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

Fourth Annual Progress Seminar Report

Submitted by

Madhumita Chatterjee
Roll No: 04429802

under the supervision of

Prof. G. Siva Kumar and Prof. Bernard Menezes

Kanwal Rekhi Institute of Information Technology
Indian Institute of Technology

January 27, 2009
Acknowledgment

I take this opportunity to thank Professor G. Siva Kumar and Professor Bernard Menezes for their invaluable guidance, encouragement and support. Acknowledgements are also due to my RPC committee members, Professor Anirudh Sahoo and Professor Om Damani for their valuable suggestions and comments during my earlier presentations.

Madhumita Chatterjee
Abstract

Self organizing and self adaptive groups evolve and adapt according to interactions within the group. Such groups thus have special security requirements not addressed by typical access control mechanisms. An example is the ability to collaboratively modify access control policies based on the evolution of the group and trust and behavior levels. We have proposed a secure integrated framework for self-organizing P2P groups with dynamic multi-level access control policies based on trust and reputation. The framework has interesting features wherein multiple policies can apply to a group simultaneously, or the system can switch between policies over time, influenced by the system’s state or environment. 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. We implemented the protocols of our integrated framework in simulator Peersim. We have implemented the dynamic policies using the declarative language Prolog. We interfaced Java and Prolog using InterProlog and have performed several experiments to validate our framework.
# Contents

1 Introduction ........................................... 1
  1.1 Motivation ........................................... 1
  1.2 Our work ........................................... 2

2 Generic Application Framework ..................... 3
  2.1 System Model ........................................ 3
    2.1.1 Basic Assumptions ................................ 4
    2.1.2 Peer Behavior and Roles ......................... 4
    2.1.3 Peer Quality .................................... 5
    2.1.4 Data Distribution ................................ 5
    2.1.5 Rating Dissemination Mode ....................... 5
    2.1.6 Task Allocation ................................ 6
    2.1.7 Layered Reputation scheme ....................... 6
  2.2 Context sensitive reputation model ............... 6
    2.2.1 Parameters for trust evaluation .................. 6
    2.2.2 Weightage for recent interactions ............... 8

3 Policy Based Access Control ....................... 9
  3.1 Hybrid Access Control Model ..................... 10
    3.1.1 Policy Driven Trust Based Access Control (PDTBAC) ... 10
    3.1.2 Modelling Policies using Prolog ................. 10
    3.1.3 Formal Model of Self Adaptation ............... 14
  3.2 Multi-policy Optimization using Collaborative Re-inforcement Learning ... 15
  3.3 Related Work ...................................... 16

4 Modelling F/OSS .................................... 17
  4.0.1 Task Allocation .................................. 17
  4.0.2 Peer Behavior ................................... 18
  4.1 Domain Specific Access Policy ................... 18
  4.2 Methodology ....................................... 19
    4.2.1 Rate calculation ................................ 19
    4.2.2 Project Quality ................................ 20
  4.3 Goals ............................................. 20

5 Trust and reputation management in Decentralized systems ................. 21
  5.0.1 Requirement for Trust Dynamics ............... 21
  5.0.2 Decentralized Trust Management ............... 22
  5.1 Threats for Peer to Peer Groups ............... 23
    5.1.1 Adversaries .................................... 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Assumptions</td>
<td>26</td>
</tr>
<tr>
<td>6.2 Integrating prolog at run time</td>
<td>27</td>
</tr>
<tr>
<td>6.2.1 Example Java Program</td>
<td>27</td>
</tr>
<tr>
<td>6.3 Implementation</td>
<td>28</td>
</tr>
<tr>
<td>6.3.1 Modelling rules for join</td>
<td>28</td>
</tr>
<tr>
<td>6.3.2 Modelling rules for level updatation</td>
<td>28</td>
</tr>
<tr>
<td>6.4 Results</td>
<td>28</td>
</tr>
<tr>
<td>7.1 Future Scope</td>
<td>31</td>
</tr>
</tbody>
</table>

7 Conclusion and Future Work                  | 31   |
7.1 Future Scope                              | 31   |
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>PDTBAC State Model</td>
<td>15</td>
</tr>
<tr>
<td>6.1</td>
<td>Overall Framework</td>
<td>25</td>
</tr>
<tr>
<td>6.2</td>
<td>Project Quality vs Time taken at code acceptance rate of 7</td>
<td>29</td>
</tr>
<tr>
<td>6.3</td>
<td>Load(no of messages passed) of admission peers vs Time taken</td>
<td>29</td>
</tr>
<tr>
<td>6.4</td>
<td>Load(no of messages passed) of control peers vs Time taken</td>
<td>30</td>
</tr>
<tr>
<td>6.5</td>
<td>Average trust value vs Time taken at code acceptance rate of 6</td>
<td>30</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Web based collaborative groups have gained enormous popularity as peers with common interests form a network among themselves and automatically tend to create interest groups among each other, called communities. The scope extends to applications like content distribution, publish/subscribe, semantic clustering, and live streaming, to name a few. The group is governed by a set of rules that describe the conditions required to be part of the group. A self-adaptive P2P group can be thought of as a system of processes that evolves and adapts according to interactions and relationships within the group. This group would work in a top-down fashion, evaluating its own global behavior and changing it when the evaluation indicates that the group is not achieving what it was intended to do or when better performance is possible. 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. The peers in such a group interact locally according to simple rules and the global behavior of the system emerges from these local interactions.

