviour

The extent to which one party depends on another party in a situation with a felling of relative security, even though negative consequences are

possible. This construct effectively describes the “act” of trusting and implies the acceptance of risk [Povey].

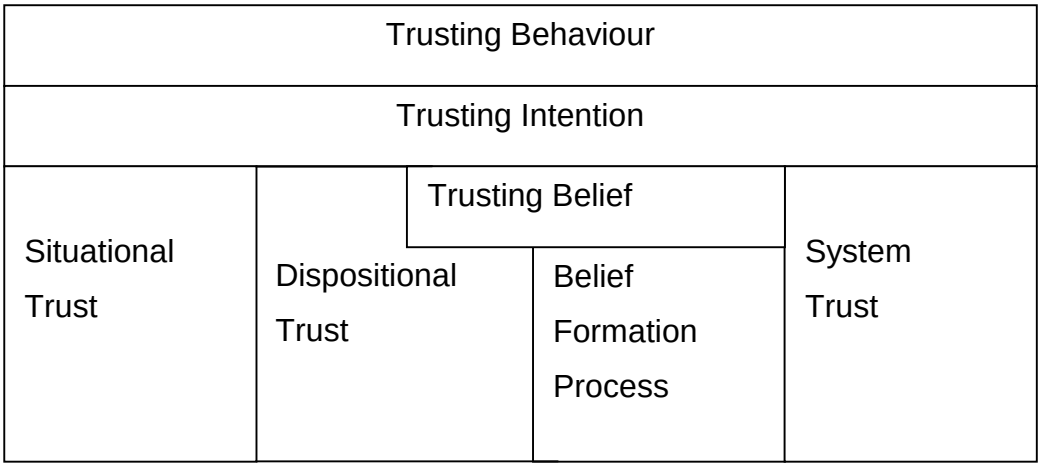

Figure 3: Related Trust Categories

2.4.4 Trust Values

Trust has no measurable units, but its value can be measured. Trust can therefore considered as a commodity like information or knowledge. There are problems using explicit values to represent trust because since trust is subjective the same value of trust may be associated with different levels of trust by different entities.

The use of continues values to model trust reflect the continuous nature of trust, allows for easy implementation and allows for comparison of different trust frameworks. The continuous value still leaves the problem of subjectivity for receiver of the trust information, if the framework and parameters on which the trust value is calculated is not understood.

Marsh in this thesis represents trust as a continuous variable over a range scale [-1,+1]. Marsh states that trust can have threshold values, which vary between people and situations. A person will have a positive threshold value,

above which that trust another, and a negative threshold value, below which they will not trust a person.

2.4.5 Summary

The creation of six trust definitions has cleared the conceptual confusion by representing trust as a broad but coherent set of constructs. The definition of these trust types is very beneficial, it will enable parties that apply and exchange trust information to agree on what they mean, when they process trust values. Using the framework to identify the parameters on which trust is based allows the users to be specific about out trust statements. This avoids the problems associate with trust being subjective, non-transitive and context specific.

2.5 Reputation

The last section explained that a framework for trust production can produce a trust value based on the information about the users surrounding and situation. In a situation where it is not possible to gather enough information to produce a trust value about a user, it is possible to seek out the users reputation from a trust third party. This section will examine the use of reputation and recommendations.

2.5.1 Recommendations

Alfarez Adbul-Rahman et al [Abdul 1, Abdul2] discusses a decentralised trust management system based on the use of recommendations between agents in a distributed system. The concept of “social control” is used to implement a soft security approach in an open system. It is up to the good actors to identify ‘cheaters’ and propagate this information around the system.

The system used the concepts of direct and recommended trust relationships existing between agents. The authors recognise that in order to make a trust

decision specific information relating to aspects of the situation must be available. They introduce the concept of a “trust category” to represent the aspect or context in which trust is evaluated. “Trust values” are then used within each category to further distinguish different levels or amounts of trust. The system captures an agents reputation as concatenation of the name, trust category and value.

Reputation = [Name, Trust Category, Trust Value]

A person wishing to establish another person’s reputation sends a Recommendation Request Message to a trusted recommender. They receive a Recommendation Message back that contains a reputation.

The work in this paper is beneficial because it clarifies the protocol that is need between parties trying to exchange reputation information. It also highlights the processing that the receiver of a recommendation must do in order to extract the correct information form it. The work carried out in these papers explains how recommendation can be used and manipulated to increase the level of trust information in a distributed group but it does not attempt to explain how trust is initially produced by any of parties in the group.

2.7 Conclusion

This aim of this review was to highlight issues that would be relevant to the design and development of a security system for the collaborative ad-hoc application environment. The areas that proved the most beneficial to the aim of this dissertation will now be refreshed. The research on the access control mechanisms proved to be very insightful. The access control mechanism implemented in the collaborative ad-hoc application will have to allow dynamic access control statements. The research also shows that a framework for the production of trust does exist and can work. Given the research carried out in this chapter, the design of a system that incorporates these ideas can proceed.

3 Design

The aim of this dissertation is to develop and implement a dynamic trust-based access control system for the collaborative ad-hoc application environment. The aim can be split into two main task, the first is the design of a trust-based access control system for the collaborative ad-hoc application environment and the second is the design of the application framework.

The chapter will begin by giving a general overview of the collaborative ad-hoc application environment. This will identify the major processes. The requirements of each process will then be discussed. Finally the design of the components that provide the required functionality will be presented.

3.0.1 Terminology

A number of terms will be used extensively throughout this chapter to refer to the various entities. This section will explain what is meant by each of these terms. The user is the person who uses the collaborative ad-hoc system. The client refers to the computer software that is installed on the users mobile devise. The application means the software system that has collaborative ad-hoc functionality. The group refers to the members of the collaborative group.

3.1 General Overview

The users of the collaborative ad-hoc application framework interact in groups in an environment where an ad-hoc networking system is in operation. Each user has a small computer device that is capable of joining the ad-hoc network. It is assumed that each device has a screen, a hard disk to save information on and a networking capability.

When the user is connected to the network the names and details of any ad-hoc collaborative applications that are available on that network are displayed to the client on their device. The user then selects a specific application and communicates directly with that application. This service discovery and download is shown as step 1 in figure 4. The arrows show which entities are involved at each step.

Having discovered what roles are available for users within the application the user attempts to join the group by making a request to the application (step 2). The application will contact all existing group members and asks them if they approve the new users request to join (step 3). The individual members process the request based on the trustworthiness of the user (step 4) and return their result to the manager. The requesting user is granted or denied membership of the group based on the result of the vote. The approved user then joins the group and takes part in the collaborative application (step 5).

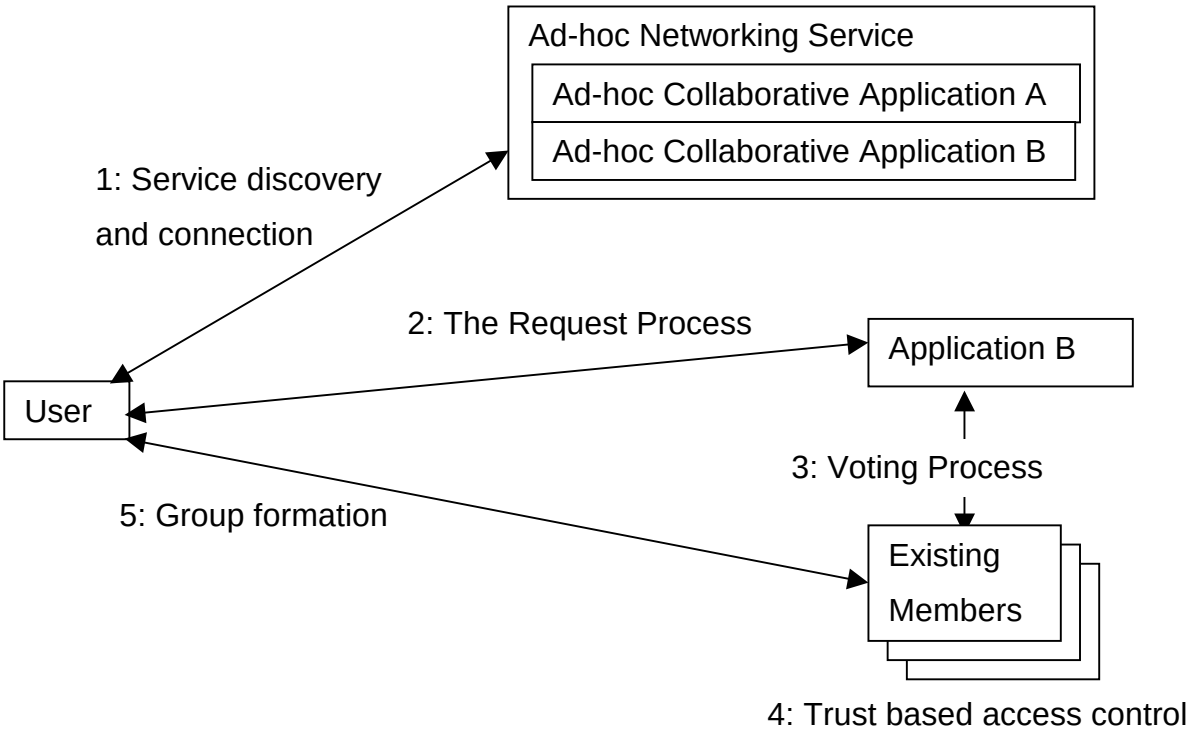

Figure 4: High-level overview of the main processes

3.2 Requirements

The last section introduced the five major process involved in the collaborative ad-hoc application. This section examines each of these processes in more detail; the focus is to identify the requirements of each process.

