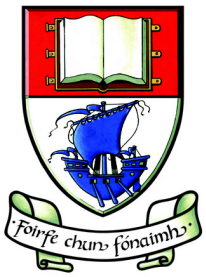
# Managing the Formation and Interaction of Groups within Emerging Social Networks




Leigh Griffin, BSc

Department of Computing, Mathematics and Physics

Waterford Institute of Technology

Thesis submitted in partial fulfilment of the requirements for the award of

*Doctor of Philosophy*

Supervisors: Dr. Dmitri Botvich and Mr. Eamonn de Leastar

September 2012

# Dedication

To my parents

Bernadette & Gerard Griffin

and

To the memory of my grandmother

Madeleine Griffin

# Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy, is entirely my own work and has not been taken from the work of others save to the extent that such work has been cited and acknowledged within the text of my work.

Signed:..... ID: 20006077
Date: September 2012

# Acknowledgements

I would first like to extend my thanks to my supervisor team of Dr. Dmitri Botvich and Mr. Eamonn de Leastar for their guidance and support throughout my PhD. In particular to Eamonn, for the hours of guidance, motivation and conversations since I joined the TSSG and without whom this thesis would simply not exist.

I would also like to thank all the colleagues, both past and present, that I have worked with in the TSSG, there are too many to name individually. I would like to particularly acknowledge the support of my fellow students, especially Brian, Cathal, Stepan and Julien whom I have shared an office and friendship with for many years, thank you for any help offered along the way. I wish you all a speedy and successful conclusion to your studies. To previous students, in particular Ray and Will, your achievements motivated me and gave me the direction to drive on, for that I thank you. I would also like to sincerely thank the broader research support within the TSSG, both postdoctoral and administrative, for giving me the chance to pursue my research. This thesis would not have been possible without the education received from Mount Sion CBS and fostered within Waterford Institute of Technology. It was not just an education, but a way of life and a chance to cement lifelong friendships.

To my parents, Bernadette and Gerard, who have supported me all my life, thank you for giving me the opportunity to pursue my education to the highest level. I promise I am done now and you can have the house to yourselves. To my brother Gerard and his wife Jenny, thanks for the encouragement and belief over the years, just don't put Jack through all this. To my wider family and friends, thank you for being there when I needed you. Finally, to Aisling, who met me at the start of this journey and who has encouraged and stood by me through to the end. I couldn't have asked for someone more loyal and committed to share this journey, you have my gratitude and love.

# Publications

- Griffin, L., Ryan, K., de Leastar, E., Jennings, B. & Botvich, D. (2012). On the Performance of Access Control Policy Evaluation. In Proceedings of the 2012 IEEE International Symposium on Policies for Distributed Systems and Networks, POLICY '12. Griffin *et al.* (2012b)

- Griffin, L., Ryan, K., de Leastar, E. & Botvich, D. (2012). Scaling Instant Messaging Communication Services: A Comparison of Blocking and Non-Blocking Techniques. International Journal of Ambient Computing and Intelligence, 4, 120. Griffin *et al.* (2012a)

- Griffin, L., de Leastar, E. & Botvich, D. (2011). Dynamic Shared Groups Within XMPP: An investigation of the XMPP Group Model. In Proceedings of the 2011 IEEE Symposium on Integrated Network Management, IM '11. Griffin *et al.* (2011a)

- Griffin, L., Ryan, K., de Leastar, E. & Botvich, D. (2011). Scaling Instant Messaging Communication Services: A Comparison of Blocking and Non-Blocking Techniques. In Proceedings of the 2011 IEEE Symposium on Computers and Communications, ISCC '11. Griffin *et al.* (2011c)

- Foley, C., Power, G., Griffin, L., Chen, C., Donnelly, N. & de Leastar, E. (2010). Service Group Management Facilitated by DSL Driven Policies in Embedded Middleware. In Proceedings of the IEEE Symposium on Computers and Communications, ISCC '10. Foley *et al.* (2010)

Additional Publications:

- Griffin, L., Elger, P. & de Leastar, E. (2011). Project Zeppelin: A Modern Web Application Development Framework. In Proceedings of the International Symposium on Formal Methods for Components and Objects, FMCO '11. Griffin *et al.* (2011b)

- Storni, C. & Griffin, L. (2009). Towards Future Health Social Networking: Patient Generated Content And The Role Of Community Pharmacists. In Proceedings of the The 4th Mediterranean Conference on Information Systems, MCIS '09. Storni & Griffin (2009)

- Griffin, L. & de Leastar, E. (2009). Social Networking Healthcare. In Proceedings of the 6th International Workshop on Wearable Micro and Nanosystems for Personalised Health, pHealth '09. Griffin & de Leastar (2009)

- Griffin, L., Foley, C. and de Leastar, E. (2009). A Hybrid Architectural Style for Complex Healthcare Scenarios. In Proceedings of the Communications Workshops, IEEE International Conference on Communication, ICC '09. Griffin *et al.* (2009)

- Griffin, L. & de Leastar, E. (2008). A Model for IM and Media Driven Communication Services, In Proceedings of the 8th International Conference on Information Technology and Telecommunication, IT&T '08. Griffin & de Leastar (2008)

**Managing the Formation and Interaction of Groups within Emerging Social Networks**

**Leigh Griffin, BSc.**

**Supervisors: Dr. Dmitri Botvich and Mr. Eamonn de Leastar**

**Abstract**

The formation, evolution and management of groups within modern social networks pose significant challenges and opportunities. Challenges arise from a combination of ubiquitous connectivity coupled with innovative and inexpensive cloud infrastructure, yielding new usage patterns which push existing service architectures, management infrastructure and technology stacks to their limits. Opportunities for remedying this situation arise from re-thinking the formation and management of social media groups and introducing innovative models, notations, algorithms and implementation strategies. To this end, this work analyses emerging group formation and interaction patterns, applies new approaches to current prominent protocols and evaluates resultant behavior and performance characteristics. This analysis is applied to more sophisticated usage patterns; models, notations and algorithms are devised to flexibly support emerging scenarios, and specific implementations are discussed and analysed. Applying Policy Based Network Management principles to manage the formation of, and interaction within, groups is demonstrated to be viable, particularly if coupled with emerging technology stacks. An innovative architecture model is evolved, which can flexibly match the performance and scalability requirements likely to emerge in next generation social network platforms. In addition, an approach to policy language development is proposed and demonstrated, capitalising on recent innovations in scripting language design and implementation. This is shown to deliver a considerably simplified, more flexible and more expressive alternative to current policy languages, and offers significant scope for further research and innovation.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Listings

# Chapter 1

# Introduction

## 1.1 Motivation

Usage patterns for Internet services and applications have evolved dramatically in the first decade of this century. In the earlier years, usage was dominated by person-to-service interaction, most often via browsing conventional static web sites, accessing email or using Internet Relay Chat (IRC) or Instant Messaging (IM) communication services. Rapid advances in network and computer infrastructure, and particularly the evolution of mobile, smart, always-connected devices, have produced a dramatic shift in usage patterns. This has culminated in the phenomenal rise of social networks. These represent a significant change in the nature of how users view and access services and applications. Conventional person-to-service interaction is now matched and often superseded by services that offer person-to-person and person-to-group experiences.

Groups have long been a natural meeting point for individuals to share experiences, make new connections and work towards a common goal. The evolution of groups to an online medium is a natural progression with the advent of new communication technology, allowing real-world interactions to take place in a virtual manner. For the purpose of this thesis I define a virtual community to be *an online community facilitated by applications and services to allow people to make a connection with others and interact via some computer mediated mechanism.* Additionally, I define a group to be *a subset of a virtual community with a membership base predominantly made up of people, but capable of hosting services and devices, with associated membership criteria.* Communities of interest, centered on groups, now have a larger and more active membership than previously possible. This has ensured significant new dynamics within a group. With the removal of physical requirements and associated costs, groups could

now becoming larger than was previously manageable. Interestingly, groups could also become smaller, but more numerous, yet still function and attain their set goals. Groups can also becoming more specialised and their membership more diverse. The lifecycle of a group could fluctuate to extremes, with transient groups numbering as many as persistent groups. The combination of changes with respect to size and lifecycle has also changed the usage patterns associated with groups. Group members have grown accustomed to requesting and retrieving information on the move, accessing services and regularly sharing experiences within their social network. Service providers have responded with new forms of services that support highly dynamic social networks.

Groups can be thought of as a subset of a virtual community populated by members of a given social network. Often this is facilitated by group communication technologies, empowering users with the capability of connecting and communicating with others. These technologies can represent standalone groups, with each user evolving their own grouping mechanisms to permit interactions and support membership of a group. The most popular and visible groups are those that exist within social networking media, particularly messaging platforms. Allowing users to segment their connections into logical groupings, joining individuals in interest based groups and creating events are some of the ways that groups are currently modeled. The concept of a group is evolving with applications and services driving the creation of small communities with internal grouping. As such, groups are becoming more popular and more visible, further driving their adoption by users. Group communication through social networks not only provides a medium for useful public announcements but a means to reach an audience that previously would not have been possible. To this end, social networks, have been used as a mass communication medium for significant social, political, economic and natural disasters in recent years (Potts (2009)). Acknowledging that rescue services have responsibilities to the public to move information as quickly as possible, officially endorsed groups became a means of information dispersal enabling members of the public to make key decisions on informed and reliable data. Groups used within this manner involve significant user interaction and information flow. However, this usage may not be adequately provisioned within the design of host technology.

Groups in their current format face significant challenges. The current group support mechanisms center on ease of use and membership control. The actual interactions are often designated to existing group communication technologies, with the role of the group largely a facilitation mechanism, enabling this initial connection to be made. As such, groups in current social networks could be described as organisational tools for structuring a social network experience. However, groups are emerging as viable

platforms for mass messaging, high volume communication, service access and service consumption. Groups are thus moving beyond structural / organisational constructs and now face challenges in two key domains:

**Group Formation**

For the purpose of this thesis, I define group formation to be *a mechanism which enables the establishment of a subset of a virtual community, while taking into account an appropriate set of constraints.* Currently group formation mechanism can be described as being weakly modeled, not yet attaining to any clear set of criteria, standards or commonly held best practice. The mechanisms are generally not influenced by the purpose of the group, the requirements for interactions within the group or the associated resources necessary to support the group. Additionally, as service deployments are increasingly becoming more flexible, service providers are empowering users with a means to create and deploy their own services. In this context, services may be poised to adopt group characteristics. Membership may thus consist of people, services and devices, becoming a way of structuring the emerging *Internet of Things* (Atzori *et al.* (2010)). How they interact will impact on the resources required to manage and support the group throughout its lifecycle. If groups become a focal point for user interaction in this manner, how the group forms will crucially dictate the resources required, impact the performance profile of the group and ultimately the cost to the provider. A new approach to managing the formation of groups that is flexible and tailored towards individual groups will need to be devised.

**Interaction Management**

Interaction Management within groups is defined for the purpose of this thesis as *the management decisions required to govern the interaction of an entity with another entity within the context of a group.* Managing interactions among users and services is a reasonably well researched and understood domain. However, management within the context of a groups is relatively new and currently weakly modeled. A range of management principles are at work within current social networks, providing various mechanisms to manage privacy concerns (Jones & O'Neill (2010)). Groups, however, are poised to become considerably more dynamic and diverse. The evolution of groups to embody services, users and devices will test current management approaches and technology stacks to their limits. A solution that will not impact on the performance or timeliness of user interaction will be essential.

## 1.2 Hypothesis

The principal hypothesis of this thesis is that a model can be evolved with accompanying algorithms and implementation recommendations that can adequately support the demands of group based interactions within emerging social networks. Specifically, a lightweight management system, capable of handling emerging usage patterns, can support group formation and management in a performant, cost effective and flexible manner.

By adhering to such a model, a number of specific problems that currently limit the usefulness and effectiveness of group based interactions can be addressed. By adapting management principles and deploying them within the proposed architecture, complex events can be handled in a timely manner, optimising the resource allocation, scalability and flexibility of group formation and management.

## 1.3 Research Questions

This thesis addresses the following research questions split into two sections:

**Research Questions on Group Formation and Management (RQ-GFM)**

**RQ-GFM1** What management mechanisms for Group Formation can be sufficiently flexible to adapt to emerging usage patterns?

Users of existing social networking services and media are interacting with the technology in a manner not envisioned within the original design. The scale of these interactions can often be global, with the potential to be disruptive and costly to manage.

**RQ-GFM2** What platforms facilitate management of group interactions including near real time[1] interactions?

Managing the interactions within a group in a near real time manner requires a management platform capable of processing and delivering a decision with near real time precision.

[1] The term near real time is used to describe programs that must guarantee a response within strict time constraints from receipt of an event to the systems response. The understanding of near real time in the context of this thesis is without perceivable delay

**RQ-GFM3** What policy representations can facilitate group management?

A number of viable formats to represent the structure and semantics of rules that govern a system are available. This thesis attempts to discover what representations can facilitate group management and the associated requirements it might have.

**Research Questions on Scalability (RQ-S)**

**RQ-S1** What is the scalability limits of the current solutions/approaches for group formation and interaction management?

Discovering the scalability limits of current solutions and approaches to group formation and interaction management can lead to recommendations based on solutions to bottlenecks. This investigation is necessary to validate any model or recommendation produced.

**RQ-S2** What performance profiles can be expected in emerging group interactions?

Understanding the performance requirements for existing and future use cases within group based interactions can better inform the underlying management platform.

**RQ-S3** What techniques, tools or algorithms can be deployed to meet these scalability and performance requirements?

Understanding the technology choices to help realise the scalability and performance requirements set forth can accompany the model produced as guideline implementation strategies.

## 1.4 Contributions

In this section, the contributions provided by this thesis are described. In line with the research questions and hypothesis outlined above, the contributions have been separated into three categories. First is the contribution towards the management of group formation and then the management of group interactions already formed. The final contribution is a potential application domain for group formation. These contributions are presented by models, algorithms and implementation recommendations ob-

tained through empirical testing and analysis of existing group management principles in diverse domains.

- Management of Group Formation
  - An outline model and accompanying algorithms for the management and formation of groups within emerging social networks.
  - Scalable components produced that realise key model characteristics and validated in high throughput simulations
  - An implementation stack, capable of managing varying extremes of group and membership sizes, that can supplement or even replace aspects of conventional management middleware
- Management of Group Interactions
  - A management model capable of handling emerging usage patterns not currently provisioned for in existing group management domains
  - A lightweight, responsive and highly scalable domain specific group management model capable of providing interaction management with minimal overhead and designed specifically for an individual group.
  - A novel policy representation format capable of syntactically being interpreted by non domain experts while retaining the management semantics required.
- Application of Group Formation
  - A case study was presented for humanitarian relief, forming and managing groups as communication mediums for disaster management coordination.

## 1.5 Thesis Outline

Chapter 2 discusses the state of the art in a number of related areas of research. The usage of groups and the communication mechanisms employed are highly relevant for highlighting the changing usage and general expectations of the user base. As such, the evolution of these approaches and their state of the art are described. As this thesis focuses on management principles, research from the general network management field within telecommunications is discussed. Additionally, a review of best practices

in software engineering is presented. This research helps motivate and inform the experiments and subsequent outputs of this thesis.

In Chapter 3 approaches to group formation and management from the point of view of group membership is presented. A representative social network is introduced and analysed, with a set of criteria emerging for how groups should be managed going forward. This analysis forms the basis of several empirical investigations within the domain, discovering and exploring the limits of the technology. A set of principles and general understanding of how to manage and represent group membership as well as the formation of groups is abstracted out.

Chapter 4 explores the role of management platforms within interaction management. An analysis of architecting management principles is presented and applied to a group based domain which may benefit from interaction management. The results of these observations drives an explorative analysis of current best practices. A competing solution, architected from technological observations successful in web based deployments is presented. The performance and behavior of the competing solution against the industry standard leads to a further revision, driven by best practices within the software domain. This analysis forms the basis for how to structure and manage interactions within a group as well as the supporting vocabulary required.

In Chapter 5, the work carried out in the previous chapters is used to inform an architectural framework for managing the formation of groups and interactions within the domain of emerging social networks. This framework is presented as a set of UML models, complete with implementation fragments and associated commentary. A complementary discussion on implementation recommendations to help realise this model concludes the chapter.

Finally, Chapter 6 presents the conclusions drawn based on the research carried out, the experiments undertaken and the analysis of the subsequent results. A number of possible future areas of research which may lead to further interesting investigations and studies are also discussed.

# Chapter 2

# State of the Art

In this chapter a review of the literature relevant to this thesis is presented for consideration. Topics covered include Group Formation and Management, Group Communication Technologies, Interaction Management within Groups and Design and Implementation Strategies for Scalable Solutions.

## 2.1 Group Formation

First, the role of the group in modern society is presented using relevant historical literature. The state of the art in group formation within existing group based environments is also presented.

### 2.1.1 Motivation for Group Membership

The ability of an individual to interact with other individuals is a fundamental social behavior that is prevalent in all human societies (Ray & Liew (2003)). Traditionally, individuals in a society, or group, interact with one another with an aim to improve either the individual, the collective or both. Individuals do much better collectively compared to the case when they forage on their own (Gazi & Passino (2004)). This instinct to join and participate in a community environment has placed the concept of a group at the core of research in many domains from psychology, to marketing, to sociology and indeed nature. Each domain has evolved numerous models of group behavior and performance, yet the majority of research assumes that the group exists before the model is applied. Yet groups must form before they can achieve a goal, with the formation stage establishing the framework necessary to act towards the goal (Owens *et al.* (1998)). The formation of natural social groups emerged as an area

of research with the goal of improving the performance of groups within a working environment. Levine & Mooreland (1991) and Moreland & Levine (1992) proposed a group formation model that included three phases: *evaluation, commitment, and role transition*. The evaluation phase serves as an investigative phase whereby the individual looks for a group that can satisfy their personal needs and the group looks for individuals to satisfy its goals. If commitment between the individual and group becomes strong enough then a transition occurs, whereby the individual attains group membership. Applications of this process can be seen in the formation of groups within environments such as education or the workforce to tackle a specific task. In such domains, the formation stage needs to identify personal traits such as experience and expertise and use the formation to strategically meet goals. The success of the group and ultimately its lifecycle, is thus linked closely to the formation stage. The lifecycle of a group can play a part in the formation of other groups. The assumption that groups of various sizes split or merge into other groups is valid when environmental conditions, encounters with other groups and internal group dynamics are taken into consideration (Gazi & Passino (2004)). Group evolution or migration is often driven by individuals. Leader figures within groups, whether formally recognised or not, communicate and collaborate externally with other leaders. While cooperation binds society into communities, it can also divide communities (Hui *et al.* (2011)). Intersociety information exchanges often lead to a leader figure migrating to, or forming, a new group, paving the way for a critical mass that affects the lifecycle of several groups and becomes a catalyst for new formations. Ray & Liew (2003) also discuss this migration feature with leaders of societies attempting to attract other leaders. Leader migration may be achieved through an information acquisition from a better-performing leader. This can have knock on effects on the population of other groups, ultimately causing membership migration and groups to expire.

### 2.1.2 Groups: A Modern Analysis

In modern society, the formation of a group, and its societal role, has centered around a specific theme, topic or activity. The formation process in this instance is thematically focused. Groups of this manner are still popular today for community meet ups in the form of coffee mornings, book clubs, cinema clubs, sports groups etc. The popularity and accessibility of modern computing has evolved this offline activity to *have a longer reach* while retaining some of the characteristics already described. The requirement for members to physically come together is removed, online groups may span beyond traditional geographic boundaries. The scale of formed groups is thus no longer limited by

physical boundaries and combined with the exponential rise in accessible information, more diverse groups are forming while retaining a thematic focus. It could be argued that a modern trend is to make the web more people focused. As a result, groups are becoming focal points for users interacting with the wider world (Adams (2011)). The model proposed by Levine & Mooreland (1991) and Moreland & Levine (1992) still holds true for forming groups in an online capacity. Rapid advances within networking infrastructure and the availability of affordable devices has meant people are dynamically networked on the move. Assignment to groups, invitation to join groups as well as free enrollment and confirmed enrollment (Haake *et al.* (2004)) dictate participation levels and formation principles. The lowering and sometimes removal of procedures to formally attain group membership is making the process a passive experience, often a by product of an action or inaction on the behalf of the user. With an increase in the number of group entry mechanisms, the style of group formed has deviated from the original goal orientation of *offline* groups. The notion of bringing together individuals to satisfy a goal or requirement still exists, but as the formation of groups has become more ad-hoc and unplanned, a goal is often missing, or replaced entirely by a topic of interest to bind the community together. Groups numbering in the thousands are now forming which can be difficult to micro manage towards an end goal and much larger than traditional interest based groups.

Digital groups have allowed for the development of non-participation, facilitated in part by the current model for groups within a social networking context being passive or read-only pages with minimal traffic. Non-participation is something that would be difficult to achieve in a face-to-face group. The notion of being active or passive within a group is now possible, with the latter term also known as *lurking* (Rafaeli *et al.* (2004)). Participating in a group via communication media or contributing to activities is central to the lifecycle of that group. However many people prefer lurking. The reasons presented for lurking are varied, limited knowledge within the domain and thus lurking for self improvement, character reasons (shyness, personality) or just simply having nothing to say or contribute. Despite the interactive potential of the web, much of the use of the web is in read-only mode so lurking is a natural activity in one sense. With the evolution of groups poised to become more service centric, the effect of a lurker might be more noticeable. If resource or service provisioning algorithms are solely based on activity rather than membership, lurkers could significantly impact on this. If the services offered are innovative or popular enough to be consumed by the entire membership base, potentially an under-provisioning of resources could occur.

### 2.1.3 Group Formation

Group formation has different definitions depending on the domain, with the definition dependent on the group purpose and behavior (Ounnas (2010)). For the purpose of this thesis, in Section 1.1, group formation was defined as *a mechanism which enables the establishment of a subset of a virtual community, while taking into account an appropriate set of constraints.* The area of research in computer group formation has focused almost exclusively on this definition of creating a group, bounded by constraints that proves to be a solution to a specific goal. These requirements can often be broken down into mathematical terms, known as Constraint Satisfaction Problems (CSP), whereby the characteristics of a set of objects must satisfy a number of constraints (Kolaitis & Vardi (2000)). CSPs can be highly complex (Bulatov (2011)) and the requirements presented in order to meet the goal, which in the context of a group is a successful formation, can be difficult to meet. This is particularly true with conflicting requirements and incomplete metadata surrounding the subjects within the community. The possibility of an *orphan* problem can arise when members of the community are not allocated into newly formed groups, requiring manual intervention to complete the formation and allocation. In Ounnas *et al.* (2008) the author developed an ontology which extended the popular Friend of a Friend (FOAF) metaphor as a means to describe people in order to build communities and social groupings. FOAF was used extensively in semantic web research in the field of community building through social relationships (Ding *et al.* (2005), Staab *et al.* (2005)). Semantically representing the data allowed a disjunctive logic programming style typically used within knowledge representation and reasoning to be applied to the problem domain. Strong and weak constraints were inferred from the requirements in order to solve the orphan problem. (Ounnas (2010)) outlined the constraint satisfaction problem and offered an alternative heuristic approach, utilising a clustering algorithm. Clustering is the assignment of objects into groups called clusters, with group members within the same cluster being more similar to each other than to members within another cluster. Clustering has successfully been used in social networks using content tagging (Lu *et al.* (2011)) in order to develop relationships between users and content, enabling recommendations to bring users together and form groups. Clustering is also relevant in domains where multiple layers of groups can form naturally. Grouping within gaming communities, in particular Massively Multiplayer Online (MMO) games (Sherlock (2007)), achieves different levels of grouping within the one virtual world and in external tools associated with the game. Defining and abstracting group membership within this context can be difficult without tools such as clustering, to logically group players based around their role within the game.

Group formation mechanisms have emerged in the field of mobile computing due to the availability of connected devices, which in a sense are socially aware and capable of connecting to communities (Zhang *et al.* (2011b)). One such formation technique involves geolocation, using mobile devices to form social networks based on the physical location of a person (Lubke *et al.* (2011)). Geofencing, a means of geographically bounding an area with a virtual fence, is a simple means of generating a group around a point of interest (Bareth *et al.* (2010)). Typically such points are chosen by service providers as part of a social networking influenced marketing promotion, for example forming a group near a shop to target potential customers. Huang & Liu (2009) notes that geospatial services are used independently to the value added services. Independently, the service acts almost in a passive manner with the information handed-off to another application to deliver the user value in a collaboration service. The authors propose that future geo-based services should be more integrated and capable of handling a lot of traffic, with users sharing information dynamically with others. This requirement can be difficult to provision for, with connectivity and resource based challenges already existing within mobile groups. Mobile users can form what is called an *ad-hoc* group, which is spontaneously deployed and established, generally with no prior infrastructure or resource based requirement. Such groups can suffer from network and service disruption due to the mobile nature of the group and the lack of dedicated available resources to support potentially large numbers of interactions (Myoupo *et al.* (2009)). Ad-hoc groups of a similar nature can also emerge within older systems where the community evolves a metaphor of a group (Hallberg *et al.* (2007)), where no group infrastructure is present. Lai & Turban (2008) note that within older mega sites, such as online stores, there is no explicit group formation, but users perform operations, both as individuals or as groups, such as review assessment, using techniques such as merchant and product reviews. These help form personal networks, such as groups of friends and interest based groups. However the interaction is limited and very much static in nature, with any real interactions handed-off to third party applications, many of which are capable of being consumed on a mobile device. Facilitating and provisioning for collaboration services is an emerging challenge currently not addressed adequately by existing infrastructure, particularly when deployed within a mobile environment.

Backstrom *et al.* (2006) examined group formation in large social networks. Specifically, they examined the membership and growth of social networks to discover what makes individuals join social network groups. The first question the researchers pursued concerned what influenced a person to join a group. Despite the different backgrounds

of the social networks examined, the authors discovered that the results were very similar. Backstrom *et al.* (2006) with this research question, are seeking to tackle a fundamental question about the evolution of communities, seeking to determine who will join in the future. Diffusion of innovation is presented as a possible means to why people join groups. People are most likely to join groups based on the number of friends who are presently members of the group and the broad range of features (services, resources, community aspects) that are available within the group. This analysis maps to social networking trends with the decline of one social network often timed with the explosive growth of another (Carlsson (2010)). Understanding the potential for growth and usage patterns within the community is very important for future proofing and ensuring a smooth scaling of the underlying hardware. Similarly, understanding the relationships among users and the overall social structure of the group can dictate the network performance within the group (Hui *et al.* (2011)). Formation analysis in that respect is very much linked to the overall lifecycle of the group, and by extension, the underling social network.

Online groups are a place for interaction and communication. That interaction is facilitated by communication technologies which come with an associated overhead. Service oriented computing is playing an important role in emerging social networks with the inclusion of web driven services (Zeng *et al.* (2004), Maaradji *et al.* (2010)). Manual tasks are being replaced by automated services, facilitating an ease-of-use for the user across the social networking experience. Examples of such services include searching for and connecting with a friend through recommendation services (Chen *et al.* (2008)) and integrating calendar events mined from information submitted to social networks (Zhao *et al.* (2012)). The trend is making these services available through cloud computing infrastructure, simply termed *the cloud*. Provisioning services in this manner ensures complete availability and reliability for the consumer. Supporting these services in a scalable manner is a challenge for administrators of large communities, particularly when charging schemes are placing an increasing emphasis on resource utilisation within the cloud. Under-provisioning communication related services, such as available bandwidth, can have an adverse affect on the performance of the group and the capability for group members to interact. Over-provisioning resources can similarly cause adverse effects in other groups but can additionally cause spikes in costs, potentially moving the group beyond an agreed Service Level Agreement (SLA) and into another charging bracket. SLAs are designed to protect both consumer and provider, controlling resource provisioning within clouds in order to minimise breaches and costs, and maximise profit (Das (2012)). Buyya *et al.* (2011) notes that the next computing

paradigm is utility computing, with computing services offered whenever a user needs them, transforming services into commodities. Cloud computing is poised to become the deployment domain with SLAs protecting and guaranteeing the resources to be provisioned. As service requirements for a group can change over time and may require amendments from a management perspective, it is important to minimise breaches related to this. Classifying the group and preparing for eventual usage at the formation stage can help mitigate costs, ensure stability and meet QoS targets and workload demand patterns.

### 2.1.4 Summary

This section presented an overview of some of the reasons why people are drawn towards groups. The evolution of groups into the digital realm has also changed how people interact with groups and view their participation within groups. Behavior patterns, such as non participation have not been addressed from a resource provisioning point of view. Indeed formation itself is a secondary feature, with resources currently not a factor when deploying groups in social networks. This oversight is due to how groups are currently used, with the usage of groups often limited to being a passive experience. Technology sets outside of social networks are encouraging more and more group based communication and interaction which will ultimately lead to a shift in how groups are viewed. Group platforms are not currently informed by user behaviour patterns. The research question RQ-GFM1 tackles emerging usage patterns which can facilitate answers to the Research Questions on Scalability (RQ-S).

## 2.2 Group Communication Technologies

Group communication has evolved dramatically since the turn of the century. What emerged from fragmented technologies, used by a minority of internet users, has evolved into consolidated applications with mass adoption. This platform allowed communication service driven applications to flourish and develop, with an ease of use that allowed more users participate. The desire to stay connected proved compelling, as technology caught up with expectations. This allowed groups and communities to transition from the offline world into cyberspace, with suites of services capable of supporting and encouraging grouping. This section identifies several group communication media.

### 2.2.1 Mailing Lists

The Simple Mail Transfer Protocol (SMTP) is the standard protocol for electronic mail, or email, transmission and is specified in Klensin (2008). Mailing lists are a special kind of email that allows for widespread distribution of information to many subscribed users. An email list is a self enclosed group, with membership consisting of individuals interested in the subject matter associated with the list. Email lists come in two varieties, an announcement list and a discussion list. An announcement list is a read only list with emails capable of being sent from selected users (Shihab *et al.* (2010)). The discussion list on the other hand allows any member of the list to reply to the general list, as well as direct a reply through the carbon copy mechanism to a specific post, thus providing context to the response.

In both cases, emails sent to the list are distributed to the membership by a multi-destination delivery of a single message, with threads of conversation possible by replying to the subject. Such a community encourages focused discussions around topics of interest, encouraging peer interaction and knowledge dissemination. The users of a mailing list, in effect, form a social network. In this network, relationships among users can be built up publicly and analysed (Chen *et al.* (2006)). What is noteworthy is the low ratio of candidate experts in mailing lists in comparison to the number of subscribed users. The experts are generally at the center of the community and relationships built privately among experts, as well as topics discussed publicly, can provide the nucleus for a new topic of discussion, with a new mailing list required. The migrations among leaders, as discussed in Ray & Liew (2003), is very much observed within this domain. A limitation of mailing lists, as a group communication medium, is the type of information that can be exchanged. Mailing lists are designed for dynamic information sharing among a community, with static content difficult to share and maintain. This leads to supplementary technologies adopting the role of static information sources, such as wikis (Eto *et al.* (2005)) which can be standalone or integrated, and knowledgebase systems, which can be integrated with information derived from the lists (Watanabe *et al.* (2004)).

Technology has empowered end users with the capability of forming mailing lists, previously an operation that required domain knowledge and often administrative access. Management of mailing lists, particularly large mailing lists (Westine & Postel (1991)) has also simplified with technology. Message filtering and the administrative checks on the veracity of messages have helped alleviate the amount of spam and other disruptive mails designed to cause a denial of service or inconvenience to the users. Improvements including automated membership through email headers and opt

in membership over time has drastically reduced the time required to manage the list effectively. Mailing list scalability and reliability has also achieved technological solutions derived from how users wish to consume their information, for example message digests (Vuillemot *et al.* (2011)), a means of receiving messages in bulk at set intervals.

### 2.2.2 Message Boards

Bulletin boards were a place where members of the public could place notices and messages for others to see. This medium of communication proved useful and transferred to the online world in the form of message boards or forums. The benefit of moving this online was the possibility for users to reply to notices and messages left, effectively creating a thread of conversation, in a time lapse manner, where replies could be left at a users convenience. An online community space designed to foster conversations in a public manner was a major selling point of this technology. Other members can pick up on a conversational thread, in some sense, a recorded artifact, and join in the conversation or start another thread from it (Churchill & Nelson (2007)). This makes for an attractive feature set, with conversations being synchronous or asynchronous and the audience intimate or vast.

Members of a message board also form a type of social network community, with capabilities added over time to include a friend or foe list, capable of tracking or ignoring posts from the member base respectively. Enhancements to message boards have included user customisation, integrating with other communication technologies such as email lists and Instant Messaging and enriching content available within the forum (Kinsella *et al.* (2010)). A user can list personal details, advertise their willingness to communicate via other mediums and share content through a profile. Internal grouping is possible within message boards, with forums representing individual groups themselves. Membership identity is strong within such communities, with many users identifying themselves as a member of a particular forum or subforum rather than the global message board platform they belong to. This behaviour arises out of social identity and reputation among the user base, with a large number of users providing an individual with the opportunity to interact with others, gaining reputation and recognition from them in the process. The more valuable the contribution to the community or individual forum, the more recognition their contribution receives, with the bonds of identity and association with a particular thread or forum growing stronger (Jiang & Carroll (2009), Chen *et al.* (2009)).

Message boards have also been used as low entry barrier to Information and Communication Technologies (ICT) for connecting an older generation of adults with limited

computer skills. Work carried out in Gonzalez *et al.* (2008) showed how simplified interfaces and implementation decisions can open up group communication mediums to demographics previously unable to access such material. This introduction to ICT has paved the way for educating a new user base by providing a purpose for communication software and a goal, in this case connecting families.

### 2.2.3 Usenet Groups

Usenet is a worldwide distributed internet discussion system that predates the modern internet by almost a decade (Kantor & Lapsley (1986)). The basic framework allows the transfer of mail and files, with the ability for users to post articles and share with others, with a standard for interchange of messages defined (Horton & Adams (1987)). The articles posted are categorised topically and placed into newsgroups. These individual groups are thematically focused and often moderated to ensure coherence. Role identification is an important topic within Usenet. The membership base of a group requires a mix of contributers, posters, lurkers, anonymous downloaders and moderators in order to flourish. This provides a membership base which is difficult to quantify, as the social network is effectively defined by the topical group. With minimal membership criteria and the possibility of anonymous interactions, identifying relationships, and thus the nucleus of a social network between users, is difficult within Usenet.

Usenets interaction model is slightly different to traditional group based communication technologies. A person does not reply directly to someone else, but rather posts a message in response to another message. The distinction is subtle, but important, as a person could post a follow up message to a thread without giving any thought to the person who is being responded to (Fisher *et al.* (2006). Another limiting factor for Usenets evolution to a social network is the signal to noise ratio of valuable content. The environment can easily become noisy and hard to navigate with poor quality messages rendering the quest for valuable content too difficult or cumbersome to pursue (Viegas & Smith (2004)). Despite this, Usenet was designed for a time when networks were much slower and not always available. Strategies adopted saw messages often delivered in bulk and thus easier to digest, or attempts to serve data faster, with techniques such as caching used (Gschwind & Hauswirth (1999)). The prevalent problem of spam became apparent in latter years as transmission of usenet articles became instant. Usenet has diminished in importance with respect to other group communication mediums but the architecture and principles behind it have paved the way for how people communicate in a group oriented manner, influencing the design principles behind modern web based interactions (Schwartz (2006)).

### 2.2.4 Peer To Peer Networks

Peer to Peer (P2P) is a term used in many contexts. The definition put forward within Camarillo (2009) states that *a system is P2P if the elements that form the system share their resources in order to provide the service the system has been designed to provide.* From a group communication point of view, a popular implementation of the definition put forward for the P2P paradigm is within the BitTorrent protocol (Cohen (2008)). BitTorrent is a protocol used to distribute files by breaking them up into several pieces and making pieces available for others to download and assemble on their client. In this manner, files can be shared, with a request to assemble a file fulfilled by multiple peers, each offering individual pieces of the overall file. This domain has a strong group aspect associated with it, with groups forming around the availability and distribution of content.

Efficiently searching for files, and thus reducing the noise within the overall network, is a popular research topic to aid the scalability and efficiency of P2P interactions (Haw *et al.* (2009), Lv *et al.* (2002)). Group management within such an environment is weakly modeled due to the nature of files being the focal point of groups. Ownership and management are thus hard to provision largely due to the anonymous and open nature of the network. User driven management, with respect to content, has largely focused on trust and reputation (Kamvar *et al.* (2003)) as a means of filtering out peers and content which could be potentially harmful. Communities have built up around the notion of content sharing, with popular websites and services having a membership base, with group communication services including email lists, message boards and Instant Messaging, facilitating a sense of community. Research has tried to layer community principles within existing P2P communities (Cheng & Vassileva (2005)) with mixed success. The anonymous nature, often associated with the copyright and legal implication of what is shared (Pouwelse *et al.* (2008)), is a barrier to a complete social network evolving out of the paradigm.

### 2.2.5 Internet Relay Chatrooms

Internet Relay Chat (IRC) is a form of client-server, real-time, synchronous, internet text messaging chat, designed for group communication which came into widespread use in the early 1990s (Oikarinen & Reed (1993)). Its original purpose was to facilitate live group chat among message board communities, with one to one communication and private messages allowed. As well as standard text based chat, data transfers were also facilitated. An IRC network consists of several servers, which helps the

performance and resilience of the system. Users connect to a server and join existing channels or create their own. Channels are used to facilitate the group conversation and are typically identified by topical names in order to keep the conversations focused somewhat. Search facilities to discover and index rooms were initially lacking but were added over time as a response to the popularity and ease of use of the service (Haveliwala (2002)).

The flexibility of the system is such that any user could create, or form a new channel if it didnt exist already, allowing the service to evolve to meet current trends and demands without the need for an external administrator to establish the channel. Social Networks evolve around popular channels, with users automatically joining them on login, via their client, and thus becoming members. Membership however is typically limited to the current session, bringing a dynamic fluctuating nature to the group. Mutton (2004) mapped out a social network based on an IRC channel, showing its evolution and decay over various time periods. Active users, the centrality of knowledge generators and the relationships among users are possible to infer from such analysis. Understanding such relationships and the importance of contributers to groups can drive the creation of new channels and the destruction of existing channels. IRC has minimal barriers to channel creation, with the process facilitated and enabled by the protocol. Management within IRC is formally defined in Kalt (2000), specifying namespaces for channels and a series of flags to provide administrative controls such as invite only channels or limiting the number of users participating.

### 2.2.6 Instant Messaging Technologies

Instant Messaging (IM) is a form of near real time communication between two or more people over a network such as the internet. IM enables short message exchanges between online users (Chatterjee *et al.* (2005)) through a client server architecture. Early IM implementations were text based, one to one chats between two users following the same principles as IRC. A defining difference between IRC and IM was the integration of a service known as presence. This provided users with the capability to advertise their willingness to be contacted and interacted with. The technology grew in popularity, particularly among the teenager demographic (Grinter & Palen (2002)) and advances were made to the infrastructure, both server and client side, to meet current demands and user expectations. Services inspired by other domains began integrating into the paradigm such as Multi User Chat, allowing for several users to communicate in a virtual room. Modern incarnations of IM have seen an increase in the functionality offered to the user. The text transmitted can now be formatted in rich text or HTML

(HyperText Markup Language). File transfer functionality was implemented allowing for large files to be transferred across the network from client to client. Clients were capable of receiving and consuming multimedia, often via mobile devices (Patterson *et al.* (2008)). This proved to be a significant upgrade on the file transfer offered by message boards and became a standard service to facilitate. A vocabulary has also developed around IM which users feel comfortable with. A user with an IM account can send an invite to a friend for them to be their *buddy*. Buddies are stored in *buddy lists* or *rosters* which are associated with the users account, allowing for them to log in on any device and still have the same buddy list associated with them. These buddy lists can be broken down into *groups*. A group will typically comprise of buddies with a similar relationship to the user. Some examples might be a group of friends, a group of family or a group of work colleagues.

Implementing protocols of interest include the Session Initiation Protocol (SIP) (Rosenberg *et al.* (2002)), which specified a means for provisioning IM within the protocol (Campbell *et al.* (2002)). ICQ[1] (Weverka (2001)) an Instant Messaging program used by AOL[2]. MSNP (Microsoft Notification Protocol) an Instant Messaging protocol developed by Microsoft and powering the MSN[3]and Windows Live[4] suite of services. The most popular implementation however is the Extensible Messaging and Presence Protocol (XMPP), the driving protocol behind IM services such as GTalk[5]. XMPP is often termed Jabber, and is specified by Saint-Andre (2004b). The remainder of this section and aspects of this thesis will focus on XMPP. File transfer as a service was formally integrated into the XMPP protocol, providing additional services within the protocol for users to consume (Ludwig *et al.* (2009)). The capability for the protocol to adapt and often amalgamate other group communication technologies, has allowed Instant Messaging to evolve into a Social Network, with groups, relationships and interactions all governed within the one technology suite. This advancement has seen non human entities, in the form of devices and services, being represented on the roster (and in groups) as buddies. This change to the usage pattern of the protocol, in addition to advances described on the client side, has changed the traffic profile within Instant Messaging. Xiao *et al.* (2007) notes that rosters are becoming larger, service interaction is becoming more popular and the traffic generated from IM clients is quickly becoming problematic for service providers

[1]The term ICQ is a homophone for the phrase "I seek you"
[2]America Online (AOL) http://www.aol.com/
[3]MSN Messenger was the original IM client later rebranded and renamed Windows Live
[4]Windows Live Messenger http://windows.microsoft.com/en-us/messenger/home
[5]Google Talk Messenger http://www.google.com/talk/

### 2.2.7 Web based Social Networking

With the users perception of the internet evolving, a new technological savvy generation had emerged into a world that was experiencing a computing paradigm change. High speed broadband networks were emerging, mobile, ubiquitous and pervasive computing brought hereto unseen penetration of connected devices. The original group communication technologies outlined in this section were being adapted and modified to meet this new generations demands. The powerful resources available meant an overhaul of the popular group mechanisms, while never straying far from their original purpose. The modern evolution of group communication technologies is social networking, in particular, web based social networking which takes on a number of forms.

Online Groups are provided by some websites as a place where a group of people can come together and have discussions about common interests and meet like-minded individuals, doing so in a public or private manner. Online Groups were in a sense an evolution of traditional collaboration software, made available publicly and inherently more usable. The term Groupware, another term for collaboration software, is defined by Johnson-Lenz & Johnson-Lenz (1991), as *intentional group processes plus software to support them.* Proprietary software products such as Lotus Notes[1] and Microsoft Exchange[2] provided integrated collaboration functionality through a suite of group communication technologies including email, group calendaring and instant messaging. These communication platforms allowed individuals, largely those of a professional nature, to effectively work in a coordinated manner towards a common goal. Service providers such as Google[3], Yahoo [4] and Windows Live [5], offered a free combined group service, mirroring the features of traditional Groupware and making it available to a larger membership. This grouping mechanism was the first such attempts at large scale integrated group communication technologies. The sites offer countless free groups around general topics such as health, sports and news with an abundance of sub groups looking at particular themes. Registered users can join these groups and avail of the outlined functionality, participating in a public manner. Users were also allowed to establish their own private group to encourage participation. A private group contains the exact same functionality as the public groups except membership is limited and exclusive. The founder of the group can set up the criteria for joining it. It might be limited to direct invites from the founder or a password to gain access. This

[1] Lotus Notes http://www-01.ibm.com/software/lotus/products/notes/
[2] Microsoft Exchange http://www.microsoft.com/exchange/en
[3] https://groups.google.com
[4] http://groups.yahoo.com/
[5] https://groups.live.com/

allows for a more personal experience within the group, with the membership base generally being small and focused for the groups purpose. Such groups had limited resources provisioned for file sharing, expanding in recent years as technology became more affordable.

