Policy Based Framework for Trust Management and Evolution of Peer to Peer Groups

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

by

Madhumita Chatterjee
(Roll No. 04429802)

Under the guidance of Prof G. Sivakumar and Prof Bernard Menezes

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY–BOMBAY
2012
DEDICATED TO

My Late Husband
Shri Anjan Chatterjee.
Thesis Approval

The thesis entitled

Policy Based Framework for Trust Management
and Evolution of Peer to Peer Groups

by

Madhumita Chatterjee
(Roll No. 04429802)

is approved for the degree of

Doctor of Philosophy

Examiner

Guide

Examiner

Co Guide

Chairman

Date: 29/07/2013

Place: IIT Mumbai
Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

[Signature]
Madhurnita Chatterjee
(Name of the Student)
04429802
(Roll No.)

Date: 29/07/2013
Abstract

Peers with common interest often create collaborative groups for achieving common goals. Being dynamic and self-organizing in nature, these collaborative groups have frequently changing application requirements and special security requirements not addressed by typical access control mechanisms. They require a secure and reliable group communication system to ensure that they evolve and adapt according to interactions within the group. In the interests of the group there is a need for peers to collaboratively modify access control policies dynamically. Self-organizing and dynamic nature of peers necessitates an environment for different functional roles. Another major challenge for collaborative P2P systems is the ability to manage risks involved in interacting and collaborating with previously unknown and possibly malicious peers.

An interesting feature of collaborative groups is that they tend to organize themselves into different system models. A system model defines the behavior of peers, different roles, events, and policies for membership control, access control, task assignment, security and others. When providing access control during an application, the security context (e.g., the policies, team members and/or roles) may change. Such changes must also be controlled and fed into access control system. While evolving, peers thus have to decide which policies to chose and which system model to follow. In this thesis we deal with two issues of group management, viz. dynamic adaptation of admission and access control policies and varying system models to optimize group performance.

We propose a framework for dynamic collaborative groups which has a novel method for policy based management of groups for membership control, access control and resource scheduling. Collaborative groups with different system models or workflows can be represented and modeled in our framework. We are able to express different policies using a declarative language and our framework permits deployment of these policies flexibly and dynamically. The group can dynamically prioritize requests for join and peers can self regulate their behavior.
based on an adaptive trust mechanism. Peers need not be symmetric and can be entrusted with different functional roles which could also change dynamically. The group can adapt and switch between these policies and varying system models. Our framework thus supports dynamic adaptation of policies and adaptive re-organization of the group system models to optimize group performance. We model this behavior of groups using statecharts, where we are able to depict both the dynamically changing policies within a system model as well as the transition of groups from one system model to another. We also propose an adaptive multi-dimensional trust metric which supports multiple context based attributes. The trust model provides incentives to good peers while malicious peers are gradually isolated and eventually ejected from the group.

We have designed a tool capable of achieving this dynamic policy adaptation and optimization by dynamically changing the group behavior model itself. We designed the tool using Peersim Simulator by integrating it with Prolog. The tool can be used to test the evolution of peer groups based on different policies. We validate this with experiments and our graphs show that such collaborative groups with varying compositions perform more efficiently in the presence of dynamic policies with adaptive trust. We further show that by dynamically changing the system model we are able to optimise the group performance. Experiments with our adaptive trust model show that collusive behavior can be detected and the effect of malicious peers is mitigated.
Contents

Abstract vii
List of Tables xiii
List of Figures xv
1 Introduction 1
   1.1 Challenges of Collaborative Peer Groups . . . . . . . . . . . . . . . . . . . . . 2
   1.2 Motivating Applications . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
   1.3 Problem Statement and Solution Approach . . . . . . . . . . . . . . . . . . . . 8
   1.4 Contributions of the Thesis . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
   1.5 Dissertation Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2 Related Work 15
   2.1 Access Control . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
      2.1.1 An Overview of Early Access Control Models . . . . . . . . . . . . . 16
      2.1.2 Identity Based Access Control (IBAC) . . . . . . . . . . . . . . . . . . 17
      2.1.3 Role Based Access Control (RBAC) . . . . . . . . . . . . . . . . . . . 17
      2.1.4 Authenticated Role Based Access Control (ARBAC) . . . . . . . . . . 18
      2.1.5 Attribute Based Access Control (ABAC) . . . . . . . . . . . . . . . . . 19
   2.2 Securing Group Communication Systems . . . . . . . . . . . . . . . . . . . . 19
      2.2.1 Access Control in Collaborative Groups . . . . . . . . . . . . . . . . . . 20
      2.2.2 Policy Based Management in Collaborative Groups . . . . . . . . . . 23
      2.2.3 Trust Management . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
      2.2.4 Reputation Based Trust Management . . . . . . . . . . . . . . . . . . . 30
   2.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
3 Generic Application Framework for Collaborative Peer Groups 37

3.1 Generic Model ................................................. 38
  3.1.1 Group Activities ........................................... 39
  3.1.2 Group Efficiency .......................................... 40
  3.1.3 Group Goals ............................................... 40

3.2 System Architecture ......................................... 41
  3.2.1 Group Roles ................................................ 41
  3.2.2 Domain Specific Role ..................................... 42
  3.2.3 Functional Components ................................... 42
  3.2.4 Detailed Working of the Framework ...................... 44

3.3 System Model .................................................. 50
  3.3.1 Peer Attributes ............................................ 52
  3.3.2 Peer Behavior and Roles .................................. 53
  3.3.3 Events ................................................... 53
  3.3.4 Tasks .................................................... 53

3.4 StateChart Based Formal Model for Dynamic Groups .............. 55
  3.4.1 States and Transitions .................................... 56
  3.4.2 Guards and Conditions .................................... 57
  3.4.3 Events and Actions ....................................... 58
  3.4.4 State Transition Depicting System Model Change ........ 58
  3.4.5 Formal System Model ..................................... 61

4 Dynamic Policy Driven Trust Based Access Control (DPDTBAC) 63

4.1 Proposed Access Control Model .................................. 64
  4.1.1 Global Policy .............................................. 65
  4.1.2 Domain Specific Policy .................................... 65

4.2 Modeling Policies using Prolog .................................. 67
  4.2.1 Modeling Rules for Join .................................... 69
  4.2.2 Modeling Rules for Level Update ......................... 70
  4.2.3 Modeling Dynamic Rules ................................... 70
  4.2.4 Modeling Constraints ..................................... 72
List of Tables

5.1 Variation of job allocation ........................................... 77
5.2 Probability of distribution of bugs with type of peers ............ 102
5.3 Default trust value of peers .......................................... 102

6.1 Comparision Of Features Of Policy Specification Frameworks .......... 110

7.1 Comparison of Trust Metrics ........................................... 132
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Role based access control</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Generic model</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>Integrated framework</td>
<td>43</td>
</tr>
<tr>
<td>3.3</td>
<td>New member join</td>
<td>46</td>
</tr>
<tr>
<td>3.4</td>
<td>Member level update</td>
<td>48</td>
</tr>
<tr>
<td>3.5</td>
<td>Policy based framework</td>
<td>49</td>
</tr>
<tr>
<td>3.6</td>
<td>StateChart model</td>
<td>59</td>
</tr>
<tr>
<td>5.1</td>
<td>Overall framework</td>
<td>82</td>
</tr>
<tr>
<td>5.2</td>
<td>Architectural model</td>
<td>83</td>
</tr>
<tr>
<td>5.3</td>
<td>StateChart model for f/oss application</td>
<td>85</td>
</tr>
<tr>
<td>5.4</td>
<td>Static vs dynamic join</td>
<td>92</td>
</tr>
<tr>
<td>5.5</td>
<td>Varying job allocation with easy jobs</td>
<td>93</td>
</tr>
<tr>
<td>5.6</td>
<td>Varying job allocation with difficult jobs</td>
<td>94</td>
</tr>
<tr>
<td>5.7</td>
<td>Static vs dynamic job allocation</td>
<td>94</td>
</tr>
<tr>
<td>5.8</td>
<td>Average current rating versus time taken</td>
<td>95</td>
</tr>
<tr>
<td>5.9</td>
<td>Static vs dynamic policy in presence of 10 percent malicious peers</td>
<td>95</td>
</tr>
<tr>
<td>5.10</td>
<td>Static vs dynamic policy in presence of 30 percent malicious peers</td>
<td>96</td>
</tr>
<tr>
<td>5.11</td>
<td>Static vs dynamic policy in presence of 30 percent malicious peers and all difficult jobs</td>
<td>96</td>
</tr>
<tr>
<td>5.12</td>
<td>Comparison of 10 percent vs 30 percent malicious peers</td>
<td>97</td>
</tr>
<tr>
<td>5.13</td>
<td>SM1 versus SM2 in presence of difficult and easy jobs</td>
<td>98</td>
</tr>
<tr>
<td>5.14</td>
<td>Varying system models with varying job compositions</td>
<td>98</td>
</tr>
<tr>
<td>5.15</td>
<td>Dynamic switching between system model 1 and system model 2</td>
<td>99</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Web based collaborative groups are quite popular, as peers with common interests form a network among themselves and automatically tend to create interest groups among each other, called communities. Some examples of intrinsic communities being formed are Yahoo Groups and Google Groups. Applications like F/OSS [65], file-sharing [28], online gaming [101], video/audio conferencing, collaborative work-space, virtual meetings, distance learning environments, discussion forums and board rooms are examples of applications that are organized as peer groups. Such communities or groups need a communication model in which all the peers must collaborate in order to provide the basic services such as content or messages. The group is governed by a set of rules that describe the conditions required to be part of the group. These initial or minimal set of rules or access policies form the group charter.

Such groups generally have frequently changing application requirements and varied security requirements, and require a secure and reliable group communication system to ensure that they evolve and adapt according to interactions within the group. Being dynamic and self-organizing in nature, collaborative groups have special security requirements not addressed by typical access control mechanisms.

Peers with common interests and functionalities can join these groups subject to acceptance and authentication. Depending on their functionality and capability, peers could join at different hierarchial levels. While authentication is required to allow only authorized members to join the group, access control is needed for restricting rights at different levels and for different roles within a level. In the dynamic scenario where peers are constantly leaving and joining this becomes a difficult task. In the interests of the group there is a need for peers to collaboratively modify the access control policies dynamically. Traditional authentication and
access control are effective only in situations where the system knows in advance which users are going to access and what their access rights are. Another major challenge for collaborative P2P systems is the ability to manage risks involved in interacting and collaborating with previously unknown and possibly malicious peers. In such cases it becomes necessary to develop strategies for establishing trust among peers [10, 11, 15, 22].

Consider an example of a developers community for some critical security related open-source software. The communication forum could be either chat or e-mail. New users would be allowed into this forum to put in their suggestions, only if they are introduced by an existing member. Depending on the sensitivity level of the code being developed, there could be hierarchial levels for the members of this forum. New members and existing members could be periodically rated or evaluated by existing members based upon the quality of their contribution. A member receiving a high rating value could be elevated to a higher level in the forum. A peer or a member in any level need not always be evaluating other peers, i.e a member can have different roles in a group. For critical software like security applications, authentication of developing members would be important as well as deciding which member is allowed to play which role and participate in which level.

1.1 Challenges of Collaborative Peer Groups

Collaborative groups are heterogeneous, dynamic, decentralized and large-scale, with large number of autonomous entities wishing to access and share resources in a secure and controlled fashion. For a group to survive it must adapt to the environment and to the behavior of the peers. The major challenges for collaborative P2P groups are given below.

1. One main challenge of P2P groups is Admission control [1, 2]. Most prior work on peer group security has focussed on key management and authentication. A group membership control mechanism must guarantee that the group members are approved to join the group by satisfying the admission criteria. Admission criteria are group specific and form an integral part of the group security policy. A prospective member must learn these rules, hence they need to be specified in a readily available document, referred to as the Group Charter. In addition to group charter, a well defined procedure to admit a new member to a group is needed. In a static group this can be achieved by a pre-defined access control list ACL. However in dynamic groups this would not work.
2. In collaborative groups the authorisation policies (i.e., the rules governing who can access which resources) can be extremely complex, and may partially rely on and interact with policies of other users, and they can frequently change. Access control therefore cannot be solely based on identification and authentication of individuals [4, 37]. In the interests of the group and for its survival there is a need for peers to collaboratively modify group policies dynamically and the system should be able to evolve and adapt by switching between different policies. As groups evolve, there is a need for policies also to evolve and the group must adapt to these evolving policies for optimum performance. These changes should be within the frame-work of a well defined group charter.

3. Collaborative groups tend to organize themselves into different system models. A system model defines the behavior of peers, different roles, events, and policies for membership control, access control, task assignment, security and others. While evolving, peers have to decide which policies to chose and which system model to follow. For a collaborative group like Wikipedia for example, the group could start with an initial system model defining three roles say readers, editors and administrators. As the group evolves the admin peers may feel the need to introduce roles like proof readers or translators, which means that the group needs to change its system model. The system model or process workflow of the group sometimes needs to undergo a change for optimum performance. Thus the group should be able to re-organize its structure dynamically by adding or deleting roles, and/or changing the process workflow.

4. Another major challenge for collaborative P2P systems is the ability to manage risks involved in interacting and collaborating with previously unknown and possibly malicious peers. Some peers might be buggy and cannot provide services as they advertise. Some might be malicious by providing bad services to get more benefit. Since there are no centralized nodes to serve as an authority to supervise peer behavior and punish peers that behave badly, malicious peers can get away with their bad behaviors. How to distinguish potential benevolent peers from potential buggy or malicious peers is a current area of research. The way to resolve this problem is by building trust and reputation [10, 30, 24, 73]. However the trust context could vary from transaction to transaction and from communities to communities. It is important to build a reputation based system that is able to adapt to different configurations and different situations. In order to enable prac-
tical information sharing in such decentralized and dynamic systems, a viable trust model needs to be incorporated that will allow peers to have varying amounts of dynamically changeable trust amongst each other [14]. The main challenges that need to be addressed are: how to describe if a peer is trustworthy, what low-cost verification algorithm can be executed by a peer to determine the trust value of some other peer, how are trust values about peers exchanged within the system, how can dishonest peers be punished.

5. Another major problem is resource management and scheduling problem.

- How to efficiently motivate peers to provide satisfactory services to users
- How to guide users to select reliable resources in the presence of malicious peers.

### 1.2 Motivating Applications

Some of the peer to peer applications that face above mentioned challenges are listed below.

**F/OSS application**

One motivating example is the Free and Open Source Software self-organizing P2P group F/OSS[64] which represents an approach for communities of like-minded participants to create, distribute, acquire and use software and software-based services that are intended to be shared freely, rather than offered as closed commercial products. Participants within these communities often participate in different roles like core developer, module owner, code contributor, code repository administrator, reviewer or end-user. They contribute software content like programs, artifacts, and code reviews to Web sites within each community, and communicate information about their content updates via online discussion forums, threaded email messages, and newsgroup postings.

Active developers in such communities join and propose patches to existing code. The system gives opportunities to developers to make contributions and enable them to adjust their actions based on evolving code. Projects are evaluated by moderators on the basis of technical merit and elegance of contribution. When a developer with a high reputation develops a new application or a thread of development, others are attracted resulting in more threads. Such systems and communities co-evolve with the contribution of a large number of participants [65].

4
Depending on the group composition and the current skill set and average trust value of peers, such groups might warrant a change in policy for secure admission control or access control. A group which had initially permitted only three roles might perform better after some time if there was an additional reviewer role or if intermediate code were submitted to a reviewer instead of complete code. Thus peers need to be able to collaboratively frame and revise the policies for group management.

**Open health Environment OHE**

Zhengyi et al [99] introduce the concept of an open health environment OHE, where multiple healthcare related entities like patients, doctors, hospitals and insurers need to interact, but do not necessarily have prior experience with each other. Environment is "open" as there is no central authority which governs who can interact with whom and participants are free to create, modify and disband interactions and groups. An unknown entity may need to join the group. In this situation, entities must make use of some official registrations and endorsements like government issued credentials for establishing identity or employer created credentials attesting to a particular status in the organization. However the exact policies regarding what credentials are required for what purpose can be specified by the participants themselves.

An effective OHE should enable secure collaboration mechanism that permits on demand formation of collaboration groups, the ability for qualified strangers to join a collaboration group, the ability to operate in a totally distributed setting without a central administration, and guarantees of privacy and security control by the users of the collaborative systems [100]. In a specific OHE application, a collaborative group may require that some services like public file sharing should be open to all, while others like sensitive file sharing should be open to only a qualified subset of users. Similar sharing needs might involve exchanges of data between health care institutions and government agencies or between clinicians, pharmacies and pharmaceutical manufactures. Interested users wishing to join the collaboration may request for a specific role. Thus a new aspirant may send credentials which are verified by existing group member in the role of a recruiter, and the applicant could be issued a token or a group membership certificate. Trust negotiation is needed to build a secure P2P content search framework.

Such groups need a dynamic access control policy framework and an adaptive trust negotiation framework.
**Multi Player Gaming:**

Massively Multiplayer Online Games (MMOGs) are computer games capable of supporting thousands of players simultaneously and are typically played in a giant persistent world. Main aspects of game-play involve grouping with fellow players to perform certain tasks e.g killing monsters or attacking the enemy etc. Virtually most multi-player gaming today relies on a server-based paradigm. Players connect to one or two of several geographically dispersed server clusters and communicate with the "persistent world" through the server. There are a number of issues surrounding this server-based approach, the most obvious being the dependency on a server [57].

To be able to handle so many players simultaneously huge amounts of computing power and bandwidth are required at the server side making MMOGs cost intensive. One of the major problems with server-based approach for multi-player gaming is that games become obsolete the moment a server fails. The use of peer-to-peer approach could improve the multi-player gaming situation [101].

Peer to peer approach in the multi-player gaming scenario would warrant the need for secure admission and access control. A new player wishing to join an existing group should not be allowed to join if he is at a rating level lower than that of the players in the group. If the group consists of players playing at multiple levels then the rating of an existing player should increase based on his performance and number of wins. Thus a player who is admitted into the group at a lower hierarchial level could later be elevated because of incremental trust thrust upon him by group fellow members because of his performance. If on the other hand a peer cheats in a game he may be deprived of his ability to join another game, by revocation of certificate and he may be forcibly ejected from the group. Existing players in the group may sometimes feel the need to change the point scoring strategy, depending on the performance of players already in the group. This would warrant change in policies for elevating from one level to another level.

**Audio/Video conference collaboration**

Collaborative applications need to allow for the building of trust between individuals through interactions within the collaboration environment. In an application like a Virtual meeting or an Online Board Discussion, you could be at a conference listening to the presentation of a speaker,
which is based on a topic which is a common interest problem that you and your collaborators are dealing with. Ideally you should be able to bring this speaker into your other collaboration on the spot without the others voting for his admission or verifying his credentials. This calls for a partial access to the group, where the new member is treated as a guest and can only listen onto the conversation in a controlled fashion. This scenario is currently not supported by prevalent group communication systems.

In a Virtual classroom scenario (e.g. online university), geographically dispersed students can attend lectures in real time, interactively. The different roles could be Professor, student, T.A, Administrator, course designer and so on. Here authentication of users would be necessary so that peers can determine who they are interacting with. At the same time authorization mechanisms are needed to limit the groups of users who can participate in online meetings, as well as restrict the access rights of students.

**Multi-Chat Application**

Consider a multi-chat application in an educational institute implemented using a group communication system. Multiple sessions could exist, each implemented as a group. Each session could have different classes of members like students, professors, Teaching assistants and so on. For joining as a student various rules may apply:

1. Only students from a particular department may join.
2. Only students from a particular hostel may join.
3. Only students connecting from specific IP addresses may join.

Several kinds of communication may be going on simultaneously in one chat session, or different sessions and they should be subjected to different access control rules. For example

1. A new professor joins an existing chat session between some n professors. He may be allowed to talk on subject Y to all these professors but should not get access to subject X which was being discussed.
2. A high priority member say the Director logs in. He may have access to all discussions.
3. There may be private communication between students and professors.
4. A professor may be allowed to simultaneously participate in a group discussion among professors and in a Question/Answer session with students.

5. Some professors (seniors and juniors) are engaged in a discussion. The Director joins in and wishes to discuss something only with the heads of departments. The others may have to be ejected from the group.

Thus in a multi-chat application, users may want to chat one-to-one in a secure manner as well as one-to-many. The access rights of users in such an application may vary, i.e. some user may be able to receive messages from all other users while some may receive messages only from restricted people in the group. Depending on the situation and the access rights may have to be dynamically changed. There may even be a case, like in a military setting where some user can only receive messages but not send any.

**Collaborative news-groups**

Consider an online self-organising newsgroup, where authorised users can post articles. Different roles assigned maybe editors, reviewers, moderators, admin etc. Articles may be rated by reviewers and an editor may get updated to the level of a reviewer or a moderator with sufficient number of good postings.

These type of decentralized and dynamic peer groups require a secure group admission policy and an adaptive access control mechanism where peers can collaboratively frame and revise access control decisions, based on behavioral attributes, current group composition as well as trust evolved in the group.

**1.3 Problem Statement and Solution Approach**

To address the challenges of collaborative peer groups discussed in section 1.1, our goal was to design an integrated system with a dynamic policy based access control model based on an adaptive trust mechanism. While it is necessary for peers in a group to be able to dynamically modify the admission and access control policies based on evolving trust parameters, the changes must however be within the groups’ constitution charter, so that loosening of access control is not permitted to such an extent that the group performance degrades. Further malicious peers should not be allowed to change the group’s policies.
Traditionally, policy-based management models[41] have considered that roles, their membership and the policies assigned to them can be determined statically in advance by the administrator. However such models are not suitable for distributed and collaborative peer groups as they require anticipation of potential behaviors, events and their likely outcomes.

In this thesis we have designed and implemented an integrated framework for collaborative groups for authentication, admission control and access control. The framework has a novel method for policy based management of groups for membership control and access control where it is possible to deploy access control policies flexibly and dynamically. The initial group policies are defined in a Group Charter, normally by the creator of the group. Based on the behavior and trust level of peers in the group and the current group composition, it is possible for peers to collaboratively modify policies governing their level.

One possible approach to handle the dynamic and unreliable behavior of peers is to provide an environment for different functional roles which are self-organizing. There can be different roles within a group and peers at each role can have different functionalities. We define two types of roles viz. Group Roles and Domain Specific Roles which differ in the context of a peer group. Peers can be allowed to dynamically make decisions based on specific conditions and assume additional functionality to ensure that the group reaches a certain satisfaction level. Peers can dynamically switch between roles. Peers can collaboratively modify policies based on current group composition and trust level of existing peers. New policies can be added dynamically by peers within the framework of the original group charter. The group policy can also dynamically prioritize requests for join. Low level join requests can be postponed in a group which has a certain threshold of peers already at the same level, by changing the join policy dynamically. Peers in one level are also allowed to be dynamically updated to a higher level. Update policy also depends on current group composition and behavior. A peer could also be collaboratively ejected from a higher level to a lower level or even out of the group if his behavior in the group degrades. It is also possible for peers in the highest level to introduce a new level in the group if the situation so demands. Multiple policies can apply to a group, and the system can switch between policies over time, influenced by the system’s state or environment. Some policies can have higher priorities than others. Thus the peers are able to self regulate and self organize, based on trust and group composition.

The diversity of emerging applications with differing security requirements has led to the development of policy languages supporting a wide variety of policy-related constructs, e.g.
role hierarchies, delegation, appointment, or separation of duties. For the peers in a group to be able to dynamically modify the admission and access control policies, a declarative language offers more advantages. Our policy language is thus a declarative language based on Prolog, where we can model dynamic policies depending on changing trust level using assert and retract clauses. Some work has been done to address the need for dynamic policy adaptation such as [69, 70, 68]. Previous research work has focused on changing group policies dynamically within the same system model. We feel this is a limitation as the group should be able to reorganize into a structure based on a different system model if need arises. This is one main focus of our work, where groups can dynamically switch between different system models to optimize performance. We have been able to achieve this using statecharts.

StateCharts [86, 87, 88] have been proposed as high end specification techniques to represent complex systems for discrete event modeling. They were originally designed to represent and simulate real time systems [85]. They have interesting features to describe complex systems, such as notions of hierarchy and parallelism. We were able to adapt statecharts to express the dynamic behavior of collaborative groups. We were able to integrate the dynamism of Prolog with the adaptive behavior of collaborative groups using statecharts. Each composite state in the state chart depicts the group in a different system model. Within a composite state governed by a system model, the peers optimize group performance by dynamically switching between different policies and thus different sub-states. The group evolves by transitioning from one composite state having a system model $SM_1$ to another composite state with system model $SM_2$ which could have a different set of policies, roles, levels and so on.

We define an abstract function Group Health which is domain specific and depends on the achievement of Group goals. Peers monitor the group health by evaluating different parameters like average trust value of peers, rate and efficiency of task completion, rate of join, rate of leave, rate of task allocation, group composition and so on. These parameters cannot be known apriori and have to be periodically measured by peers as the group evolves. When the performance of the group falls beneath a certain threshold, the framework allows the group to dynamically change its entire behavior and adapt to a different system model, which may have different number of roles, levels and different policies, or even a different trust model. Thus the framework is adaptive and supports re-organization of the system model itself. In this new model again the peers can self-regulate their behavior based on an adaptive trust mechanism and the system can dynamically switch between different policies. The reorganization depends on
the qualities of peers joining the group and their behavior within the group. This is controlled
to some extent by policies for admission and access control. Thus we are able to achieve both
dynamic policy adaptation as well as system model adaptation, enabling the group to evolve
and adapt to evolving policies. We show that this dynamic adaptation and switching between
different system models and different policies helps to optimize group performance.

We propose an adaptive multi-dimensional trust model with a hierarchy of privileges
which builds upon the Peertrust [7] model. Our trust metric combines context specific attributes,
self-proclaimed ratings and recommendations received from other peers. Parameters of the trust
metric can be dynamically adjusted. We incorporate various attributes for feedback to add mul-
tiple dimensions for peer selection, into the basic trust metric with a weight function. We assign
weights with different attributes depending on the type of application. We include a decay
factor as trustworthiness is not always the same and may change with time. We incorporate
different methods to calculate the credibility of the peer who is giving feedback, based on the
situation. We propose a dynamic trust policy which dynamically assigns weights to different
trust attributes. Our trust model along with our dynamic trust policy helps us to slowly remove
malicious peers whose behavior is dependent upon a specific pattern. We compare the results
of our model with the P2P Trust model [6] which calculates trustworthiness of a peer using di-
rect reputation, and indirect recommendation. We show that our model handles malicious peers
more efficiently and their effect is mitigated and they are gradually ejected from the group.

1.4 Contributions of the Thesis

We have designed an integrated framework [26] for collaborative groups which has a novel
policy based method for membership control, access control and trust management. The main
thrust of our work is that collaborative peer groups cannot give optimum performance by adapt-
ing dynamic policies within a system model fixed apriori. The novelty of our work is that the
framework proposed allows a collaborative peer system to modify its system model and process
workflow and adapt between different system models to achieve optimum performance. The
framework is flexible and peers are able to self regulate their behavior in presence of dynamic
policies. We are able to express dynamic policies and varying system models in our framework.
Main features of our framework are:

- A dynamic policy driven trust based access control model [83]. Dynamic policies are
written using a declarative language (Prolog), so that they can be easily modified without changing the code. We have also designed a GUI based converter for converting text based input to Prolog format to assist the user for writing the policies.

- Ability to represent varying system models for collaborative groups.

