



Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin

Copyright statement

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

Liability statement

By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

Access Agreement

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Policy Based Management for Dynamic Organisations

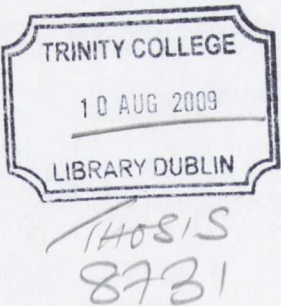
A thesis submitted to
the University of Dublin, Trinity College,
in fulfilment of requirements for the degree of
Doctor of Philosophy
January 2009

Kevin Chekov Feeney

Declaration

I, the undersigned, declare that this work has not been previously submitted as an exercise for a degree at this or any other University, and that, unless otherwise stated, it is entirely my own work.


Kevin C. Feeney



Permission to lend or copy

I, the undersigned, agree that the Trinity College Library may lend or copy this thesis upon request.



Acknowledgements

Firstly, I must thank my supervisor Vincent Wade, for the benefit of his clear-headed advice and practical experience throughout the work. I must also thank all the members of the Knowledge and Data Engineering Group, (KDEG) in Trinity College Dublin, which has proved an open and stimulating environment for the exploration of new ideas. Special thanks to Dr. David Lewis for his tremendous support, intellectual and practical, throughout the work.

Extra special thanks to Ciara for all her support and for so gracefully putting up with my pre-occupied, impoverished and frankly dishevelled state for such a long time during the production of this thesis. Similarly, all those members of my family and friends who have put up with my preoccupied state without irretrievably falling out with me deserve praise.

The inspiration for this work stems from the observation of and the participation in a wide range of voluntary, self-managed communities, both online and in the real world. My thanks and praise to all those who have put their time and energy into the creative task of organising those communities who give us free software and have enormously increased our collective access to information. The Indymedia Network, some of whose members participated in an experiment as part of this work, deserves special mention.

Abstract

Policy-Based Management (PBM) is a recent development in systems and network management research which aims to address the challenge of efficiently managing complex and heterogeneous information systems. Broadly speaking, the goal of policy based management is to enable information systems to be adaptive and to dynamically change their behaviour during operation in order to reflect the high-level business goals of the organisation. Although current PBM systems contain elements of an organisational model, their support for highly-dynamic organisations and devolved decision making is limited. This thesis proposes the Community Based Policy Management System (CBPMS), a system consisting of a formal model of the organisation, a clear definition of its state transitions and a service oriented architecture for deployment. The architecture and implementation of the CBPMS is tested by applying it to the management of information resources in several different domains, through practical experiments. The experiments are augmented by a case study in order to illustrate the capacity of the model to capture the operations of a wide range of organisations operating in dynamic environments. Finally, the system is compared experimentally to a role-based model, with results which suggest that there may be general, concrete, practical advantages to the CBPMS approach.

Table of Contents

<i>Policy Based Management for Dynamic Organisations</i>	<i>i</i>
<i>Acknowledgements</i>	<i>iv</i>
<i>Abstract</i>	<i>v</i>
<i>Table of Contents</i>	<i>vi</i>
<i>List of Figures</i>	<i>x</i>
Chapter 1. Introduction	1
1.1 General Motivation	1
1.2 Research Goals	5
1.3 Contribution	6
1.4 Technical Approach	7
Chapter 2. Background & State of the Art Review	10
2.1 Introduction	10
2.2 Organisational Modelling	10
2.2.1 Organisation in Multi-Agent Systems	14
2.3 Policy Based Management	16
2.3.1 Policy Deployment	17
2.3.2 Policy Specification	17
2.3.3 Mapping Management Goals to Concrete Policies	19
2.3.4 Policy Conflicts	20
2.3.5 From Management Goals to Organisational Goals	23
2.3.6 Evaluation Criteria	26
2.4 Organisational Models in Access Control Systems	28
2.4.1 Mandatory & Discretionary Access Control	29
2.4.2 Role Based Access Control	30
Summary of Organisational Models in RBAC	32
2.5 Practical Policy Based Management Systems	33
2.5.1 IETF Policy Model	34
2.5.2 Policy Description Language	37
2.5.3 Open Distributed Programming Reference Model (ODP-RM)	39
2.5.4 PONDER	40
2.5.4.1 Ponder's Organisational Model	46
2.5.5 Open Architecture for Secure Interworking Services	49
2.5.6 XACML	51
2.5.7 Other PBM systems.	55
2.6 Human Interaction with policy based management	56
2.7 Summary	57
Chapter 3. Community Based Policy Management – Theoretical Model	61
3.1 Introduction & Background	61
3.2 Informal Description	67
3.2.1 The Community	68
Community Federations	74
3.2.3 Modelling Resources	76

3.2.4 Community Membership	78
3.2.5 Community Policy	79
3.2.6 Steps Required to Model an Organisation	80
3.2.7 Dynamic Aspects – Spawning, Delegation and Escalation	80
3.3 Formal Definition	82
3.3.1 Communities and their relationships	83
3.3.2 Resource Specification	85
3.3.3 Community State Specification	86
3.3.4 Community Policies, Policy Reasoning and Context	87
3.3.4 Overview of the CBPM Model	94
3.3.5 Policy Decision Algorithm	95
3.4 Conflict Resolution	97
3.5 CBPMS Administration	100
3.5.1 CBPMS Management Primitives	100
3.5.2 Community Hierarchy Resource Model	102
3.6 Conclusion	103
Chapter 4. CBPMS Architecture and Implementation	105
4.1 Conceptual Architecture	105
4.1.1 State Management – Community & Resource Repositories	107
4.1.2 Serialised Community Structure	107
4.1.3 CBPMS Service Based Architecture	109
4.2 RAIS - Resource Authority Information Service	110
4.3 CRMS - Community Record Management Service	112
4.3.1 Interactions between CRMS and CPDS	112
4.3.2 Interactions between CRMS and CPMS	113
4.4 CPDS Community Policy Decision Service	115
4.4.1 CMDS - Context Mapping & Discovery Service	116
4.4.2 PRS - Policy Reasoning Service	118
4.4.3 CPSS - Community Policy Selection Service	119
4.4.4 Community Policy Management Service & User Interface	121
4.5 Full Services API	121
4.6 Full Implementation with XACML	122
4.6.1 Community Record Management Service	123
4.6.2 CRMS Distributed Architecture	124
4.6.3 Resource Authority Information Service	125
4.6.4 Context Mapping & Discovery Service	127
4.6.5 Community Policy Selection Service	128
4.6.6 Policy Reasoning Service	129
4.6.7 Community Policy Management System	130
4.7 Community Policy Management - User Interface	132
4.7.1 Authentication and community membership control.	133
4.7.2 Community Management Overview	133
4.7.3 Community Management Screen	135
4.7.5 Policy Management	138
4.7.6 Community Structure Visualisation	142
4.8 Summary	144
Chapter 5 Case Studies, Experiments and Results	145
5.1 Introduction	145
5.2 Experiment 1: Physical Security Management in a Ubiquitous Computing Environment	146