Examples of systems encompassing both self-adaptive and self-organizing aspects are found in socio-technical applications like ambient intelligence and ubiquitous computing systems, emergence response or e-health systems.

1.1 Motivation

In unpredictable environments such SASO groups must adapt to survive. Thus there is a need for self-organizing groups to change the access control policies in the interests of the group. These changes should be within the frame-work 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. Trust [5, 6, 7] is a means for understanding and adapting to the complexity of the environment. It can be a useful judgement in the experience of the behavior of others.

A motivating example is the self-organizing P2P group Free Open Source Software Development F/OSSD[34] which represents an approach for communities of like-minded participants to develop software systems and related artifacts 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 (programs, artifacts, execution scripts, code reviews, comments, and others) to Web sites within each community, and communicate information about their content updates via online discussion forums, threaded email messages, and newsgroup postings. F/OSS systems co-evolve with their development communities. This means the evolution of one depends on the evolution of the other. A F/OSS project with a small number of developers (most typically one) will not produce and sustain a viable system unless/until the team reaches a larger critical mass of 10-15 core developers.

In this type of a decentralized and dynamic peer group scenario it is necessary to have an adaptive access control mechanism where peers have the right to collaboratively make access control decisions, modify these decisions and frame rules based on attributes as well as trust evolved in the group.

1.2 Our work

We had earlier proposed an integrated framework [21] for collaborative P2P groups with protocols for secure join, leave and level update within the group. We had also proposed a hybrid access control framework which integrates policy based access control and trust based access control. We augment our earlier work with a dynamic policy based framework for group management. We are able to show that our policies are able to dynamically prioritize requests for join, or level update at any level of the group. Based on the policies and a layered context sensitive reputation model peers can be migrated between levels. Multiple policies can apply to a group simultaneously and the group can also switch between policies over time, influenced by the system’s state environment.

We have modelled our policies using Prolog which we integrated with our over simulator Peersim [22]. We have performed several experiments to validate our results.
Chapter 2

Generic Application Framework

In our generic framework, peers with a common interest join together to form group. We have an initial admission control algorithm which is policy based. Peers wishing to join the group declare an initial self-proclaimed rating 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. Once in the group peers are periodically rated by others based on some pre-decided rating algorithm which is also part of a dynamic policy. Each peer may store his updated ratings in his local rating history table. Expert peers would store the ratings of all peers in their level, and would also update the ratings of peers in their level. Rating certificates are signed by recommending peers and hence cannot be modified. A peer may have some maximum potential or skill set and could rise only up to that potential.

2.1 System Model

We consider a peer based model where every peer $P_i$ has a unique user identity $UUID_i$ associated with it. A group is a set of peers that are governed under a set of rules that describe minimal conditions to be part of the group and is formed based on a particular interest criterion. These initial or minimal set of rules or access policies form the group charter. For a group to survive it must adapt to the environment and to the behavior of the peers. Thus it is necessary for peers in a group to be able to dynamically modify the 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.

The group has an admission control policy [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 is enforced using voting. A group membership certificate is issued to a joining peer which contains tokens signed by the existing peers. 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 recommendations that he has received from other peers as well as the trust value of the recommending peer. 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.

2.1.1 Basic Assumptions

The following are the assumptions made in our model.

- Inputs to our system are good peers, average peers and O.K peers.
- No centralized control: Traditional access control models such as ACL or RBAC rely on central servers for authorization. In our model a peer has a high level of autonomy and can frame and manage his own policies.
- Peers in a group can belong to different levels within the group and can perform several roles. A role of a peer is independent of the level which the peer is in.
- 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.
- Peers are not malicious

Each group has a group policy which describes the roles permitted in the group and the rules associated with each role. A well formed group charter could have some initial rules as follows:

- Well defined rules for adding new peers and discarding malicious peers.
- Members should be able to share resources that exist among different peers based on the trust value associated with the peers.
- Framing group policies like minimum trust levels to be maintained.
- Rules for upgrading trust level of a peer or degrading trust level.
- Rules for creating and destroying a group.

2.1.2 Peer Behavior and Roles

A peer joins the group at a lower level with an intrinsic self-proclaimed rating. Once in the group he is periodically evaluated by other peers based on his performance and based on the reputation algorithm and update policy applicable he could be given a new reputation score and subsequent level updates in the group. A peer could have a maximum skill set which would be the deciding upper limit for his reputation or rating score.

A peer can have two roles in the group, 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 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 identify the following roles:
• **Member peer** 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 Peer** A peer with this role would participate in the voting algorithm, and would therefore store public keys of every other peer. However it need not store the updated ratings.

• **Control peer** A peer in this role would have all the functional components and would participate in updating levels of existing peers also.

After a peer joins the system it knows some peers and puts them in its neighbor list. Neighbors with highest trust values will be chosen when a peer needs help from neighbors. When neighbors are identified as bad peers with trust value below a certain threshold they will be purged from the neighbor list.

### 2.1.3 Peer Quality

As a rater, peer quality can be Good, Bad or Honest. A Good(Bad) peer would send absolutely correct(wrong) ratings each time based on truth. This is actually very difficult without a global view of the system, but we model it for studying extreme case of peer quality. Honest raters provide best-effort truth based on their own views.