Web based social networking emerged from online groups with individuals constructing a public or semi-public profile within a bounded system, with the intention of interacting on a more personal level with others. Identifying a list of other users with whom they share a connection, users can view and traverse their list of connections and those made by others within the system (Boyd & Ellison (2007)). Their social network is in effect a self contained group realised through intelligent interfaces and interactions. Timed with a more socially aware generation, several styles of social networking sites, appeared. These include what has come to be known as traditional social networking sites, including Bebo [1], Facebook[2], MySpace[3] and Google+[4]. Blogging and microblogging social networking sites, such as Twitter [5], Blogger[6], Wordpress[7] and Friend Feed[8]. Service oriented social networking is also another style as characterised initially by Gowalla[9] and later by FourSquare[10]. The initial concepts of groups within this generation of social networks was an extension of the standard profile with little to no additional group services offered. The principle idea was to use these groups in order to expand a persons direct contacts, a mechanism still achieved through their profile, thus using the overall service suite at a profile, rather than group level. The rationale behind keeping interactions at a profile leve,l rather than a group level, is not quiet clear. Concerns about scalability is a possible explanation as the volume of users social networking sites attract has caused disruptions in the past, which have had to be technologically addressed (Eriksen (2010)).

Group communication through social networks not only provides a medium for useful public announcements, but a means to reach an audience that previously would not have been possible. To this end, social networks have been used as a mass communication medium for significant social, political, economic and natural disasters in recent years (Potts (2009)). One of the early adopters of social networking by an Emergency

[1] http://bebo.com
[2] http://facebook.com
[3] http://myspace.com
[4] http://plus.google.com
[5] http://twitter.com
[6] http://blogger.com
[7] http://wordpress.com
[8] http://friendfeed.com
[9] Now working with Facebook as their check-in service
[10] https://foursquare.com/

Service is the Los Angeles Fire Department (LAFD). The LAFD has a twitter account (LAFD (2011)) reporting status updates, via short messages known as tweets, of current open cases to serve as a public warning system. Their reports include current fires, road traffic accidents and environmental problems which might affect residents of the city of Los Angeles. Resolutions of cases are also tweeted providing a reassurance for the public. The LAFD used twitter successfully during wild fires in 2007 as a means of distributing information for the public during an environmental disaster that threatened the city. The LAFD began looking at Web 2.0 technologies after the devastation of Hurricane Katrina in August 2005. They noted that those stranded in New Orleans were in serious distress from a lack of information (Havenstein (2007)). Acknowledging that rescue services have responsibilities to the public to move information as quickly as possible, the LAFD adopted twitter as a means of information dispersal, enabling residents to make key decisions on informed and reliable data. In recent natural disasters, social networks took central stage as an information medium, this time driven by the people. The Great Tohoku earthquake and subsequent devastating tsunami that struck Japan on March 11th 2011 was the fifth biggest earthquake ever recorded. Towns were wiped off the map, over 27,000 people died or were missing and a nuclear crisis was averted by the authorities. In the immediate aftermath of the earthquake, the emergency services blocked voice communication channels to prioritise their traffic and co-ordinate the rescue effort. The data network however remained intact, with the media reporting that Twitter was the only communication tool functional after the earthquake. Users took to Twitter to report their location and seek help for themselves and for others. A study carried out by Acar & Muraki (2011), whom analysed the usage of Twitter for crisis communication, highlights the lessons that were learned from the Japanese usage of Twitter during this emergency. The emergency response teams rescued people from information received via Twitter, however, effectively managing and controlling this information is a difficult task, particularly when the usage was not provisioned for within the design of the social network. A lack of formal groups to facilitate, sanitise and present information was a noted limitation.

Using Facebook as an example, Facebooks initial attempts at group integration had three distinct styles, two of which were officially supported and one of which evolved from user behavior. *Networks*, where one could join based on profile information such as location, educational institution or employer. These networks served little purpose only to allow an additional means for users to connect with others. The second group style promoted was the *like page*. Here users could create a page based on a topic or concept and people could join the page by *liking* it, an act of publicly declaring an

interest in a topic such that it appears on the newsfeed of your buddylist. These pages functioned the same as a profile page with the ability to post pictures and messages for everyone to see. Users circumvented the lack of groups by creating a third style, *the profile group*. In this instance a user would create a new profile, typically promoting an event or business and add friends to their buddylist to create a community via the newsfeed. Facebook responded by introducing events as a separate service and introducing a formal representation for groups, as well as means to recommend groups to others (Baatarjav *et al.* (2008)). Now users can create a formal group and invite others into it. Groups can be public or private and full administrative control is possible. Photos can be shared and private events organised within the group, with richer service consumption not yet provisioned for.

The Social Networking domain is a competitive industry generating billions in revenue annually (Rushe (2012)). The emergence of new platforms and subsequent migration of users tends to stem from technological problems and usability issues. Groups were not defined and implemented coherently enough, as the Facebook style of one giant buddylist and aiming updates en masse promoted Google to release Google Plus (Google+). Observations by Adams (2011) showed how people are drawn towards small groups and more focused groups of friends. The size of buddylists or rosters on existing social networks was not sustainable and Google+ attempted to address that by bringing in Circles. A Circle, effectively is a group, allows you to place friends into categories. Interaction could then be tailored directly at small groups rather than an entire list allowing tighter control of what you release and whom you release it to. Privacy and security management within social networks has allowed fine grained access control for content generated by users (Fang & LeFevre (2010), Jones & O'Neill (2010)). The circle idea is an additional layer of privacy for sharing content.

### 2.2.8 Semantically Enhanced Groups

Group Communication was originally designed to facilitate participation and discussions among members. Technological advances were made in supporting fields, allowing group communication to leverage some of these features. The user base had also profoundly changed, with the need for users to stay connected and informed of the status and activities of others in their online address book. This desire for information led to the development of applications with added contextual information, enhancing the user experience. These semantic additions are present within existing group infrastructure and will be more prevalent in emerging social networks as social applications. Social applications refer to a class of applications that integrate with one or more social net-

works (Zolfaghar & Aghaie (2012)). Applications on social networks are increasingly becoming a focal point for group formation and a means of generating a group full of like minded individuals. Masked as community games and retaining the goal of groups, that being a common purpose or task, applications are quickly becoming an additional means of group formation within social networks. Social Applications are designed to be cross platform, accessible on mobile devices, potentially provisioned in the cloud or in standalone hardware and designed to emulate native applications. While the evolution of this trend towards socialisation has added more incentive to participate in groups it has potentially raised more questions about the structure of groups and their long term scalability. The effect of socialisation is that it can cause some applications to become very popular amongst a user base very quickly. This is often referred to as viral spread because the application is passed from one user to others in their social graph (Facebook (2012)) and then on to friends of friends. The impact of this viral spread on an application is to cause sudden spikes in server load for a web application. Presence, Location and Context are three such services deployed within semantically enhanced groups, which are in turn impacting on how groups are used and how they form.

Presence is a current status indicator showing the ability and willingness of a user to communicate across a range of devices. The presence mechanism is designed to allow a real time update for a person, keeping their friends informed. It is typically employed in instant messaging applications to show a persons friends list what they are doing. Typical default presence statuses include *chatty, online, offline, busy, do not disturb,* and *away.* The presence feature can be used for custom messages and many applications integrate with this feature. Messenger programs have developed plugins allowing a users music or video choices be echoed as a presence status to the persons buddylist. GPS technology on mobile phones can advertise the persons current location through the presence mechanism. Presence is also at the core of enabling protocols such as XMPP (Saint-Andre (2004b)) and can cause issues when used outside of the intended scope of the protocol (Xiao *et al.* (2007)). Privacy concerns with the usage of presence as a service medium (Wu (2007)) within the context of an entire buddylist is making groups a more attractive place to direct presence and retain a form of control over the privacy and security of the updates.

The desire of people to stay connected and inform friends and family of their current activity and location has led to the integration of location based technology into devices. Mobile phones, laptops and even watches have the capability to track their location. Services have been built to take this data and use it to inform others, often through

a group communication medium such as social networks, posting the information to a newsfeed. Location Based Services, termed Geospatial services (Granell *et al.* (2010) are playing an increasing role in social networks, moving beyond personal usage to become a marketing tool, a gaming mechanism as well as a formal outlet for handling emergencies and announcements. The notion of a check-in is a geolocation announcement to a social group advertising your presence at a particular location. Services have emerged to support users who have registered their location (Hsu *et al.* (2012)), however, the check-in notion has started to evolve away from the original geographic only usage. Check-ing into events such as sports performances, cinema performances or TV shows[1] has abstracted the core grouping centric logic from the requirement to physically be present. This has seen a scalability profile not yet associated with mainstream check-in services. It is conceivable, for a popular event, that the average daily check-in metrics of a geographical based service (Belic (2012)) could be exceeded within the opening credits of a popular TV show. That initial burst could potentially be handled by the current suite of services and infrastructure in place, but adequate service provisioning to target this newly created group is problematic. For example, layering a service on top of that, such as sending a Quick Response (QR) code with a discount, in order to deliver a marketing campaign, would create an enormous amount of strain on an underlying system.

In recent years the notion of context awareness has emerged in communication research, particularly in the field of ubiquitous computing (Beach *et al.* (2008), Raento *et al.* (2005)). Devices became smarter and started to make assumptions about the users current situation based on information being polled from the user and the environment. Watches with built in heart monitors, GPS enabled devices (laptop, mobile, watch), accelerometers on mobile phones and wireless health body kits (such as insulin monitors) are examples of context aware devices. These devices are enabling technologies, providing data to services for translation. Services can take this data and abstract understanding from it, providing a feature rich service for the end user. The users social environment i.e. the co-location of others, their social interaction and group dynamics, as outlined in Schmidt *et al.* (1999), plays a major role in context awareness. It is often the end consumer of such information as well as a major provider. Intelligence (Zhang *et al.* (2011a)) abstracted from the physical environment and community groups is driving innovative services from environmental services to urban sensing, such as traffic planning and public safety.

[1] For example http://getglue.com/ offers a means to check-in to TV, Movies and Music

### 2.2.9 Summary

Based on the work reviewed within the area of Group Communication Technologies a number of conclusions can be drawn. Group communication tools and mechanisms have facilitated user interaction in a scale that was not previously possible and RQ-S2 attempts to investigate this. Groups of contacts are now numbering in the hundreds and thousands, creating a large social sphere from which users can access. The evolution of supporting and access technologies has created a means by which people are comfortable accessing information, however those technologies have quantifiable limits (RQ-S1). Expectations from how to access and consume information, as well as the degree of management that users have over their interactions has traditionally been at a level just beyond the current scope of existing group communication technologies. This has driven the evolution and functionality of the current mechanisms and the same can be predicted for emerging social networks. Users have developed a social identity and social activities surrounding groups further enhances that. Participation within current social networking groups has declined to a more passive form, largely due to their current implementation. As semantically enhanced applications and usage patterns, possibly driven from marketing campaigns, turn towards active participation, a scalability conundrum has to be addressed.

## 2.3 Interaction Management Within Groups

### 2.3.1 Interaction Management

For the purpose of this thesis, in Section 1.1 a definition for interaction management was presented as *the management decision required to govern the interaction of an entity with another entity within the context of a group*. Interaction Management is dependent on the structure of the group and the nature of the group. The group membership can be assigned *roles* within the group, placing responsibility and often management roles on a number of users. One such role can be termed a moderator, who presides over a group, adjudicating on the interactions of the users to ensure that group rules are adhered to (Dennis & Wixom (2002)). Moderation is very much a reactive process to interaction management, otherwise it may present a barrier to actual interactions, particularly if messages or content has to be approved before publication. Effective moderation should facilitate dialogue and interactions within the community and not present a barrier to usability (Heinze & Procter (2006)). Interactions also govern the access and consumption of content within the group. A traditional means of

controlling this interaction is the use of Role Based Access Control (RBAC) (Sandhu *et al.* (1996)). Permissions are associated with roles, governing what a member of the group is allowed to interact with. RBAC has been successfully deployed within group communication technologies (Park & Hwang (2003), Maruoka *et al.* (2008)) and RBAC rule generation has close ties into the formation stage of a community. Constraints outlined at formation time can influence the rules that are required in terms of what level of access the group participants might require (Ahn & Sandhu (2000)). Automated versions of RBAC, such as granting roles through group delegation (Wang *et al.* (2009)), have addressed some of the manual issues with deploying an RBAC system. Attempts to move away from traditional RBAC models have seen management protocols emerge as an alternative in order to effectively provide large group management in a scalable manner, by customising the interaction media. Ponnusamy *et al.* (2003) investigated managing a group of hosts and their network activities, managing the membership of the group and the interactions that occur within it. Liu *et al.* (2005) similarly investigated managing groups over mobile ad hoc networks. Both bodies of research take a networking perspective on managing interactions, relying heavily on traditional networking, failing to take into account user behavior and treating the domain as a classical networking problem.

### 2.3.2 Policy Based Network Management

The management of interactions among entities within a network is often referred to as Policy Based Network Management (PBNM). This section deals specifically with access control within this domain, with respect to the access of services and content. PBNM is a condition, action, response mechanism, designed to provide an automated response to conditions within a network according to pre-defined rules encoded within policies. The definition put forward by Strassner (2003), and supported by the wording within RFC 3198 (Westerinen *et al.* (2001)), defines a Policy as *a set of rules that are used to manage and control the changing and/or maintaining of the state of one or more managed objects*. Westerinen *et al.* (2001) also defines responsibilities for core elements that interact with Policies, namely the Policy Decision Point (PDP) and Policy Enforcement Point (PEP). The definitions put forward and used within this thesis, are as follows:

- PDP: The system entity that evaluates requests against applicable policies and renders an authorisation decision
- PEP: The system entity that performs access control, by routing requests and

enforcing received authorisation decisions.

Policies are initially described in high level natural language and are capable of describing a range of activities ranging from predefined tasks, which must occur, performance related metrics, which may be protected by Service Level Agreements (SLA), and simple access control rules. Examples include:

- Allow the admin group write access to the membership database
- Run backups at midnight
- Network availability must be 99% over the course of the week

Understanding the variety of requests that may be encoded within policies and that may manifest itself as an access control request is important. The high level examples above are all variants of an access control policy.

### 2.3.3 XACML

The eXtensible Access Control Modeling Language (XACML) is an OASIS (Organisation for Advancement of Structured Information Standards) standardised specification for an access control policy language and request language. XACML is implemented in the eXtensible Markup Language (XML), with an accompanying processing model to evaluate access control requests according to the rules contained within the policies (Moses (2005)). XACML extensively references Westerinen *et al.* (2001) ensuring commonality with respect to definitions and terminology used within the world. XACML defines access control as *controlling access in accordance with a policy* returning an authorisation decision, typically *permit, deny* or *Not Applicable*, derived from evaluating the applicable policy with respect to the incoming request, as returned by the PDP to the PEP. This tightly defined sequence of events has ensured that XACML has become an industry standard for access control and the management of interactions, deployed within multiple domains.

#### 2.3.3.1 Policy Authoring and Conflict

Mapping XACML closer to the source domain which needs to be controlled has potential benefits. The researchers in Mourad *et al.* (2011) deployed an abstract language on top of XACML in order to provider greater control over web service security. A user friendly language deployed on top of XACML provides a layer of abstraction on top of

the heavy XML policies which can be difficult to read and interpret from an end user perspective. Attempts to make policy authoring more transparent and user friendly is a popular theme within XACML research. The price of simplifying the authoring is often conflicts appearing within the policies. A conflict, in policy terms, is when multiple policies are applicable to the same request, but result in different responses. XACML provides combining algorithms such as *permit-overrides*, where the first *permit* will override the overall decision, among other algorithms (Huonder (2010)). The inbuilt combining algorithms can often be difficult to apply, particularly when XACML has grown out beyond its initial intention of a single domain environment. Federated domains and multiple layers of policies bring additional complications to detecting and resolving conflicts (Barron *et al.* (2011)). A drawback of integrating ease of policy authoring mechanisms and the associated infrastructure required to ensure conformity, with respect to conflict analysis, is the size of the overall deployed system. The potential impact on the response times if policy conflict analysis has to run constantly can be profound in a high volume request environment. In an environment with a large number of policies, the time it takes to converge on a conflict can be in the order of minutes (Wang *et al.* (2002)). The broader execution context, such as the impact on CPU and memory resources, is often ignored when evaluating policy authoring and conflict tools.

#### 2.3.3.2 Policy Performance

The performance of a PDP is an important access control system requirement. It is relatively easy to scale out the stateless PEP functionality, but the stateful PDP function is difficult (Butler *et al.* (2010)). Therefore having an efficient and performant PDP is a critical requirement, particularly in domains where a large volume of lightweight requests are issued. Making XACML policy evaluation more efficient is difficult from the point of view of the complex structures that are contained within XACML policies. Liu *et al.* (2008) proposed a radical rethinking of XACML, converting policies and requests from XML to a numerical format. A supporting prototype PDP, capable of understanding the numerical format, showed dramatic performance improvements over the traditional PDP implementations that used XML. It is difficult however to abstract the performance gain from the representation format and the implementation logic behind the PDP. Liu *et al.* (2011) examined the algorithms implemented within XACML identifying problem scenarios which XACML currently struggles with. Additional solution algorithms were presented as a means to empower the PDP with logical choices that would represent a performance improvement when dealing with policy requests

from the problem domains identified.

Bindings to XACML are a popular way of increasing evaluation and performance by making the PDPs more flexible and less reliant on external systems. Mirza (2011) presented a means of translating REST (Representational State Transfer) based requests in order for a single PDP to be able to interpret requests from multiple PEPs. REST is a popular style of software architecture for distributed systems supporting web services (Fielding (2000)). The work carried out largely centered around SOAP (Simple Object Access Protocol) (Box *et al.* (2000)) based PDPs receiving non SOAP queries which traditionally would have been an interoperability issue requiring an external translation, pushing out the overall request-response time. Securing communication with group environments (Sjöholm *et al.* (2008)) has also been explored within XACML. Available research in this domain seems to focus on the surrounding architecture in low volume transactions. The performance profile at larger levels of scale in a traditional social network are quiet varied. Access control has not been deployed within this domain largely due to a lack of privacy protection when dealing with third party applications and a lack of interoperability among different Access Control Policy Languages (Carreras *et al.* (2009)). Considerations for response times within such an environment have not been explored extensively as a consequence.

#### 2.3.3.3 Policy Representation

Policy representation within XACML, by its specification, was bounded to XML. The extensible nature of the specification allowed for the development of XACML within diverse fields. The authors of Hsieh *et al.* (2009) embedded access control policies within digital content using a XACML like document. This approach encoded the item to be protected with the protecting policy. By encoding both resource and policy information within one single XACML document, the PDP no longer requires a large scale search of a policy store, as all relevant information is at hand. The research carried out implemented a number of prototypes, largely focusing on the authoring of the single XACML document. While no performance characteristics were recorded, the approach showed that a rethink on the structure of a request and policy could bring performance benefits from a PDP point of view.

Liu *et al.* (2008) took a numerical approach to policy representation as a means of policy normalisation. The idea of policy normalisation is to develop a common policy language or normal form to represent policies from any application. The major benefits put forward include a normal form that has a simple logical structure and the reuse of policy evaluation algorithms. The resulting representation however is opaque, non-

symbolic and difficult to analyse. Presenting such a representation to domain experts and XACML experts alike would be difficult, despite the performance benefits presented in the research.

### 2.3.4 Summary

Strong management principles have relied extensively on classical networking approaches, with a human centric mechanism a secondary requirement to securing the network. The rulebases are difficult to interpret and establish, with administration often limited to a number of individuals. XACML, one of the most successful access control management platforms for management interactions is currently being deployed in domains outside of the scope of the original intention of the specification. The heavyweight nature of the design and the all encompassing specification, which must be met in its entirety, to ensure conformance, can have a negative impact on the performance and scalability of a system. Implementing a Group Management policy based system, as per RQ-GFM3, is possible but has the potential to conflict with the requirements of RQ-GFM2. XACML is not currently suitable for deployment within large social networks due to the sheer number of requests and the requirement that interactions occur with near real time precision, so as to not interrupt the user experience. The language and structure currently used within policy representation is difficult to understand for a non technical expert. Added to this difficulty is the interpretation time and *scaffolding* required to interpret the overall policy system. A rethink is therefore necessary to bring XACML principles into the management of groups, in a performant manner (RQ-GFM1).

## 2.4 Design and Implementation Strategies for Scalable Solutions

This section deals with implementation strategies widely adopted across a number of domains within modern computing. As implementation strategies evolve, new uses not previously considered within disparate domains might benefit from a change in perspective. This section presents the current state of the art within implementation strategies showing where they derived from.

### 2.4.1 Design Patterns

Programming is a rich discipline and practical programming languages are usually quite comprehensive in breadth and terminology. Solving a programming problem re-

quires choosing the right concepts, with knowledge about the structure and strengths of programming languages vital (Abelson & Sussman (1996)). All but the smallest problems require different sets of concepts to address aspects of the problem at hand. This is why programming languages should support many paradigms. A programming paradigm is an approach to programming a computer based on a mathematical theory or a coherent set of principles. Each paradigm supports a set of concepts that makes it the best for a certain kind of problem (Assayag (2009)). Complementing a language paradigm is the concept of a design pattern, which is a general, repeatable solution to a commonly occurring problem. Gamma *et al.* (1994) presents a collection of popular design patterns tailored at solving the problems of object generation and interaction rather than a global solution. The patterns are broken down into three categories; *Creational Patterns*, used to provide ways to instantiate objects or groups of objects; *Structural Patterns* which define relationships among objects and *Behavioral Patterns* defining manners of communication among objects. Systems of patterns interacting with each other helped achieve an overall solution architecture that was maintainable over time. The POSA (Pattern-Oriented Software Architecture) movement began with Buschmann *et al.* (1996) and subsequently followed up with four more volumes (POSA 2-5)[1]. The volumes published represent a catalogue of design patterns that addressed core problems within software engineering, covering performance, availability and minimising risk to deployments. Developing software to pattern based specification provides a documentation of the system at a design level. In modern design methodologies, such as the Agile methodology (Cohn (2009)) or the Extreme (XP) Methodology (Beck & Andres (2004)), documentation is key. The ability to interact with the end user in an implementation independent manner is crucial for requirement gathering. The initial requirement gathering is supplemented by iterations of requirement meetings with the end user and responding to changes and requests in a timely manner. Design Patterns help in faster and more effective design for software, with reduction in effort achieved through making use of existing/standard patterns in design. Modern methodologies are suited to the current demands of shorter time frames and easily adaptable software (Dattatreya *et al.* (2012)). Marrying the two approaches has seen a greater understanding in how to produce sustainable software within tighter timeframes.

[1]The POSA series of books:
POSA2: Patterns for Concurrent and Networked Objects (Schmidt *et al.* (2000))
POSA3: Patterns for Resource Management (Kircher & Jain (2004))
POSA4: A Pattern Language for Distributed Computing (Buschmann *et al.* (2007a))
POSA5: On Patterns and Pattern Languages (Buschmann *et al.* (2007b))

### 2.4.2 Language Paradigms

An understanding of such design patterns, methodologies and paradigms was matched at an implementation level. A case for object technology as a solution to many issues within software engineering was investigated as far back as Simula 1 (Dahl (2002)), Simular 67 (Simula (2007)) and Smalltalk (Goldberg & Robson (1983)), culminating with the work of Meyer (1997). Object Oriented (OO) languages gained much traction as a result, with complementary design patterns (Gamma *et al.* (1994)) emerging. OO became one of the core reference paradigms due to the large uptake. The language style brought alternative mechanisms, including inheritance, object composition and polymorphism to bear on problems. This fresh perspective and innovative way of thinking was aided by the easier learning curve. Memory management (garbage collection), a means of dereferencing memory automatically, is a simple example of a complex mechanism now handled by the language (Wilson & Hayes (1991)). OO languages such as Java (Gosling *et al.* (2005)) gained widespread adoption. However, Java, among other languages, gained such popularity that it was inevitably used in scenarios that were outside the envisioned scope of the initial design. For example, Ostermann & Mezini (2001) showed OO languages to be weak at modeling non standard composition scenarios. However, the language evolved over time and addressed flaws and weaknesses (Nasseri & Counsell (2009)) to remain a staple language in modern implementations.

The code execution component of Java is termed the Java Virtual Machine (JVM) with the specification freely available and documented (Lindholm & Yellin (1999)). The JVM executes Java Byte code, an instruction set derived from the source language, which was primarily compiled Java programs. As Java Byte code was simply an instructional set, other languages availed of JVM compatibility by compiling down their source code into Java Byte code. Scala (Odersky *et al.* (2011)) and Groovy (Koenig *et al.* (2007)) are two such languages, although bindings exist for multiple languages Alez (2012). JVM compatibility is attractive due to the portable nature of how the code is executed, leading to the term *write once, run anywhere* (Sun Microsystems (1996)). Taking this approach, languages such as Groovy allow for direct Java authoring within Groovy classes. As the compiled output is Java Byte code, this is a valid approach. This allowed for a generation of programmers to adopt a new style, different in many respects to Java but allowing that safety net of pure Java coding to integrate with legacy systems. The JVM based languages that emerged brought with them a new set of features which programmers could avail of. Dynamic typing, where the type of a variable is not known until run time (van Noort *et al.* (2010)) and closures, where a safely scoped function can execute while retaining captured variable state (Jeannin

(2011)), emerged with these languages.

ECMAScript is a standardised object-oriented programming language for performing computations and manipulating computational objects within a host environment (ECMAScript (2011)). JavaScript (JS) is the more traditional name for ECMAScript, and JavaScript as a language is the most widely deployed programming language in history, with every browser ever developed containing a JavaScript interpreter. Initially regarded as a very limited language, its true nature and power had only recently been appreciated in any depth (Crockford (2008)). Deployment on the server side has been attempted in the past, (Husted & Kushlich (1999)), but the true power of the language in that regard only emerged due to constructive thinking around the concept of concurrency and the challenge of handling mass amounts of simultaneous connections.

### 2.4.3 Concurrency

The challenge of concurrency is made concrete by what is known as the C10K problem, first posed by Kegel (1999). The C10K problem is this: how can you service 10,000 concurrent clients on one machine. The idea is that you have 10,000 web browsers, or 10,000 mobile phones all asking the same single machine to provide a bank balance or process an e-commerce transaction. The reverse of this problem is a similar challenge (Liu & Deters (2009)). While the number of connections is largely arbitrary, the problem itself is a classical concurrency problem which saw attempts at programmatic solutions emerge to provide safe access to resources. Threads, a lightweight process termed a thread of execution, popular in some languages such as Java (Oaks & Wong (2004)), emerged as a solution. Threads evolved as technology became multiprocessor and parallel execution was sought, allowing multithreading, furthering the efforts to meet the C10K problem. In threaded systems, threads can share resources and memory but locks are associated with specific data structures. Emerging languages such as Scala took an alternative approach to memory management within threading, termed Actors (Haller & Odersky (2009)), which avoids shared data structures and consequently any resource locking. However, both approaches can be termed *Blocking Input/Output (I/O)* as separate threads are created with their own stacks and program counters. The opposite of Blocking I/O, termed Non-Blocking I/O is achieved through extensive use of callbacks in API design and usage and has been shown to address this C10K problem. In this instance, any possible opportunity for blocking, be it a computational intensive task or accessing a resource, is replaced by passing a callback parameter to be invoked on completion of the deferred task or I/O request. JavaScript is a Non Blocking language designed around callback execution and the development

of high performance network programs (Tilkov & Vinoski (2010)) is now a possibility. The popularity and widespread availability of JavaScript execution environments, may see an effort akin to the Byte Code compatibility of the JVM begin to emerge. Already some languages such as Smalltalk[1] and Java[2] can run on top of the JavaScript Engine. While this cross compatibility among languages is attractive, languages such as CoffeeScript (CS) have been designed specifically around compilation to Javascript (MacCaw (2011)) .

### 2.4.4 Domain Specific Languages

A Domain Specific Language (DSL) is defined as *a computer programming language of limited expressiveness focused on a particular domain* (Fowler (2010)). DSLs excel at taking certain narrow parts of programming and making them easier to understand and therefore quicker to write, quicker to modify, and less likely to breed bugs, overall increasing the productivity of the programmer. An additional benefit goes beyond programmers, to the field of domain experts or end users who can be exposed to the end code and make more informed input on the structure and design of the solution. Common to all DSLs is that they do this by making the data structures and operations from their problem domain the basic building blocks of the language (Van Cutsem (2008)). DSLs can come in two variants, an internal DSL and an external DSL (Fowler (2005)). An internal DSL is written and embedded within an existing host language. This can take the form of internal mini languages, where a subset of the overall language is used, or language enhancements, where DSL techniques through metaprogramming enhances the base language to make it look and feel like a DSL. The host language should provide a feature set to facilitate a DSL, the closure capability, for example, is a desirable feature. An external DSL has its own syntax with a full parser required to process the language, the freedom attained is at the cost of extensive development time. Valuable lessons were learned from initial DSL deployments and developments (Wile (2004)) and best practices established and documented (Volter (2009)). DSLs are much more mature with a support in the form of toolkits and design guidelines (Zdun (2010)) available to end users.

[1] Amber and Squeak are implementations of the Smalltalk language that runs on top of the JavaScript runtime: http://amber-lang.net/ and http://squeak.org respectively

[2] The Google Web Toolkit allows the authoring of JavaScript front end applications in Java: https://developers.google.com/web-toolkit/

### 2.4.5 Representation Formats

JavaScript Object Notation (JSON), (Crockford (2006a)) is a lightweight, text-based, language-independent data-interchange format, derived from the object literals of the ECMAScript programming language standard. JSON objects are analysed as string arrays, permitting higher parsing efficiency and easier preparation than heavier transport formats such as XML (Downes *et al.* (2010)). Crockford (2006b) describes JSON as *the fat-free alternative to XML* arguing that XML is not as well suited to data interchange as XML is much more verbose than JSON and rarely matches the data model of its host programming language. By contrast, JSON is built on just two data structures, a collection of name-value pairs and an ordered list of values. Having a data format that is interchangeable with a programming languages built in data structures eliminates translation time and reduces complexity and processing time. Furthermore, Wang (2011) notes that the strengths of XML are also present within JSON and the two are functionally equivalent.

Schemas are another means of representing metadata, in this instance a way of defining the structure, content and semantics of a system. Schemas are used to both validate the data entered into the system and document the design decisions of the administrator at the time of creation. Databases were one of the original domains where schemas were deployed. The structure of a database and the representation of the facts that can enter the database (Imielinski & Lipski (1982)) are contained within the schema, termed Document Type Definition (DTD) schemas. Schemas to represent XML based documents became popular with a number of styles developed such as Content Assembly Mechanism (CAM) (OASIS (2007)) and the Document Definition Markup Language (DDML) (Kimura (1998)). XML based Schemas (Campbell *et al.* (2003)) that are capable of documenting and validating themselves as well as XML documents became the de facto manner for Schema representations. However, XML Schemas have a high complexity (Martens *et al.* (2006)) which can present a barrier to users not familiar with XML syntax. Familiarity with XML does not guarantee an understanding of the schema, with large schemas becoming very verbose and difficult to interpret from a human point of view, as the language itself is XML and not readily human readable. Supplementary documentation and comments are often required to achieve a degree of understanding within a document. With the transition of languages, particularly DSLs, to become more readable in nature, representation formats are currently lagging behind. A re-imagination of representation formats has been attempted (van der Vlist (2007)) but a full syntax change and move away from static representation formats to a more dynamic model of representing and capturing schema semantics is still not

available. It may be the case that markup languages might not be suitable to describe meta data in a user centric manner.

### 2.4.6 Deployment Platforms

Cloud computing is one of the key drivers compelling the current wave of innovation (Liang *et al.* (2011)). For the first time, corporations are moving their sensitive data and operations outside of the building. They are placing mission critical systems into *the cloud* with computing capacity now metered by usage. Technology challenges are not solved by sinking more capital into powerful computing infrastructure. The model is moving away from the costly Infrastructure as a Service (IaaS), where companies provide physical host machines, to the more economical Platform as a Service (PaaS) (Garcia & Ketel (2012)). When Ruby on Rails (RoR) was launched it was a highly innovative development framework (Maximilien (2006),Viswanathan (2008)). The key driver for mass adoption of RoR was hugely increased developer productivity through *convention over configuration*, an approach which has now certainly entered the zeitgeist and which has been adopted by almost all of the current development stacks: for example Pythons Django[3] or PHPs Cake[4]. The predominant application deployment model for most organisations during the rise of RoR was owned server infrastructure: i.e. make some capital investment in server hardware on which to deploy applications. Under this model, operational expenditure was relatively static and was based on monthly costs for colocation and bandwidth. Operational efficiency of deployed, in the field, applications was not so important for anyone but the really large sites, as long as the application could scale horizontally to some degree, capacity could be added by purchasing more hardware and making appropriate infrastructural changes, such as clustering or simply buying faster machines (Wong *et al.* (2007)).

With the mass adoption of cloud computing, this model is superseded by PaaS as a more affordable solution. Deploying to the cloud requires little or no capital investment; however, operational expenditure is now directly tied to the efficiency of deployed applications. There is now a clear economic driver for efficient web applications, services and deployment platforms. Addressing the latter allows a platform for the applications and services of the future to run on. Google required fast JavaScript so that its services like Gmail [5] and Google Calendar [6] would work efficiently and render quickly for end-users.

[3] https://www.djangoproject.com/
[4] http://cakephp.org/
[5] http://gmail.google.com
[6] https://www.google.com/calendar/

To do this, Google developed the V8 JavaScript engine (Google (2010)), which compiles JavaScript into highly optimised machine code on the fly. Google open-sourced the V8 engine and it has been adapted by the open source community for cloud computing. The cloud computing version of V8 is known as Node.js (Dahl (2009)), a high performance JavaScript environment for servers (Lerner (2011)). In the PaaS model, an entire computing platform including operating system, execution environments such as Node.js and Vert.X[7] and storage mechanisms are provided in a manner capable of scaling to match the demand. The customisation and modularity of offering such as Heroku[8] and Amazon Web Services (AWS)[9] offer support stacks for applications with complementing language and application choices. With this new costing model, performance becomes important as the operating expense dominates, driving the need for highly efficient, lightweight solutions. Architectural choices can thus dictate whether a domain is viable or non-viable, from a cost point of view.

### 2.4.7 Summary

The evolution of languages and best practice approaches to problem solving are driven by consumer requirements and consumer usage patterns (RQ-GFM1). Suites of software tools and complementing techniques are available to tackle a number of problem domains (RQ-S3). The industry momentum is currently coming from cloud and mobile computing, with this support structure only now beginning to facilitate emerging social networks through innovative and highly scalable services. The conceptual movement towards more event-based systems and the cultural movement towards accepting JavaScript as a serious language has helped achieve a possible new paradigm shift, along the lines of the major acceptance of OO. This shift can only be defined and identified with retrospective analysis, however, what is clear is the pace of software evolution to meet the current demands is exhibiting all the characteristics of a paradigm shift.

## 2.5 Summary

This section presented a historical overview and state of the art of the current challenges facing the design and management of a group formation and interaction management platform. The challenges identified have failed to address the Research Questions

[7] Vert.X is an alternative asynchronous application development framework to Node.js: http://vertx.io/

[8] http://www.heroku.com/

[9] http://aws.amazon.com/

outlined in Section 1.3. Summarising the questions in relation to the literature, the following aspects are of interest to Group Formation and Interaction Management:

- Group are weakly modeled
- User behaviour is changing
- Micro second execution is challenging
- Strong management not deployed within group environments
- Services using group communication as a deployment medium

This thesis will demonstrate that when investigated and addressed, these observations will satisfy the Research Questions on Group Formation and Management (RQ-GFM). Addressing these observations required a complementary analysis on the current state of the art in Software Engineering. Advances have occurred allowing for the design and implementation of highly scalable solutions, with Table 2.5 summarising the modern approaches that developers are adopting in comparison to the traditional technological stacks that would have been used.

| Development Choice | Traditional | Modern |
|---|---|---|
| Language | Paradigm based | DSL |
| Representation Format | XML | JSON |
| Deployment Platform | Dedicated, Bespoke | Cloud |

Table 2.1: Traditional and Modern comparison of popular development choices

This knowledge will help to address the Research Questions on Scalability (RQ-S) and support the contributions provided to the domain of successfully managing the formation and interaction of groups within emerging social networks.

# Chapter 3

# Group Membership: Formation and Management

This chapter presents challenges surrounding Group Membership, particularly focusing on Formation and Management across domains of interest which are representative of emerging social networks. The first section of this chapter describes XMPP and analyses how the XMPP Roster can be used to form and manage groups. The current group models are critiqued and principles required for stronger group management are presented and analysed. The second section of this chapter deals with emerging user driven styles of groups forming, from large scale groups with mass membership to multiple small groups. The challenges of forming and managing these groups are explored firstly in the context of XMPP. A second experiment focuses on a case study for an application of large scale group management within the context of humanitarian relief.

The topics outlined within this chapter attempt to address some of the properties of emerging social networks. Such properties can be inferred from the state of the art analysis in Chapter 2, and are described as follows:

- Dynamic memberships within groups
- Unified view of the social network
- Lightweight, responsive service provisioning and consumption
- Efficient applications from a client resource perspective

To investigate these issues, a group communication mechanism was required for experimentation. A number of viable implementations were considered and touched upon

briefly within Section 2.2.6, with SIP and XMPP two strong contenders. Both protocols are open, with multiple language bindings allowing implementation and experimentation. SIP and XMPP attempt to solve two different problems and are designed for two different architectures. The SIP protocol was designed primarily for session negotiation and media transport, such as voice and video. The XMPP protocol was designed primarily for text based message exchange among a group of clients. SIP is inherently a P2P protocol whereas XMPP is inherently client-server. The XMPP protocol is chosen as it has a number of advantages which would facilitate the experimentations required within this thesis. Features such as shared state and roster storage are easily facilitated in a client-server architecture and while possible within a P2P protocol, it would incur an overhead. Additionally, as the protocol is designed around short messages among multiple participants, the style of groups, the diversity of the group model and the interactions that could occur matched the behavior and attitude towards groups that was observed within Chapter 2. XMPP is also more than a protocol, it is an example of an emerging social networking platform[1].

This thesis proposes examining the limits of existing group models both in terms of functionality and performance as presented on the XMPP Roster. By extending the group model new usages might emerge and a greater understanding of the principles required to effectively manage groups be abstracted from the investigated domain. Of particular interest is the handling of dynamic groups, which this thesis defines as *Groups with a membership base that is prone to fluctuation. The membership levels change rapidly and often without notice. The membership turnover is high with the groups generally being barrier free, with users joining and leaving at will.* This definition is based on related work in Hallberg *et al.* (2007) and is representative of the style of groups emerging with respect to the behavior and attitudes of users, explored in depth in Section 2. Additionally, the popularity of the protocol is a contributing factor to the emerging usages which has seen XMPP deployed in a manner not envisioned within the original design. The scalability limits of the protocol, as well as candidate architectural solutions are subsequently investigated. User interaction with social networking services, such as geolocation based check-in applications, is an emerging social network previously discussed in Section 2.2.7. A candidate case study is presented which might benefit from a managed group formation platform. The components designed, influenced by the interactions investigated within XMPP, are validated in a

[1]For example, the BuddyCloud Social Network: https://buddycloud.org. Buddycloud is a Social Network designed on channels of communication (groups) and built on principles outlined in XMPPs core protocol

high scale scenario.

## 3.1 Roster Group Formation

### 3.1.1 XMPP

The eXtensible Messaging and Presence Protocol (XMPP) is an open source, XML based protocol tailored specifically to provide extensible instant messaging and presence information (Saint-Andre (2004b)). XMPP assumes a client-server architecture, with multiple clients able to connect to an XMPP server over the TCP/IP protocol. A client is an entity that establishes a virtual XML stream with a server, over TCP/IP sockets, by authenticating using the user credentials and binding itself to the connecting resource. The stream, as denoted by the *</stream>* tag, acts as a container for XML stanzas which provision for both upstream and downstream communication. An XML stanza is a discrete semantic unit of structured information that is sent from one entity to another over an XML stream. The three stanzas defined are:

**</message>** A stanza to facilitate the transmission of a message from a sender to a recipient.

**</presence>** Used to broadcast the availability, and thus willingness to be contacted, of a user to anyone subscribed.

**</iq>** The Info/Query (IQ) tag is a request/response mechanism

The streams allow for the delivery of stanzas from client to client via the server. The servers responsibilities include storing and managing XML data used by the clients and managing the delivery of XML streams to local clients. The routing and delivery of streams to foreign clients is possible through local service policies allowing server to server federation. The protocol clearly outlines how communication should occur over an XML stream. Adhering to the key tags and XML semantics outlined, the entire stream can be viewed as one valid XML document. This highly structured means of communication allows for core extensions to be integrated without breaking the design rules and functionality of the protocol. Extensions take the form of XML schemas, describing the structure, constraints and content of the document. This shared vocabulary allows servers and clients to understand the requests received and formulate an appropriate response

### 3.1.2 Roster Explored

The XMPP Communication mechanism outlines the process in which groups of contacts are managed. A contact list, or roster, is used to manage a set of users, termed buddies, and to optionally group them together. The roster is stored on the server side and upon a user authenticating successfully, a request is sent to the server, by the client, to retrieve the users roster. The roster is specified formally in the roster schema and an edited version, showing entities relevant to this discussion, can be seen below in Listing 3.1. The full schema is available from XMPPRosterSchema (2012).

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="jabber:iq:
    roster" targetNamespace="jabber:iq:roster" elementFormDefault="
    qualified">
...
<xs:element ref="item" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="ver" type="xs:string" use="optional"/>
...
<xs:element name="item">
<xs:complexType>
<xs:sequence>
<xs:element ref="group" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="approved" type="xs:boolean" use="optional"/>
<xs:attribute name="ask" use="optional">
...
<xs:enumeration value="subscribe"/>
...
<xs:attribute name="jid" type="xs:string" use="required"/>
<xs:attribute name="name" type="xs:string" use="optional"/>
<xs:attribute name="subscription" use="optional" default="none">
...
<xs:enumeration value="both"/>
<xs:enumeration value="from"/>
<xs:enumeration value="none"/>
<xs:enumeration value="remove"/>
<xs:enumeration value="to"/>
...
<xs:element name="group" type="xs:string"/>
</xs:schema>
```

Listing 3.1: Edited View of the XML Roster Schema

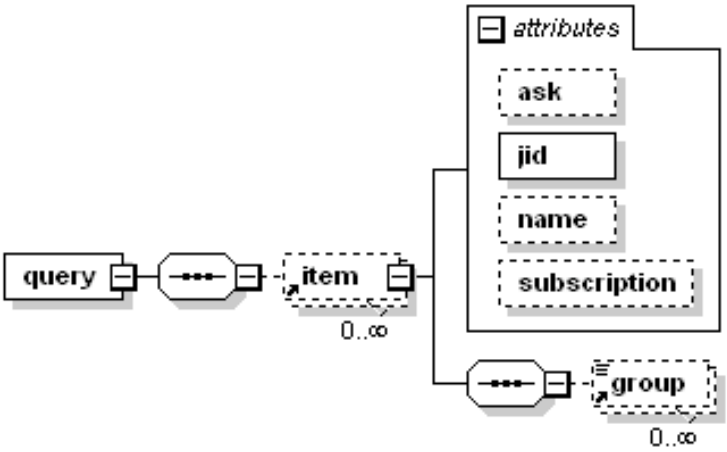



Figure 3.1: XMPP Roster Schema Design View

Figure 3.1, shows a visualisation of the XMPP roster schema[1]. The *item* element is a representation of a roster entry. The attributes within the item store important information about this contact. The Jabber Identifier (JID), is a unique means to identify an individual. The syntax is based on the structure of an email address, with a username associated to a domain name, which represents their home server. An optional resource mechanism is associated with a JID, specified by a slash suffix, allowing multiple simultaneous logins by the user with the XMPP server able to route the messages appropriately. The subscription attribute of the request is used to establish the type of presence subscription that will exist between the two entities, the sender and the recipient, or user and contact respectively. The allowable values for this attribute are:

**None** The user does not want to subscribe to the contacts presence information and does not wish for the contact to have a subscription to the users updates

**To** The user wishes to have a subscription to the updates of the contact but does not offer a reverse subscription.

**From** The contact will have a subscription to the users presence updates but the user will not subscribe to the contacts presence updates

**Both** Both the user and the contact are subscribed to each others presence information

[1]This visualisation is obtained using the XML viewer on the Altova product which can infer a visualisation of an XML schema (http://www.altova.com/)

The optional attribute, *ask*, can have the value *subscribe*, which means that an acknowledgement from the recipients server must be received to verify the connection. When the ask attribute is present, the presence subscription is then set to none, by default, until the response is received. The name attribute is an optional nickname to associate with the roster entry. The schema allows for the formation or population of groups during a subscription request. The group attribute is used to store the text based name of the group associated with the roster entry. This group attribute is optional and if it is not included in a request the contact will still be stored and rendered in the user client. Figure 3.2 shows the XML representation of a user roster including the optional group element.