5.2.1 Scenario	146
5.2.2 Experiment Environment	147
5.2.3 Creating the CBPMS Models	151
5.2.4 Running the Experiment	155
5.2.5 Results	156
5.3 Constitutional Modelling Case Study	160
5.3.1 Direct Democracy	161
5.3.2 Formation of the Executive and Judicial Branches	162
5.3.3 The Constitution and Separation of Civil and Criminal Law	164
5.3.4 Legislative Bodies	165
5.3.5 Legislative Dominance	165
5.3.6 Regional and Local Organisation	166
5.3.7 Federal Organisations	167
5.3.8 Summary	169
5.4 Experiment 2 – Modelling Internet Communities	169
5.4.1 Oscailt	169
5.4.2 Community Model	171
5.4.4 Community Model Implementation	172
5.4.5 Community Evolution	175
5.4.6 Summary	177
5.5 Experiment 3 - Comparing Communities and Roles	177
5.5.1 Scenario	177
5.5.2 Constructing the Community Model	179
5.5.3 Simulation Design	183
5.5.4 Simulation – First Run	187
5.5.5 Simulation – Second Run	190
5.5.6 Simulation – Third Run	192
5.5.7 Simulations – Summary	194
Chapter 6. Conclusions and Further Work	195
6.1 Analysis of Research Goals	195
Goal 1: CBPMS Development	195
Goal 2: Model Generality, Flexibility, Dynamism and Decentralisation	197
Goal 3: CBPMS Advantages	201
6.2 Contribution	203
CBPM Model	203
Analysis of Modelling Approaches	203
6.2.1 Minor Contributions	204
Novel Conception of Delegation	204
Distributed Service Oriented Policy Architecture	204
PBM Metrics	205
6.3 Further Work	205
6.3.1 Resource Models and Ontologies	205
6.3.2 Integration of Multiple Policy Languages	206
6.3.3 Policy Authoring Dialectics	206
6.3.4 Realizing Decentralised Policy Authoring	207
6.4 Conclusion	208
Bibliography	209
Prior Publications	225
Appendix 1: Organisational Modelling & Access Control	227
A1.1 Mandatory Access Control	227
A1.2 Discretionary Access Control	228

A1.3 Other Traditional Models	232
A1.4 Role Based Access Control	232
A1.5 ANSI Standard RBAC Models	233
Core RBAC	233
Hierarchical RBAC	234
Constrained RBAC	235
A1.6 RBAC Administration	236
Role Delegation	238
A1.7 RBAC's Organisational Model	238
Role Hierarchies	239
A1.8 RBAC – Extensions & Variations	242
RB-RBAC – Rule Based RBAC	243
TBAC – Task Based Access Control	243
TRBAC – Temporal Role-based Access Control Model	243
TMAC – Team Based Access Control	244
CBAC – Coalition Based Access Control	245
OrBAC – Organisation Based Access Control	245
A1.9 Access Control Policy Specification Languages	247
A1.10 Conclusion	249
<i>Appendix A2. CBPMS Decision Algorithm</i>	252
<i>Appendix A3. CBPMS Primitive Definitions</i>	258
<i>Appendix A4: Community Resource Modelling</i>	266
<i>Appendix A6. Isancestor Function</i>	277
<i>Appendix A7: CBPMS XML Schema</i>	278
<i>Appendix A8. Sessions and Credentials</i>	284
<i>Appendix A9: Oscailt Evaluation</i>	288