As a service provider Peers can be Good, Average, O.K or Dynamic. We model good peers as expert peers who always provide excellent service. In the case of Average peers and OK peers the service quality gradually deteriorates. Dynamic peers are those who provide the services with one of the three qualities uniformly i.e 33% of its actual services are good.

### 2.1.4 Data Distribution

While addressing persistent data requirements of distributed systems many P2P systems assume that all peers have uniform capability and data distribution is uniform among all nodes. However use of peers with lower trust would degrade the performance of the entire network. To solve this problem many systems allow data to be stored only with reliable super peers. In a self-organizing system peers could dynamically elect super peers. In our model, roles are assigned to peers related to their capabilities and this could change dynamically based on performance and behavior in the group. Thus data storage at every level is done by Admission peers to some extent and Control peers to a larger extent.

### 2.1.5 Rating Dissemination Mode

We categorize the rating disseminations into Push Mode and Pull Mode. In the Push mode, once a neighbor’s trust value changes over a threshold, its update will be propagated to k other neighbors with highest trust values. In the pull mode a neighbor chooses k other neighbors who are control peers with high trust values to ask for ratings when it needs to calculate ratings of neighbor peer. Both the push pull method propagate the bad reputation of the neighbors in blacklist. In a bad community in which all peers are bad peers, the neighbor list will shrink to a very small size.
2.1.6 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.

2.1.7 Layered Reputation scheme

The peers in our groups can belong to n reputation layers. Higher the layer, better is the environment. A new peer joining our group would start at one of the lower layers since his behavior is not yet determined. He would have an intrinsic self-proclaimed rating and a maximum capacity based on his potentials. Moving up the layers requires improving reputation score, which is done mainly by providing content of good quality. However this would be application specific. A peer could be moved down the reputation layers, resulting from a decrease in his reputation score computed by the group. Thus there is a growth curve associated with every peer in the group which would be affected by the environment and would also depend on a peer’s maximum intrinsic capability. As a future aspect we would like to include learning from feedback, so that a peer can improve his performance to optimize group performance and not out of selfish interests.

2.2 Context sensitive reputation model

We propose a layered model with a hierarchy of privileges, so peers are encouraged to exhibit good behavior and reach the upper layers of the community, which provides them with a better environment. In this 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.

2.2.1 Parameters for trust evaluation

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.

**Peer’s reputation state**

A peer has a reputation grade that includes him in a corresponding layer. This inclusion is done by policies. The reputation of a peer is described as follows:

Let $I_u$ denote the total number of transactions performed by peer u and $P$ denote the set of peers.

$RI_u$ is the global rating of peer u.

$RI_v$ is the rating of peer v.

$T_{uv}$ is the trust value of u as computed by v.

**Weights and attributes** We assign weights with different attributes depending on the type of application. Let $x$ be the set consisting of the different attributes. $x = \{x_1, x_2, \ldots, x_n\}$

Relative importance assigned to each attribute can be modelled as weight $w_{xi}$ such that $\sum w_{xi} = 1$.

Peer v evaluates peer u against attributes $x_i$ and the rating thus computed is multiplied by $w_{xi}$ which is the weight associated with attribute $x_i$.

Thus $\sum T_{uv} = \sum R x_i * w_{xi}$

In the basic reputation metric we compute peer u’s reputation as the weighted average of the reputations stored in the history by every peer v with whom u has had direct interactions, multiplied by the credibility or reputation of the peer giving the rating.

$$RI_u = \frac{\sum_{v \in P, v \neq u} T_{uv} * RI_v}{I_u}$$

**For e-commerce domain**

In this domain we identify the attribute set $x = \{P, D, Q, S, R, T\}$

where

1. $P$ $\rightarrow$ price of product.
2. $D$ $\rightarrow$ Delivery time
3. $R$ $\rightarrow$ Reliability of payment
4. $S$ $\rightarrow$ prompt shipment
5. $Q$ $\rightarrow$ Quality of product
6. $T$ $\rightarrow$ quantum of trust in unknown

To account for high value transactions we add the cost of the transaction and fine tune its weightage with a constant.

$$RI_u = \frac{\sum_{v \in P, v \neq u} T_{uv} * RI_v * \alpha * C(u, i)}{I_u}$$
For F/oss domain

The attribute set for this domain would consist of parameters like...

1. \( P \rightarrow \) price of software.
2. \( D \rightarrow \) Delivery time
3. \( R \rightarrow \) Lines of Code
4. \( Q \rightarrow \) Quality

Adding incentives for rating

\[
RI_u = \sum_{v \in P, v \neq u} T_{uv} \cdot RI_v \cdot \alpha \cdot C(u, i) \cdot \frac{I_v}{I_u} + \gamma \cdot \frac{F_u}{I_u}
\]

where \( F_u \) is the total number of feedback ratings given by peer u. \( \alpha \) and \( \gamma \) are the fine tuning constants to control the amount of weightage to be given to transaction cost and amount of reputation gained by rating others respectively.

2.2.2 Weightage for recent interactions

We adopt the 3Win method to calculate the weightage to be given to recent interactions. A node’s history is divided into 3 performance windows named as Reputation Index Windows(RIWs) numbered 1 to 3. RIW1 contains the newest Feedback Indexes and RIW3 holds the oldest. FIs are pushed through the windows as new FIs arrive. An FI pushed out of RIW3 is discarded. \( RI = \alpha x RIW1 + \beta x RIW2 + \gamma x RIW3 \) where \( \alpha, \beta, \gamma \) could take on values 0.66, 0.22 and 0.11 respectively.
Chapter 3

Policy Based Access Control

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, authorization and management policies among participating parties etc.