3.2.1 Service Discovery and Connection Process

This process allows the client to get a list of all collaborative ad-hoc applications. This information is displayed to the user who selects which application they wish to join. The client will download a proxy that allows it to communicate with the application and initiate the join process. The technical issues associated with application discovery and proxy download are handled by the JINI Discovery and Join protocols [Sun].

3.2.2 The Request Process

The client will use this process to request the application for membership of a group. The process allows the client to make a request to the application to join the group in a particular role. The application will confirm or reject the users request to join the group in that role. If the user is successful in their request to join the group they will then register officially with the application an be included in future voting processes. The user will typically be interested in joining the group in the role with most access rights, to facilitate this the client is allowed to make multiple requests to join the group. When the application receives a request from a client to join the group in a particular role it will initiate the voting process.

3.2.3 Voting Process

The application initiates the voting process when a client requests to join the group. The manager will ask each member of the existing group to verify that the requesting user has the right to join the group in the requested role. There

are a number of different voting policies that the application can implement. Unanimous voting and majority voting are two examples of different voting policies. A unanimous voting policy requires every group member to agree on the decision. A majority policy only requires that more than half of the group to agree. At the end of a vote the requesting client is informed of the decision and so is the group. This allows a user to leave the group if they do not agree with the result. This happens when the user rejects a request to join but the overall group decides to approve the request. This provides a back-out mechanism.

3.2.4 Trust-based Access Control Process

This process is responsible for using trust to making the decision whether to accept or reject a request to join the group. The operation of this process is the main focus of attention for this dissertation. The process is executed every time the application looks for verification of a request to join the group. The client is passed the requesting users identifier and role the client must accept or reject the request. The process will calculate the requesters trust value and decide based on the local policy whether the user is trusted enough to join.

3.2.5 Group formation process

If the user is successful in there request to join the group they will be given a graphical user interface which provides all the functionality needed for the application. The application will decide how the group members will communicate. All communication between the group may be multicast or members may unicast to each other.

3.3 Design of system components

The last section introduced the requirements of the various processes that are needed in the collaborative ad-hoc application. This section describes the design of the three major components of the application. It will show how the

requirements identified in the last section have been integrated into the design of the components. The next section 3.3.1 discusses the design of the join protocol that allows users to form into, manage and exit groups. In the following section 3.3.2 the trust based access control design is presented. Section 3.3.3 handles the design of the components involved in the group formation and interaction.

3.3.1 Join Protocol Design

The request and voting processes both involve interaction between the client and application, it makes there for to combine the requirements of both in the design of the join protocol. The join protocol allows users in the system to form into collaborative groups. The protocol is designed so that every member of the group gets to verify the new user and can back-out of the group at any stage. This ensures that each member has accepted the risk of letting the new user join. The protocol allows a decentralised group decision to be made on the request by a user to join. The identities of the group members are hidden from the requesting user.

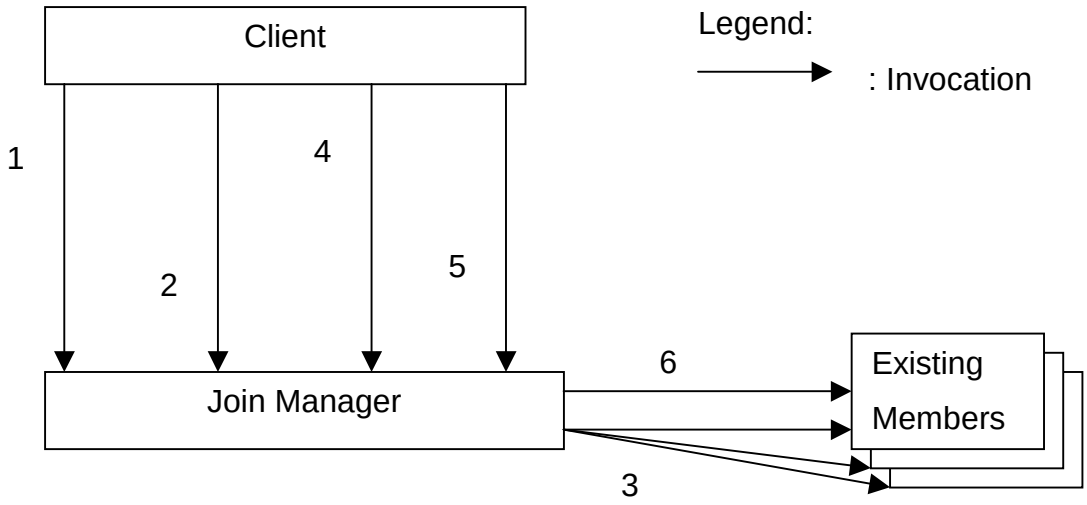

Figure 5: The Join Protocol

Once the user has selected an application and downloaded a service proxy, the join protocol will be initiated. The protocol involves six steps, which are labelled in figure 5 and explained below.

1. **Get Roles**: The client gets a list of possible roles that the application supports.
2. **Request to join**: The client requests to join the group in the highest role. The client will pass an identifier and the requesting role to the join manager.
3. **Verify User**: When the join manager receives a “request to join” from a client it will start the voting process. Each member of the group will be asked to verify the right of the user to join the group in the requested role. At the end of the vote, the application manager will inform the user of the result. If there is no group the user becomes the first member of a new group.
4. **Verify Group**: When the user’s request to join is accepted by the group, the client verifies the existing members. The client gets the identifiers and current roles of the existing group members. The client verifies the roles held by the existing members of the group to confirm that they pass the users own access policy. This is to ensure that there is no member of the group in a role that the client does not agree with. If the client’s request to join is rejected, they can attempt to join in a lower role.
5. **Officially Join**: When the user has verified the existing membership it will then officially request to join the group.
6. **Inform All**: All members of the group are informed that a new user has been granted access. Members can this opportunity to exit the group If they disagree with the result of the vote.

3.3.2 Trust-Based Access Control System Design

The design goal for the trust-based access control system used in the collaborative ad-hoc application is to minimise the amount of configuration the user has to do. The trust-based access control system requires a number of

support components to gather information, manage policies and product trust values. An overview of the system is shown in figure 6 and the design of these components is presented in this section.