```
<User>
  <Username>lgriffin</Username>
  <Password>*********</Password>
  <Email></Email>
  <Name>Leigh</Name>
  <Roster>
    <Item jid="sam@labtest" askstatus="-1" recvstatus="-1" substatus="3" name="Sam">
      <Group/>
    </Item>
    <Item jid="john@labtest" askstatus="-1" recvstatus="-1" substatus="3" name="John">
      <Group/>
    </Item>
    <Item jid="brian@labtest" askstatus="-1" recvstatus="-1" substatus="3" name="Brian">
      <Group>Friends</Group>
    </Item>
    <Item jid="frank@labtest" askstatus="-1" recvstatus="-1" substatus="3" name="Frank">
      <Group/>
    </Item>
    <Item jid="gary@labtest" askstatus="-1" recvstatus="-1" substatus="3" name="Gary">
      <Group>Work</Group>
    </Item>
    <Item jid="barry@labtest" askstatus="-1" recvstatus="-1" substatus="3" name="Barry">
      <Group>Friends</Group>
    </Item>
  </Roster>
</User>
```

Figure 3.2: Sample Roster with optional Group elements

Users can add new contacts to their roster through a roster set message. The user can send a request to another user requesting a friendship link be established. Figure 3.3 shows the sequence of messages involved in establishing a presence relationship using the roster set mechanism.

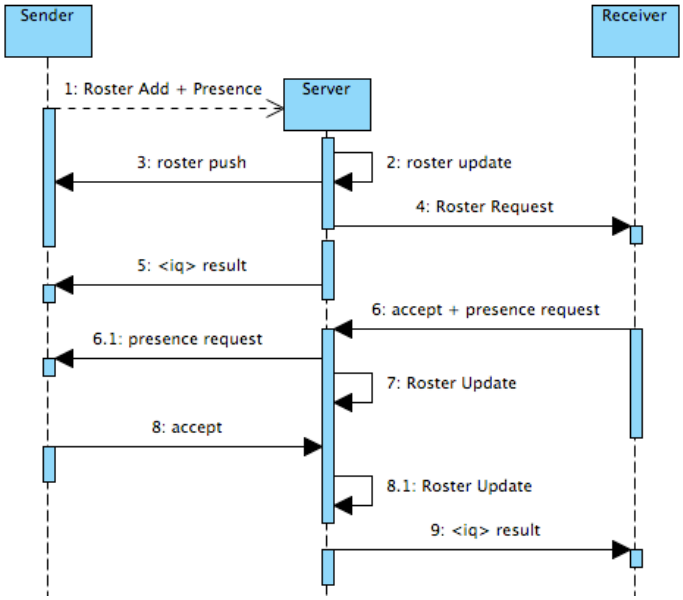



Figure 3.3: XMPP Roster Addition Sequence Diagram

### 3.1.3 Group Formation Styles

XMPP facilitates group formation through three key mechanism :

- User Generated Groups
- Publish-Subscribe Groups
- Item Exchange Groups

#### 3.1.3.1 User Generated Groups

A user generated group is created by a user from within their own client. The group is created through a simple interface on the client device and populated by the creator. This action prompts a roster set message being sent to the server, as the optional group attribute has been updated. This modification occurs so the client, on future logins, understands what groups to place roster items in. Listing 3.2 shows the structure of the IQ request sent to modify the group entry.

```
<iq from='user@example.org'
    id='rs123'
    type='set'>
   <query xmlns='jabber:iq:roster'>
     <item jid='user2@test.com' name='user2'>
```

```
        <group>My Group</group>
      </item>
    </query>
</iq>
```

Listing 3.2: Group set IQ message

Some observations about user generated groups will now be discussed:

- Membership is anonymous
  Users placed into a group are passive participants, completely unaware they have possessed membership of this group. The group is thus private and serves no purpose other then the logical placement of buddies within an end users client.
- Membership is not enforced or shared
  Once a user has authorised a friend request and presence subscription, they have bi-directional visibility on their IM clients. Any groups created by either user are not enforced across the buddy lists and no membership notification occurs.
- A 1:1 relationship exists between users and groups
  With user generated groups a buddy can only exist once and once only. Thus, a buddy can only have membership of one user generated group at a time on a users roster. Moving a buddy from one group to another causes them to lose their existing membership in order to be associated with the new group.

#### 3.1.3.2 Publish-Subscribe Groups

The second means of managing and creating groups within XMPP is a variation of the publish-subscribe (pub-sub) model as described in XEP-0060 (Millard *et al.* (2002)). This extension provides a framework for subscription nodes and event notification that is compatible with XMPP. A variety of applications dependent on event notifications, such as network management systems, can then benefit from the integration of XMPP. An adapted version of this model can be implemented server side, allowing an administrator the capability of creating groups and subscribing contacts to them. Figure 3.4 show the creation of a publish subscribe group from the administrative point of view within a popular XMPP server (Openfire (2012)).

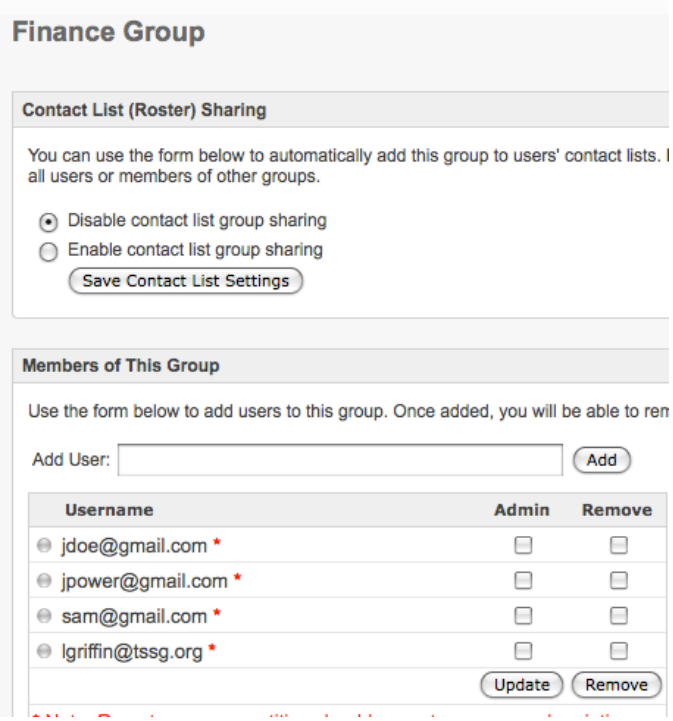



Figure 3.4: Administrative view of Pub-Sub Groups on an Openfire XMPP server

These pre-populated groups can then be published to end user rosters, effectively bypassing the process described in Figure 3.3. Entities, groups and presence subscriptions can be forced onto end users rosters. This is an effective way of subscribing users to default groups, with all editing attributes removed to ensure the group structure remains intact. Some observations of group formation in this manner include:

- Membership is enforced completely
  The end user has no say in their participation of a pub-sub group and do not have the choice of declining the invitation or leaving the group at will. The membership is completely enforced and the group cannot be modified by members who do not possess server administrator access.

- Overloaded Rosters
  The creation of a pub-sub group which has roster sharing enabled causes all members of that group to replicate the groups structure on their roster. This means the addition of the groups population onto the end users roster. The complications arising from such a scenario are the enforced subscriptions, potentially generating a large amount of additional presence updates. As it stands, Saint-

Andre (2009) identifies presence as accounting for 90% of XMPP traffic, with the majority of it being redundant broadcasts. Generating additional presence broadcasts is thus an expensive side effect of enforcing memberships.

- Administrative Interaction required
  To create a pub-sub group administrative access to the XMPP server is required. The creation and management of the groups needs to be performed by an administrator due to the modifications required to end users rosters

#### 3.1.3.3 Item Exchange Groups

This XMPP community recognised that shared groups should have a place within XMPP, with an extension proposal submitted accordingly (Saint-Andre (2004a)). The principles behind the extension eventually came into the XMPP protocol through the Roster Item Exchange (RIE) extension, as outlined in XEP-0144 (Saint-Andre (2005)). This extension proposed a means to recommend additions to another users roster which can in turn be used to form groups. The extension provides a mechanism for a user to share elements of their roster with another user, recommending additions, deletions and modification. The roster items sent can range from individual entries, whereby a single contact is shared, to sharing a roster group or indeed an entire roster. The extension allows a recommendation of which group the roster item should be placed. A sample XML fragment can be seen in Listing 3.3:

```
<message
from=user1@example.org
       to=afriend@sample.org
     <body>Add Jdoe from soccer!!</body>
     <x xmlns=jabber.org/rosterx>
      <item action=add
         jid=jdoe@sample.org
         name=John>
       <group>Friends</group>
       </item>
       </x>
</message>
```

Listing 3.3: Structure of an RIE recommendation

A number of observations about creating and sharing groups in this manner include:

- Draft Format
  The extension remains in draft format and its authors acknowledge that the

requirements set forth by the community for shared groups and synchornisation of rosters are not provisioned for completely within this extension but will be addressed in future work.

- Not Optimised
  The extension is currently not optimized for group management and group member distribution. For groups of size N, an RIE request must be sent to N-1 accounts, containing recommendations for N-1 changes to be made.

- Recommendations not enforced
  The changes sent are purely recommendations, which are free to be rejected by the recipient of the RIE request. If the recommendation is accepted, a standard friendship request is issued by the recipient to the recommended entity. This request in turn can be denied.

- No acknowledgment
  The requests sent are potentially blind, as no notification is returned to the originator of the request if the IQ mechanism is not utilised.

With two possible points of failure, the extension is not stable enough to guarantee a shared and unified roster view across group participants.

### 3.1.4 Criteria for Group Formation and Management

The observations presented in Section 3.1.3 represent an analysis of the present group models as implemented within XMPP. Some of the issues highlighted between the existing models are conflicting when compared directly, but represent potential extension points. The following are therefore considered as criteria for extending and better informing the group model, to ultimately empower a roster to better manage and facilitate groups and end users.

**End user created groups that can be shared and validated** The end user should have the functionality to create, share and populate private groups while ensuring that membership across the group is consistent and optional. To extend the current model this responsibility needs to be entrusted to the end user rather than an administrator.

**The ability to search for a group** Taking other forms of group communication, particularly in the social networking domain, the ability to search for a group of

interest is a key cornerstone for connecting people. This functionality is not provisioned for in the current group models.

**Lightweight rosters** The users roster should reflect subscription requests and friendships that they have made. The addition of groups to a users roster should not overburden the roster with additional entries for group members. To enhance the group models performance, the roster should be clean and lightweight, while maintaining the necessary group information.

**Group directable presence** Presence updates in XMPP are delivered to those with the appropriate subscription requests. The current model for presence updates allows users to direct their presence updates towards the entire subscribed roster rather then a specific group. Presence for large interconnected groups can be expensive. An extended group model should provision for presence to be directed at a group rather than an entire roster.

Such requirements are representative of user behavior and expectations emerging from current usage of social networks and are representative of the type of behavior that emerging social networks will encounter. The rest of this chapter examines how this criteria can be realised within XMPP by examining the scalability bottlenecks that might prevent this realisation within the technology. Section 3.2 attempts to investigate how end user created groups can be shared and validated to maintain a lightweight roster. Section 3.3 examines the case for group directable presence by looking at the effect that presence has on large scale rosters. Overburdened rosters are used to justify the requirement for lightweight rosters. The ability to search for a group is a standard feature that should be present within a group formation environment. Implementing this feature is trivial but is noted for completeness around the criteria and not examined within this thesis.

## 3.2 Group Roster Design

The weakly modeled group management principles within XMPP has prompted an investigation into how a group roster could be designed. This body of work is an attempt to investigate how the XMPP Roster can facilitate end user groups that can be shared and validated. An investigation into designing a group roster is also presented and critiqued.

### 3.2.1 Strengthening the XMPP Group Model

An XMPP server was configured, with Openfire (Openfire (2012)) the chosen server. The Smack client library (Smack (2012)) was used to communicate with the XMPP server and the Groovy programming language used to create and control the messages generated by the clients within the simulation. In order to simulate the activity from the clients, traffic breakdown characteristics outlined by Xiao *et al.* (2007) are used. A poisson process is used with additional implicit assumptions about the independence between clients activities. A working day, of eight hours, was simulated, with the weightings changed for start of day, lunch time and end of day, as per the observations of Xiao *et al.* (2007). The start and end of the working day sees more activity in terms of service control as users login/logout. Lunch time is a quiet period of inactivity with messaging ranked highest. The normal operation represents all other times with presence being dominant. With these weightings, the first simulations were carried out to give a normal operation view of an XMPP server across varying population sizes with groups enforced through the pub-sub mechanism. Presence, messaging, login events and roster maintenance were weighted accordingly within the simulation. The second set of simulations represented how dynamic shared groups could be represented within XMPP.

As per our dynamic group definition, groups should be barrier free. The pub-sub model examined in Section 3.1.3.2 is thus not viable for managing groups. The administrative overhead of requesting a group to be created is not realistic and would lead to the system being circumvented, with any benefits the system could offer being ignored and under utilised. Properties of the pub-sub model are still useful though, particularly the unified view of a group across all member accounts. Similarly the user generated group model does not meet the requirements, but the ease of end user group creation is desired. As such, the examination of dynamic shared groups within XMPP will look at the only current means of distributing a group, Roster Item Exchange, in conjunction with User Generated Groups. For this simulation, the pub-sub enabled groups were turned off, with RIE taking control of the management of the groups. This simulation considered messaging, roster updates (via RIE), presence and login events, with the simulation weighted accordingly. Any updates to a group would be replicated out via RIE to ensure the group view remained intact. The breakdown of events generated by the client over the course of all the simulations averaged out at the following percentages. For normal operations the generated traffic breakdown was; 74% presence messages; 21% IM chats; 4% for login/logout and 1% for roster maintenance. In comparison, the RIE simulations saw 44% messaging, 28% roster item exchange

messages, 25% presence and 3% login/logout generated.

The simulations were carried out on initial population sizes of 10, 20, 30, 40, 50, 100, 200, 300, 400 and 500 accounts. In each instance, the individual accounts had a roster size equal to the population size, with full bi-directional presence subscription for each simulation population. The roster for each user was divided into groups possessing 10 members each and the group view replicated across all accounts at simulation start. As the simulation population size increased the number of groups per roster subsequently increased, with 50 groups per roster in the 500 population run. An additional 10 accounts, with no relationship to the data set were made available for roster requests within both simulations. The metrics gathered included the Packets Per Minute (PPM) that the server had to process for each simulation run at varying community sizes. Figure 3.5 below shows this data.

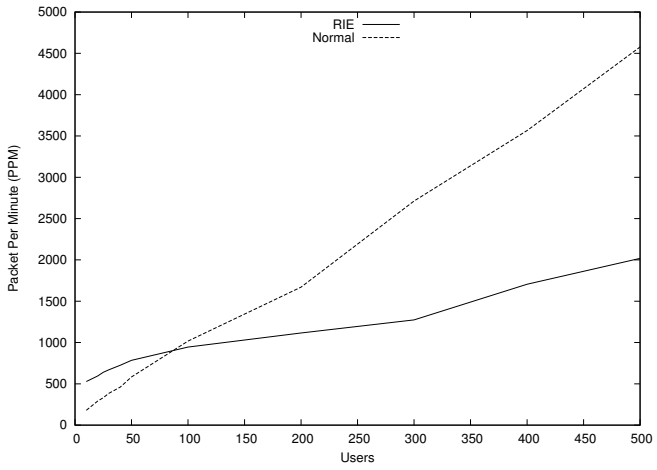



Figure 3.5: Packets Per Minute of RIE and Normal Operations Vs Number of Users

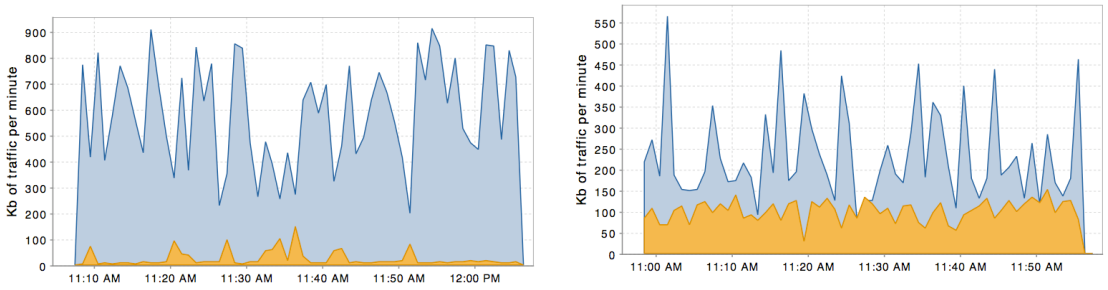
The load generated and memory consumed by the server were recorded for the larger runs. It must be noted that the results obtained for server load and memory consumption have a number of variables which can affect the measurement, including but not limited to, the hardware employed, the programming of the simulations and the nature of the Just In Time (JIT) compilation (Inagaki *et al.* (2003)). Additionally, due to the sensitivity of the monitoring plugin runs below 300 users were not recorded and have thus been excluded:

- At 300 users, a load of 15.64% and memory consumption of 11.01% was recorded for the normal simulation. This compares to 24.36% load and 25.96% memory consumption for the RIE simulation.

- At 400 users, the load generated by a normal simulation run placed a load of 20.61% on the server with memory consumption at 12.97%. This compares to a load of 24.20% and 29% memory consumption for the equivalent RIE run.
- At 500 users for a normal run, the load was recorded at 22.21% with memory consumption at 25%. The same run for RIE saw 27.40% load on the server and 28.23% memory consumed.

### 3.2.2 Server Load Analysis

An XMPP server has identifiable stress points with traffic capable of being distinctly broken down into client generated traffic and server generated traffic. The server has to manage normal client related activities, including messaging and presence traffic as well as the traffic profile generated by the resulting action, typically the propagation of messages or presence updates. Even at low levels of traffic the split is noticeable. Figure 3.6(a) below shows the breakdown of traffic within an XMPP server loaded with 500 inter connected users rapidly changing their presence. The orange traffic within this figure is a result of the client generating a presence change and pushing it to the server. The blue traffic is the resulting server side operation which updates the user rosters and pushes the resulting new view down to the inter connected clients. This traffic profile can be described as bursty, with sharp presence spikes incurring a lot of server side overhead with the problem more evident at high levels of concurrent usage. Figure 3.6(b) shows a typical usage scenario where the end users are not interconnected presence wise. The overhead the server has to deal with in this capacity is the processing of end user messages and the correct routing of the message. The split in client generated overhead and server side generated overhead is much closer in this scenario. Our problem domain envisions a mixture of both, accurately reflecting real world server usage where a mixture of messaging and intense presence traffic is evident.




(a) Client-Server for Presence

(b) Client-Server for Messaging

Figure 3.6: Client-Server Traffic Analysis

### 3.2.3 Observations

**Scalability of Presence:** The packets per minute generated by the normal traffic profile, which is weighted towards presence and messaging scales linearly as expected. The more accounts connected that have subscription relationships the higher the load, memory footprint and PPM that the server has to deal with. From a scalability point of view, presence distribution represents one of the biggest challenges that designers of services that utilise the presence mechanism for broadcasting will have to tackle. By oversubscribing the rosters through enforced group subscription this problem is magnified unnecessarily.

**RIE is costly:** Roster Item Exchange as a means for distributing groups is very expensive, particularly at low community levels. Up to a community level of 100 users, RIE is still comparable with normal presence and messaging operations in terms of Packets generated for the server to deal with. The load placed on the server and memory consumption for RIE simulations in direct comparison to the normal presence weighted simulations is an interesting statistic. Despite a PPM difference of up to 2500 in some runs, RIE has a larger footprint and resource drain on the server. In a live use case scenario, the figures and client-server throughput would be a lot higher for RIE then recorded, as more presence broadcasts would come through the server with real world usage. The experiments recorded centered on the viability of RIE as a means to distribute dynamic shared groups with messaging also a priority. Presence was a relatively small component, but one that offers a more consistent and predictable cost. The simulations did not consider varying group sizes, with the controlled setting of the group size initially starting at 10 users but allowed to change independently. This was

to prevent a scalability bottleneck by starting with artificially high groups, far removed from the scenario outlined. An initial group size of 50, for example, would see a lot more RIE requests required to retain the shared aspect, something which might have caused problems with timing and packet loss as the server had the potential to get overwhelmed with such large groups. It is worth noting the breakdown of traffic observed from client to server in both simulations. In the largely presence based simulation, the majority of the traffic flows from server to client, as presence updates are broadcast to the subscribed population. For roster item exchange the client generates over half of the traffic as roster item exchange messages are broadcast to other users. Within the scenario outlined, smart phones and similar ubiquitous devices would be required. A large client to server traffic stream would place significant strain on client resources and create an additional scalability problem where group size and number of groups per client device would have to be considered.

**RIE disadvantages:** A number of guarantees were required within the RIE based simulation to ensure that the groups would be distributed accordingly. Simultaneous RIE requests were not allowed in order to preserve the integrity of the group structure. Two RIE requests from different sources had the potential to create different interpretations of the group structure. Additionally, presence subscriptions were forcibly injected into the simulation. RIE, by design, does not include a presence subscription when adding a new user to a group. When adding a new user it could not be assumed that the intended account to be recommended to other users already had a presence relationship with the RIE recipients. As such, an additional subscription packet had to be sent. All rosters also had an auto accept enabled for presence and RIE subscriptions, something which the RIE specification strongly advises against due to legitimate security concerns. It would be possible for a Denial of Service (Mirkovic *et al.* (2005)) attack to occur by pushing through a large volume of RIE requests that conflict in a short amount of time. If this feature were turned off, it would be possible for individual recipients to simply reject the RIE request and therefore not have a shared group view. The usage of RIE deletion requests is not provisioned for in the simulation. The behavior of forcing the permanent deletion of a roster entry through a recommendation system was deemed a security vulnerability and one which could adversely effect the results of the simulation.

### 3.2.4 Managing Roster Groups within XMPP

The deployment of dynamic shared groups within XMPP, while possible, has too many assumptions associated with it and no formal management provisioned. A working implementation is possible with clever programming and a community willing to stick rigidly to the guidelines, but a long term, scalable management solution would allow XMPP evolve group based applications in a controlled manner. The group management structure is dated but serves faithfully the original purpose of XMPP. The management of groups is important enough to be abstracted away from XMPP, to provide more control for group management and formation. The experimental architecture proposed to investigate the viability of managing groups external to XMPP is shown in Figure 3.7.

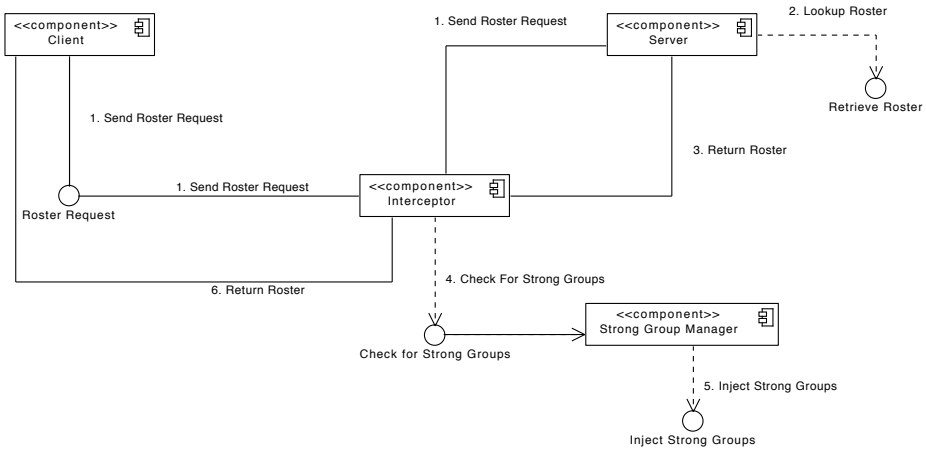



Figure 3.7: Component Architecture View

Using the notion of a JID, the authors extend this concept to a Group ID, or GID for short. A GID would be used to hold a reference to a Group which would reside on a group server and would structurally take the same format as a JID address: group_name@group_server_domain

The group server in this case is an unmodified XMPP server. A standard XMPP server would also be used to handle normal operations. A modified client would create one connection to the standard XMPP server and download the users roster. The client would make multiple connections to the GID server, with each connection representing a group that the clients owner is a member of. The interceptor is a modified connection manager, modeled on Openfire (2009), which traditionally would be used to alleviate scalability by operating as an anonymous proxy. The nature of the interceptor allows

the user to intercept and view packets and route them appropriately. The interceptor would be associated with the group server and would communicate with an internal data-structure to assist in the routing decisions Experimentally the viability of this setup was tested by using a novel approach. Two XMPP servers were established, with one designated the GID server. A set of accounts were created on the GID server, each one representing a group. Client devices held a connection to their own JID server and several connections to GID accounts. The multiple connections to the GID accounts were facilitated by the novel use of the resource mechanism. A sample GID account name and resource could be group1@groupserver/jid@normalserver

By making the resource equal to the JID of the owning account, modifications in the interceptor allowed the routing of messages to the appropriate account. The simulations carried out in Section 3.2.1 were repeated for population sizes of 10, 20, 30, 40, 50, 100 and 200, population sizes where RIE was close, if not more expensive than the equivalent presence overhead. The populations were connected to the JID server with their roster sizes varying. No roster sharing occurred on the JID server accounts. The GID server had 10 groups populated with members of the JID server, with each connected JID account logging into several GID accounts through the resource system described. The simulation was run using the same weighing as the RIE simulations. The graph in Figure 3.8 shows the results in terms of aggregated PPM. The results show that using the resource mechanism in this manner drastically reduces the number of PPM required to share groups out in a dynamic manner.

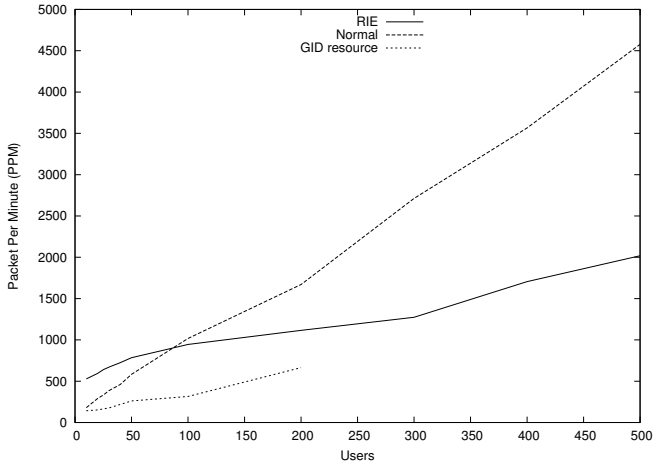



Figure 3.8: Packets Per Minute of RIE and Normal Operations and GID Resource based Groups Vs Number of Users

The approach taken was experimental but it shows that shared groups through multiple servers are possible and viable for small groups. Taking the resource approach is not a long term solution with group account credentials shared out among users being an obvious security concern. The impact on client and server side hardware will also be a barrier as multiple stream management causes a resource drain on the hardware deployed. Due to such limitations, the experiments could not scale safely beyond 200 accounts due to threading and memory management issues. The results are useful however, in that they do show a significant improvement at low population levels.

### 3.2.5 Summary

This experimental work highlighted the inefficiency of the XMPP protocol to share and distribute groups. RIE is a very costly mechanism to distribute groups with too many guarantees to ensure the groups are accurate. The attempts at engineering a Group Server, using a GID mechanism showed that XMPP could benefit from an external entity managing groups. Group Management through a Roster mechanism is possible within XMPP but ultimately in a limited capacity. The design is hindered by scalability issues within existing clients and by virtue of using the protocol outside of the scope of the inital protocols intention. Abstracting the Group Roster from the in built communication mechanisms group management principles is the only logical way a group can be maintained in a safe, scalable manner. A unified view of a dynamically changing social network, in an efficient manner is beyond the current state of the art within the protocol. This partly satisfies RQ-GFM1 while also establishing some results to discover additional scalability limits which RQ-S1 set out to examine. The next section will examine the viability of the existing roster to manage large scale group communication in an efficient and lightweight manner.

## 3.3 Scalability and Performance of Groups

This section deals with emerging usage patterns, centered around content delivery and the formation of large groups. These usage patterns are representative of how people are interacting with technologies in ways that were not predicted. The effect this has on the Roster and the case for minimising the presence footprint and for maintaing light rosters is initially investigated. A complementary investigation outside of XMPP is also carried out, investigating how a domain, with no limitations or boundaries with respect to a protocol, can perform group formation on a roster style mechanism.

### 3.3.1 Large Group Management

Scaling messaging services beyond the levels for which they were originally architected is the motivation for this analysis. In particular, the large scale delivery of individual messages, based on presence, to traditional Instant Messaging clients. Typically, IM systems assume that a users buddy list is scaled to human dimensions. So a roster might typically have 50-100 contacts (buddies). However, in some circumstances it might be interesting to propose a usage pattern whereby a given user appears as a contact (buddy) on thousands, or tens of thousands of rosters. This could be for emergency services, direct marketing, customised alerts or other forms of usage that leverage presence of messaging on a large scale. In functional terms, reliable, scalable short message broadcasting means being able to account for every message sent, and to be able to deliver messages quickly and efficiently, even as the number of users attached to the XMPP roster increases. It could be argued that this use-case for XMPP was not envisaged, i.e. the normal use-case, is a private individual with a few hundred or less contacts on his/her roster where reliability and scalability are not crucial. As with many other technologies, actual use-cases cannot always be predicted with business requirements dictated by user needs. Mass presence handling and message broadcasting built on top of XMPP is attractive for both publisher and consumer. Implementing this use-case seems possible, but is certainly not trivial. Efficiently handling the concurrency from a language perspective can help facilitate such use cases.

#### 3.3.1.1 Concurrency Programming Paradigms

Diverse approaches to programmatically coping with concurrency have long been a source of contention among software developers. The evolution of the various approaches to concurrency is well illustrated in the C like languages, particularly Java. Although Java was designed with thread based concurrency in mind (unlike C and C++), its concurrency support has evolved signicantly since its inception, with adjustments made to the core syntax, the libraries and the recommended approaches. The fundamental mechanism (synchronised keyword to serialise method access), has been supplemented with concurrent data structures, more expressive annotations, and an extensive rework of the concurrency model in Java 5 to incorporate a new executor framework (Long & Long (2003)). However, concurrent programming in Java is still regarded as complex and error prone, with non-determinism an ever present worry, even for systems long deployed in the eld.

The java concurrency model is founded on the shared state semantics of a single

multi-threaded process, whereby threads can share resources and memory, but with locks associated with specic data structures. Alternatives to this model have gained some ground. The actors model rules out any shared data structures (and their resource hungry locks), with concurrency achieved by message passing between autonomous threads. Each thread, termed an actor, has exclusive access to its own data structures. In functional languages derived from Java (Scala, Clojure), immutability itself is elevated to be the default programming model. This requires wholesale adoption of functional approaches (or object-function hybrids in the case of Scala), with the consequent profound change in programming style and heritage. With all of these approaches there is one common characteristic. Separate threads are created, with their own stacks and program counters. Although the opportunities for inter-thread synchronisation vary, such synchronisation must occur at some stage, with consequent overhead associated with task switching, memory usage and general processor load.

There is an alternative, which has its origins in an era that predates the general acceptance of multi threaded infrastructure. Evolved to meet the requirements for responsive I/O in single processor systems, it sometimes takes the term Non Blocking I/O, although this term has also been applied to threaded designs. Originally devised as a set of interrupts and associated daisy chained interrupt handlers, in the modern sense, Non-Blocking I/O implies an extensive use of callbacks in API design and usage. In this context, all opportunities for blocking are replaced by passing a callback parameter, to be invoked on completion of the deferred task or I/O request. A somewhat counter-intuitive programming style, it has been criticised for its verbosity and general awkwardness. In certain programming languages it is indeed verbose, Java in particular is encumbered with a high ceremony anonymous inner class syntax which makes callbacks quite difficult to orchestrate. Also, in Java and other languages of that generation, the callbacks are limited in scope and place severe restrictions around the context they can access. What they lack is a closure capability - essentially a form of delegate/callback/function handle - which also carries (encloses) a well defined context that can be safely accessed when it is activated. A closure thus allows a function and associated reference environment safely access non local variables even when they are invoked outside of its immediate lexical scope. Essentially, it enables the introduction of Lambda expressions within algorithms, enabling a degree of expressiveness and conciseness difficult to match with more conventional approaches. Closures have become a hot topic in programming language recently, and Java itself is slated to this capability in future versions. JVM derived languages such as Scala and Groovy have this capability, as does Closjure, via its Lisp heritage, and JavaScript. In fact the term closure

originates from these functional languages.

#### 3.3.1.2 Experimental Approach

For the purposes of this investigation an XMPP server was required. Openfire (Openfire (2012)), an open source Java based XMPP server was chosen for the tests. The extensible nature of the server, delivered in the form of plugins and components, along with the availability of its core API facilitated this investigation. The service scenarios of interest, such as direct marketing campaigns or emergency communication had the potential to involve tens of thousands of users. In comparison to the previous investigations within the XMPP domain, this scenario assumes the groups are already formed and the performance of such large groups is under investigation. Such a scenario will further the justification that group management is a discipline in its own right and needs enhancements and instrumentation to function. The messages sent would be time sensitive and only distributed to those in a position to receive the message, dictated by their presence status. Consulting a roster with thousands of users was not possible to implement within existing XMPP servers without the aid of a plugin. Rosters of that size are unwieldy and have the potential to cripple the performance of the server. Additionally, no guarantee is provided that the message sent was received by the server and processed for delivery. The approach taken saw the design of three plugins to be tested with the Openfire server. These results could then be compared to a fourth plugin, the default or legacy plugin within Openfire. The XMPP plugins extend the functionality of the Openfire XMPP Server, interacting with the server, the roster and the end-user in the form of a buddy. The plugins under investigation receive messages through a Java Message Service (JMS) (Richards *et al.* (2009)) queue for delivery to the roster recipients. The plugins under investigation are described below:

**Legacy Plugin** The Legacy Java plugin was not developed with a realisation of the concurrency issues that would come into play, especially under load conditions. The approach employed is to wait for an event on the XMPP or JMS interfaces and to process the event to completion on the event thread. The main class employed was not thread-safe due to access to a shared hash-map and the approach was monolithic rather than decomposed into tasks. This plugin is included for completeness and as a point of reference for one set of experiments. The plugin could be classified as non-blocking by virtue of not using threads, but no attempts are made to optimise the performance.

**Java Plugin** The Java plugin uses a fixed size thread pool with a tunable thread parameter encapsulated by a custom demultiplexer abstraction. Each type of event is modeled as a Task which performs a discrete unit of work, or calls on other Tasks to perform work. In each case the Task is submitted into an Executor for queueing and execution by the next available thread from the thread pool. This plugin uses the Java Executor framework. There are two such thread pools employed in this plugin, one for JMS events produced by the application server and the other for XMPP events. Each one may produce new tasks for the other. For example, a JMS message request from the application server will produce an XMPP message event to an XMPP end-user. Reliable access to shared data was identied as a problem for the system in this study owing to the use of a shared presence map. With thread safety a crucial requirement for the plugin, it was necessary to implement a reliable thread safe data structure. The current best-practice mechanism would be to use Javas concurrent collections (Bloch (2008)). The existing non thread-safe HashMap implementation was replaced by a Concurrent HashMap[1]. This implementation employs its own thread-safe concurrency mechanisms, is highly efficient and is already thoroughly tested.

**Scala Plugin** The third plugin uses the Scala language (version 2.8) and Scala Actors. Five actors are employed: a PresenceActor, MessageActor, ControlActor, JMSActor and an XMPPActor. The MessageActor routes XMPP chat messages to the outgoing JMS queues via the JMS Actor. The ControlActor processes requests from the application server, as well as XMPP Query packets, and routes outgoing messages to the XMPP interface. The JMSActor sends control, chat and presence messages over the JMS queues to the application server, and the XMPPActor sends XMPP packets out via the XMPP server. Each actor uses the *react()* method rather than *receive()* method which is well suited to event-based applications, and ne-grained tasks where the work scheduler can employ *work-stealing* techniques (Haller & Odersky (2009)). The PresenceActor provides guaranteed thread-safe presence lookups on a Scala Map. This presence facility was an important aspect of the design of the plug-in. Since the data contained in this map is shared between objects, the default choice for a Java developer would be a synchronised or concurrent HashMap. This choice was deliberately avoided in favour of an ordinary Scala HashMap free of any locking and simultaneous access. The consequence of this choice is that the lookup becomes asynchronous. The only practical way for the calling actor to know which presence result belongs to which

[1]Within the java.util.concurrent package is a Concurrent HashMap Class. The API is available on http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/ConcurrentHashMap.html

message, and without maintaining state information in the calling Actor, is to pass the message along with the lookup and to have the PresenceActor return it along with the result. The possible disadvantage here is the additional data transferred between the entities but since Instant Messages are generally short the tradeoff seems acceptable. The use of Case Classes and pattern matching keeps the code short and easy to read.

**Node.js Component** With the Node.js framework (Dahl (2009), JavaScript is no longer just a language supporting user interaction within browsers (client-side). Based on the Google initiated, open source V8 JavaScript engine, JavaScript, or languages that can compile down into JavaScript, can be be compiled into highly optimised server-side machine code on the fly. The Non-Blocking nature of JavaScript is present within Node.js with all requests gradually executed in sequence through the usage of callbacks. Node.js allows an elegant solution to be engineered for traditional scalability problems. A different approach was taken to create a non blocking plugin. The open source community developed xmpp.js (Wild (2010)), a useful Node.js library enabling access to an XMPP server as a component. An XMPP component (Saint-Andre (2012)) behaves in the same manner as a plugin, implementing new features but with the added benefit of not being tied to a specific server implementation, thus making it portable and reusable. The component binds to the XMPP server domain and becomes a part of the server in the same manner as a plugin. It is addressable and has its own JID in the form of a domain name making it accessible e.g. *component1.myserver.com*. All incoming stanzas addressed to that domain or to entities on that domain e.g. *buddy@component1.myserver.com* will be routed to the component base code where they can be subsequently analysed and delivered. Outgoing stanzas can be sent on behalf of any user on the domain giving the component full control of message delivery. Using the power of this library a simple component which would deliver the same functionality as the Java and Scala plugin discussed above was created. This Non-Blocking design resulted in the component having two functions, an *onPresence()* and an *onMessage()* function which would be used for callbacks to handle presence and message events respectively.

The role of each plugin was to accept presence messages from contacts on the servers roster, record the presence state and deliver chat messages to designated recipients who were online and available to receive them. Messages to users with a presence indication that they were not in a position to chat would not be sent. Figure 3.9 shows the internal structure of the XMPP plugins.

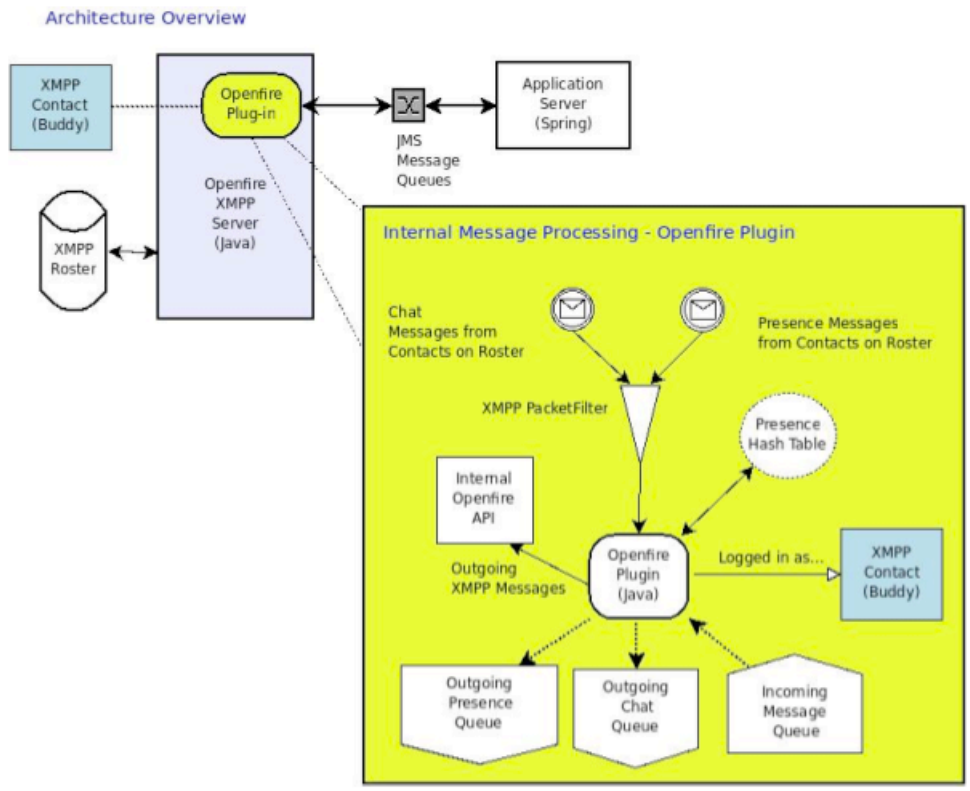



Figure 3.9: Overview of XMPP Plugin Internals

Figure 3.10 shows the architecture used for the scalability and reliability experiments. It shows the Openre XMPP plug-ins (shaded in yellow) and the role they play within the architecture. The plugins interacts with the XMPP server, the XMPP Roster, the end-user in the form of a buddy, and with a JMS Broker. The application server represents an external service that requires mass message delivery to online and available contacts. This service maintains groups of JIDs with the end user capable of sending a message to a specific group or groups. For the purposes of these experiments the application servers message load and recipient list is generated by the simulator and fed to the JMS message queue. JMS Messages can also be created by the plugin if required, such as when presence changes occur. These messages are checked to measure accuracy i.e. correctness of the plugins behavior.

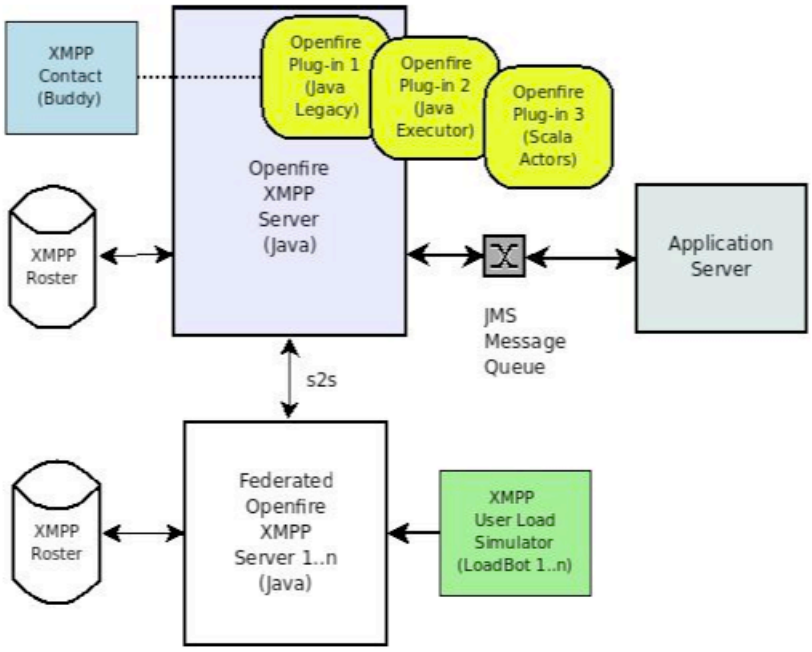



Figure 3.10: Experimental Setup

A customised Botz library (Botz (2007)) was developed to enable users to rapidly create user accounts and authenticate with the system. The experiments performed saw a load generator assume the role of the application server. The load generators role was to login 10,000 users, and produce 10,000 messages to be distributed to the user base, a single message per user. These messages would be fed to the JMS and subsequently handled by the plugins. A second set of tests was also devised to investigate how the plugins throughput would be affected by a heavy presence load. A second load generator was set up using the Botz library with 5000 users set to login and logout rapidly. This action produced *Available* and *Unavailable* presence statuses upon connecting and disconnecting with the server, respectively. This kind of rapid flooding of presence messages is designed to replicate a busy XMPP server and provide a more realistic performance evaluator of each plugin as they attempted to deal with the messages sent from the original generator. For each series of tests, the plugins were attached to the same server independent of each other and every effort was made to ensure accuracy and independence in the results gathered. The machine used for all tests was a 2.13GHz Intel Xeon powered 8.04 Ubuntu Server with 2Gb of RAM. Openfire version 3.6.4 and JVM version 1.6.020 were also used. Tests were run 20 times and the results gathered

for analysis are presented below

#### 3.3.1.3 Message Throughput

Table 3.1 shows the single load generator results. All figures below are in terms of messages per second that the plugin dealt with.

| Plugin | Min | Mean | Max |
|---|---|---|---|
| Java Exec | 109 | 270 | 322 |
| Scala | 214 | 249 | 267 |
| Node.js | 1360 | 1485 | 1608 |

Table 3.1: Single Load Generator results in messages per second delivered

The Java Plugin showed a trend of decreased throughput most noticeable as the thread pool size increased. The Scala plugins performance was comparable with the Java plugin but only at the higher end thread-pool settings. For the smaller thread pool settings the Java plugin was on average 15% faster. The Node.js components Non Blocking approach saw considerably higher throughput. The peak results are of an order of magnitude higher than the Java equivalent. Table 3.2 shows the results of the dual load generator with the Legacy plugin included as a baseline comparison

| Plugin | Min | Mean | Max |
|---|---|---|---|
| Legacy | 8 | 21 | 69 |
| Java Exec | 14 | 76 | 151 |
| Scala | 29 | 82 | 108 |
| Node.js | 518 | 621 | 745 |

Table 3.2: Dual Load Generator results in messages per second delivered

The Legacy plugin was used in this series of tests to provide a base figure for how a standard Openfire server would perform while trying to deliver messages in an environment with a lot of background presence noise generated by the second load generator. As to be expected the plugin performed poorly, dropping to as low as 8 messages per second. The Java Exec plugin had the highest peak throughput of the Blocking I/O based plugins but performance was somewhat erratic. The more controlled nature of the Scala plugin led to more predictable results with a marginal improvement on average throughput. Figure 3.11 below shows the three Blocking I/O plugins performance and the variability observed within test runs.

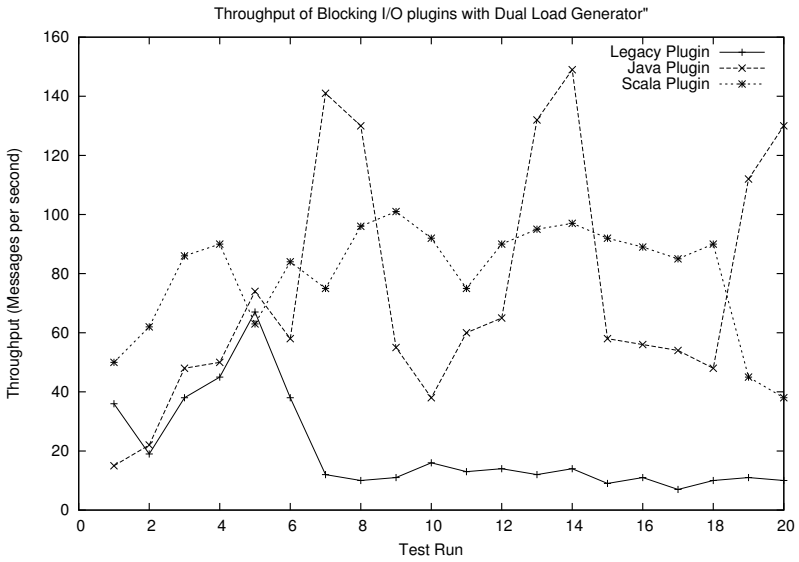



Figure 3.11: Dual throughput performance of Blocking I/O plugins

The Non Blocking I/O component did not suffer the same percentage drop in average throughput when it was faced with the noise of the rapid user logins and presence updates. Figure 3.12 shows the results gathered over the 20 runs and compared with the Blocking I/O based plugins

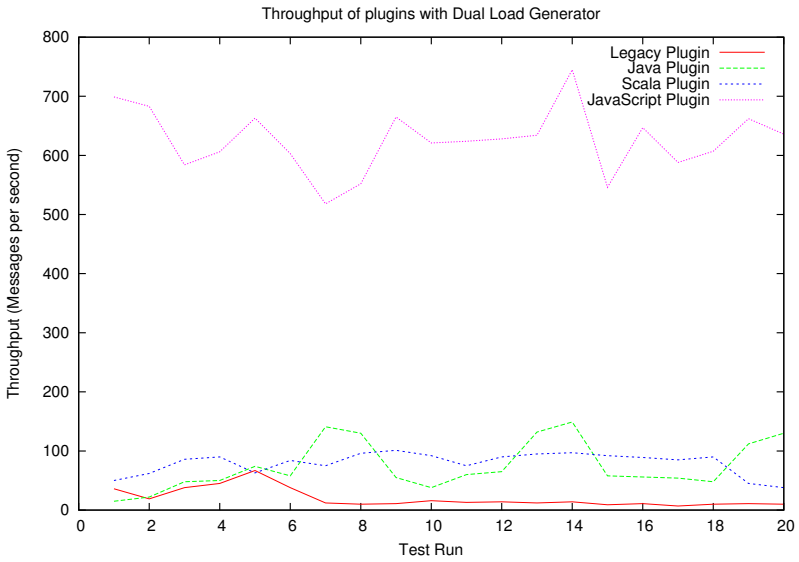



Figure 3.12: Dual throughput performance of plugins

#### 3.3.1.4 Memory Footprint

Memory usage was relatively light for all four plugins with some noticeable differences depending on the load scenarios. For the single load generator tests, the Scala profile was less then 50% of the Java profile. This situation was reversed when the second load generator became active. Scalas memory footprint increased from an average of 20Mb to over 180Mb. This was to be expected in an environment where shared memory is kept to an absolute minimum. The memory overhead was eventually enough to cause throughput degradation and eventually heap space errors on the Openfire JVM. The Legacy, Java and Node plugins had roughly the same memory footprint of around 70Mb across all tests.

#### 3.3.1.5 Observations

The CPU utilisation for the tests was also recorded. The Blocking I/O based plugins rarely troubled the CPU and did not consume many CPU cycles. The Node.js component however consumed 100% of available CPU resources when run on the single load generator tests. The multiple callbacks to handle events and deliver messages required a lot of CPU usage but delivered a far superior throughput for this trade off. On the dual load generator tests the throughput of the node.js plugin was directly related to the available CPU. The average CPU usage for the Node.js process was 68% with the

min and max results outlined earlier having a corresponding CPU usage of 54% and 79% respectively. The chance to take more CPU cycles was denied by the prioritisation of the roster updates by the XMPP server. This costly, but necessary action limited the potential of the Node.js component. Running the Node.js process on a separate machine to the XMPP server would increase the performance but was not within the scope of this work due to the nature of the other plugins developed.

#### 3.3.1.6 Message Accuracy

Table 3.3 shows the message and presence delivery accuracy of the plugins within the dual load generator tests.

| Packet Type | Legacy | Java | Scala | Node.js |
|---|---|---|---|---|
| Messaging | 70% | 90% | 100% | 100% |
| Presence | 100% | 100% | 100% | 100% |

Table 3.3: Dual Load Generator Message Accuracy

The legacy plugin became overwhelmed quiet quickly and the resulting loss of 30% represents a serious QoS problem for using a default XMPP server within a high load scenario. The Java plugin was a marked improvement in terms of accuracy but at times of high contention the concurrency issues were reflected by the number of messages lost. The Scala and Node.js plugins resulted in 100% message delivery. It is interesting to note the prioritisation of presence, which is directly related to roster management. The 100% presence delivery is required to guarantee the accuracy and integrity of the roster.

#### 3.3.1.7 Summary

This experimentation of emerging usage showed viable scenarios where existing infrastructure struggles to complete the task at hand (RQ-GFM1, RQ-S1). Delivering mass messages to groups in a short time span needs to be performed in a timely manner with complete accuracy. This experimentation puts forward a case for Non-Blocking I/O as core components in any deployed system that has to manage group communication services in a near real time manner (RQ-GFM2). The criteria presented in Section 3.1.4 shows that lightweight rosters are desirable as large scale rosters, particularly those of interconnected users, can create a scalability bottleneck on the server side. Presence in this instance is also very expensive, causing a sizeable performance drop that programmatically can be handled but should be avoided. The case for being able to limit and direct presence at groups rather then the group membership should yield performance

benefits, even in cases where Non-Blocking I/O is used, as per the results in the single load generator tests.

### 3.3.2 Mass Group Management

The investigations carried out within the XMPP protocol have proved useful for the understanding of how to form, structure and record a group within a roster. However, the protocol in many respects is limited by its own roster specification. In particular, the presence aspect of the protocol has limited the potential to investigate mass group management. This is due to the priority of presence messages to preserve the integrity of the roster and the scalability issues surrounding presence in highly connected rosters. The previous section highlighted the impact that presence can have on the performance of a relatively straightforward user requirement. Moving away from XMPP, lessons can be learned which can facilitate the structure and design of a group formation and group roster model. A social networking system can be designed from the bottom up with scalability in mind, thereby facilitating a greater understanding of where bottlenecks might emerge. A candidate for this is the notion of a check-in, a term described earlier in Section 2.2.8 which is capable of having groups form around it and thus represents a domain of interest. The usage profile of check-in based services is significantly different from XMPP based groups. The current style of a check-in group typically follows a *gamification* trend (Deterding *et al.* (2011)) with respect to locations. However, the check-in concept allows for richer, more serious applications to emerge. One such domain of interest is Disaster Management, a domain where social networking applications can play an important role for accessing and consuming information, this was highlighted in Section 2.2.7. This section presents a case study within the humanitarian relief domain, where group formation and management principles could be applied in the future.

First Responders, the medical personnel that make Triage decisions to prioritise care when vastly outnumbered can structure and co-ordinate their rescue efforts by efficiently parsing this information. However, the current means of using social networks is in an unofficial capacity, with potential drawbacks including the granularity of information available, misinformation, timeliness of information and which information to prioritise. By using the check-in route, the possibility of empowering the end user to make choices during their check-in process can alleviate some of these issues. Information can be more structured, precise and possibly leverage contextual information contained within devices or the environment. Information can be abstracted and parsed from the checkins which might better educate potential first responders before they reach the scene. This

scenario takes checkins encoded within JSON and parses the metadata to dynamically form groups based on the included metadata. This experimentation examines how mass group management can facilitate first responders in dealing with an emergency scenario and prioritising care through a triage algorithm. The experimentation is designed to validate technological choices with a naive view of rescue management in an idealised world.

A user centric view of the domain under investigation is visible in Figure 3.13.

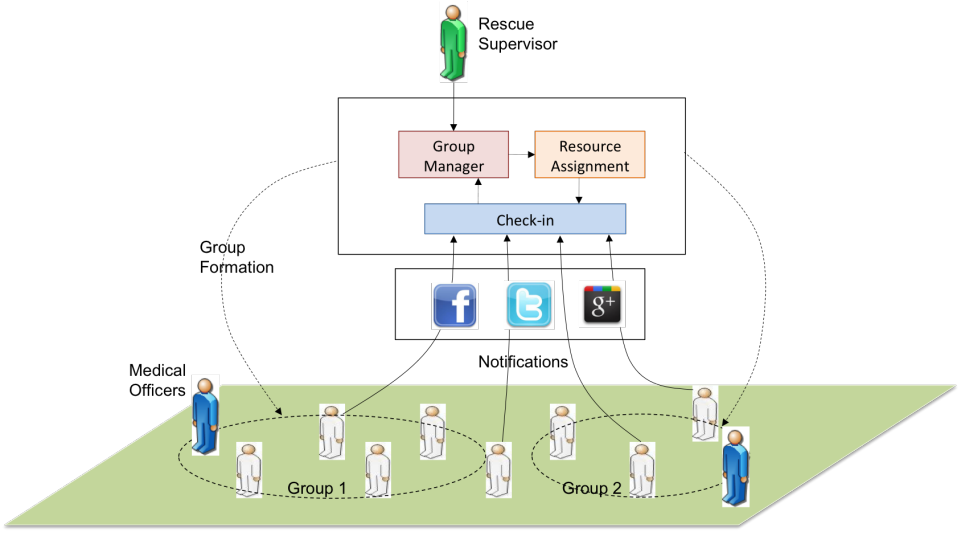



Figure 3.13: User View of Checkin Solution

The Triage system in place within the system takes user inputted information and places it within an algorithm. The users within their check-in inform the system of their ailment which has a weighting associated with it. The application also uses a scale of 1-7 to describe how serious their injury is, with 1 being minor and 7 being severe. The values 1 to 7 were chosen based on a comparative injury scale which can classify injuries into minor, moderate and severe (Stewart *et al.* (2004)). Minor implies that their injury does not interfere with most activities, moderate interferes with a lot of activities and severe is unable to engage in activities. As pain and illness tolerance vary, the subjective nature of the descriptive words tend for patients to gravitate around the middle of the scale, distorting the assessment somewhat. With a known shortage of medical personnel, people would tend to gravitate towards the far end of the moderate scale or into the severe in order to prioritise their health care. By using a combination of weighting with respect to the injury type and removing the traditional base 10 weighted scale, the end user has to make a conscious decision on where they think their injury

fits in. Combining these inputs within an algorithm (see Algorithm 3.1 for a sample) a 3 step Triage system based on priority can be achieved.

**Algorithm 3.1:** Triage Algorithm for Triage 1 classification