List of Figures

Figure 1.1 High Level Operation of Policy Based Management	1
Figure 2.1 Ontology of organisational concepts present in MAS models.	15
Figure 2.2 Strassner's Policy Continuum Model	20
Figure 2.3 The Brittle Nature of One-Off Requirements Engineering for PBM	24
Figure 2.4 Using a dynamic organisational model to enable limited self-management	25
Figure 2.5 Organisational Models Embodied in AC Systems	27
Figure 2.6 IETF Policy Architecture	34
Figure 2.7 Ponder Domain Structure Example	42
Figure 2.8 Ponder Composite Policy Object Class Relationships	45
Figure 2.9 XACML Data-Flow Model	53
Figure 2.10 XACML Policy Model	54
Figure 3.1 Organisation as a Hierarchy of Authority	63
Figure 3.2 Decision Making Hierarchy	64
Figure 3.3 Delegation / Escalation Cycle	66
Figure 3.4 Segment of Simple Organisation Chart in terms of Organisational Units	67
Figure 3.5 Hierarchy of Authority with Decision Making Units Highlighted	70
Figure 3.6 Authority Relationship of Organisational Units to Resource	72
Figure 3.7 Simple Schematic of Organisational Units' Authority over Resources in a Federation	74
Figure 3.8 Example of Partial CBPM Resource Target Tree Representing a Database Server	77
Figure 3.9 Example Resource Action Tree for Database Resource	77
Figure 3.10 UML class relationships of CBPMS model	94
Figure 3.11 CBPMS Management Primitives Invoked in Sequence	101
Figure 3.12 CBPMS Hierarchical Resource Model Representing the Community Hierarchy Itself	103
Figure 4.1 CBPMS Community Policy Decision Service Architecture	106
Figure 4.2 CBPMS Policy Management Service Architecture	107
Figure 4.3 CBPMS Community Record Structure	108
Figure 4.4 Community Based Policy Management System (CBPMS) – Service Architecture	109
Figure 4.5 CPDS architecture	115
Figure 4.6 APIs & Interactions between Component Services of Community Policy Decision Service ..	122
Figure 4.7 Interactions between Component Services of CBPMS	122
Figure 4.8 Distributed CRMS Architecture	125
Figure 4.9 Resource Authority Information Service Design	126
Figure 4.10 Context Mapping & Discovery Service Design	127
Figure 4.11 Policy Reasoning Service (PRS) Implementation	129
Figure 4.12 CBPMS Community Management System	134
Figure 4.13 CBPMS Community Management System – Community Management Screen	136
Figure 4.14 CBPMS Community Management System – Resource Authority Trees	137
Figure 4.15 CBPMS Policy Management Screen	138
Figure 4.16 CBPMS Community Management System – XACML Policy Definition Screen	139
Figure 4.17 CBPMS XACML / PHP Condition Definition Screen	141
Figure 4.18 CBPMS Community Management System – Community Resource Model	142
Figure 4.19 CBPMS Community Management System – Limited Delegation of Self-Management	144
Figure 5.1 Screenshot from Simulation of Trinity's Lloyd Building and actual photo	148
Figure 5.2 Screenshot of IM Application	150
Figure 5.3 Architecture for CBPMS integration experiment	151
Figure 5.4 Resource Models Employed (screenshots from CBPMS GUI)	153
Figure 5.5 Segment of Community Model for Integration Experiment	154
Figure 5.6 Voting Resource Authority Tree Model	161
Figure 5.7 Segment of Sample State Community Structure	167
Figure 5.8 Segment of EU Federal Structure	168
Figure 5.9 EU structure with institutions	169
Figure 5.10 Indymedia Community Model	172
Figure 5.11 Oscailt Community Structure and Resource Target & Action Trees (montage of screenshots from GUI)	173
Figure 5.12 Updated Indymedia Community Model	176
Figure 5.13 RBAC / ARBAC role hierarchy (higher roles inherit lower roles)	178
Figure 5.14 Equivalent Community Hierarchy	180

Figure 5.15 Simple Resource Model used in simulation experiment 185

Figure 5.16 Community Hierarchy for Comparison 185

Figure 5.17 Estimated expensiveness of CBPMS and RBPMS with respect to the number of policies... 189

Figure 5.18 Simulation 1: Frequency of policy conflicts in RBPMS and CBPMS 189

Figure 5.19 Results of Second Simulation – policy evaluations required 191

Figure 5.20 Frequency of Policy Conflicts in Second Simulation 191

Figure 5.21 Results for third simulation – policy evaluations required 193

Figure 5.22 Frequency of Policy Conflicts in third simulation 193

Figure A1.1 Core RBAC model 234

Figure A1.2 Hierarchical RBAC Model 235

Figure A1.3 Constrained RBAC Model 236

Figure A1.4 ARBAC97 Administrative Model 237

Figure A1.5 Hierarchical RBAC Example 239

Figure A1.6 RBAC Private Role Hierarchy 240

Figure A4.1 CBPMS Resource Tree & Section of Action Tree 266

Figure A4.2 Target Tree for Self-Managed Community Example 266

Figure A4.3 CBPMS Resource Target Tree & Section of Action Tree 268

Figure A5.1 Using Federations to Represent Cross-Organisational Links 273

Chapter 1. Introduction

1.1 General Motivation

Policy-Based Management (PBM) is a recent development in systems and network management research which aims to address the challenge of efficiently managing complex and heterogeneous information systems. Broadly speaking, the goal of policy based management is to enable information systems to be adaptive and to dynamically change their behaviour during operation in order to reflect the high-level business goals of the organisation [slooman02]. PBM solutions, taken to their logical conclusion, promise a future where human decision makers can concentrate on defining high level goals and, once decided, have these goals automatically translated into appropriate configurations on managed information resources across the entire enterprise [damianou02b].

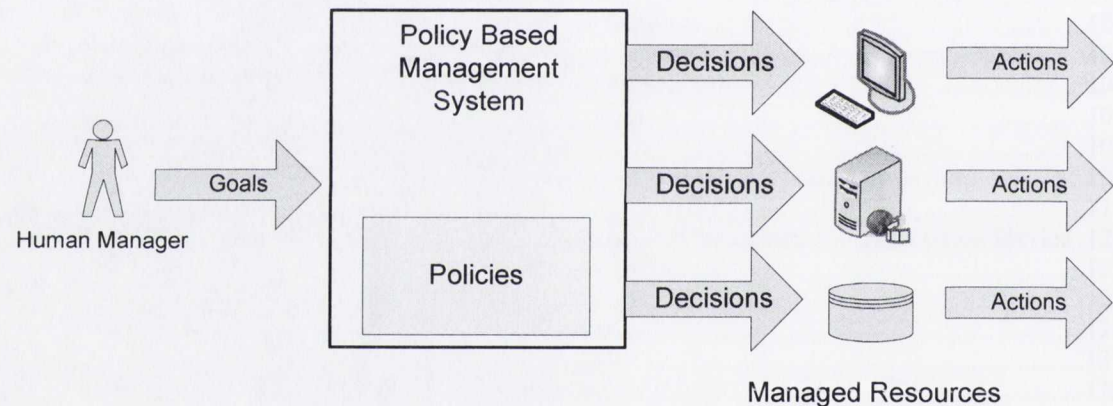


Figure 1.1 High Level Operation of Policy Based Management

The goal of a PBM system, as depicted in Figure 1.1, is to translate an organisation's management goals into decisions about the behaviour of its managed resources in such a way as to ensure that the behaviour of the managed resources reflects the management goals, even as those goals change. Policies, frequently defined as *rules specifying choices in the behaviour of a system* [dulay01, damianou02b], can be broadly subdivided into two major categories. *Authorisation policies* [moffet90], which can be positive or negative [lupu99, dunlop03], are used to control access to the managed resources, while *obligation policies* [marriott96, gama05] are used to specify what actions managed resources must take in particular contexts.