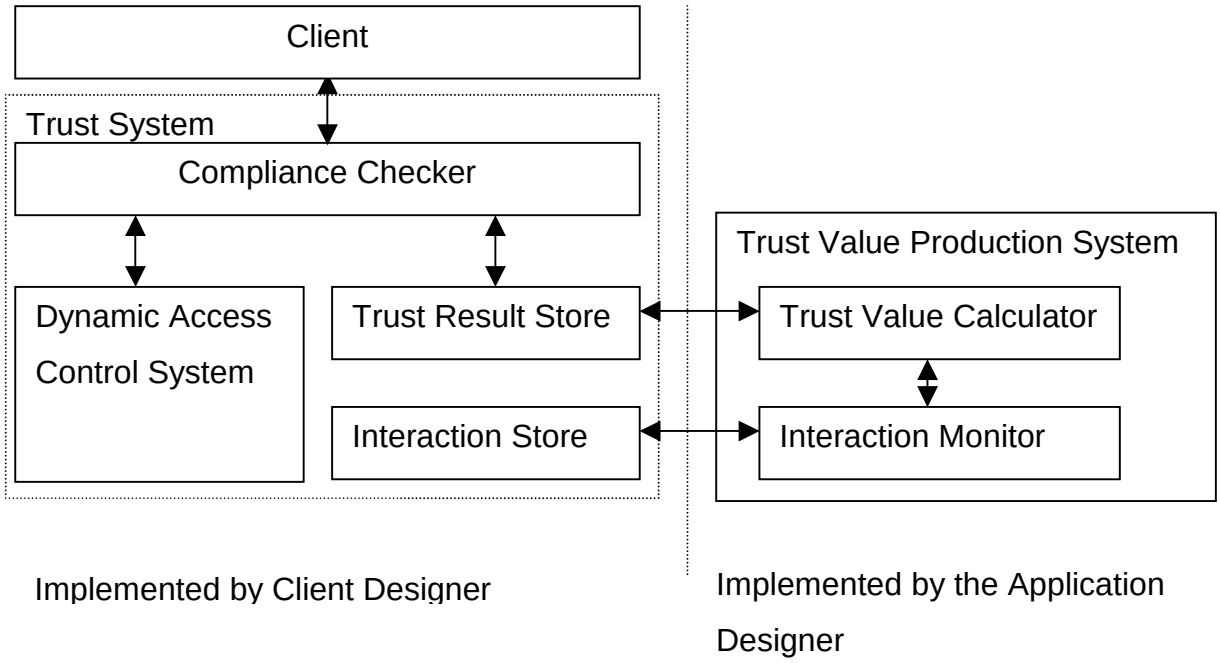

Figure 6: Trust-Based Access Control System

3.3.2.1 Compliance Checker

When a client has joined a group they are expected to verify the requesting users has the rights to join the group in a specific role. The compliance checker is passed the users name and requested role and will inform the client that the request should be accepted or rejected. The compliance breaks does this in three broad steps.

1. The compliance checker gathers the access control statements that are relevant to the requesting user.
2. The checker calculates the users trust value.
3. The users trust value and requested role are compared to the access policies. If the values agree with any of access policies the client is informed to accept the users request.

3.3.2.2 Dynamic Access Control System

The manageability of any access control system was identified in the literature review as being inversely related to the flexibility of the bindings used within the system, the more flexible the policy the easier it is to manage the system. We have seen that the access control list system is neither flexible nor scaleable because it binds the user's identity directly to the right to access resources. The role based access control system has poor scalability because it still requires one policy per user but it is more flexible because it binds the user's identity to a role this allows for easier management of the access rights per role.

The access control design used in this system achieves the requirement of dynamic access by binding a minimum trust value to the right to access a role. The statement reads, "If a user U has a trust value greater than X, then they can access role R". By binding a trust value to the right to access a resource, one statement will apply to every potential user. This means the role owner only has to define an access statement for every resource and not every user, this greatly reduces the number of policies the user has to initially write. The access control statements are designed to assign roles because it is easier to manage resource access for large groups this way.

The user will potentially be interacting in many different collaborative applications. A single access control statement format will apply across all applications so the name of the application to which the policy applies must be included in the statement. The user can also create a specific access control statement for a specific user by including the user's identifier in the statement. This allows the user to promote a user to a role that they would not be assigned by the general access policies.

The trust-based access control statement is must contain five piece of information.

- The name of the application to which the policy is relevant.

- The identifier of the user to which the policy applies.
- The condition that must be met by the users trust value for the policy to be true.
- The trust value threshold for the policy.
- The name of the role the user will be approved for if trust value meets the condition applied.

3.3.2.3 Trust Value Production System

The trust value production system is responsible for gathering information on the interactions with other users and producing trust values for the user about those people. The trust values created by each application are context (application) specific. Therefore it is the responsibility of the application designer to develop this component. The system has two sub-components, they are the interaction monitor and trust calculator.

3.3.2.3.1 Interaction Monitor

The interaction monitor gathers and saves information about the user's interaction with other users in the group. This information is used later by the trust calculator to produce trust values on each user. The application designers select what parameters of the interaction to monitor. The interaction monitor saves and updates the details of the user's interactions on the user disk, the format in which this information is stored is proprietary.

3.3.2.3.2 Trust Calculator

The trust calculator uses the interaction details saved by the interaction monitor to produce trust values. The trust values produced by the calculator are only relevant to one user. Because the trust values are all based on the same parameters the user can compare and rank the trustworthiness of the users they have interacted with.

The calculator will allow the user to customise the algorithm by changing the scaling factors used on the different parameters. This acknowledges the subjective nature of trust in that different users will rank different parameters as the most important.

The trust calculator would typically use linear equations when processing the interaction details and producing the trust result. The use of linear equations means mistakes in the calculator can be spotted and corrected easily. Users can also grasp the concept of a trust value increasing or decreasing on a linear scale as the monitored value changes.

The trust-value produced by the calculator is design to be within the range zero to one. It reflects that trust is a probability that a user will behave as expected, the higher the trust-value the more likely it is that the user will behave correctly. The trust value becomes asymptotic as it approaches one and zero, this reflects that you can never completely trust or distrust a user. The use of this range means the scaling of recommendation value is still within the range.

3.3.2.4 Trust results

When the trust calculator is executed at the end of the user's session it will save the trust values in a generic trust result on the users device. The trust result will contain the application name, which calculated the trust value. The need to identify which application produced the trust value arises because a user may interact with another user in many different contexts. The trust values produced from any two applications have completely different meaning because they are based on different parameters and produced by different calculators. This allows the compliance checker to identify the correct trust result where there are multiple trust results for the same user.

The trust value will also include the user name and the date on which the trust value was calculated. This date is important because trust derogates with time, if the trust result for a user shows the value was calculate a long time

ago the user should not believe that the user is still as trustworthy. The trust result must include four pieces of information.

- The name of the application to which the result applies.
- The name of the user to which the trust-value relates.
- The calculated trust value.
- The date on which the trust-value was calculated.

3.3.2.5 Trust System

The trust result store and interaction store allow the trust value production system to save and access the details on the client device. When the compliance checker and dynamic access control system are included, these four components are known as the client trust system.

3.3.3 GUI and Group Interaction Design

Once the collaborative group has formed the users will download a graphical user interface. The GUI is role specific and provides the trust-value production system to the client. The application designer will decide what communication mechanism is needed between the users of group.

3.3.3.1 GUI Design

There are two possible configurations that allow the client to use a graphical user interface with the application. The first has the GUI code stored on the client's device, when the user accesses the application the GUI is initialised. The second option has the GUI code stored with the application. The user must download the code whenever they use the application. The second design option was chosen for implementation in our system. The download option allows the application developer to make upgrades to the GUI and still ensure that all users have the most up-to-date version. The centralised control makes management easier. The main issue with this option is that the code

must be downloaded every time. As mobile networking technology gets faster the delay in downloading the code will become less noticeable.

The steps involved in accessing the GUI are shown in Figure 7.

1. The client completes the join protocol and has been given a reference to the group manager.
2. The client calls on the group manager to provide a GUI object. The client downloads the GUI Factory.
3. The client makes a call to the GUI Factory passing the user's role, a reference to the local trust system and a reference to the remote group manager.
4. The GUI Factory produces a role specific GUI for the user. When the GUI is initialised, the trust-value production system connects to the trust system. The interaction and trust result details are accessed and updated.
5. The role specific GUI will contact the group manager.