```
if checkin.userTriage ≥ 5 then
    We have a high user entered value, now check their injury
    if checkin.injuryType ∈ seriousInjurySet then
        Their injury is on our defined list, genuine Triage 1 case.
        if Triage1Group ∈ currentActiveGroups then
            Triage1Group = Triage1Group ∪ {checkin}
            return Triage1Group
```

By applying such an algorithm to the received check-ins, the group formation system is better informed as to what category to associate each check-in and consequently what groups to form. Triage 1 represents patients that are most in need of immediate care and need to be engaged by medical personnel as fast as possible. Triage 2 patients have injuries which are not currently life threatening but would need to be treated sooner rather then later. Triage 3 patients are designated walking wounded and their care is not prioritised until more serious patients are dealt with first.

#### 3.3.2.1 Simulation Discussion

Figure 3.14 shows the architectural view of the simulation.

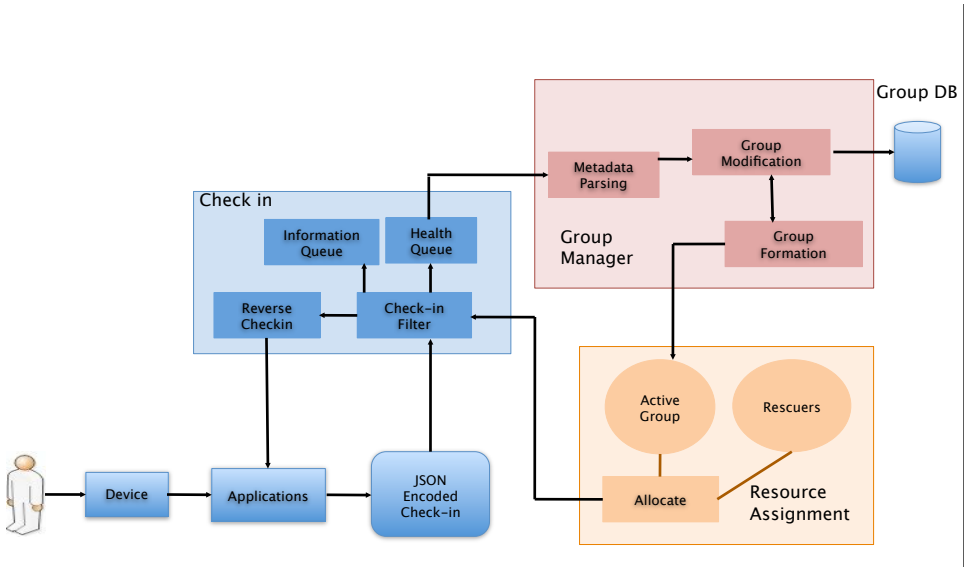



Figure 3.14: Overall Architecture View

The following components were specified and authored within Javascript using the Node.js platform. The throughput capabilities of a Non-Blocking I/O architecture would be required to meet the concurrent check-ins envisioned within this simulation.

**Group Manager** The Group Manager has the responsibility of establishing and modifying groups. Fed directly by the output of the check-in component and with access to the underlying data store, the group manager generates several groups from the metadata contained within the check-in. The Group Manager also passes on rescue groups to the rescue assignment component, dictated by the current strategy outlined by the rescue coordinator. In addition to creating the groups, the group manager associates a management object with each group. This object, a closure with defined context, acts as a means to update the group membership at the programmatic level, via callbacks. Allowing the groups a degree of self management reduces the overall strain on the system.

**Rescue Assignment Component** Designed around management documents dictating the strategy for rescuing people, the Rescue Assignment component requests a relevant group from the Group Manager. The component contacts the rescue teams and care givers passing them relevant information obtained from the check-ins including injury descriptions and location. A geolocation marker is pushed to the rescue teams

devices showing the location of people in need of care. A reverse check-in is sent to the patient device updating their map with the current geolocation of the rescue team and an estimated ETA based on historical evidence gathered throughout the rescue process

**Check-in Component** The check-in component is the main entry point for check-ins into the system. Designed with scalability in mind, this component makes the greatest use of the capability of spawning a replica child process and migrating process to another machine in times of extremely high load. The main function of this component is to strip the check-in information into information capable of being understood by the system as a whole. The check-ins are passed on to a database component for storage as well as directly to the group manager.

**Database: Redis** The database chosen complements the language and architectural decisions made in the upper layers. Redis, a NoSQL database with a Node.js interface was chosen. NoSQL databases are non-relational, distributed databases that deliver horizontal scalability. Redis is a data structure orientated storage system, providing a storage mechanism that complements the in built language data structures. This not only removes the mental disconnect from a development point of view, but also reduces friction between the core components. Redis additionally brings an in built publish subscribe mechanism, allowing a user register their interest in being notified of key events. Complementing the in built pub-sub system of Node.js and the Hook.io enabled distributed pub-sub, it ensures the system is reactive and constantly undergoing self monitoring.

**Additional Libraries: Hook.io** All components are enabled with the Hook.io library (Nodejitsu (2011)) allowing for a Spoke and Wheel Pattern for communication within the architecture. This allows components to be split across several processes and indeed machines, distributing the core components of the system for redundancy and security reasons. At the core of the library is a distributed publish-subscribe mechanism allowing cross process communication. Interestingly, the library can adapt any Javascript based legacy systems into a hook enabled component with a few short lines of code. The library has a useful, UNIX inspired feature, which allows the spawning of child processes, identical copies of the running program. This mechanism is used to great effect during times of high load.

The architecture was designed around components built and tested for resilience using the Node.js infrastructure and encoded within Javascript. A Group Manager

component was responsible for receiving the check-ins and processing them into appropriate groups. The groups created included Triage Groups, injury specific groups and location specific groups. The component was responsible for the lifecycle of the rescue groups, allocating groups as resources to be assigned to available rescuers. The actors within this simulation are patients trying to inform first responders of their current location and medical condition in order to facilitate a more efficient rescue response.

The term simulation environment is misleading as the components produced are in fact implementable and robust. The inclusion of the Hook.io library allows the system to scale out appropriately, replicating functionality in order to alleviate strain on the system. As such, the simulation environment would reflect the timing and speed of a real world working system with guarantees in place to protect what would be a mission critical system. The architecture was hosted on an Intel dual-core machine with 2Gb of RAM, running recently patched Ubuntu 10.10. Table 3.4 shows the simulation paramaters and their associated values:

| Paramaters | Value |
|---|---|
| Number of Users | 30,000 |
| check-in Rate | Poisson Process $\lambda = 100$ per second |
| Rescuer Number | 100 |
| Time for Attending Patient | Mean Time based on Injury Type, SD: 10 mins |
| FIFO Buffer Size | Infinity |
| Time to Travel for Rescuer | 20kmp/h |
| Patient Condition Types | 3 |
| User Location on Map | Random Placement |

Table 3.4: Simulation Inputs

Figure 3.15 shows the results of the randomised distribution of checkins and the initial point from which the simulations begin.

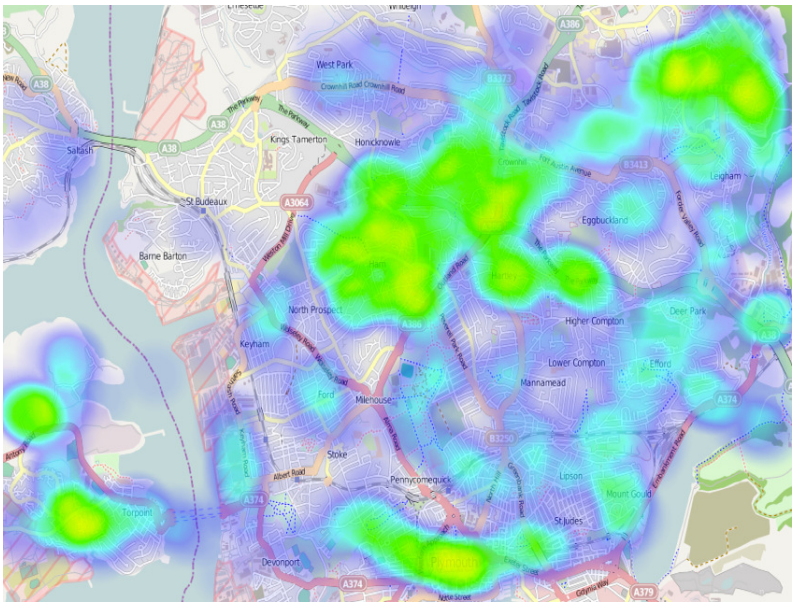


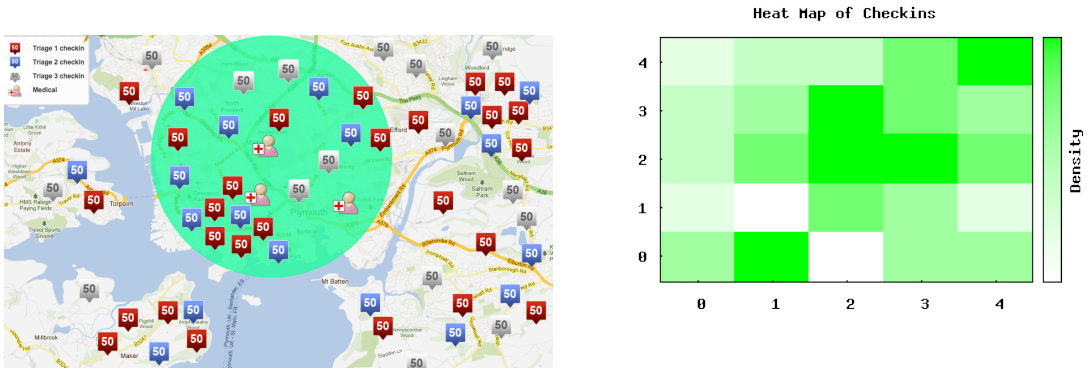

Figure 3.15: Initial Check-in Distribution Heatmap

For timing reason, simulation time used in the running of the tests was set at 1000ms = to 1min, with graphics reported in the latter. The graphics, for readability purposes, include every 250th point. The check-in set generated had a random chance, seeded by the system clock within the GNU Scientific Library (GNU (2012)), of having a triage classification weighted towards both ends of the scale, an equal chance of one of three injury types (each with associated times to fix) and a randomised geolocation within a city map grid. The rescuers were located from a central unit and after any patient has been dealt with, their geolocation would update to reflect their new location before moving to the next patient within their active group. Group formation plays a key role in this experimentation, groups form from the first checkin onwards, with groups having members added and removed dynamically. As such the system needs to coherently manage a group ensuring that no checkin is lost or that first responders are not allocated checkins that have already been dealt with. The results presented in the following subsections focus on the Triage 1 and Triage 2 cases, the users whom need identifying and treatment the most. It must be stated that the simulation is naive from the rescue management point of view. Environmental factors, which would slow progress, are not considered within the simulation. Attending to a patient is based on a mean time based on injury with a standard deviation built in. Additionally, rescuers ignore Triage 2 patients after the core grouping logic completes, with the simulation

entering a triage based rescue. Within a real life disaster management scenario, tough decisions would need to be made with regards Triage 1 patients as time goes on.

#### 3.3.2.2 Centralised Simulation

The first set of results adopted a centralised approach to grouping strategy. The densest location was chosen with a 20km radius established around it. All medical personnel travelled to the grid and systematically worked through the Triage 1 and Triage 2 cases within the area. Figure 3.16(a) illustrates the victims on the map of the medial officers. Figure 3.16(b) shows the same map based on density of check-ins.

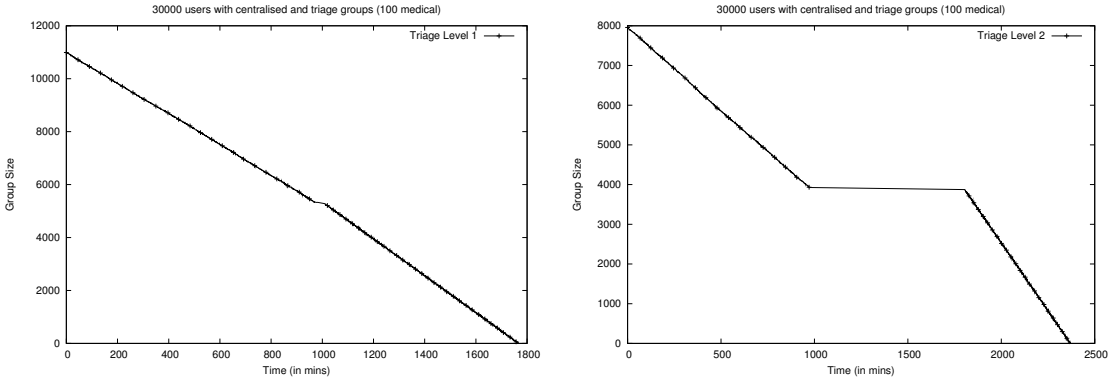



(a) Initial Map.

(b) Initial Heatmap

Figure 3.16: Map of initial Centralised classification and associated Heatmap

The approach then switches to a triage based rescue operation, where rescuers focus on remaining Triage 1 patients initially. Triage 2 patients are then dealt with after the Triage 1 patient group has exhausted and then finally Triage 3 patients are seen to. In terms of performance within the system Figure 3.17 show the group size with respect to time for the Centralised, Triage weighted approach.



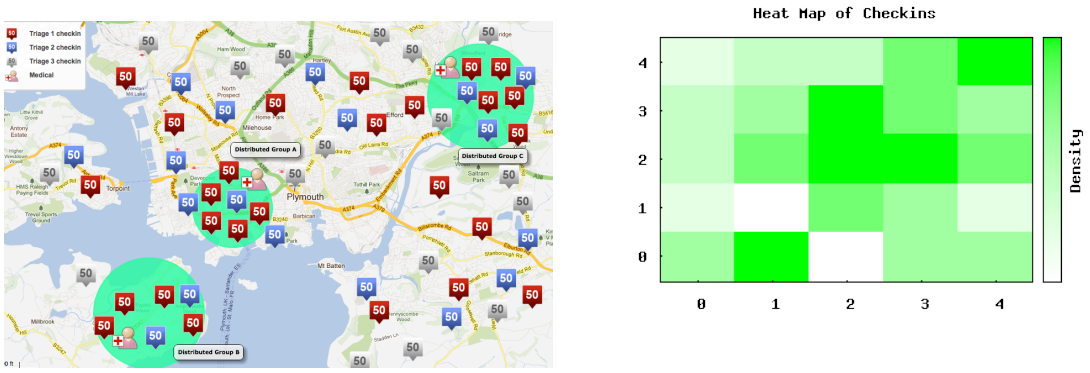

(a) Central Group for Triage 1.

(b) Central Group for Triage 2.

Figure 3.17: Centralised Approach to Rescuer Allocation

The performance of the Triage 1 group during the initial centralised grouping is stronger than the Triage 2 performance due to a larger number of Triage 1 patients involved in the catchment area. The plateau on the Triage 1 group is minimal compared to that experienced for the Triage 2 group. This is when the centralised group is cleared out and the rescue givers disperse to other parts of the map. In the case of the Triage 2 group the plateau lasts as long as the Triage 1 group has members. When the priority group is exhausted the rescuers switch to the Triage 2 group.

#### 3.3.2.3 Distributed Simulation

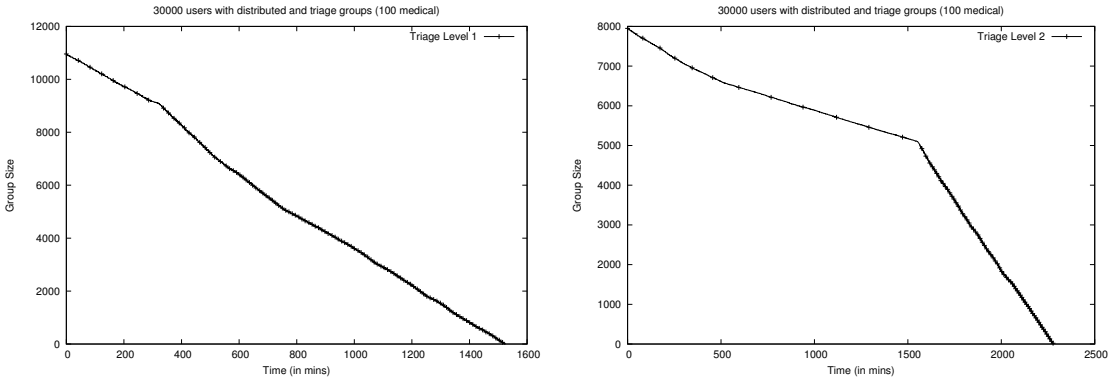



(a) Initial Map.

(b) Initial Heatmap

Figure 3.18: Map of initial Distributed classification and associated Heatmap

An alternative approach to the initial strategy was sought. Centralising the rescuers in order to deal with the densest area had a negative effect on the Triage 2 patients. A fairer approach to dividing up the check-ins received was sought with a more Distributed approach to grouping investigated. The simulation specifications, as outlined in Table 3.4 remained the same and the map was this time divided up by check-in density with a number of hot spots chosen as candidates for a distributed grouping. Three groups were formed as a result, with varying radius of 3km, 6km and 9km which can be seen in Figure 3.18. The rescuers are divided equally among the three groups and when the individual groups clear out the associated rescuers switch to the Triage model. As with the centralised approach the same metrics were recorded and they can be seen in Figure 3.19:




(a) Distributed Group for Triage 1.

(b) Distributed Group for Triage 2.

Figure 3.19: Distributed Approach to Rescuer Allocation

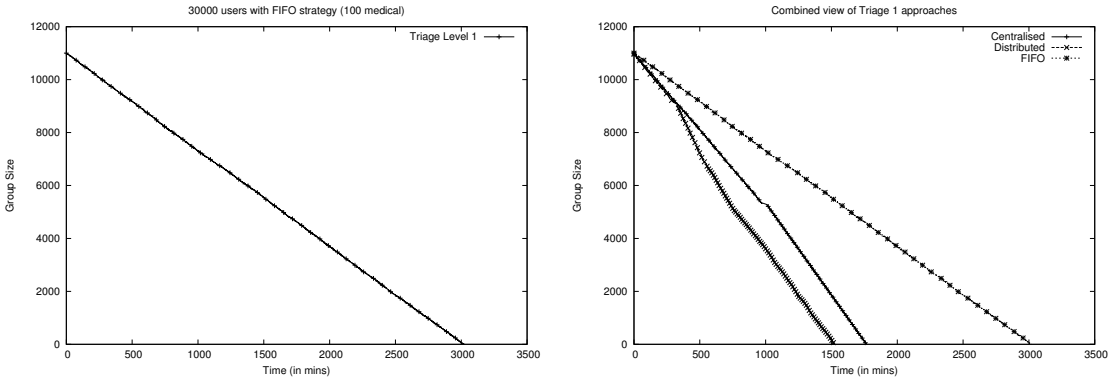
The distributed grouping in this manner strongly favors Triage 1 due to multiple groups pulling in more Triage 1 check-ins initially. Triage 2 however is the trade off, with a lower presence in the distributed groups. It however has no plateau as the rescuers within the larger group still work through it. Any of the rescuers that clear out the smaller groups switch to the Triage approach. This sees Triage 1 take some sharper drops as more medical personnel switch to this approach. The objective for this simulation was achieved, to attempt a fairer distribution of limited resources and the additional distributed grouping mechanism incurred no measurable overhead.

#### 3.3.2.4 FIFO Simulation

The third simulation run saw a first come, first served approach to the check-in allocation using a FIFO (First In, First Out) queue. No intelligent groups were formed and

rescuers worked through the backlog in an inefficient manner with travel distances not taken into account. Equal weighting was thus given to Triage 1, Triage 2 and Triage 3 patients.

The graph in Figure 3.20(a) shows the FIFO approach for Triage 1 patients and Figure 3.20(b) a comparison of Triage 1 patients using the three approaches.




(a) FIFO Approach for Triage 1.

(b) Combined Approaches for Triage 1

Figure 3.20: FIFO Approach and Combined View

The FIFO approach is almost a doubling the amount of time to reach patients that are in a critical condition, many of which have time sensitive injuries.

#### 3.3.2.5 Group performance

Table 3.5 shows the SET rate recorded from Redis for a payload of size 398 bytes, the size of a JSON encoded check-in, for 1000 concurrent connections. The infrastructure was tested and scaled up to 1,000,000 checkins. Crucially, the retrieval rate (GET) from the Redis database was equally performant, with no noticeable loss in terms of speed visible for retrieving groups. Table 3.6 shows these results.

| Requests Sent | Time Completed (second) | Messages (per second) |
|---|---|---|
| 1000 | 0.07 | 14,285 |
| 10,000 | 0.47 | 21,052 |
| 30,000 | 1.48 | 19,381 |
| 100,000 | 5.35 | 18,677 |
| 1,000,000 | 54 | 18,519 |

Table 3.5: Check-in database set rates

| Requests Sent | Time Completed (second) | Messages (per second) |
|---|---|---|
| 1000 | 0.07 | 13,698 |
| 10,000 | 0.53 | 18,975 |
| 30,000 | 1.63 | 18,164 |
| 100,000 | 5.59 | 17,885 |
| 1,000,000 | 53.99 | 18,522 |

Table 3.6: Check-in database get rates

#### 3.3.2.6 Summary

This simulation work showed how to efficiently manage large scale groups that had dynamic properties. 30,000 users were supported within the system with groups changing on the fly, giving an indication of the kind of performance profiles emerging usage will see (RQ-S2). This scenario showed that the technology choices of Node.js, Redis and a Non Blocking style can effectively handle mass group formation expanded beyond the C10K analysis previously carried out. This work was a validation of the underlying technology choices.

The scenario described showed that effectively grouping injured patients can possibly improve the response times for rescuers charged with assessing and providing first response care. However, the scenario could be further improved with greater intelligence dictating the management decisions and subsequent group formation. With respect to the criteria outlined in Section 3.1.4, this work examined a domain external to XMPP, showing how group management, stored on a roster style representation, could facilitate large scale groups. End user groups can be shared and distributed freely within such a system. The requirement for lightweight rosters is not an issue at the scales investigated within this experimentation. The group rosters contained the necessary group information in addition to metadata associated with each user check-in, allowing further group inferrals in the future.

## 3.4 Summary

In this section, group formation and management techniques were critiqued in domains of interest, particularly dealing with group formation and management with various levels of scalability. Recommendations were put forward that address some of the shortcomings within the respected domains and programmatically tackled. A case

was established for the limitations of group management within XMPP for handling emerging usage patterns. Analysis of new styles of group formation (RQ-GFM1) and the impact it has on the underlying management system were presented. By probing the scalability limits (RQ-S1), and establishing performance profiles (RQ-S2) a number of techniques and tools were presented which can be deployed in order to meet the requirements established, as per RQ-S3. The tools chosen are well suited to the task at hand as the Non-Blocking I/O style present within JavaScript and the Node.js platform is designed with handling high levels of concurrency, a task other platform and language combinations were shown to struggle with. Any future deployment will need to adhere to a concurrency model that has the characteristics of a Non-Blocking I/O approach in order to efficiently handle the volume of services that could be deployed on a global scale. The outputs of this section are used in Section 5 to inform a model on Roster Design and elements of Group Formation and Group Management. The next chapter looks at the latter problem in more detail, in particular how to provision intelligent management to a group level that will not impact on the performance of the group in a negative manner.

# Chapter 4

# Group Interaction: Managing Performance

The previous chapter examined issues surrounding Group Formation and the required principles and architectures that can facilitate large scale interactions. Managing this formation and managing the interactions that occur within groups in a manner which does not impact on performance requires a more detailed analysis. This chapter deals with performance and scalability issues in current scenarios involving the management and formation of groups. A number of domains are chosen for analysis which exhibit principles of emerging social networks currently deployed in existing solutions. The approach taken is empirical, with the results presented for analysis.

## 4.1 Architecting Management Platforms

The Policy Based Network Management (PBNM) domain is used to control the behaviour of nodes within a network, typically controlled by policies. Policies are used to describe behaviour, usage patterns and appropriate responses to network based stimulus. Policies are generally defined and deployed to protect a system and govern interactions. In recent years PBNM approaches have been successfully deployed in increasingly diverse domains. The technological and cost barriers to creating and sharing content are much lower than in previous decades, facilitating this deployment. Direct communication between people is also easier, particularly when participants share their *presence* information. However, some communication events, with or without intermediate content, are considered "harmful". Consequently, access control *mechanisms* are used to prevent such harmful communication. For example

- A parent might wish to share family photographs with their relatives, but not with the wider user community of a photograph-sharing site;
- In an organisation, corporate governance procedures impose restrictions on staff using communication tools across groups, both internal and external.

Such scenarios require simple but robust and performant access control *procedures* that are informed by access control *policies*. Implementing such control procedures requires an elegant architecture with separation of concerns among the components.

### 4.1.1 Candidate Components

The eXtensible Access Control Markup Language (XACML) as specified by OASIS XACML-TC (2005) and reviewed in Section 2.3.3 defines an access control policy language and associated processing model. The specification covers in detail how to evaluate authorization requests according to rules defined in policies. The interest of this thesis in the XACML model lies in the associated processing model and the formal terminology used to describe the rules contained within the encoded policy. XACML specifies a number of core components that make up the processing model. The interest from a group management perspective focuses on three of those components with their interactions visible in Figure 4.1

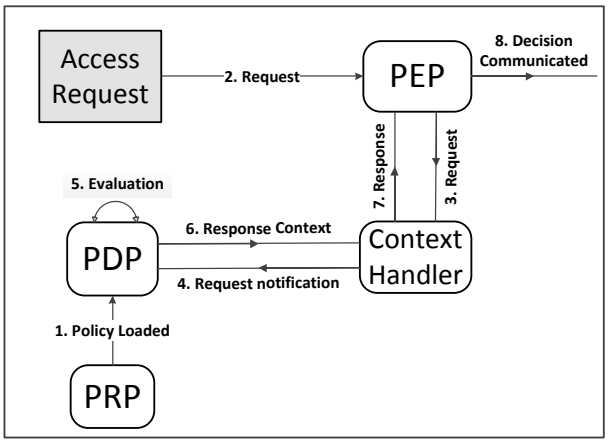



Figure 4.1: Candidate Components based on XACML data flows

The responsibilities associated with each component is as follows:

- Policy Decision Point (PDP) The PDP is a run time component that evaluates incoming request with appropriate policies requested from a PRP. The PDP is

the core logic within the system, charged with interpreting the semantics encoded within a policy set and matching them against the request passed in. The PDP has the power to make recommendations on whether a request is to be accepted or rejected and any additional processing that needs to be carried out, termed *obligations*.

- Policy Enforcement Point (PEP) The PEP is the gateway to the policy system, intercepting requests and passing them to the PDP for adjudication. The PEP is also responsible for enforcing the PDPs decision and carrying out any obligations that the PDP has identified.

- Policy Retrieval Point (PRP) The PRP acts as a gateway to a policy repository. A set of requests for a specific policy set is passed in by the PDP and the PRPs sole responsibility is to retrieve the requested policy set and return the bundle for evaluation.

Having a clearly defined role, responsibility and message flow per component allows for a degree of freedom with respect to architectural design decisions. One such design decision is discussed in the next section.

### 4.1.2 Group Specific PDP

A XACML compliant PDP has to implement, understand and remain compliant with the full XACML suite. In Network Management, this approach is viable, with the cost of arriving at a decision, measurable in milliseconds, well within reasonable consideration. However, in the realm of near real time service access, it can often take longer to arrive at the decision whether to allow the request access (or not), then it does to actually process the logic behind the request. In such instances deploying a fully implemented PDP can be overly complex, with this complexity causing potential delays, delays which are unavoidable if the PDP has to be fully compliant. A case can be made for tailoring the logic behind a PDP to closely match the semantics captured by the Policy Sets associated with a domain or in the case of this thesis, a group. A terse PDP, would not suffer the structural overhead associated with traditional PDPs and would arguably be more portable and easier to maintain. Creating a specific PDP that is semantically equivalent to the core principles that XACML documents is possible, with the development effort potentially reduced. Authoring such a PDP in a DSL that is tailored towards the group can also bring additional benefits. Interpreting a technical vocabulary, already imbued with architectural semantics can be a daunting experience

for a domain expert. By expressing the solution space as a readable DSL, an important bridge between the domain expert and the system is established. An added benefit of developing a customised DSL is the lightweight nature of the design, making it an ideal candidate language for deployment within this problem domain.

### 4.1.3 Distributed PDPs

Distributing the PDP architecture through smaller domain specific PDPs makes groups and their management more portable. This makes logic easy to distribute to multiple management nodes housed locally or distributed across a network. Every network node that runs an interpreter capable of understanding the language the PDP is encoded in can potentially assume the role of a PDP. By implementing in portable code, one that is specified in a text base format that is easily distributed, a new PDP can be created and managed at run time. *Inflating* a PDP in this manner and subsequently tearing it down would allow the management system evolve in a scalable manner, with redundant PDP functionality naturally removed as the lifecycle of the groups come to their conclusion. By distributing the PDP intelligence, the management system becomes more fault tolerant. The impact of a failure, be it physical or logical, to a central PDP can be very costly from a management point of view. A single PDP failure will have consequences for an individual group but the overall management platform remains resilient and functional. Additionally, by adopting a distributed PDP approach, PDPs can physically reside on multiple machines within the networks. As scale increases, the likelihood of a hardware failure increases. By abstracting PDP responsibility to a distributed model, it should be possible for a PDP under intense load to migrate to a more powerful host, thus limiting the potential for a hardware failure on a particular machine.

### 4.1.4 Advanced Management Potential

Social media deployments are increasingly turning to cloud based solutions to host their services. As services are deployed into third- party clouds the cost of operating these services change in their profile to become more opex (operational expenditure) based, as they typically incur monthly, recurring fees based on their usage of cloud resources. Charging profiles are often used against service providers with Service Level Agreements (SLA) protecting and guaranteeing aspects of the service including uptime, response time and resource utilisation such as bandwidth or CPU cycles. SLA breaches can be expensive and understanding the probability of an SLA violation can offset potential

costs before they become an issue. An intelligent management system, capable of representing group specific SLA semantics within a group specific policy system has inherent advantages for service providers over existing techniques.

### 4.1.5 Performance Benefits

Request evaluation within a PDP typically takes the form of Algorithm 4.1. With $T_{req}$ the total time taken to process a request equal to the time it takes to evaluate the request, $T_{eval}$ and the time it takes to respond, $T_{response}$.

**Algorithm 4.1:** Time for processing a request

$$\mathrm{T}_{req} = \mathrm{T}_{eval} + \mathrm{T}_{response}$$

The PDP itself has complete control over the $T_{eval}$ metric which is broken down into Algorithm 4.2. The time required to search the policy sets for the specific policy, $\mathrm{T}_{search(policy)}$ added to the time required to process the policy $T_{process(policy)}$

**Algorithm 4.2:** Time for evaluating a request

$$\mathrm{T}_{eval} = \mathrm{T}_{search(policy)} + \mathrm{T}_{process(policy)}$$

Minimising the time taken to search the policy sets is the key to minimising the overall evaluation as the time taken to process the policy should be uniform with respect to the complexity of the request. Policy sets for generic PDPs can encode thousands of rules and processing time to traverse the tree and identify the correct policy can be costly. In a domain specific PDP, the policy sets would be tailored towards the specific domain. While large policy sets for an individual group or domain are still possible, in general, the policy sets will be smaller due to not having to capture and encode the semantics of the overall system. Consequently the time taken to traverse the tree should be minimised. Additionally, if the encoding language of the underlying PDP takes the form of a Domain Specific Language (DSL), the traversal of data structures, of which policies are often stored upon retrieval, can be greatly improved. Algorithm 4.3 represents the optimal goal of any PDP and policy set combination, minimising the time taken with respect to the number of Policies, $P$ and the time taken to find the correct policy, $T_{search(policy)}$

**Algorithm 4.3:** Optimal searching algorithm

$$\min\{T_{search(policy)}\}$$

Where $T_{search(policy)} = F(\#P)$ and P is the set of all policies at the PDP whose performance is being evaluated PDP: $P = \{p_i \epsilon P_{PDP}\}$

What the basic algorithms outlined does not factor in is additional benefits that a domain specific PDP will bring to the overall system. The cost of operating the PDP should be considerably less then supporting a generic PDP ecosystem. The PDPs will have a smaller code base due to being tailored for the particular task at hand. This translates to tangible benefits relating to CPU cycles and power output required to maintain the PDP.

### 4.1.6 PDP Design Patterns

The overall management system could be described as a cluster of loosely connected PDPs, working together to manage the groups contained within the system. Managing the lifecycle of multiple individual PDPs requires a design influenced by classical software engineering design patters. The Factory and Builder design patterns fit this requirement.

A Factory design pattern for PDPs is shown in 4.2. The definition put forward by Gamma *et al.* (1994) for a Factory is as follows: *Define an interface for creating an object, but let the classes that implement the interface decide which class to instantiate. The Factory method lets a class defer instantiation to subclasses.*

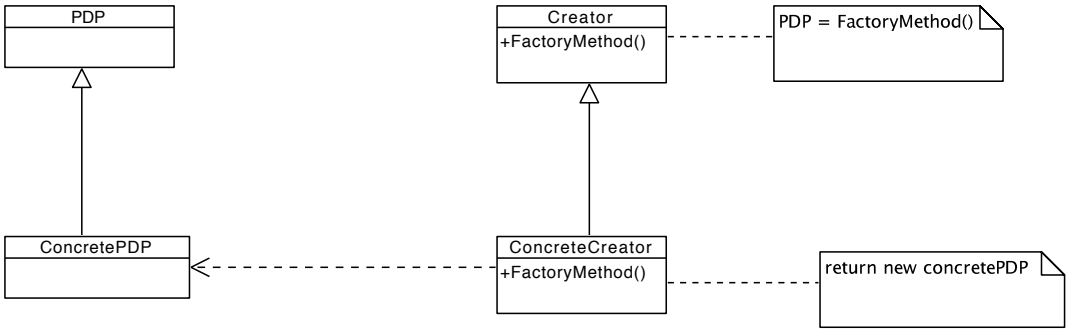



Figure 4.2: PDP Factory Design Pattern

Where the *PDP* element defines the interface of objects the factory method creates, in this instance a PDP. *ConcretePDP* implements the PDP Interface. *Creator* declares the factory method, which returns an object of type PDP. A default implementation of the factory method may also define a default ConcretePDP. *ConcreteCreator* overrides the factory method to return an instance of ConcretePDP.

Skeleton PDP implementations have to follow the same logical structure and implement a minimal set of functionality. A factory style approach should be able to produce a domain tailored PDP by receiving a specification for the type of PDP to be produced and consequently creating it around the default skeleton structure already established. The factory logic should be able to influence changes to individual PDPs during their lifecycle, adding and removing functionality as well as terminating the PDP when the group naturally expires. The responsibility of the factory might be embedded within an existing PDP or multiple PDPs which have the resources and capability to sustain the entire PDP ecosystem. As the PDP style put forward is distributed, it stands to reason that a Factory approach should also be distributed with any PDP capable of adopting the role, providing another layer of resilience and protection to the management system.

An alternative design pattern is the Builder Pattern, defined by Gamma *et al.* (1994) as follows: *The intent of the Builder design pattern is to separate the construction of a complex object from its representation. By doing so, the same construction process can create different representations.*. Figure 4.3 shows a Builder oriented approach to creating PDPs.

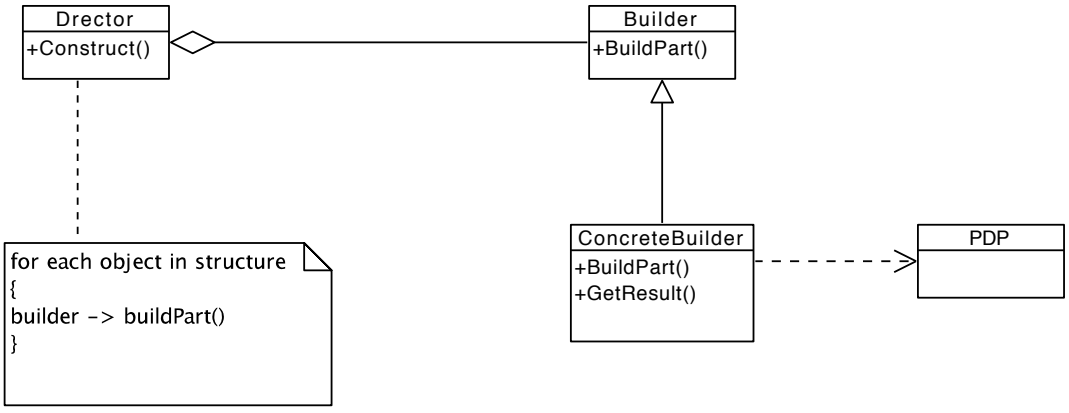



Figure 4.3: PDP Builder Design Pattern

Where the *Builder* element specifies an abstract interface for creating parts of a PDP object. *ConcreteBuilder* constructs and assembles parts of the PDP by implementing the Builder interface, additionally keeping track of the representation it creates and providing an interface for retrieving the PDP. *Director* constructs an object using the Builder interface. *PDP* represents the object under construction, with ConcreteBuilder building the PDPs internal representation. A Builder design pattern would again decouple default behavior associated with a PDP, allowing a repository of logical components to be built up. Selecting individual components and wiring them together to form a domain specific PDP could still be distributed. Running PDPs could act as a means to access the core *parts* required to build a new domain specific PDP. Similarly to the factory based approach, logic would be required to influence changes to an individual PDP during its lifecycle. Both approaches are not mutually exclusive and design patterns are used to guide an implementation, with often a hybrid approach more suited.

The Factory Pattern is chosen due to the emphasis on creating a family of *product* objects, be they simple or complex. In this case the product produced is a functional PDP, with the Factory pattern offering more customisation. The ability to create the PDP in one step, as opposed to the multiple steps required within a Builder pattern, is also an attractive feature, minimising the additional overhead required to create and instantiate the PDP.

## 4.2 Policy Informed Domains

Adopting Node.js and related technology has offered substantial opportunities in devising a robust simulation and viable deployment environment as discussed in Section 3.3.2. The approach also offers a very specific advantage uniquely applicable to this work, namely the ability to bring Policy Based Network Management (PBNM) techniques into play. This is a mature and well-understood field, with substantial systems in the field managing some of the most complex and reliable communications infrastructure ever constructed. In such systems, Policies enable critical systems to be controlled in a secure manner, enabling services to be flexibly configured without interruption. However, network management policies are conventionally constructed in XML using the XACML standard. These have some specific characteristics which have been documented in Chapter 2 which would inhibit their deployment in certain contexts. Such contexts include highly scalable, short burst scenarios such as the one under consideration here. Taking this well understood policy-based approach and architecture into our

system in order to provide a degree of control and management over how check-ins are processed and consequently how groups are formed offers potential benefits. The traditional approaches as embodied by XACML specify an architecture and approach to deploy management policies. Encouraged by the performance gains visible by changing the representation format, we decided to rework the traditional approaches to exploit unique opportunities within the Node.js technology stack. Specifically, this work uses JSON and JavaScript itself as the policy language, with supporting components and patterns. This facilitates a highly expressive notation that can be further tuned into a full Domain Specific Language (DSL) over time with the aid of rescue organisations. Thus a number of group formation policies which can be adapted, modified or replaced on the fly can be expressed in a highly expressive notation. This notation is terse, allowing for domain experts, with minimal programming experience, a means to influence the strategy deployed on the ground. We exploit JSON principles, language features such as closures, and prototypical object creation, to produce what is effectively a policy language for group formation strategies. Policies expressed in this language are directly executed within the simulation environment, enabling the elicitation of valuable experimental data on the viability of alternative approaches to group formation within the scenarios we have selected. Figure 4.4 shows where a policy component can fit into the overall architecture of the simulation.

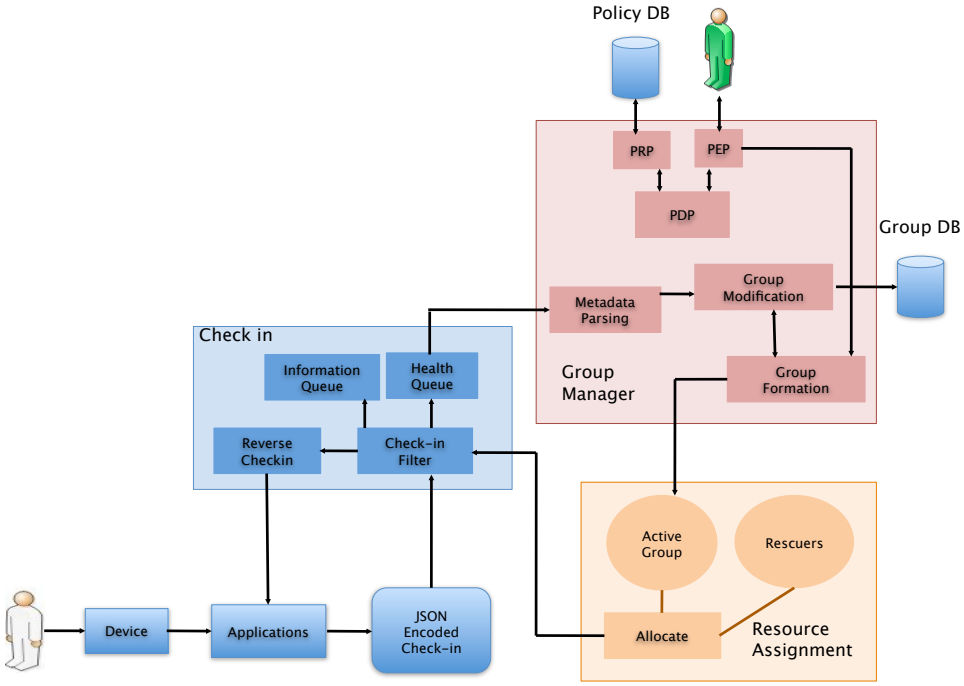



Figure 4.4: Overall Architecture View with Policy Component

Inspired by the XACML architecture OASIS XACML-TC (2005) and best practices for division of responsibility as outlined in the reference documentation, a light weight, JavaScript authored Policy Management environment was added to the system. A Policy Decision Point was implemented along with a Policy Enforcement Point (PEP). The PEP was responsible for handling incoming policy requests and relaying the decision to the relevant components, typically the Rescue Assignment Component and the Group Manager. A Policy Retrieval Point (PRP) was also implemented to communicate directly with the policy repository. The policies themselves, were authored in JSON, providing a lightweight, readable policy capable of being authored by non domain experts on the fly without requiring the system to stop.

### 4.2.1 Policy Algorithm and Implementation

Policy Algorithms encoded within JavaScript are used within the system. Algorithm 4.4 describes the process for the Distributed strategy of grouping patients and rescuers.

This algorithm is implemented logically within a JavaScript closure, visible in Listing 4.1, which can subsequently be associated with a group Object within the system.

**Algorithm 4.4:** Distributed Group Rescuer Allocation Algorithm

**if** $checkins.size \geq currentStrategyGroup.size$ **then**
  We still have people who need help in our current group:
  **if** $policy.name == distributed$ **then**
    We match Patient p1 with Rescuer r1:
    **if** $a \in currentStrategyGroup$ && $r \in availableRescueGroup$ **then**
      Create group $\{p1, r1\}$
      $currentStrategyGroup = strategyGroup \setminus p1$
      $availableRescueGroup = availableRescueGroup \setminus r1$
      **return** $\{p1, r1\}$