- Framework supports dynamic adaptation of policies and re-organization of group structure by switching between system models. We have designed a tool capable of achieving this dynamic policy adaptation and optimization by dynamically changing the group system model itself. Our tool allows us to test the evolution of peer groups based on different policies. We give a statechart model to capture the dynamic adaptation of varying system models and policies.

- Our experiments show that dynamic policies based on the adaptive trust and changing group composition lead to better group efficiency as compared to static access control policies. Further the dynamic adaptation of system models and policies help to optimize group performance. Users are able to evaluate the effect of different dynamic policies on groups and adapt to the right set of policies for evolution of the group.

- We give an adaptive multi-dimensional trust model for P2P applications [84]. Experiments show that the model is able to detect malicious peers and mitigate their effect on group performance.

### 1.5 Dissertation Outline

The remainder of the dissertation is structured as follows.

In Chapter 2 we give an overview of access control. We then review research related to access control in collaborative groups, followed by related work in policy management and trust management in dynamic peer groups.

In Chapter 3 we detail a generic application framework for management of collaborative groups. We give the details of the architecture of the proposed framework for collaborative groups and the protocols for membership control. We define a formal system model for collaborative groups and represent it using statecharts.

In Chapter 4 we explain our dynamic policy driven trust based access control model. We show
how different policies can be expressed using a declarative language (Prolog) which supports dynamic adaptation of policies.

In Chapter 5 we model a sample application FOSS. We give the statechart model for the same. The framework is simulated using a discrete event simulator Peersim integrated with Prolog. Experiments show that our tool permits the group behavior to be optimised by being able to dynamically switch between different system models with varying dynamic policies in each model.

In Chapter 6 we give a comparative evaluation of different policy frameworks with our framework. In Chapter 7 we give details of our proposed adaptive multi-dimensional trust model for P2P applications. We model malicious peers in this model and our trust policy helps us to detect malicious peers and gradually eject them from the group.

We give details of experiments conducted for the adaptive trust model for an e-commerce application and a comparison with other trust models shows that our model performs better when the percentage of malicious peers increases.

We give the scope for future work and conclude in Chapter 8.
Chapter 2

Related Work

Access control models in collaborative groups are used to decide on the ways in which the availability of resources in a system are managed. There can be two types of scenarios, Static Access Control and Dynamic Access Control. In former case information about all group members is known in advance. So management of access policies is easy and can be done using access control lists (ACLs). However in a dynamic group the peers can join and leave at any time. It should be possible to therefore to specify and change policies at run time depending on the environment or collaboration dynamics. Traditional access control models like DAC, MAC and RBAC are not suitable for collaborative environments whose entities vary dynamically. In such environments the access control system must be based on user’s trust which continuously evolves.

In this chapter, we first give some background information on access control, explain why mandatory, discretionary and identity based access control models are insufficient for collaborative groups and how this led to the development of role-based access control. We discuss different frameworks proposed for access control in decentralized groups. Next we discuss related work in policy based access control. This is followed by a review of major work in the field of trust and reputation based access control for dynamic peer groups.

2.1 Access Control

An access control framework regulates the access to functionalities (be it web services, processes or companies) by authorized users depending on an access control policy. In its simplest form, this relation is represented by a list associating authorized users to resources. When a user...
provides sufficient proof for his identity, an authorization decision can be achieved with respect to user resource relationship. Access control is defined through the notions of users, subjects, objects and actions.

2.1.1 An Overview of Early Access Control Models

The first access control models can be divided into discretionary and mandatory access control. The discretionary access control (DAC)[35] is used to control access by restricting a subject’s access to an object. It is generally used to limit a user’s access to a file. In this type of access control it is the owner of the file who controls other user’s accesses to the file. The mandatory access control (MAC)[36] assigns security labels or classifications to system resources and allows access only to entities (users, processes, devices) with distinct levels of authorization or clearance. These controls are enforced by the operating system or security kernel. Policy enforcement is then based on two principles that force information flow to be unidirectional:

- The no read-up principle states that a subject has read access to an object only if the clearance of the subject is greater than or equal to the classification of the object.

- The no write-down principle allows a subject to write to an object only if the subject’s clearance is less than or equal to the classification of the object.

MAC offers a safe structure for a highly centralized access control system that regulates access to multi-level objects by authorized subjects depending on their clearance level. Ideally, the objects are labeled with security labels ranging from Top secret for the most sensitive to public or unclassified for the least sensitive, and the subjects are divided into different clearance levels. With respect to the security level of a subject and the security level of an object, the Bell-LaPadula [33] model defines access control decisions in accordance with simple security property and star property. The star property supports the MAC policy indirectly, by preventing high information from ending up in a low container where a low user could read it. The Biba integrity model [34] was introduced in 1977 as an adjunct to the Bell-LaPadula model, and it addresses integrity while sacrificing confidentiality. Under Biba, read and write restrictions are based on integrity levels assigned to subjects and objects. Existing access control models are well suited for the centralized and static environment, where the subjects, objects and resources are static, and the permission that is granted for a subject to access a object is relatively change-
less. In these access control models, the subjects are client processes or users, and the objects are usually server resources.

### 2.1.2 Identity Based Access Control (IBAC)

This employs traditional access control schemes such as access control lists or ACLs. Authorization decisions are based on the identity of the requestor. Identity based credentials can be further classified into encryption based and non-encryption based. An encryption based identity credential is a piece of information produced and cryptographically signed using the key possessed by the supplicant in order to verify its possession of the key, and hence prove its identity.

One form of non-encryption based identity credential is information that is hashed using a one-way key-based hash function and the key possessed by the supplicant. In order to verify the supplicant’s identity, the authenticator must possess the same key (symmetric key) and the hashed information as the supplicant in order to re-generate the hash value and verify the claimed identity of the supplicant. However in de-centralized environments, the resource owner and requestor are often unknown to each other, making access control based on identity very ineffective or expensive to maintain.

Traditional access control models are not obviously suitable for the collaborative environments whose entities vary dynamically, such as Grid and P2P. Both MAC and DAC models are not sufficient for large commercial organizations. MAC is clearly too rigid, and DAC is hard to administer. In most organizations, users act in the capacity of a role or a job function, and access control decisions are determined by the responsibilities and privileges associated with a role. This observation has led to the development of role-based access control (RBAC).

### 2.1.3 Role Based Access Control (RBAC)

Role-based access control [31, 32] is an alternative to discretionary and mandatory access control. With RBAC, security is managed at a level that corresponds closely to the organization’s structure. Each user is assigned one or more roles, and each role is assigned one or more privileges that are permitted to users in that role. Security administration with RBAC consists of determining the operations that must be executed by persons in particular jobs, and assigning employees to the proper roles. A RBAC system has two phases in assigning a privilege to a user,
first the user is assigned roles and then the roles are checked against the requested operation i.e users are mapped to roles and roles are mapped to permissions. When a new member wishes to join the group he/she will get a list of available roles that the application (group) supports. The new member will request for a role (may be highest available) passing his identity to other peers. If new member’s admission request is rejected, he can attempt to join in a lower role. Access rights are grouped by role name, and the use of resources is restricted to individuals authorized to have associated role. Thus permissions are associated with roles not with individual peers, refer figure 2.1

![Figure 2.1: Role based access control](image)

2.1.4 Authenticated Role Based Access Control (ARBAC)

Systems that require a stronger authentication [45] can replace the login process of username, password with a public key certificate that is used to identify the accessed user. Certificates contain a public key, and properties of the owner of the corresponding secret key; in this case, the relevant property is the identity of the owner.

A trusted Certification Authority (CA) digitally signs each certificate, binding the attributes with the owner- specifically, providing the identity of the owner of the private key. The login process is replaced by an authentication protocol such as SSL, which verifies that the user has the secret key. This is the mechanism used, e.g., by certificate-based authentication by current browsers and servers and is based on the assumption that the users are known to the system in advance.
2.1.5 Attribute Based Access Control (ABAC)

In this case the access permission are determined based on authenticated attributes of user. Attributes are normally a function of the user’s identity and its characteristic. So a policy rule which decides whether a peer p can access some resource r is a function of peer and resource attributes. A rule will therefore look like:

\[
\text{RuleX} : \text{can\_access}(s, r) \leftarrow f(\text{ATTR}(s), \text{ATTR}(r)) \quad (2.1)
\]

In case of RBAC, fine grained access control polices often involve multiple subject and attributes. As the number of attributes involved increases, number of roles and permission needed to encode these attribute increases exponentially. RBAC might need some centralized server to manage user to role and role to permission assignment. Peer groups share the properties of peer-to-peer overlay networks, including full de-centralization, symmetric abilities and dynamism, making security problems more complicated.

2.2 Securing Group Communication Systems

Several efforts have been made towards securing group communication systems, for e.g Secure Spread [80], Antigone project [79] and Secure group layer SGL [81]. Secure Spread [80] is a secure group communication system that uses a fully distributed group key generation protocol, but it does not provide any authentication or group access control mechanisms and focusses primarily on LAN and interconnected LAN environments.

Antigone project includes a flexible framework for secure group communication and utilizes a centralized member admission approach which is controlled by a session leader \(SL\) who interacts with an on-line trusted third party (TTP) in order to admit a new member. The TTP shares a symmetric key both with the SL and every potential new member. However Antigone is not designed for peer-to-peer networks.

Secure group layer SGL, is a secure group communication system aimed at WAN environments. It bundles a reliable group communication system, a group authorization and access control mechanism and a group key agreement protocol to provide a comprehensive and practical secure group communication platform. However the access control mechanism is not dynamic or scalable.
2.2.1 Access Control in Collaborative Groups

A lot of research has been done to provide access control to shared resources in distributed environments. CAS [40] uses IETF standard X.509 proxy certificates to delegate rights from a central server to a user. Akenti [94] and VOMS [38] use attribute certificates to assign attributes to users in a verifiable way. In Akenti the policy is held in certificates, and its decision engine can gather and verify the policy and attribute information. However allowing the user to access resources requires the user to have an X.509 certificate from a trusted CA. Prima [39] introduces the idea of privilege management, hence the rights that a user has to a resource is stored in a verifiable certificate. However each of these systems depend on central servers to provide the authorized access information and depend on a few specified CA’s to provide the X.509 public key certificate. Berket et al.[89] use the Akenti mechanism to realize the fine grained access control which relies on some centralized knowledge. In [90] they use both X.509 certificates and pseudo certificates which are self-signed certificates provided to users the first time they use the system. They give a flexible architecture for authentication.

In a collaborative system, the members and resources are dynamic, and the members might join or exit the system at any moment, and might be on line or off line. Admission control is therefore an important feature of such systems. One way is to appoint a trusted Group Authority GAuth to handle admission procedures. This however goes against the principles of decentralization and limits scalability. Kim et al. [1] proposed an admission control framework suitable for different kinds of peer groups and suggested some realizations using various cryptographic techniques. Their framework relies on two basic elements: group charter and group authority. Group charter is a document which contains information about the group including the admission rules. Group authority is the entity which decides about group membership requests. The framework includes various admission policy types depending on the structure of the group authority: (i) public access control lists (ACLs) which can be used without any group authority; (ii) admission decision made by a centralized group authority (e.g. a group manager or a trusted third party), and (iii) admission decision made collectively by current group members. Obviously, public ACLs are unsuitable for dynamic peer-to-peer groups because all prospective group members should be known in advance which is an unrealistic assumption. The centralized admission decision made by a group manager or a trusted third party violates the trust relationship in such groups where no such trusted parties are available. Hence, the admission policy based on collective decisions is surely the most suitable form for collaborative
peer-to-peer groups. The framework uses the notion of group membership certificates (GMCs) that allow members to prove own group membership. Every group member obtains own GMC by the end of the admission protocol. However this scheme lacks the attributes of peers and cannot simplify authorization in collaborative environments.

Saxena et. al. [2] proposed an abstract model for the access control protocol based on the framework from [1] and provided three concrete realizations. Here the difference is that upon collecting enough approval votes prospective user computes his own group membership certificate GMC without further interaction.

Another approach is to let group members handle the admission procedures themselves. So any subset of at least t members can jointly decide to admit a new member, via a voting protocol which is based on (t,n) threshold signature scheme where n\geq t is the current group size. In [3] a distributed CA model is used where the CA private key is distributed among a specific subset of peers without complete replication. Each peer knows only part of the key and collaboration is required to retrieve it. The distribution is done using Shamir’s secret sharing scheme. In a (t,n) threshold scheme, the secret is distributed to n peers allowing to compute the secret with the data of any t peers, but obtaining no information about the secret in case t-1 collude. The challenges are determining the size of t in a dynamic environment. Besides, traditional secret sharing has its disadvantages as the secret shares which have been distributed remain unchanged, posing a threat to backward secrecy. Further malicious secret share holders may collude.

In [5] the authors propose a method for access control in peer groups using a web of trust. Their method has the ability to adapt to a broad range of group policies and group membership scenarios.

Yu Zhang et al in their work [4] propose a fine-grained attribute based access control framework for peer-to-peer systems. They extend role-based trust management language, RT [104]to satisfy security requirements of peer groups. They define two kinds of roles: group role and application role. Group role peers are predefined by peer groups and application role peers are defined according to different collaborations. They propose a distributed delegation authorization mechanism where multiple authorities could exist to grant peer group membership. Their access control policy defines the relations of roles and permissions including six credentials from RT and describes admission and removal policies of roles. Trust management is used as the authentication method. Joint authorization is done by voting schemes as in the
framework proposed by Kim et al [1]. The actual model consists of the following steps:

- **Group Initialization**: The group authority peer, creates a secure peer group and advertises it on the network. The advertisement contains the access policy of peer groups along with other parameters like group name, voting type etc.

- **Join Request**: A new peer wishing to join a group, searches for the advertisement, provides his related credential to an authority peer along with a request for issuance of a peer group certificate with its desired privileges.

- **Authorization of Request**: The authority peer first verifies the signature and then propagates the request to call a vote. Multiple peers authenticate the requestor’s attributes and reply with a signed message to approve or reject the authorization request.

- **Certificate Issuance**: Once enough votes are collected the Group Authority verifies the votes and decides whether to accept the new node as a member. If the requestor is qualified, the GA peer issues a peer group certificate to it and updates related peer group information.

The approach from Sandhu et al. [92] use trusted computing technology to enforce the privileges on the client. In that way, not only the access to the data is protected but also the use of the data can be restricted. In contrast, Crispo et al. [95] use eXtensible Access Control Markup Language (XACML) policies [96] to specify the access restrictions on the data offered by the peers. These policies are evaluated and stored at supernodes. This restricts the use of the approach to hierarchical unstructured P2P networks. A similar approach by da Silva et al.[93] also uses XACML policies. Tran et al. [15] relate the privileges of the participants to their individual trust values. However, none of these approaches considers existing access control components and information residing on the peers. Moreover, administration distribution is not included in these approaches.

Christina et. al. in [37] propose a fine-grained access control framework for group communication systems where they allow an application to define specific policies while the enforcement is performed in an efficient manner by the GCS. Their focus is on group communication systems built around a client-server architecture. The basic approach they follow is that for a group there is a set of basic operations that can be performed, by principals (entities) based on their role in a given context. Their framework combines role based access control mechanisms
with environment parameters such as time, IP address etc, to provide policy support for a wide range of applications with different requirements. They identify the set of all possible group operations that can be controlled and define the group policy as a mapping between roles and operations using context as constraints. Thus applications defining specific policies are translated to this set of basic operations that the GCS is aware of and can enforce access control on.

2.2.2 Policy Based Management in Collaborative Groups

Policy is a set of rules for directing and determining how to manage devices and allocate resources. These rules composed of a set of Condition and Action, describe that the actions should be adopted in special conditions.

Traditionally, security policies are classified as access control, information flow, and availability policies depending on the type of behavior they control. Different types of properties include safety properties which assert that something bad never happens, liveness properties that assert that something good eventually happens, and fairness properties that assert that everybody gets a chance to use a resource [69]. Access control policies can be specified as safety properties, and can be specified using simple temporal logic operators. A dynamic policy is a program consisting of a set of guards and actions, created by a policy administrator. It encodes not only the logic to modify the system implementation to change operational parameters, but also includes all the necessary guards to enforce good behavior and prevent its misuse.

Enforcement of dynamic policy management in collaborative groups implies that peers are not assigned policy to perform operations on an individual basis, but rather policies are associated with roles, of which, peers become members. As the group evolves, role association with new and modified operations can be established as well as redundant operations deleted as organizational functions change. This fundamental concept has the advantage of simplifying the management of group-level policy without the need to modify the underlying access infrastructure.

There exist many architectures that have developed policy-based approaches to the management of open distributed systems. Role-Based Access Control (RBAC)[37] brought forward the notions of role and policy to manage database security. RBAC however, provides limited static policy definition and analysis and limited conflict detection and resolution mechanisms. There is no support for heterogenous distributed domains, compound actions or trust.
The policy-based management architecture proposed in [66] provides a clear policy notation, however policies within this model are necessarily a static concept. Supporting open distributed groups relies on defining and analyzing policies at run-time, so users are able to interact with new resources for which they have no established policy on access.

Naldurg et al. [69] have described a framework for constructing dynamic policies. Through policy development lifecycle, the model uses suitable formal notations and methods for the specification, verification and validation of these policies. Quinn et al. [70] have explored the creation and use of dynamic policies that had trust conditions embedded through the introduction of trust meta-policies that expose the trust conditions explicitly. Walker et al. [58] suggested an adaptive policy approach that allows policymaker to cope with the uncertainties that confront them by creating policies that respond to changes over time and that make explicit learning. The authors state that this approach implies fundamental changes in what they call the three major elements of policymaking: the analytical approach, the types of policies considered, and the decision-making process, and describe in detail their rationale.

Dunlop et al. [68] have introduced a dynamic policy model for the management of large, open evolving enterprises. The model includes novel concepts in support of enterprise including enterprise domain, policy space and policy authority that are helpful to support dynamic policy-based management and is able to cope with the high rate of change inherent in such environment. Toninelli et al. [77] present a semantic context-aware adaptive policy model that enables dynamic adaptation of policies depending on context changes by combining two design guidelines, i.e., context-awareness which allows operations on resources based on context visibility and semantic technologies which allow the high-level description and reasoning about context/policies.

Bahati and Baur [78] explored how reinforcement learning in adaptive policy driven autonomic management model can be adapted to accommodate changes such as "whether a model 'learned' from the use of one set of policies could be applied to another set of 'similar' policies or whether a new model must be learned from scratch as a result of changes to an active set of policies?" The authors have demonstrated the value of reinforcement learning in enabling an autonomic management system to improve its use of policies.
Policy based Languages

Distributed systems involve frequent ad hoc collaborations between entities with no pre-established trust relation, each with their own authorization policies. To deploy and maintain such applications, it is essential that all mundane authorization decisions be automated, according to some human readable policy that can be refined and updated, without the need to change (and re-validate) application code. To this end, several declarative authorization management systems have been proposed; they feature high-level languages dedicated to authorization policies; they aim at improving scalability, maintenance, and availability by separating policy-based access control decisions from their implementation mechanisms. We present here a brief survey of the features of some of these languages.

Cassandra[40, 42] is an authorization language that defines the actions of activating a role and deactivating a role. Users can thus write state-dependent and implicitly state manipulating policies, but this rather ad-hoc approach is inflexible. Some languages such as Ponder[107] support obligation policies. An obligation is a task to be executed after evaluating and enforcing an access request. However it does not provide a precise semantics for the state changes. Some work has been done on analyzing security properties in dynamic role-based systems, in the context of the role-based authorization language RT[104] and Administrative RBAC (ARBAC)[45], where members of administrative roles can modify the role membership and privilege assignments. In [46], policies written in Datalog can refer to facts in the authorization state. Hezberg et al. propose in [47], a prolog-based trust management language, but do not focus on dynamically changing policies with the state of the environment.

PolicyMaker is a general and powerful solution that allows the use of any programming language to encode the nature of the authority being granted as well as the entities to which it is being granted. Keynote is a derivation of PolicyMaker, and has been supported by IETF. Blaze, Feigenbaum and Lacy introduced in [51] the notion of Trust Management. In that original work they proposed the PolicyMaker scheme as a solution for trust management purposes. It addresses the authorization problem directly, without considering two different phases (one for authentication and another for access control). KeyNote (RFC 2704) was proposed and designed to improve two main aspects of PolicyMaker viz to achieve standardization and to facilitate integration into applications. Keynote uses a specific assertion language that is flexible enough to handle the security policies of different applications. Assertions delegate the authorization to perform operations to other principals. KeyNote considers two types of assertions
called policies and credentials.

- Policies. This type of assertions does not need to be signed because they are locally trusted. They do not contain the corresponding Issuer of PolicyMaker.

- Credentials. This type of assertions delegate authorization from the issuer of the credential, or Authorizer, to some subjects or Licensees if some Conditions hold. They have to be signed by the authorizer.

SDSI/SPKI is a unification of two similar proposals, SDSI (Simple Distributed Security Infrastructure) and SPKI (Simple Public Key Infrastructure). SPKI was proposed by the IETF working group and, in particular, by Carl Ellison [49]. SDSI [50] was an alternative design for a public-key infrastructure to X.509, designed by Ronald L. Rivest and Butler Lampson. The SPKI/SDSI certificate format is the result of the SPKI Working Group of the IETF (SPKI Certificate Theory, (RFC 2693). The main feature of SDSI/SPKI is that its design provides a simple public key infrastructure which uses linked local name spaces rather than a global, hierarchical one. All entities are considered analogous; hence, every principal can produce signed statements. The data format chosen for SPKI/SDSI is S-expression. This is a LISP-like parenthesized expression with the limitations that empty lists are not allowed and the first element in any S-expression must be a string, called the type of the expression. SDSI establishes four types of certificates: Name/Value, Membership, Autocert and Delegation. SPKI/SDSI unifies all types of SDSI certificates into one single type of structure. The SPKI/SDSI certificate contains at least an Issuer and a Subject, and it can contain validity conditions, authorization and delegation information. Therefore, there are three categories: ID (mapping name,key)), Attribute (mapping (authorization,name)), and Authorization (mapping (authorization,key)).

**Logic frameworks**

Logic programming offers a powerful mechanism to represent authorization and access control decisions [48, 52, 53]. In this context, authorizations are represented as predicates and decisions are based on formulae verification. There are many solutions for formulae verification but the most known is probably PROLOG [60], which has several implementations for different platforms (Windows, Linux, Macintosh,). It is very easy to express facts as well as complex relationships in Prolog. Prolog gets its power from logical tools like quantification and unification. Logic programming is well-suited for expressing trust management, because logic
provides a framework for defining attributes, and enables reasoning, in particular about chains of trust. After defining a set of statements or facts, an authorization request can be formulated and logic rules can be used to check whether the authorization request is a valid consequence of these facts.

**Secpal**

The Security Policy Assertion Language (SecPAL)[53] is a logic-based policy language developed by Microsoft that addresses authorization and trust in distributed scenarios that involve frequent ad-hoc collaborations between entities with no pre-established trust relationships. SecPAL has been designed to support advanced requirements such as:

- Establishing fine-grained trusts both within and across organizational boundaries
- Distributed policy authoring and composition with enforced separation of duties
- Constrained (and unconstrained) delegation of rights
- Fine grained revocation
- Cryptographically strong, Internet scale, authentication information
- Proof graphs showing the logical reasoning behind an authorization decisions.

The SecPAL formal model [55] describes the abstract types, logical expressions, language semantics and rules for evaluating authorization decisions in a deterministic and efficient manner. The fundamental SecPAL concept is the security assertion. An assertion is a statement made by a principal that may: define a binding between a principal and an attribute; specify a principals permissions to operate on a resource; express a trust or delegation policy; express an authorization policy; revoke a prior assertion; or declare principal identifier alias relationships. These assertions are expressed using a uniform grammar which maps directly both to simple English sentences (for human consumption) as well as an XML schema (for machine consumption). SecPAL can thus express a wide range of idioms, including policies with parameterized roles and role hierarchies, separation of duties and threshold constraints, expiration requirements, temporal and periodicity constraints, policies on structured resources such as file systems, and revocation.

However to the best of our knowledge none of the policy languages address the issues for expressing system model change in dynamic environments.
2.2.3 Trust Management

Trust management has been emerging as an essential complementary to security mechanisms in self-organizing and collaborative networks. Self-organizing networks are open, dynamic, and prone to a lot of security threats. Without the protection of security mechanisms, it is impossible to run crucial applications properly. In a trust management system, evidence about nodes, such as honest, selfish, and malicious behaviors, is collected to conduct trust evaluation (i.e., trust computation, trust combination) among nodes. Based on trust evaluation, decisions can be made in order to encourage the interactions between honest nodes, punish selfish nodes, and exclude malicious nodes. That is, evidence collection, trust evaluation, and decision making are three major components of a trust management system. Evidence can be one dimensional or multi-dimensional.

Distributed establishments typically involve peers that do not know each other and have never met before. This brings in the concept of risk when peers perform transactions without knowing the reputation of those whom they are interacting with. This has precipitated work on trust and reputation mechanisms in peer-to-peer networks [15, 13, 11, 10, 17]. Trust is a measure of how much a peer is willing to transact with another peer. It can be defined as a peer’s belief in attributes such as reliability, honesty and competence of the trusted peer, either based on the peer’s own experiences or based on recommendations by other peers. Reputation [22, 8, 7, 18, 24] is one specific way of establishing trust. It defines an expectation about a peer’s behavior, based on recommendations received from other peer’s or information about the peer’s past behavior within a specific context at a given time.

Trust is defined by Grandison and Sloman [72] as a peer’s belief in another peer’s attributes such as reliability, honesty and competence based on its own direct experiences. Trust can be broadly categorized by the relationships between the two involved peers. It measures whether a service provider can provide trustworthy services. Reputation [22] is one specific way of establishing trust. Reputation of a peer defines an expectation about its behavior, which is based on recommendations received from other peers or information about the peer’s past behavior within a specific context at a given time. It can be decentralized, computed independently by each peer after asking other peers for recommendations.

Reputation based Trust Management Systems derive the trust-worthiness of a peer from past history and behavior. Reputation is an evaluation of behavior done by third parties reporting the quality of past interactions. It provides a frame of reference that makes the environment
predictable. However existing systems do not provide adequate reaction to quick changes in peers’ behavior and are therefore not able to successfully cope with dynamic malicious peers.

**Requirement for Trust Dynamics**

Trust is conditionally transitive and can be multi-dimensional and depends on the context. Trust develops over time, based on experiences. Positive experiences can lead to increase of trust while negative experiences can lead to decrease of trust. Trust can also vary over time. A peer may behave with different trust degree in varying situations. Trust is multi-faceted in the same context. For example a customer might evaluate a restaurant from several aspects like quality of food, price, kind of service and so on. For each aspect he/she develops a trust. Thus dynamics of trust relates to evolution of trust over time and in the face of new experiences.

Main requirements of a dynamic trust metric as summarized by [19] are

- Trust metric must be sensitive to new experiences.
- Positive and negative changes of trust are not symmetric
- Sensitivity to new experiences should not depend upon total number of experiences
- While trust must be sensitive to new experiences the long term behavior must be taken into account.
- Trust metric should detect and penalize both sudden misbehavior and possibly long term oscillatory behavior.

Thus what is needed is a flexible trust metric.

Based upon the approach adopted to establish and evaluate trust relationships between peers, trust management has been classified into three categories [74]:

- **Credential and policy-based trust management** where peers use credential verification to establish a trust relationship with other peers. e.g Policy maker, Keynote.