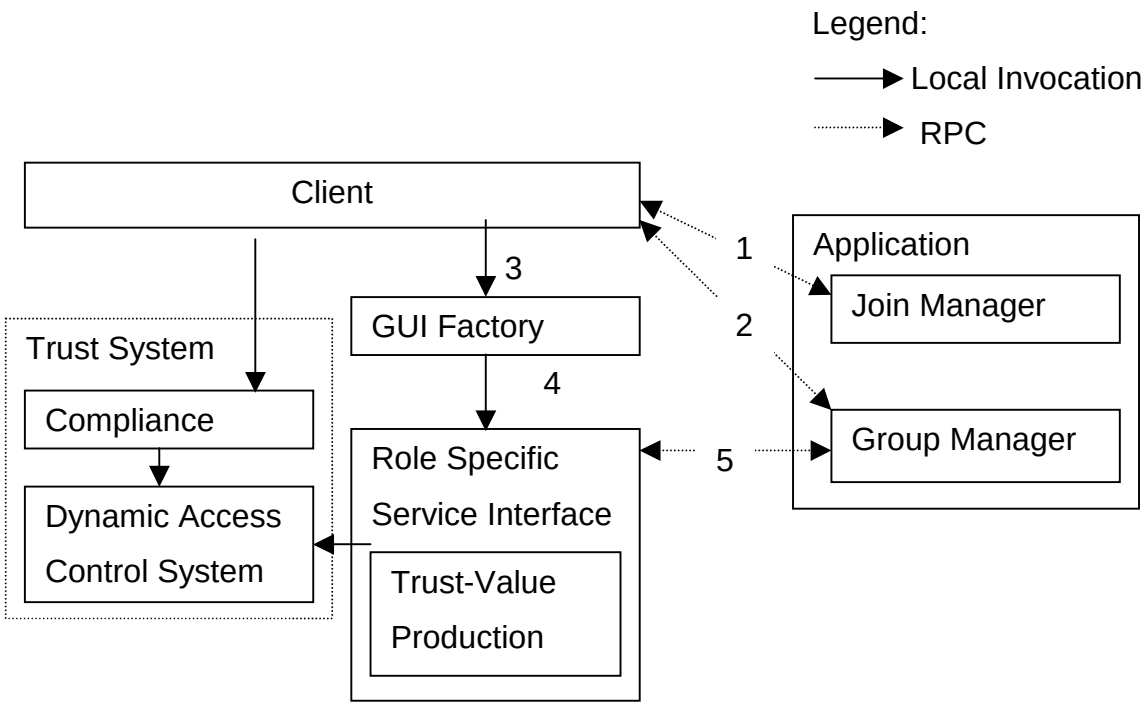

Figure 7: Accessing the Role Specific GUI

The application is designed to have to two management components they are the join and group manager. The functionality of the join manager does not change across different application, so designing a reusable component makes sense. The group manager provides the application specific services and so must be design from scratch for every new application.

3.3.3.2 Group Interaction

The last section described now the system is designed to allow the client to access a role specific GUI with the trust production system. This section describes now the GUI can communicate with others in the group. The application designer determines the requirement for communication between the clients. The client GUI is given a reference to the group manager when it is initialised; this allows the client GUI access to the centralised group manager. The group manager now has a reference to every client GUI and can decide how to manage this.

3.4 Summary

The design chapter first described the general architecture of the system. The five major processes involved in the system were then introduced and the requirements for each were identified. The functional design of three components was then undertaken. The functionality of the join protocol, the components involved in the trust-based access control system and the group communicating GUI has been identified. The next chapter describes how these components implement the required functionality.

4 Implementation

This chapter describes the implementation of the collaborative ad-hoc application framework and a sample application. Chapter 3 described the design of the various components and in the majority of cases the programming task was fairly simple. In this chapter, the implementation of the components is described as well as some of the issues that were encountered during the implementation process.

The chapter will first introduce the technologies used in the implementation. The next section gives a high level overview and explains the interfaces that make up the framework in some detail. The last section describes the implementation of the sample application.

4.1 Technologies

The Java programming language is used for the implementation of all components in this system, it assumed that the reader is familiar with the object-oriented nature of this language. The implementation used two specialised technologies for certain components. This section gives a short introduction to both.

4.1.1 JINI

A JINI system is a Java technology-centred, distributed system designed for simplicity, flexibility and federation [Sun]. The JINI architecture provides a mechanism for devices and programs to federate into groups and offer resources and share resources within that group. The architecture exploits the ability to move and execute Java programming language code from machine to machine [Jini].

The key concept in the within the JINI architecture is that of a service. A service can be downloaded and used by any entity in the federated group. Systems provide services to the federation by publishing them on a lookup service. The lookup service is central bootstrapping service and provides the central point of contact between the system and its users. The service is added to the lookup service by a pair of protocols known as the Discovery and Join Protocol [Sun].

The users of the JINI system contact the lookup service and download the service they require. All communication with the JINI architecture uses Java Remote Method Invocation (RMI), this allows data and code to be transferred around the network between objects. The JINI architecture assumes that network for connecting devices and services is available, and that each device has a Java virtual Machine installed. For more information on JINI see Appendix 8.1. The design chapter introduced the Join Protocol used in the collaborative ad-hoc application this is in no way related to the JINI Join Protocol.

4.1.2 XML

In the design of the access control system the saving of access control policies, trust results and application interaction details on the users devise was identified as a requirement. The decision to use Extensible Mark-up Language (XML) to save information was made for a number of reasons.

The information being stored on the file was already structured so it made sense to use a format that maintains this structure. The saved files are easier to understand by people because the structure of the information is included in the file.

The ease with which the XML files can be parsed and converted into Java objects was another advantage of this approach. Sun provide two API’s for parsing XML documents, each suited to different types of applications. The

“Simple API” for XML (SAX) is an event-driven, serial-access mechanism that processes the document in an element-by-element fashion. This mode was used in the implementation for reading all file because it is more memory efficient.

Another factor that contributed to the choice of XML as the storage format was the possibility of exchanging the interaction details of users in commercial transactions. If the collaborative ad-hoc application system becomes popular the interaction details stored by the services will potentially become more detailed. This details information might be valuable to companies who specialise in user profiling. Storing the data in XML format makes the information more valuable.

4.2 Framework Interface Implementation

The design identified a number of components that have to interact in a standard way in all collaborative ad-hoc applications. In order to create a standard framework the interaction between the various components will be defined in different interfaces. This section will discuss the design of these interfaces; a high-level overview of the collaborative ad-hoc application framework is shown in figure 8.

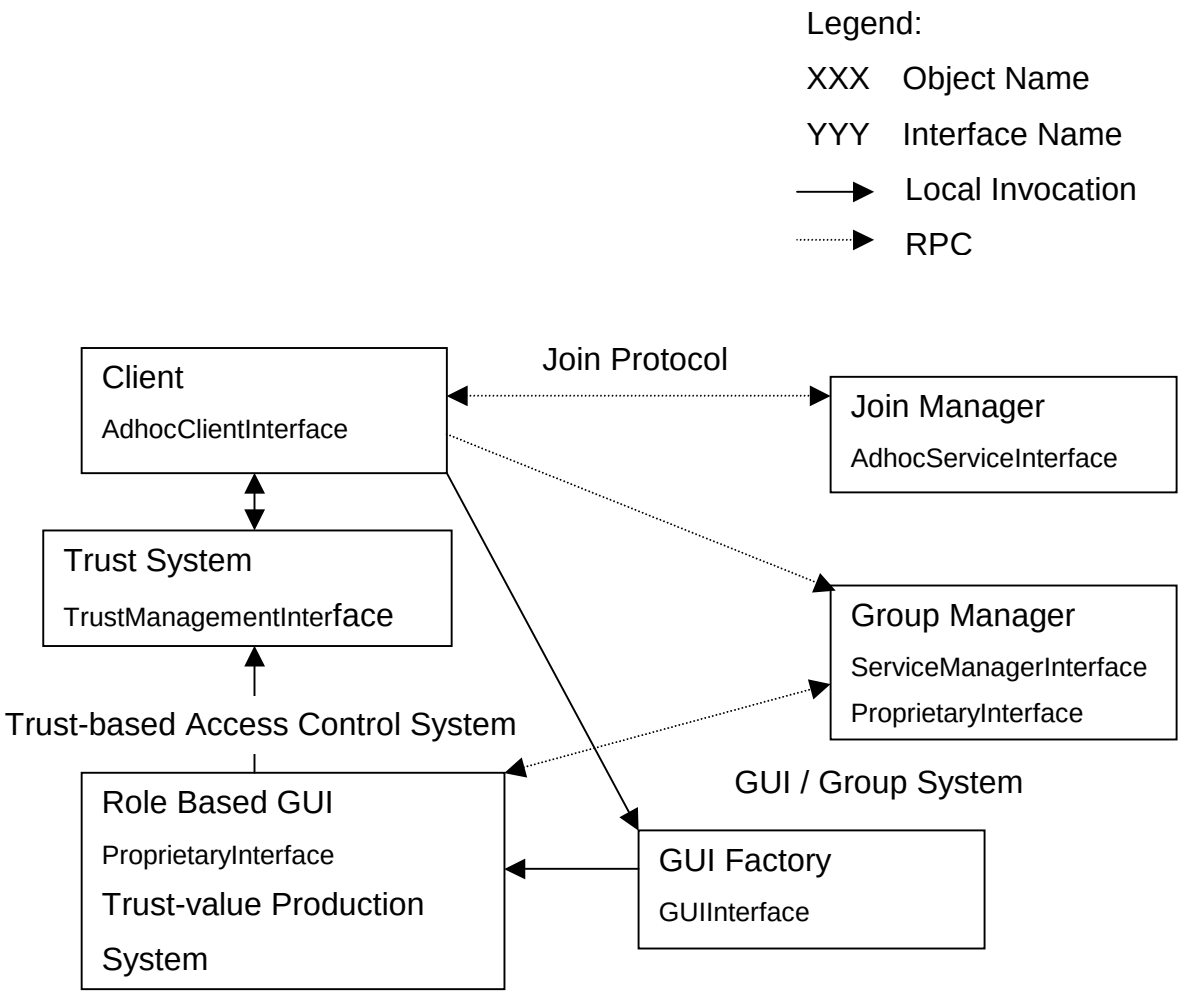

Figure 8: Implementation Overview

4.2.1 The Join Protocol Implementation

The join protocol involves the join manager and client. Both implement an standard interface that allows the other to invoke remote method calls. Section 3.3.1 discussed the design of the six-step join protocol.