```
return _.groupBy(currentStrategyGroup,
function(checkin,rescuer){return
allocateRescuer(checkin.name,rescuer)});
```

Listing 4.1: Closure Representation of Rescuer Allocation

This allows powerful and complex policy algorithms be realised in a short few lines of DSL style code. Taking this concept further then simple grouping algorithms, which are effectively paramaterised control structures, the prototypical inheritance within JavaScript can be leveraged. A Policy Object, encoded against a paramaterised control structure, which is human readable, can be attached to groups and modified on the fly. The Object created can be modified at run time, effectively allowing a strategy be pushed to it by an end user without physically changing the source code or stopping the system. The code in Listing 4.2 is a working Distributed Policy document:

```
function Distributed() {
    return {
        primary: "distributed",
        secondary: "triage",
        location: "true",
        numGroups: 3,
        groupCreation: function (checkins) {
            return _.groupBy(checkins, function (checkin) {
                return checkin.location;
            });
        },
        allocateRescuer: function (name, rescuer) {
            return newRescueGroup(name, rescuer);
        }
    };
```

```
}
```

Listing 4.2: Distributed Group Formation Function

This essentially renders the previous closure example to a more simplified level:

```
Distributed.allocateRescuer(name,rescuer);
```

Listing 4.3: Function simplification

More importantly it allows a JSON encoded document, such as the example in Listing 4.4, to be authored by non programming experts. This can directly influence the properties of an Object, and by extension the strategy dictated.

```
{
  "primary" : "centralised",
  "secondary" : "FIFO",
  "location" : "true",
  "numGroups" : 1,
}
```

Listing 4.4: Policy Document Sample 1

With associated program level listeners, invoking a policy change of direction in an external JSON file, the primary and secondary approaches of a Distributed object could change. More programmatic medical domain experts could associate new constructs in an external manner by appending functions such as the example in Listing 4.5

```
{
"primary" : "centralised",
"secondary" : "FIFO",
"location" : "true",
"numGroups" : 1,
},
triageWeighting: function (checkins){
return _.groupBy(checkins,
function("triage1, "triage2"){return checkin.triage;});
}
```

Listing 4.5: Policy Document Sample 2

In this case, a new function is associated with the Distributed object and available to it immediately. This would allow a trained domain expert to micro manage a rescue operation in a manner currently not envisioned. The results could dramatically affect a running operation and ensure that the system has the flexibility to manage in different disaster domains. The sequence diagram in Figure 4.5 shows the message flow that an external policy change could make on a live system, filtering down to the rescuer level.

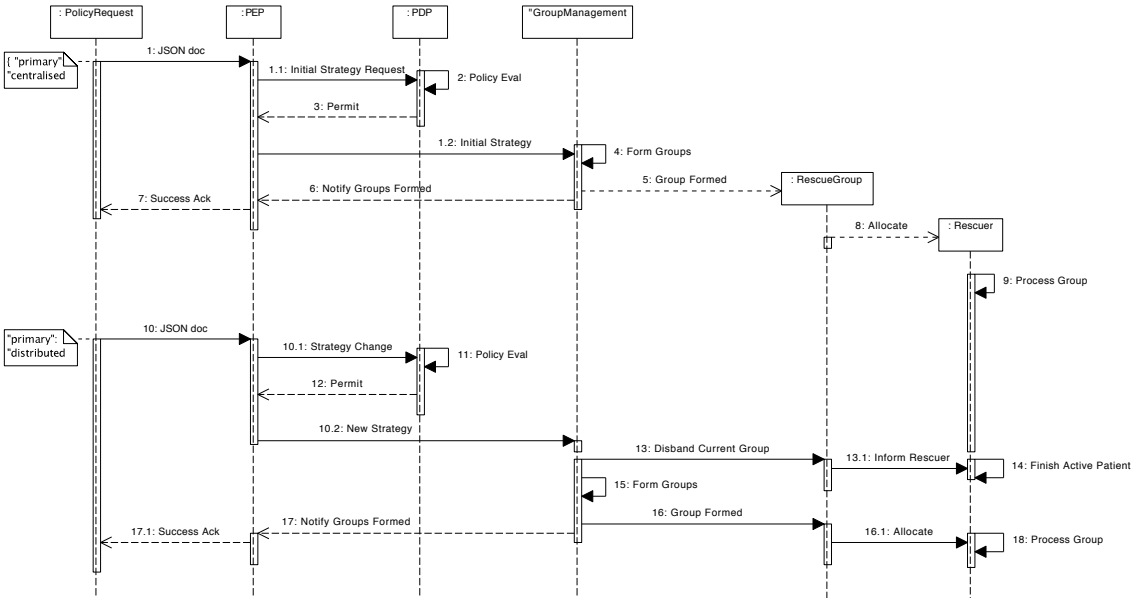
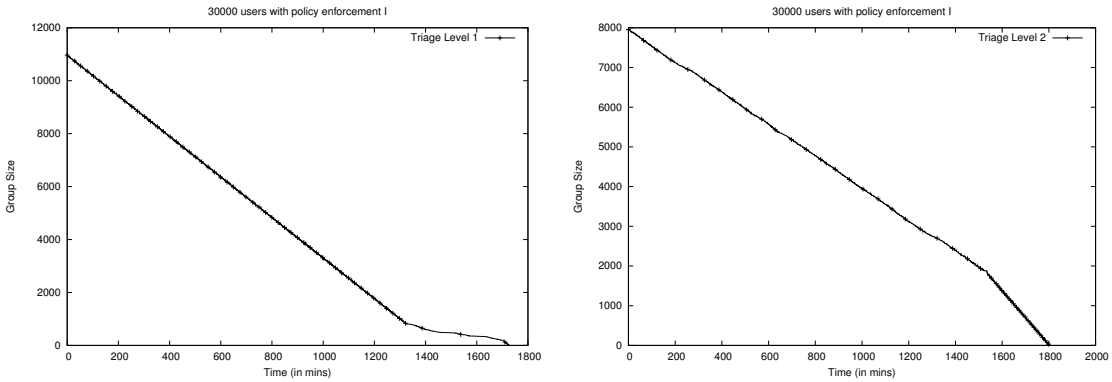



Figure 4.5: Sequence Diagram for Policy Changes

### 4.2.2 Policy Results

Applying this language level feature to the simulation used in the previous Chapter results in a different behaviour pattern for the rescuers. The ability is now in the hands of the rescue coordinator to make changes on the fly according to incoming reports. Changing the strategy on a live operation produces some interesting results. A simulation was designed where the approach started as Centralised before switching to a combined triage level. The combined level is initially weighted to favor Triage 1 but including some Triage 2 patients. As the triage 1 numbers exhaust the weighting is flipped, pushing more triage 2 patients out of the system as the active group ratio now favors their rescue. The policy changes occur at timestamp 900 when the central group expires, at this point the combined Triage approach kicks in. Again at timestamp 1300 the Triage weighting is changed by an external policy call. Figure 4.6 shows this effect:




(a) Policy Group for Triage 1.

(b) Policy Group for Triage 2.

Figure 4.6: Policy Approach to Rescuer Allocation

The approach benefits both Triage 1 and Triage 2 patients and passively affects Triage 3. With the decreased service time, Triage 3 indirectly benefits by triggering earlier in the scenario.

### 4.2.3 Summary

The deployment of DSL style policies within the simulation showed a marked improvement on providing strategic responsiveness to a disaster management scenario. It also showed that management policies could be deployed in a manner that could potentially improve overall performance. However, the representation of the policies and the design of a policy execution environment is a challenging task. The next section attempts to addresses this and expand upon the design of DSL simulations.

## 4.3 Performance of Policy Evaluation

### 4.3.1 Motivation

Simple but robust and performant access control *procedures* that are informed by access control *policies* are required to manage the interactions within emerging social networks. Any access control mechanism used to protect group communications must provide a decision in near real time in order to preserve usability of the underlying communication medium. Much of the public literature on access control policies focuses on techniques to ensure that sets of deployed policies are consistent with high level security requirements. This section instead examines how to optimise the policy eval-

uation performance at the Policy Decision Point (PDP) in order to meet performance objectives.

In related work by Butler *et al.* (2011) the performance characteristics of two open source PDP implementations were analysed. Based on this analysis, the following features could lead to improved PDP evaluation performance:

- Policies and requests should be encoded more efficiently
  - policies and requests should be relatively terse, to reduce the string handling overhead per request
  - policies and requests should be encoded in a way that minimizes the parsing overhead.
  - policies should be directly implementable.
- PDP implementations should be more efficient
  - policies and requests should be stored in ways that make retrieval more flexible and efficient
  - the PDP should scale outwards, to enable more efficient use of available resources.

This work attempts to address some of these issues by examining policy representations and the impact this has on performance.

### 4.3.2 Architecture

With the Node.js framework Dahl (2009), an elegant solution can be engineered for traditional scalability problems and an alternative approach taken for domains that might benefit from a non-blocking IO approach. In addition to using Node.js, a complementary data persistence system is required. Maintaining the low-friction objective, a NoSQL database Cattell (2011) with a Node.js interface was sought. NoSQL databases are non-relational, distributed databases that deliver horizontal scalability. Several options were considered, with each having bindings to Node.js ensuring compatibility. The data structure-oriented Redis server Lerner (2010) was chosen because of its speed (in 1 minute, on one instance of the experimental platform, it averaged 11300 answered requests per second) and its data structures directly supported sets, lists and hashes, easing policy management.

#### 4.3.2.1 Advantages for access control

Node.js provides a number of advantages that benefit the design of the access control system.

- It uses the CommonJS standard for APIs and best practices Kowal (2009).
- It provides many useful features without extra coding.
- It supports better modular design through component-based programming, benefiting from specifications supporting interoperability between server modules and even between server-side and client-side modules.
- The Non-Blocking architecture facilitated by JavaScript callbacks makes more efficient use of CPU resources than traditional systems that sit idle while a complex IO request is executing. In a Node.js deployment, the spare CPU cycles can be used more productively, e.g., to handle the next request or prepare the response.
- It uses events to trigger callback execution. A benefit of the event model is a publish/subscribe mechanism built into the environment. This promotes greater flexibility, since the access control system can include *listeners* (handlers) for particular events, such as *obligations*.
- The absence of complicated blocking semantics simplifies development.

In the PBNM world this lends itself to tackling high input requests or requests with near real time processing characteristics, while also endearing itself to greater self checking and management. The *spare* processor cycles while waiting for the policy algorithm to reach a conclusion could be put to better use, without impacting on the response times as all requests ultimately execute in parallel. This opens up the possibility of the architecture becoming self monitoring and regulating, with a suite of conformance and conflict analysis tests passively ensuring the validity of the system.

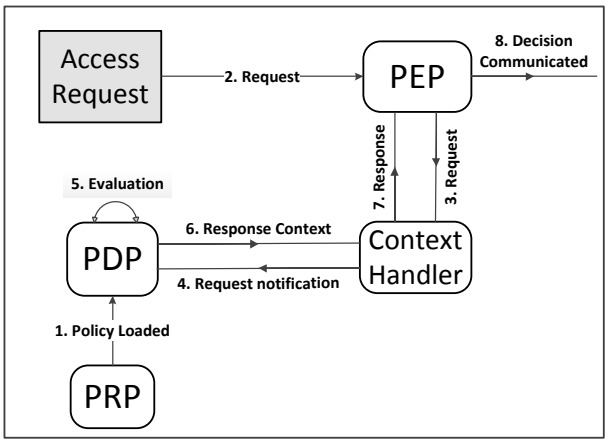



Figure 4.7: XACML data flows and components relevant to this work

The OASIS XACML TC defined data flows, as previously highlighted in Figure 4.1, formed the basis of the architecture. Figure 4.7 shows this diagram again for clarity. The flows required for a minimal installation of XACML were implemented in a Node.js supported Policy Execution Point (PEP), Policy Decision Point (PDP) and Policy Retrieval Point (PRP).

The other characteristics identified in Section 4.3.1 relate to implementation efficiency. The prototype PDP uses Redis to ensure flexibility and high performance when processing policy data structures, and the non-blocking I/O coding style fostered by the Node.js framework makes more efficient use of computing resources. The simplicity arising from streamlining policy evaluation should not be underestimated: the domain (policy) model is based on events, so if the execution model is likewise, there is better alignment between the two. The PDP conforms to the CommonJS standard specification for module development. This standard ensures the interoperability of a module and allows for integration with existing systems or future systems with syntax extensions. It also ensures that any Node.js modules will have the minimum features required to provide interoperability within the PDP. The PDP itself is composed of several functions governing both general and policy set specific behavior, mostly relating to style and formatting issues within XACML. The design is flexible with the ability to handle various policy and request combinations. This is evident throughout the njsrPDP scenario testing with multiple input types supported. The PDP is thus a fully independent generic PDP, with a core set of generic functions to interpret and successfully parse specific policy sets.

### 4.3.3 JSON based Policy Representation

A JSON representation was chosen to represent the policies. As discussed in 2.4.5 JSON as a direct replacement for XML loses nothing syntactically. Having already opted for a JavaScript implementation for the policy engine, and opted for a no-SQL database, with a JavaScript binding, a JavaScript interpretable representation format ensures a *low-friction* environment for policy evaluation. The term JSONPL, or JSON Policy Language, is used to reference the policy language represented in the JSON format.

#### 4.3.3.1 Conversion of existing XML policies or requests to JSON

Several attempts (such as BugLabs (2012)) to automate conversion between XML and JSON formats have emerged, thus enabling interoperability between XML policies/requests and a PDP that handles JSON natively. The conversion process makes structural changes to how data might traditionally be represented within JSON. For example, XML is designed as a language independent data representation format, which means metadata is associated with elements in order to ensure correct interpretation at a language level. Translation from XML to JSON indirectly brings this metadata, which is unnecessary within a JSON compliant language such as JavaScript. The result is a "bloated" JSON format. Additionally, XML can contain sibling elements with the same outer identifier. However, JSON is a key:value storage mechanism, so it cannot have the same name for two keys. The solution, when translating, is the extensive use of arrays in JSON. While the JSON produced from a translation process is valid and semantically identical, it is harder to read and extra programmatic safeguards are needed to ensure correct interpretation.

#### 4.3.3.2 JavaScript Object Notation Policy Language: JSONPL

Examining the output of the translated XML to JSON policy sets, the predefined vocabulary and encoded semantics expressed in the XACML 2.0 specification was apparent. Stripping away the redundant metadata and cleaning up the array translation process produced a policy vocabulary encoded in JSON that semantically was identical to the original XML policy. The resulting representation is termed JSONPL (JavaScript Object Notation Policy Language) and Figure 4.8 highlights a policy represented within this language. As with XML, deep nesting of arrays and objects is possible within JSON, allowing complex hierarchical structures to be represented. The dot notation is the most natural way to access data within a JSON document. For example, in Figure 4.8, the value associated with `role` can be accessed directly by calling

```
"Policy":{
    "id":"RPSlist.7.0.1",
    "target":{
        "subjects":{
            "subject":{
                "role":"admin"
            }
        },
        "resources":{
            "resource":{
                "isPending":"false"
            }
        },
        "actions":{
            "action":{
                "action-type":"write"
            }
        }
    },
    "rule":{
        "id":"RPSlist.7.0.1.r.1",
        "effect":"permit"
    }
}
```

Figure 4.8: JSONPL Policy Excerpt. The original XACML-encoded policy had 1473 characters versus 454 characters for the JSONPL encoding.

`Policy.target.subjects.subject.role`, yielding the value `admin`.

The hierarchical structure of JSONPL mirrors that of XACML so a comparison is instructive. As with XQuery/XPath processing of XML documents, the execution time for retrieving a value from a known location in a JSON data structure is independent of the size of that structure. Policy and/or rule combining algorithms can be applied in JSONPL in the same way as in XACML. The policy language is as expressive as an XML based XACML policy, is arguably more human-readable and provides native compatibility with many programming languages, easing authoring and interpretation issues. To the best of our knowledge, a JSON-based access control policy language has not been attempted before and as such represents a novel contribution. All the desirable features relating to policy and request specification and encoding identified in Section 4.3.2.1 are provided.

### 4.3.4 Evaluation

#### 4.3.4.1 Comparison of PDPs

Experiments were performed to compare the JSON/Node.js/Redis implementation described above with more traditional XACML/Java implementations of SunXACML and EnterpriseXACML. A set of XACML policies and their related requests was chosen and were translated manually to their JSONPL equivalents. The two Java-based PDP implementations were placed in STACS (Scalability Testbed for Access Control Systems) Butler *et al.* (2011) so that service times per request could be recorded in a repeatable fashion. The prototype Node.js implementation was instrumented in the same way, taking advantage of the Node.js eventing model to collect service times based on the same triggering events that were used in STACS:

- PDP Policy Read *start*
- PDP Policy Read *end*
- Request *arrives at* PDP
- Response *leaves* PDP.

A simplified queueing discipline was employed, namely, when response $n$ from the PDP arrived at the PEP, it triggered the submission of request $n+1$ from the PEP to the PDP. This sequential processing was easily achieved in STACS using loops and in the njsrPDP harness using callbacks. The entire experiment was replicated $N_{\text{rep}} = 100$ times, in random order, for each set of `host` $\times$ `pdp` conditions. When measuring elapsed times in the PDP, we do not have full control over other processes that can use computing resources needed by the PDP. Thus the measured service times generally have a positive bias. More formally, assume that service time $\hat{t}$ measured by $t_i, i = 1, \ldots, N_{\text{rep}}$ is subject to additive nonnegative error $e_i$ according to

$$t_i = \hat{t} + e_i, \text{ where } e_i \geq 0, \ i = 1, \ldots, N_{\text{rep}}. \tag{4.1}$$

The best estimate of $\hat{t}$ is given by $\min_i\{t_i\}$, $i = 1, \ldots, N_{\text{rep}}$.

The measured service time data was standardized to use the same labels and time units to ensure that data features were consistent between STACS and non-STACS sources.

| Name | Type | Possible Values |
|---|---|---|
| policy | Common | `continue` |
| reqGrp | Common | `single` |
| host | Factor | `bear`, `inisherk` |
| pdp | Factor | SunXACML, EnterpriseXACML, njsrPDP |
| duration | Response | Numeric |

Table 4.1: Service time measurements and their context.

The factors considered in our main experiment are shown in Table 4.1. The `continue` policy and `single` request group are published (in XACML form) as part of the test suite for XEngine Liu *et al.* (2008), and were translated to JSON format as described earlier. This policy set and associated requests was used in the experiments and models access control rules and requests for a Conference Paper Management System. While that domain does not require microsecond evaluation times, the policy set contains reasonably complex business rules such as separation of duties constraints and other features representative of real-time corporate communications. The two `host` instances are Intel 64-bit dual-core machines, each with 2GB RAM but differing in other computing resources, running Ubuntu 11.04.

The primary experiment compares njsrPDP with two existing XACML PDP implementations. The secondary experiment examines *how* njsrPDP achieves increased performance.

#### 4.3.4.2 Experimental Scenarios

Six experimental scenarios are considered, as described in Table 4.3.4.2 and visible in Figure 4.9, and were used to compare the effects of different policy and request formulations for a given PDP (in this case, the njsrPDP prototype).

| | Manually Generated Policies | Auto generated Policies (bloated) |
|---|---|---|
| Manually generated Requests | Scenario 1a | Scenario 1b |
| Auto generated requests (bloated, pre prepared) | Scenario 2a | Scenario 2b |
| Auto generated requests (bloated, on the fly) | Scenario 3a | Scenario 3b |

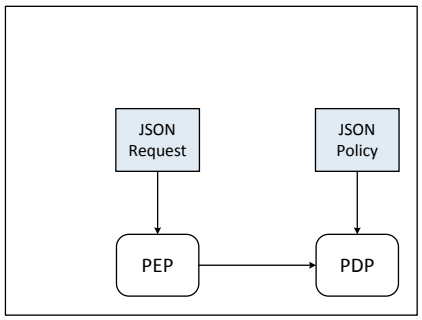
Table 4.2: Scenario conditions

Summarising, the scenarios under investigation:

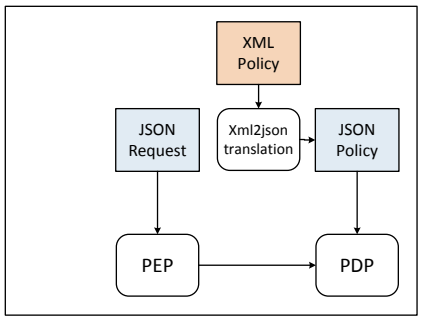
- the A scenarios formulate the *policies* as JSON in ways that are "optimised" for evaluation performance, while the B scenarios use the policies as translated by a generic XML to JSON converter.
- the "1" scenarios formulate the *requests* as JSON in ways that are "optimised" for evaluation performance.
- the "2" and "3" scenarios use the requests as translated by a generic XML to JSON converter. They differ in respect of when the translation occurs: *before* reaching the PEP for "2", or within the PEP itself for "3".

Note that the experimental conditions in Scenario 1A are those used when comparing njsrPDP with the SunXACML and EnterpriseXACML PDPs, see Section 4.3.4.1.

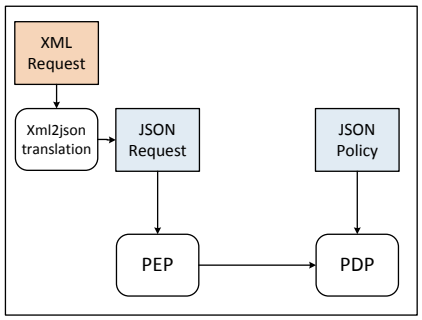
In summary, all of the policies and request artifacts were originally encoded as XACML and so benefit from the XACML ecosystem. However the artifacts are converted to JSON by different methods and at different stages of policy evaluation. The performance improvements arising from each research contribution can be estimated by comparing the timing results.



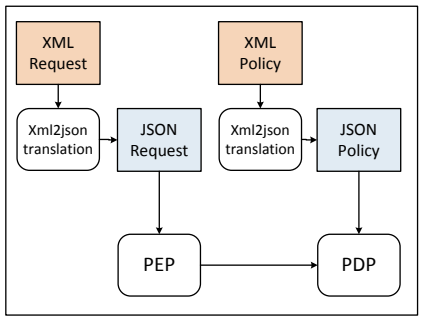

(a) Scenario 1A: Manual×Manual



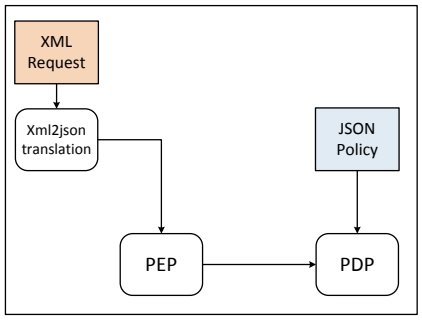

(b) Scenario 1B: Manual×Auto



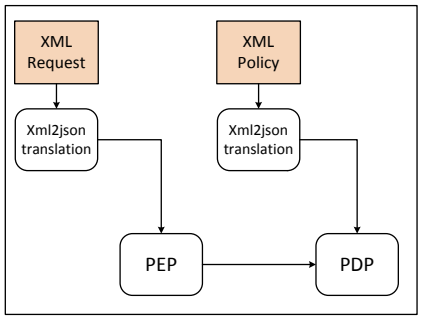

(c) Scenario 2A: Prepared-Auto ×Manual




(d) Scenario 2B: Prepared-Auto×Auto




(e) Scenario 3A: Inflight-Auto×Manual




(f) Scenario 3B: Inflight-Auto×Auto

Figure 4.9: njsrPDP policy×request scenarios; scenario conditions are defined in Table 4.3.4.2.

### 4.3.5 Results

#### 4.3.5.1 Comparing njsrPDP with its peers

| | Df[a] | SumSq | MeanSq[b] | F value[c] | Pr(>F) |
|---|---|---|---|---|---|
| host | 1 | 1.97e-05 | 1.97e-05 | 2.24e+04 | $< \varepsilon$ |
| pdp | 2 | 1.15e-04 | 5.75e-05 | 6.55e+05 | $< \varepsilon$ |
| decision | 2 | 1.00e-09 | 5.00e-10 | 5.14e-01 | 0.60 |
| requestIndex | 190 | 1.61e-07 | 1.00e-09 | 9.67e-01 | 0.61 |
| host:pdp | 2 | 7.50e-06 | 3.75e-06 | 4.27e+03 | $< \varepsilon$ |
| Residuals[d] | 954 | 8.37e-07 | 1.00e-09 | | |

[a](Number of) degrees of freedom
[b]SumSq/Df
[c]F ratio: MeanSq/MeanSq_Residuals
[d]Other, unspecified factors

Table 4.3: Analysis of Variance: host, pdp, host:pdp effects are very significant—$\alpha$ probability underflows machine epsilon $\varepsilon$.

| host | bear | inisherk |
|---|---|---|
| time | 6.3e-04 | 3.7e-04 |
| #replicates | 576 | 576 |

Table 4.4: Analysis of Means: host `inisherk` has better performance than `bear`.

| pdp | SunXACML | EnterpriseXACML | njsrPDP |
|---|---|---|---|
| pdp | 5.56e-04 | 8.61e-04 | 9.3e-05 |
| #replicates | 384 | 384 | 384 |

Table 4.5: Analysis of Means: PDP njsrPDP has better performance than the other PDPs.

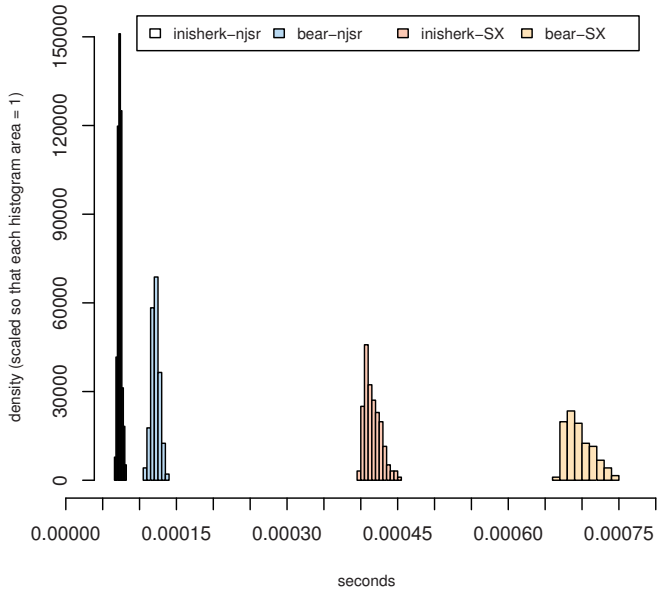
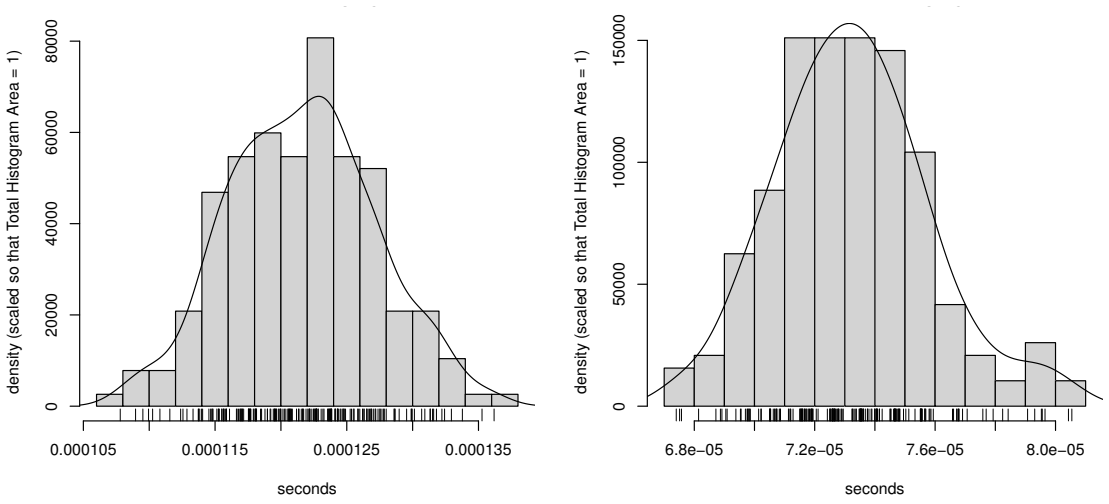



Figure 4.10: Comparative service time histograms for hosts `bear` and `inisherk` and PDP implementations SunXACML and njsrPDP, for Scenario 1A.




(a) njsrPDP on bear

(b) njsrPDP on inisherk

Figure 4.11: njsrPDP request service times on hosts `bear` and `inisherk`.

Figure 4.10 shows histograms of the service times for SunXACML, a *reference* Java-based XACML PDP, compared with the service times for njsrPDP, the implementation introduced in this paper. The influence of the `host` and `pdp` factors can be seen clearly. Indeed, the new implementation has noticeably better performance when other factors are equal.

The Node.js/Redis prototype PDP implementation, labeled njsrPDP in Figure 4.11 has the following performance features:

1. The mean service time per request is much less (one sixth that of SunXACML, one eighth that of EnterpriseXACML), see Table 4.3.5.1.

2. The performance profile for njsrPDP is bell-shaped; for EnterpriseXACML it is approximately uniformly distributed; for SunXACML it is a skewed mixed distribution.

3. The implementation on the two hosts shows a similar profile (see Figure 4.11) though different performance levels, see Table 4.3.5.1 because `inisherk` has a faster CPU and more L1 cache. This suggests that performance scales vertically on a single host and also that the performance profile and observations are *reproducible*.

It should be noted from Table 4.3.5.1 that these differences are statistically significant Dalgaard (2008) and hence are highly unlikely to arise by chance. The challenge is to show how the design principles outlined in Section 4.3.1 and implemented in the JSONPL prototype described in Section 4.3.3 contribute to the statistically significant performance improvements summarized in Table 4.3.5.1.

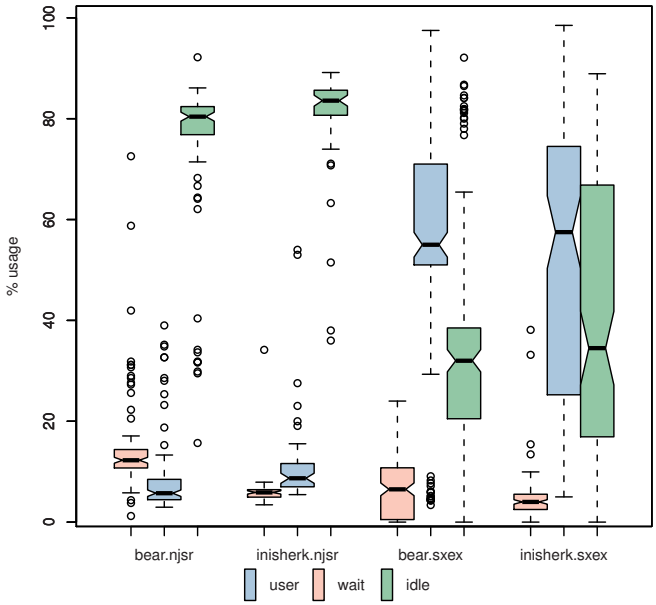



Figure 4.12: CPU usage for selected host × pdp combinations. Hosts are `bear` and `inisherk` and PDPs are SunXACMLEnterpriseXACML (`sxex`) and njsrPDP (`njsr`).

The system resources used by the JSON and XACML implementations were captured using `dstat`, which collects resource statistics (cpu, memory, disk usage, etc) on a timed basis while the experiments run in the testbed. Figure 4.12 shows that njsrPDP uses far less CPU (10% versus 60%, say). The cpu `wait` time is generally low, suggesting that both Node.js and the JVM are quite efficient. However, the `user` cpu cycles are much greater for the Java/XACML implementations. The CPU has to work much harder to evaluate policies in Java/XACML PDP implementations. Furthermore, the `idle` cpu usage is much higher for njsrPDP, suggesting there is much more capacity available for increased throughput.

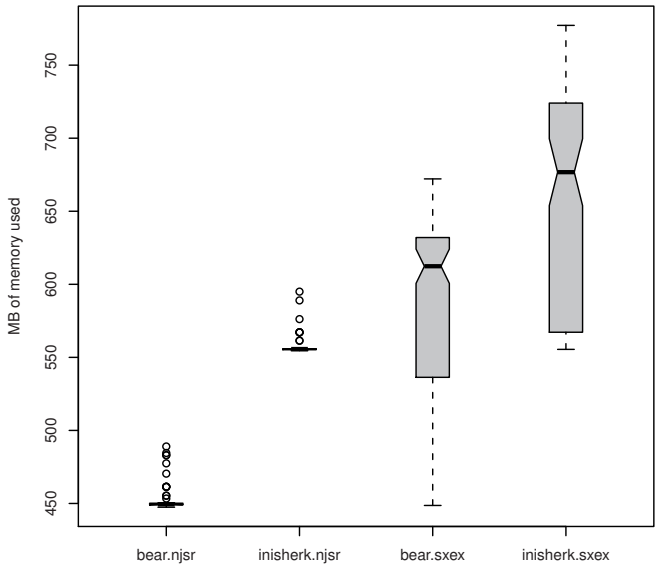



Figure 4.13: Memory usage for different host × pdp combinations

The memory usage was also recorded, supporting the contention that njsrPDP makes particularly efficient use of computing resources, including 35% less memory. Figure 4.13 shows this comparison.

### 4.3.6 Policy-Request Scenario Comparison

Scenario 3 incurs significant overheads. Firstly, the converter requires 83% of the total time needed to make the access decision. Secondly, translation introduces a large object that needs to be maintained at the top of the callback chain. When the PDP evaluates and wishes to pass its decision to the PEP it must "walk" back up the callback chain. The top level callback needs to retain a link including the context of the request and its arguments throughout the whole chain. By placing such a large object in the top callback, translation imposes greater overheads down the callback chain, so the computation time is increased. Therefore the next step is to investigate how the system would perform if the penalty for translation, which increases evaluation time in two ways, were removed.

By pre-translating the requests the overhead incurred in translating the requests at run time is removed as well as the added overheads in the callback chain. The challenge becomes that of guaranteeing safe and accurate policy evaluation. Some approaches, identified in Section 4.3.3, impose performance losses while traversing arrays, with other

losses emerging when developing the PDP. One complication is that, depending on the XML schema, the ordering of some child elements may be unspecified. Consequently the position of sibling child elements in policies within the same policy set can be different. A XACML PDP's XML parser has no difficulty in this regard but the translating program makes no allowance for consistency in the generated JSON. Thus njsrPDP has to account for this, handling all ordering permutations so as to operate correctly. While these problems also occur in Scenario 3, the additional translation overhead masked this feature. A minor performance gain was identified when using a combination of pre translated JSON requests and optimized (JSONPL-formatted) policies, as there is less overall bloat.

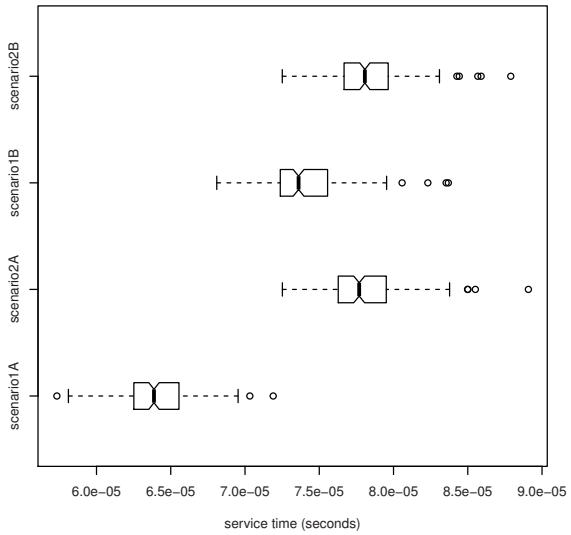



Figure 4.14: Service times for Scenarios 1A, 1B, 2A, 2B

The boxplot in Figure 4.14 indicates that the best performance is obtained when optimized (JSONPL) policies and requests are used (Scenario 1A) and that performance degrades as bloated/more complex automatically translated JSON is used to represent polices and requests (Scenario 2B).

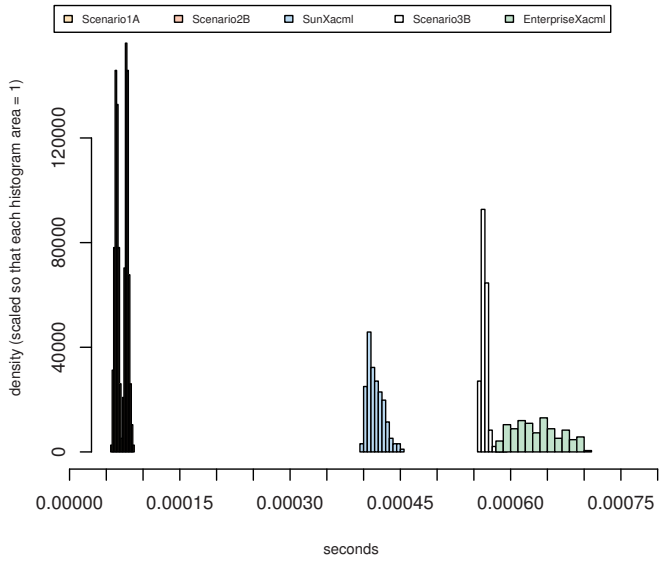



Figure 4.15: Service time comparison. Ranked in decreasing order of performance (left to right in the figure above), they are: njsrPDP Scenario 1A, 2B; SunXACML; njsrPDP Scenario 3B, EnterpriseXACML.

The service time histograms in Figure 4.15 show how different njsrPDP scenarios compare with "traditional" PDP implementations. Clearly there is no net performance benefit of the JSON implementation when requests are translated on the fly: mean services times for njsrPDP Scenario 3A are greater than those for SunXACML, but njsrPDP has much greater performance potential (see Scenario 1A).

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| scenario | 5 | 6.4e-05 | 1.27e-05 | 7.43e+05 | $< \varepsilon$ |
| decision | 2 | 1.0e-09 | 4.00e-10 | 2.54e+01 | 1.8e-11 |
| requestIndex | 190 | 8.0e-09 | 0.00e+00 | 2.32e+00 | $< \varepsilon$ |
| Residuals | 954 | 1.6e-08 | 0.00e+00 | | |

Table 4.6: Analysis of Variance for Scenario service times

| | | | |
|---|---|---|---|
| *1A* | 6.4e-05 | *1B* | 7.4e-05 |
| *2A* | 7.8e-05 | *2B* | 7.8e-05 |
| *3A* | 57.8e-05 | *3B* | 56.5e-05 |

Table 4.7: Mean service times for each of the Scenarios

Table 4.6 confirms that factors such as scenario type, decision and request type are significant variables when modeling service times. The comparison of mean service times for each Scenario in Table 4.7 shows how different Scenario 3 is to the others, and how much request format optimization affects performance (compare Scenario 1A and 2A).

## 4.4 Scalable Group Management

The CoffeeScript language (MacCaw (2011)), which transcompiles to JavaScript offers syntactic advantages over JS. The code produced by CoffeeScript is invariably cleaner, more direct and produces highly optimized JavaScript. The JavaScript PDP with JSONPL inputs offered a sizeable increase in terms of performance over rival PDPs. Further optimizing this process would bring the Management potential for groups well within near real time requirements.

#### 4.4.0.1 CoffeeScript PDP

Designing a PDP in CoffeeScript brings a number of advantages over a JavaScript implementation.

**A more intuitive Class system:** While Classes are possible within JavaScript they are complex to establish and manage. CoffeeScript masks this functionality with exposed keywords, lending a PDP system class based features such as inheritance and a more intuitive and fluid design.

**Scope handling:** CoffeeScript takes care of properly scoping your variables, all CoffeeScript output is wrapped inside an anonymous function, ensuring global variables will not leak.

**Reduced Developer Effort:** A CoffeeScript implementation will deliver a semantically equivalent solution at a fraction of the code base size and developer effort.

**Domain readable code:** The language used within CoffeeScript is much more human readable to the point it is self documenting code.

In Appendix A a CoffeeScript PDP is created. This PDP is semantically the same as the njsrPDP created within JavaScript and used in the previous experiments. The PDP is 200 lines long and 6,168 bytes in size, this compares to the njsrPDP which is 1020 lines long and 21,821 bytes. While code size and length are not significant factors, they do highlight the reduced development effort and the potential portability of the produced code as discussed in Section 4.1.2.

#### 4.4.0.2 CoffeeScript Policy Representation

The scenarios highlighted in Section 4.3.6 showed that a complementing policy representation format and PDP can yield powerful results. Representing policies within traditional representation formats still requires a translation to occur for the host language to interpret them. In the case of XML, this is a heavy process requiring external libraries[1]. In the case of JSON, this process is simpler with an appropriate host language, and very much measurable with respect to the size of the document. In the case of the scenarios considered, a JSON document was taking 8$\mu$s to process. This thesis considered CoffeeScript as a potential representation format for requests and policies for the following reasons:

**Human Readable:** The syntax of JSON is brackets which need to be nested correctly, this can be difficult to troubleshoot.

**Directly interpretable within a CoffeeScript PDP:** No translation is required, once the request is received in the system the PDP has full visibility of all elements contained within it.

**Turing Complete** Functions and algorithms could be encoded within the request or policy documents. As CoffeeScript is a *Turing complete* language, any computationally feasible algorithm can be expressed within it (Turing (1936-7)[2]).

A sample request in Listing 4.6 shows the structure and syntax that a CoffeeScript Request could take.