Authorization in a distributed environment should be determined as a result of evaluating the request of an authenticated user against various policies like privacy policy, trust policy, authorization policy and so on. In the context of dynamic self-organizing collaborative groups we define policy as:

Policy(P):- Set of rules/requirements associated with a group/peer/service/resource or domain. Thus it can be represented as a set

\[ P = (A_uP, TP, PP, MP, AP, SP, OP) \]

where

- \( A_uP \) → authentication policy
- \( TP \) → Trust policy
- \( PP \) → Privacy policy
- \( AP \) → Admission policy
- \( MP \) → Management policy
- \( SP \) → Security policy
- \( OP \) → Other policy

Trust is not a fixed value and can change dynamically depending on the behavior of the peer and the context in the environment. We define a set of Global Meta Policies and a set of Context 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.
3.1 Hybrid Access Control Model

Our hybrid access control model is based on an integration of role based, policy based and trust based access control [3, 4]. The trust parameters in our model build upon the PeerTrust[14] 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. 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. 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.

3.1.1 Policy Driven Trust Based Access Control (PDTBAC)

Our model focuses on the genetic evolvement of groups based on group composition. In PDTBAC 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 simultaneously, or 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. To the best of our knowledge a group with such enhanced peer capabilities has not been discussed so far in the group communication systems.

3.1.2 Modelling Policies using Prolog

In 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 rules. A prolog rule is an expression of the form

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

where Ri are predicates and ui 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 modelling 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 modelled using the assert and retract clauses.

The Prolog inference engine provides a mechanism to derive consistent access control decisions at runtime. It may also be used to analyse the correctness and consistency of access control and other rules. 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 modelled 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 modelled in prolog as follows:

Allow(d, submit_code, c):-d(developer), c(code), phase(submission)
Allow(r, review_code, c):-r(reviewer), c(code), assigned(r, c)
Allow(r, read_scores, c):-r(reviewer), c(code), has_reviewed(r, c), phase(assessment)
Deny(d, read_scores, c):-d(developer), code(c).

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.

We define two levels of policies viz: Global Policies and Domain Specific Policies.

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 > some threshold x must collaborate. This ensures that malicious peers cannot frame new policies.

Modelling Rules for Join

An example of rules for peer with unique identity to join as a member peer in a group at level1 of a groups is

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

verify(Npeer, Request, Rl, Level, Rate, Vote) :- join(Request),
member(Rl), Level =:= 1, Vote >= 40, assert(belongs(Npeer, Level)).

verify(Npeer, Request, Rl, Level, Rate, Vote) :- join(Request),
member(Rl), Level =:= 2, Rate >= 3, assert(belongs(Npeer, Level)).

Modelling Rules for Level Update
When a member peer requests for level update, the policy verifies the current level of the
peer and checks his trust level.

verify(Npeer, Request, Rl, Level, Rate, Vote) :- belongs(Npeer, Level - 1),
update(Request), admission(Rl), Level =:= 2, Rate >= 5,
retract(belongs(Npeer, Level - 1)), assert(belongs(Npeer, Level)).

verify(Npeer, Request, Rl, Level, Rate, Vote) :- belongs(Npeer, Level - 1),
update(Request), maximal(Rl), Level =:= 3, Rate >= 8,
retract(belongs(Npeer, Level - 1)), assert(belongs(Npeer, Level)).

Domain specific policy
This set of policies inherits the properties of the global policy and contains application spe-
cific 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 optimise group performance. Another domain specific pol-
icy could be the framing of adaptive trust policies, i.e the group would work with trust
framework TF\_1 until it reaches a particular state and then adapt TF\_2 and so on.

Modelling Dynamic Policies
We are able to express dynamic rules at runtime, as well as change existing policies de-
pending 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.
One sample rule to prioritize join requests if a peer requests to join the group as a member
peer after the group has evolved for some time and the number of member peers in the
group is greater than 50% of current group size then the previous join rule is deleted from
the database and a new join rule is asserted, thereby postponing low-level requests for
join. Similarly other rules for checking no of control peers.

update_engine(Levels, MPs, APs, CPs) :- CPs > 20,
retract((verify(Npeer, Request, Rl, Level, Rate, Vote) :-
update(Request), maximal(Rl), Level =:= 3, Rate >= 7)),
assert((verify(Npeer, Request, Rl, Level, Rate, Vote) :- update(Request), maximal(Rl), Level =:= 3, Rate >= 9)).

update_engine(Levels, MPs, APs, CPs) :- CPs < 20,
retract((verify(Npeer, Request, Rl, Level, Rate, Vote) :- update(Request), maximal(Rl), Level =:= 3, Rate >= 9)),
assert((verify(Npeer, Request, Rl, Level, Rate, Vote) :- update(Request), maximal(Rl), Level =:= 3, Rate >= 7)).

Here update_engine function can be used to update the new rule by retracting the old rule and then assert a new rule. update_engine takes 4 inputs as Levels(no of levels), MPs(% of member peers), APs(% of admission peers), CPs(% of control peers).