- **Reputation-based trust management** where peers establish trust relationship with other peers and assign trust values to those relationships. Trust value assigned to a trust relationship is typically a function of the combination of the peer’s global reputation and the evaluating peer’s perception of that peer. e.g Eigen[12] trust model
• **Social network-based trust management** where social relationships between peers are taken into account when computing trust relationships. e.g. Regret, NodeRanking.

Role Based Trust Management (RT) proposed by Li [56] makes use of Roles to define a full framework composed by different languages, each of them with different characteristics. Roles can be interpreted as privileges or attributes. It is based on a subset of Prolog, Datalog, which is a language of facts and rules. Datalog is a logic based query language for the relational model that has been mainly used in the field of knowledge discovery but also in some other fields. One of the more attractive properties of DATALOG, regarding its tractability, is the absence of function symbols as arguments in the predicates. It is the main reason for DATALOG to have efficient procedures for answering queries.

### 2.2.4 Reputation Based Trust Management

In a P2P system every peer has a dual role of a client and server; hence they are popularly called as servent. The design requirements of a P2P system are as follows:

- **Anonymity.** Every peer-to-peer system should be able to remain anonymous.
- **Decentralized Authority.** Every peer should be self-managed or autonomous.
- **Robustness.** Every peer should be robust to unfriendly attacks and the network itself should not be susceptible to attacks by unfriendly peers.

Reputation systems provide a way for building trust through social control without trusted third parties. Most research on reputation-based trust utilizes information such as community-based feedbacks about past experiences of peers to help making recommendation and judgment on quality and reliability of the transactions. Community-based feedbacks are often simple aggregations of positive and negative feedbacks that peers have received for the transactions they have performed and cannot accurately capture the trustworthiness of peers. In addition, peers can misbehave in a number of ways, such as providing false feedbacks on other peers. The challenge of building a trust mechanism is how to effectively cope with such malicious behavior of peers. Another challenge is that trust context varies from transactions to transactions and from communities to communities. It is important to build a reputation based system that is able to adapt to different configurations and different situations.
Reputation based systems establish trust among members by using feedback from peers. The main challenges in building a reputation based trust system in decentralized groups are:

- How to cope with potential malicious peers
- How to cope with indirect trust given input of direct trust valuations
- How to continuously revise trustworthiness of a peer at runtime
- How can trust evaluations be done when peers have unequal roles in the group

Some of the reputation systems proposed are discussed here. A well-known reputation based system is the rating scheme used by eBay an on-line auction site [27]. Ebay has a reputation system where buyers and sellers rate each other after each transaction and the overall reputation is the sum of the ratings over last six months. However the system relies on a centralized system to store and manage the ratings. The Ebay system can be easily compromised because the seller can gain reputation very fast by fulfilling small orders and defaulting on a large order. Credence uses a voter correlation scheme that weighs peers opinions [23].

One of the earliest works in this area is the protocol by Aberer and Despotovic [20] which aims to identify dishonest peers by a complaint-based system. Their model relies on negative feedback or complaints to convey reputation information. Peers do not store information about successful interactions, only negative experiences are stored. Complaints are then forwarded to other peers. When a peer wants to evaluate the trustworthiness of a target peer, it first searches its own history to locate complaints registered by itself. It then queries other peers.

A shortcoming of this protocol is that it maintains only the negative feedbacks, providing no means for a trustworthy peer to be distinguished from a newcomer. The trust evaluation is also rather simplistic, classifying every peer either as trustworthy or untrustworthy. Moreover, maintenance of a P-Grid architecture is required on top of the existing P2P structure.

Eigen Trust [12] introduces the concept of global reputation where each peer collects reputation value of a peer from all other peers in the network and then this information is used to make a decision about whether a peer should be trusted or not. It aggregates the local trust assessments of all peers in the network in an efficient and distributed manner, but introduces significant computational overhead. Each time a peer $i$ has a transaction with peer $j$ say a file download, it may rate the transaction as positive or negative depending on the authenticity of the received file as $(tr(i,j)=1)$ or $(tr(i,j)=-1)$. The local trust value for a peer is then calculated
as \( s_{ij} = \sum tr_{ij} = (\text{sat}(i,j) - \text{unsat}(i,j)) \) where \( \text{sat}(i,j) \) is the number of satisfactory transactions and \( \text{unsat}(i,j) \) is the number of unsatisfactory transactions. The global reputation of each peer \( i \) is then given by local trust value assigned to peer \( i \) by other peers, weighted by global reputations of assigning peers. However the drawback of Eigen trust is that the trust value is normalized to lie between 0 and 1 and thus it does not differentiate between peers with whom peer \( i \) did not interact, or peers with whom peer \( i \) had a poor experience. Further it does not take into account user dynamics nor does it consider the effect of credibility.

Another proposal with a similar scope is the XREP protocol proposed by Damiani et al. [75], which assesses the trustability of a file to be downloaded by voting of the peers. They propose a system for peer-to-peer systems in which servents ask for a vote for a proposed resource. They select a subset of votes received and then verify that they originated from the original servent to prevent any shill voting. The problem with using this algorithm to determine the best resource to use is that the other peers or servents have no incentive to provide accurate information or even respond to a query on any given resource, especially ones that perform well in order to prevent overloading their favorite resource. The proposed system does evaluate how votes are cast and reputation is also built for other servents which gets reset if they were to change their ID to discourage pseudospoofing. They also propose P2Prep [76], where peers poll the network for reputation opinions on service providers and they describe how to make a Gnutella [29] servant reputation aware with a polling protocol. Basically, peers flood reputation queries throughout the network to learn about reputations of others.

Both the protocols make no distinction between the votes from trustworthy and non-trustworthy peers, and there is no authentication of the vote messages. Further, no quantitative trust metric is specified for choosing among alternative versions. Neither do the models account for credibility of information sharing peers in their trust calculations.

Kazaa [28] is a peer to peer file sharing system that provides a reputation system based on Integrity rating and participation level. The main problem with this system is that it does nothing to punish peers engaging in obnoxious behavior, although it rewards good behavior. A study with a different but relevant scope is a protocol by Xiong and Liu [7] on trust evaluation in P2P ecommerce communities. They present a reputation-based trust framework PeerTrust to qualify the trustworthiness of peers [8]. They state that a trust framework that is merely based on previous transaction feedback is insufficient and inaccurate. PeerTrust includes five important factors to evaluate the trustworthiness for each peer: (1) the feedback obtained from
other peers; (2) the feedback scope; (3) the credibility factor for the feedback source; (4) the transaction context factor; (5) the community context factor. In this way, PeerTrust can evaluate peers from multiple facets.

Lee, et. al. propose NICE [16] which is a solution that can identify cooperative and non-cooperative users and creating cooperative groups based on these identifications and applying it as a distributed system. The authors were able to achieve their goals by creating an algorithm that selects a trust path based on whether it is the strongest path or using a weighted sum of strongest disjoint paths.

Global history schemes are complicated, requiring long periods of time to collect statistics and compute a global rating.

Marti and Garcia [25] propose a limited reputation sharing scheme to reduce the number of failed interactions between peers. They study the performance of a peer-to-peer resource-sharing network in the presence of malicious nodes, which, in contrast to global history schemes, uses only limited or no information sharing between nodes.

The authors in [11] propose a decentralized access control system for ad hoc collaborative environments where user permissions are determined and assigned based on a reputation management process which does behavior grading of a user and provides a reputation index that nodes use to determine trustworthiness of their peers. Each resource owner uses the combined characteristics of identity and reputation to make access decisions. The trust management system (TMS) track’s a user’s behavior based on past behavior. Each node gathers and processes feedback to calculate a usable reputation index RI for its peers. The proposed node-based TMS provides the ability to enforce multiple access levels dynamically, based on behavior information. A node provides its peers customized views of its resources based on the individual peer’s trust profile and the peer’s trustworthiness.

Hu Tran et al in [15] present a Trust Based Access control Framework for P2P file sharing systems. The access values are evaluated via combinations of four types of scores: direct trust, indirect trust and direct contribution and indirect contribution. Direct trust represents the hosts belief on the clients capacities, honesty and reliability based on the hosts direct experiences. Indirect trust represents the hosts belief on the clients capacities, honesty and reliability based on recommendations from other peers. Direct contribution measures the contribution of the client to the host in terms of information volume downloaded and uploaded between them. Indirect contribution measures the contribution of the client to the network in terms of information
volume the client exchanged with other peers.

Cheng et al in [6] combine P2P trust model and e-transaction characteristics, and present a new P2P-based trust model for e-commerce. By computing interactive experiences, recommendations of other peers, risk factors and the transaction context, the model evaluates every peer’s globe trustworthiness in order to guide transaction participants.

Selcuk, et al. [73] designed a protocol for Peer-2-Peer systems in which a peer wishing to download a file uses reputation from itself and others to help determine which host to download from. They defined several types of malicious resource nodes and proposed a reputation based trust management scheme for P2P networks. The authors compare groups of peers that offer different versions of the same file first and then selects a node that is a subset of the selected group to download the file from. The protocol uses only one criteria for rating a peer that is the correctness of the file. However, other criteria could be examined such as download time.

Ramaswamy and Liu [67] discuss the free rider problem that infests many peer-to-peer systems. They proposed an incentive system that is aimed at keeping peers on the network to share their files which is based on three characteristics: the amount of files shared, the amount of data the peer has shared, and the popularity of the shared file. Peers are thus rewarded based on these metrics. The authors however do not address the spoofing that may occur when a peer wishes to gain access to the system. It would be possible for the peer to create many files with the name of a popular share in order to receive the reward which is additional download capability. Implementing file hashes would solve one problem, but does not prevent the peer from putting up a significant amount of large sized files in order to enhance its ability to download from the system.

Golle P et. al [9] proposed using game theory in addressing the free rider problem in peer-to-peer applications. Their goal was to provide an incentive model to peers in such a way that it balances the amount of downloading and uploading they do. To do this, they introduce a micro-payment system in which peers pay to download and are paid when uploaded from. It was also stated the users would get annoyed from having to pay each and every time a few cents for a download, so they suggest selling blocks of credits to use with downloading. It appears there is no recourse for a peer to correct a transaction in which a bogus file may have been downloaded other than not use that resource again. The only form of motivation to peers to prevent uploading bogus files is that they only receive download credits.
Chen et. al in 2009 [30] propose a new trust model combining reputation and credential. It enables a peer to combine reputation and credential in order to cope with the situation that there is little interaction history about this peer. Besides the trust value of credential is totally derived from the correlation of credential and reputation adaptively, so it can track the change of the credentials character accurately and adaptively.

Most research on reputation mechanisms for peer-to-peer systems use reputation values as a selection criterion and not as a reward scheme. Universal participation and co-operation should be encouraged by rewarding highly reputable peers by providing them with better quality of service, while at the same time taking care about computational overhead.

2.3 Summary

Access control policies have evolved from simple access control rights of subjects on objects within an access control list ACL, to complicated policies that take into account the existence and interaction of different access control systems or organizations each with its own set of access rights. In this chapter we have discussed different access control models for collaborative groups taking into account their evolution in terms of decision making, and their ability to define a trust policy to collaborate in a distributed environment. However with increase in the number of dynamic collaborations which evolve rapidly, one needs to take into account the modification of the environment in the decision making. Collaborative groups tend to re-organize themselves depending on many factors such as evolving trust, thereby warranting the need for evolving policies. We have designed a framework which combines the features of role based and attribute based access control and integrates it with dynamic policies and adaptive trust. We provide a tool that assists participants in collaborations to control the evolution of access control policies as they join and leave collaborations. Our framework allows the decision maker to see the implications of changes to various policies, such as those governing collaboration membership, message transfer and access control.
Chapter 3

Generic Application Framework for Collaborative Peer Groups

P2P collaborative groups are formed by a set of peers with some common interest and controlled membership. Such groups evolve and adapt according to interactions and relationships within the group. Self-organizing groups are those that have the tendency to generate new patterns spontaneously. For example, a work team will generate norms, structures, and procedures based on the behavior of the members of the group, maybe even evolving its original mission. In unpredictable environments, groups must adapt to survive. Such groups need a communication model in which all the peers must collaborate in order to provide the basic services such as content or messages. In order for the peer groups to operate effectively, additional security services must be provided. Peers should be able to prove group membership to other peers of the group. As discussed in section 1.3 in unpredictable environments, there is a need for self-organizing groups to collaboratively modify access control policies dynamically in the interests of the group. These changes should be within the framework of a well-defined group charter. Further, in such dynamic P2P communities where peers are unknown to each other and uncertain about each other’s reputation, it is necessary to develop strategies for establishing trust among peers.

In this chapter, we give the details of our proposed framework for collaborative groups. The framework integrates authentication, admission control, authorization, fine-grained access control, and key management. We give the detailed system architecture for the same. We propose a system model to represent collaborative groups. Our framework permits groups to start in one system model and adapt to a different system model to maintain group efficiency. We
also give a statechart based formal model to represent our framework.

### 3.1 Generic Model

In our generic framework (figure 3.1), peers with a common interest join together to form a group. Each group has a **Group Charter** with information about Group Description, Group Activities, Meta Rules for membership and Group Composition.

![Figure 3.1: Generic model](image)

The peers are governed by a set of rules defined as policies in the Group Charter. We define two types of policies, Global and Domain Specific. The global policy describes the roles permitted in the group and the rules associated with each role. A well-formed group charter could have some initial global policies such as:

- Rules for creating and destroying a group.
- Well-defined rules for adding new peers and discarding peers.
- Roles permitted in the group and rules associated with each role.
- Rules for sharing resources that exist among different peers based on the trust value associated with the peers.
- Rules for minimum trust levels to be maintained and trust model to be used.
Domain specific policies are typically those policies that could be changed dynamically in order to maintain group efficiency. Thus rules for join, role or level updation, or maybe number of optimum number of peers required at each level could be part of these policies.

### 3.1.1 Group Activities

We broadly classify the set of activities performed by peers as: Group Management Activities, Task Oriented Activities, Social Tasks and Malicious or Destructive Activities.

#### Group Management Activities

Activities related to creation of the group and group charter and publishing its existence, admission and access control, and resource management are some of the major group management activities.

#### Task Oriented Activities

Peers belonging to a group are expected to perform different tasks like providing resources, rating other peers, maintaining certificates and keys, evaluating performance by other peers and application specific tasks like development of software or content for applications like F/OSS or wikipedia. These are listed as Task Oriented activities.

#### Social Activities

Those activities which contribute towards improving the overall performance of the group fall in this domain. Some such activities are:

- Reward actions
- Punish actions
- Observer actions i.e maintaining records of those group activities that help in providing feedback.

#### Malicious Activities

One of the major challenges of collaborative P2P groups is the ability to manage risks involved in interacting with malicious peers. We model malicious activities as:
• Aggressor behavior i.e peers who lower the rating of other peers.

• Recognition seeker i.e colluding peers who call attention to self by providing very good behavior for some time and increasing each others ratings.

3.1.2 Group Efficiency

We define some of the metrics for evaluating group efficiency as:

• Evolution of the group

• Current Aggregate trust

• Current Group composition

• Past history

To compute the group efficiency we define an abstract function GroupHealth which depends on different parameters like Average Reputation of peers in the group, Rate or percentage of task completion or goal achievement, Frequency of Updation of peers, Current Skill set of Peers, Rate of Peer Join, Group size or composition and so on. These parameters cannot be known apriori and have to be periodically measured by peers as the group evolves. We define it as GHP.

\[ GHP = \{ \text{AvgTrustRateOfPeers, AvgSkillSetOfPeers, RateOfCompletion, RateOfUpdate} \]

\[ \text{RateOfJoin, RateOfLeave, RateOfAllocation, SizeOfGrp} \} \]

3.1.3 Group Goals

The goals of collaborative peer groups are

• Secure Group operations

• Resource Management and Scheduling

• Optimum performance in presence of malicious peers
3.2 System Architecture

We present here the overall architecture for secure group communication in decentralized peer groups, with a dynamic access control framework and an adaptive trust model. The architecture is suitable for applications that involve sharing of high value resources, e-commerce applications, as well as general collaborations where security is not a major issue. The access policy framework is flexible and allows peers to change their levels as well as change the rules for membership within an existing level. Peers are rated based on their performance in a group along with contributory recommendations by other peers. A peer can then invoke the rekeying algorithm to elevate his level. Policy rules for elevating members from one level to another as well as for introducing new levels are dynamic and peers are able to collaboratively frame and manage such policies. Peers can have different functional components depending on their role in the group. We define two types of roles viz Group Roles and Domain Specific Roles.

3.2.1 Group Roles

We have defined different functional roles for peers as Group Roles.

GR={GR_1, GR_2, ..., GR_k}

A typical application has three group roles viz: GR={MP, AP, CP}.

- **Member peer** (MP). A peer in this role is a minimal functionality peer who participates in the normal group activities but does not contribute to the admission of new peers, nor updating of levels of existing peers. Thus this peer is only a service provider.

- **Admission Peers** (AP). These peers are allowed to admit new group members if the access policy permits. Thus these peers have the additional tasks of doing authentication and invoking voting if required.

- **Control peer** (CP). The control peers are the super nodes of the framework. Every group must have at least one control peer. These peers are responsible for broadcasting essential messages like

  1. Joining of a new peer
  2. Policy change (if it happens)
  3. Updated ratings of peers.
Thus a peer in this role would have all the functional components and would participate in periodically updating levels of existing peers and also permitting change of role. These peers are also responsible for monitoring group activity and keeping a track of group performance index GPI. If GPI falls below a certain threshold then a CP can call for a consensus of AP’s or MP’s to decide on a policy change. Framing of new policies like adding a new level to the group can also be done by a CP.

### 3.2.2 Domain Specific Role

These are system roles at hierarchial levels, depending on the application domain, access rights/capability and trust value. So, for example for a e-learning application the roles could be teacher, student, T.A or Grader, Examiner, Candidate. We abbreviate Domain specific Role as DR. Thus for example DR = \{DR_1, DR_2, DR_3\} for F/OSS application is the set of roles Developer, Reviewer, Moderator.

We define the set of Users/Peers as (Pi), set of Roles(Ri), set of Permissions assigned as (PA), and set of Permissions as (P). The actual role assigned to a peer R is a union of group role and domain role.

Thus R = DR ∪ GR where DR = domain role and GR = group role and set of permissions assigned PA is a mapping from permissions to roles. PA ∈ PxR.

### 3.2.3 Functional Components

Peers can have different functionality depending on their group role. A control peer for example, is the maximal role peer and has the following functional components:

**Attribute Manager**: Responsible for storing the attributes and certificates of the peers. The Attribute Manager is also responsible for updating the certificates of the peers with the new rating.

**Authentication and Authorization Manager**: Verifies the validity of the identity, and can invoke the voting algorithm.

**Trust Engine**: Responsible for calculating the trust values of each peer, based on different parameters, like direct interactions and recommendations from other peers. It has a calculation engine and a feedback engine associated with it.

**Policy Manager**: Frames rules or policies based on attributes, behavior of peers and inputs
from trust engine. It also checks the rules to allow members to join. It is called upon either by the Authentication Manager in case of a new member join, or the Updation Manager in case of an existing member level updation.

**Update Manager:** Responsible for checking the rating levels of peers after they have collected feedback from other peers and gives the input to the Access policy manager. It is also responsible for granting or revoking access rights to peers based on the dynamic policy.

**Key Management System:** Invoked whenever a new peer joins or an existing peer’s access level is changed depending on the dynamic policies, and is responsible for computing the group key. These components interact as shown in the figure 3.2.

![Figure 3.2: Integrated framework](image)

### Assumptions

Our framework is based on the following assumptions:

- The system is asynchronous i.e there is no global clock.
- Domain specific rules and policies reside in the local memory of a peer.
- Reliable message delivery is assumed.
- No peer symmetry, i.e peers do not have equivalent functionalities.
• Peers hold a “template” that defines their group specific activities/capabilities and other information

• Every group must have a well defined Group Charter at the time of creation, which is updated periodically.

• Peers are individually capable of performing tasks of authentication, voting, access control, key management etc.

• Peers can compute their own public-private key pair and provide self signed certificates, binding their identity with their public key.

### 3.2.4 Detailed Working of the Framework

The group has an initial admission control policy similar to the work proposed in [1, 2] based on a dynamic threshold i.e t-out-of-n where t(no of peers collaborating) grows or shrinks in tandem with n(current group size). The admission policy could be changed collaboratively by peers and this change is enforced using voting. Peers wishing to join the group, declare an initial self-proclaimed rating(SPR) and request for some task allocation. Based on the join access policy, he may be allowed to join at a lower level and would be allotted some role based on his self-proclaimed intrinsic capability. If permitted to join, a peer is given a signed membership certificate which serves as an authentication token. For all practical purposes a new peer will be allowed to join a group at the lowest level initially. The membership certificate contains a field which has the trust level or rating of the member.

A new peer joining the group is assigned a minimum trust level. Once in the group peers are periodically rated by other peers in the group and signed rating certificates are saved by some of the peers having higher functionality. Trust value of a peer at any point of time is calculated based on the direct interactions as well as recommendations that he has received from other peers as well as the trust value of the recommending peer. Rating certificates are signed by recommending peers and hence cannot be modified. Each time a new peer joins or an existing peer leaves, group rekeying is done so as to ensure backward and forward secrecy. The framework defines policies for a new peer to join, or leave and for an existing peer to update its level. The access policy would decide the role a peer assumes, based on the request of the peer and his credentials as well as rating calculated by the other peers. A peer may have some
maximum potential or skill set and could rise only up to that potential. Each peer maintains information related to

- its own private-public key pair
- Signatures from other peers
- List of peers trusted by him

Group access is done in stages viz Group Discovery and Advertisement, Join Request, Authentication, Access Control and Key management, as detailed below.

**Group Discovery and Advertisement**

A peer willing to serve as Control peer can create a new group and define an Initial Group Charter. The peer can then advertise and publish his group along with his charter which contains documents that specify the parameters of the group, such as group type, admission policies, group name, group members etc. Subsequent peers can then find this group and join it. The group advertisement should be periodically broadcast by the peers of the group.

**New Member Join**

This is the process by which a peer applies to be accepted into a group (figure 3.3). During this the peer may receive credentials(keys, passwords, tokens) that it may use later to prove group membership. The registration process requires knowledge of existence of the group. A new peer wishing to join a group would first search for an advertisement of a group that matches his requirements. He would then submit a signed request to any peer which would include his credentials that he obtains from a CA say X.509 certificates or self-generated certificates, along with a request for the role that he wishes to join in. Given that each peer has his own certificate which could be self signed or signed by a CA, a peer credential is created by hashing the concatenation of unique user ID PID and public key fields and then signing this hash with the private key of the user and using this digital signature as the identity of the peer. This identity could be used as the peer’s credential in the messages.

\[
P_{\text{new}} \rightarrow P_i : \{\text{JoinREQ}\}_{SK_{\text{new}}}, Cert_{\text{new}}
\]

where

\[
Cert_{\text{new}} = PID_{\text{new}}, PK_{\text{new}}, RC_{\text{new}}, \{H[PID] || PK_{\text{new}}\}_{SK_{\text{new}}}
\]
where $PK_{new}$ is the public key of the new peer and $SK_{new}$ is his private key.

PID is the unique ID of the peer. The hash of this PID is concatenated with his public key and
the result signed with the private key of the peer to form his unique user ID.

The $Join_{req}$ consists of a tuple $\{GR, DR\}$. For a new peer his Rating Certificate i.e $RC_{new}$
field will contain a single entry which is his self proclaimed rating SPR.

**Peer rating certificate RC**

A rating certificate can be used as a means of recommendation. This certificate contains the
original trust value of the peer along with the recommended rating value given by each peer. The
recommendation is signed by the recommending peer thus preventing a peer from being able
to modify the recommendations/ratings given to him by other peers. The rating certificate also
contains a validity period beyond which it cannot be used. Thus for each peer it has interacted
with, a peer needs to keep a copy of the certificates it issued to that peer for validation purpose
and for trust and contribution score updating. In addition a peer could maintain a revocation list
of peers who are not to be trusted and periodically this information could be exchanged amongst
peers. The fields in the rating certificate are Recommending peer’s identity, Recommended peer’s identity, Original trust value, Issuing date and time, Expiry date and time, Contribution score and Signature of recommending peer.
Authentication

If the peer receiving this request is a member peer, he would forward it to an Admission peer who will obtain the identity of the requesting peer from the Certificate. If it is a signed certificate then the verification is easy. If however the Public Key pair is pseudo generated then the receiving peer will compute the hash of UUID and $PK_i$ and tally this with received message. If the two hashes match then the user is authenticated and will be granted access permissions based on the access control policy. If the user’s credentials contain a rating field with a single entry he will be treated as a fresh user.

We assume that each joining peer stores a local copy of the policy database. When a CP or an AP receives a join-request from a new peer it checks the policy file for the join policy. If all parameters match then the later then gives a signed membership token to the new peer.

$$P_i \rightarrow P_{new} : \{GC_{new}\}_{G_{key}}$$

Since this is signed with the current group key which is unknown to the peer the later cannot modify it.

Access Control

We define a dynamic policy driven trust based hybrid access control model for the peers in our framework. The detailed model is discussed in chapter 4. The initial group policies are defined in a Group Charter, normally by the creator or the group. As peers join, the policies can be dynamically modified by control peers in the group. Thus the peers are able to self regulate and self organize, based on trust and composition. Our trust metric combines context specific attributes, self-proclaimed ratings and recommendations received from other peers. Parameters of the trust metric can be dynamically adjusted.

Key Management

After the peer receives the signed Group Membership certificate it is the task of the CP to perform the group rekeying at the necessary level. The new peer is allowed to join and his public key and unique user ID (i.e hash of his user ID and public key) is broadcast to the group. CP invokes the rekeying algorithm (TGDH)[82] and the new group key computed is sent to this peer encrypted with his public key, and to all other peers encrypted with the old group key.
Every member keeps a list of keys and the associated set of members that share that key. The list is updated whenever a new key is generated.

The entry of the new peer is broadcast to all peers of the group. The membership token $GC_{new}$ would contain details of the access rights granted to the joining peer.

**Existing Member Level Update**

Once a member of the group, a peer can periodically request the members of the group to rate him based on his performance and the trust level he has already acquired. Appropriate access privileges are granted to a peer based on a scoring system. The actual protocol is as follows (figure 3.4):

**Request for Update**

$$P_i \rightarrow P_j : \{UpdateREQ\}_{SK_i}, Cert_i$$

where $Cert_i$ contains rating certificates from other peers. $P_j$ computes the global rating value after verifying the authenticity of individual ratings.

![Figure 3.4: Member level update](image)

The Update Manager interacts with the trust Engine to calculate the trust value or rating of the peer. Different parameters like Feedback by Direct Interaction, Indirect Recommendation, Self Proclaimed Rating and others are used to compute the trust value. It then forwards this request with rating score to the Policy Manager.
**Peer Evict**

A peer $pi$ may be evicted from a group $G$ upon invoking the leave protocol. The leaving peer normally broadcasts its information to the overlay and retires from all maintenance functionalities.

**Member Freeze**

If a peer has not been participating in group activities for a long time, the system does not send it any further key updates.

**Group Removal**

This is done by shutting down the group overlay, so that no nodes remain. A control peer is needed for this task. Group and membership information is removed from the overlay.

![Figure 3.5: Policy based framework](image-url)
Thus we have a policy based framework which interacts with the evolving trust parameters of peers in the group. Figure 3.5 depicts the interaction between global policies, application specific meta policies and the trust engine for peers having multiple roles in multiple levels. In our thesis we mainly address the dynamic policy based access control section which is detailed in chapters 4 and 5 and the adaptive trust engine which is detailed in chapter 6.