In order for a PBM system to fulfil its function, it needs to have access to information about the context in which it is deployed. It needs to know what resources it is

responsible for managing; it needs to know how to capture management goals; it needs to know which users or software agents should be permitted to access the managed resources and in which contexts they can do so. This information is required to translate management goals into behaviour that is most likely to achieve those goals. Thus, PBM systems incorporate ways of representing information about users, resources and administrators within the system.

The increasing complexity of information systems and the organisations within which they are deployed is a fundamental challenge driving research into PBM. One basic requirement of PBM systems is that they allow policies to be specified that apply to classes of entities, in order to allow decision makers to apply policies to large numbers of entities in a way that is relatively efficient to maintain. Thus, PBM systems include various abstractions, which can be used to group related resources, users, administrators and other significant entities in the system. This allows policies to be specified in such a way that they apply to classes of entities.

The problem of identifying useful grouping abstractions to model managed information resources within an organisation is relatively straightforward at first sight. The utility of grouping together systems with similar capabilities and functions is immediately obvious – for example creating a *database* class as an abstraction for grouping together policies related to databases. However, there are many other types of useful grouping abstraction which can be used to ease the administration of managed resources; according to the department that they are deployed within, for example. Thus PBM systems often use both resource classes and hierarchical *domains* in order to capture resource and management groupings [yialelis96, eyers06]. Virtually all current PBM systems use the *role* as the primary grouping abstraction for modelling human users of the system [sandhu00]. Administration of PBM systems is similarly modelled through administrative roles [sandhu99b], which group together policies related to the users who specify the policies and manage the models of the PBM system.

The models of the resources, users and administrators, combined with the set of policies, represent the PBM system's internal model of the organisation. Based upon this model, and whatever contextual information is available to it, the PBM must choose which decision will further the organisation's goals when a choice in the system's behaviour is encountered in any give context. Thus, a basic requirement of the model is that it is rich enough to capture those elements of the organisation's structure that are

required for the PBM to arrive at the best decision from the point of view of achieving the organisation's high level goals in any given context. However, modern understandings of human organisations reveal that they have several characteristics which are particularly difficult to capture in a policy-model.

- **Continual Change.** Organisations undergo continual change. Most of the time, the change is incremental. At other times, the change is more radical and manifests itself as a management action. [wickham04, mackenzie07]. This continual change affects not only business goals, but also structures, subdivisions of authority, distribution of responsibility and decision making. The proliferation of networked information systems has created a variety of new organisational paradigms where rapid and continual change is particularly important. Internet communities, SME value chains and virtual organisations and ubiquitous computing services are all areas in which organisational dynamism and flexibility are a necessary part of the paradigm.
- **Formal, Informal and Devolved Decision Making.** Modern understandings of human organisations highlight the fact that decision-making is not the straightforward hierarchical, top-down process that organisational charts describe. In practice, rather than simply obeying rules, individuals throughout the organisational hierarchy often take decisions within their own domains and have a crucial role in feeding back information about problems in organisational policy to the higher decision-making bodies [thompson02, barrett04]. While decision-making might be a decentralised, collaborative activity, decisions taken at the lower levels of the organisation's hierarchy are bound by the fact that their authority has limited scope and formal hierarchies still allow senior decision makers, who are in a better position to take high-level decisions more closely related to the strategic goals of the organisation, to over-rule the decisions of those beneath them in the hierarchy who may be focused on less important sub-goals. They are also in a position to take decisions to resolve conflicts based upon conflicting demands from people beneath them in the hierarchy [mcsweeney06, hopfl06].
- **Incomplete Specification.** Research into human interaction with policy systems has revealed that humans tend to progressively accumulate policy rules as new situations arise. That is to say, that rather than exhaustively describing the policy

rules for all possible situations, people tend to define general rules which provide boundaries for decisions and progressively add detail to these rules over time as they encounter situations which require the general rules to be refined, a process termed *progressive grounding* in [barrett04].

Limitations of Current PBM Approaches

Current PBM approaches are limited in their capacity for modelling human organisations, in particular the evolving, decentralised decision-making hierarchies and progressive approach to policy definition. While a Role Based Access Control (RBAC) model has been standardised to include hierarchies which can form a basic hierarchical organisational model [sandhu00], its application to human organisations remains problematic. In particular, attempts to define hierarchical relationships between roles have proved insufficient to deal with the complex interrelationships between individuals within human organisations [moffett98, goh98]. Role inheritance is based upon the concept of specialisation and generalisation, as used in object oriented programming, and is not well suited for modelling structural hierarchies, supervision hierarchies or the other hierarchies that exist within organisations [moffett98]. There have been several proposed extensions and alternatives to the RBAC models which have augmented them by adding extra abstractions such as *teams* [thomas97a, alotaiby04], *coalitions* [cohen02] and *organisations* [miege05] to enable the model to capture the organisation's structure more accurately.

The authoring of policy rules in a PBM system amounts to making decisions on behalf of the organisation. Rather than being distributed throughout the organisation, PBM systems normally vest decision-making power in specialised administrators and it is their duty to ensure that the policies reflect the goals of the organisation. Although some PBM systems provide some support for delegation, it is conceived as a temporary transfer of rights from one individual to another, which amounts to a very limited means of decentralising decision making and this mechanism is not sufficient to capture the permanent, structural nature of decentralised collaborative decision making in real world organisations [feeney07a, feeney07b].