3.1.3 Formal Model of Self Adaptation

We introduce a formal model of self organization in distributed groups as the adaptation of the group to improve desired system properties in the current environment.

System State

The set of initial states of the peers’ comprises the system state. At time t it is denoted as

\[ S_t = (S^i_t)_{i \in N_t} \in S \]

\( N_t \) is the set of peers at time t. \( S^i_t \) are the initial states of each peer. S is the set if all possible system states. If \( e_0, e_1, e_2 \ldots \) represent environment state then system behavior is modelled as a sequence of pairs of system state and total environment state. A system state transition is caused by transitions of individual states resulting in change of policy triggered by events like receiving feedback from peers, and the local environment.

\((s_1, e_1) \rightarrow (s_2, e_2) \rightarrow (s_3, e_3) \ldots \ldots \)

We introduce an abstract function \( behavior \) which measures the quality of some desired system property for a particular state of the system and the system’s environment.

\[ behav : SX \xi \mapsto R \]

The behav function measures how well the system state is matched to the environment. Self organizing systems should be engineered to adapt towards states that maximize system behavior. We define \( S^*(e) \) as the set of optimized states.

\[ S^*(e) = \{ s \in S : \text{behav}(s, e) = \max_{s'} \text{behav}(s', e) \} \]

So at time t, set of optimal system states is

\[ S^*_t = S^*_{(et)} \]

We apply this system model to our application of self-organizing groups F/OSS.

At a specific instant the system is in state \( S_i \) and environment \( e_i \) where policy \( P_i \) applies and group events like Join, Leave, Update, Review, Submit, Moderate and so on keep occurring. These group events have been labelled in the figure [3.1] as E1, E2, E3. The inputs to the system are good peers, average peers and malicious peers. Control Peers,
Figure 3.1: PDTBAC State Model

Admission Peers and Member peers have been labelled as CP, AP and MP respectively. Some control peers periodically monitor the Group Performance Index (GPI) which could be measured by factors like the current trust value of peers in the group, percentage of member peers and control peers in comparison with total group size and permissible joins as low-level peers within a stipulated time frame. IF GPI falls below a certain threshold, a control peer measuring group behavior could trigger an internal event like a request for policy change which would be done in consensus with other control peers at that level and group would now go to state $S_j$ with policy $P_j$. Group events $E_1,...,E_n$ would continue to occur in this new state with variations decided by the new policy. Thus state changes are triggered internally due to the effect of external events on the group.

3.2 Multi-policy Optimization using Collaborative Re-inforcement Learning

Self-organizing algorithms are suitable for engineering large scale decentralized systems as they enable the system to deal with the lack of global knowledge and central control. Such techniques have successfully been used to optimize software systems, such as optimization of route stability in ad hoc network routing and optimization of the use of storage space or processing power using load balancing. These algorithms could find a wider application in engineering decentralized and ubiquitous systems once they are also capable of dealing with optimizing the system’s performance towards multiple policies. All agents in a collaborative multiagent system can potentially influence each other. It is therefore important to ensure that the actions selected by the individual agents result in optimal
decisions for the group as a whole.

Collaborative Reinforcement Learning (CRL)\textsuperscript{[35]} is a multi-agent learning algorithm where agents perform actions that affect the environment and receive variable rewards for actions they perform. Agents learn the best actions to perform for given environmental conditions and entity characteristics, based on the rewards received. Changes in an agent’s environment provide feedback into the agent’s state transition model and connection cost model, while changes in an agent’s optimal policy provide collaborative feedback to the cached values of its neighboring agents using advertisement. As a result of the different feedback models in CRL, agents can utilize more information when learning an optimal policy in a distributed system. Collaborative feedback also enables agents to learn from their neighbors to solve collective system problems. CRL has been used to optimize routing and load balancing in peer to peer networks for single policy. We would like to use CRL techniques to optimize our group behavior, where multiple policies apply to the group simultaneously.

3.3 Related Work

Cassandra\textsuperscript{[26]} 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\textsuperscript{[27]} 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\textsuperscript{[28]} and Administrative RBAC (ARBAC)\textsuperscript{[29]}, where members of administrative roles can modify the role membership and privilege assignments. In \textsuperscript{[30]}, policies written in Datalog can refer to facts in the authorization state, as in our model. Hezberg et al. propose in \textsuperscript{[25]}, a prolog-based trust management language, but do not focus on dynamically changing policies with the state of the environment.
Chapter 4

Modelling F/OSS

To model our P2P collaborative group we chose the self organizing group F/OSS 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 newsgroup. 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 or 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.

4.0.1 Task Allocation

We assume a Poisson distribution of task arrival. Tasks are modelled here as software modules of different difficulty levels 1, 2, 3 etc. Peers are allocated tasks based on availability and capability of the peer as per his self proclaimed rating. Peers then get rated based on their contribution or according to the code that have submitted. 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.
4.0.2 Peer Behavior

For our simulations we are dividing peers into 3 categories as follows:

1. Expert Coders/Control peers: Expert coders are those who are more experienced ones and they have acquired very good reputation in the group and assume the role of ‘Control Peers’
   If they at developer level they rarely make mistakes and always submit their code before time and if they are at reviewer level they are very accurate in giving ratings.