In the next section we give a formal system model for our framework for collaborative groups and represent it using Statecharts.

### 3.3 System Model

Collaborative groups tend to organize themselves into different system models. A system model defines the behavior of peers, different roles, events, and policies for membership control, access control, task assignment, security and others. While evolving, peers have to decide which policies to choose and which system model to follow. A group which has started with some initial set of policies may need to dynamically change some of the policies for survival in unpredictable environments. As an explicit example suppose the group has a policy for re-allocating an unfinished task when a peer leaves abruptly. Suppose 20 peers decide to leave simultaneously or are ejected due to mal-performance, then the current policy may not be viable and the group may need to restructure its policy decisions. This could be achieved by monitoring rate of join and rate of leave of peers as the global health parameters. For a collaborative group like Wikipedia for example, the group could start with an initial system model defining three roles say readers, editors and administrators. As the group evolves the admin peers may feel the need to introduce roles like proof readers or translators, which means that the group needs to change its system model. The process workflow of the group sometimes needs to undergo a change for optimum performance.

Our framework permits the group to start in one system model with a pre-defined set of roles, levels, policies and a specific trust model. The peers in our model are governed by a set of rules defined in the Group Charter, that describe the minimal conditions to be part of the group. Peers have the flexibility to monitor the group health and accordingly change the group policies thus making it adaptive. The group can thus dynamically switch between different policies for membership control, and access control in order to optimise performance. Not only can peers dynamically switch between policies, they can collaboratively modify the group charter.
itself and adapt to a new system model where different policies, roles, levels and global health parameters may apply.

We define our system model as the set

\[ SM = \{\{Peers\}, \{Events\}, \{Tasks\}, \{Policies\}, \{Roles\}, \{GHP\}\} \]

The behavior of peers in the group is governed by the elements of the system model. A change in the system model therefore means a change in the structure and organization of the group as well as a change in the behavior of the peers. We are able to achieve both dynamic policy adaptation as well as system model adaptation, enabling the group to evolve and adapt to evolving policies.

An application like F/OSS could have different process workflows. There could be varying levels of reviewers or moderators or the process of reviewing, moderating or submitting code could be different in different workflows. So a system model change in F/OSS would mean changes like

- submit code to 3 reviewers in parallel instead of a single reviewer
- allocate same task to 3 coders instead of one.
- add an additional reviewer role and submit intermediate code to Reviewer 1.
- adapt to a new trust model to calculate ratings of peers.

A gaming application could have different difficulty levels and one could decide on different scoring schemes. So for an application like gaming, system model change could mean

- Increase/decrease the number of difficulty levels in the game
- Change the scoring scheme.
- Switch to a different trust model.

On the other hand a policy change F/OSS could mean changes like

- more stringent join policy as peer count increases
- more stringent update policy as control peer count increases beyond a threshold
- change in job allocation policy depending on type of jobs and trust value of peers

whereas in gaming policy change could mean
- more stringent join policy for a particular level
- change in task allocation policy depending on current group composition and peer performance
- relaxation in join/update policy when rate of peer update to next level is slow.

### 3.3.1 Peer Attributes

Previous basis for peer to peer group formation has been “symmetry” where peers are always capable of performing equivalent tasks. Our model proposes a framework for peers with different functional capabilities to form special interest peer groups. Peers are assigned Group Roles based on the functionalities. Peers with different capabilities can belong to different levels, which are mapped as Domain Roles. Peers have an initial self proclaimed rating (SPR) which is the initial trust value. A peer has a maximum intrinsic potential MIP and can rise only upto that level.

A peer in our framework has a unique user id UUID, which is a binding of his user id PID to his public key which is part of the self generated key pair (Sk,Pk). We define the attributes of the peer as the set.

\[ Peer\_attrib = \{PID, Peer\_info, Auth\_info, Security\_attrib, GID\} \]

PID is the identity of the peer.

\[ Peer\_info = \{IP\_addr, time\_of\_join\}, Auth\_info = \{h(Pk||PID)\} \]

Auth\_info is a hash of the public key with the Peer ID.

\[ UUID = \{Auth\_info\}_{Sk} \]

\[ Security\_attrib = \{Pk, UUID, SPR\} \]

UUID is Auth\_info signed by secret key Sk of peer and serves as the unique user ID.

GID is the group ID of the group which peer belongs to.

SPR is the self proclaimed rating which a peer proclaims when he gives a join request.
3.3.2 Peer Behavior and Roles

Peers can self regulate their behavior based on learning experience in the group. The behavior of a peer is categorized by the role which he is in. A peer can either a service provider or a rater. Quality of a peer as a service provider is independent of the quality of the peer as a rater. The additional functions that a peer is capable of performing in a group are storing and verifying certificates, authentication, voting, updating of levels, key management etc. The extent to which a peer performs these tasks is decided by the role the peer wants to play in the group. We define two types of Roles viz Group Roles and Domain Specific Roles. A group role is based on the functionality of the peer while Domain Role is based on capability of the peer in the specific application domain. Thus the domain role can be mapped to levels.

Change of Roles

At any time a member peer may want to take up the role of an Admission Peer or a Control peer. Since the model focusses on the self-organizing and collaborative nature of peers, in order to achieve group efficiency, any peer which is part of the group may apply for a role change or level change. As peers belonging to CP or AP perform more operations, nodes with sufficient bandwidth or computing power may apply. Similarly if a CP decides to change its role to MP for some reason, such a peer would lose group membership unless it can transfer control to some other peer. If a peer is a single CP then it cannot be allowed to change its role. Thus change of role is also part of the global group policy.

3.3.3 Events

Events occurring in the system are categorized as external events and internal events. External events are join-request, new-task-entry and leave. Some internal events are update role, update level, allocate-task, re-allocate task, policy-change, rate-peers, eject peers and other application specific events.

3.3.4 Tasks

The tasks are categorized as group tasks and application specific tasks.

Task-Set = \{GT_i, AT_i\}
\[ GT_i = \{T_1, T_2, ..., T_n\} \]
\[ AT_i = \{T_{a_1}, T_{a_2}, T_{a_3}, ..., T_{a_n}\} \]

The attributes associated with application tasks are defined as follows:

\[ Task\_Attrib = \{DiffLevel, Time0fEntry, ExpectedTimeOfCompletion, Cost, ErrorCount, SecurityClassification, TaskStatus\} \]

\[ ExpectedTimeOfCompletion = functionOf\{DiffLevel, Role\} \]

\[ TaskStatus = \{Allocated, Waiting, Running, Completed, PartiallyCompleted\} \]

Application tasks are specific to the domain. For example in the F/OSS domain a sample task-set is \{submit code, review code, moderate code\} and so on. Associated with the tasks are parameters like difficulty level, start-time, end-time, cost, error count which could be the bugs in the code, Security Classification i.e whether it needs encryption or not, whether privacy is needed or not.

The group level tasks or meta tasks are common to all application domains. Some such tasks are:

- Information/Resource Request \( (T_1) \)
- Information/Resource Response \( (T_2) \)
- Feedback/Rating Request \( (T_3) \)
- Feedback/Rating Response \( (T_4) \)
- Voting Request \( (T_5) \)
- Voting Response \( (T_6) \)
- Record Maintenance (certificates/keys/ratings) \( (T_7) \)
- Monitoring Group Performance Index GPI \( (T_8) \)
- Providing incentives for correct ratings \( (T_9) \)
- Punish actions for incorrect ratings \( (T_{10}) \)

The Group Role decides which meta tasks would be performed by the peers. A peer in the Member Role for example would perform tasks \( T_1...T_4 \) while a control peer can perform all the tasks \( T_1..T_{10} \).
3.4 StateChart Based Formal Model for Dynamic Groups

Statecharts is a visual formalism, introduced by Harel in the mid Eighties [85], for specifying the behavior of reactive systems, i.e., concurrent systems that are characterized by their ongoing interaction with their environment. A Statechart (an extended version of state diagram) is a visual construct that enables you to define event and timedriven behavior of various objects. They are very helpful in simulation modeling. They are used a lot in agent based models, and also work well with process system dynamic models. They define internal states, reactions to external events, and the corresponding state transitions of a particular object: a person, a physical device, an organization or a project. They can therefore be suitably used to accurately describe the behavior of complex systems [88].

The StateChart is a tool which is an extension for state-transition diagrams that adds concepts of hierarchy (depth), orthogonality (representation of parallel activities) and interdependence (broadcast-communication). The basic elements that are part of this high-level specification tool are: States, Events, Conditions, Actions, Expressions, Variables, Labels and Transitions. An event is a very important element to observe the model of a given system. The occurrence of an event can be considered as an interference to the system in the sense that the present system behavior as a whole is changed to another behavior with its (events) occurrence. Statecharts [86, 87] provide some special events such as true (condition), false (condition), entered(X) and exited(X). When a condition is evaluated and it is true a true event is generated. The same situation occurs with the false event.

We were able to model our dynamic system appropriately using StateCharts. The basic elements of our model consists of States, Events, Transitions, Conditions/Guards, Actions, Expressions, Variables and Labels. Our system workflow can be formally represented by a statechart. Each workflow instance has its own copy of the statechart. An activity of a workflow, is represented by a state in the statechart. An activity is carried out only if the system resides in the state that corresponds to that activity. Transitions between activities are thus modeled as transitions between states in the statechart, which are triggered by events and guarded by conditions. Events can be external (generated by elements outside the statechart) or internal (generated by elements inside the statechart). A transition between a source state and a target state takes place if and only if the system currently resides in the source state, and the event of the transition occurs, and the conditions that guard the transition are true. In other words, the
system must be in the source state, the event must occur and the conditions must be true for the transition to take place. Transitions are thus driven by ECA triple, \( X \rightarrow Y \) occurs if \( E \) occurs and \( C \) holds. State \( X \) is left, \( Y \) is entered and action \( A \) is executed. An action \( A \) can explicitly start or stop an activity and can generate an event \( E \) and a condition \( C \). ECA rules of this kind are noted as \( E[C]A \). A state change is possible without any pre-condition.

A Statechart may react to an event by engaging in an enabled transition, thereby performing a so called micro step, which may generate new events, as described by the transitions action, that may in turn trigger new transitions while disabling others. When this chain reaction comes to a halt, one execution step, a so called macro step, is complete.

### 3.4.1 States and Transitions

A state is an observable condition that the system can be in. The behavior of the peers is governed by the state the system is in. Associated with a state therefore is an Action Set and a Policy set which defines the actions to be taken when certain events occur. If an event occurs and the guards holds (i.e. evaluates to true), object can change its state. Such a state change is called a transition. So, a statechart diagram represents the relations among states and transitions. Statecharts are drawn as directed graphs in which nodes represent states and edges represent transitions. A state is represented as a rounded rectangle including the activity it performs and the name separated by line from the action part. A transition is drawn as an arrow among source and target state. A statechart also has special kind of states, the initial states and the final states. An initial state represents the starting state or point of statechart. A final state is the one that has no outgoing transition. There can be more than one final state in a statechart but only one initial state. Reaching a final state means that now no event can cause transition to any other state. An initial state is represented as filled circle whereas final state is represented as bull’s eye icon. A state also has associated with it global state variables or data structures which are modified on occurrence of each event and its action. So for example an event like a join-request followed by an action allow-join, would modify the queue-size as well as counter for number of peers in joined-list, depending on the type of peer. Some of the state variables are number of Roles, Levels, size of Group, number of member peers, admission peers and control peers.

\[
S_v = \{ \#Roles, \#Levels, GroupSize, \#MPs, \#APs, \#CPs \}
\]

These state variables are monitored by the control peers in the group after every event.
occurrence.

3.4.2 Guards and Conditions

We have defined an abstract function GroupHealth whose parameters are Average Trust Rate or Reputation of the Group, Average Skill Set of the Peers, Rate of Job Completion, Rate of Peer Update, Rate of Join (Number of joins in last x minutes), Rate of Leave (No of leaves in last x minutes), Rate of Job Allocation and Total Group Size. Individually each of these parameters represents a guard condition.

\[ c1 = \text{AvgTrustRateOfPeers} \]
\[ c2 = \text{AvgSkillSetOfPeers} \]
\[ c3 = \text{RateOfCompletion} \]
\[ c4 = \text{RateOfUpdate} \]
\[ c5 = \text{RateOfJoin} \]
\[ c6 = \text{RateOfLeave} \]
\[ c7 = \text{RateOfAllocation} \]
\[ c8 = \text{SizeOfGrp} \]

These guards cannot be known apriori and have to be periodically measured by the control peers as the group evolves. They are monitored as weighted vectors in the state space. Put together, these parameters contribute to the Group Performance Index GPI, which can be measured periodically by control peers.

Boolean expressions are used to specify a combination of these guards. For example
\[ c1 \lor c2 \lor c3 \]
represents a Boolean combination of some conditions.

The Group Health Parameters are affected by the peer behavior i.e way a peer rates other peers, the way a peer handles a task, the task allocation policy adopted, the join and update policy adopted and so on. For a healthy group the join policy should be such as to permit a minimum threshold \( t \) for the rate of join, hence if RateOfJoin \(<< t\) then there is a need to change the join policy. Similarly if the ratio of group composition i.e \( MPs:APs:CPs \) is \( 80:10:10 \) it would mean that the number of Control and Admission peers are not sufficient to carry out the group admin tasks, necessitating a change in the group update and or join policy. The dependancy on this ratio could probably depend on the current group size i.e a group with Total 100 peers may perform better with a ratio of \( 50:30:20 \) whereas a group of size 300 may perform better with a ratio of \( 40:40:20 \). These cannot be known apriori and peers therefore
need to be able to collaboratively modify the group policies dynamically.

The policy to be applied depends on the outcome of both the state variables and the guards. So for example if the control peers observe that the RateOfJoin has decreased below a certain threshold since join in last 10 minutes is nill, then they can collaboratively modify the join policy and make it less stringent.

3.4.3 Events and Actions

Events could be internal or external and have Actions associated with them. Some external events are join request, join, job-entry, while internal events are update request, policy change, rating, evaluating trust value and so on.

3.4.4 State Transition Depicting System Model Change

Transitions are represented by the expression "event/condition/action" (e/c/a) i.e they occur in reaction to events, provided the guard conditions are satisfied. Triggers and guards determine the specific combination of events present when a state transition may occur. Trigger and guard conditions are specified with Boolean expression models that define the conditions under which a given transition should be enabled. The occurrence of an event could serve as a trigger condition while evaluation of existing state variables serve as the guard conditions which ultimately decide whether or not a transition is enabled. Actions define the operations to be performed when a transition occurs. Actions also change the value of the state variables.

A transition is drawn as an arrow among source and target state. An initial state represents the starting state or point of statechart and is represented by a filled circle. We define three different kinds of transitions viz $T^0$, $T$ and $T'$.

When the system is in a particular state, both internal and external events occur. The control peers monitor the state variables and periodically measure the group health parameters. If the group performance is within a specific threshold then on occurrence of events the system transits to the same state. We call this a $T^0$ transition.

A state can be simple i.e Basic or Composite(OR/AND). A simple state does not consist of any sub-states. An OR-state consists of sub-states and to be in an OR state the system must be in exactly one of the sub-states. The global parameters share the same state space in the OR-state. When in a sub-state the system is governed by a Policy $Pi$. A specific combination
of event and guard trigger a change in state variables, which in turn trigger an action which is a policy change, taking the system to a new sub-state within the OR-state. Thus a $T$ transition happens when $e/c/a$ occurs where $a$ is a policy change. In this new sub-state the processing of events change as they are now governed by Policy $P_j$ which is active in this state. Thus the system keeps switching between different sub-states whenever there is a policy change triggered by control peers monitoring the Group Performance(GP).

We extend the StateChart model to introduce a new type of transition $T'$ which occurs when a special $e/c'/a'$ occurs causing a system model change as well as local policy changes. This would mean change in the Global Health Parameters as well as State Variables. So new Roles and Levels could be introduced or the process itself could be changed, thereby changing the rules in the Global Charter. Thus on a $T'$ transition system transits from state $S_i$ with system model $SM_i$ to state $S_j$ with system model $SM_j$. The Peer set in the new model would be a subset of the existing peer set, while the roles, levels, global health parameters and policies in the new model could be either a subset or a superset in this new model. Completely new policies may be added in the new model. These would be done in consensus with the control peers.

A sample statechart model is shown in figure 5.3. There are three complex states $S_1, S_2$ and $S_3$ in this model. The system could be in any of the states initially. Suppose it is in sub-
state $S_{21}$ within State $S_2$. Assume policy $P_i$, is active in this state which is a join policy with no constraints. Events $e_1,e_2,...,e_n$ can keep occurring in this state. Whenever an event occurs, the CPs check the GP by monitoring the global health parameters. If GP does not fall below a certain threshold $t$, then the system continues to remain in this sub-state, i.e there is no policy change and no transition outside this state. This is the $T^0$ transition.

The system continues to remain in same state (i.e $T^0$ transition) till certain conditions trigger a policy change and transit the system to sub-state $S_{23}$ or $S_{22}$. For example suppose the RateOfJoin increases beyond a certain threshold $t$, and the number of member peers also increases to greater than 50% of group-size. This triggers an action which is a policy change and which takes the system to sub-state $S_{22}$. In this new state also events $e_1,e_2,...,e_n$ keep occurring but the response of peers to events is now as per policy $P_j$, which has more stringent rules for join and update. Thus the system can switch between substates in $S_2$ whenever a condition triggers a change in policy. In state $S_2$ therefore dynamic switching between policies occur. These transitions between substates are $T$ transitions.

Transitions leaving an OR-STATE are said to leave any of the enclosing sub-states. So when the system is in any one of the substates of $S_2$, and the GP falls drastically below a certain threshold, then on occurrence of some event $e_i$, the system switches to state $S_3$ where the system model changes from $SM_2$ to $SM_3$. Here new policies may apply, new roles and levels may be introduced and new global variables may be monitored. For the F/OSS application, if the global parameters RateOfAcceptance and AvgSkillSet fall beneath a certain threshold, control peers may decide to adopt a different system model where an extra reviewer role is introduced and the way in which codes are submitted and reviewed is changed. Thus this would mean a $T'$ transition to state $S_3$ where system model changes. Or suppose the trust model itself were to be changed then it would mean a $T'$ transition to state $S_1$. Within the new state now, some new policies would be added and dynamic switching between sub-states could again occur.

We depict the change in system model from some $SM_i$ to $SM_j$ by changes in its components as follows:

$$SM_i = \{P_i, GR_i, DR_i, Pol_i, GHP_i\}$$

$$SM_j = \{P_j, GR_j, DR_j, Pol_j, GHP_j\}$$

where

$$P_i \subseteq P_j, DR_i \subseteq DR_j or DR_i \supseteq DR_j$$
Thus in the new system model the set of peers and the policy set could be a superset of the previous model, while the group roles and domain roles in the new model could either be a subset or a superset of the previous model.

When the system is in any sub-state of state $SM_i$, all the State Variables are modified except $\#Roles$ and $\#Levels$. The Action Set =\{A1, A2, .....An\} is linked to the policy Set.

For example On Event Join, Actions that would occur are

- Add Peer to JP list.
- Increase count of Peer Type (MP,AP or CP).
- Increase count of Total Peers (TP)

On event job completion the action taken would be *Increase Count of Free Peers*.

### 3.4.5 Formal System Model

Our formal system model is defined as a 10 tuple

$SM = (S, T^0, T, T', E, C, A, i, f, \psi)$ where

- $S$ is a finite set of States
- $E$ is a finite set of events
- $C$ is a finite set of conditions or guards
- $A$ is a finite set of actions
- $T \subseteq S \times E \times C \times 2^A \times S$ is a finite set of transitions where $2^A$ denotes the power set of $A$.
- $T': S_i \rightarrow S_j$ is a set of transitions from one OR-state to another OR-state.
- $T^0$ is a transition to the same state.
- $i \in S$ is an initial state.
- $f \in S$ is a final state.
• $\psi : S \rightarrow $ BASIC, AND, OR is a function that defines state decomposition.

Thus we have a statechart formalism for our framework for collaborative groups which supports dynamic adaptation of policies and group system models.

In the next chapter we give details of our dynamic policy framework, and its implementation using a declarative language Prolog.
Chapter 4

Dynamic Policy Driven Trust Based Access Control (DPDTBAC)

Policy based approaches to the management of self-organized groups allow the separation of rules that govern the behavior of the system, from the functionality provided by the system. Thus it is possible to adapt the behavior of the system without the need to recode functionality and changes can be applied without stopping the system.

There are several types of policy useful in managing distributed systems. Broadly policies can be classified into authorization policies and management or obligation policies. The former speaks about access control requirements of a system while the later holds requirements related to system behavior. Authorization deals with issues like who can access which resources/services under which conditions. Many mechanisms e.g Role based, Rule Based, Identity Based access controls exist. However these authorization mechanisms alone cannot satisfy the access requirements of distributed services as access depends on many factors like privacy requirements of the requestor, authentication requirements of the service, trust relationship with the requestor, and so on. A policy in this context can be defined as a set of rules or requirements associated with a group or peer or resource.

Authorization in a distributed environment can be granted after evaluating the request of an authenticated user against various policies like privacy policy, trust policy, authorization policy and so on.

Traditionally, policy-based management models have considered that roles, their membership and the policies assigned to them can be determined statically in advance by the administrator. A role in this context is a consistently specified collection of policies governing the
actions of assigned users. In dynamic peer groups, the behavior of peers is unpredictable and so are the sequence of events and their likely outcomes. Hence dynamic policies are needed.

4.1 Proposed Access Control Model

Our improved access control model is based on an integration of collaborative roles, policies and trust [37, 4]. A role represents a set of privileges and rights. It involves meta-level policies in regard to admission of users to the role. Integration of the roles, tasks, privileges and security policies needs a proper framework for policy specification.

Our model focusses on the genetic evolvement of groups based on group composition. In DPDTBAC it is possible to deploy access control policies flexibly and dynamically. Based on the behavior and trust level of peers in the group and the current group composition, it is possible for peers to collaboratively modify policies governing their level. The group policy can also dynamically prioritize requests for join. Join priorities would depend on current group composition. Thus low level requests for join could be postponed in a group which already has a large number of members performing the lower roles. Peers in one level are also allowed to be dynamically updated to a higher level. Update policy also depends on current group composition and behavior. A peer could also be collaboratively ejected from a higher level to a lower level or even out of the group if his behavior in the group degrades. It is also possible for peers in the highest level to introduce a new level in the group if the situation so demands.

Multiple policies can apply to a group, and the system can switch between policies over time, influenced by the system’s state or environment. Some policies can have higher priorities than others. Policies can apply to different levels of the system and can be global or apply to only parts of the system.

We define a set of Global Meta Policies and a set of Domain specific policies. The Management policies are part of global policies and specify policies for join, leave, update etc. To address privacy issues the domain plays an important role. Domain or context specific policies refer to the set of rules/regulations/requirements of a domain to which an entity must conform to in order to be in that domain. To implement privacy based access, a variable can be attached to the information to indicate the privacy level of information. So if full privacy is chosen then the information will be sent with encryption and signature.

The trust parameters in our model build upon the PeerTrust[7] model. We specify context
based attributes for specifying trust and assign specific weights to these attributes. We further define a tunable trust metric to combine all trust parameters viz: direct trust, indirect trust based on recommendations and credibility of the peer. As peers perform in the group, trust of each peer evolves based on the interactions. We are able to express dynamic rules at runtime, as well as change existing policies depending on the status of the group at runtime through Prolog.

### 4.1.1 Global Policy

This is the policy initiated by the creator of the group. It includes rules for admission, updating and leave as well as rules for deciding the no of control peers needed to collaborate to frame new rules, or change existing rules. Some sample rules are:

- A peer can join as a member peer at level 1 of the group if he has received votes greater than 50% and if his current rating is greater than some threshold x.

- To update level a peer must have an accumulated trust value \( > x \) and should be an authenticated member at the lower level.

- A new level could be introduced if at least y no. of peers are present in the current highest level and all 100% of them collaborate.

- For a new policy to be framed at least 75% of existing peers with trust value \( > x \) must collaborate. This ensures that malicious peers cannot frame new policies.

The initial policies are framed by the creator of the group and a hash of the global policy is also stored in the Group Charter.

### 4.1.2 Domain Specific Policy

This set of policies inherits the properties of the global policy and contains application specific policies. Say for example if we had a group for some multi-project software ecosystem and publish/subscribe etc, where peers could join as developers, reviewers, or moderators depending on their capability level. Further depending upon additional functionality each peer wishes to perform in the group a peer has different roles. A peer is allowed to elevate his level in the group based on some policy. Domain specific policies decide the optimum number of actors required
at each level. Maybe some policy could prune down no of peers in a particular level/role to optimize group performance. Another domain specific policy could be the framing of adaptive trust policies, i.e the group would work with trust framework TF1 until it reaches a particular state and then adapt TF2 and so on.

**Role Admission Constraints**

Conditions that need to be satisfied when a user requests to join a group in a particular role.

- List of users who can be allowed to join.

- Previous qualification requiring that the requesting user is currently admitted in some lower role.

- Role membership cardinality

- Events that must happen before a user is admitted in the role.

For e.g a maximal role peer in a software development workflow context could have the following admission constraints:

1. The no of peers in maximal role should be between 2 and 5

2. If a peer is a Moderator then he cannot take on the role of developer

3. At least one admission control peer must exist.

These constraints can be expressed in a declarative language as follows:

#members(Maximal_role)> 2 & #members(Maximal_role) < 5 & member(thisUser,Moderator) &!member(thisUser,developer) &#members(admission_control_peer)>0

**Co-ordination requirements** Co-ordination between peers in different roles within an activity is referred to as *Intra-role coordination*. For example the inter-role co-ordination requirement in a software development project can be that a verifier can test a software only after the developer has submitted the final product. When multiple users are there in a role it might be necessary to have some co-ordination between such as jointly doing certain tasks, known as *Intra-role coordination.*
4.2 Modeling Policies using Prolog

In dynamic policy driven TBAC, the policy description language besides being expressive and easy to use must be flexible enough to allow extension of a policy by adding or modifying its constraints easily, without affecting existing policies. We use a logic programming system to realize our access control policies. We capture the policies using Prolog [60] rules. A prolog rule is an expression of the form

\[ Ro(uo) :- R1(u1) \ldots \ldots Rn(un) \]

where \( R_i \) are predicates and \( u_i \) are (possibly empty) tuples of variables and constants. The head of the rule is \( Ro(uo) \) and the sequence of formulae on the R.H.S is the body of the rule. If a knowledge base contains a rule head \(-\) body, and Prolog knows that body follows from the information in the knowledge base, then Prolog can infer head. This fundamental deduction step is what logicians call modus ponens. There are only three basic constructs in Prolog: facts, rules, and queries. A collection of facts and rules is called a knowledge base (or a database) and Prolog programming is all about writing knowledge bases. That is, Prolog programs simply are knowledge bases, collections of facts and rules which describe some collection of relationships that we find interesting. One uses a Prolog program by posing queries. That is, by asking questions about the information stored in the knowledge base.

A policy is a set of Prolog rules. We chose Prolog as our policy description language because of the following features.