```
class Request
  @subject:
    category: "access-subject"
```

[1] For example dom4j (http://dom4j.sourceforge.net/) within Java or Node-XML within JavaScript (https://github.com/robrighter/node-xml/).

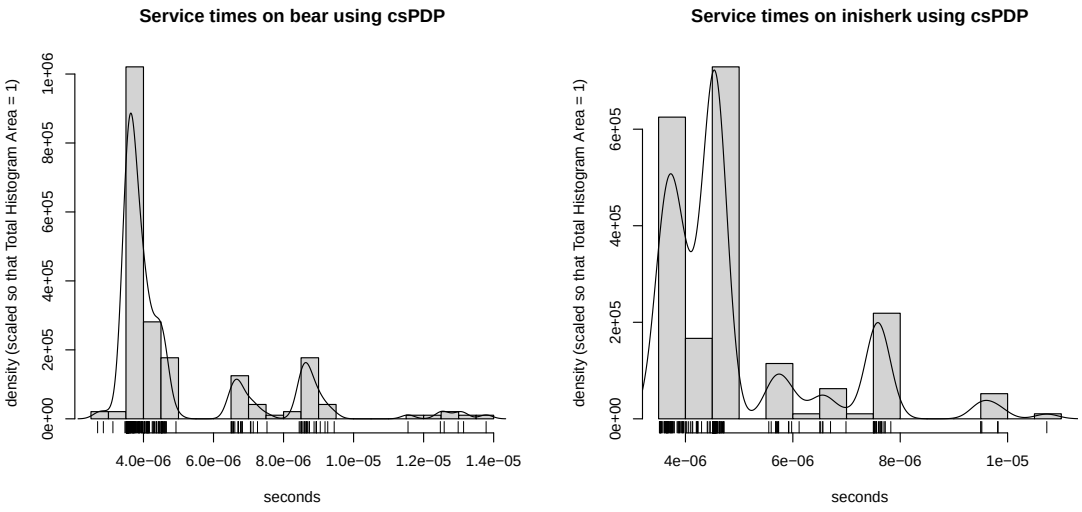
[2] This paper was the initial research carried out around the problem. For a more relevant and historical viewpoint the work of Petzold (2008) is recommended.

```
    isConflicted : true

  @resource:
    class: "paper_review_content_commentsAll_rc"
    id: "DEFAULT RESOURCE"

  @action:
    type: "create"

exports.Request = Request
```

Listing 4.6: CoffeeScript Policy Request

### 4.4.1 Analysing CoffeeScripts Performance

The CoffeeScript PDP (csPDP) and request combinations were run in the same manner as the njsrPDP simulations carried out earlier. The same assurances and simulation setups were used and carried out on identical machines.




(a) CSPDP on bear

(b) CSPDP on inisherk

Figure 4.16: CSPDP request service times on hosts `bear` and `inisherk`.

Figure 4.16 shows the average request service times. The service times are clustered with three visible clusters identifiable in Figure 4.16(a) on host bear and similarly three clusters in Figure 4.16(b) with some overlap visible in a compressed manner. The *host* machine factor is not as significant within the CSPDP results showing that the same

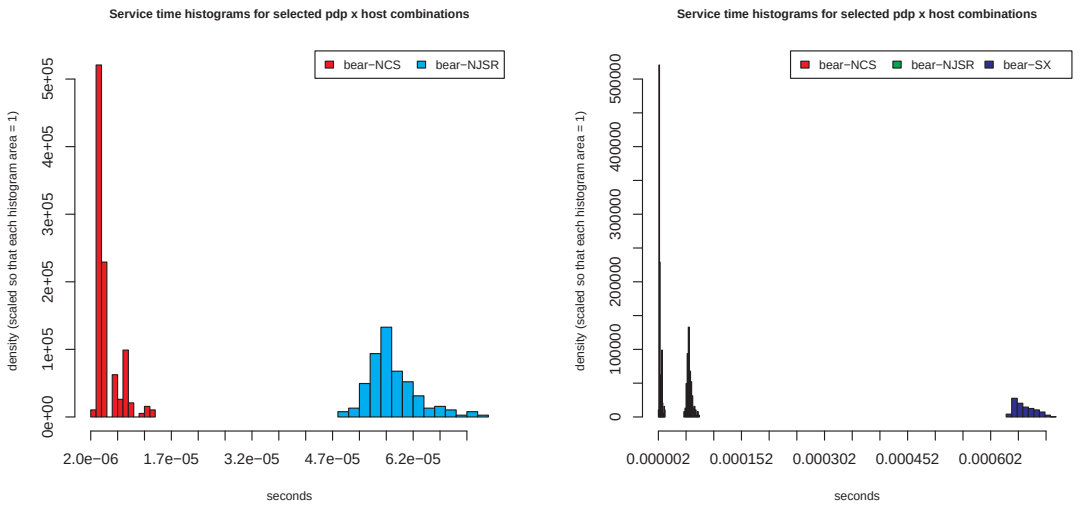
performance profile is possible regardless of server type, making the results repeatable. From a *scale up* point of view this repeatability and predictable grouping of service requests makes response time provisioning easier to facilitate.

Table 4.8 shows the mean service times for all PDPs under investigation in this section.

| | |
|---|---|
| *csPDP* | 4.95 e-06 |
| *njsrPDP* | 4.66 e-05 |
| *SunXacmlPDP* | 5.35 e-05 |
| *EnterpriseXacmlPDP* | 8.33 e-04 |

Table 4.8: Mean service times for each of the PDPs

Figure 4.17 shows a comparison between csPDP and its peers.




(a) CSPDP on bear

(b) csPDP compared with njsrPDP and SunXACML

Figure 4.17: csPDP request service times on hosts `bear` compared with other PDP implementations

The csPDP has a clear performance increase over the njsrPDP, visible in Figure 4.17(a). With both platforms based on Node.js, the performance increase is attributed to a more optimised JavaScript and the representation format used. Having the policy and request accessible without an interpretation step, as is the case with JSON, offers a measurable performance increase, with the interpretation time of a JSON document typically static with respect to the size of the input document. Additionally, using class

based data structures that can be directly accessed, removes an element of searching and looping that would be required in a traditional representation format. In terms of improving the njsrPDP to bring the performance closer to csPDP, the PDP code base could be further optimised and the representation of data within the JSONPL documents improved upon. This would yield minor performance improvements as a significant performance improvement is not possible with traditional data representation formats. Figure 4.17(b) shows the service times of the SunXacmlPDP with respect to the Node.js based PDPs. In this instance, Node as a deployment platform is compared to a non optimsed JVM. There is some scope for improving the JVM, with a deeper understanding of the semantics contained within the JVM (Belblidia & Debbabi (2007)), various approaches can be taken such as tackling memory management (Velasco *et al.* (2012)) or how methods are dispatched (Takeuchi *et al.* (2012)). Optimzing the JVM for Access Control requests should bring the response times down, but nowhere near the level of the Node.js based execution platforms.

### 4.4.2 Summary

Access Control management in a *near real time* manner is currently difficult to provision and was identified in RQ-GFM2. A change in implementation technology and representation format, while retaining direct mapping to XACML encoded policies and requests is possible. No performance gain emerges from this, however, removing the verbose metadata which accompanies XML to produce a concise representation shows a significant improvement. Further research has shown that removing the dependency on XACML semantics and encoding policy requests and documents directly within a programming language can yield response times not capable of being reproduced in traditional XACML. The cost is interoperability but the result is a policy language representation capable of scaling beyond current state of the art and capable of managing high volume requests and potentially emerging usage patterns (RQ-GFM1). As XACML is currently not deployed within social networks, abstracting the principles and information flow behind the access control mechanism can inform an access control deployment. A resulting access control implementation that is complementary to emerging social network architectures can thus be architected, something which is an achievement of this thesis.

## 4.5 Summary

This section presented a discussion on the performance characteristics of managing Group Interactions within emerging social networks. A contribution is presented in the form of a discussion and design surrounding a case for Group Specific PDPs implemented within CoffeeScript. The current state of the art was shown to not satisfy the real time requirements required within emerging scenarios and solutions to RQ-GFM2, RQ-GFM3 and RQ-S3 were presented. The understanding of this Management potential and Policy structure will be utilised within the next section, assisting in the design of an output model.

# Chapter 5

# Towards a Unified Model for Group Formation and Interaction Management

This section presents the model abstracted from the experimental work carried out in the previous sections. A unified model is evolved, informed by experiences in analysing and implementing the range of scenarios discussed in Chapter 3 and Chapter 4. Additionally, further refinements of selected algorithms are presented in CoffeeScript.. These algorithms support and illustrate the behavior of the evolved model. CoffeeScript is chosen as a representation format, as it usefully embodies Domain Specific Language capabilities and features.

## 5.1 Model Responsibilities

A conceptual model, visible in Figure 5.1, encompassing Group Formation, Roster, Management and Policy domains, serves as a useful starting point for elaborating and exploring the model further. Each domain within this model is examined individually, with focus on core responsibilities, relationships and key algorithms.

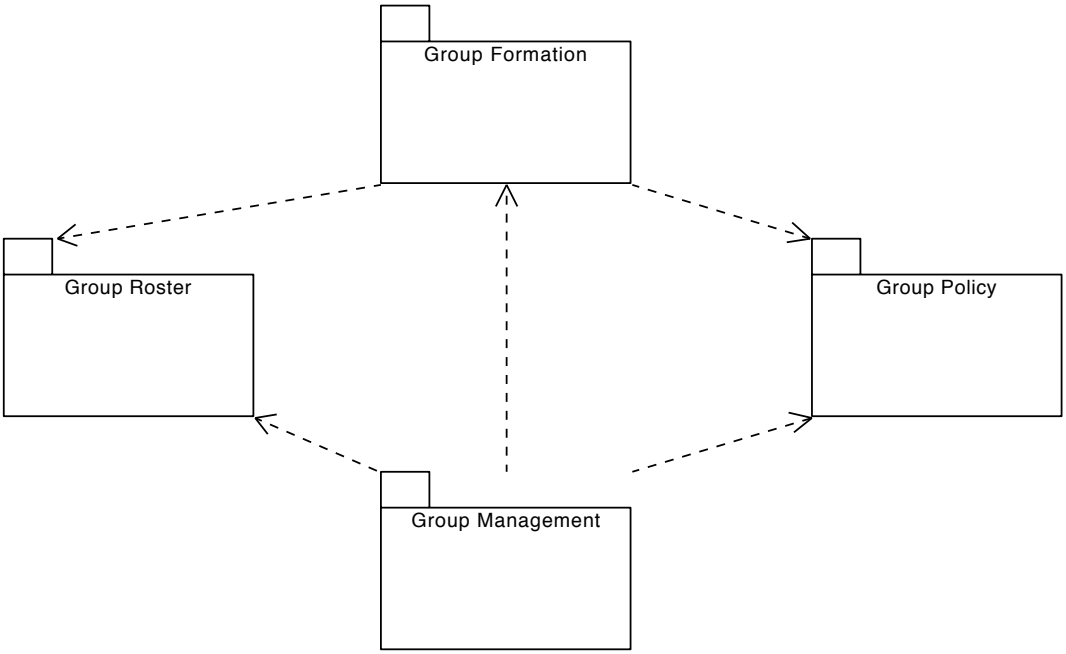



Figure 5.1: Abstract Group Formation and Management Model

### 5.1.1 Group Formation

Group Formation is the public facing interface allowing users to request groups and receive requests to query existing groups. Figure 5.2 highlights the elements of the model and the relationships they contain. This model was arrived at from the analysis carried out in Chapter 3 on the formation of groups within the XMPP domain and in the domain of Disaster Management.

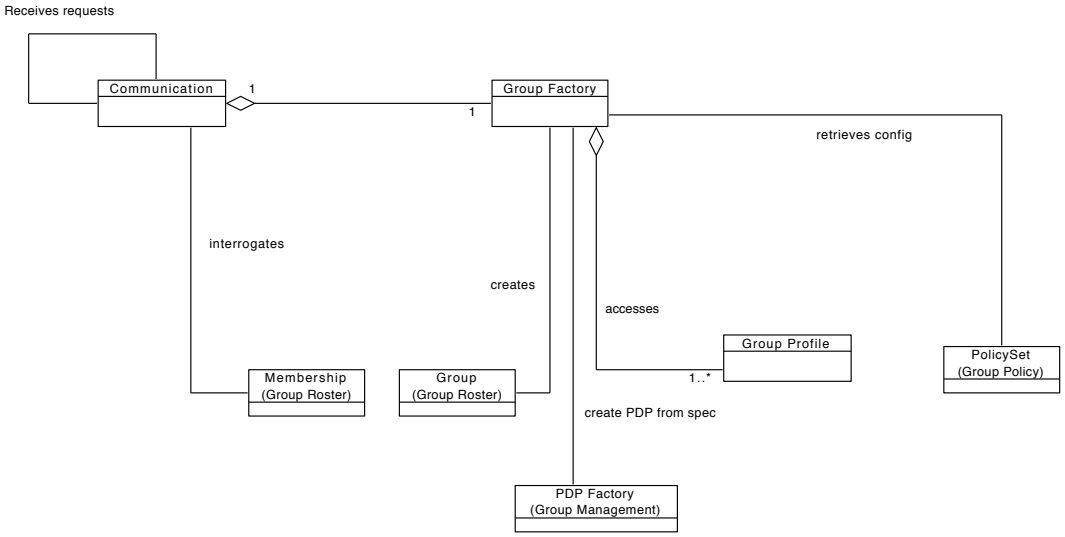



Figure 5.2: Group Formation Model

The core elements of the model are described as follows:

**Communicator** : The Communicator assumes the role of the public interface to the system. Requests are received including membership requests and group membership queries, both forwarded to the Group Rosters Membership entity. Requests to form a new group are received here also and sent to the Group Factory for realisation.

**Group Factory** : The Group Factory is responsible for interpreting the request to form a group and subsequently the formation of that group. This element interacts with a Group Profile, retrieving a candidate profile containing references to the associated policy configuration of PDP and PolicySet. This configuration reference is subsequently validated by the Group Policy entity. The Group Factory then writes the newly created Group to the Group Roster. The Group Factory concludes by passing the PDP specification, retrieved in the validation of the PolicySet, and passes it into the PDP Factory, a Group Management component.

**Group Profile** : A Group Profile is a schematic representation of a Group which can be used to infer a particular style for a group. By having a profile the underlying management system can better provision resources and customise the management principles associated with the group. The profile includes a reference to an appropriate PolicySet, resource recommendation and group specific structures

such as hierarchies and role specification.

The associated sequence diagram, visible in 5.3, describes the interaction of the Formation mode.

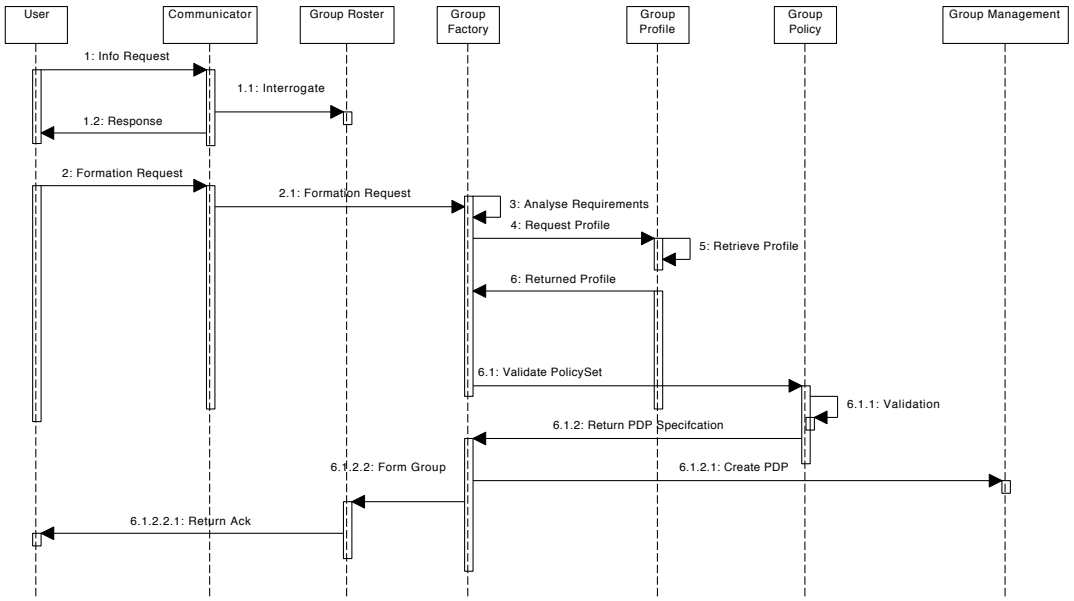



Figure 5.3: Group Formation Sequence Diagram

#### 5.1.1.1 Group Formation Algorithms

The Group Formation Algorithm, encoded within CoffeeScript and visible in Listing 5.1, represents the three main actions of receiving a request, querying the roster for information and forming a group. For convenience, Appendix B contains a reference guide for the terminology and syntax used within CoffeeScript

```
receiveRequest: (context) ->
  queryRoster(context) if context query?
  formGroup(context) if context formation?

formGroup(paramaters,ack) ->
  retrieveProfile(paramaters.requirements, profile) ->
   retrievePolicyConfiguration(profile.policyConfig, policyBundle) ->
    groupManagement.injectPDP(policyBundle.PDP, paramaters.groupName)
    Roster.createGroup(profile.resources, paramaters.groupName,
        paramaters.owner, policyBundle.policySet)
    ack("Group successfully created")
```

```
queryRoster(query.response) ->
  interpretRequest(query) ->
    Roster.retrieveInformation(query, data) ->
      response(data)
```

Listing 5.1: Group Formation Algorithm

The algorithm in Listing 5.2 shows a generic Group Profile and two sample profiles for Dynamic Groups and for Ad-Hoc Groups.

```
class Profile
 constructor: (@name, @visibility = "public") ->
 maxSize: (@maxSize) ->
 minSize: (@minSize) ->
 membershipBarrier: (logic) -> @barrier = logic
 numAdministrators: (@numAdmins) ->
 containsServices: (logic) -> @services = logic
 requiresResources: (logic) -> @resources = logic
 fileSharing: (logic) -> @les = logic
 PolicyConfig: (bundle) ->
   @PDP = bundle.PDP
   @PolicySet = bundle.PolicySet

class Dynamic extends Profile
 super("football")
 membershipBarrier:(value = no) -> super value
 containsServices: (value = no) -> super value
 requiresResources: (value = yes) -> super value
 fileSharing: (value = no) -> super value
 bundle.PDP = "dynamic sport"
 bundle.PolicySet = "dynamic sport"

class AdHoc extends Profile
 super("rugby","private")
 membershipBarrier:(value = no) -> super value
 containsServices: (value = yes) -> super value
 requiresResources: (value = no) -> super value
 fileSharing: (value = yes) -> super value
 bundle.PDP = "private adhoc sport"
 bundle.PolicySet = "private adhoc sport"
```

Listing 5.2: Group Profile Algorithms

### 5.1.2 Group Management

The design of the model is inspired by the performance profiles associated with the management principles derived in Chapter 4. It presents a means to manage the requests and activities associated with high traffic modern groups, with custom logic implemented per group. Figure 5.4 describes the overall architecture.

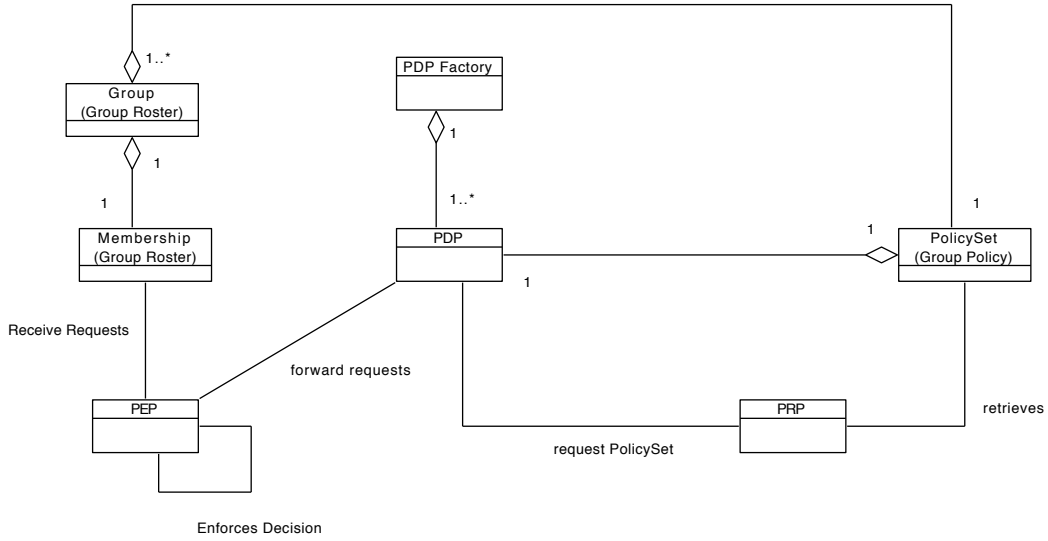



Figure 5.4: Group Management Model

**PEP** : The Policy Enforcement Point has a number of responsibilities within the Management system. Firstly, the retrieval of requests from within the membership of the group. These requests are interpreted and evaluated before being passed onto a specific PDP for which the request is targeted.. On receipt of an evaluation decision from the PDP, the PEP is charged with the responsibility of enforcing the decision made by the PDP, informing the original requester of the decision.

**PRP** : The Policy Retrieval Point is responsible for retrieving a PolicySet or aspects of a PolicySet from the Group Roster and passing it to the PDP for evaluation.

**PDP Factory** : The PDP Factory allows management and administration decisions to be taken across a selection of PDPs. The Factory additionally functions as a point where new PDPs can be created, taking the required specification and instantiating a fully functioning PDP.

**PDP** The PDP is tailored at a language level to manage a specific group. A tailored PDP maps one to one with a group within the system providing an evaluation engine to make an informed decision on whether an action within the group is allowed. Retrieving a message from the PEP, the PDP interfaces with the PRP to retrieve the required PolicySet, or aspects of it, and makes an evaluation with the data at hand. A decision is returned to the PEP.

Figure 5.5 shows a Sequence Diagram highlighting the interactions and message passing among the core elements of the Management System.

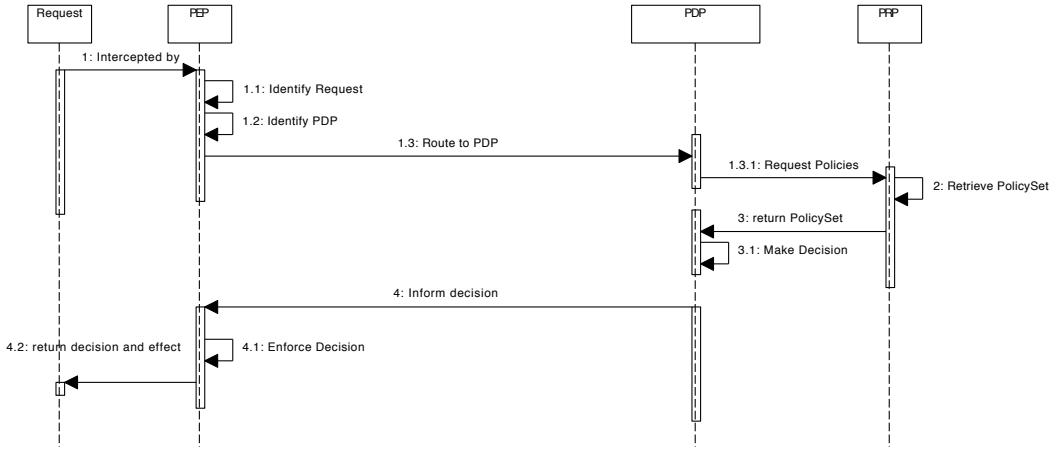



Figure 5.5: Group Management Sequence Diagram

#### 5.1.2.1 Group Management Algorithms

The Group Policy Algorithm includes the PEP and the PRP within Listing 5.3. An extensive example of a PDP can be found in Listing A.1 within Appendix A. The PRP has a function to retrieve the PolicySet for the group that the request is associated with. The policySet is stored on the Roster with the group, which may be a database or in memory storage.

```
class PEP
 receiveRequest (request, decision) ->
   interpret(request, interpretedRequest) ->
     if interpretedRequest isnt "NotApplicable"
      interpretedRequest.group.PDP.evaluate(interpretedRequest, result)
          ->
```

```
        obligations(result.obligations)
        decision(result.decision)

class PRP
onRequest = (request, callback) ->
  policySet = GroupRoster.retrieveGroup(request.group).policySet
  if request.type is "fullSet" then callback(policySet)
  else
    callback(parsePolicySet(request.paramaters))

class PDPFactory
 activePDPs = []
 createPDP (name, PDPspec) ->
   configurePDP(PDPspec)
   activePDPs.push([name,PDP])
```

Listing 5.3: Group Management Algorithm

### 5.1.3 Group Policy

The Group Policy component is represented in Figure 5.6 and shows the core entities making up the model. The model is influenced by the structure of the XACML specification for policies, available in Appendix C, and by the experimentation on the performance of access control within Chapter 4. The simplified version of the access control implementations used within the experiments carried out shows that additional metadata surrounding the policy structure is not required. The simplified structure presented facilitates both domain readable code and faster execution by specifying the minimal information required.

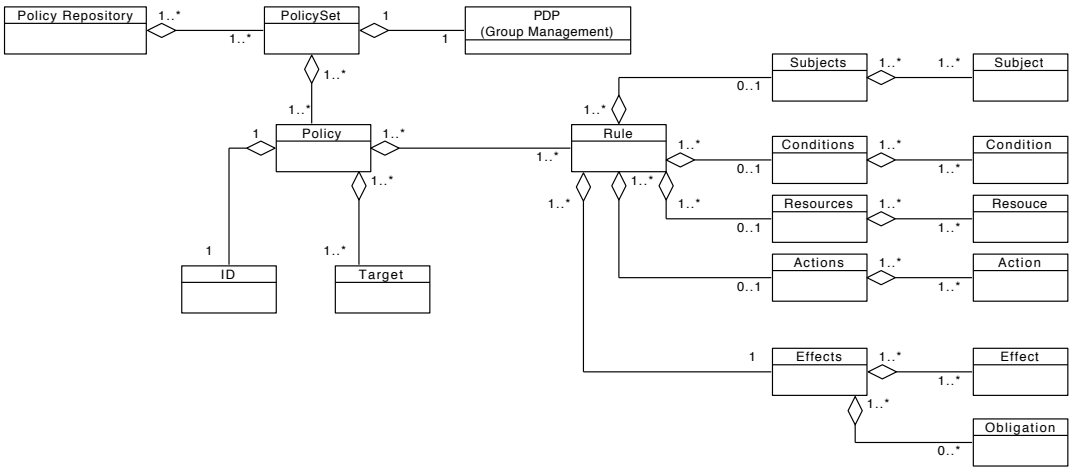



Figure 5.6: Group Policy Model

The following entities are described:

**Policy Repository** The overall structure, typically a data storage mechanism, is the Policy Repository. This contains several PolicySets.

**PolicySet** The PolicySet represents a grouping for policies. A PolicySet is made up of multiple individual Policy elements and a specific PDP tailored towards the PolicySet.

**Policy** : A policy is a collection of rules, associated with a specific target.

**ID** : A unique identifier to identify the particular policy

**Target** : The target resource or action that the policy is designed to protect.

**Rule** : A Rule is a combination of elements, optionally containing a mix of Subjects, Conditions, Resources and Actions.

**Subjects** represents a set of individual Subjects with whom the rule is targeting. For a successful deliberation the subject or subjects presented in the interaction request must match those contained within the Rule.

**Conditions** are logical gateways protecting the overall rule. Typically if a condition or conditions are met it satisfies the constraints placed on the firing of the rule, provided the Subjects, Resources and Actions also match.

**Resources** are the items, be they physical or virtual, that the Rule is designed to protect.

**Actions** are the intended interactions with which a Subject wishes to invoke on a Resource, provided the right Condition is met.

**Effects** represent the answer and optional obligations which the PDP must deliver based on the Rule being evaluated with respect to the Subjects, Conditions, Resources and Actions. Effects are typically *Permit* or *Deny*.

**Obligations** are additional conditions that the policy authors wish to impose on the interaction that has been approved (or denied).

#### 5.1.3.1 Group Policy

Group Policies are specified by the generic Policy outlined in Listing 5.4. The individual policy examples also specified in Listing 5.4 represent the PolicySet described for the experiments in Section 4.4.

```
class Policy
 constructor: (@id, @mastertarget) ->
 Subject: (@subjects...) ->
 Resource: (@resources...) ->
 Action: (@actions...) ->
 Effect: (@effects) ->

RPSlist000 = new Policy("RPSlist.0.0.0","conference_rc");
RPSlist000.Subject(["role", "admin"])
RPSlist000.Action("read","write")
RPSlist000.Effect("permit")

RPSlist001 = new Policy("RPSlist.0.0.1","conference_rc")
RPSlist001.Subject(["role","pc-chair"])
RPSlist001.Action("read")
RPSlist001.Effect("permit")

RPSlist002 = new Policy("RPSlist.0.0.2","conference_rc")
RPSlist002.Subject(["role","pc-member"],["isMeeting", true])
RPSlist002.Action("read")
RPSlist002.Effect("permit")

RPSlist003 = new Policy("RPSlist.0.0.3","conference_rc")
RPSlist003.Effect("deny")
```

```
conference_rc = [RPSlist000,RPSlist001,RPSlist002,RPSlist003]
policySet = [conference_rc]
```

Listing 5.4: Group Policy Algorithm

For completeness, Listing 5.5 shows a number of sample requests, highlighting the structure associated with an access control request.

```
class Request
  @subject:
    category: "access-subject"
    role : "pc-chair"
  @resource:
    isPending: false
    id: "abc123"
  @action:
    type: "write"

class Request
  @subject:
    category: "access-subject"
    subjReviewsThisResPaper : true
  @resource:
    class: "paper_rc"
    id: "abc1234"
  @action:
    type: "delete"

 class Request
  @subject:
    category: "access-subject"
    isMeeting : true
  @resource:
    class: "paper_review_content_commentsAll_rc"
    id: "abc12345"
  @action:
    type: "read"
```

Listing 5.5: Group Policy Requests

### 5.1.4 Group Roster

The Group Roster is represented in two models, Figure 5.7 shows a minimal Group Roster and Figure 5.8 shows a Group Roster with a Social Network modeled. Analysis

of the XMPP roster in Chapter 3 motivated the design of this model.

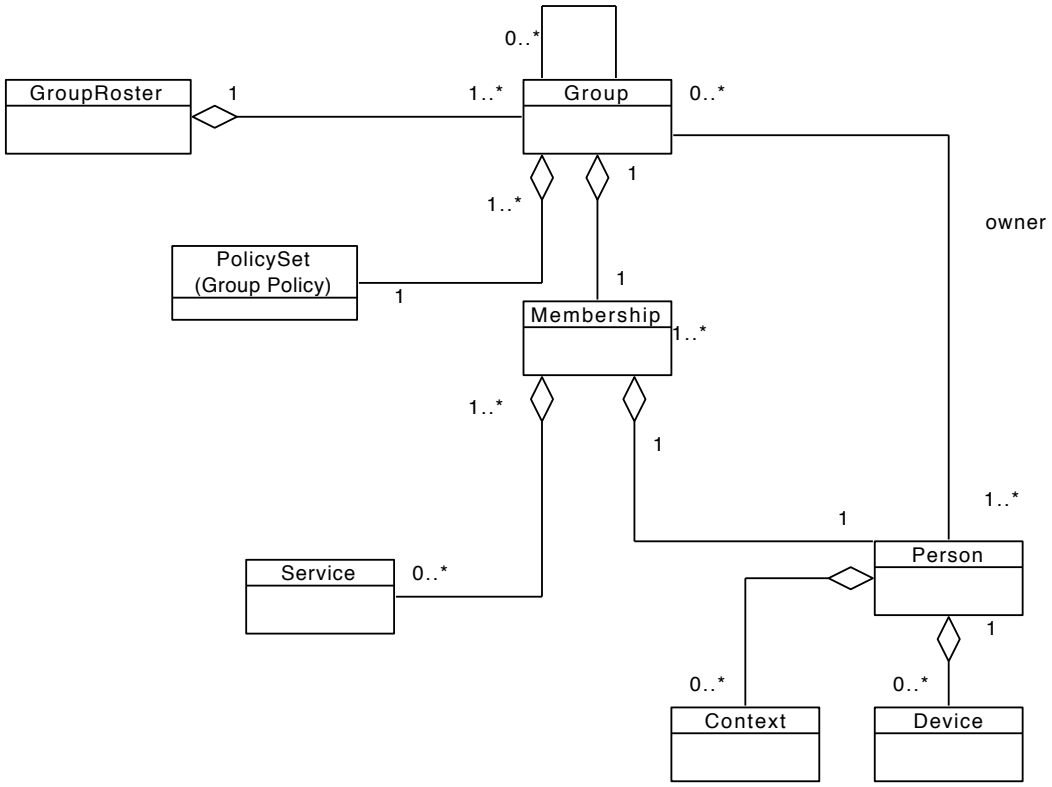



Figure 5.7: Group Roster Model

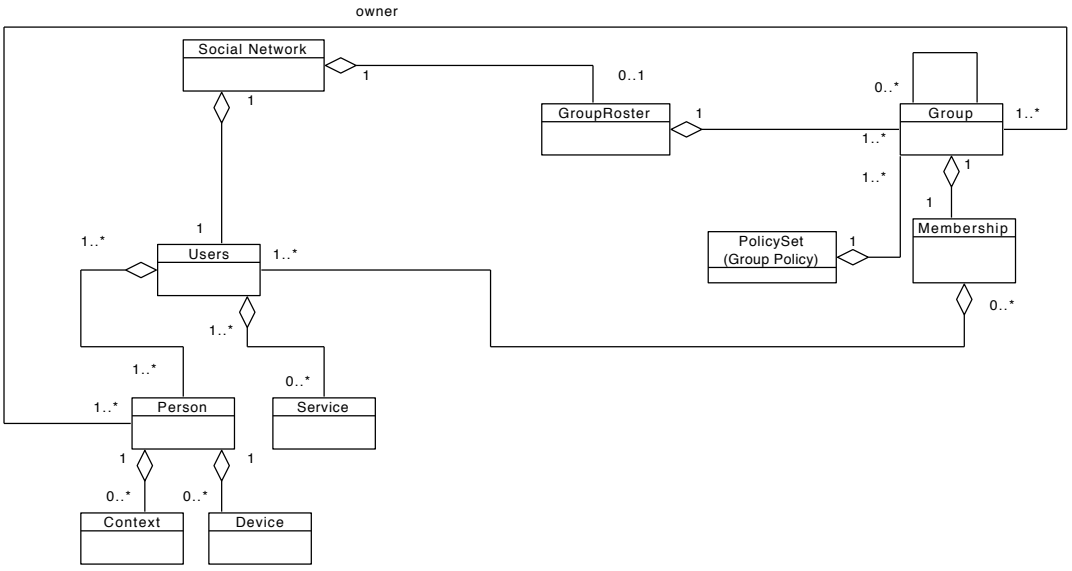



Figure 5.8: Extended Group Roster Model

The core elements of Figure 5.7 are described below:

**Group Roster** : The Group Roster is a data set which contains the list of all Groups within the system.

**Group** : A Group is a logical representation of a community of interest containing the membership set and the associated PolicySet to govern the group. Groups can contain subgroups.

**Membership** : The membership of the Group including metadata about the members is stored as an internal Group Roster.

**Service** : A Service is a non living entity that is classed as a member of the group, and thus accessible and available for interaction.

**Person** : A Person is a living entity who claims membership of the Group. At least one person is classified as the owner of the Group.

**Context** : Additional information is available about the Person including contextual information that they may wish to release.

**Device** : The device or devices associated with the Person from which they typically access the group.

Figure 5.8 has the same definition and role identity as the above sections, with the addition of two elements.

**Social Network** : A Social Network represents the fact that a Group Formation and Management System may hook into existing networks.

**Users** : The userbase of a Social Network, which may contain People or Services. The Membership of the Group takes a subset of the Users as its membership.

Figure 5.9 shows a Sequence Diagram highlighting the interactions and message passing among the core elements of the Roster for a group addition.

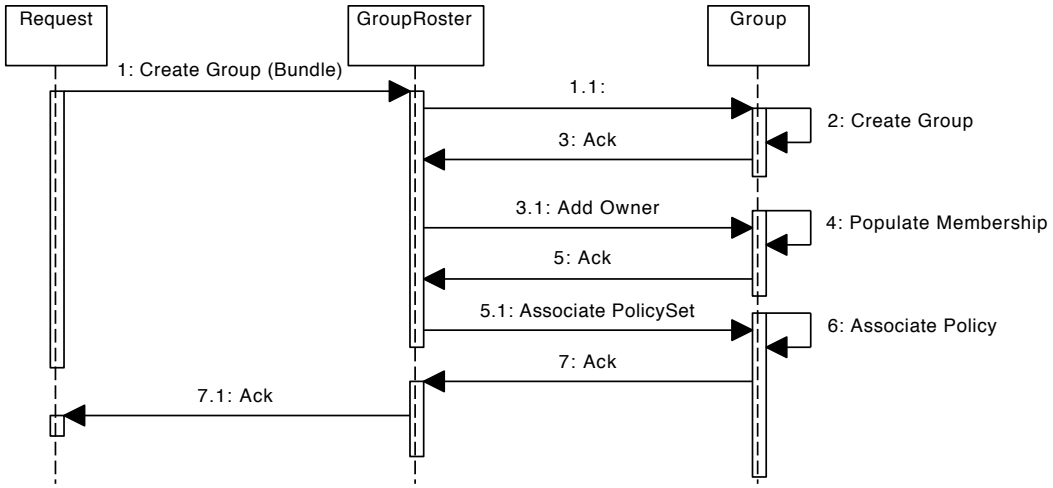



Figure 5.9: Group Roster Sequence Diagram

#### 5.1.4.1 Group Roster

The Group Roster is represented in Listing 5.6. Included is the specification of a Group.