4.2.1.1 AdhocServiceInterface

The AdhocServiceInterface supports a number of methods. The service proxy that the application publishes on the lookup service will be of this type. The users of the ad-hoc environment will be pre-programmed to search for services of this type first.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface AdhocServiceInterface extends Remote
{
public Vector getServiceRoles() throws RemoteException;

public boolean joinAsRole(PlayerDetails playerdetails, Role role)
throws RemoteException;

public ServiceManagerInterface requestLogon(AdhocClientInterface
adhocclientinterface, Role role, PlayerDetails playerdetails) throws
RemoteException;

public void logoff(AdhocClientInterface adhocclientinterface, PlayerDetails p)
        throws RemoteException;
}
```

The first two methods in the interface represent the first two steps of the join protocol. The third method represents the fifth step, the client officially joining the group. When this step happens the client is returned an object that implements the ServiceManagerInterface.

4.2.1.2 AdhocClientInterface

The client software must implement the AdhocClientInterface. The first three methods in the AdhocClientInterface interface are involved in the join protocol. The last method allows other clients to request a recommendation on a user.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface AdhocClientInterface extends Remote
```

```
{
public  boolean verifyPlayerForRole(PlayerDetails playerdetails, Role role)
      throws RemoteException;

public void informAboutNewUserRole(AdhocClientInterface adhocclientinterface, Role role, PlayerDetails playerdetails) throws RemoteException;

public  void userExit(AdhocClientInterface adhocclientinterface, PlayerDetails playerdetails) throws RemoteException;

public double recommendationRequest(PlayerDetails playerdetails, String context) throws RemoteException;
}
```

4.2.2 Graphical User Interface Implementation

Once the client has finished the join protocol they are returned a reference to the group manager. The client will use this reference to request access to a role-based GUI from the group manager

4.2.2.1 ServiceManagerInterface

The group manager will extend this interface, this allow any client to request access to a role specific GUI. The group manager will return an object that implements the GUIInterface.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface ServiceManagerInterface extends Remote
{
   public abstract GUIInterface getService() throws RemoteException;
}
```

4.2.2.2 GUIInterface

The application developer has to provide a role based graphical user interface to the users of the system. When the user has been approved for a role in the application, the group manager will return a factory object that implements GUIInterface. The interface contains a single method that has four parameters. The client passes a reference to its local trust system and the group manager. The trust-value production component of the GUI will use the TrustManagementInterface reference to contact the client trust system. The groupmanager_reference allow the GUI to contact the group manager. The GUIFactory will cast the object to specific proprietary interface type.

```
import java.io.Serializable;

public interface GUIInterface extends Serializable
{
public Object getUI(Object groupmanager_reference, Role role, PlayerDetails
playerdetails, TrustManagementInterface trustmanagementreference);
}
```

4.2.3 Trust Management Interfaces

As discussed in section 3.3.2, the design of the trust-value production system stated that the two subcomponents must be allowed to save and access details to and from the client's disk.

4.2.3.1 TrustManagementInterface

This interface is implemented by the client's trust system. It allows the downloaded trust-value production system to access the relevant application files on the client machine. As the session proceeds the interaction monitor can update the information held on the users in the current group. At the end of the session, the interaction monitor writes the details to the client, it also call the trust-value calculator to update its trust results on each client.

```
import java.io.Serializable;
import java.io.File;

public interface TrustManagementInterface extends Serializable
{
public File getXMLFile(String context);
public void writeXML(String context, String details);
public void addTrustManagementResult(TrustManagementResult result);
}
```

4.3 Trust Based Access Control System Implementation

This section presents some of the issues encountered in the implementation of the trust-based access control system. The design of this system requires that the dynamic access control statements and trust result should be uniform for all users and applications.

4.3.1 Dynamic Access Control Policy Object

The design identified that the access control statement had to contain five pieces of information. The access control policy is relevant to a particular role and application. The access control policy must also identify which users the are subject to the constraints of the policy. The implementation of this object in the system allows the user two combinations. The first choice states that the policy applies to all users, and is noted by a '*' in the user field of the statement. The second choice is more specific binds users identifier directly in the statement. This is similar to the mechanism used in the traditional access control list systems.

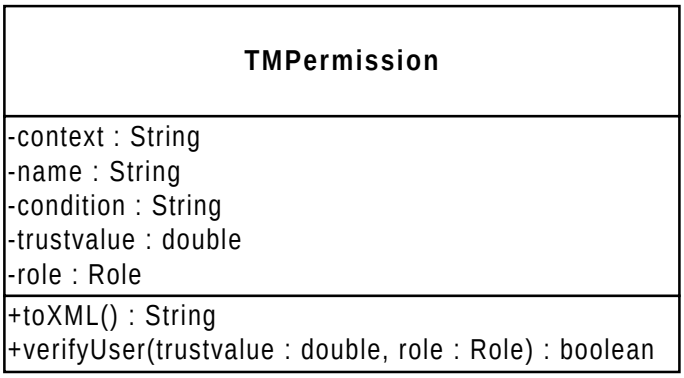

Figure 9: TMPermission object

The functionality of the access control statements is implemented in the TMPermission object. This object allows the user to set and get the data held in the five data fields. The object also implements a method that verifies if a user's trust value and requested role meet the conditions set out in the access control policy, it returns a boolean result. The suedo code for the verifyUser method is shown in Figure 10.

```
public boolean verifyUser(double users_trustvalue, Role requested_role)
{
if( (users_trustvalue>trust_threshold) && (requested_role==access_role ) )
        return true;
        else
        retrun false;
}
```

Figure 10: TMPermission verifyUser() seudo code.

The object supports a XML conversion function that is implemented to support the storage of the access control policy information on the client disk. An example of a generic access control policy can be seen in Figure 11.

```
<permission>
        <context>application_name</context>
        <applysto>*</applysto>
        <condition>greater_than</condition>
        <trustvalue>0.2</trustvalue>
        <role>player</role>
</permission>
```

Figure 11: Generic Access Control Policy Statement [XML Format]

4.3.2 Trust Result Object

The trust result object has four fields that can be accessed and modified. The object will also produce a XML format string for the purposes of storage, an example is shown in Figure 12.