- It is declarative. A rule in PROLOG defines a relationship between several objects.
- It is based on a subset of First Order Logic, thus properties such as soundness and completeness can be guaranteed.
- It supports back tracking and can express non-deterministic constraints.
- It is a unification based language which allows writing policy templates.
- It is a productive modeling language supporting incremental policy writing and refinement.
- It is possible to reason from a set of Prolog rules, supports meta-level reasoning, thus making policy conflict detection possible.
• Dynamic rules can be modeled using the assert and retract clauses.

The Prolog inference engine provides a mechanism to derive consistent access control decisions at runtime [61]. It may also be used to analyze the correctness and consistency of access control and other rules. Authors in [97, 98] show how Prolog is formally used to verify the security policies of a multi-level secure high assurance computer system. Access control rules, defined in Prolog, can be more expressive than the traditional (subject, object, action) tuple. The access control policy is machine readable and directly under the control of the administrator. It also possesses dynamic updating capabilities. Dynamic rules can be modeled using the assert and retract clauses. New facts from independent policy sources can be added to the policy base before decisions are made, ensuring dynamic decisions at runtime. A policy interacts with its dynamic environment by consulting facts in the environment and constraining certain actions in the environment. In order to ensure that totally new policies outside the original group charter cannot be framed dynamically, we maintain a hash value of the original group charter with every peer. Thus a peer can at any time verify whether the policies being applied are as per the constitution framework.

We define here two sample policies for a software ecosystem for controlling access to review scores and model them using Prolog.

1. During submission phase a developer may submit a code c for project p

2. During review phase, reviewer r may submit a review for code c if r is assigned to review codes under project p.

3. During assessment phase, reviewer r can read scores for code c if r has submitted review for c.

4. Authors may never read scores.

The second policy varies in rules 2 and 3

• During review phase, reviewer r may submit review for a code c if r does not have any conflict of interest with c.

• During assessment phase, reviewer r can read scores for code c if r has submitted review for c and r is not conflicted with c.
If the application allows conflict of interest to change after code assignment, the semantic change is considerable, because if a reviewer who is initially assigned a code and submits a review, but later the moderator learns that the reviewer was conflicted with the code, then by first policy, the reviewer can read the scores of the conflicted paper.

The above policies are modeled in prolog as follows:

\[
\text{Allow}(d, \text{submit\_code}, c) :- d(\text{developer}), c(\text{code}), \text{phase}(\text{submission})
\]

\[
\text{Allow}(r, \text{review\_code}, c) :- r(\text{reviewer}), c(\text{code}), \text{assigned}(r, c)
\]

\[
\text{Allow}(r, \text{read\_scores}, c) :- r(\text{reviewer}), c(\text{code}), \text{has\_reviewed}(r, c), \text{phase}(\text{assessment})
\]

\[
\text{Deny}(d, \text{read\_scores}, c) :- d(\text{developer}), c(\text{code}).
\]

This policy governs the use of actions submit\_code, submit\_review, and read\_scores based on information from the environment. Predicates are Allow, Deny, has\_reviewed, assigned and so on. The environment in this case is the software ecosystem, credentials of end-users and run-time factors like current time, passage of time etc. Transitions in the policy’s environment could be triggered by various conditions.

To address privacy issues the domain plays an important role. Domain or context specific policies refer to the set of rules/regulations/requirements of a domain to which an entity must conform to in order to be in that domain. To implement privacy based access, a variable can be attached to the information to indicate the privacy level of information. So if full privacy is chosen then the information will be sent with encryption and signature. Domain specific policies are part of a dynamic policy database and could be modified by peers, depending on peer behavior, group composition and evolving trust of peers in the group.

All peers maintain their local\_view of the policies and any policy change is broadcast by the control peers. In the following subsections we give some sample rules for group activities like join and update.

### 4.2.1 Modeling Rules for Join

An example of rules for peer with unique identity to join as a admission peer in a group at level 1 of a group is

\[
\text{verify}(\text{Request}, \text{Rl}, \text{Level}, \text{Rate}, \text{SPR}, \text{Expert}, \text{Tot}) :-
\]

\[
\text{join}(\text{Request}), \text{admission}(\text{Rl}), \text{Level} = : = 1, \text{SPR} > 40.
\]

The verify predicate takes seven parameters which include, type of request(\text{Request}),
Role(Rl), Level, Trust Value(Rate), Self proclaimed rating(SPR), percentage of expert peers(Expert) and total number of peers(Tot) in the system. If the RHS is true for all given cases then the output will be true.

A more stringent join policy which checks ensures that the percentage of expert peers is greater than 20, for a total group size upto 200 is

\begin{verbatim}
RHS:
verify(Request,Rl,Level,Rate,SPR,Expert,Tot):-
    join(Request), admission(Rl), Level:=:= 2, SPR > 50, Expert > 20, Tot < 200.
\end{verbatim}

4.2.2 Modeling Rules for Level Update

Similarly rule for update request is:

\begin{verbatim}
RHS:
verify(Request,Rl,Level,Rate,SPR,Expert,Tot):-
    update(Request), maximal(Rl), Level:=:= 2, Rate ≥ 6, Expert < 20.
\end{verbatim}

Here a peer is allowed to update his level1 to level2 if he is a maximal peer(or control peer) and his trust value is greater than or equal to 6 with a restriction that percentage of expert peers should not be more than 20 %.

4.2.3 Modeling Dynamic Rules

Some examples of dynamic policies are:

1. If number of member peers is > 50% of Group size and Rate Of Job Completion is < some threshold t1, then change join policy for member peers and make it more stringent.

2. If no work done by peer X in last 30 minutes then remove access to resource.

3. If Rate Of Join in last 20 minutes is < some threshold t2, then change join policy (relax).

4. If Rate Of Update in last 20 minutes is > some threshold t3, then change update rule.

5. If too many updates as CP in last 20 minutes and percentage of CPs is greater than 40% of group size then change join rule and update rule for Control peers.

Multiple policies can apply to a group, and the system can switch between policies over time, influenced by the system’s state or environment. Some policies can have higher priorities than
others. Policies can apply to different levels of the system and can be global or apply to only parts of the system.

We are able to express dynamic rules at runtime, as well as change existing policies depending on the status of the group at runtime by exploiting the assert and retract clauses of Prolog. A rule is asserted in prolog when all the clauses on the L.H.S of assert are true.

The assert and retract clauses of Prolog facilitate the modeling of dynamic rules. One sample rule to prioritize join requests can be expressed as follows. Check the ratio of member peers to group-size and when the % of MPs is > 50 %, and the Rate of Job Accept is less than 30 %, then the previous join rule is retracted and a more stringent join rule is asserted into the database. Thus low level requests for join could be postponed in a group which already has a large number of peers performing in lower roles.

\[
\text{update}_{-}\text{engine}(\text{Tot}, \text{MPs}, \text{APS}, \text{CPs}, \text{RateOfAccept}) :\text{- RateOfAccept} < 30, \text{ MPs} > 50, \text{ Tot} > 200, \text{retract(old join rule),assert (new join rule).}
\]

The actual rule is written as
\[
\text{update}_{-}\text{engine}(\text{Tot}, \text{MPs}, \text{APS}, \text{CPs}, \text{RateOfAccept}) :\text{- RateOfAccept} < 30, \text{ MPs} > 50, \text{ Tot} > 200, \text{retract((verify(\text{UUID}, \text{Request}, \text{GR}, \text{Level}, \text{SPR}) :-join_{-}\text{req}(\text{Request}), \text{grouprole(\text{GR})}, \text{level1(\text{Level})}, \text{SPR} > 20, \text{assert(assignRole(\text{UUID}, \text{GR})), assert(assignLevel(\text{UUID}, \text{Level})))),}
\text{assert((verify(\text{UUID}, \text{Request}, \text{GR}, \text{Level}, \text{SPR}) :-join_{-}\text{req}(\text{Request}), \text{grouprole(\text{GR})}, \text{level1(\text{Level})}, \text{SPR} > 40, \text{assert(assignRole(\text{UUID}, \text{GR})),assert(assignLevel(\text{UUID}, \text{Level}))}})).
\]

The update_{-}engine predicate can be used to update the new rule by retracting the old rule and then asserting a new rule. Here update_{-}engine takes 5 inputs viz: MPs(% of member peers), APs(% of admission peers), CPs(% of control peers), Total(total number of peers) and RateOfAccept. Above rule states that if RateOfAccept is less than a threshold, and total number peers in the system exceeds a threshold of 200 and the percentage of MPs is greater than 50 % of total group size then update the previous join rule. Similarly other rules can be written for checking the number of control peers.

Another example of a dynamic rule is
\[
\text{update}_{-}\text{engine}(\text{Levels, MP}s, \text{ AP}s, \text{ CP}s, \text{ Total, RateOfJoin}) :\text{- Total} > 100, \text{ RateOfJoin} < t1, \text{retract(verify(\text{Request, RL, Level, Rate, SPR, Expert, Tot})).}
\]
join(Request), admission(Rl), Level\(=:= 1\), SPR > 40, Expert > 20,
assert(verify(Request,Rl,Level,Rate,SPR,Expert,Tot:-
join(Request), admission(Rl), Level\(=:= 1\), SPR\(>20\), Tot\(<200\).

Above dynamic rule, states that if the Rate of New Peer Join is less than some threshold \(t_1\)
and total no of peers in the system exceeds a threshold of 100, then replace it with a less string-
genjoin rule which permits join at SPR greater than 20, and removes restriction on minimum
number of expert peers, upto a group-size of 200 peers.

### 4.2.4 Modeling Constraints

In a role-based access control environment, there are six sets of entities that can be part of
a separation of duty constraint: users(U), roles(R), permissions(P), objects(O), types(T) and
actions(A). A peer role in our model is a union of Group Roles and Domain Roles. The Group
Roles are mutually exclusive and hence static separation of duty rules must be applied. That is
a peer in Group Role GR1 cannot perform roles GR2 or GR3 and so on. To be more specific a
Member Peer cannot simultaneously be an Admission Peer of a Control Peer. Further a member
peer can view resources of Admission peer but cannot modify resources. For example, in F/OSS
if a code is submitted for which the actions review and moderate are defined, we may wish to
constrain the application so that the same user (acting in the role of reviewer, say) cannot invoke
both actions on the same code. This is an example of a separation of duty constraint defined for
users and permissions.

In a workflow environment, it might happen that the review action for the specific code
must be invoked twice by different reviewers and by a role that is more senior than the role
that invoked the submit-code action. Alternatively, we may have an approve action and insist
that the permissions (code approve) and (code submit) are not assigned to the same role. To be
specific some of the constraints could be listed as follows:

1. If a peer is a member peer then he cannot be an admission peer or a control peer.
2. If a peer is in the role of an admission peer, he cannot be a control peer.
3. A control peer has access to all resources and can view or modify them.
4. An admission peer can view all resources.
5. A member peer cannot modify any resources.

We model some of these constraints as follows:

\[
\text{member(thisUser,MP) } \& \text{!member(thisUser, AP) } \& \text{!member(thisUser,CP)} \\
\text{member(thisUser, AP) } \& \text{!member(thisUser,CP).}
\]

\[
\text{can_view(Resource,CP)} \\
\text{can_view(Resource,AP)}
\]

\[
\text{can_modify(Resource,X) } \text{can_view(thisUser,Resource,X) } \text{:member(thisUser,MP), X=AP.} \\
\text{cannot_modify(thisUser,Resource,Y):-member(thisUser,MP), Y=AP.}
\]

**User Interface**

To facilitate the user, we have developed a GUI based interface for specifying the policies. The user need not write the Prolog rules. He only needs to specify the conditions and the actions in the interface and our tool converts it to prolog rules as described above.
Chapter 5

Case Study of F/OSS Collaborative group in our Framework

To model our P2P collaborative group we chose the self organizing group F/OSS [64, 65] which represents a group of like minded participants to develop software systems and related artifacts intended to be shared freely. Hundreds of F/OSS systems are now in widespread use by thousands of end-users, and some of them (e.g Mozilla Web browser, Eclipse) entail millions of lines of source code. Participants in this group could be core developers or code contributors, reviewers, module owners or moderators and so on. Tasks in the group could be development of software modules of different difficulty levels, content distribution, resource sharing, publish/subscribe and postings news-group. The participants communicate via online discussion forums, threaded email messages, newsgroup postings etc. F/OSS systems, hyperlinked artifacts and tools and project web sites serve as venues for socializing, building relationships and trust, sharing and learning with others. What emerges from this is a kind of multi-project ecosystem, where ongoing development and evolution of one F/OSS system gives rise to propagated effects, changes in one of more projects linked with it. Thus software evolution in a multi-project F/OSS ecosystem is a process of co-evolution of interrelated and inter-dependent projects, people, artifacts, tools, code and project specific processes.

In our model we segregate peers into different levels based on their level of contribution. For simplicity we map developers to level 1 in the group, reviewers to level 2, and moderators to level 3. In the absence of a centralized mechanism the challenges are recognizing good peers and differentiating them from bad ones, storing and accessing reputation values in a decentralized fashion, measuring the credibility of reported ratings, encouraging universal participation
and co-operation.

Most research on reputation mechanisms for use reputation values as a selection criteria and not as a reward criteria. Not only should good behavior be rewarded but bad behavior should be penalized. Our model provides a hierarchy of privileges so that peers are encouraged to exhibit good behavior to reach the upper layers of the community.

5.1 Task Allocation

The group has different tasks or modules at different difficulty levels. These tasks could be totally independent, sequential or concurrent or there could be some dependency among concurrent tasks. Say a peer could be allocated job C provided he has earlier met some pre-requisites. Task scheduling is done amongst peers based on peer capability, difficulty level of tasks and completion requirements. Consider a scenario where there are m peers with available resources needed by x peers for job completion. Each time K resources are generated by the system and each resource node can process only one job per time. If \( k \geq m \) then a job allocation policy could be

- Allocate jobs to resources depending on their trust values/roles without considering the complexity of jobs.
- Allocate jobs depending on the complexity level.
- Allocate jobs depending on deadline to finish.
- Allocate jobs depending on security policy, i.e whether it needs encryption or not.

We assume a Poisson distribution of task arrival. Tasks are modeled here as software modules of different difficulty levels easy, medium and difficult. Completion of modules and arrival of the same for review or moderation is also as per Poisson Distribution. Each module has an expected time of completion. The task allocation policy decides the task to be allocated to a peer depending on the complexity level of the task and the trust value of the peer. The Job allocation to peers is varied as shown in table 5.1. Alloc1, Alloc2 and Alloc 3 are the three task allocation policies. As indicated in the table, for easy tasks the trust value for Alloc 1 lies in the range of 0 to 40 while for Alloc 3 it is from 0 to 70.
5.2 Peer Roles

As specified in our generic model the role assigned to a peer is a union of Group Roles and Domain Specific Roles. The Group Roles are Member Peer(MP), Admission Peer(AP) and Control Peer(CP). The domain specific roles that we have considered here are Developers, Reviewers and Moderators. We map the Developers to level 1 peers, Reviewers to Level 2 peers and Moderators to level 3 peers. Peers at every level can take on a Role which is a union of the Group Role and domain specific Role. Some examples are:

- **R1 = MP ∪ Developer.** Peers in this Role are responsible for contributing new features and fixing bugs. They can contribute to the group by developing and submitting code, but have no access to view/modify any code/resources submitted by other peers. If a level 1 peer is assigned a particular module, it has no access to software/tools for other modules.

- **R2 = AP ∪ Developer.** These peers are Developers with additional functionality. They maintain certificates of peers, contribute to ratings of peers and contribute to join.

- **R3 = CP ∪ Developer.** These level 1 peers, do the tasks of submitting code and additionally do the Group management tasks like updating a peer, monitoring the group and so on.

- Level 2 peers or reviewers can view/review code submitted by Level 1 users belonging to a particular sub-group which they have been assigned. They could be assigned more than one sub-group depending on capability. They cannot modify code. The peers at this level too can take on the additional role of Admission Peer or Control Peer.

- Level 3 peers can modify code, reviews, fix bugs and take the final decision regarding code acceptance. Additionally they could take on Roles of Admission Peer and or Control Peer.
5.3 Modeling Events and Rules

Events, group roles and domain roles in the system are defined as prolog facts. The sets, Group role i.e $GR=\{\text{member, admission, control}\}$ and Domain role i.e $DR=\{\text{developer, reviewer, moderator}\}$ are defined as prolog facts as follows:

- grouprole(member).
- grouprole(admission).
- grouprole(control).
- level1(developer).
- level2(reviewer).
- level3(moderator).

We define different types of events like join, leave, update, submit-code, review-code and so on as prolog facts.

- req(join).
- req(update).
- req(leave).
- req(submit_code).
- req(review_code).

The rules are defined as predicates; We give below some rules for join in different roles.

- **Rule1**

  \[
  \text{verify}(\text{UUID}, \text{Request}, GR, Level, SPR) : - \text{join\_req}(\text{Request}), \text{grouprole}(GR), GR=\text{member}, level1(\text{Level}), SPR > 20, SPR < 40, \text{assert}(\text{assignRole}(\text{UUID}, GR)), \text{assert}(\text{assignLevel}(\text{UUID}, Level)).
  \]

  The verify predicate on L.H.S checks the rules on R.H.S after unifying the variables. In Rule 1 for example if the Request is of type join, Group role desired is member peer at level of developer then the join may be permitted if SPR value is between 20 and 40. A new predicate, assignRole is asserted into the database which assigns the group role to the said peer with ID, UUID.

  So if the input to the prolog engine is verify(peer1, join, member, developer, 30) then it would give a true output and the predicates assignRole(peer1, member) and assignLevel(peer1, developer) would be added to the database.
Rule 2

\[
\text{verify}(\text{UUID}, \text{Request}, \text{GR}, \text{Level}, \text{SPR}, \text{Vote}) :\text{-join}\_\text{req} (\text{Request}), \text{grouprole}(\text{GR}), \text{GR}=\text{admission}, \text{level1}(\text{Level}), \text{SPR} > 40, \text{SPR} < 60, \text{Vote} > 50, \text{assert}(\text{assignRole}(\text{UUID}, \text{GR})), \text{assert}(\text{assignLevel}(\text{UUID}, \text{Level})).
\]

Rule 2 and 3 are similar rules for join as admission peer and control peer in the domain role of developer, and Rule 2 also checks whether 50 percent of the total peers in the group have voted positively or not.

Rule 3

\[
\text{verify}(\text{UUID}, \text{Request}, \text{GR}, \text{Level}, \text{SPR}) :\text{-join}\_\text{req} (\text{Request}), \text{grouprole}(\text{GR}), \text{GR}=\text{control}, \text{level1}(\text{Level}), \text{SPR} > 60, \text{assert}(\text{assignRole}(\text{UUID}, \text{GR})), \text{assert}(\text{assignLevel}(\text{UUID}, \text{Level})).
\]

The predicates update_role and update_level are defined for updating the group role and domain role. In Rule 4 for example, if a peer gives an update request for the role of admission peer, update_role predicate checks whether the peer is currently in the lower role of member peer and whether his current acquired rating is greater than 5. If these conditions are met then it assigns the role of admission to peer and retracts the previous assigned role of member peer.

Rule 4

\[
\text{update}\_\text{role}(\text{UUID}, \text{Request}, \text{GR}, \text{Level}, \text{Rating}) :\text{-assignRole}(\text{UUID}, \text{member}), \text{update}\_\text{req}(\text{Request}), \text{admission}(\text{GR}), \text{level1}(\text{level}), \text{Rating} > 5, \text{retract}(\text{assignRole}(\text{UUID}, \text{member})), \text{assert}(\text{assignRole}(\text{UUID}, \text{GR})).
\]

5.3.1 Domain Specific Access Policy

One sample policy for the F/OSS application could have rules like:

- Developer d in Member role can be updated to reviewer r if he has submitted at least t codes and percentage accepted is greater than some threshold.

- Reviewer r can submit review for project c if he is assigned c and there is no conflict.

- Reviewer may be updated to role of moderator if he has successfully reviewed x projects.
A new level could be introduced if the number of maximal role peers at highest level exceed some threshold t and all agree to introduce a new level.

We model some of these rules using Prolog as follows:

\[
\text{updatelevel}(\text{CodesSubmitted}, \text{PercentAccept}, \text{Level}, \text{Role}) :\text{-member}(\text{Role}), \text{CodesSubmitted} > 6, \text{PercentAccept} > 50, \text{Level} := 1.
\]

\[
\text{updatelevel}(\text{CodesReviewed}, \text{PercentAccept}, \text{Level}, \text{Role}) :\text{-member}(\text{Role}), \text{CodesReviewed} > 10, \text{PercentAccept} > 50, \text{Level} := 2.
\]

\[
\text{updatelevel}(\text{CodesSubmitted}, \text{PercentAccept}, \text{Level}, \text{Role}) :\text{-admission}(\text{Role}), \text{CodesSubmitted} > 20, \text{PercentAccept} > 50, \text{Level} := 1.
\]

\[
\text{updatelevel}(\text{CodesReviewed}, \text{PercentAccept}, \text{Level}, \text{Role}) :\text{-admission}(\text{Role}), \text{CodesReviewed} > 20, \text{PercentAccept} > 70, \text{Level} := 2.
\]

The next Rule is a domain specific update for F/OSS application, which checks the number of codes submitted and percentage of acceptance before updating from developer role to reviewer role.

\[
\text{update\_level}(\text{UUID}, \text{Request}, \text{CodesSubmitted}, \text{PercentAccept}, \text{Rating}) :\text{-assignLevel}(\text{UUID, developer}), \text{update\_req(update)}, \text{CodesSubmitted} > 20, \text{PercentAccept} > 50, \text{Rating} > 6, \text{retract(assignLevel(UID, developer))}, \text{assert(assignLevel(UID, reviewer))}.
\]

**Modeling rules for task allocation**

A sample prolog rule for task allocation is

\[
\text{job\_req(Request,Difflevel,Rate,SPR)} :\text{-joballoc(Request)}, \text{Difflevel} := 1, \text{SPR} < 40, \text{Rate} < 40.
\]

**Modeling rules for access control**

The following rule processes submit-code requests for peers.

\[
\text{allow(PeerA,GRA,DRA,Request,PeerB,GRB,DRB)} :\text{- req(Request)}, \text{Request} = \text{submit\_code, grouprole(GRA), level1(DRA), grouprole(GRB), level2(DRB), not\_equal(PeerA,PeerB), entity(PeerA), entity(PeerB)}.
\]
This states that a peer belonging to a particular group role and domain role can submit code to a different peer belonging to any group role but a different domain role. Similar rules can be written for other actions like review code and moderate code.

The next rule defines two predicates not_equal() and not() which determine whether two identifiers are equal.

\[
\text{not_equal}(X,Y) :- \text{not}(X=Y).
\]

\[
\text{not}(X) :- X,!,\text{fail}.
\]

\[
true.
\]

5.3.2 Modeling Rules for System Model Change

When the group performance index GPI falls below a certain threshold the group can decide to change its system model by introducing new roles and levels. The following rule shows that when the code acceptance rate falls below 30 percent and total number of peers is greater than 400, the previous domain roles or levels are retracted and new levels are asserted into the database. The group then adapts a policy as per these new roles.

\[
\text{introduce_level}(Tot,CPs,CP\_vote,RateAccept) :- RateAccept < 30, Tot > 400, CPs > 30, CP\_vote > 90, \text{retract(level2(reviewer))}, \text{retract(level3(moderator))}, \text{assert(level2(reviewer1))}, \text{assert(level3(reviewer2))}, \text{assert(level4(moderator))}.
\]

Here CP\_vote > 90 ensures that at least 90 percent control peers agree to introduce a new level into the group, and change the workflow process so as to have two levels of reviewers. Thus we are able to model both dynamic policy change as well as system model change using prolog rules.

5.4 Overall Working of Framework with Prolog

Figure 2 shows the interaction between the events and policies in our framework. External events like join, leave, and internal events like level update, or application specific events like submitting a review, or submitting code, in case of F/OSS might trigger some other events, like change in policy. Change in policy in turn affects the way in which events like join, leave, update occur. All events are simulated in peersim simulator [59]. Simulator triggers the policy
file which is prolog rules. Database contains all the information about all the peers. Prolog rules use the database and according to the changes in database, prolog rules or policy might change or be modified.

5.4.1 Interplay of Statecharts with Prolog

We integrate the Statechart formalism with the declarative language Prolog. We read the diagrammatic elements of a given Statechart, together with the implicit state semantics as Prolog Rules. The static semantics includes the structure of a Statechart and consists of rules such as the following: (i) an OR state is active if and only if one of its substates is active, (ii) an AND state is active if and only if all of its substates are active. Declarative features of Prolog are used to define the rules for the behavior of the system within a state. For example we have join rules, leave rules, job allocation rules, access control rules and so on. We use the non-declarative features of prolog to depict the transitions between states. For example the system may start in a specific system model, which has a predefined set of domain roles, group roles, group policies ....all modeled in Prolog as facts and rules. On occurrence of external or internal events, state variables may change. Prolog rules are used to check the conditions and assert retract features of prolog are used to trigger a change of state. When the change of state does not involve modification of global variables like group roles, domain roles then it is a jump to a substate within
the same state where local policies like join-leave, job allocation change. However when the conditions require change of state with change of global variables then it is a change to a different state which could have different OR substates. Thus we use Prolog (both declarative and non-declarative features) to define logical rules for activities within states or along transitions. To depict transitions under dynamic conditions we use the assert retract features of Prolog. We use statecharts to reflect the behavior of the system and specify the control flow between the activities.

5.4.2 Architectural Model

Components of the model are Statecharts, Discrete event simulator and Prolog-enforcement engine. Figure 5.2 depicts the interplay between these components.

Our system starts in some initial state of the Statechart, in a specific system model. On arrival of an external event, the statechart gives input to the discrete event simulator about current state of system, values of state variables, guards etc. The simulator transfers the current control to Prolog engine. Depending on the event and the current state of the system Prolog policies are checked. If in the current state of the system, the conditions call for a local policy change represented by \((GH < t1)\), then the appropriate Prolog assert retract rules are applied and a
new local policy is asserted. Prolog transfers control back to statechart wherein the state is updated and system transits to a substate where the local policy rules have changed but the system model and global variables like no. of roles, no. of levels remain unchanged. If the group health is much lower than the required threshold \((GH < t_1)\) then Prolog uses assert retract rules to switch the system to a state with a new system model where the no of roles, levels and policies are different. The update_state transition takes the statechart to the new state. Internal events like update and leave also transfer control to the Prolog engine for appropriate action. If the Group Health is above the required threshold then the same state is maintained and control is passed back to the statechart module. Group Health is measured by a list of atomic conditions interpreted conjunctively. Each state transition can be implemented by the following scheme:

\[
\text{behave:-}\nonumber \\
\text{current state(S)}
\text{event(E)}
\text{test_condition(S,E,C)}
\text{schedule_action(S,E,C,A)}
\text{update_state(E,C,A,S,N)}
\]

where S is current state and N is next state. Update_state could take the system back to the same state in case of no policy change, to a substate in case of local policy change and to a state with a new system model in case of global policy change.

5.5 Sample StateChart Model for FOSS Application

Initially the system is in state S1 which has a static join policy with no restrictions and a static job-allocation policy say FCFS. System continues to be in this state as peers dynamically join and leave. Now suppose at time t the state variables and conditions are modified to

\(\text{GroupSize} > 100\) and \(\text{GHP} < t_1\) (t1 is predefined)

Further suppose \(#MPs > 50\%\) of GroupSize and \(#CPs < 10\%\) of GroupSize.