Finally, rather than building up the organisational model through progressive grounding, PBM systems generally depend upon a lengthy and detailed period of analysis before they are deployed within complex organisations [crook02] using techniques borrowed

from software engineering [moffett98] to attempt to exhaustively define policies, roles and other groupings. Such an approach is expensive and brittle [jude01], as any major changes to the organisation's structures or goals may require repeated extensive analysis to ensure that the model still properly encapsulates the requirements.

1.2 Research Goals

This thesis sets out to examine the problem of modelling organisations within PBM systems. Starting from an analysis which concludes that current approaches to modelling organisations within PBM systems either lack the facility to incorporate an integrated, dynamic and flexible model of the organisation into the system, or provide little structural support for the controlled development of such a model, the following thesis statement is adopted:

This thesis aims to devise and develop a model of the organisation that can be incorporated into PBM systems in order to facilitate the management of the progressive changes to organisational structures and goals that are a common feature of evolving organisations operating in complex and heterogeneous environments. This research aims to facilitate this management through the incorporation of constraints into the organisational model that allow policy-authoring to be decentralised in a controlled way. The model's constraints serve to limit the potential consequences of changes to the model, thereby reducing the complexity of each change and facilitating the management of the system's evolution over time. This statement can be broken down into three distinct sub-goals which need to be achieved in order to validate the research and to demonstrate the truth of the basic hypothesis and the assumption on which the work is based. The sub-goals are:

1. To devise and develop a formal organisational model for PBM, defined in non-ambiguous terms, and to demonstrate that the model can be integrated into an organisation's IT systems to deliver a PBM capability.
2. To demonstrate that the model is capable of capturing a wide range of organisational forms and that it is capable of efficiently capturing organisational change as well as being capable of modelling the decentralised decision making hierarchies that exist in real world organisations.

3. To show the practical advantages over current systems, in terms of reduced complexity in managing change. This reduced complexity will be demonstrated in two ways:
 - a) The organisational model will allow policy conflicts to be automatically resolved, or, where that is impossible, it will enable the automatic identification of the organisational decision making authority that must address the problem.
 - b) The information embodied in the organisational model will reduce the number of policy rules that must be evaluated for any given context. This will reduce the cognitive load of policy engineering as well as limiting the scope of policy searches.

1.3 Contribution

This thesis describes a novel theoretical and practical framework for the modelling of organisations for the purpose of managing their interactions with policy based management solutions. This approach, termed *Community Based Policy Management* (CBPM), is based upon a theoretical conception of the organisation that is powerful enough to model a wide range of organisational structures, and is flexible and expressive enough to efficiently model organisational change in environments characterised by flexibility, dynamism, heterogeneity of organisational structures and where decision making authority may be distributed through the organisation. The CBPM model of the organisation allows the system to automatically identify the sources of conflict within the organisation and to automatically resolve many of these conflicts to reflect the higher-level goals of the organisation. The top-down, dynamic, evolutionary approach to organisational modelling that is facilitated by the system can reduce the complexity of requirements engineering that must be carried out before a Policy Based Management system is deployed within an organisation. The CBPM model, and the demonstration of the advantages – automatic conflict resolution and reduced policy search space - that it offers, is the main contribution of this work. However, this thesis also contains several other novel contributions to the field.

- A conception of delegation, novel in PBM, not as a temporary transfer of permissions, but as a transfer of authority from more general levels of an organisation to more specific levels, a transfer of authority which forms the basis of the organisation's internal distribution of rights and privileges. This concept

of delegation is counter-posed with the concept of escalation, whereby problems encountered by low-level decision makers are escalated to the higher level decision makers who have authority to solve them, to form the delegation-escalation cycle, a novel approach to progressively grounding the policy set and refining it towards a more stable configuration over time.

- A formally defined set of policy language semantic requirements for the integration of any given policy language into the CBPM framework. This presents the possibility of integrating multiple policy specification languages into a unified PBM framework, with precise and well-defined semantics.
- A service oriented architecture, potentially supporting distributed, federated server deployment, spanning multiple organisations and integrating them all into a unified management framework.
- A novel empirical approach to the assessment, evaluation and comparison of the complexity of applying changes to organisational models within PBM systems, based upon the probability of irresolvable policy conflicts and the size of the policy search space. This measure is significant as management complexity is an important practical concern motivating PBM research, yet there are currently no proposed approaches to evaluating it empirically.

1.4 Technical Approach

The basic argument of this thesis is that a CBPM model of an organisation offers advantages over existing Role-Based Policy Management (RBPM) models when dealing with real world organisations. However, it is essentially impossible to demonstrate this in general since the number of possible organisational requirements is intractable. It is even extremely difficult to directly compare the benefits of a CBPM approach when compared to a RBPM approach in practice. Even if a large, complex, dynamic organisation was found which had used both models to manage their information systems, it would be invalid to assume that any differences in performance measures such as user satisfaction could be attributed to the different organisational models used. PBM systems have no standard interface and their feature sets vary widely and any of the differences have the potential to affect the outcome. Therefore, although an experiment which introduced CBPM to a small group which had formerly used a RBM system was carried out as part of this research, from an appreciation of the

above problems, the experiment did not aim to infer any firm conclusions about the relative benefits of the underlying models in use from such limited and unreliable data.

This thesis does not, therefore, set out primarily to demonstrate the superiority of the CBPM approach over RBPM. Rather, it sets out to show the viability of the CBPM system, and to demonstrate that it can be applied successfully to a broad range of dynamic organisations. This is demonstrated by constructing a series of case-studies and experiments, designed to test the capabilities of the model against diverse examples of real world organisations. The comparison between CBPM and RBPM is limited to a specific area, at a much lower, and much more easily measured, level. The experiment in question, which deals with the frequency of conflicts and the size of the policy search space between a standard role-model and an equivalent community model, was devised in order to validate, in as concrete a manner as possible, the existence of practical advantages as a consequence of the organisational model that is embodied in the CBPM system. Therefore, although the complexity of the problem domain renders it practically impossible to prove that a CBPM system will lead to superior outcomes than a RBPM system in general, it is possible to show that the CBPM system does offer measurable advantages in certain areas and make a strong argument that these advantages are likely to lead to superior outcomes in practice, and that is what this thesis sets out to do.