```
class Roster
 groups = [Group...]
 retrieveInformation (query,response) ->
 retrieveGroup(groupName) ->
 createGroup (resources, name, owner, policySet) ->
   group = new Group(name,owner)
   group.membership.push(owner)
   group.setPolicySet(policySet)
```

```
    group.setResources(resources)
    groups.push(group)

class Group
 constructor: (@name,@owner) ->
 @membership = []
 setPolicySet: (input) -> @policySet = input
 setResources: (input) -> @resources = input
```

Listing 5.6: Group Roster Algorithm

### 5.1.5 Combined UML Model

The individual models outlined in the previous sections are combined and shown in Figure 5.10[1]. Additionally, Figure 5.11 shows the same diagram with a package overlay. Combining the model clarifies the relationships among the entities and may help identify bottlenecks within potential deployments. Section 5.2 outlines some implementation recommendations to accompany the model. These recommendations primarily address suitable deployment platforms, storage solutions, presentation formats and strategies for programmatically handle concurrency. Taken in their entirety, the recommendations complement one another, proving a *friction-free* environment for handling the formation and management of groups and their interactions.

[1]Both Figure 5.10 and Figure 5.11 are presented in landscape format for readability purposes

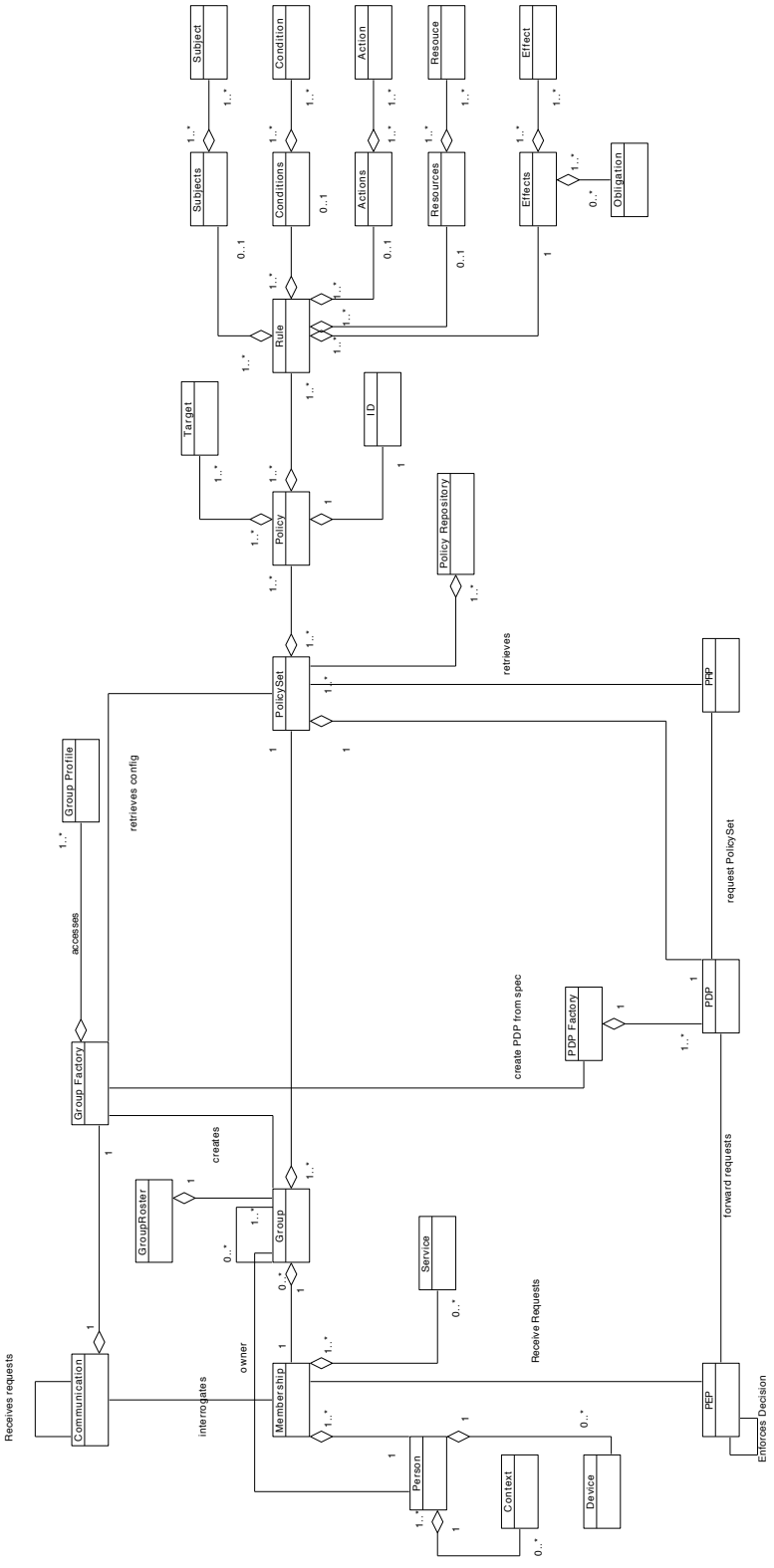



Figure 5.10: Group Formation and Management Model Combined

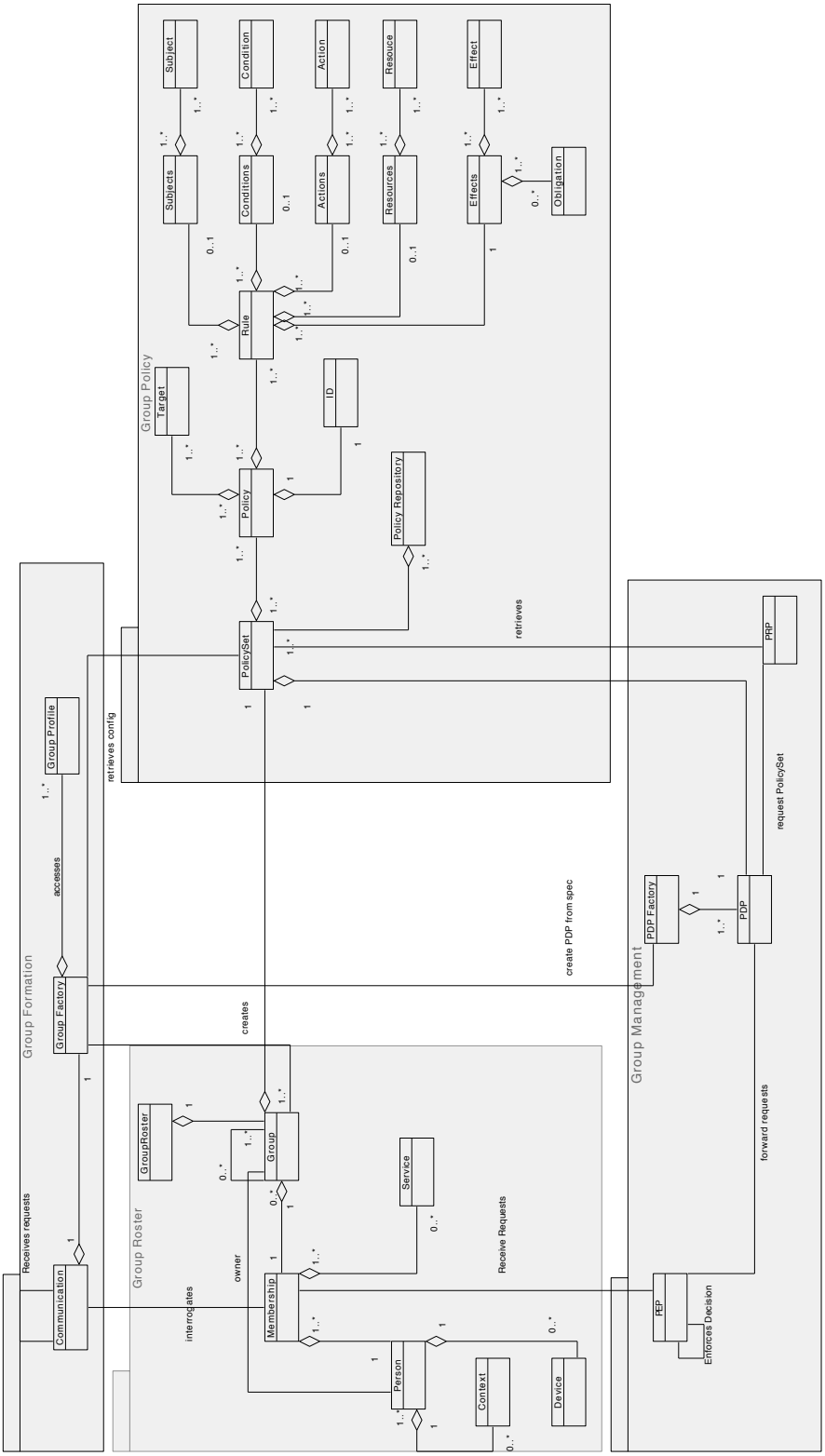



Figure 5.11: Group Formation and Management Model Combined with Packages

## 5.2 Implementation Recommendations

The following implementation recommendations to help realise the above models were derived from extensive simulations and observations. They include a discussion on handling concurrency with language recommendations that can help attain scalability and improve response times within the domain of group based interactions in emerging social networks. A section on recommended architectures including both platform and backend systems is also presented. Representing the management domain in an effective language and manner is also very important to the overall usefulness of managing high volume requests within the group environment. Candidate representations are recommended based on the investigations carried out and on future uses of the system.

### 5.2.1 Handling Concurrency: Language Level

Diverse approaches to programmatically coping with concurrency have long been a source of contention among software developers. This thesis has shown that within the context of emerging social networks, a Non-Blocking I/O approach to concurrency can facilitate more responsive and scalable services. This thesis used two Non-Blocking languages, JavaScript and CoffeeScript, to highlight how they can facilitate the requirements outlined within the current state of the art, as described within the Research Questions. In the future, new approaches to programmatically handling concurrency will emerge. These approaches will be challenged by the volume of requests and additionally the potential global membership base that groups will contain. The contributions of this thesis to the understanding of the current challenges facing groups can inform future approaches within this domain.

### 5.2.2 Underlying Platform

With the Node.js framework, JavaScript is no longer just a language supporting user interaction within browsers (client-side). Based on the Google initiated, open source V8 JavaScript engine, JavaScript, or languages that can compile down into JavaScript, can be be compiled into highly optimised server-side machine code on the fly. The Non-Blocking nature of JavaScript is present within Node.js with all requests gradually executed in sequence through the usage of callbacks. Node.js allows an elegant solution to be engineered for traditional scalability problems and an alternative approach for domains that might benefit from a Non-Blocking I/O approach. Node.js provides a number of advantages that benefit the design of management system.

- It supports better modular design through component based programming, benefiting from specifications supporting interoperability between server modules and even between server and client side modules
- The system design is extensible with the possibility of swapping out modules and including modules created in the future.
- The non-blocking architecture facilitated by JavaScript callbacks makes more efficient use of CPU resources than traditional systems that sit idle while a complex IO request is executing. In a Node.js deployment, the spare CPU cycles can be used more productively, e.g., to handle the next request or prepare the response.
- It uses events to trigger callback execution. A benefit of the event model is a publish/subscribe mechanism built into the environment. This promotes greater flexibility, since the system can include listeners (handlers) for particular events of interest.
- The absence of complicated blocking semantics simplies development.

Competing platforms have emerged within the Non-Blocking domain including Vert.x[1] which offers support for multiple languages including JavaScript, Ruby, Python, Groovy and Java. Aspects of the experiments carried out within this thesis could be realised within Vert.x and may yield performance improvements due to the multicore[2] support within Vert.x. However, at the time the experiments within this thesis were carried out, the Vert.x platform and supporting ecosystem was not in a mature enough state to investigate the scale of experiments under consideration. The technology is currently at a point where it could be considered a viable candidate platform for an implementation of the model produced as part of this thesis.

### 5.2.3 Storage Solutions

The dominance of the relational database is no longer a given. The NoSQL movement is gathering pace with many open implementations of this broader, and perhaps more scalable architecture for the data store (MongoDB, CouchDB, Redis). More highly capable and intelligent systems can be constructed at a fraction of the cost of traditional

[1] Vert.x is an asynchronous application development platform for modern web applications. http://vertx.io/

[2] Node.js cannot officially support multiple processors. However the open source community have successfully demonstrated this is possible, for example Multi-Node (https://github.com/kriszyp/multi-node). This functionality is slated for future integration within the official Node.js release.

relational systems. Costing aside, the style of database can often map nicely to the problem at hand. Redis, for example, is a data structure oriented database that directly supports sets, lists and hashes. Having a storage mechanism that host language data structures understands not only removes the mental disconnect of the developer but also reduces the amount of code required to facilitate parsing data from storage. In domains where near real time execution is a requirement, minimsing the translation time of accessing and inferring data is critical.

### 5.2.4 Policy Component Representation

A case was put forward for moving traditional access control based policy representations away from the XML style that is so closely associated with the XACML specification. Attempts to use JSON and CoffeeScript as policy representation formats showed that a performance gain could be achieved while used with a complementing architecture. This thesis recommends CoffeeScript as a policy representation format to encode both policy documents and requests catering for both access control policies and SLA policies. As a representation format, CoffeeScript has all the necessary semantics to encode the constraints of a policy language that would traditionally be represented within XML or JSON. CoffeeScript can use object literals in the same form as JSON, representing name:value pairs and arrays within its syntax. Anything that is possible to represent within traditional policy documents is capable of being represented within CoffeeScript with additional context. In fact, CoffeeScript can be viewed as enhancing the object literal syntax of JSON with richer types, such as string supplemented with booleans and numerical types.

Using CoffeeScript the resulting compiled code is JavaScript, allowing execution within any JavaScript interpreter, essentially facilitating a portable representation. A limitation, not investigated within the simulations carried out of the JSONPL language is that policies and requests expressed in JSON are limited to the JSON specification. Indeed this is also true for XML based policies. As such, JSON and XML as representation formats cannot directly contain functions or algorithms directly encoded within the policy language. CoffeeScript is a programming language and as such is *Turing Complete*. Any computationally feasible algorithm can be expressed within it and thus can be encoded as a policy. Such a capability would provide more fine grained management, for example within SLA management from a service providers perspective. Take the calculation to check for the probability of an SLA violation using a very optimistic technique which assumes no penalties, as highlighted in Algorithm 5.1.

**Algorithm 5.1:** Probability of an SLA violation

$Pr(Violation) = Pr(\overline{RT} > SLA(max(RT))$

Where $\overline{RT}$ is the mean response time. Listing 5.7 shows how this algorithm could be represented within CoffeeScript.

```
calculateSLAViolation: (requests) ->
  averageRT = mean requests
  if averageRT >= maxSLA_RT then yes else no

mean = (array) ->
return 0 if array.length is 0
 sum = array.reduce(s,i,0) -> s += i
 sum / array.length
```

Listing 5.7: RT check for SLA Violations

By calculating the number of requests by the probability that they don't violate an SLA threshold, the ability to decide which services to prioritise based on their economic value is possible. Algorithm 5.2 shows a naive Max Economic Value algorithm for Service $s$ in time $t$. A sample implementation is visible in Listing 5.8.

**Algorithm 5.2:** Max Economic Value

$\forall$ s $\in$ ServiceGroup (max($\sum(Val(s,t))$)

```
economicValue ->
  serviceValues = calculateMaxValue service, timestamps for service in
      serviceGroup
  Math.max.apply null, serviceValues

calculateMaxValue: (service, timeframe) ->
```

Listing 5.8: Max Economic Value Algorithm

Encoding such algorithms in CoffeeScript is trivial, with the max value calculations provided by the service provider capable of being changed at run time. CoffeeScript is ideally suited to representing such policy specifications and such applications of policy management. As such, this thesis recommends CoffeeScript as a representation format for encoding policies.

## 5.3 Summary

This chapter presented the output models associated with the work carried out within this thesis. A set of UML based models capturing the relationships and responsibilities of core components necessary for optimised and scalable management of groups within emerging social networks. A set of implementation recommendations are also included and presented, showcasing current state of the art technologies and recommendations which complement the models produced.

# Chapter 6

# Conclusions and Future Work

In this chapter the thesis is concluded, following which some potential areas for future work are outlined.

## 6.1 Conclusion

### 6.1.1 Thesis Summary

Chapter 2 presented an overview of the literature involved in group formation and management, including group communication technologies and current approaches to implementation strategies. The area of Group Formation was first discussed in the context of why people are attracted to groups. An analysis of the current role that groups play in modern telecommunications was presented with the changes within user behavior presented as a challenge for future formation mechanisms. A comprehensive review of group communication technologies was then discussed, showing the history and evolution of group based communication. The improvements offered by each new technology suite was driven in part by changing user requirements and expectations. Managing the interactions within groups is an important feature with emerging usage patterns exposing more and more information. A review of the best practices in PBNM was presented within this context and some of the challenges within this domain presented for consideration. The final section discussed how to programmatically design and implement a solution that will scale to address the challenges of emerging social networks and user interaction. A best practice approach to design patterns, deployment platforms, representation formats and some relevant language paradigms was presented.

In Chapter 3, approaches to Group Formation and Management with respect to

Group Membership around a core domain of interest was investigated. The XMPP protocol, a group communication service that has evolved into a fully functioning social network, had its grouping functionality, as handled by the roster, examined. A set of criteria for strengthen group management was put forward and analysed further. The flexibility of the roster within XMPP was used to evolve a Group Roster entity. The scalability limits of this approach were quickly reached and the performance limitations of the roster was then examined. The capability of XMPP to handle large scale groups was quickly established with recommendations put forward to programmatically handle this requirement. Moving beyond XMPP, mass group management was investigated within the context of a case study surrounding group formation driven by a reactive event. A thorough investigation of technologies and architectures to evaluate mass group formation was presented.

Chapter 4 looked at the performance and scalability of current group interaction mechanisms. This chapter began with a discussion on architecting management platforms to tailor them more closely to the target domain. A case was presented for group specific PDPs to fulfill this role instead of trying to manage an entire system. This tailored approach would be domain specific and highly responsive. Emerging usage patterns of interest were revisited, showing that a management layer could offer performance improvements within certain contexts. The performance of policy evaluations for controlling basic access control interactions within groups was then examined. An implementation based on a modern web development stack was shown to offer significant performance gains against the industry standard XACML. Additional experimentation investigated encoding policy representations within a modern programming language. The results are beyond current state of the art capabilities, in both response times and overall performance.

Chapter 5 presents the first steps towards creating a unified model for managing groups. The responsibilities of the model are presented individually before being presented as a unified model. Complementary implementation algorithms supplement the models presented. The chapter concludes with implementation recommendations for handling groups, including recommendations for storage, platforms and handling concurrency.

### 6.1.2 Contributions

The popularity and usage of social networking will continue to expand. A significant driver for this increased popularity will be mobile consumption of services and content.

In this regard, groups will become a focal point for user interaction. In order to provide a better quality of service through formation and management principles, this thesis has proposed an outline architecture and model for groups. This approach contains a number of novel contributions, arrived at from directly answering relevant Research Questions posed:

- *An Outline Model for Group Formation and Interaction Management:* This represents a contribution to the understanding and structure required to successfully facilitate groups in emerging social networks. Research Questions RQ-GFM1 and RQ-GFM2 were addressed by this contribution.

- *CoffeeScript Policies:* Policy representations that are capable of encoding algorithms and functions, with a performance improvement that satisfies near real time requirements. RQ-GFM3 prompted this investigation.

- *Scalable Component Design:*. The individual components of the model were implemented as a proof of concept, realising the key model characteristics. The performance of the models were validated in high throughput simulations. Addressing RQ-S1 and understanding the load posed by RQ-S2 allowed for an understanding of components that could facilitate RQ-GFM2.

- *Group Specific PDPs:* In order to provision adequate management structures for a group, this thesis proposes that a group specific PDP should be associated with each group. The lightweight, portable design of the PDPs presented within this thesis facilitates rapid deployment without the cost of semantics or overhead. Discovering the correct toolset to satisfy RQ-S3 allowed an understanding develop to tackle RQ-GFM2 and RQ-GFM3. The resulting analysis contributed to RQ-GFM1.

- *Modern Stack for Communications Management:* Modern cloud service development principles have been applied to PBNM middleware, demonstrating this highly innovative technology stack can supplement and even replace aspects of conventional application server technology in this context. The stack is capable of handling emerging usage patterns not currently provisioned for in existing group management domains. Understanding the limits within RQ-S1 allowed a solution stack emerge to satisfy RQ-S3.

- *Group Formation Case Study:* A case study for the application of group formation in a global domain was put forward. It was shown that group formation

and management techniques could cope with the load such a use case would bring. Additionally, the results are sufficiently encouraging enough to warrant further study. This scenario was an example of the scalability profile that RQ-S2 documented. The solution management mechanisms and representation formats, developed to satisfy RQ-GFM1 and RQ-GFM3, respectively, evolved through addressing this Research Question.

Table 6.1 provides a reference to the core sections and peer reviewed publications where Research Questions were investigated. In many instances, solutions to Research Questions emerge throughout multiple chapters and supporting publications, for clarity the table presents core sections and publications surrounding each question.

| Research Question | Thesis Section | Publication |
|---|---|---|
| RQ-GFM1 | Section 3.2.1 and Section 3.3 | Griffin *et al.* (2011a), Griffin *et al.* (2011c) Griffin *et al.* (2012a) |
| RQ-GFM2 | Section 3.3.1 and Section 4.3 | Foley *et al.* (2010), Griffin *et al.* (2011c), Griffin *et al.* (2012b) |
| RQ-GFM3 | Section 4.3 | Foley *et al.* (2010), Griffin *et al.* (2012b) |
| RQ-S1 | Section 3.3.1 and Section 4.3 | Griffin *et al.* (2011c), Griffin *et al.* (2012a), Griffin *et al.* (2012b) |
| RQ-S2 | Section 3.3 | Griffin *et al.* (2011c),Griffin *et al.* (2012b) |
| RQ-S3 | Section 5.1.5 and Section 5.2 | Griffin *et al.* (2011c), Griffin *et al.* (2012a), Griffin *et al.* (2012b) |

Table 6.1: Research Question Core Section Reference Table

### 6.1.3 Conclusions

The aim of this thesis was to demonstrate how the management of group formation and its interactions could be provisioned in emerging social networks. In line with this, a model was designed and presented to provide a scalable means of managing groups and their interactions. Through the simulations and results presented, this thesis has shown that current state of the art mechanisms are not meeting the requirements set out within this thesis, and the derived model and recommendations attempts to address this. Specifically, in terms of the original research questions posed:

- Case studies were presented showing emerging usage patterns which current group mechanisms cannot facilitate, showing a scalability bottleneck (RQ-S1 and RQ-S2)
- A novel management design was presented to handle large scale group based interactions (RQ-GFM1 and RQ-GFM2)
- A technology stack and associated best practices were used to derive a model to support group formation and management. (RQ-S3 and RQ-GFM3)

## 6.2 Future Work

Many fertile areas worth investigating have been identified and a number are listed here as possible extension points:

### 6.2.1 Humanitarian Relief: Disaster Management Applications

While investigating emerging usage patterns, the area of Disaster Management offered interesting and relevant use cases. Currently Social Networks plays an informal role in Disaster Management communication, a structured attempt to better inform First Responders was investigated, with the emphasis placed on stress testing the underlying infrastructure. A further investigation of the technology and outputs of this thesis in the realm of disaster management with respect to service deployment would help better inform and refine the model produced. The components produced will be available as Open Source (Griffin (2012)) allowing for a more rigorous testing. The technology is highly scalable and could benefit from a domain expert insight into how the communication channels should be organised and how a real world response would be coordinated.

Contributions have already been made to a research projects, SOCIETIES[1], which is investigating the facilitating role of Social Networking within Disaster Management (Roussaki (2011)).

### 6.2.2 Fine Grained Management through SLAs

Complex algorithms are used to check for SLA violations (Das (2012)) which could be encoded into policy documents and checked on each access request. This is particularly useful when it comes to prioritising services within a group, particularly when maximising revenue from the service provider is a factor. From a service providers perspective, maximising revenue with limited resources might involve prioritising specific services at a cost of reducing the response time of another service (Macias *et al.* (2009)). Through calculating the number of requests by the probability that they don't violate an SLA threshold, the ability to decide which services to prioritise based on their economic value is possible. Maintaining this balance for as long as possible could maximise profit overall. From a computational point of view, executing an algorithm that calculates the probability that a response time associated with a service request is in breach of an SLA, with respect to the complexity of the calculation, is minimal for individual requests. In high volume request scenarios, the number of response time calculations over a time period could have a negative impact on the system, particularly if the system has to deal with other standard access control requests alongside the SLA violation checks. Encoding SLA breach checking algorithms at a PDP level rather then a third party tool allows for a more flexible polling algorithm. The CoffeeScript policy representation and supporting environment is a potential candidate for delivering SLA checking in a manner that does not impact on the timeliness of the response, allowing a service provider maximise the revenue generated from service provisioning on their platform.

### 6.2.3 Policy Continuum

The flexibility of the CoffeeScript Policy representation lends itself to analysis within the overall Policy Continuum (Davy *et al.* (2008)), with the DSL nature of the represented policies allowing potentially clearer abstractions and hierarchies to be built. From a policy authoring point of view the clarity and simplification of the CoffeeScript

[1]Self Orchestrating CommunIty ambiEnT IntelligEnce Spaces (SOCIETIES) http://www.ict-societies.eu/

language could support more terse specifications. The language semantics and policy representation could additionally yield a performance gain on the basic operations that are performed within the continuum, particularly for searching and combining policies. The performance benefits over a JVM and XML based implementation could also allow for a more efficient utilisation of computing resources. The free CPU cycles could perform consistency checks, an important feature within the conflict analysis domain. However, the CoffeeScript PDP performed analysis on simple access control policies. The full power of the PDP would be realised when tackling a more complicated set. Access and availability to policy requests and documents are limited by security and privacy concerns regarding the data they contain. As such access to *real* policies is difficult. A refinement of the policy language would only be possible with access to more rigorous sets.

# References


Abelson, H. & Sussman, G.J. (1996). *Structure and Interpretation of Computer Programs - 2nd Edition (MIT Electrical Engineering and Computer Science)*. The MIT Press. 33

Acar, A. & Muraki, Y. (2011). Twitter for Crisis Communication: Lessons Learned from Japan's Tsunami Disaster. *International Journal of Web Based Communities*, **7**, 392–402. 23

Adams, P. (2011). *Grouped: How Small Groups of Friends are the Key to Influence on the Social Web (Voices That Matter)*. New Riders Press. 10, 24

Ahn, G.J. & Sandhu, R. (2000). Role-Based Authorization Constraints Specification. *ACM Transactions on Information and System Security*, **3**, 207–226. 28

Alez, G. (2012). *Java Virtual Machines: Introduction, JVM Languages, Bytecode Verifier, Secure Execution of Remote Code, and More*. Webster's Digital Services. 34

Assayag, G. (2009). *New Computational Paradigms for Computer Music*. Editions Delatour France. 33

Atzori, L., Iera, A. & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, **54**, 2787–2805. 3

Baatarjav, E.A., Phithakkitnukoon, S. & Dantu, R. (2008). Group Recommendation System for Facebook. In *Proceedings of the 2008 OTM Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems*, OTM '08, 211–219, Springer-Verlag, Berlin, Heidelberg. 24

Backstrom, L., Huttenlocher, D., Kleinberg, J. & Lan, X. (2006). Group Formation in Large Social Networks: Membership, Growth, and Evolution. In *Proceed-*

*ings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, 44–54, ACM, New York, NY, USA. 12, 13

Bareth, U., Kupper, A. & Ruppel, P. (2010). geoXmart - A Marketplace for Geofence-Based Mobile Services. In *Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference*, COMPSAC '10, 101–106, IEEE Computer Society, Washington, DC, USA. 12

Barron, J., Davy, S. & Jennings, B. (2011). Conflict Analysis During Authoring of Management Policies for Federations. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, 1180 –1187. 30

Beach, A., Gartrell, M., Akkala, S., Elston, J., Kelley, J., Nishimoto, K., Ray, B., Razgulin, S., Sundaresan, K., Surendar, B., Terada, M. & Han, R. (2008). WhozThat? Evolving an Ecosystem for Context-aware Mobile Social Networks. *Network, IEEE*, **22**, 50 –55. 26

Beck, K. & Andres, C. (2004). *Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series)*. Addison-Wesley. 33

Belblidia, N. & Debbabi, M. (2007). A Dynamic Operational Semantics for JVML. *Journal of Object Technology*, **6**, 71–100. 119

Belic, D. (2012). Foursquare Surpasses 20 Million Users, 2 Billion Check-ins http://www.intomobile.com/2012/04/21/foursquare-surpasses-20-million-users-2-billion-checkins/ Last accessed: 16/07/2012. 26

Bloch, J. (2008). *Effective Java (2nd Edition)*. Addison-Wesley. 64

Botz (2007). Botz: Internal Bot Library for Openfire http://community.igniterealtime.org/docs/DOC-1130/version Last accessed: 16/07/2012. 67

Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S. & Winer, D. (2000). Simple Object Access Protocol (SOAP) 1.1. 31

Boyd, d.m. & Ellison, N.B. (2007). Social Network Sites: Definition, History, and Scholarship. *Journal of Computer-Mediated Communication*, **13**, 210–230. 22

BugLabs (2012). Node XML2JSON https://github.com/buglabs/node-xml2json Last accessed on 10-08-2012. 102

Bulatov, A.A. (2011). Complexity of Conservative Constraint Satisfaction Problems. *ACM Transactions Computational Logic*, **12**, 24:1–24:66. 11

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M. (1996). *Pattern-oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc., New York, NY, USA. 33

Buschmann, F., Henney, K. & Schmidt, D.C. (2007a). *Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing (v. 4)*. Wiley. 33

Buschmann, F., Henney, K. & Schmidt, D.C. (2007b). *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages*. Wiley. 33

Butler, B., Jennings, B. & Botivch, D. (2010). XACML Policy Performance Evaluation Using a Flexible Load Testing Framework. In *Proc. 17th ACM Conference on Computer and Communications Security (CCS 2010)*, 648–650, ACM, short paper. 30

Butler, B., Jennings, B. & Botvich, D. (2011). An Experimental Testbed to Predict the Performance of XACML Policy Decision Points. In *IM 2011 - TechSessions*, Dublin, Ireland. 99, 104

Buyya, R., Garg, S. & Calheiros, R. (2011). SLA-oriented Resource Provisioning for Cloud Computing: Challenges, Architecture, and Solutions. In *Cloud and Service Computing (CSC), 2011 International Conference on*, 1 –10. 13

Camarillo, G. (2009). Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability. RFC 5694 (Informational). 18

Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C. & Gurle, D. (2002). Session Initiation Protocol (SIP) Extension for Instant Messaging. RFC 3428 (Proposed Standard). 20

Campbell, C.E., Eisenberg, A. & Melton, J. (2003). XML Schema. *ACM SIGMOD Record*, **32**, 96–101. 37

Carlsson, J. (2010). An Assessment of Social Media Business Models and Strategic Omplications for Future Implementation http://tinyurl.com/c2z3b65 Last accessed: 16/07/2011. 13

Carreras, A., Rodríguez, E. & Delgado, J. (2009). Using XACML for Access Control in Social Networks. In *W3C Workshop on Access Control Application Scenarios*, W3C. 31

Cattell, R. (2011). Scalable SQL and NoSQL Data Stores. *ACM SIGMOD Record*, **39**, 12–27. 99

Chatterjee, S., Abhichandani, T., Li, H., TuIu, B. & Byun, J. (2005). Instant Messaging and Presence Technologies for College Campuses. *Network, IEEE*, **19**, 4 – 13. 19

Chen, H., Shen, H., Xiong, J., Tan, S. & Cheng, X. (2006). Social Network Structure Behind the Mailing Lists: ICT-IIIS. In *The Fifteenth Text REtrieval Conference (TREC)*. 15

Chen, M., Gu, B. & Konana, P. (2009). Social Capital, Social Identity and Homophily Behavior in Virtual Communities: An Anaylsis of Interactions in Stock Message Boards. 16

Chen, R.S., Tsai, Y.S., Yeh, K.C., Yu, D.H. & Bak-Sau, Y. (2008). Using Data Mining to Provide Recommendation Service. *WSEAS Transactions on Information Science and Applications*, **5**, 459–474. 13

Cheng, R. & Vassileva, J. (2005). User Motivation and Persuasion Strategy for Peer-to-Peer Communities. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Volume 07*, HICSS '05, 193.1–, IEEE Computer Society, Washington, DC, USA. 18

Churchill, E.F. & Nelson, L. (2007). Interactive Community Bulletin Boards as Conversational Hubs and Sites for Playful Visual Repartee. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, 76. 16

Cohen, B. (2008). The BitTorrent Protocol Specification, Version 11031. http://www.bittorrent.org/beps/bep_0003.html Accessed on 10-Aug-2012. 18

Cohn, M. (2009). *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional. 33

Crockford, D. (2006a). The Application/JSON Media Type for Javascript Object Notation (JSON). RFC 4627 (Informational). 37

Crockford, D. (2006b). Json: The Fat Free Alternative to XML. 15th International World Wide Web Conference. 37

Crockford, D. (2008). *JavaScript: The Good Parts*. O'Reilly Media, Inc. 35

Dahl, O.J. (2002). Software Pioneers. 78–90, Springer-Verlag New York, Inc., New York, NY, USA. 34

Dahl, R. (2009). Node.js https://github.com/joyent/node Last accessed: 16/07/2012. 39, 65, 99

Dalgaard, P. (2008). *Introductory Statistics with R*. Statistics and Computing, Springer. 110

Das, A. (2012). Maximizing Profit using SLA-aware Provisioning. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, 393 –400. 13, 148

Dattatreya, V., Rao, C. & Rayudu2, V. (2012). Agile Programming and Design Patterns in Web Development - A Case Study. *International Journal of Software Engineering & Applications*, **3**, 37 – 45. 33

Davy, S., Jennings, B. & Strassner, J. (2008). The Policy Continuum-Policy Authoring and Conflict Analysis. *Compututer Communication*, **31**, 2981–2995. 148

Dennis, A.R. & Wixom, B.H. (2002). Investigating the Moderators of the Group Support Systems Use with Meta-Analysis. *Journal of Management Information Systems*, **18**, 235–257. 27

Deterding, S., Dixon, D., Khaled, R. & Nacke, L. (2011). From Game Design Elements to Gamefulness: Defining "Gamification". In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '11, 9–15, ACM, New York, NY, USA. 72

Ding, L., Zhou, L., Finin, T. & Joshi, A. (2005). How the Semantic Web is Being Used: An Analysis of FOAF Documents. In *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, 113c. 11

Downes, S., Belliveau, L., Samet, S., Abdur Rahman, M. & Savoie, R. (2010). Managing Digital Rights Using JSON. In *7th IEEE Consumer Communications and Networking Conference (CCNC)*, 1–10. 37

ECMAScript (2011). ECMAScript Language Specification Version 5.1. 35

Eriksen, M. (2010). Scaling Scala at Twitter. In *ACM SIGPLAN Commercial Users of Functional Programming*, CUFP '10, 8:1–8:1, ACM, New York, NY, USA. 22

Eto, K., Takabayashi, S. & Masui, T. (2005). qwikWeb: Integrating Mailing List and WikiWikiWeb for Group Communication. In *Proceedings of the 2005 International Symposium on Wikis*, WikiSym '05, 17–23, ACM, New York, NY, USA. 15

Facebook (2012). Facebook Graph API Specifcation http://developers.facebook.com/docs/reference/api Last accessed: 16/07/2012. 25

Fang, L. & LeFevre, K. (2010). Privacy Wizards for Social Networking Sites. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, 351–360, ACM, New York, NY, USA. 24

Fielding, R.T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. thesis, aAI9980887. 31

Fisher, D., Smith, M. & Welser, H. (2006). You Are Who You Talk To: Detecting Roles in Usenet Newsgroups. In *System Sciences, 2006. HICSS '06. Proceedings of the 39th Annual Hawaii International Conference on*, vol. 3, 59b. 17

Foley, C., Power, G., Griffin, L., Chen, C., Donnelly, N. & de Leastar, E. (2010). Service Group Management Facilitated by DSL Driven Policies in Embedded Middleware. In *Proceedings of the 2010 IEEE Symposium on Computers and Communications*, ISCC '10, 483–488, IEEE Computer Society, Washington, DC, USA. iv, 146

Fowler, M. (2005). Language Workbenches: The Killer-App for Domain Specific Languages? Http://martinfowler.com/articles/languageWorkbench.html Last accessed: 18/07/2012. 36

Fowler, M.J. (2010). *Domain-Specific Languages (Addison-Wesley Signature Series (Fowler))*. Addison-Wesley Professional. 36

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional. 33, 34, 90, 91

Garcia, Y.K. & Ketel, M. (2012). An Economical Approach to PaaS. In *Proceedings of the 50th Annual Southeast Regional Conference*, ACM-SE '12, 357–358, ACM, New York, NY, USA. 38

Gazi, V. & Passino, K. (2004). Stability Analysis of Social Foraging Swarms. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, **34**, 539 –557. 8, 9

GNU (2012). GNU Scientific Library http://www.gnu.org/software/gsl/. 78

Goldberg, A. & Robson, D. (1983). *Smalltalk-80: The Language and its Implementation*. Addison-Wesley. 34

Gonzalez, V.M., Rodriguez, M.D. & Colsa, L.M. (2008). Connecting Families with ICTs: A Board Messaging System for Older Adults and their Family Abroad. In *Technology and Society, 2008. ISTAS 2008. IEEE International Symposium on*, 1–4. 17

Google (2010). Google V8 Javascript Engine http://code.google.com/p/v8/ Last accessed: 17/07/2012. 39

Gosling, J., Joy, B., Steele, G.L. & Bracha, G. (2005). *The Java Language Specification*. Addison-Wesley, Upper Saddle River, NJ, 3rd edn. 34

Granell, C., Díaz, L. & Gould, M. (2010). Service-Oriented Applications for Environmental Models: Reusable Geospatial Services. *Environmental Modelling & Software*, **25**, 182–198. 26

Griffin, L. (2012). Leigh Griffin Github Repository https://github.com/lgriffin Last accessed: 09/08/2012. 147

Griffin, L. & de Leastar, E. (2008). A Nodel for IM and Media Driven Communication Services. In *8th International Conference on Technology and Telecommunication*. v

Griffin, L. & de Leastar, E. (2009). Social Networking Healthcare. In *Wearable Micro and Nano Technologies for Personalized Health (pHealth), 2009, 6th International Workshop on*, 75 –78. v

Griffin, L., Foley, C. & de Leastar, E. (2009). A Hybrid Architectural Style for Cmplex Healthcare Scenarios. In *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on Communications*, 1 –6. v

Griffin, L., de Leastar, E. & Botvich, D. (2011a). Dynamic Shared Groups Within XMPP: An Investigation of the XMPP Group Model. In *Integrated Network Management*, 634–637, IEEE. iv, 146

Griffin, L., Elger, P. & de Leastar, E. (2011b). Project Zeppelin: A Modern Web Application Development Framework. In *Formal Methods for Components and Objects, 2011, 10th International Symposium on*. v

Griffin, L., Ryan, K., de Leastar, E. & Botvich, D. (2011c). Scaling Instant Messaging Communication Services: A Comparison of Blocking and Non-Blocking Techniques. In *Proceedings of the 2011 IEEE Symposium on Computers and Communications*, ISCC '11, 550–557, IEEE Computer Society, Washington, DC, USA. iv, 146

Griffin, L., Ryan, K., de Leastar, E. & Botvich, D. (2012a). Scaling Instant Messaging Communication Services: A Comparison of Blocking and Non-Blocking Techniques. *International Journal of Ambient Computing and Intellgience*, **4**, 1–20. iv, 146

Griffin, L., Ryan, K., de Leastar, E., Jennings, B. & Botvich, D. (2012b). On the Performance of Access Control Policy Evaluation. In *Proceedings of the 2012 IEEE International Symposium on Policies for Distributed Systems and Networks*, POLICY '12. iv, 146

Grinter, R.E. & Palen, L. (2002). Instant Messaging in Teen Life. In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, CSCW '02, 21–30, ACM, New York, NY, USA. 19

Gschwind, T. & Hauswirth, M. (1999). A Cache Architecture for Modernizing the Usenet Infrastructure. In *Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 8*, HICSS '99, IEEE Computer Society, Washington, DC, USA. 17

Haake, J.M., Haake, A., Schümmer, T., Bourimi, M. & Landgraf, B. (2004). End-user Controlled Group Formation and Access Rights Management in a Shared Workspace System. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04, 554–563, ACM, New York, NY, USA. 10

Hallberg, J., Norberg, M.B., Kristiansson, J., Synnes, K. & Nugent, C. (2007). Creating Dynamic Groups using Context-Awareness. In *Proceedings of the 6th International Conference on Mobile and Ubiquitous Multimedia*, MUM '07, 42–49, ACM, New York, NY, USA. 12, 42

Haller, P. & Odersky, M. (2009). Scala Actors: Unifying Thread-based and Event-based Programming. *Theoretical Computer Science*, **410**, 202–220. 35, 64

Haveliwala, T.H. (2002). Search Facilities for Internet Relay Chat. In *Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '02, 395–395, ACM, New York, NY, USA. 19

Havenstein, H. (2007). LA Fire Department all a Twitter over Web 2.0 http://tinyurl.com/6hlx24 Last accessed: 16/07/2012. 23

Haw, R., Hong, C.S. & Kim, D.S. (2009). Group P2P Network Organization in Mobile Ad-Hoc Networks. In *Proceedings of the 12th Asia-Pacific Network Operations and Management Conference on Management Enabling the Future Internet for Changing Business and New Computing Services*, APNOMS'09, 477–480, Springer-Verlag, Berlin, Heidelberg. 18

Heinze, A. & Procter, C. (2006). Online Communication and Information Technology Education. *Journal of Information Technology Education*, **5**, 235–249. 27

Hoigaard, E. (2011). *Smooth CoffeeScript*. xii, 179, 180, 181

Horton, M. & Adams, R. (1987). Standard for Interchange of USENET Messages. RFC 1036, obsoleted by RFCs 5536, 5537. 17

Hsieh, G., Foster, K., Emamali, G., Patrick, G. & Marvel, L. (2009). Using XACML for Embedded and Fine-Grained Access Control Policy. In *Availability, Reliability and Security, 2009. ARES '09. International Conference on*, 462 –468. 31

Hsu, C.I., Chao, C.C. & Shih, K.Y. (2012). Dynamic Allocation of Check-in Facilities and Dynamic Assignment of Passengers at Air Terminals. *Computers & Industrial Engineering*, **63**, 410–417. 26

Huang, Q. & Liu, Y. (2009). On Geo-social Network Services. In *Geoinformatics, 2009 17th International Conference on*, 1 –6. 12

Hui, P., Crowcroft, J. & Yoneki, E. (2011). BUBBLE Rap: Social-Based Forwarding in Delay-Tolerant Networks. *Mobile Computing, IEEE Transactions on*, **10**, 1576 –1589. 9, 13

Huonder, F. (2010). Conflict Detection and Resolution of XACML Policies. Masters Thesis, University of Applied Sciences Rapperswil. 30

Husted, R. & Kushlich, J.J. (1999). *Server-side JavaScript: Developing Integrated Web Applications*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 35

Imielinski, T. & Lipski, W., Jr. (1982). A Systematic Approach to Relational Database Theory. In *Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data*, SIGMOD '82, 8–14, ACM, New York, NY, USA. 37

Inagaki, T., Komatsu, H. & Nakatani, T. (2003). Integrated Prepass Scheduling for a Java Just-In-Time Compiler on the IA-64 Architecture. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '03, 159–168, IEEE Computer Society, Washington, DC, USA. 54

Jeannin, J.B. (2011). Capsules and Closures. *Electronic Notes in Theoretical Computer Science*, **276**, 191–213. 34

Jiang, H. & Carroll, J.M. (2009). Social Capital, Social Network and Identity Bonds: a Reconceptualization. In *C&T '09: Proceedings of the Fourth International Conference on Communities and Technologies*, 51–60, ACM, New York, NY, USA. 16

Johnson-Lenz, P. & Johnson-Lenz, T. (1991). Rhythms, Boundaries, and Containers: Creative Dynamics of Asynchronous Group Life. *The International Journal of Man Machine Studies*, **34**, 395–417. 21

Jones, S. & O'Neill, E. (2010). Feasibility of Structural Network Clustering for Group-based Privacy Control in Social Networks. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, SOUPS '10, 9:1–9:13, ACM, New York, NY, USA. 3, 24

Kalt, C. (2000). Internet Relay Chat: Channel Management. RFC 2811 (Informational). 19

Kamvar, S.D., Schlosser, M.T. & Garcia-Molina, H. (2003). The Eigentrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, 640–651, ACM, New York, NY, USA. 18

Kantor, B. & Lapsley, P. (1986). Network News Transfer Protocol. RFC 977 (Proposed Standard), obsoleted by RFC 3977. 17

Kegel, D. (1999). The C10K Problem http://www.kegel.com/c10k.html Last accessed: 16/07/2012. 35

Kimura, T. (1998). JTC 1SC 34 Document Description and Processing Language http://www.iso.org/iso/iso_technical_committee.html/commid=45374 Last accessed: 16/07/2012. 37

Kinsella, S., Passant, A. & Breslin, J.G. (2010). Using Hyperlinks to Enrich Message Board Content with Linked Data. In *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, 1:1–1:9, ACM, New York, NY, USA. 16

Kircher, M. & Jain, P. (2004). *Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management*. Wiley. 33

Klensin, J. (2008). Simple Mail Transfer Protocol. RFC 5321 (Draft Standard). 15

Koenig, D., Glover, A., King, P., Laforge, G. & Skeet, J. (2007). *Groovy in Action*. Manning Publications. 34

Kolaitis, P.G. & Vardi, M.Y. (2000). Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Sciences*, **61**, 302 – 332. 11

Kowal, K. (2009). CommonJS Effort sets JavaScript on Path for World Domination. 100

LAFD (2011). LAFD Twitter Feed http://twitter.com/LAFD. 23

Lai, L.S. & Turban, E. (2008). Group Formation and Operations in the Web 2.0 Environment and Social Networks. *Group Decision and Negotiation*, **17 (5)**, 387–402. 12

Lerner, R.M. (2010). At the Forge: Redis. *Linux Journal*, **197**. 99

Lerner, R.M. (2011). At the Forge: Node.js. *Linux Journal*, **2011**. 39

Levine, J.M. & Mooreland, R.L. (1991). *Culture and Socialization in Work Groups*. American Psychological Association. 9, 10

Liang, H., Chen, W. & Shi, K. (2011). Cloud Computing: Programming Model and Information Exchange Mechanism. In *Proceedings of the 2011 International Conference on Innovative Computing and Cloud Computing*, ICCC '11, 10–12, ACM, New York, NY, USA. 38

Lindholm, T. & Yellin, F. (1999). *Java Cirtual Machine Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edn. 34

Liu, A.X., Chen, F., Hwang, J. & Xie, T. (2008). Xengine: a Fast and Scalable XACML Policy Evaluation Engine. In *Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 265–276, ACM, New York, NY, USA. 30, 31, 105

Liu, A.X., Chen, F., Hwang, J. & Xie, T. (2011). Designing fast and scalable xacml policy evaluation engines. *IEEE Transactions on Computers*, **60**, 1802–1817. 30

Liu, D. & Deters, R. (2009). The Reverse C10K Problem for Server-Side Mashups. International Conference on Service-Oriented Computing, ICSOC 2008 Workshops. 166–177, Springer-Verlag, Berlin, Heidelberg. 35

Liu, J., Sacchetti, D., Sailhan, F. & Issarny, V. (2005). Group Management for Mobile Ad Hoc Networks: Design, Implementation and Experiment. In *Proceedings of the 6th International Conference on Mobile Data Management*, MDM '05, 192–199, ACM, New York, NY, USA. 28

Long, B. & Long, B.W. (2003). Formal Specification of Java Concurrency to Assist Software Verification. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, IEEE Computer Society, Washington, DC, USA. 61

Lu, C., Hu, X. & ran Park, J. (2011). Exploiting the Social Tagging Network for Web Clustering. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, **41**, 840 –852. 11

Lubke, R., Schuster, D. & Schill, A. (2011). MobilisGroups: Location-based Group Formation in Mobile Social Networks. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*, 502 –507. 12

Ludwig, S., Beda, J., Saint-Andre, P., McQueen, R., Egan, S. & Hildebrand, J. (2009). Jingle. XEP 0166 (Proposed Standard). 20

Lv, Q., Cao, P., Cohen, E., Li, K. & Shenker, S. (2002). Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of the 16th International Conference on Supercomputing*, ICS '02, 84–95, ACM, New York, NY, USA. 18

Maaradji, A., Hacid, H., Daigremont, J. & Crespi, N. (2010). Towards a Social Network Based Approach for Services Composition. In *Communications (ICC), 2010 IEEE International Conference on*, 1 –5. 13

MacCaw, A. (2011). *The Little Book on CoffeeScript*. 36, 115

Macias, M., Smith, G., Rana, O., Guitart, J. & Torres, J. (2009). Enforcing Service Level Agreements Using an Economically Enhanced Resource Manager. In D. Neumann, M. Baker, J. Altmann & O. Rana, eds., *Economic Models and Algorithms for Distributed Systems*, Autonomic Systems, 109–127, Birkhuser Basel. 148

Martens, W., Neven, F., Schwentick, T. & Bex, G.J. (2006). Expressiveness and Complexity of XML Schema. *ACM Transactions on Database Systems*, **31**, 770–813. 37

Maruoka, M., Nemati, A., Barolli, V., Enokido, T. & Takizawa, M. (2008). Role-Based Access Control in Peer-to-Peer (P2P) Societies. In *Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on*, 495 –500. 28

Maximilien, E. (2006). Web Services on Rails: Using Ruby and Rails for Web Services Development and Mashups. In *Services Computing, 2006. SCC '06. IEEE International Conference on*, xxxix. 38

Meyer, B. (1997). *Object-Oriented Software Construction (2nd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. 34

Millard, P., Saint-Andre, P. & Meijer, R. (2002). XEP-0060: Publish-Subscribe http://xmpp.org/extensions/xep-0060.html Last accessed: 16/07/2012. 48

Mirkovic, J., Dietrich, S., Dittrich, D. & Reiher, P. (2005). *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall. 57

Mirza, Q.K.A. (2011). Restful Implementation of Authorization Mechanisms. In *International Conference on Technology and Business Management*. 31

Moreland, R.L. & Levine, J.M. (1992). *The Composition of Small Groups*, vol. 9. JAI. 9, 10

Moses, T. (2005). eXtensible Access Control Markup Language TC v2.0 (XACML). xii, xiii, 29, 182, 183, 184

Mourad, A., Otrok, H., Yahyaoui, H. & Baajour, L. (2011). Toward an Abstract Language on Top of XACML for Web Services Security. In *Internet Technology and Secured Transactions (ICITST), 2011 International Conference for*, 254 –259. 29

Mutton, P. (2004). Inferring and Visualizing Social Networks on Internet Relay Chat. In *Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on*, 35 – 43. 19

Myoupo, J.F., Naimi, M. & Thiare, O. (2009). A Clustering Group Mutual Exclusion Algorithm for Mobile Ad-hoc Networks. In *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*, 693 –696. 12

Nasseri, E. & Counsell, S. (2009). An Empirical Study of Java System Evolution at the Method Level. In *Proceedings of the 2009 Seventh ACIS International Conference on Software Engineering Research, Management and Applications*, SERA '09, 199–206, IEEE Computer Society, Washington, DC, USA. 34

Nodejitsu (2011). Hook IO Library, https://github.com/hookio/ Last accessed: 05/08/2012. 76

Oaks, S. & Wong, H. (2004). *Java Threads*. O'Reilly Media, Inc. 35

OASIS (2007). OASIS Content Assembly Mechanism Specification V1.1. 37

OASIS XACML-TC (2005). XACML 2.0 http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-ALL.zip Last accessed on 18/08/2012. 86, 94

Odersky, M., Spoon, L. & Venners, B. (2011). *Programming in Scala: A Comprehensive Step-by-Step Guide, 2nd Edition*. Artima Inc. 34

Oikarinen, J. & Reed, D. (1993). Internet Relay Chat Protocol. RFC 1459 (Experimental), updated by RFCs 2810, 2811, 2812, 2813. 18

Openfire (2009). Openfire Connection Manager, http://www.igniterealtime.org/projects/openfire/connection_manager.jsp Last accessed: 16/07/2012. 58

Openfire (2012). Openfire XMPP Server 3.7.1, http://www.igniterealtime.org/projects/openfire/l Last accessed: 16/07/2012. 48, 53, 63

Ostermann, K. & Mezini, M. (2001). Object-Oriented Composition Untangled. In *Proceedings of the 16th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA '01, 283–299, ACM, New York, NY, USA. 34

Ounnas, A. (2010). *Enhancing the Automation of Forming Groups for Education with Semantics*. Ph.D. thesis, University of Southampton. 11

Ounnas, A., Davis, H. & Millard, D. (2008). A Framework for Semantic Group Formation. In *Proceedings of the 2008 Eighth IEEE International Conference on Advanced Learning Technologies*. 11

Owens, D.A., Mannix, E.A. & Neale, M.A. (1998). Strategic Formation of Groups: Issues in Task Performance and Team Member Selection. *Research On Managing Groups and Teams*, **1**, 149165. 8

Park, J.S. & Hwang, J. (2003). Role-Based Access Control for Collaborative Enterprise in Peer-to-Peer Computing Environments. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, SACMAT '03, 93–99, ACM, New York, NY, USA. 28

Patterson, D.J., Baker, C., Ding, X., Kaufman, S.J., Liu, K. & Zaldivar, A. (2008). Online Everywhere: Evolving Mobile Instant Messaging Practices. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, UbiComp '08, 64–73, ACM, New York, NY, USA. 20

Petzold, C. (2008). *The Annotated Turing: A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine*. Wiley. 116

Ponnusamy, V., Karuppiah, E. & Abdullah, R. (2003). Anycast Group Membership Management Protocol. In *Communications, 2003. APCC 2003. The 9th Asia-Pacific Conference on*, vol. 3, 1052 – 1056 Vol.3. 28

Potts, L. (2009). Peering into Disaster: Social Software use from the Indian Ocean Earthquake to the Mumbai Bombings. In *Professional Communication Conference, 2009. IPCC 2009. IEEE International*, 1 –8. 2, 22

Pouwelse, J.A., Garbacki, P., Epema, D. & Sips, H. (2008). Pirates and Samaritans: A Decade of Measurements on Peer Production and their Implications for Net Neutrality and Copyright. *The International Journal of ICT Economy, Governance and Society*, **32**, 701–712. 18

Raento, M., Oulasvirta, A., Petit, R. & Toivonen, H. (2005). Contextphone: a Prototyping Platform for Context-aware Mobile Applications. *Pervasive Computing, IEEE*, **4**, 51 – 59. 26

Rafaeli, S., Ravid, G. & Soroka, V. (2004). De-lurking in Virtual Communities: a Social Communication Network Approach to Measuring the Effects of Social and Cultural Capital. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, 10 pp. 10

Ray, T. & Liew, K. (2003). Society and Civilization: An Optimization Algorithm based on the Simulation of Social Behavior. *Evolutionary Computation, IEEE Transactions on*, **7**, 386 – 396. 8, 9, 15

Richards, M., Monson-Haefel, R. & Chappell, D.A. (2009). *Java Message Service*. O'Reilly Media. 63

Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. & Schooler, E. (2002). SIP: Session Initiation Protocol. RFC 3261, updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630. 20

Roussaki, I. (2011). Societies Deliverable D2.2: Scenario Description, Use Cases and Technical Requirements Specification. 148

Rushe, D. (2012). Facebook Shares Open at $42 as it Begins Trading on Nasdaq http://guardian.co.uk/technology/2012/may/18/facebook-nasdaq Last accessed: 16/07/2012. 24

Saint-Andre, P. (2004a). XEP-0140: Shared Groups http://xmpp.org/extensions/xep-0140.html Last accessed: 16/07/2012. 50

Saint-Andre, P. (2004b). Extensible Messaging and Presence Protocol (XMPP): Core. RFC 9320 (Proposed Standard). 20, 25, 43

Saint-Andre, P. (2005). XEP-0144: Roster Item Exchange. Http://xmpp.org/extensions/xep-0144.html Last accessed: 16/07/2012. 50

Saint-Andre, P. (2009). XMPP: Lessons Learned from Ten Years of XML Messaging. *Communications Magazine, IEEE*, **47**, 92 –96. 49

Saint-Andre, P. (2012). XEP-0114: Jabber Component Protocol. Http://xmpp.org/extensions/xep-0114.html Last accessed: 05/08/2012. 65

Sandhu, R., Coyne, E., Feinstein, H. & Youman, C. (1996). Role-Based Access Control Models. *Computer*, **29**, 38 –47. 28

Schmidt, A., Beigl, M. & Gellersen, H.W. (1999). There is More to Context than Location. *Computers & Graphics*, **23**, 893 – 901. 26

Schmidt, D., Stal, M., Rohnert, H. & Buschmann, F. (2000). *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*. Wiley. 33

Schwartz, R. (2006). Web 2.0, Meet Usenet 1.0 http://www.stonehenge.com/merlyn/LinuxMag/col82.html Last accessed: 16/07/2012. 17

Sherlock, L.M. (2007). When Social Networking meets Online Games: The Activity System of Grouping in World of Warcraft. In *Proceedings of the 25th Annual ACM International Conference on Design of Communication*, SIGDOC '07, 14–20, ACM, New York, NY, USA. 11

Shihab, E., Bettenburg, N., Adams, B. & Hassan, A.E. (2010). On the Central Role of Mailing Lists in Open Source Projects: an Exploratory Study. In *Proceedings of the 2009 International Conference on New Frontiers in Artificial Intelligence*, JSAI-isAI'09, 91–103, Springer-Verlag, Berlin, Heidelberg. 15

Simula (2007). Simula Historical Documentation, http://www.edelweb.fr/Simula/ Last accessed: 16/07/2012. 34

Sjöholm, A., Seitz, L. & Sadighi, B. (2008). Secure Communication for Ad-hoc, Federated Groups. In *Proceedings of the 7th Symposium on Identity and Trust on the Internet*, IDtrust '08, 48–58, ACM, New York, NY, USA. 31

Smack (2012). Smack Client Library, http://www.igniterealtime.org/projects/smack/ Last accessed: 16/07/2012. 53

Staab, S., Domingos, P., Mike, P., Golbeck, J., Ding, L., Finin, T., Joshi, A., Nowak, A. & Vallacher, R. (2005). Social Networks Applied. *Intelligent Systems, IEEE*, **20**, 80 – 93. 11

Stewart, B., Lancaster, G., Lawson, J., Williams, K. & Daly, J. (2004). Validation of the Alder Hey Triage Pain Score. *Archives of Disease in Childhood*, **89**, 625–30. 73

Storni, C. & Griffin, L. (2009). Towards Future Health Social Networking: Patient Generated Content and the Role of Community Pharmacists. In *Mediterranean Conference on Information Systems )MCIS), 2009*. v

Strassner, J. (2003). *Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann. 28

Sun Microsystems (1996). JavaSoft Ships Java 1.0 http://www.skytel.co.cr/sun/research/1996/0123.htm Last accessed on 20-08-2012. 34

Takeuchi, M., Zakirov, S., Kawachiya, K. & Onodera, T. (2012). Fast Method Dispatch and Effective use of Primitives for Reified Generics in Managed X10. In *Proceedings of the 2012 ACM SIGPLAN X10 Workshop*, X10 '12, 4:1–4:7, ACM, New York, NY, USA. 119

Tilkov, S. & Vinoski, S. (2010). Node.js: Using Javascript to Build High-Performance Network Programs. *Internet Computing, IEEE*, **14**, 80 –83. 36

Turing, A.M. (1936-7). On Computable Number with an Application to the *Entscheidugsproblem. Proceedings of the American Mathematical Society*, **42**, 230–265. 116

Van Cutsem, T. (2008). *Ambient References: Object Designation in Mobile Ad Hoc Networks*. Ph.D. thesis, Vrije Universiteit Brussel, Faculty of Sciences, Programming Technology Lab. 36

van der Vlist, E. (2007). *Schematron*. O'Reilly, 1st edn. 37

van Noort, T., Achten, P. & Plasmeijer, R. (2010). Ad-hoc Polymorphism and Dynamic Typing in a Statically Typed Functional Language. In *Proceedings of the 6th ACM SIGPLAN Workshop on Generic Programming*, WGP '10, 73–84, ACM, New York, NY, USA. 34

Velasco, J.M., Atienza, D. & Olcoz, K. (2012). Memory Power Optimization of Java-based Embedded Systems Exploiting Garbage Collection Information. *Journal of Systems Architecture: Embedded Software Design*, **58**, 61–72. 119

Viegas, F. & Smith, M. (2004). Newsgroup Crowds and AuthorLines: Visualizing the Activity of OIndividuals in Conversational Cyberspaces. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, 10 pp. 17

Viswanathan, V. (2008). Rapid Web pplication Development: A Ruby on Rails Tutorial. *Software Magazine, IEEE*, **25**, 98 –106. 38

Volter, M. (2009). Best Practices for DSLs and Model-Driven Development. *Journal of Object Technology*, **8**, 79–102. 36

Vuillemot, R., Petit, J.M. & Hacid, M.S. (2011). Generalizing Email Messages Digests. In *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems*, CHI EA '11, 1933–1938, ACM, New York, NY, USA. 16

Wang, G. (2011). Improving Data Transmission in Web Applications via the Translation between XML and JSON. In *3rd IEEE International Conference on Communications and Mobile Computing*, CMC '11, 182–185, IEEE Computer Society, Washington, DC, USA. 37

Wang, H., Zhang, Y. & Cao, J. (2002). Design and Evaluation of XACML Conflict Policies Detection Mechanism. *International Journal of Computer Science & Information Technology*, **2**, 65 –74. 30

Wang, H., Zhang, Y. & Cao, J. (2009). Effective Collaboration with Information Sharing in Virtual Universities. *Knowledge and Data Engineering, IEEE Transactions on*, **21**, 840 –853. 28

Watanabe, Y., Sono, K., Yokomizo, K. & Okada, Y. (2004). A Question Answer System using Mails Posted to a Mailing List. In *Proceedings of the 2004 ACM Symposium on Document Engineering*, DocEng '04, 67–73, ACM, New York, NY, USA. 15

Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J. & Waldbusser, S. (2001). Terminology for Policy-Based Management. RFC 3198 (Informational). 28, 29

Westine, A. & Postel, J. (1991). Problems with the Maintenance of Large Mailing Lists. RFC 1211 (Informational). 15

Weverka, P. (2001). *Mastering ICQ: The Official Guide*. Wiley. 20

Wild, M. (2010). Xmpp Javascript Component Library. Https://github.com/mwild1/xmppjs Last accessed: 05/08/2012. 65

Wile, D. (2004). Lessons Learned from Real DSL Experiments. *Science of Computer Programming*, **51**, 265–290. 36

Wilson, P.R. & Hayes, B. (1991). Garbage Collection in Object Oriented Systems. *ACM Special Interest Group on Programming Languages (SIGPLAN), Object Oriented Programming Systems (OOPS) Messenger*, **3**, 63–71. 34

Wong, R.M., Dalmadge, C.L. & Fiedler, A.M. (2007). Managing eBusiness Continuity: Formulating an Appropriate Strategy to Manage System Scalability. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, 148b. 38

Wu, F. (2007). Presence Technology with its Security and Privacy Implications. In *Consumer Electronics, 2007. ISCE 2007. IEEE International Symposium on*, 1 –6. 25

Xiao, Z., Guo, L. & Tracey, J. (2007). Understanding Instant Messaging Traffic Characteristics. In *Distributed Computing Systems, 2007. ICDCS '07. 27th International Conference on*, 51. 20, 25, 53

XMPPRosterSchema (2012). XMPP Roster Schema. Http://xmpp.org/schemas/roster.xsd Last accessed: 19/07/2012. 44

Zdun, U. (2010). A DSL Toolkit for Deferring Architectural Decisions in DSL-Based Software Design. *Information and Software Technolgy*, **52**, 733–748. 36

Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J. & Chang, H. (2004). Qos-Aware Middleware for Web Services Composition. *Software Engineering, IEEE Transactions on*, **30**, 311 – 327. 13

Zhang, D., Guo, B. & Yu, Z. (2011a). The Emergence of Social and Community Intelligence. *Computer*, **44**, 21 –28. 26

Zhang, D., Wang, Z., Guo, B., Zhou, X. & Raychoudhury, V. (2011b). A Dynamic Community Creation Mechanism in Opportunistic Mobile Social Networks. In *Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on Social Computing (socialcom)*, 509 –514. 12

Zhao, Z., Liu, J. & Crespi, N. (2012). Dig-Event: Let's Socialize Around Events. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work Companion*, CSCW '12, 279–280, ACM, New York, NY, USA. 13

Zolfaghar, K. & Aghaie, A. (2012). A Syntactical Approach for Interpersonal Trust Prediction in Social Web Applications: Combining Contextual and Structural Data. *Knowledge-Based Systems*, **26**, 93–102. 25

# List of Acronyms

| | |
|---|---|
| CS | CoffeeScript |
| CSP | Constraint Satisfaction Problem |
| DSL | Domain Specific Language |
| FIFO | First In, First Out |
| FOAF | Friend of a Friend |
| GID | Group ID |
| HTML | HyperText Markup Language |
| I/O | Input/Output |
| IaaS | Infrastructure as a Service |
| ICT | Information and Communication Technologies |
| IM | Instant Messaging |
| IQ | Info/Query |
| IRC | Internet Relay Chat |
| JID | Jabber Identifier |
| JS | JavaScript |
| JSON | JavaScript Object Notation |
| JSONPL | JavaScript Object Notation Policy Language |
| JVM | Java Virtual Machine |

| | |
|---|---|
| OASIS | Organisation for Advancement of Structured Information Standards |
| OO | Object Oriented |
| P2P | Peer to Peer |
| PaaS | Platform as a Service |
| PDP | Policy Decision Point |
| PEP | Policy Enforcement Point |
| POSA | Pattern-Oriented Software Architecture |
| RBAC | Role Based Access Control |
| REST | Representational State Transfer |
| RoR | Ruby on Rails |
| SMTP | Simple Mail Transfer Protocol |
| SOAP | Simple Object Access Protocol |
| STACS | Scalability Testbed for Access Control Systems |
| XACML | eXtensible Access Control Modeling Language |
| XML | eXtensible Markup Language |

# Appendix A

# Domain Specific PDP

Below is a sample of a Domain Specific PDP based off of the Continue set. This PDP is 200 lines long and compliant semantically with the equivalent PDPs tested in Chapter 4