Group can no longer do the tasks optimally as there are very few CPs and APs. Further join of peers as Member peers should be postponed. Control peers switch the group to a policy JP2 which puts a restraint on the percentage of Member peers and the system transits to state S2
and enters S21. Global policy for this state specifies the number of roles and levels as 3 and specifies the actions in the task set as follows:

1. When code is submitted by a developer it is forwarded in parallel to two reviewers.
2. Reviewers review code and submit review to moderator.
3. Moderator accepts code if average rating is 7 out of 10.
4. The Trust Model to be used to calculate the ratings is TM1.

Now in state S21, as the events occur they are processed as per the new join policy and job-allocation policy FCFS. State variables are modified in response to events and when the condition occurs wherein rate of updates has reduced in last x minutes, the group can switch to a new policy for update (i.e state S22). In either state S21 or S22, if the rate of job-allocation falls below some threshold t2, thereby reducing GPI, then the system switches to sub-state S23 where new job-allocation policy JA-1 applies.

Thus the Control peers continuously monitor the group health parameters and the system switches between substates in state S2 with the local policy changing in each sub-state but the global policy remaining constant. However if the GPI falls beneath a certain threshold, and rate of job completion as well as average trust of peers both reduce then system switches to state S3 which
has a different workflow model. In this state the no of domain roles permitted is four i.e an additional reviewer role is introduced. The Action Set for tasks is as follows:

- Intermediate code is submitted by developer to 3 reviewers in parallel
- After feedbacks are received, code is modified and resubmitted to another set of 3 reviewers
- Moderators accept code if rating is 6 out of 10.

5.5.1 Statechart Representation in Prolog

A statechart can be represented as a set of Prolog facts[103]. We represent our sample FOSS statechart shown in figure 5.3 as prolog facts as shown below.

Initially the system is in state S1 which has a static join policy with no restrictions and a static job-allocation policy say FCFS. The default state in S1 is S11.

statediag(S1)
state(S11,S1)
state(S12,S1)
default_state(no_action,S11,S1)

This represents the fact that S1 is an exclusive state having substates S11 and S12. S11 is a default sub-state and has no associated action when it is started.

From State S1 depending on the event triggered and on the current state of the variables, the system could either transit to sub-state S12, if only a local policy change is needed or it could transit to state S2 or S3 having different system models and different policies.

statediag(S2)
state(S21,S2)
state(S22,S2)
state(S23,S2)
state(S24,S2)
default_state(no_action,S21,S2)
Exclusive state S2 has four substates.

statediag(S3)
state(S31,S3)
state(S32,S3)
default_state(no_action,S31,S3)

The different kinds of transitions are represented as follows:

1. transition(event,cond,no-action,State,State)
2. transition(event,cond,action,State,substate)
3. transition(event,cond,action,State,NextState)

The first one represents a transition to the same state hence no action. The second one represents a transition to a substate within a state and the third one represents a transition from a source state to a destination state when an event occurs and the conditions match.

A state is an abstraction of the attribute values and relationships of an object and hence specifies the response of the object to events. Estaban Zimanyi [103] has shown how Prolog can be used in the construction of a system supporting the task of dynamic specification. The operational semantics of statecharts can be reflected in a declarative way by using Prolog predicates to implement the statecharts’ functionality. Statecharts are attached to classes with a predicate attached. For example attached(SM1,S,main) realizes the link between the class SM1(system model 1) and its statechart S, where main is the outermost state. We start an object of a class by starting its associated statechart. We give here an example of predicates used for arriving at and leaving states.

When arriving at a state, its entry action(if any) is executed. The predicates \texttt{mark\_arrival\_state}\ and \texttt{arrive\_substates}\ are used to depict arrival at a state or recursive arrival at substates.

\begin{verbatim}
arrive_state(Obj,State) :- exec_entry_action(Obj,State),
mark_arrival_state(Obj,State),
arrive_substates(Obj,State).
\end{verbatim}
Simulate statechart transitions:

\[
\text{leave state}(\text{Obj}, \text{State}) : -\text{leave substate}(\text{Obj}, \text{State}), \text{mark leaving state}(\text{Obj}, \text{State}), \\
\text{exec exit action}(\text{Obj}, \text{State}) .
\]

5.6 Simulation of FOSS Application framework

We simulated a typical F/OSS application with core developers, reviewers, documenter, moderators etc. For simplicity we mapped core developers of code to Level1, reviewers to Level 2 and moderators to Level 3. Peers join the group either as Developers, Reviewers or Moderators and assume roles of Member peers, Admission Peers or Control peers depending on their functional capability. A peer in Developer Level for e.g can be updated to reviewer based on his performance and reputation earned in the group. At level 1 member peers develop code, while control peers do additional administrative tasks. The group permits peers to collaboratively work on different projects, and efficiency of the group is based on completion time, quality and price of the different projects.

5.6.1 Implementation details

Implementation of the statechart formalism has been done using a discrete event simulator. The simulator begins in some state with a set of peers, a set of policies, a value for group health, some measured values of state variables, and a workflow defined as a system model. The input to the simulator is external events which could be peer join requests or jobs. Peers requesting to join have some self proclaimed rating and some intrinsic capabilities. At the same time there could be internal events occuring in the simulator or in the state in which the system is in. Internal events are update requests, policy change requests, leave requests and so on. Either external or internal events could trigger some guard conditions, which in turn trigger an action like a policy change, causing the system to change its state and transit to either a substate or a new state. This transition and the dynamic policy change or system model change is achieved through the assert retract features of Prolog. The java code of our discrete event simulator initializes the prolog engine and loads the prolog file. Once the prolog file is loaded java objects can be passed to prolog and verification is done according to our prolog rules.
5.6.2 Peer Behavior

For simplicity we assume that the Member peers behave as OK peers, Admission peers behave as average peers and Control peers behave as expert peers while providing services as well as while rating other peers. If a control peer is a developer, probability of bugs is very less, code quality is good and submission time is less. If he is a reviewer then ratings are accurate. Further we assume that OK peers develop code with a certain higher percentage of bugs, at a slower rate as compared to Average and Expert peers. Similarly while rating, the Expert peers give more accurate ratings as compared to OK and Average peers.

System Model

We define two system models for the F/OSS application. In the first model $SM_1$ we define three group roles i.e Member Peer, Admission Peer and Control Peer and three domain specific roles viz developer, reviewer and moderator. In the second model $SM_2$ an additional reviewer role is included in domain role.

5.6.3 Simulations and Analysis

We performed different simulations to prove the validity and capability of our framework.

The simulator starts with some initial peers in the group, and then events such as join, leave, update are triggered. The events have been triggered in the event mode of peersim. Another event which is triggered is the entry of different job modules into the group. A module could have different difficulty levels associated with it such as easy, moderate or complex. A module has parameters like Starting time, Current time, Expected Time of completion, Bugs, and Difficulty Level. Peers are assigned modules based on the current active job allocation policy. A module can have more than one peer working on it at a time. As soon as a module is finished, those peers will be allocated some other modules to work on. A priority queue is maintained based on the current time of start of a module so as to maintain the concurrency between the modules. Time is calculated on the basis of Poisson distribution where the lambda is the average time needed by a peer. Simulation runs until all modules of the project are completed. During the simulation, new peers can continuously join the group at any level depending on join policy and existing peers can also apply for update of levels. Dynamic leave of peers is also simulated. As a new peer joins he is allotted to a new module or a currently running
module based on the allocation policy.

In the initial set of experiments we tested the behavior of peers with varying join policies and varying job allocation policies. Various runs show that the group performance varies with changing group composition and evolving trust of peers. Thus a policy which may give good results initially may not be viable after the group size increases or decreases drastically. We then show that when peers have the flexibility to change the policies dynamically depending on various group parameters the group gives optimum performance.

In the next set of experiments we gradually inject malicious peers into the network. We conducted the same set of runs with varying policies in presence of 10 percent and 30 percent malicious peers. It was observed that the dynamic policy gave better results than the static in the presence of 10 percent malicious peers. However one surprising observation was that when the malicious peer percentage was increased to 30 percent the group took more time to converge in the presence of the dynamic policy except when all easy jobs were allowed to enter the system. This allowed us to conclude that the dynamic behavior has to change depending on the type of peers and their trust value acquired while performing in the group as well as the type of jobs entering the system. So while a policy which switches to a group composition of 40-30-30 when peers increase from 200 to 300, would work fine for a group with no malicious peers, the same policy may not give good results in the presence of 30 percent malicious peers. Thus only dynamically switching between different policies are not effective in optimizing group performance. The process workflow or the system model of the group needs to be changed at times to achieve the desired optimum results.

In the next set of experiments we test the behavior of groups in presence of different system models. The initial group charter specifies some process workflow which we define as a system model and peers behave according to that model and change the policies dynamically as in the first set of experiments. We go on to show that under specific conditions just the change of a policy does not lead to optimum performance and it is desirable that the group changes its process workflow and adapts to a different system model. Our tool permits the peers to collaboratively decide and switch between system models when required. The details of the experiments are given below.
Simulation 1:

In the first set of experiments we ran the simulations for an initial 40 peers and then allowed 100 peers to join. The job allocation policy here was a simple first come first served (FCFS) policy and we allowed 300 jobs to enter the system. We tested various join policies. In one of the policies we permitted peers to join without any constraints. Then we ran the tests by putting constraints on the compositions of different types of peers. It was observed that when the peer composition was 40-40-20 i.e 40 member peers, 40 admission peers and 20 control peers out of a total of 100 peers, then initially the peers attain a good job success rate but with increase in number of transactions and number of peers, the time taken increases considerably. On the other hand the composition of 20-60-20 gives good results as no. of peers increases. We repeated this for different join policies by putting constraints on the number of control peers and admission peers in each policy. We increased the total peer join to 300 peers. Various runs allowed us to come to the conclusion that depending on the type of peers entering the system (which can be measured by their trust values), the group performs better with 20 percent control peers upto a certain threshold of total peers, beyond which it performs better if control peers are increased to 30 percent. We then gave runs with a dynamic join policy wherein the policy has no restriction on control peers upto a total of 100 peers in the system, then switches to a policy of 40-40-20 from 100 to 200 peers and to a policy of 40-30-30 from 200-300 peers.

In the dynamic run, the system initializes in state $S_2$ with system model $SM_1$ with 40 peers initially. In the initial state $S_{21}$, the join and update policies are policies with no constraints on group composition. The job allocation policy is FCFS. As the group evolves the control peers monitor the different parameters of group health. When the RateOfJoin of member peers increases beyond a certain threshold and RateOfAccept decreases and the condition, group_size $> 100$ && MPs $> 50$ is met, the system switches to a new state $S_{22}$ where the join and update policies are more stringent and ensure that at any time there are at most 50 % member peers. Similarly if RateOfJoin decreases below a certain threshold, system switches back to a state with a relaxed join policy.

Graph in figure 5.3 shows the plot of Percentage task completion versus Time (X-Y) for 4 different runs for varying join policies and constant job allocation policy viz FCFS. Results show that the dynamic join policy($dy_ja_{100}.txt$) gives better results as expected.
Simulation 2:

In the second set of experiments we tested the system for varying job allocation policies. We designed 3 job allocation policies viz job alloc1, job alloc2, and job alloc3. In job alloc 1 when a peer enters the system if his SPR is less than 40 then an easy job is allocated to him, if his SPR lies between 40 to 60 then a moderate job is allocated and if his SPR is greater than 60 then a difficult job is allocated. Further in case a peer with SPR less than 40 enters and there is no easy job in the queue then the next level complex job will be allocated and so on. Thus a peer is not kept idle at any time and all jobs are allocated.

In policy job alloc2 again the jobs are allocated as per the values in table. However here if a peer with SPR less than 40 enters the system and there is no easy job in queue then that peer is kept idle and the next peer is fetched. When all peers are fetched, the system checks for idle peers to allocate jobs. Previously allocated peers whose trust value might have increased, can now be allocated the difficult jobs. Thus job allocation in this policy is done exactly according to the access rights of a peer.

Policy job alloc3 is a combination of the above two policies. A peer with less SPR is not allocated a difficult job but a peer with higher rating may be allocated a moderate or easy job if a difficult job is not in queue.

A sample prolog rule for above policies is

\[\text{job_req}(	ext{Request}, \text{Difflevel}, \text{Rate}, \text{SPR}) :- \text{joballoc}(	ext{Request}), \text{Difflevel} = := 1, \text{SPR} < 40, \text{Rate} < 40\]
We started the simulator with 100 peers in the system and allowed a maximum of 300 peers to join. We defined a project which contains 1000 modules defined as jobs of different complexities, namely easy, moderate and difficult. The group chooses a join policy which ensures that there are 30% ok peers, 40% average peers and 30% expert peers.

We then gave runs for different job compositions and different static job allocation policies. As observed in fig 5.6, when maximum difficult jobs entered the system (job composition 10-10-80) then job_alloc1 gave better results since difficult jobs were distributed amongst all peers, whereas when percentage of easy jobs was dominant (job composition 80-10-10) then job_alloc3 gave better results as seen in fig 5.5.

Graph in fig 5.7 is the output when the job allocation policy is varied dynamically, by control peers by switching between policies depending on job composition taking the group from state S22 to S23. It shows that this dynamism in job allocation allowed the group to perform better.

In the previous experiments we assumed that the peers are honest about their capabilities and declare a correct value of self proclaimed rating i.e SPR. However that may not always be the case. In the next experiment we show that even if peers falsely proclaim good initial ratings they are eventually spotted and their average trust values are decreased. Graph in figure 5.8 is a plot of the average current rating of a peer. Q indicates the current rating as computed by the group and SPR is the self proclaimed rating declared by the peer at the time of join.

It is observed that for peers who over estimate their initial rating, the group eventually de-
creases their average trust value. Under estimation of initial trust value gives better performance than over estimation.

**Simulation 3**

In this set of experiments we gradually injected malicious peers into the group. We modeled malicious peers as peers who do not give satisfactory behavior. As coders they submit code with large no of bugs and as raters they always give wrong ratings. We gave different runs for 10 percent and 30 percent malicious peers.
Figure 5.8: Average current rating versus time taken

Figure 5.9: Static vs dynamic policy in presence of 10 percent malicious peers

It was observed in fig 5.9 that in presence of 10 percent malicious peers the dynamic policy gave better results than static.

However when the malicious peer percentage increased to 30 percent and above then the same set of dynamic policies made the group take more time to converge as seen in figure 5.10 and 5.11.

Graph in figure 5.12 shows that when all easy jobs entered the system the dynamic policy worked better than the static in presence of both 10 percent and 30 percent malicious peers.
Simulation 4

Here we tested the performance of the group in presence of varying system models. System model $SM_1$ has three domain roles viz. developer, reviewer and moderator whereas in system model $SM_2$ we define an additional domain role of an intermediary reviewer. The type of jobs entering the system for the run was 10 percent easy jobs, 80 percent moderate jobs and 10 percent difficult jobs. The system is initially in state S2 with system model $SM_1$. As the group evolves, peers with varying capabilities join and different complexities of jobs also enter the system. Based on their performance the peers are rated and they accumulate a trust value which is periodically measured. We observe with different experiments that depending on the group
Health, specifically the parameters like rate of code accept, average trust value and rate of peer join, the group efficiency improves with an additional level of reviewers when the code acceptance rate falls beneath a certain threshold. Graphs in figure 5.13, and 5.14 show the behavior of the group in presence of different policies and differing job compositions for different system models. The notations used in these graphs are as follows:

Names beginning with sr indicate system model with single level of reviewing and those beginning with dr indicate system model with double level of reviewing. In fig 5.15 ”sroutput.txt” is the plot for system model $SM_1$ with single review level and ”droutputpol2.txt” is the plot for system model $SM_2$ with double review level. The third plot viz $dya_dlr_sr.txt$ is of the run when dynamic switching was done from system model with single review level to system model with double review level after 30 percent job completion as code acceptance rate decreased beneath the defined threshold. Once in this new state the peers dynamically adapt to different join and update policies as before, which are now defined for this system model.

Since adding an additional level of reviewing leads to considerable increase of cost, we plotted the cost metric along with time metric. Graph in figure 5.16 is a comparative plot of the system behavior with system models $SM_1$ and $SM_2$ in presence of moderate jobs. The thin lines are a plot of time taken versus percentage completion of jobs and the boxes indicate the cost associated with percentage completion of jobs. It is observed that for 100 jobs with job composition 10-80-10, the time taken to complete jobs in single review model is more as compared to double review model. On the other hand the cost associated with double level of
reviewing is more as compared to that with single review level.

Figure 5.17 shows the behavior of the system with dynamically changing system model. It was observed that for policy 2 with 100 projects in the system and 600 peers, the run with double reviewers completed faster as compared to the run with single reviewers.

The graph in figure 5.17 shows the behavior of the system with dynamic switching between the models. The group starts in system model $SM_1$. After about 40% job completion, since the group performance index (GPI) is less than the required threshold the group switches to another model $SM_2$ resulting in more optimum performance, and then again switches back to model $SM_1$ after around 80% job completion to balance the cost. The cost associated with two levels of reviewing is much more as indicated hence the group tries to maintain a balance.
between time taken and optimum cost.

**Simulation 5**

In the next set of experiments we increased the number of projects to 200 and gave runs for different policies. The graph in figure 5.18, is a plot of time and cost metric against percentage task completion for 200 projects with job-allocation policy 2.

In this case the time taken for double review model is less than that for single review model.

Graph in figure 5.19 shows that when policy 1 is applied the system behaves differently
5.6.4 Implementation Details

We modeled the P2P groups using a java based overlay simulator namely PeerSim[59] which is a Peer-to-Peer simulator. We chose Interprolog as our bridge between Java and Prolog as it directly loads the prolog file from java. The global and domain specific policies were implemented using SWI-Prolog which we integrated with Peersim at runtime wherein java and prolog talk to each other by means of sockets.

and double review takes more time for completion as compared to single review.
Interprolog

InterProlog [62] is middleware for Java and Prolog, providing method/predicate calling between both, either through the Java Native Interface or sockets; the functionality is basically the same in both cases. InterProlog’s innovation to this problem is its mapping between (serialized) Java objects and their Prolog specifications. Java Serialization API does the most of the work on java side while prolog side is built upon DCG that analyses/generates serialized objects. A definite clause grammar (DCG) is a way of expressing grammatical relationships. It has small part dedicated to each of the prolog systems (XSB and SWI) and each Prolog system has a specific PrologEngine subclass. To use this one has to first install XSB or SWI Prolog. We have used XSB Prolog [63].

Example Java Program

This is the sample Java code which initializes the prolog engine and loads the prolog file. Once the prolog file is loaded java objects can be passed to prolog and verification can be done according to our policy rules.

```java
PrologEngine engine = new XSBSubprocessEngine("......./xsb");
File f = new File("verify.P"); engine.consultAbsolute(f);
// or consultRelative (to the class location), or consultFromJar(to the jar location),...
Object[] var = new Object[4,1];
boolean b = engine.deterministicGoal("ipObjectSpec('java.lang.Integer',Obj1,[Peer],_", " +
```

Figure 5.19: Cost metric vs time metric for 200 projects with policy1
\texttt{ipObjectSpec('java.lang.Integer', Obj2, [Level], \_), belongs(Peer, Level\_), \"[Obj1, Obj2]\", var); System.out.println(b);}

**Assumptions and Parameters**

The following table gives the probability distribution of the no. of bugs with the type of peer considered for our simulations:

<table>
<thead>
<tr>
<th>No. of bugs/Type of peer</th>
<th>Expert Peer</th>
<th>Average Peer</th>
<th>OK Peer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5</td>
<td>0.33</td>
<td>0.25</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.33</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>0.33</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Table 5.2: Probability of distribution of bugs with type of peers

While initializing the simulation we have fixed the reputation/trust value of the peers at different levels as shown in table 5.3:

<table>
<thead>
<tr>
<th>Trust Value</th>
<th>Expert Peer</th>
<th>Average Peer</th>
<th>OK Peer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.8</td>
<td>0.6</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table 5.3: Default trust value of peers

**Time Calculation**

Since the time of execution of a peer to complete a module depends on the type of job and peer’s MIP value the following assumption has been made for execution time:

\[ time = 1 + c \times \frac{100}{MIP} \]  \hspace{1cm} (5.1)

For job of level 2,

\[ time = 3 + c \times \frac{100}{MIP} \]  \hspace{1cm} (5.2)

For job of level 3,

\[ time = 4 + c \times \frac{100}{MIP} \]  \hspace{1cm} (5.3)

Where \( c \) is some constant. The average MIP values assumed were
1. Avg MIP value for
   - level1 job = 50
   - level2 job = 70
   - level3 job = 90

2. Average time taken by an ok peer to complete level1 job = Average time taken by an average peer to complete level2 job = 7 in this case.

### 5.7 Summary

Our experiments show that we have a framework capable of testing the evolution of dynamic peer groups. The behavior of dynamic peers in collaborative groups cannot be predicted beforehand. Depending on the application, the type of peers joining, the behavior of the peers once they are part of the group, our framework provides the flexibility to the user to chose the right set of policies for admission and access control. Users can test the behavior of the group in presence of different admission and access control policies. The framework permits the group to start in any system model, and switches between different system models to achieve efficiency. Dynamic switching of policies and dynamically adaptive system models helps the group to achieve optimum efficiency.
Policy can be defined as the specification of the behavior of a system. Policy has long been employed in the management of traditional distributed systems and many policy frameworks have been proposed. There have been a number of efforts evaluating various aspects of different policy frameworks for different application scenarios.

Since long logic programming and rule-based reasoning have been proposed as a basis for policy specification languages. Policy encompasses different notions like:

**Security Policies** pose constraints on the behaviour of a system. They are typically used to control permissions of users/groups while accessing resources and services.

**Trust Management** policy languages are used to collect user properties in open environments, where the set of potential users spans over the entire web.

**Action Languages** are used in reactive policy specification to execute actions like event logging, notifications, etc. Authorizations that involve actions and side effects are sometimes called provisional.

**Business Rules** are "statements about how a business is done" and are used to formalize and automatize business decisions as well as for efficiency reasons. They can be formulated as reaction rules, derivation rules, and integrity constraints.

All these kinds of specification interact tightly with each other: Credential based user properties are typically used to assign access control permissions; logging, monitoring, and other actions are part of the high-level security specification documents of many organizations;
many business rules—e.g., for granting discounts or special services—are based on the same kind of user properties that determine access control decision. Moreover, this kind of business decisions and access control decisions are to be taken more or less simultaneously—e.g. immediately before service access.

There has been extensive research focusing on each of these different notions of policies. Figure 6.1 shows that these different types of policies can all be represented using simple declarative languages like Prolog.

![Figure 6.1: Chart Showing Different Policy Languages](image)

### 6.1 Rule Based Policy Language frameworks

Security Policies are best expressed by rule based languages. Security Policies may change with time and rule based policy languages should be able to express time dependent behaviour. Main advantages of rule-based policy languages are as follows:

- Users spontaneously tend to formulate security policies as rules
- Rules have precise and simple formal semantics which is an excellent help in implementing and verifying access control mechanisms

- Rule languages can be flexible enough to model in a unified framework the various policies introduced.

Logic programming languages satisfy all the above properties and have an efficient inference mechanism, hence are popular as rule based policy specification languages.

KAoS, Ponder and Rei are frameworks based on active rule based constructs. They can formulate obligations that cause an agent to execute some action whenever a specified event triggers the rule and some pre-condition is satisfied.

Ponder [107] allows general security policies to be specified as a set of rules that define a choice in the behavior of the system. It is a declarative object oriented language which allows definition of positive and negative authorization policies. Obligation policies are event triggered. It provides a group construct for group related policies and a Role construct for role policies. It describes meta-policies as policies about policies.

Rei [107] allows actions and policy objects to be defined separately and allows them to be dynamically linked to subjects. It also defines meta-policies like Ponder. It does not distinguish between role-based, group based and individual policies, allowing them to be described using the same set of constructs, leading to simpler policies. The Rei framework does not provide an enforcement model and runtime changes to the policy are not possible.

### 6.2 Trust Management policy languages

"Trust management problem” is defined as a collective study of security policies, security credentials and trust relationships. The solution proposed is to express privileges and restrictions using a programming language. They are further classified as credential based and role based.

#### 6.2.1 Credential Based Trust Management

PolicyMaker [105] is designed to provide a general trust management framework across a wide range of applications at some expense of efficiency. It is a credential based framework which
has additional power as compared to SPKI/SDSI certification framework. For any trust policy which SPKI cannot express, PolicyMaker provides a language. PolicyMaker allows use of any programming language. Keynote is a simple trust management system for PKI. It has a unified approach to specifying and interpreting security policies, credentials and relationships. Trust management credentials bind keys to authorization to perform specific tasks. Keynote predicates are written in simple notation based on C-like expression and regular expressions. Uses a specific assertion language.

It is important to note that neither PolicyMaker nor KeyNote enforce policies but give advice to applications that make calls to them. It is up to the calling application whether to follow their advice or not.

6.2.2 Role-Based Trust Management

The RT framework [104] is a set of languages for representing policies and credentials. It uses roles to represent attributes and is specially suited for decentralized collaborative systems. Cassandra [40] is a role-based trust management system. It uses a policy language based on Datalog with constraints and its expressiveness can be adjusted by changing the constraint domain. Policies are specified using predicates like permits(e,a), canActive(e,r), has Activated(e,r) and so on. Policy managers can define and use new predicates as they need.

6.3 Desirable features of policy specification frameworks

A complete policy framework should include a policy specification language to allow for the expression of policies, a policy deployment model for the distribution of policies into enforcement points, a policy analysis mechanism for making the correct policy enforcement decisions, and a policy monitoring and enforcement mechanism to allow for the identification and execution of policy actions. Some of the desirable features are:

- Language’s formal semantics and extensibility. A policy language should also be easily extensible to support new types of policy expressions as needed. In this regard, a language with general constructs that can be combined to express different types of policies is preferred to a language which is a collection of predefined types of policy.
• Domains and other forms of grouping. A policy framework should have a mechanism to support the grouping of elements in a managed system so that policy can be systematically applied to them. The support for domain is most desirable as domains allow for the hierarchical grouping of objects. In addition, policies can themselves be managed objects thus policy grouping should also be enabled so that management actions can be performed on a policy set rather than on individual policies.

• Distributed policy enforcement. Distributed policy enforcement increases the flexibility and scalability of the framework. Traffic and the impact of a central point of failure can both be minimized when policy enforcement and some of policy decisions can be made locally.

• Meta policy (or policy about policies) allows rules and constraints (such as timing constraints, prioritization rule) to be placed on policies themselves thereby facilitating policy analysis and conflict resolution.