Chapter 2 examines current understandings of organisational theory in order to identify the factors that must be present in an organisational model in order for it to accurately reflect the structure of an organisation. These criteria are then used to evaluate the organisational models incorporated into current PBM systems, an analysis which concludes that there are several features of how organisations make decisions and how humans interact with policy systems which are not well supported by current models.

Chapter 3 describes the theoretical approach to organisational modelling employed in this research. The chapter starts by introducing the insights upon which the approach is based, then introduces the CBPM model and the various elements that constitute it, including the decision making algorithm employed, first informally, then formally in first order logic.

Chapter 4 describes the service oriented architecture and XACML implementation of the Community Based Policy Management System (CBPMS).

Chapter 5 present the core experimental work of this research.

- **E1:** The first experiment, described in section 5.2, applies the CBPMS to a ubiquitous computing environment, demonstrating the practical viability of integrating the CBPMS into a dynamic application domain with a complex network topology. It also provides some practical measurements for validation of the final experiment.
- **C1:** The first case study, described in section 5.3, is based upon the modelling of a political system and shows how the basic institutions of a state can be developed in an evolutionary fashion over time. It illustrates the CBPM model's ability to allow complex dynamic organisations to progressively define their structure over time and shows how complex policy requirements can be efficiently defined within the model without sacrificing its flexibility.
- **E2:** The second experiment, described in section 5.4, focuses on the CBPMS's ability to flexibly and efficiently model the evolution of a very different variety of self-managed community. An internet community is modelled over a period of time through the CBPMS and the model is evaluated by the community's administrators in comparison to their existing role based system.
- **E3:** The final experiment, described in section 5.5, illustrates the practical benefits of incorporating the organisation's authority model into the CBPMS. An organisational structure defined through CBPMS communities is compared to a RBAC hierarchy. A series of simulations is performed in order to compare the number of policies that must be evaluated and the number of conflicts encountered, illustrating how the CBPMS can resolve policy conflicts and reduce the number of policies that must be considered in any particular context.

Chapter 6 concludes the thesis with an evaluation of the experimental results against the research goals described in section 1.2; an identification of areas where further work is needed in order to progress our understanding of the area, and an overall summary of the findings of this research.

Chapter 2. Background & State of the Art Review

2.1 Introduction

A basic assumption of this work is that an understanding of how organisations work in practice will be helpful in effectively applying Policy Based Management (PBM) approaches within organisations. In particular, there is a need to understand how high-level policy decisions are traditionally made and how they are then refined, deployed and enforced within the organisation. If a technological solution applies an organisational model without reference to the organisation's existing means for policy specification and enforcement in the traditional sense, the system will require the organisation to mutate to match the technology, a mutation which will not necessarily benefit the organisation, no matter how thorough or complete the model is. Thus, the topology of the policy system's human governance interface should, wherever possible, be based upon the organisation's traditional governance model.

Due to the dynamic nature of human organisations, which can exhibit rapidly changing structures and operational requirement, a corollary of the above is that the organisation's interface to the policy system must adapt to reflect the changes in the organisation. In effect this means that, in order for it be adaptive, the policy system will need to incorporate a model of the organisation which can be dynamically modified in a reasonably efficient manner to reflect changing requirements. This chapter takes a brief look at current organisational theory in order to identify the basic outlines of governance and decision making models within human organisations which should be reflected in the organisational model embodied within the policy system. Based on this analysis, existing PBM systems are examined with a view to identifying the elements of an organisational model that they contain: how they models users of the system, their administration model; how the resources that they manage are structured and whether the models are capable of capturing the identified requirements of human organisations.

2.2 Organisational Modelling

In the early years of the 20th century, Frederick Winslow Taylor pioneered "scientific management", primarily as a means for management to gain control over the production process. Taylor's top-down approach to organisation and management was geared towards the fast growth of industrial corporations in the early years of the 20th century, particularly within the factories and large workplaces that were the hallmark of the

industrial revolution. Taylor's system was aimed towards taking traditional knowledge, such as the production techniques of artisans, and "classifying, tabulating and reducing this knowledge to rules, laws and formulae" [taylor11]. The aim being that "all possible brain work should be removed from the shop floor and centred in the planning and layout department" [braverman74]. Taylor's systematic approach to the production process was developed by theorists such as Weber and Fayol, who introduced motivational elements which were notably absent from Taylor's simple conception of 'economic man', to form the theoretical bedrock of the modern bureaucratic organisation. Richard Edward's research on companies such as Polaroid, IBM and General Electric [edwards79] revealed just how much of these theories had been absorbed into the new bureaucratic corporations.

Although Taylorism was a totalitarian system, intended to be implemented in its entirety, in practice it never quite worked like that. There is a negative side to the rigid enforcement of the "standardisation of work processes, outputs and skills" through detailed rules prescribing company policy. "Standardisation and predictability could easily degenerate into rigidity and defensive behaviour...resistant to innovation" [thompson02]. Blau's studies of a state employment agency and a federal law enforcement agency in the US show how the 'work to rule' can in fact be an extremely effective form of industrial action, rather than a tool for more efficient production [blau55].

Organisational theory continued to develop from the apparent flaws in Taylorism. The new approach, which can be loosely classified as human relations (HR) or its modern equivalent Human Resources Management (HRM), quickly came to the realisation that "it is as well to recognise that informal organisation is not 'bad' as it is sometimes assumed to be" [roethlisberger64]. The school of HR, arising out of the famous Hawthorne researchers of the 1930's, is often seen as oblivious to questions of organisational structure and even in some cases as a "gigantic, if dangerous, con-trick" [mayo46], intended to provide the appearance of devolved decision making structures, more than the reality. However, HR did at least acknowledge that micro-control was not necessarily the most effective approach and that organisational models had to take account of some form of autonomy. HR also crucially first focused on the workgroup, later to become the 'team', as an important unit for organisational analysis. Although HR approaches went out of fashion for several decades, "the idea that the internal

dynamics of the small group could be turned around so that a degree of self-governance could favour management resurfaced in the substantial wave of interest in team-working from the 1980's onwards" [thompson02].