```
<result>
        <applysto>liam</applysto>
        <context>blackjack</context>
        <lastupdate>28/8/2000</lastupdate>
        <trustvalue>0.1584</trustvalue>
</result>
```

Figure 12: Trust Result XML format.

4.3.3 Client Trust System Implementation

The client trust system designed in section 3.3.2.5 is integrated into the client software, an overview of the system is shown in figure 13. When the client software is initialised the trust result and access control files are parsed and the respective objects are stored in the trust result and policy store. When the

ad-hoc client system receives a call to verify a user during the Join protocol the compliance checker is called. The compliance checker calls the trust result store and policy store to gather the relevant policies and trust results. The trust value for the user and the requested role is verified with each policy, if any of the policy indicates a match the users request is confirmed.

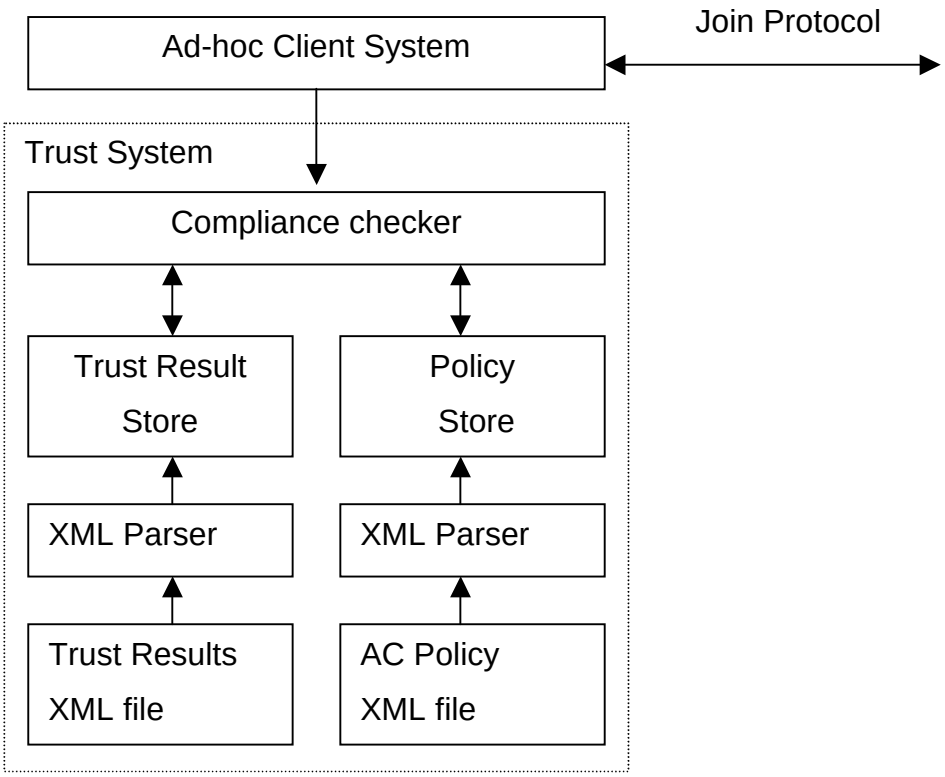

Figure 13: Trust System Implementation Overview.

4.4 Sample Application Implementation

The sample application that was implemented for demonstration of the trust-based access control systems use in the collaborative ad-hoc application was the card game blackjack. Blackjack is a popular card game in which users gamble with a dealer over the value of a set of cards. The game was chosen because the use of trust is ingrained in the way players join the game.

The similarity in the use of trust to assign roles to users makes it an ideal game to implement, this allow for comparison to be made between the human and the computer model of trust. In implementing the game it was hoped that the trust-based access control system developed for this dissertation would behave in a similar way to a humans.

In blackjack there are two distinct roles, one dealer role and up to five player roles. The role a user has in the game has a major impact on the willingness of other user to play with that user. The user holding the dealer role must be considered trustworthy to the other players. The trustworthiness of the user is based on three parameters, the number of times the two users have played, the number of days since the two played last and the amount of debt that exists between the two.

4.4.1 Trust-Based Access Control

After every session the interaction monitor will update the values of the monitored parameters associated with a user. These details are stored on a client device in a XML file. The TrustManagementInterface allows the interaction monitor to save the result, the format is shown here.

```
<details>
        <name>liam</name>
        <sessions>2</sessions>
        <lastsessiondate>8/8/2000</lastsessiondate>
        <debt>10</debt>
</details>
```

Although the parameters that are monitored are the same for all users, the user can still customise the trust calculator by defining individual scaling factor for each trust level calculation. A configuration window is presented with the users GUI. These details are also stored in the XML file in this way when the client has to update their trust values the general trust calculator will produce client specific results.

```
<trustmanagementconfiguration>
        <sessionlimit>10</sessionlimit>
        <lastplayedlimit>100</lastplayedlimit>
        <debtlimit>80</debtlimit>
</trustmanagementconfiguration>
```

When a trust value is required for a user, the trust level for each of the parameters is calculated, the three trust level are then multiplied to return trust value which combines the three parameters. This value is then stored as a trust result, the format is shown here.

```
<trustresults>
        <result>
                <applysto>liam</applysto>
                <context>blackjack</context>
                <lastupdate>28/8/2000</lastupdate>
                <trustvalue>0.1584</trustvalue>
        </result>
<trustresult>
```

4.4.2 Blackjack Group Formation

When a user has been assigned their role and successful joined the group the blackjack group formation process begins. It was stated in section 3.3.3 that the application developer determines how the users of the group will interact. In the blackjack implementation it was decided that all players are required to connect directly to the user who has the dealer role.

When the BlackJackDealerGUI is initialised it will register with the Group Manager, which implements the BlackJackManagerInterface. When the BlackJackPlayerGUI is initialised it connects to the Group Manager and gets a reference to the dealer of the group. The BlackJackPlayerGUI now has a reference for the dealer, and it will register itself to start the game.

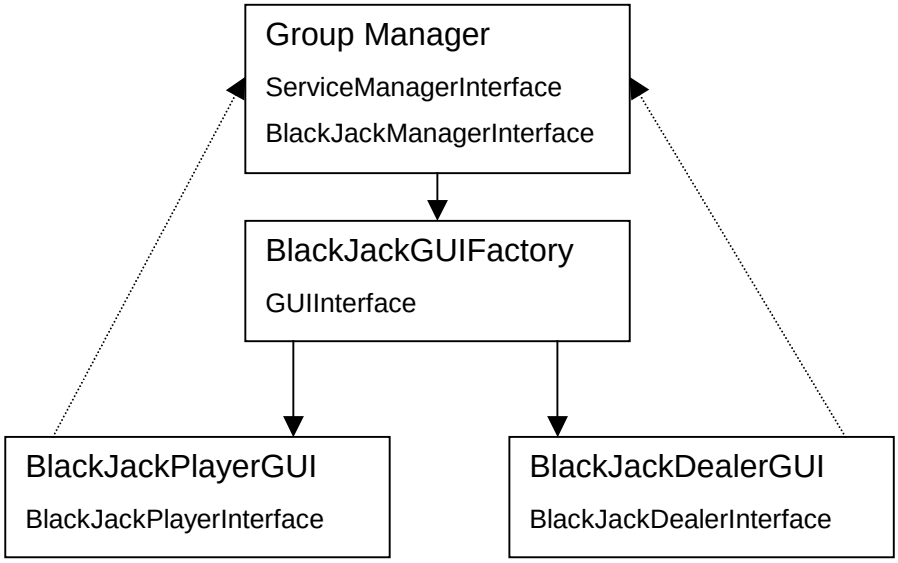

Figure 14: BlackJack Group Formation Overview

4.4.2.1 BlackJackManagerInterface

The BlackJackManagerInterface allows the dealer to register its proxy in a central well known location. The players can then ask the GroupManager to pass them a reference to the dealer. The interface essentially allow the users to find each other.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface BlackJackManagerInterface extends Remote
{
        public abstract void registerDealer(BlackJackDealerInterface
        blackjackdealerinterface) throws RemoteException;

        public abstract void unregisterDealer() throws RemoteException;

        public abstract BlackJackDealerInterface getDealerRef()
        throws RemoteException;
```

}

4.4.2.2 BlackJackDealerInterface

The BlackJacKDealerInterface implements two methods that allow the players to join and leave the blackjack game. The remaining methods provide the basic functionality for the execution of the blackjack game.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface BlackJackDealerInterface extends Remote
{
        public abstract String getAlias() throws RemoteException;

        public abstract void logon(BlackJackPlayerInterface
        blackjackplayerinterface) throws RemoteException;

        public abstract void logoff(BlackJackPlayerInterface
        blackjackplayerinterface, int settleamount) throws RemoteException;

        public abstract void hit() throws RemoteException;
        public abstract void stand() throws RemoteException;
        public abstract void doubledown() throws RemoteException;
        public abstract void splitpair() throws RemoteException;
        public abstract int settle(Bet bet) throws RemoteException;
}
```

4.4.2.3 BlackJackPlayerInterface

The BlackJackPlayerInterface allow the player receive updates and commands about the game from the BlackJack dealer.