6.3.1 Comparative evaluation

Based on desired features mentioned in above section, we identify some parameters for evaluation of different policy specification frameworks and do a comparison of IETF, KAoS, Rei, Ponder and our Dynamic policy based framework. The criteria chosen are as follows: i) Specification Language ii) Tools for policy specification, iii) Reasoning support, iv) Languages’ formal semantics and extensibility v) Domains and other forms of grouping vi) Meta-policies vii) Dynamic policy updates viii) Dynamic System Model ix) Ease of use x) Enforcement Mechanism
<table>
<thead>
<tr>
<th>Policy Framework Features</th>
<th>IETF</th>
<th>Ponder</th>
<th>KAoS</th>
<th>Rei</th>
<th>Our Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy specification Language</td>
<td>Specific policy language not defined</td>
<td>Ponder - Declarative specification</td>
<td>DAML OWL semantic language</td>
<td>Prolog like syntax plus RDF, Declarative plus semantic</td>
<td>Prolog - Declarative</td>
</tr>
<tr>
<td>Tools for policy specification</td>
<td>nil</td>
<td>Graphical editor and compiler</td>
<td>KPAT graphical editor for ontology and policy management</td>
<td>Under development</td>
<td>Simple graphical editor and converter from text to Prolog</td>
</tr>
<tr>
<td>Reasoning support</td>
<td>Event calculus</td>
<td>Java Theorem prover</td>
<td>Prolog engine</td>
<td>Prolog engine</td>
<td>Prolog engine</td>
</tr>
<tr>
<td>Meta policies</td>
<td>No direct support for meta policies</td>
<td>Yes</td>
<td>Partial</td>
<td>Yes</td>
<td>yes</td>
</tr>
<tr>
<td>Dynamic policy update</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>yes</td>
</tr>
<tr>
<td>Language's formal semantics and extensibility</td>
<td>Nil</td>
<td>Partial</td>
<td>Yes</td>
<td>Yes</td>
<td>yes</td>
</tr>
<tr>
<td>Domains and other forms of grouping</td>
<td>Yes</td>
<td>Yes - group construct for group related policies and role construct for role policies</td>
<td>Partial</td>
<td>Partial - No distinction between grp based, role based and individual policies</td>
<td>yes</td>
</tr>
<tr>
<td>express dynamic system model</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ease of use</td>
<td>Language specifically designed for simple policy specification and direct readability</td>
<td>Need of specialized tools for policy specification and interpretation</td>
<td>Need of specialized tools for policy specification and interpretation</td>
<td>Simple policy specification</td>
<td></td>
</tr>
<tr>
<td>Enforcement mechanism</td>
<td>Partial</td>
<td>Simple policy enforcement mechanism</td>
<td>Sophisticated policy automation mechanisms for enforcement</td>
<td>Does not provide an enforcement model</td>
<td>Simple policy enforcement mechanism</td>
</tr>
</tbody>
</table>

Table 6.1: Comparision Of Features Of Policy Specification Frameworks
Chapter 7

Trust Model

In collaborative systems, since peers are heterogenous, some peers might provide benevolent services while others may be buggy and may not provide service as they advertise or as per expectations. Some might be malicious and provide bad services. One of the main concerns regarding reputation systems is their effectiveness in coping with peers which maliciously change their behavior[19]. For instance, a malicious peer could provide good services for a while, building a good reputation, and then suddenly start cheating and exploiting that reputation. Thus a malicious peer who behaves honestly for low-valued transactions could defunct for an important transaction. Moreover, malicious peers could even oscillate between periods of good behavior and bad behavior.

7.1 Threat Model in P2P groups

Anonymous nature of peer-to-peer (P2P) systems exposes them to malicious activity. Establishing trust among peers can mitigate attacks from malicious peers. Peers trust each other to perform operations such as routing file search queries and downloading/uploading files. However, a malicious peer can use the trust of others to gain advantage and can harm the operation of a system. Detecting malicious behavior is difficult without collaboration. However, feedbacks from peers might be deceptive, and thus, identifying a malicious peer with high confidence becomes a challenge. The two primary types of adversaries in peer-to-peer networks are selfish peers and malicious peers[25]. They are distinguished primarily by their goals in the system. Selfish peers wish to use system services while contributing minimal or no resources themselves. A well-known example of selfish peers are freeriders in file-sharing networks, such as
Kazaa [28] and Gnutella [29]. To minimize their cost in bandwidth and CPU utilization free-riders refuse to share files in the network. The goal of malicious peers, on the other hand, is to cause harm to either specific targeted members of the network or the system as a whole. To accomplish this goal, they are willing to spend any amount of resources. Examples include distributing corrupted audio files on music-sharing networks to discourage piracy or disseminating virus-infected files for notoriety. Some of the adversaries are[73]:

1. Honest Peers. These peers are not malicious. However they try to achieve monopoly earning a high trust by giving best service always. Could prove to be a threat if it later becomes malicious

2. Naive peers. These peers try to gain trust by promising good services but always provide bad services

3. Hypocritical peers. Some malicious peers may behave properly for a period of time in order to build up a strongly positive reputation, then begin defecting. This technique is effective when increased reputation gives a peer additional privileges, thus allowing malicious peers to do extra damage to the system when they defect. They could be random or structured. Random peers act maliciously with some probability say 20%. Structured peers follow a fixed pattern, say provide one malicious service after four good ones.

4. Colluding peers. Peers could collude and form a special group. They act differently depending on whether requesting peers are from within the group or outside it, and normally provide malicious service to an outsider.

5. Front peers. These malicious colluding peers always cooperate with others in order to increase their reputation. They then provide misinformation to promote actively malicious peers. This form of attack is particularly difficult to prevent in an environment where there are no pre-existing trust relationships and peers have only the word and actions of others in guiding their interactions [25].

6. Whitewashers. Peers that purposefully leave and rejoin the system with a new identity in an attempt to shed any bad reputation they have accumulated under their previous identity [18].
7.2 Background

As defined by Marsh[21][72], Trust is the firm belief in the competence of an entity to act independently, securely and reliably within a specified context. Mui [71] defines trust as the subjective expectation an agent has about another agent’s future behavior. Thus the trust A has in B may differ from the trust C has in B. We adopt the definition of trust as proposed by Mui. Reputation as defined by Aberer and Despotovic [20] is a measure that is derived from direct or indirect knowledge on earlier interactions of agents and is used to access the level of trust an agent places in another agent. Recommendation trust therefore is a measure of how much trust peer A can place in the experiences reported by peer B.

A peer should build two kinds of trust in another peer

1. The trust peer A has in peer B’s competence in providing services.

2. The trust peer A has in peer B’s reliability in providing recommendations about other peers.

Here reliability includes two aspects viz: Truthfulness i.e whether B gives true information and Similarity i.e whether B is similar to A in preferences and ways of judging issues. Since peers are heterogenous, they may have different preferences and judge others by different criteria. For example some peers may rate a movie provider as good because he provides high quality movies whereas others may consider him bad as his download speed is slow. Thus if A and B are similar in their evaluation criteria, A can trust B’s recommendations if he knows B is truthful. However if they have different evaluation criteria then A cannot trust B even if he is truthful.

Most of the current trust and reputation models in the literature follow these five general steps [25].

1. Forming opinions about a certain peer based on Direct interactions with that peer.

2. Collecting information about a certain peer in the community by asking other users their opinions or recommendations about that peer.

3. Aggregating all the received information properly and computing a score for every peer in the network.
4. Selecting the most trustworthy or reputable entity in the community providing a certain service and effectively having an interaction with it, assessing a posteriori the satisfaction of the user with the received service.

5. According to the satisfaction obtained, a last step of punishing or rewarding is carried out, adjusting consequently the global trust (or reputation) deposited in the selected service provider.

Traditional trust models only consider the reputation accumulated by peers’ long-term behaviors, and are not adaptable to dynamics of the behaviors.

7.3 Proposed Adaptive MultiDimensional Trust Model

We propose an adaptive layered trust model with a hierarchy of privileges which builds upon the Peertrust [8] model. In an incentive based reputation system different context categories can be incorporated into the basic trust metric with a weight function. For example for an e-commerce transaction, peers can submit feedback about each other with different parameters such as quality of product, delivery time, payment reliability, etc. The transaction cost could also be taken into account. The final trust would then be a weighted aggregation of feedback along these different parameters multiplied by the transaction cost for high-value transactions. Similarly for a software development community peers would submit feedback along dimensions such as quality of software, Lines of Code, completion time and reliability.

We propose a context based trust model that determines the weight function for such different parameters and integrates the different extensions into our trust metric.

7.3.1 Features of our adaptive trust model (ATM)

We build upon the peer trust model which is based on a weighted sum of five peer feedback factors, namely, peer records, scope, credibility, transaction context, and community context. However unlike Peertrust, we differentiate between peers who give poor service because they are incapable of giving better service and malicious peers who give varying levels of service in various transactions. This is done by measuring the variation of service among random windows in the history of a peer. High degree of variability is penalized and those peers are classified as malicious peers. Peers are classified as malicious either due to maliciousness in providing
service or in providing recommendations. The latter is achieved by the similarity algorithm for measuring credibility of a peer. A peer is classified as malicious if his credibility falls below a certain threshold. The effect of giving false ratings is decreased by reducing the credibilities of such peers.

### 7.3.2 Basic Trust Metrics

We identify different factors for evaluating trust of a peer in different application scenarios

1. Feedback in terms of reputation of a peer as computed by peers with whom it had direct interactions.

2. Reputation computed on the basis of indirect recommendations given by unknown peers.

3. Total number of transactions and cost of transaction in e-commerce scenario.

4. Credibility of peer giving the feedback.

5. Context specific attributes associated with the transactions and current trust value of peer u as calculated by peer v based on the attributes

6. Trust value of peer u based on past history

7. Weightage to be given to recent interactions.

We incorporate various attributes for feedback to add multiple dimensions for peer selection, into the basic trust metric with a weight function. We assign weights with different attributes depending on the type of application. We include a decay factor as trustworthiness is not always the same and may change with time. We incorporate different methods to calculate the credibility of the peer who is giving feedback, based on the situation.

We propose a dynamic trust policy which dynamically assigns weights to different trust attributes. Our trust model along with our dynamic trust policy helps us to slowly remove malicious peers whose behavior is dependent upon a specific pattern. We compare the results of our model with the P2P Trust model[6] which calculates trustworthiness of a peer using direct reputation, and indirect recommendation and show that our model handles malicious peers more efficiently.

The Basic trust metrics in our model are:
Reputation

This is a metric measuring overall quality of previous transactions between peer x and y. We represent \( f_E(x, y) \) as the feedback given by peer x for peer y for a transaction.

Let \( a_i = a_1, a_2, a_3, \ldots, a_n \) be the set of attributes used for computing the feedback. Then

\[
f_E(x, y) = \sum_{i=1}^{n} f_{a_i} * w_{a_i}
\]

where \( f_{a_i}(x, y) \epsilon [0, 1] \) is the feedback score given by peer x about peer y for attribute \( a_i \).

Relative importance assigned to each attribute can be modeled as weight \( w_{a_i} \) such that \( \sum w_{a_i} = 1 \).

For a P2P file sharing application the attributes could be parameters like

1. Quality of File QOF
2. Size of File
3. Ratio of Amount of File Received to amount of File Demanded
4. Download Rate

   For an ecommerce application the attributes are parameters like

1. Cost=Production cost + Transaction Cost
2. Ratio of Actual Delivery time to Expected Delivery Time
3. Incentives
4. Quality of Product

For a FOSS application the attributes would be

1. Lines of Code LOC
2. Cost
3. Number of bugs
4. Delivery Time
Assume that peer x stores up to n feedback ratings of previous transactions with peer y. Then reputation of peer y is calculated by peer x as

\[ R(x, y) = \frac{\sum_{i=1}^{n} \alpha_i \cdot f_{E_i}(x, y)}{\sum_{i=1}^{n} \alpha_i} \]  

(7.1)

where \( f_{E_i}(x, y) \) denotes the ith feedback given by peer x to y and \( \alpha \in [0, 1] \) is a decay factor which indicates how important the most recent interaction is to the user. Each user may set a different value of \( \alpha \) for itself.

**Direct Trust**

This is derived from a peer’s all Direct transaction experience, their credibility and the number of times the peer has been interacted with them.

\[ T_D(y) = \frac{\sum I(x, y) \cdot R(x, y) \cdot Cr(x)}{\sum I(x, y) \cdot Cr(x)} \]  

(7.2)

where \( R(x, y) \) is the reputation which can be calculated using equation 1, I(x,y) is the number of times peer x interacted with peer y, Cr(x) denotes the credibility of peer x.

**Indirect Trust**

If a peer does not have sufficient number of transactions with a resource it can ask the opinion of other nodes to gain an overall evaluation \( T_{ID} \). Assume that each user x receives job ratings for peer y from n references \( k = (1, 2, ... n) \) and all nodes use the same decay factor \( \alpha \). Then user x can compute the reputation \( f_{E_i}(k_i, y) \) of each indirect peer \( k_i \) to resource y.

\[ T_{ID}(y) = \frac{\sum_{k=1}^{n} I(x, y) \cdot R(k, y) \cdot Cr(k)}{\sum_{k=1}^{n} I(x, y) \cdot Cr(k)} \]  

(7.3)

**Motivation based Trust Factor**

To add incentives for rating we add the factor

\[ \frac{F_x}{I_{x,y}} \]

where \( F_x \) is the total number of feedback ratings given by peer x. It is a positive factor for correct ratings and a negative factor for incorrect ratings. Thus we increase the credibility factor of peers who give proper recommendations and decrease the credibility of peers giving incorrect recommendations.
**Transaction Factor**

We define a transaction factor to give weightage to the type of transaction. For an ecommerce application this could vary with the cost of the transaction and for a P2P file application this factor depends on the importance of the type of file being demanded. Files are categorized into different types according to their importance. The transaction factor is multiplied to the Direct Trust value.

**Credibility of evaluating peer**

The evaluator’s credibility is important for a peer to decide whether to accept the reputation value or not. We incorporate different methods to calculate the credibility of the recommending peer.

**Direct Credibility**

Here we assume that the current trust value of a peer is his credibility. If an evaluator has credibility beneath a certain threshold, his evaluation will be ignored. The group dynamically decides on the selection threshold for the credibility to protect against malicious peers. This method is selected when there are few peers in the group and only the Direct Trust component is used to calculate trust. This works well when the well-performing or trustworthy peers provide proper feedback. However it is possible that a peer maintains good reputation by giving good service, but sends malicious or improper feedback to its competitors. In such a case using trust to approximate the credibility of feedback gives errors a different method has to be used.

**Credibility using Feedback Similarity**

Here the credibility factor is calculated using the feedback similarity of the requesting peer itself i and the recommenders over the common set of peers $X_1, X_2, X_n$, denoted as $IR(X)$ who have interacted with both requesting peer i and the recommending peer R. $R_1, R_2, R_3$ are the recommenders of the host peer u, and $X_1, X_2, X_n$ are the peers who have transacted previously with both the requesting peer i and the recommenders.

Peer i now uses the feedback similarity on how the recommenders have evaluated the trust of commonly transacted peers $IR(X)$. The root mean square or standard deviation (dissimilarity)
Figure 7.1: Credibility using Feedback Similarity

of the two feedback vectors is calculated. Credibility using this method is calculated as,

\[ C_r = 1 - \sqrt{\frac{\sum (T_D(R, X_j) - T_D(i, X_j))^2}{IR(X)}} \]

where,

\( T_D(R, X) \) is the direct trust that recommender R have on the peer X and \( T_D(i, X) \) is the direct trust that peer i have on the peer X

The similarity method of evaluating the credibility of feedback is the personalized credibility measure and it gives more weight to the feedback from similar rater. Hence it also acts as an effective defense against colluding peers. Colluding peers are the peers that give bad ratings to the peers outside the group irrespective of the quality of service they receive. The feedback similarity between a peer R in the collusive group and a peer i outside the group will be low and hence very less weight will be given to its feedback.

**Credibility using Distance Attenuation**

In this method[102] the Credibility is measured based on Transitive Trust, which means trust is derived from an existing trust relationship between entities. The feedback aggregation algorithm should take into account the transitive attenuation of trust relationship. Thus if peer \( P_j \) collects feedback about peer \( P_j \) from \( W=W_1, W_2, \ldots, W_m \) other nodes then the feedbacks collected from
closer peers should carry more weightage as compared to feedback collected from distant peers. Then user $P_i$ can compute the reputation $f_E(k_i, y)$ of each indirect peer $k_i$ to resource $P_j$ as

$$T_{ID}(P_i, P_j) = \begin{cases} \frac{\sum_{k=1}^{m}(\omega(W_k) \cdot (TD(W_k, P_j)))}{\sum_{k=1}^{m} \omega W_k} & M \neq 0 \\ \frac{1}{M = 0} & \end{cases}$$

(7.4)

where $TD(W_k, P_j)$ is the direct trust of $P_j$ as calculated by $W_k$ and $\omega$ is the path function which is used to assign weight to the credibility of $W_k$.

Since $\omega W_k$ should reflect the attenuation and transitive trust factor it is defined as:

$$\omega W_k = \begin{cases} \prod_{m=1}^{level} T(P_m, P_{m+1}) & level > 0 \\ 1 & level = 0 \end{cases}$$

(7.5)

where $T(P_m, P_{m+1})$ is $P_m$‘s trustworthiness for the descendant node $P_{m+1}$ along the direction of the trusted path.

This method of calculating the credibility is selected when the percentage of malicious peers is high.

**Global Trust value**

Let $T_x$ be the global trust value of a peer $x$. Then $T_x$ is an aggregation of the direct experiences of every peer about peer $x$ as well as the recommendations received about peer $x$, and the context factor.

$$T = \alpha \cdot TD(x, y) + \beta \cdot T_{ID}(x, y) + \gamma \cdot Context\ factor$$

(7.6)

where $\alpha$ is the weight associated with direct experience, $\beta$ with indirect reputation and $\gamma$ is a fine tuning constant to control the amount of reputation gained by rating others.

The trust policy dynamically assigns the values of $\alpha$, $\beta$ and $\gamma$ depending on the current group composition and average group trust.

**7.3.3 Malicious Peer Behavior**

To model malicious peer behavior, each peer builds a knowledge base represented by three list structures viz: Doubted List, Black List and Trusted List. The doubted List contains unique
user ID (UUID) and public key of peers who have suddenly given deviated performance. If the difference in trust value of some peer X as calculated by peer Y in earlier transactions is greater than some threshold $\epsilon$, then a peer is included in the doubted list.

The Similarity method is used to calculate the credibility of feedback when the selected host peer is in the doubted list of the client peer i.e when there is a large variance observed in previous rating and current rating. If peers in the doubted list continue giving bad performance then they are moved to Black list and the trust policy does not permit transactions with peers in this list. The Black List is purged according to an ageing policy. This strategy helps us to slowly remove peers whose misbehavior is dependent on some pattern. A peer in the doubted list is given a chance to improve his performance by allowing transactions and is moved to Trusted list if the trust value crosses the desired threshold. The control peers have a view of these lists and can broadcast the information about malicious peers after consensus. When there are a large number of peers and the percentage of peers in the doubted list is very high, then we use the distance attenuation method to calculate the credibility.

### 7.3.4 Detecting and penalizing malicious behaviors

Peers with intention of cheating other peers by manipulating the trust model are penalized and finally flagged as malicious by the trust model. The various types of maliciousness considered are variability of service, giving better service in smaller transactions to drive the rating higher and then giving poor service on a higher value transaction, cartelization of users to promote the ratings of that group and giving false ratings to users etc. Malicious behavior is detected and penalized in the following ways:

1. The ratings of the transactions are weighed by the value of the transaction thus denying the strategy of cheating on a high value transaction and giving good service on lower value transactions.

2. The variance in the level of service is detected and penalized by the Variable window method which is explained in detail in section 7.3.4.

3. Cartelization of users is discouraged by the use of credibility in the computation of indirect trust obtained by the similarity method detailed in section 7.3.2.

4. The effect of giving false ratings to peers is also decreased in the course of computation
of indirect trust by reducing the credibilities of the mischief causing peers in the eyes of the honest peers.

5. Peers are flagged as malicious either due to maliciousness in providing services or in the process of giving feedback to other peer’s behavior. Here the distinction between peers with poor service and malicious service is also considered before flagging a peer malicious to avoid false-positives.

6. A peer is classified as malicious if his average credibility among his peers falls below a threshold. It means that the peer is either part of a collusive group or is giving false feedbacks about other peers.

To evaluate how effectively this model can combat the problem of collusion between peers, a simulation was conducted with 1000 peers and 600 feedback raters and a collusive group among the feedback raters of size 100. The simulation was run for 1000000 transactions and the average credibilities of collusive group among the other peers and the average credibilities of the other peers was plotted. The results of the simulation is shown in graph 7.2. As we can see in the graph 7.2, the average credibility of the other peers is significantly higher as compared to the average credibility of the collusive peer, thus the feedbacks of the collusive group get very little weightage in the computation of indirect trust and also the low credibility
among the collusive group enables us to classify them as malicious.

To evaluate the performance of our model we compare the average rating given by our system to the malicious peers and the average rating given by our system to the non malicious peers. This is simulated with 1000 peers of which 20 percent of the peers are malicious and the average trust of malicious and non malicious peers are plotted across number of iterations, where each iteration represents 1000 transactions. This is shown in graph 7.3.

**Variable window method**

The difference between the maximum and minimum rating acquired by a peer in return for service given(or task performed) is monitored within a window of length 1. Whenever a peers’ rating difference is greater than a certain threshold, a violation is added to the peers’ record of violations and his direct trust value is penalized by applying a reduction factor. We measure the maximum variation of a peer in various random window sizes. Thus both oscillatory behavior and high variance in level of service is detected and penalized.
7.4 Modeling E-commerce Application

We performed some experiments to validate our model. The proposed model can be applied to any Peer to Peer framework. The roles of peers can be categorized as resource nodes, reference nodes and user nodes. Resource nodes provide services utilizing their resources such as shared files, memory, or processors to users, while the users submit their jobs, e.g., to download some files, to request for computation or usage of some types of resources. Physically, a particular node can be either a user or a resource, or even both at the same time, i.e., the node can provide its resource to others and simultaneously submit its own job to request the resources from others. A transaction between a user and a resource is referred as the processing of a job. After receiving the result of a job, the user calculates a job rating to evaluate the quality of this finished job.

7.4.1 Peer Behavior

We categorize peer behavior as honest and malicious. Honest nodes always provide good service and correct ratings. Malicious peers are of two kinds: those who provide good service but poor ratings, and those who provide bad service and bad ratings. Malicious peers having low trust values try to increase their trust values by providing good services for some period of time. However they always provide poor ratings to good peers and good ratings to other malicious peers.

7.4.2 Modeling Malicious Peers

Peers behave as either service providers or raters. Malicious peers are peers which either provide bad service or poor ratings or both and use unfair methods to lower the trust values of their neighbors. In the simulator the roles of peers as Feedback Raters can be either honest FR(HFR) or malicious FR(MFR). HFRs give correct ratings with a probability of 80 percent while a MFR always gives incorrect ratings with a high probability. Malicious feedback raters could also behave in a collusive manner and send out 1 for the peers in the collusive group, and 0 for the peers outside the group.

As service providers peers are of two types Good Service Providers(GSP) and Bad Service Providers(BSP). GSPs provide good service with a high probability while BSPs can dynamically change their identity between good service and bad service. Random malicious peers act as BSPs with some probability say 20%, while structured BSPs change their quality of service
in a specific interval.

### 7.5 Simulations

In the initial set of experiments we chose the value of decay factor $\alpha$ defined in equation 7.1 as 0.9. The weights assigned to the three components of global trust (equation 7.6) are varied dynamically depending on whether the resource peer is a totally unknown peer, or a highly trusted peer or a peer with low trust values. We started with 20 peers in the network initially and upto 100 peers were allowed to join. All nodes could act as user, 20 as resources and 80 as references. At every time slot a user generates a number of jobs. We assume that each resource can process 3 jobs simultaneously. The quality of each job processing is randomly determined depending on whether the resource is honest or malicious.

We performed a comparative analysis on a network of 100 nodes with 800 transactions in each simulation. The parameters used for comparison were

1. Average trust value of good peers
2. Average trust value of malicious peers
3. Transaction success ratio in presence of varying percentage of malicious peers
4. No of times the malicious peers are selected for transacting with increasing percentage of malicious peers.

We refer to our model as ATM i.e adaptive trust model and we compare it with the an average trust model TM which is explained below.

TM [6] is a trust model for P2P ecommerce based on Direct and Indirect trust. In the TM model attributes for peers are not considered, nor are peers segregated into a doubted list and black list. In this model the trustworthiness of a peer is calculated by three steps: calculate direct reputation, assess recommendation reputation and determine global reputation. The steps are briefed below.

1. Calculates Direct Reputation
   The direction reputation is derived from a peer’s all direct transaction experience. Decay quotient (for example $\alpha = 0.95$) is used which reflects the older transaction will gain the smaller weights of ratings.
2. Assesses recommendation reputation

Let $T_r$ denote the recommendation reputation of peer $i$, $S$ is the set of peers with whom peer $i$ has conducted transactions, $t_{ji}$ is the local trust score of peer $i$ rated by peer $j$, and $W_j$ is the aggregation weight of $t_{ji}$. The TM model uses the formula below to assess recommendation reputation of peer $i$.

$$
T_r = \frac{\sum_{j \in S} W_j \cdot t_{ji}}{\sum_{j \in S} W_j}
$$

$W_j > W_{th}$ Where $W_{th}$ denotes the trustworthiness threshold

3. Determines global reputation

The final step is aggregating the global reputation. Let $T$ denote the global reputation of $u$ to $v$, $T_d$ is the direct reputation of $u$ to $v$, $T_r$ is the recommendation reputation of $u$ to $v$.

$$
T = \lambda \ast T_d + (1 - \lambda) \ast T_r
$$

Where, $\lambda$ reflects weights of direct reputation and recommendation reputation.

7.5.1 Results and Analysis

Initially we injected 10 malicious nodes in the network and did a comparison of the average trust value of good peers versus average trust value of malicious peers for both the models. We repeated the experiment for 20%, 30% and 40% malicious nodes.

Analysis 1

Graphs in figure 7.4, 7.5 and 7.6 show that in our model the average trust value of good peers has a gradual increase curve while in TM model there is a steady increase in trust value of good peers. Average Trust value of malicious peers on the other hand shows a gradual degrading curve in our model and with increase in % of malicious peers it degrades further, whereas in case of TM model the average trust value of malicious peers is more or less constant and does not change much with increase in % of malicious peers.

Graph in Fig. 7.7 shows the trust value of malicious peers decreases considerably with our dynamic task allocation policy whereas for the TM model[6] the malicious peers are able to maintain their trust value. Thus in presence of our dynamic trust policy, gradually with increase in interactions the group is able to isolate the malicious peers as their trust value falls beneath a certain threshold and they would not be allocated any further tasks.
Figure 7.4: Aggregate trust values of peers with 10 percent malicious peers

Figure 7.5: Aggregate trust values of peers with 20 percent malicious peers

Analysis 2

The graph in fig 7.8 shows the ratio of successful transaction in both the models with increase in percentage of malicious peers. Graph shows that in our model the successful transaction ratio does not degrade with increase in malicious peers while in the TM model it degrades considerably.
Analysis 3

We did a comparison of the service selected of good peers versus malicious peers for both the models in presence of increasing percentage of malicious peers.

Graphs in fig 7.9, 7.10 and 7.11 show that in our model as the percentage of malicious peers increase, the service selected of malicious peers decrease whereas in case of the TM model with increase in percent of malicious peers service selected of malicious peers also increases. With increase in the number of malicious peers in the network, the number of times malicious peers are selected for transaction increases while number of times good peers se-
Figure 7.8: Successful transaction ratio

![Successful transaction ratio graph](image)

Figure 7.9: Comparison of frequency of selection of good and malicious peers in presence of 10 percent malicious peers

![Comparison of selection graph](image)

lected for transaction decreases in the TM model. In our model the number of times malicious peers are selected for transactions is very low and number of times good peers are selected for transactions is very high.

**Analysis 4**

We plotted the transaction success ratio when service selected is only of malicious peers in order to see whether malicious peers are allowed to provide service.

The graph is fig 7.12 shows that in our adaptive model very few malicious peers are al-
Figure 7.10: Comparison of frequency of selection of good and malicious peers in presence of 30 percent malicious peers

Figure 7.11: Comparison of frequency of selection of good and malicious peers in presence of 50 percent malicious peers

allowed to provide service.