The evolution of organisational theory in the 20th century was the story of an attempt to reconcile the seemingly contradictory demands of work organisations. Management needs to control the operations, since there is a contradiction between the needs and goals of the organisation as a whole and the motivations of the individuals and groups that make up the organisation. On the other hand, there is a need to involve the individual subjects in the organisation's decision making processes in order to gain their good-will and take advantage of the inherent human propensity for innovation. This dialectic has given rise to many insights, management fads and behaviourist theories. For example, contingency theory was the dominant approach of the 1970's. It held that different levels of autonomy and direct control depended on the particular environmental factors within which an organisation found itself and that different approaches should be applied in different industries, divisions and market sectors. So, for example, research into plastics manufacturing companies [lawrence67] revealed that whereas their research departments operated in dynamic, innovative environments that were reflected in non-hierarchical structures with minimum bureaucracy, the production division had a stable, technical environment and was dominated by short-term concerns and more bureaucracy. The researchers found that high-performing firms were those that manifested a high degree of differentiation of structures and goals.

Overall, it can be said that the development of organisation theory has been dominated by the conflicting forces of Taylorist "scientific management" and notions of autonomy and self-management, often embodied in teams. These conflicting forms of structure and organisation exist in parallel in all organisations, depending on the particular environment and history. All management innovations and fads can be viewed in terms of this conflict. Modern production methodologies such as Total Quality Management (TQM) [boje93] and Just In Time (JIT) [schoenberger97] are modern Taylorist trends, in that they aim to precisely control the production and labour processes. However, they also include elements of self-managing, self-supervising, team-working paradigms. The trend towards flat hierarchies, team-working and autonomous groups is most obviously seen in many 'new economy' companies and it has been widely predicted that this trend will extend itself across all organisations – creating a new model of a "post bureaucratic

organisation” [heckscher94]. On the other hand, many researchers are sceptical about claims that organisations will ever be able to entirely do away with bureaucracy, hierarchy and formal rules. They argue that many of the examples commonly cited as post-bureaucratic still retain these elements, albeit in a less clearly defined way [hopfl06, mcsweeney06, Freeman72].

The formal structure of an organisation does not necessarily correspond with the way that decisions are actually made. In bureaucratic organisations, the individuals who implement policy can be given the ability to take decisions when there are policy conflicts using the innate human ability to refine high level policy into concrete policy in complex situations, or they can simply choose to ignore policy. If there is too much emphasis on concrete rule-bound policy, and not enough on delegation or autonomy, situations of deadlock can ensue. For example, unanticipated factors can create a situation in which policy dictates that no subjects have the authority to take the actions required to deal with a situation. If there is too much emphasis on individual decision making, contradictory decisions can ensue and the organisation’s overall goal can get lost, leading to a chaotic situation.

Furthermore, as well as the capacity for autonomous policy conflict resolution and refinement, real-world organisations contain informal decision making structures. In many organisations, key decision makers may, in practice, be personal assistants, secretaries or system administrators. Due to their access to information and their monopoly of crucial skills, some individuals, despite holding relatively modest positions in the organisation’s formal hierarchy, may exercise a disproportionate amount of control over what choices are made in the behaviour of the system. In effect, they influence organisational policy.

Thus, overall, an examination of organisational theory suggests that a naïve approach to organisational modelling, based on the exhaustive accumulation of rules by administrators, for the purpose of deploying automated policy based management solutions, will be unlikely to succeed. Even when human organisations employ centralised, hierarchical, rule-based decision making approaches, they invariably still contain many informal elements which may be crucial to the organisation’s operation but will not be captured by the organisation’s policies. Furthermore, even within traditionally Taylorist organisations, modern management approaches are tending to de-emphasise the importance of strict bureaucracy in favour of decentralised decision

making. For example, US military doctrine, which has traditionally been dependant on extensive central planning and control, now incorporates elements of decentralized decision making to take advantage of advances in information systems [lee95].

2.2.1 Organisation in Multi-Agent Systems

Multi-Agent Systems (MAS) use multiple, simple autonomous agents in order to solve complex problems through coordination. Due to the nature of the domain, the problem of how such agents should be organised in order to achieve their collective goals is a key area of research [e.g. mao05, dignum02, dignum05]. Although the problem of creating a model to allow multiple software agents to form an effective organisation is quite different to the problem of applying policy to a human organisation, there are some commonalities. Many of the engineering approaches are based upon models drawn from administrative and economic studies of human organisations [ferber03]. Furthermore, the problem of translating the global goal of a MAS into individual choices in the behaviour of the agents is closely related to the problem of refining high level goals into concrete policies within a human organisation. On the other hand, MAS research does not concern itself with the human governance interface of such systems and the problem of how the system's policy is defined is beyond its scope. Nonetheless, it is worthwhile to briefly consider the organisational models employed in this domain since it addresses a closely related problem and MAS architectures have even been proposed for implementing policy based security management [boudaoud01].

Coutinho et al. [coutinho05] have classified a variety of these MAS organisational frameworks, namely AGR (Agent, Group, Role) [ferber03], MOISE+ (Model of Organization for multi-agent SystEms) [hubner02], ISLANDER [esteva04], ODML - Organizational Design Modeling Language [horling04], AUML (Agent UML) [parunak02], MAS-ML - MAS Modeling Language [silva04], OMNI (Organizational Model for Normative Institutions) [dignum05]. Based on this analysis, they identified four major modelling dimensions in these models, namely *structural elements*, *functional elements*, *normative elements* and *dialogical elements*. Their preliminary ontology of organisational concepts used in these models is shown in figure 2.1. Structural elements refer to roles that agents assume and groups that agents belong to within the organisation. Functional elements allow the definition of organisational goals and the ability to decompose these into plans consisting of sub-goals. Normative elements are the commitments, obligations and rights of the agents who belong to the

organisation. Dialogical elements are provided by some models and allow the interactions between agents in the organisation to be co-ordinated. These can range from simple protocols to complex orchestrations of interactions between agents as part of an arrangement of scenes.