```
import java.rmi.Remote;
```

```
import java.rmi.RemoteException;

public interface BlackJackPlayerInterface extends Remote
{
	public void enableClient(boolean flag) throws RemoteException;
	public void updateGame(Game game) throws RemoteException;
	public void sendMessage(String s) throws RemoteException;
	public String getAlias() throws RemoteException;
	public Bet getBet() throws RemoteException;
}
```

5 Evaluation

The evaluation of the work carried on collaborative ad-hoc application will examine a number of aspects of the system. The first section will examine the design and implementation of the components and interfaces that make up the collaborative ad-hoc applicaion framework. The second section will look at the implementation of the sample application. The focus here will be on the behaviour of the trust based access control system.

5.1 Evaluation of Framework

The framework is made up of five interfaces; the focus of this evaluation is to examine if interfaces offer the complete functionality and to recommend what improvements might be made in future.

5.1.1 The Join Protocol

The AdhocClientInterface and the AdhocServiceInterface designed in section 3.3.1 and implemented in section 4.2.1 provide the functionality for the join protocol. The major problem that has been identified with the join protocol is the client might cause the protocol to run a number of times before they receive a role. In an application where there are multiple roles, this could put a large load on the application and clients. A possible solution to this problem is to allow the clients say what role they will approve a user for, in the first Verify User step of the protocol instead of just confirming or denying multiple requests. This would complicate the join managers voting scheme and the clients trust management system but it would reduce the number of times the protocol is run.

5.1.2 TrustManagementInterface

The methods provided in the interface allow the application based trust-value production system to access and store the information they produce as per the requirements set out in section 3.3.2. In hindsight the interface should have included one more method, this would separate the trust-based access control system from the client and provide greater flexibility. The method would allow the client to ask the trust-based access control system to “verify a user”. In the current implementation the trust-system is integrated into the client, by including this method the trust-system can be separated. By separating the two it allows for easier updating of both components.

5.1.3 ServiceManagerInterface

The ServiceManagerInterface provides two benefits to the framework. It allows the group manager to take over control once the join manager has finished the join protocol with the client. It also provides a standard way for client to get access to an object that implements the GUIInterface.

5.1.4 GUIInterface

The GUIInterface can be considered a success because it is so accessible to the client and application. It provides a single way for all client to get access to role-specific application-specific graphical user interface. For the application developer it provides an easy mechanism for getting access to the clients trust-system. It also supports the formation of communication links between the application GUI’s, by giving them a reference to the group manager.

5.2 Evaluation of Sample Application

The dissertation used an implementation of the blackjack card game to evaluate the performance of the trust-based access control system in a collaborative ad-hoc environment. The focus of the evaluation on the sample

implementation is to examine the behaviour of the trust-based access control system.

5.2.1 Trust-Based Access Control Results

A sample of interaction details, the trust-values calculated based on them and the achieved roles are presented in table 1. These are the details gathered on one of the users in the blackjack collaborative ad-hoc application.

No Sessions	Days Since	Debt Level	Trust Value	Role
1 (.25)	1 (.99)	+10 (.99)	.245	Player
2 (.5)	1 (.99)	+10 (.99)	.495	Player
3 (.75)	1 (.99)	+10 (.99)	.735	Dealer
4 (.99)	2 (.75)	-10 (.8)	.594	Player
5 (.99)	2 (.75)	0 (.99)	.735	Dealer

Table 1: Interaction details, trust-value and user role.

The Blackjack interaction monitor follows three parameters of the interaction between two users. It monitors the total number of sessions the two user have shared, the number of days since the last session, and the amount of debt between users. It is assumed in the blackjack application that users will settle their debt at the end of each session. The trust calculator takes these details and produces a partial trust value for each parameter, seen in brackets in table 1. The partial trust-values are then multiplied together to produce the overall trust-value display in column 4.

The calculator uses linear equations to produce the partial results. The trust value for the “number of session’s” parameter rises gradually and the user is completely trusted after four sessions. The trust value for the “days since last session“ parameter fall as the number of days increase, in this case the limit is also four days. The trust value for the “debt” parameter is more complicated. If the debt level is positive it means the other user is extending credit, in this situation a creditor always fully trusted. In the other case where the user is a debtor, the trust level falls as the amount of debt gets larger in the this case the debtor is no longer trusted once their debt is greater than 50.

The user in the blackjack game has defined two access control policies that apply to all users of the system. The two policies are seen in figure 16, the first policy state that any user with a trust value greater then 0.2 can join as a player.The second allows any user with a trust value greater than .6 to join as a dealer.

```
<permission>
        <context>blackjack</context>
        <applysto>*</applysto>
        <condition>greater_than</condition>
        <trustvalue>0.2</trustvalue>
        <role>player</role>
</permission>
```