7.6 Evaluation of Trust Metrics

Trust evolves with time based on experiences. Positive experiences can lead to increase of trust while negative experiences can decrease trust. One of the main concerns of reputation systems is their effectiveness in coping with malicious peers who change their behavior with time. Claudiu Duma and Nahid Shahmehri [106] identify the main requirements of a dynamic
trust metric as follows:

- It should be sensitive to new experiences
- Sensitivity to new experiences should not depend on total number of experiences.
- Long term behavior must be taken into account.
- It should be able to penalize both sudden errant behavior and long term oscillatory behavior.

Keeping the above features in mind we do a qualitative comparision of the behavior of two trust models: viz an average trust model(TM) and and adaptive trust model (Peertrust) with our dynamic trust model which builds upon Peertrust model. In the average trust model sensitivity of the trust model decreases with increase in the number of experiences. It is less sensitive to changes in peer behavior.

The peertrust model includes a context factor to distinguish the trust given to a peer for different transactions. It is capable of detecting and penalizing sudden misbehaving peers.

However the way it measures credibility of a peer does not distinguish between confidence placed on a peer while giving a service or carrying out a task and when giving recommendation about other peers. Further the model is unable to detect and penalize long term oscillatory behavior.
<table>
<thead>
<tr>
<th>Metric Features</th>
<th>Average Trust Model</th>
<th>Peertrust Model</th>
<th>Our trust Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity to new experiences</td>
<td>Less sensitive</td>
<td>More sensitive</td>
<td>More sensitive</td>
</tr>
<tr>
<td>Sensitivity does not depend on total no</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>of experiences</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incentives provided for giving</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>accurate ratings</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Long term behavior taken into</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>account</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detects and penalizes sudden errant</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>behavior</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detects and penalizes long term</td>
<td>No-unable to detect</td>
<td>No-unable to</td>
<td>Yes-partially</td>
</tr>
<tr>
<td>oscillatory behavior</td>
<td></td>
<td>detect</td>
<td></td>
</tr>
<tr>
<td>Credibility measure distinguishes</td>
<td>No</td>
<td>No, only false</td>
<td>Yes, false</td>
</tr>
<tr>
<td>between confidence placed for services</td>
<td></td>
<td>ratings are</td>
<td>ratings detected</td>
</tr>
<tr>
<td>and recommendations</td>
<td></td>
<td>detected by</td>
<td>by similarity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>similarity</td>
<td>algorithm and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>algorithm</td>
<td>variance in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>service detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>by random window</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>method.</td>
</tr>
</tbody>
</table>

Table 7.1: Comparison of Trust Metrics

The table in figure 7.1 gives a qualitative comparative evaluation of the average trust model(TM), adaptive trust model (peertrust) and our adaptive trust model(ATM) based on a few criteria.
7.7 Summary

Our experiments prove that our trust model performs well in the presence of malicious peers and we are able to gradually eject the malicious peers from the group. In case of colluding malicious peers our credibility method based on similarity helps to detect large deviations from behavior and thus mitigates the effect of such peers. The transaction success ratio decreases considerably in our approach with increase in percentage of malicious peers whereas in case of the P2P model [6] there is no change with increasing percentage of malicious peers. Further in our model, the frequency of selection of malicious peers decreases gradually with increase in percentage of malicious peers. Thus we have an adaptive trust model, which is integrated with our dynamic policy based model.
Chapter 8

Conclusions and Future Work

We conclude by providing a summary of our contributions and a discussion on directions for future work.

8.1 Summary

Web based collaborative groups face many challenges due to the dynamic and unreliable nature of the peers. In the interests of the group and for its survival there is a need for peers to collaboratively modify group policies dynamically and the system should be able to evolve and adapt by switching between different policies and varying system models. As groups evolve, there is a need for policies also to evolve and the group must adapt to these evolving policies for optimum performance.

Our experiments prove that we have a flexible framework for modeling dynamic collaborative groups. The framework permits us to evaluate and measure different group metrics like job success, average trust value of peers, effect of varying join policies, varying job-allocation policies, effect of varying percentage of malicious peers and so on. Different applications can be modeled and given as inputs to our simulator. We are able to express dynamic policies and varying system models in our framework. It supports dynamic adaptation of policies and dynamic re-organization of group structure by switching between system models.

Experiments show that this dynamic adaptation of system models and policies help to optimize group performance. It was observed that even if the group begins with a system model and a set of policies that does not give optimum performance, it ultimately adapts to the system model and policy set achieving optimum performance. Thus users are able to evaluate the effect
of different dynamic policies on groups and adapt to the right set of policies for evolution of the group. Use of a declarative language like prolog to model the policies permits dynamic switching between policies which results in optimum group performance in most cases. Admission control is achieved through prolog policies as the group can prioritize join requests and postpone them if required.

We also give an adaptive multi-dimensional trust model for P2P applications. Our adaptive trust model integrated with our dynamic policy model helps us to mitigate the effect of malicious peers and they are gradually ejected from the group. Our trust policy is able to adaptively select different trust models depending on group health.

Collaborative applications like F/OSS, multi-player online gaming and others can use this framework to test the evolution of peer groups and decide which policies to use to optimise group performance.

8.2 Scope for future work

Though we have a tool where peers can collaboratively switch between different policies and different process workflows, we feel that the use of collaborative reinforcement learning (CRL) would further help to optimise group behavior. CRL is a technique that enables groups of reinforcement learning agents to solve system optimization problems in dynamic, decentralized networks. Another scope for enhancing the work is that the system should be able to suggest new policies for optimizing behavior. The Trust model can also be made more adaptive by dynamically assigning weights to each trust parameter. The system should be able to dynamically adapt to the most optimum trust model. Currently we are able to mitigate the effect of malicious peers to some extent, however in case of largely oscillating behavior of malicious peers an improved malicious peer model would be desirable.
Appendix A

Peersim Platform

We give here the details of the simulator we have used to build our tool.

A.1 Simulation Platform

We developed our code in PeerSim [59] which is a Peer-to-Peer simulator. It has been designed to be both dynamic and scalable. The engines consist of components which may be ‘plugged in’ and use a simple ASCII file based configuration, mechanism which helps reduce the overhead. PeerSim is a library written in the Java language which consists of different components or classes which help in constructing and initializing the underlying network, which can handle different protocols, can control and modify the network. Every component can be dynamically loaded through the configuration file and almost every component can be replaced with alternative implementations.

It works in two different modes cycle-based and event driven mode. In cycle-based mode, nodes communicate with each other directly, and the nodes are given the control periodically in some sequential order, when they can perform arbitrary actions such as call methods of other objects and perform some computations. While in event-driven mode, controls have to be scheduled explicitly since there are no cycles. In this mode we can send events (messages) to protocols. Event driven mode is more realistic because generally the system will be driven completely or partially by external events such as queries by users. We have used the event-driven mode for simulation.
A.1.1 PeerSim Architecture

PeerSim Architecture mainly consists of a Network and a Simulation Engine. The network of the PeerSim Simulator can be divided in 2 parts:

- **Protocols**: They are used to define the behavior of the different peers, for example handling and simulating the overlay network, or implementing an algorithm.

- **Nodes**: They represent the peer themselves in the P2P network. Every node has a stack of protocols which will define their behavior and network can be seen as the stack of nodes.

The Simulation Engine of a PeerSim Simulator consists of 2 components:

- **Initializers**: It initializes all the nodes at the start of the simulation.

- **Controls**: Controls can control the simulation, either at regular intervals or during the initialization of the simulation.
  
  - They can be simple *observers* which can gather statistics and print them.
  
  - They can also be *dynamics* which can modify the simulation itself to change its behavior.

The simulation engine of PeerSim is based on 2 modes:
1. **Cycle-based:** As the name implies, the cycle-based engine is based on cycles. At each cycle, the simulator goes through each node of the network and executes every protocol associated to this node. Controls are also executed periodically to control the simulation. It is based on the class `CDSimulator` from the `peersim.cdsim` package.

2. **Event-based:** The event based engine has a different way of scheduling events. Instead of scheduling the execution of the different protocols with cycles, they are scheduled through events. Events (or messages) are sent to the different protocols (for example by the control components, or by the protocols themselves), and the protocols can handle these messages and respond to them accordingly. It is based on the class EDSimulator from the `peersim.edsim` package. Due to the fact that it relies on messages, the event-driven simulator can emulate a transport layer, thus adding more realism to the simulation.

**Config File**

For enabling the simulation engine to understand our protocols and controls we define a configuration file which is a simple ASCII file which defines the components that we want to use in the simulation and how they should interact with each other. Every comment is prefixed with a `#` at the beginning of the line. There is no order in the configuration file for the instructions.

```
#random.seed 1234567890
network.size 40
simulation.experiments 1
simulation.endtime 10000
simulation.logtime 1
################### protocols ===========================
protocol.link IdleProtocol
protocol.p JoinProtocol
protocol.p.transport tr
protocol.p.link link
```
The following lines give a brief explanation of the sample config file.

- The lines from 1 to 5 represent the global peersim configuration, such as the seed for the random number generator, default network size, the number of simulation cycles to perform, the finishing time of the simulation and the log time.

- The protocol defined is the join protocol (lines 9, 10, 11)

- Lines 13-19 represents in-built protocols needed for passing message from one peer to the other in event driven mode

- The first initializer (lines from 22 to 25) runs at the starting of simulation and we can initialize all the peers using this. *graphs and SWILocation* are the parameters used in the initializer

- Lines 27-29 represents WireInetTopology which is one more control type initializer which runs at the starting of simulation
• Finally, a TrafficGenerator is defined from line 32 to 35. This is used to generate events from outside in the event driven mode. Step is the in-built parameter used to indicate the number of steps in which an event should be generated.

A.1.2 PeerSim Simulation Life Cycle

PeerSim was designed to encourage modular programming based on objects (building blocks). Every block is easily replaceable by another component implementing the same interface (i.e., the same functionality). The general idea of the simulation model is:

• Choose a network size (number of nodes)
• Choose one or more protocols to experiment with and initialize them
• Choose one or more Control objects to monitor the properties you are interested in and to modify some parameters during the simulation (e.g., the size of the network, the internal state of the protocols, etc)
• Run your simulation invoking the Simulator class with a configuration file that contains the above information

The objects created during the simulation are instances of classes that implement one or more interfaces. The life-cycle of an event-based simulation is as follows:

• The first step is to read the configuration file, given as a command-line argument. The configuration contains all the simulation parameters concerning all the objects involved in the experiment.
• Then the simulator sets up the network initializing the nodes in the network, and the protocols as stated in configuration file. Each node has the same kinds of protocols so basically initializations set up the initial states of each protocol. In the configuration file, the initialization components are stated by prefix init.
• Initialization also includes the wiring of the nodes at the start thereby specifying the connectivity of the network.
• After initialization, there should be some traffic or events generated so that event driven engine can call those events from the event queue and perform them. This can be done...
through control objects. The simulation ends when there are no events left in the queue. Messages or events can also be passed to each other by nodes while performing the simulation. The control components can be stated by prefix control in configuration file.

A.1.3 Protocols

The main protocols that have been used are:

1. **IdleProtocol** This is the predefined class of PeerSim library which is used to link all the nodes initially.

2. **JoinProtocol** This is the main class which implements the EDProtocol used for event driven simulation. This class contains all the functionalities of a peer used for joining, updation, leaving etc. This class extends a class named Peer which has all the variables required for a peer and some basic functions like encryption, decryption, verification etc. JoinProtocol contains a predefined method known as `processEvent` which is used to process incoming messages to a peer.

3. **FossProtocol** To model the application Foss we designed the FossProtocol. This class implements EDProtocol used for event driven simulation. It includes functionalities which permit a developer to send a request to the reviewer to review a module, or to a moderator to accept the module and so on. Every incoming message will be processed by `processEvent`.

A.1.4 Initializers

Initializers are classes which are called initially at the start of the simulation to initialize all the nodes or peers. We define two Initializers as follows.

1. **Initializer**: This class is used to initialize some nodes or peers, groups and group policies by loading a prolog engine in the network. Before starting the simulation we need some predefined peers and groups already in the network and later on new peers can join using protocols that been made.

2. **WireInetTopology**: This class is used to wire or link the peers in the network. All the peers in a group should be linked to every other peer in that group. Class `WireInetTopology`
ogy extends the predefined class WireGraph of peerSim library and make links between peers of groups.

A.1.5 Controls

Controls are used to control the simulation in between. We define two controls, viz. TrafficGenerator and FossControl. TrafficGenerator is a control which generates the traffic or messages of requests of join, update or leave and adds these requests periodically to the EventQueue of EDSimulator which is taken by Simulator one by one in the order of the time at which they have been added.

The structure of the message has been defined as a class MessageEvent. This class consists of mainly String requestType, senderNode (who is requesting) and a vector of messages to send with one message of Request. Request consists of three things viz type, role and level. These are all stored in a class named Request. This message is added to the EventQueue of the EDSimulator in the form of "requestedNode, MessageEvent". Simulator sends the MessageEvent to the requestedNode by extracting the event from the EventQueue. The different events or requests generated by the TrafficGenerator are as follows:

- **LeaveRequest**: This request is made by a peer who wants to leave the group and sends it to any one peer of the group. We maintain a vector (leavingNodes) which contains nodes that have left till now in the TrafficGenerator only. When a node leaves it is added in that vector so that they can join later in a new session.

- **JoinRequest**: This request can be made by a fresh peer or a peer who left the group previously. For joining as a fresh peer we make a new node and then add it to the network. A peer who was previously a member of the group and wishes to join in a new session can randomly be a peer from the vector 'leavingNodes' and can make a join request.

- **Feedback**: Feedback is given randomly by a peer to another random peer in his group. It sends a Rating Certificate which contains a rating signed by him with his trust value.

- **UpdateRequest**: Randomly a peer from the network sends an UpdateRequest containing the requestType and his certificate which contains all the rating certificates given to him by other peers as a feedback.
FossControl helps in creating events like sending finished modules to peers for reviewing. It is also responsible for assigning modules to idle peers if available and checking for the project completion.

**How to Run**

The command to run the simulation is:

```
java peersim.Simulator config-file.txt
```

Simulator class is the predefined class of the peerSim library which reads all inputs like protocols, initializers and controls from the config file and loads them at the start of simulation. The simulator used is the EDSimulator from the protocol *JoinProtocol*. It schedules all the events accordingly as follows:

```java
for i := 1 to simulation.experiments do

    initialize EventQueue events
    create Network

    create prototype Node:
        for i := 1 to #protocols do
            create protocol instance
        for j := 1 to network.size do
            clone prototype Node into Network
        create (controls, initializers)

    execute initializers

    time = 0
    while (time < simulation.endcycle) do
        (node, e) = EventQueue.getMin();
        node.getProtocol().processEvent(node, event)
        if (event is a control that returned true) then
            break
```
Appendix B

Prolog Policies

B.1 Static Policy

A sample Static policy is given below:

```
join(join).
update(update).
leave(leave).

member(member).
admission(admission_peer).
maximal(maximal_peer).

verify(Request,Rl,Level,Rate,SPR,Expert,Tot) :- join(Request), member(Rl),
          Level =:= 1, SPR > 0, Expert > 28, Tot < 200.

verify(Request,Rl,Level,Rate,SPR,Expert,Tot) :- join(Request), admission(Rl),
          Level =:= 1, SPR > 40, Expert > 28, Tot < 200.

verify(Request,Rl,Level,Rate,SPR,Expert,Tot) :- join(Request), maximal(Rl),
          Level =:= 1, SPR > 70, Expert < 32, Tot < 200.

verify(Request,Rl,Level,Rate,SPR,Expert,Tot) :- leave(Request).

verify(Request,Rl,Level,Rate,SPR,Expert,Tot) :- update(Request), member(Rl), Level =:= 2, Rate >= 4, Expert > 18, Tot < 200.

verify(Request,Rl,Level,Rate,SPR,Expert,Tot) :- update(Request), admission(Rl), Level =:= 2, Rate >= 7, Expert > 18.
```
verify(Request, Rl, Level, Rate, SPR, Expert, Tot) :- update(Request), maximal(Rl), Level =:= 2, Rate >= 8, Expert < 22.

verify(Request, Rl, Level, Rate, SPR, Expert, Tot) :- update(Request), member(Rl), Level =:= 3, Rate >= 7, Expert > 18.

verify(Request, Rl, Level, Rate, SPR, Expert, Tot) :- update(Request), admission(Rl), Level =:= 3, Rate >= 8, Expert > 18.

verify(Request, Rl, Level, Rate, SPR, Expert, Tot) :- update(Request), maximal(Rl), Level =:= 3, Rate >= 9, Expert < 22.

update_engine(Levels, MPs, APs, CPs, Tot) :- MPs < 0.

B.2 Sample Dynamic Policy for Join

A sample dynamic policy is given below:

\[
\begin{align*}
&\text{join(join).} \\
&\text{update(update).} \\
&\text{leave(leave).} \\
&\text{member(member).} \\
&\text{admission(admission_peer).} \\
&\text{maximal(maximal_peer).} \\
&\text{verify(Request, Rl, Level, Rate, SPR, Expert, Tot) :- join(Request), member(Rl), Level =:= 1, SPR > 0, Expert > 18.} \\
&\text{verify(Request, Rl, Level, Rate, SPR, Expert, Tot) :- join(Request), admission(Rl), Level =:= 1, SPR > 40, Expert > 18.} \\
&\text{verify(Request, Rl, Level, Rate, SPR, Expert, Tot) :- join(Request), maximal(Rl), Level =:= 1, SPR > 70, Expert < 22.} \\
&\text{verify(Request, Rl, Level, Rate, SPR, Expert, Tot) :- join(Request), member(Rl), Level =:= 2, Rate >= 4.} \\
&\text{verify(Request, Rl, Level, Rate, SPR, Expert, Tot) :- leave(Request).} \\
&\text{verify(Request, Rl, Level, Rate, SPR, Expert, Tot) :- update(Request), member(Rl), Level =:= 2, Rate >= 5, Expert > 18.} \\
&\text{verify(Request, Rl, Level, Rate, SPR, Expert, Tot) :- update(Request), admission(Rl), Level =:= 2, Rate >= 6,} \\
\end{align*}
\]
Expert > 18.

verify(Request,RL,Level,Rate,SPR,Expert,Tot) :- update(Request), maximal(RL), Level =:= 2, Rate >= 7, Expert < 22.

verify(Request,RL,Level,Rate,SPR,Expert,Tot) :- update(Request), member(RL), Level =:= 3, Rate >= 6, Expert > 18.

verify(Request,RL,Level,Rate,SPR,Expert,Tot) :- update(Request), admission(RL), Level =:= 3, Rate >= 7, Expert > 18.

verify(Request,RL,Level,Rate,SPR,Expert,Tot) :- update(Request), maximal(RL), Level =:= 3, Rate >= 8, Expert < 22.

update_engine(Levels, MPs, APs, CPs, Total) :- Total > 100, Total < 200,
   retract((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), member(RL), Level =:= 1, SPR > 0, Expert > 18)),
   retract((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), admission(RL), Level =:= 1, SPR > 40, Expert > 18)),
   retract((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), maximal(RL), Level =:= 1, SPR > 70, Expert < 22)),
   assert((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), member(RL), Level =:= 1, SPR > 0, Expert > 22)),
   assert((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), admission(RL), Level =:= 1, SPR > 40, Expert > 22)),
   assert((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), maximal(RL), Level =:= 1, SPR > 70, Expert < 26)).

update_engine(Levels, MPs, APs, CPs, Total) :- Total > 200,
   retract((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), member(RL), Level =:= 1, SPR > 0, Expert > 22)),
   retract((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), admission(RL), Level =:= 1, SPR > 40, Expert > 22)),
   retract((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), maximal(RL), Level =:= 1, SPR > 70, Expert < 26)),
   assert((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), member(RL), Level =:= 1, SPR > 0, Expert > 28, Tot < 300)),
   assert((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), admission(RL), Level =:= 1, SPR > 40, Expert > 28, Tot < 300)),
   assert((verify(Request,RL,Level,Rate,SPR,Expert,Tot)
   :- join(Request), maximal(RL), Level =:= 1, SPR > 70, Expert < 32, Tot < 300)).

B.3 Sample Dynamic Policy for Level and Role Update

A sample policy for update, join and addition of new levels:
:-dynamic verify/5, update_level/5, update_role/5, update_engine/4, introduce_level/4.

req(join).
req(update).
req(leave).

grouprole(member).
grouprole(admission).
grouprole(control).

level1(developer).
level2(reviewer).
level3(moderator).

verify(UUID,Request,GR,Level,SPR) :-req(Request), Request=join, grouprole(GR),
    level3(Level), SPR >80, assert(assignRole(UUID,GR)),
    assert(assignLevel(UUID,Level)).

verify(UUID,Request,GR,Level,SPR) :-req(Request), Request=join,
    grouprole(GR), GR=admission, level1(Level),
    SPR >40, SPR < 60, assert(assignRole(UUID,GR)),
    assert(assignLevel(UUID,Level)).

verify(UUID,Request,GR,Level,SPR) :-req(Request), Request=join,
    grouprole(GR), GR=control, level1(Level),
    SPR >60, SPR<80, assert(assignRole(UUID,GR)),
    assert(assignLevel(UUID,Level)).

verify(UUID,Request,GR,Level,SPR) :-req(Request), Request=join,
    grouprole(GR), GR=member, level1(Level),
    SPR >20, SPR <40, assert(assignRole(UUID,GR)),
    assert(assignLevel(UUID,Level)).

update_role(UUID,Request,GR,Level,Rating):-assignRole(UUID,member),
    req(Request), Request=update, grouprole(GR),
    GR=admission, level1(Level), Rating >5,
    assert(assignRole(UUID,GR)), retract(assignRole(UUID,member)).

update_role(UUID,Request,GR,Level,Rating):-assignRole(UUID,admission),
    req(Request), Request=update, grouprole2(GR),
    GR=control, level1(Level),Rating >7,
    assert(assignRole(UUID,GR)), retract(assignRole(UUID,admission)).

update_level(UUID,Request,CodesSubmitted,PercentAccept,Rating)
    :- assignLevel(UUID,developer), update_req(update),


148
CodesSubmitted > 20, PercentAccept > 50, Rating >6,
retract(assignLevel(UUID,developer)),
assert(assignLevel(UUID,reviewer)).

update_engine(Tot,MPs,APS,CPs):- MPs > 50, Tot > 200,
retract((verify(UUID,Request,GR,Level,SPR)
:-req(Request), Request = join, grouprole(GR),
GR=member, level1(Level), SPR >20, SPR <40,
assert(assignRole(UUID,GR)),assert(assignLevel(UUID,Level)))),
avert((verify(UUID,Request,GR,Level,SPR)
:-req(Request), Request = join, grouprole(GR), GR=member,
level1(Level), SPR >40, SPR < 50, assert(assignRole(UUID,GR)),
assert(assignLevel(UUID,Level)))).

introduce_level(Tot,CPs,CP_vote,RateAccept):- RateAccept < 30, Tot > 400,
CPs > 30, CP_vote > 90, retract(level2(reviewer)),
retract(level3(moderator)), assert(level2(reviewer1)),
assert(level2(reviewer2)), assert(level4(moderator)).

B.4 Sample Access Control Rules

Following are some rules which permit a developer to submit code to a reviewer, and permit a
reviewer to review code submitted by developer.

:-dynamic allow/7, not_equal/2.

req(submit_code).
req(review_code).
req(moderate_code).

grouprole(member).
grouprole(admission).
grouprole(control).

level1(developer).
level2(reviewer).
level3(moderator).

allow(PeerA,GRA,DRA,Request,PeerB,GRB,DRB): req(Request), Request = submit_code,
grouprole(GRA),level1(DRA),grouprole(GRB),level2(DRB),
not_equal(PeerA,PeerB),entity(PeerA),entity(PeerB).
B.5 Sample Java Code to Initialize Prolog Engine

Object[] var = new Object[]{req.type, req.role, req.level, rating1, spr, expert, tot};
JoinProtocol jp = (JoinProtocol) node.getProtocol(pid);
ret = engine.deterministicGoal("ipObjectSpec('java.lang.Integer',Obj2,[Level],_)," +
"ipObjectSpec('java.lang.Integer',Obj3,[Rate],_)," +
"ipObjectSpec('java.lang.Integer',Obj4,[Expert],_)," +
"ipObjectSpec('java.lang.Integer',Obj5,[Tot],_)," +
"ipObjectSpec('java.lang.Integer',Obj6,[SPR],_)," +
"verify(Request,Rl,Level,Rate,SPR,Expert,Tot)",
"[string(Request),string(Rl),Obj2,Obj3,Obj6,Obj4,Obj5]", var);

messages++;
Bibliography


[8] Li Xiong and Ling Liu, PeerTrust: supporting reputation-based trust for Peer-to-Peer eCommerce Communities, IEEE Transactions on Knowledge and Data Engineering, (IEEE TKDE), 16(7), pp 843-857, 2004.


[59] PeerSim Simulator Documentation From http://peersim.sourceforge.net/


PUBLICATIONS


• Madhumita Chatterjee, G. Sivakumar, and B. Menezes, *Dynamic policy based model for trust based access control in P2P applications* 44th IEEE International Conference on Communications(ICC ’09), Dresden, Germany, 14-18 June 2009, pages 1-5.

Acknowledgments

No work is complete without acknowledging the efforts of all who made it happen. I shall remain indebted to my advisors Professor Dr. Sivakumar and Professor Dr. Bernard Menezes for their relentless support and guidance throughout. Prof. Bernard gave me intellectual freedom to follow my interests while at the same time pushed me to aspire higher and improve the quality of my work. Professor Sivakumar was more than a mentor. His committed guidance, patience, motivation, understanding and encouragement have made a lasting impression on me and helped me to survive the ups and downs of this long journey. He inspired me with new ideas, that helped improve my research skills and prepared me for future challenges. My heartfelt thanks to him.

I am grateful to my committee members Professor Om Damani and Professor Anirudh Sahoo for their insightful comments and constructive criticism of the work. Their suggestions and valuable feedback after every progress seminar helped me to improve the quality of my work.

I thank my family for their love and belief in my abilities. I thank my daughter Ipsita for displaying maturity beyond her age and for the constant encouragement she gave me to pursue my dreams. I thank my son Arko for bearing with me and understanding that I need to spend hours together without being disturbed. I owe a huge thanks to my parents without whose support and inspiration I probably would not have ventured to pursue research. It was mainly my father’s belief in my abilities that drove me forward. I thank my in-laws for their constant support, love and encouragement. I thank my extended family for their support.

I thank the head of CSE department, Prof Amitabh Sanyal for his encouragement and support during this tenure. I thank all the staff in CSE office, especially Vijay Ambre, Homcy and Madam Athvankar for their support. I thank all my friends and fellow research scholars for providing the moral support that I needed. I am grateful to the Principal of Pillai’s Institute of Information Technology, Engineering, Media Studies and Research, Dr. R.I.K Moorthy, and the
Director Dr. K.M Vasudevan Pillai, for their support and for giving me the flexibility to pursue my work in Indian Institute of Technology. I would also like to acknowledge the support given to me by the Management of Dr. D.Y Patil Group when I was in their services.

There are many who have helped me in different ways during this tenure. I thank Ashish Arya for his help with the initial experiments. Since it is difficult to name everyone, I thank all those who have directly or indirectly contributed in helping me to achieve this goal.

Date: _______________ Madhumita Chatterjee