2. Average Coders/Admission peers: Average coders are those who are little less experienced ones. They are also working as admission peers who can be trusted enough to give them control of handling admission of new peers. Average coders can be treated as the ones who if at developer level make no mistakes but take little extra time in finishing a code and if they are at reviewer level they allow developers to take little extra time and give them decent rating.

3. Ok/Bad Coders/Member peers: Ok Coders are considered as member peers only as they are not capable of managing the group behavior. If they are at developer level, they make some mistakes but finish their work on time and if they are at reviewer level because of less experience they may give good ratings to the bad codes also.

4.1 Domain Specific Access Policy

To model F/OSS in our framework we need specify some domain level policy to perform the work according to that domain. Sample policies require information such as assignment of projects to reviewers, assigning moderators to projects reviewed and so on. We model such interactions through events that share same names as actions in policy requests. For e.g. a transition labelled join req() would correspond to a join request sent to the 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.

Some of these rules are modelled as:

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

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

\[
\text{update\_level}(\text{CodesSubmitted}, \text{PercentAccept}, \text{Level}, \text{Role})
\] :=\text{admission}(\text{Role}), \text{CodesSubmitted} > 20, \text{PercentAccept} > 50,
4.2 Methodology

The main steps of the simulation are as follows:

1. At the start of the simulation we define a project which requires some certain number of codes with some time limit.

2. Each time some random peer comes and submits a code and send it to some reviewers to review. Module arrival for review is as per Poisson distribution.

3. If developer has got some particular amount of reviews (say 50%) from reviewers then he may send the code with all the ratings to the moderator. Moderator calculates the overall rating of the code and check with the access policy. If it is greater then the code will be added to the project.

4. If code is accepted moderator sends it to other moderators also signed with the level3 key because accepted code should be present at every other moderator.

5. Simulation runs until project gets the required number of codes. Events occurring continuously are new peers joining, existing peers level update and dynamic leaving of some peers. Events triggered by our policy could be level degrade of some peers.

4.2.1 Rate calculation

After reviewing is done by at least 50% of peers, code goes to any one expert moderator which calculates the overall rating of code by the formula:

$$Rate(c) = \frac{\sum R(c, y) \cdot T(y)}{\sum T(y)}$$  \hspace{1cm} (4.1)

$R(c, y)$ is the rate given by peer $y$ to code $c$, $T(y)$ is the trust value of peer $y$ and $Rate(c)$ is the overall rate of the code calculated using all the reviews it gets.

$$R(x, y) = \frac{\sum_{i=1}^{n} \alpha^i \cdot f_i(x, y)}{\sum_{i=1}^{n} \alpha^i}$$  \hspace{1cm} (4.2)

where $f_i(x, y)$ denotes the $i$-th rating given by $y$ to $x$. $\alpha \in (0, 1]$ is a decay factor which indicates how important the most recent interaction is to the reputation.
4.2.2 Project Quality

For simplicity we model the job satisfaction as Project Quality where Project is defined as the number of modules needed & the estimated time. Cost factor of a project can depend on no. of peers working on the modules, total number of bugs, total no of lines of code, and time taken. For a single module we define Quality of project as

\[
\text{Quality of Project}(p) = \frac{\alpha \cdot \text{no of bugs}}{\frac{\text{Time taken}}{\text{Expected time}}}
\]  

(4.3)

We include the system scale factor as the no of modules completed, since in an open system as the scale of the system increases, it results in increase of workload, no of bugs and increase of communication. Thus Quality till j modules are completed would be a summation of above equation from i to j.

\(\alpha\) is the factor by which the quality of project decreases with number of bugs. As number of bugs increases, quality decreases exponentially at the rate of \(\alpha\). It also depends on the Time taken to complete it so if the project is completed in time less than what required we say that quality increases and if time taken is more we will say that quality decreases.

4.3 Goals

We want to be able to show the following aspects

1. Peers with good behavior rise to higher levels more quickly.
2. Average trust rate of expert peers is the highest.
3. Our decentralized methods of storing data and reputation values ensures that the group performance does not degrade as data is stored only with trusted peers.
4. Peers with bad performance are penalized and moved down the layers.
5. Allowing peers to dynamically and collaborative change group policies results in more efficient group formation.
6. Peers learn in the group according to a learning curve which then affects the growth curve.
Chapter 5

Trust and reputation management in Decentralized systems

One of the major challenges for collaborative P2P systems is the ability to manage risks involved in interacting and collaborating with previously unknown and maybe malicious peers. 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.

5.0.1 Requirement for Trust Dynamics

Trust is the firm belief in the competence of an entity to act independently [17][19], securely and reliably within a specified context. It 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. Thus dynamics of trust relates to evolution of trust over time and in the face of new experiences. Reputation is an expectation about an individual’s behavior based on information about or observations of its past behavior [18]. Reputation-based trust management allows each peer to manage its own trust, hence there is no global centralized map of trust relationships, and it isn’t convenient to manage.

Main requirements of a dynamic trust metric 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:

- **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[9] trust model

- **Social network-based trust management** where social relationships between peers are taken into account when computing trust relationships. e.g Regret, NodeRanking.

### 5.0.2 Decentralized Trust Management

In a Distributed trust model[5], trust relationship is always exactly between two entities and is conditionally transitive. It is either a Direct trust relationship where one peer trusts another based on direct interactions or a recommender trust relationship where a peer trusts other peers to give recommendations about another peer’s trustworthiness. 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

Different models have been proposed.