```
PRP = require("./PRP")
microtime = require("microtime")
masterPolicy = require('./policy.coffee')
onPEPMessage = (event, callback) ->
  timestamps = []
  timestamps[0] = microtime.now()
  PRP.onRequest "./policy.coffee", (policySet, timings) ->
   timestamps[2] = microtime.now()
   timestamps[1] = timings[1]
   callback policySet, timestamps

evaluate = (request, policySet, decision) ->
  timestamps = []
  timestamps[0] = microtime.now()
  masterPolicy2 = policySet
  processRequest request, (callback) ->
   timestamps[1] = microtime.now()
```

```
    processPolicy callback, (result) ->
     timestamps[2] = microtime.now()
     decision result, timestamps

processRequest = (request,callback) ->
  action = request.action.type
  subject = "test"
  resource = "test2"
  subjectCheck request.subject, (val) ->
    subject = val
  resourceCheck request.resource, (value) ->
    resource = value
  callback([subject,resource,action])

subjectCheck = (subject,callback) ->
  cb = []
  i = 0

  unless subject.subjReviewsThisResPaper is `undefined`
   cb[i] = ["subjReviewsThisResPaper", subject.subjReviewsThisResPaper]
   i++

  if subject['isEq-subjUserId-resUserId'] is true
   cb[i] = ['isEq-subjUserId-resUserId',subject['isEq-subjUserId-
       resUserId']]
   i++

  unless subject.isConflicted is `undefined`
   cb[i] = ["isConflicted", subject.isConflicted]
   i++

  unless subject.role is `undefined`
```

```
    cb[i] = ["role", subject.role]
    i++

  unless subject.isMeeting is `undefined`
    cb[i] = ["isMeeting", subject.isMeeting]
    i++

  unless subject.isSubjectsMeeting is `undefined`
    cb[i] = ["isSubjectsMeeting", subject.isSubjectsMeeting]
    i++

  callback(cb);

resourceCheck = (resource, callback) ->
  unless resource.isPending is `undefined`
    callback(["isPending",resource.isPending])
  unless resource.phase is `undefined`
    callback(["phase",resource.phase])
  unless resource.isSeeUnassignedAllowed is `undefined`
    callback(["isSeeUnassignedAllowed",resource.isSeeUnassignedAllowed])
  unless resource.class is `undefined`
    callback(["class",resource.class])
  unless resource['isEq-meetingPaper-resId'] is `undefined`
    callback(["isEq-meetingPaper-resId",resource['isEq-meetingPaper-resId
        ']])


processPolicy = (request,callback) ->
 identify_policySet request, (workingSet) ->
  if workingSet is "NotApplicable" then callback("NotApplicable")
  else
    makeDecision request,workingSet,(decision) ->
```

```
    callback(decision)


makeDecision = (request, policySet, decision) ->
  action = request[2]
  subject = request[0][0]
  resource = request[1]
  notApp = false
  oneDeny = false
  specialCase = false
  for policy in policySet
   if specialCase is true
    break
   subjectMatch = false
   actionMatch = false
   hasExtraRes = false
   effect = "tempDeny"
   correctSubjectWrongAnswer = false
   subjectNotApplicable = false
   specialSubjectMatch = false
   endOfTheLine = false
   undef = false
   undefSwitch = false
   noAct = false # we have no actions if this is true
   if policy.subjects is `undefined`
    if policy.actions is `undefined`
     if policy.resources is `undefined` and specialCase is false
      oneDeny = false # reset
      notApp = false # reset
      specialCase = true
      endOfTheLine = true
      decision(policy.effects)
    else
```

```
    for act in policy.actions
      if action is act
       actionMatch = true
       specialCase = true
       oneDeny = false # reset
       notApp = false # reset
       decision(policy.effects)
       break
  else
    if policy.actions is `undefined`
     noAct = true # but we have a subject though
    if policy.resources isnt `undefined` and specialCase is false
     if resource[0] is "class"
      hasExtraRes = true
      specialCase = true
      decision("deny")
      break
    for sub in policy.subjects
     if subject[0] is sub[0] and subject[1] is sub[1]
      subjectMatch = true
      undefSwitch = true # just in case we turn it off by accident
      undef = false
     else
      if subject[0] is sub[0]
       undef = true
       specialSubjectMatch = true
      else
       undef = true # get outta dodge
       subjectNotApplicable = true
     if noAct is true and subjectMatch is true
      specialCase = true
      oneDeny = false
```

```
      notApp = false
      decision(policy.effects)
      break
     if noAct is false and undef is false
      for act in policy.actions
       if action is act
        actionMatch = true
        specialCase = true
        oneDeny = false # reset
        notApp = false # reset
        decision(policy.effects)
        break
       else
        oneDeny = true # right subject wrong action

 if subjectMatch is true and actionMatch is false and specialCase is
     false
  oneDeny = true
 if subjectMatch is false and specialCase is false
  notApp = true
 if specialSubjectMatch is true
  oneDeny = true

if specialCase is false
 if oneDeny is true
  if notApp is true
   decision("deny")
  else
    decision("deny")
 else if notApp is true
  decision("NotApplicable")
```

```
identify_policySet = (request,callback) ->
 action = request[2]
 subject = request[0]
 res_type = request[1][0]
 res_val = request[1][1]
 reply = []
 if res_type is "class"
  target = masterPolicy[res_val]
  if target is `undefined`
   callback('NotApplicable')
  else
   callback(target)
 else
  findPolicy res_type,(cb) ->
   callback(cb)



findPolicy = (keyword,callback) ->
 found = []
 for policy in masterPolicy.policySet
  for p in policy when p.resources isnt `undefined`
   if(p.resources[0][0] is keyword)
    found.push(p)
 callback(found)


exports.onPEPMessage = onPEPMessage
exports.evaluate = evaluate;
```

Listing A.1: PDP Specification

# Appendix B

# CoffeeScript Glossary

The quick reference guide visible in Figure B.1 and Figure B.2 are reproduced from Hoigaard (2011).

# CoffeeScript Quick Reference

coffeescript.org

## General[1]

- Whitespace is significant
- Ending a line will terminate expressions - no need to use semicolons
- Semicolons can be used to fit multiple expressions onto a single line
- Use indentation instead of curly braces `{ }` to surround blocks of code in functions, *if* statements, *switch*, and *try/catch*
- Comments starts with `#` and run to the end of the line

## Functions

- Functions are defined by an optional list of parameters in parentheses, an arrow, and an optional function body. The empty function looks like: `->`
- Mostly no need to use parentheses to invoke a function if it is passed arguments. The implicit call wraps forward to the end of the line or block expression.
- Functions may have default values for arguments. Override the default value by passing a non-null argument.

## Objects and arrays

- Objects and arrays are similar to JavaScript
- When each property is listed on its own line, the commas are optional
- Objects may be created using indentation instead of explicit braces, similar to YAML
- Reserved words, like *class*, can be used as properties of an object without quoting them as strings

## Lexical Scoping and Variable Safety

- Variables are declared implicitly when used (no `var` keyword).
- The compiler ensures that variables are declared within lexical scope. An outer variable is not redeclared within an inner function when it is in scope
- Using an inner variable can not shadow an outer variable, only refer to it. So avoid reusing the name of an external variable in a deeply nested function
- CoffeeScript output is wrapped in an anonymous function, making it difficult to accidentally pollute the global namespace
- To create top-level variables for other scripts, attach them as properties on `window`, or to `exports` in CommonJS. Use: `exports ? this`

## Splats

- Splats `...` can be used instead of the variable number of `arguments` object and are available for both function definition and invocation

## Loops and Comprehensions

- Comprehensions *for* `...` *in* work over arrays, objects, and ranges
- Comprehensions replace for loops, with optional *when* guard clauses and the value of the current array index: *for* `value, index in array`
- Array comprehensions are expressions, and can be returned and assigned
- Comprehensions may replace *each*/*forEach*, *map* or *select*/*filter*
- Use a range when the start and end of a loop is known (integer steps)
- Use *by* to step in fixed-size increments
- When assigning the value of a comprehension to a variable, CoffeeScript collects the result of each iteration into an array
- Return *null*, *undefined* or *true* if a loop is only for side-effects
- To iterate over the key and value properties in an object, use *of*
- Use: *for own* `key, value` *of* `object` to iterate over the keys that are directly defined on an object
- The only low-level loop is the *while* loop. It can be used as an expression, returning an array containing the result of each iteration through the loop
- *until* is equivalent to *while not*
- *loop* is equivalent to *while true*
- The *do* keyword inserts a closure wrapper, forwards any arguments and invokes a passed function

## Try/Catch/Finally

- *try*/*catch* statements are as in JavaScript (although expressions)

## If, Else, Unless, and Conditional Assignment

- *if*/*else* can be written without parentheses and curly braces
- Multi-line conditionals are delimited by indentation
- *if* and *unless* can be used in postfix form i.e. at the end of the statement
- *if* statements can be used as expressions. No need for `?:`

## Chained Comparisons

- Use a chained comparison to test if a value is within a range:
  `minimum < value < maximum`

## Array Slicing and Splicing with Ranges

- Ranges can be used to extract slices of arrays
- With two dots `[3..6]`, the range is inclusive $(3, 4, 5, 6)$
- With three dots `[3...6]`, the range excludes the end $(3, 4, 5)$
- The same syntax can be used with assignment to replace a segment of an array with new values, splicing it
- Strings are immutable and can not be spliced

## Embedded JavaScript

- Use backquotes `` to embed JavaScript code within CoffeeScript

Figure B.1: CoffeeScript Quick Reference 1 reproduced from Hoigaard (2011)

### Everything is an Expression

- Functions return their final value
- The return value is fetched from each branch of execution
- Return early from a function body by using an explicit `return`
- Variable declarations are at the top of the scope, so assignment can be used within expressions, even for variables that have not been seen before
- Statements, when used as part of an expression, are converted into expressions with a closure wrapper. This allows assignment of the result of a comprehension to a variable
- The following are not expressions: `break`, `continue`, and `return`

### Operators and Aliases

- CoffeeScript compiles `==` into `===`, and `!=` into `!==`. There is no equivalent to the JavaScript `==` operator
- The alias `is` is equivalent to `===`, and `isnt` corresponds to `!==`
- Logical operator aliases: `and` is `&&`, `or` is `||` and `not` is an alias for `!`
- In `while`, `if/else` and `switch/when` statements the `then` keyword can be used to keep the body on the same line
- Alias for boolean `true` is `on` and `yes` (as in YAML)
- Alias for boolean `false` is `off` and `no`
- For single-line statements, `unless` can be used as the inverse of `if`
- Use `@property` or `@method` instead of `this.something`
- Use `in` to test for array presence
- Use `of` to test for object-key presence

### Existential Operator

- Use the existential operator `?` to check if a variable exists. `?` returns `true` unless a variable is `null` or `undefined`
- Use `?=` for safer conditional assignment than `||=` with numbers or strings
- The accessor variant of the existential operator `?.` can be used to soak up null references in a chain of properties
- Use `?.` instead of the dot accessor `.` in cases where the base value may be `null` or `undefined`. If all of the properties exist then the expected result is returned, if the chain is broken, then `undefined` is returned instead

### Classes, Inheritance, and Super

- Object orientation as in most other object oriented languages
- The `class` structure allows to name the class, set the superclass with `extends`, assign prototypal properties, and define a `constructor`, in a single assignable expression
- Constructor functions are named as the `class` name, to support reflection
- Lower level operators: The `extends` operator helps with proper prototype setup. `::` gives access to an object's prototype. `super()` calls the immediate ancestor's method of the same name
- A class definition is a block of executable code, which may be used for meta programming.
- In the context of a class definition, `this` is the class object itself (the `constructor` function), so static properties can be assigned by using `@property: value`, and functions defined in parent classes can be called with: `@inheritedMethodName()`

### Destructuring Assignment

- To make extracting values from complex arrays and objects convenient, CoffeeScript implements destructuring assignment
- When assigning an array or object literal to a value, CoffeeScript breaks up and matches both sides against each other, assigning the values on the right to the variables on the left
- The simplest case is parallel assignment `[a,b] = [b,a]`
- It can be used with functions that return multiple values
- It can be used with any depth of array and object nesting to get deeply nested properties and can be combined with splats

### Function binding

- The fat arrow `=>` can be used to define a function and bind it to the current value of `this`
- This is helpful when using callback-based libraries, for creating iterator functions to pass to `each` or event-handler functions to use with `bind`
- Functions created with `=>` are able to access properties of the `this` where they are defined

### Switch/When/Else

- The `switch` statement do not need a `break` after every case
- A `switch` is a returnable, assignable expression
- The format is: `switch` condition, `when` clauses, `else` the default case
- Multiple values, comma separated, can be given for each `when` clause. If any of the values match, the clause runs

### String Interpolation, Heredocs, and Block Comments

- Single-quoted strings are literal. Use backslash for escape characters
- Double-quoted strings allow for interpolated values, using `#{ ... }`
- Multiline strings are allowed
- A heredoc `'''` can be used for formatted or indentation-sensitive text (or to avoid escaping quotes and apostrophes)
- The indentation level that begins a heredoc is maintained throughout, so the text can be aligned with the body of the code
- Double-quoted heredocs `"""` allow for interpolation
- Block comments `###` are similar to heredocs, and are preserved in the generated code

### Extended Regular Expressions

- Extended regular expressions are delimited by `///` and are similar to heredocs and block comments.
- They ignore internal whitespace and can contain comments

### Aliases

```
and : &&     or  : ||    not : !
is  : ==     isnt: !=
yes : true   no  : false
on  : true   off : false
```

### Miscellaneous

- Twitter comments to @autotelicum

http://autotelicum.github.com/Smooth-CoffeeScript/

Figure B.2: CoffeeScript Quick Reference 2 reproduced from Hoigaard (2011)

# Appendix C

# XACML Glossary

The Glossary from Moses (2005) is reproduced in Figure C.1 and Figure C.2.

227

228 # 1. Introduction (non-normative)

229 ## 1.1. Glossary

230 ### 1.1.1 Preferred terms

231 ***Access*** - Performing an ***action***

232 ***Access control*** - Controlling ***access*** in accordance with a ***policy***

233 ***Action*** - An operation on a ***resource***

234 ***Applicable policy -*** The set of ***policies*** and ***policy sets*** that governs ***access*** for a specific
235 ***decision request***

236 ***Attribute*** - Characteristic of a ***subject***, ***resource, action*** or ***environment*** that may be referenced
237 in a ***predicate*** or ***target*** (see also – ***named attribute***)

238 ***Authorization decision*** - The result of evaluating ***applicable policy,*** returned by the ***PDP*** to the
239 ***PEP.*** A function that evaluates to “Permit”, “Deny”, “Indeterminate” or “NotApplicable", and
240 (optionally) a set of ***obligations***

241 ***Bag*** – An unordered collection of values, in which there may be duplicate values

242 ***Condition -*** An expression of ***predicates.*** A function that evaluates to "True", "False" or
243 “Indeterminate”

244 ***Conjunctive sequence*** - a sequence of ***predicates*** combined using the logical ‘AND’ operation

245 ***Context -*** The canonical representation of a ***decision request*** and an ***authorization decision***

246 ***Context handler -*** The system entity that converts ***decision requests*** in the native request format
247 to the XACML canonical form and converts ***authorization decisions*** in the XACML canonical form
248 to the native response format

249 ***Decision –*** The result of evaluating a ***rule, policy*** or ***policy set***

250 ***Decision request*** - The request by a ***PEP*** to a ***PDP*** to render an ***authorization decision***

251 ***Disjunctive sequence*** - a sequence of ***predicates*** combined using the logical ‘OR’ operation

252 ***Effect -*** The intended consequence of a satisfied ***rule*** (either "Permit" or "Deny")

253 ***Environment*** - The set of ***attributes*** that are relevant to an ***authorization decision*** and are
254 independent of a particular ***subject, resource*** or ***action***





Figure C.1: XACML Access Control Glossary 1, reproduced from Moses (2005)

***Named attribute –*** A specific instance of an ***attribute***, determined by the ***attribute*** name and type, the identity of the ***attribute*** holder (which may be of type: ***subject***, ***resource***, ***action*** or ***environment***) and (optionally) the identity of the issuing authority

***Obligation*** - An operation specified in a ***policy*** or ***policy set*** that should be performed by the ***PEP*** in conjunction with the enforcement of an ***authorization decision***

***Policy -*** A set of ***rules,*** an identifier for the ***rule-combining algorithm*** and (optionally) a set of ***obligations.*** May be a component of a ***policy set***

***Policy administration point (PAP)*** - The system entity that creates a ***policy*** or ***policy set***

***Policy-combining algorithm*** - The procedure for combining the ***decision*** and ***obligations*** from multiple ***policies***

***Policy decision point (PDP) -*** The system entity that evaluates ***applicable policy*** and renders an ***authorization decision***. This term is defined in a joint effort by the IETF Policy Framework Working Group and the Distributed Management Task Force (DMTF)/Common Information Model (CIM) in [RFC3198]. This term corresponds to "Access Decision Function" (ADF) in [ISO10181-3].

***Policy enforcement point (PEP)*** - The system entity that performs ***access control***, by making ***decision requests*** and enforcing ***authorization decisions***. This term is defined in a joint effort by the IETF Policy Framework Working Group and the Distributed Management Task Force (DMTF)/Common Information Model (CIM) in [RFC3198]. This term corresponds to "Access Enforcement Function" (AEF) in [ISO10181-3].

***Policy information point (PIP)*** - The system entity that acts as a source of ***attribute*** values

***Policy set*** - A set of ***policies,*** other ***policy sets,*** a ***policy-combining algorithm*** and (optionally) a set of ***obligations.*** May be a component of another ***policy set***

***Predicate -*** A statement about ***attributes*** whose truth can be evaluated

***Resource*** - Data, service or system component

***Rule -*** A ***target***, an ***effect*** and a ***condition.*** A component of a ***policy***

***Rule-combining algorithm -*** The procedure for combining ***decisions*** from multiple ***rules***

***Subject -*** An actor whose ***attributes*** may be referenced by a ***predicate***

***Target -*** The set of ***decision requests***, identified by definitions for ***resource***, ***subject*** and ***action***, that a ***rule***, ***policy*** or ***policy set*** is intended to evaluate

***Type Unification*** - The method by which two type expressions are "unified". The type expressions are matched along their structure. Where a type variable appears in one expression it is then "unified" to represent the corresponding structure element of the other expression, be it another variable or subexpression. All variable assignments must remain consistent in both structures. Unification fails if the two expressions cannot be aligned, either by having dissimilar structure, or by having instance conflicts, such as a variable needs to represent both "xs:string" and "xs:integer". For a full explanation of ***type unification***, please see [**Hancock**].





Figure C.2: XACML Access Control Glossary 2, reproduced from Moses (2005)