```
<permission>
        <context>blackjack</context>
        <applysto>*</applysto>
        <condition>greater_than</condition>
        <trustvalue>0.6</trustvalue>
        <role>dealer</role>
</permission>
```

Figure 15: Generic Access Control Statement for the Player Role

A graph of the overall fluctuation is shown in figure 16.

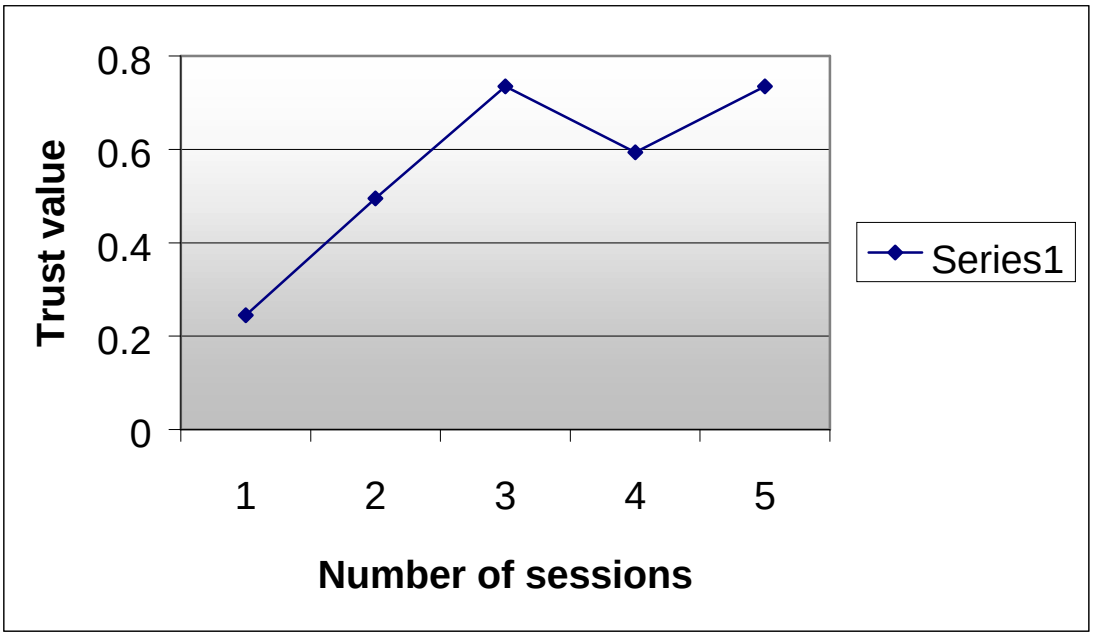

Figure 16: The change in a users trust value over a number of seesions

5.3 Summary

The evaluation of the collaborative ad-hoc application framework has shown that all the requirements of framework designed in chapter 3 have been implemented successfully into the components and interfaces described in chapter 4. The evaluation of the framework has identified some improvements that can be made to improve the flexibility of the overall system.

The evaluation of the trust-based access control system show that is will react correctly to a change in behaviour of the user. The results presented in this chapter are very limited due to a lack of time. Future evaluation of the trust-based control system should test the performance of the compliance checker.

6 Conclusions

The introduction chapter of this dissertation identified that users currently have no way of establishing trust in others directly. There is a reliance on a perceived or recommended trust in users, which is too vague when security critical issues are being considered. A proposal was suggested that a trust-based access control system would provide a solution to the problem of establishing direct trust in an open system.

A literature review was carried out that examined the technologies used in traditional access control and authentication systems. The concept of trust, a framework for trust production and the use of recommendations were also examined. A design was developed for a collaborative ad-hoc application framework and trust-based access control system so we could test out proposal. The design was then implemented and evaluated. In this chapter a general review of the whole work and the issues raised will be presented as well as a perspective on where future work may be directed.

6.1 Dissertation Review

Traditional access control systems bind users names to the right to access a resource. Looking at the access control statements a user cannot establish why the administrator trusts an individual to perform an action correctly. The assumption is that a user must be prepared to rely on the administrator ability to correctly perceive a user's trustworthiness.

This assumption is rejected at the beginning of this dissertation and mechanism for establishing trust in a user became the focus of the dissertation. Having identified the focus for our research the of the

6.2 Future Work

There are a number of avenues future work on this system could take. The framework for trust production could be examined in more detail. This would allow the selection of monitoring parameters to be more structured way, it might allow identify common monitoring parameters across all trust systems. The second avenue is the performance of the compliance check and trust-value production system, this dissertation did not get the time to fully examine this question.

6.3 Application of trust based access control

Mobile computer devises are becoming more wearable every day with palm-top and mobile phones becoming fashion accessories. The possibility exists that a collaborative ad-hoc client could run on one of these devises, the client would represent the user in gaining access to resources in the locality automatically. The user will not have to make explicit requests to access resources because the system will have already arranged it, this increases the efficiency of the user.

6.3.1 Intelligent doors

In a large public building access to certain rooms might be restricted to frequent trustworthy visitors. An airport waiting lounge where first class ticket holders and frequent flyers can use a VIP lounge, but the majority of normal users must stand is a good example of this. Given the large number of people moving through the airport and the benefit of the VIP room, the management have to maintain a presence at the entrance and check all users.

A trust-based system could be used to protect access to the VIP room. The first class and frequent flyers will build up trust relationship as they interact with the airport trust system. When a VIP flyer arrives into the airport there

mobile devise will immediately join the group. The client will request to join the group in the VIP role, which allows access to the VIP waiting room. As the user approaches the entrance to the VIP lounge, the trust based system controlling the doors will recognise that the user has the right to enter and it will open automatically.

6.4 Summary

The trust-based access control system designed, implemented and tested for use with in the collaborative ad-hoc application environment shows that direct trust establishment is possible in large open networks. The benefit of using a trust model to establish access control for mobile users inn open networks is it provides a highly scalable and manageable solution when compared to the traditional access control list and role based access control systems. The implementation of a more extensive trust model should allow the use of trust-based access control systems to be applied to the protection of resources in real situations.

7 Bibliography

[Abdul1] A Abdul-Rahman, Stephen Hailes. A Distributed Trust Model. New Security Paradigms 97.

[Abdul2] A Abdul-Rahman, Stephen Hailes. Using Recommendations for Managing Trust in Distributed Systems.

[Abdul3] Alfarez Abdul-Rahman. The PGP Trust Model

[Blaze1] M Blaze, J Feigenbaum, J Lacy. Decentralised Trust Management. Proceedings of the IEEE Symposium on Security and Privacy, pages 164-173, 1996.

[Blaze2] M Blaze, J Feigenbaum, et al. The role of Trust Management in Distributed System Security.

[Herzberg] Amir Herzerg. Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers.

[Jensen] Christian Jensen, Mads Haahr. Towards a Security Framework for Ad Hoc Applications. Position Paper.

[Jini] W Keith Edwards. Core JINI. Prentic Hall

[Khare] Rohit Khare, Adam Rifkin. Trust Management on the World Wide Web. First Monday.

[Kuhn] Rick Kuhn. An Introduction to Role Based Access Control. NIST CSL Bulletin on RBAC

[Marsh] Stephen Marsh. Formalising Trust as a Computational Concept. Ph.D Thesis, Department of Computer Science and Mathematics, University of Stirling, April 1994

[McKnight] D.H. McKnight, L.C. Chervany. The Meaning of Trust. Working Paper, Carlson School of Management, University of Minnesota, 1996.

[Povey] Dean Povey. Developing Electronic Trust Policies using a Risk Management Model

[Reagle] Joseph Reagle, Trust in Electronic Markets: The convergence of crypyographers and economists. First Monday, August

1996.

[Sun] Sun MicroSystems. Jini Architecure Specification. Revision 1.0 January 25,1999.

[Swarup] Vipin Swarup, Javier Thayer Fabrega. Trust: Benefits, Models, and Mechanisms.

8 Appendix

8.1 Appendix A: JINI

8.1.1 Introduction

JINI is a new distributed paradigm developed by the Sun Microsystems in the late 1990's. The role of JINI is to "federate groups of devices and software components into a single dynamic distributed system" [Sun]. The JINI architecture is based on the concept of service. A service is an entity that provides some functionality to a user and is connected to a network. The JINI model provides a set of components that enable services to be federated into groups, these components are known as the infrastructure components.

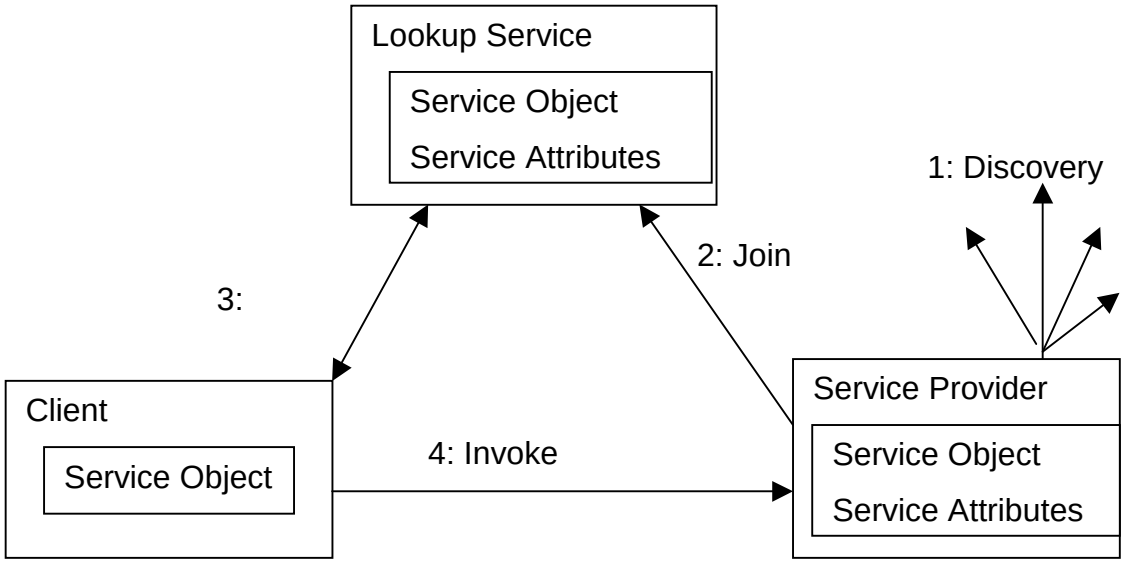

Figure 17: The JINI architecture

There are three types of entities within a JINI framework.

1. The Lookup Service is a central bootstrapping mechanism for the system and it provides a central point of contact between the services on the system and the users of the system.
2. Services: are initially registered in the lookup service by the system.

3. Clients are entities, which use the services provided by the lookup service.

The components that the JINI model provides can be segmented into three categories. Infrastructure components: allow the entities in a JINI framework to discover each other and form themselves into groups. Programming components: a set of JINI interfaces that provide a distributed extension to the standard Java Programming model. The JINI programming components allows for leasing, events and notifications, and transactions. Services components: use the infrastructure and programming model to make themselves available in the networked federation. The operations that a service can provide are defined through an interface

The process of forming federated groups for the three basic entities of the JINI framework is handled by the infrastructure components. The Lookup service is assumed to be initially available.

8.1.2 Discovery Protocol

When a service provider initially connects to the network they will initially use the discovery protocol to announce their presence. This process is known as “service initiated discovery” and involves the service multicasting to a predefined multicast address, this is seen in arrows 1 of figure 17.
Any lookup services that are running on the network will respond to the multicast message by sending a proxy that implements the *ServiceRegistrar* interface to the service, so further unicast communication can occur between them. At the end of the discovery protocol the service will have a reference to every lookup service running on the network.

8.1.3 Join Protocol

The join protocol now begins. The service provider creates a *ServiceItem* that contains the service proxy object and attributes that describe the service. The service provider will then publish the service on a lookup service, by calling

the register method on that lookup services proxy passing the *ServiceItem* as a parameter. The second join stage is shown in arrow 2 of figure 17. The service is now available to all users on the network.

The discovery and join protocols allow service providers to register their services with the lookup service in a high customisable way. The second aspect that the infrastructure components must handle is interaction between the clients and the lookup service.

8.1.4 Lookup

The alternative aspect of the infrastructure components is that they provide support for the discovery by clients of what services are available on the lookup service. When a client joins a network it will start to listens for periodic multicast messages from the lookup service. When a client discovers a lookup service it can search the lookup service for services that it is interested in via the *ServiceRegistrar* interface. The client can create a search template that describes the type and extra attributes of the service it is looking for, this object is known as a *ServiceTemplate*. The client can then ask the lookup service to search for services that match the clients criteria by calling the lookup method on the *ServiceRegistrar* object passing the *ServiceTemplate* as a parameter. If the lookup service contains matching services these are returned. This step is shown in arrow 3 in figure 17.

8.1.5 Service Proxy

The final step in this stage is for the client to make direct contact with the service provider via invocation on the service object. Since the service object originates from the service provider, the communication protocol used between them can be service specific. This step is shown in arrow 4 in Figure 17.