**Complaint based model** Alberer and Despotovic were the first to propose a reputation based trust management system for a decentralized peer-to-peer infrastructure []. This 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. However there is no measure to prevent peers from inserting arbitrary complaints about other peers. This scheme is adopted by P-Grid data structure.

**Voting based Reputation Model** XREP is a distributed protocol that allows these reputation values to be maintained and shared among the servants. It consists of the phases of resource searching, resource selection and vote polling, vote evaluation, best servant check, and resource downloading.

**eBay Reputation Management System** This does not include a trust computation mechanism [10]. It presents the reputation information about a user and leaves trust determination and decision making to its users.

**Credence** uses a voter correlation scheme that weighs peers opinions[15]. NICE platforms includes trust algorithms that enable good peers to form robust co-operative
groups with other good peers\cite{13}. Marti and Garcia propose a limited reputation sharing scheme to reduce the no. of failed interactions between peers. EigenTrust proposed by Kamvar et al. is a reputation system that computes local trust values which are distributed in order to help reduce the number of inauthentic files in the network, so that the impact of malicious peers is minimized. It aggregates the local trust assessments of all peers in the network in an efficient and distributed manner, but introduces significant computational overhead. PeerTrust is a reputation based trust model for e-commerce communities\cite{12}. It focuses on incorporating the credibility of recommenders in the evaluation of trust. It also includes the number of transactions and transaction community context to make trust evaluation more accurate.

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 cooperation 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.

5.1 Threats for Peer to Peer Groups

The two primary types of adversaries in peer-to-peer networks are selfish peers and malicious peers. 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 and Gnutella. To minimize their cost in bandwidth and CPU utilization freeriders 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. Reputation system designers usually target a certain type of adversary. For instance, incentive schemes that encourage cooperation may work well against selfish peers but be ineffective against malicious peers. The number or fraction of peers that are adversaries also impacts design.

5.1.1 Adversaries

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. Some of the adversaries for P2P networks are:

**Traitors:** 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. An example of traitors are eBay \cite{10} merchants that participate in many small transactions in order to build up a high positive reputation, and then defraud one or more buyers on a high-priced item. Traitors may also be the computers of well-behaved users that have been compromised through a virus or trojan horse. These machines will act to further the goals of the malicious user that
subverted them.

**Collusion:** In many situations multiple malicious peers acting together can cause more damage than each acting independently. This is especially true in peer-to-peer reputation systems, where covert affiliations are untraceable and the opinions of unknown peers impacts one’s decisions. Most research devoted to defeating collusion assume that if a group of peers collude they act as a single unit, each peer being fully aware of the information and intent of every other colluding peer.

**Front peers:** Also referred to as moles, 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 [20].

**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 [16].
Chapter 6

Simulation and Experiments

We have modelled P2P groups using a java based overlay simulator namely PeerSim. We modelled our access control policies using SWI-Prolog which we integrated with Peersim at runtime wherein java and prolog talk to each other by means of sockets. We modelled the F/OSS application as a P2P group with some varying values of initial number of nodes as 20,40,80,100, etc and took sample random runs for 100,200,400 events where peers trigger events such as join, leave, updating, code submission and code review. The events have been triggered in the event mode of peersim. Some sample roles modelled are reviewer, developer and moderator of code. As the group evolves with random runs of peersim events, the policies get modified as per current group composition which in turn depends on the behavior of peers in the group, their trust levels and the original group charter global policy. Figure 6.1 depicts the overall system model. There are certain assumptions which have been done for simulations.

![Figure 6.1: Overall Framework](image-url)

25
6.1 Assumptions

We make some assumptions here

- No of bugs is a Uniform Distribution
- Module arrival is as per poisson distribution.
- Average completion time of a module depends on difficulty level.
- A project in our model consists of different modules of different difficulty levels.
- The group could be working simultaneously on multiple projects
- Modules could run concurrently, currently we have not assumed any interdependence between modules.
- Attributes considered while rating the performance of a peer developing some code is No of bugs, Time Taken, Lines of Code.

A project module consists of following parameters:

- Starting time
- Current time (clock of every module is different and works concurrently)
- Time of completion
- Expected time of completion
- Status (finished, started, working)
- Bugs (number of bugs)

Peer behavior considered for our simulations is shown in table below:

<table>
<thead>
<tr>
<th>No. of bugs probability</th>
<th>Expert Peer</th>
<th>Average Peer</th>
<th>Ok Peer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Time taken</td>
<td>[0,1]</td>
<td>[0,1]</td>
<td>[0,1,2,3]</td>
</tr>
</tbody>
</table>

We are varying the percentage of expert coders, average coders and ok coders for our simulations. The policy parameters for the acceptance of join and the updation of level is as follows:

<table>
<thead>
<tr>
<th>Request</th>
<th>Role</th>
<th>Level</th>
<th>Vote(%)</th>
<th>Rate(/10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Join</td>
<td>Member</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>Join</td>
<td>Admission</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>3</td>
<td>Join</td>
<td>Control</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>4</td>
<td>Join</td>
<td>Member</td>
<td>2</td>
<td>–</td>
</tr>
<tr>
<td>5</td>
<td>Update</td>
<td>Member</td>
<td>2</td>
<td>–</td>
</tr>
<tr>
<td>6</td>
<td>Update</td>
<td>Admission</td>
<td>2</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>Update</td>
<td>Control</td>
<td>2</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>Update</td>
<td>Member</td>
<td>3</td>
<td>–</td>
</tr>
<tr>
<td>9</td>
<td>Update</td>
<td>Admission</td>
<td>2</td>
<td>–</td>
</tr>
<tr>
<td>10</td>
<td>Update</td>
<td>Control</td>
<td>2</td>
<td>–</td>
</tr>
</tbody>
</table>
While initializing the simulation we have fixed the reputation/trust value of the peers at different levels as follows:

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

Also when some new peer joins the group we initialize its initial trust value as 0.4 only.

6.2 Integrating prolog at run time

For prolog and java to run simultaneously we need to have a library which acts as a bridge between java and prolog. Some of the common Prolog-Java interfaces are Interprolog, JIProlog, CiaoProlog and PrologCafe. We found Interprolog suitable for our purpose as it directly loads the prolog file from java which was required for the regular interaction between java and prolog.

InterProlog [31] is an open source library for developing Java + Prolog applications. Currently it supports XSB and SWI Prolog on Windows, Linux and Mac OS. It promotes coarse-grained integration between logic and object-oriented layers, by providing the ability to bidirectionally map any class data structure to a Prolog term; integration is done either through the Java Native Interface or TCP/IP sockets.

InterProlog 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 we have to first install XSB or SWI Prolog System on our system. We are using XSB [33] Prolog system in this project.

6.2.1 Example Java Program

```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],_),"+
"ipObjectSpec('java.lang.Integer',Obj2,[Level],_), belongs(Peer,Level),
"[Obj1, Obj2]", var);

System.out.println(b);
```

Above code is the sample code which initializes the prolog engine and loads the prolog file. After loading prolog file java objects can be passed to prolog and verification can be done according to our policy rules and appropriate decisions are taken.
6.3 Implementation

As mentioned in chapter 3 we have modelled our policies using prolog and we give here some sample rules and explain them.

6.3.1 Modelling rules for join

An example of rules for peer with unique identity to join as a member peer in a group at level1 of a groups is

\[
\text{verify}(N_{\text{peer}}, \text{Request}, R_l, \text{Level}, \text{Rate}, \text{Vote}) :\neg \text{join(\text{Request})}, \text{member}(R_l), \text{Level} =:= 1, \text{Vote} >= 40. 
\]

Verify function takes six parameters as Peer Id, type of request, Role, Level, Overall Rating and ‘yes’ vote percentage. If the RHS is true for all given cases then function will return true. To call it from java, we first pass above six parameters from java itself and call the verify function. Interprolog reads the input as a string compiles it and then passes it to the XSB Prolog for verification.

6.3.2 Modelling rules for level updation

Similarly rule for update request will be:

\[
\text{verify}(N_{\text{peer}}, \text{Request}, R_l, \text{Level}, \text{Rate}, \text{Vote}) :\neg \text{update(\text{Request})}, \text{maximal}(R_l), \text{Level} =:= 2, \text{Rate} >= 6. 
\]

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.

6.4 Results

We varied our initial network size from 20 peers to 40, 100 and 200 peers and took sample runs. Average of 3 runs in each case was considered for plotting the graphs. For a initial peer network size of 40 we considered a total of 20 modules while for a network size of 100 we considered 50 modules. For each case, we varied the policy parameters.

Graph in Figure 1 is a plot of the project quality versus time. The policies were varied dynamically to restrict join of admission peer to 40% while varying policies for update of control and admission peers. When the policy for update of control peers was changed dynamically to tighten when 10 and 20 % control peers joined the project quality output appeared to be better.

Graphs in Figure 2 and 3 are plots of the average load per admission peer and control peer respectively.

Graph in Figure 3 is a plot of the average trust value of peers with time. Here we observe that the rate of increase of average trust value of expert peers is much faster as compared to the other peers.
Figure 6.2: Project Quality vs Time taken at code acceptance rate of 7

Figure 6.3: Load (no of messages passed) of admission peers vs Time taken
Figure 6.4: Load (no of messages passed) of control peers vs Time taken

Figure 6.5: Average trust value vs Time taken at code acceptance rate of 6
Chapter 7

Conclusion and Future Work

We have modelled a dynamic policy driven trust based access control model for P2P applications. The model permits peers to collaboratively frame, and modify policies at various levels. The policy also permits peers to update their access rights within a group. Join priorities can also be decided by the group policy and depend upon current group composition. Changes to the policy itself can obviously affect the set of actions that are permitted or denied. Group behavior is defined as a system state and the group can have state transitions based on dynamic policy changes.

We have run a few sets of experiments by modelling self-organizing group F/OSS. We modelled different peer behavior and different task graphs for the group.

Experiments show that we have a model capable of testing different parameters in the behavior of self-organizing groups in presence of different dynamic policies.

7.1 Future Scope

We plan to conduct several more experiments with different parameters to validate our framework. We would like to build an adaptive trust framework capable of handling malicious peers with temporal behavioral changes. We would also like to incorporate collaborative re-inforcement learning algorithm into our framework, so that peer behavior can be continuously evolving which is a major requirement of adaptive self-organizing groups.
Bibliography


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


[31] Interprolog www.declarativa.com/interprolog


[33] XSB xsb.sourceforge.net/manual1/index.html

[34] F/OSS Documentation http://foss.in