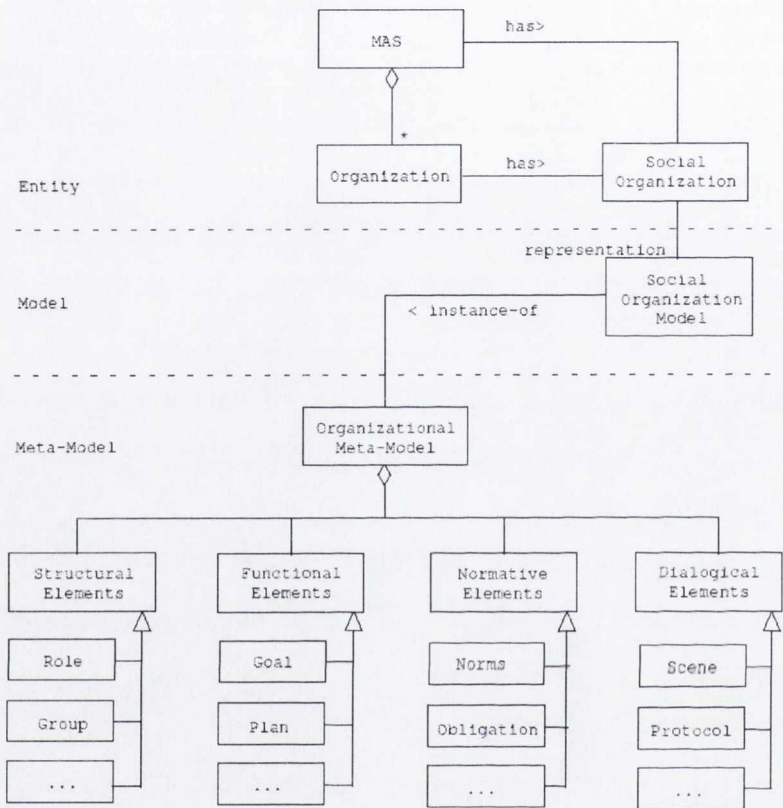


Figure 2.1 Ontology of organisational concepts present in MAS models.

To a large extent, the concepts used in MAS organisational models correspond with similar concepts used in policy based management systems. For example, roles and groups both exist in the Ponder policy framework [damianou02b] as well as in many other policy languages. The Normative elements in the MAS ontology are broadly equivalent to the rules of policy languages. However, it should be noted that not all of the modelling dimensions are supported by all of the MAS systems evaluated. Many MAS organisational models adopt a minimalist approach and, rather than over-specifying the organisational model, assume that much of the intelligence will be embedded in the agents. Indeed the theoretical underpinning of MAS architectures relies upon the fact that the agents' behaviour is not centrally controlled, and that it is the emergent behaviour of the autonomous agents which solves the collective problem rather than a prescriptive, centralised solution. What is true for software agents is even more true for human organisations.

2.3 Policy Based Management

Within network and system management, the term *policy* is used in a range of contexts. It has been described as "a much-overloaded concept ... used to describe everything from the specification of corporate policies on computer use to a data driven configuration in a router" [mont99]. Due to the overloading of the concept and the wide range of different contexts in which it is used, there is no universally accepted definition of the term when it is applied to network and system management. One definition is offered by Damianou, based on his work in Imperial College London on the Ponder policy language. He defines a policy as "*A persistent declarative specification, derived from management goals, of a rule defining choices in behaviour of a system*" [damianou02b]. However, the term 'choices' in this definition is ambiguous as it can refer to the set of available options or the set of options actually taken. It would be clearer if it was written as follows:

"A persistent declarative specification, derived from management goals, of a rule defining decisions about choices in the behaviour of a system"

This definition, which is adopted as a working definition for the purpose of this thesis, captures the fact that a policy specifies a particular choice that has been decided upon rather than specifying the range of choices available. A policy can be seen as a persistent record of a decision that, in a given context, out of all the available choices, a certain choice should be taken. Thus, it follows that, in a policy driven system, those who define policy are the decision makers of the system.

Broadly speaking, the goal of policy based management is to enable information systems to be self-adaptive and to dynamically change their behaviour during operation in order to reflect the high-level business goals of the organisation. Policy based management solutions, taken to their logical conclusion, promise a future where human decision makers can concentrate on defining high level goals and, once decided, have these goals automatically translated into appropriate configurations on managed information resources across the entire enterprise, which will then act in such a way so as to achieve management goals. There are several important challenges to the creation of such systems, which have attracted the attention of researchers. The following sections briefly describe some of the key research areas in the domain.

2.3.1 Policy Deployment

Policy Deployment is a broad area which addresses the problem of how policies can be deployed on the information systems which are managed by PBM Systems. Research in this field includes the development of architectures, communication protocols, data formats and methods for mapping policies to low level device configurations. The most notable work in this area is the Internet Engineering Task Force (IETF) policy model [moore01] and the related IETF standards such as the Common Open Policy Service (COPS) Protocol [ietf00]. For the most part, policy deployment issues are beyond the scope of this thesis, which is primarily concerned with the interactions between human organisations and policy systems. However, the IETF policy architecture is the most widely used standardised model, much of its terminology is used in this work, and it addresses more than deployment issues. It is discussed in greater detail in section 2.5.1 below.

2.3.2 Policy Specification

In policy based management, the instructions to the system are persistent. That is to say that policies are relatively static compared to the state of the system. This contrasts with traditional network management which often relies upon commands – one-off instructions which change the state of the system but do not persist as instructions. Persistent instructions allow the system to accumulate knowledge about the organisation to which it is applied – the policy system comes to embody information about the organisation which allows it to adapt to its requirements. Thus, one focus of research in PBM is the problem of how these persistent decisions can be specified and encoded.

In traditional IT management, each type of managed resource generally has its own language for specifying its configuration and its access control permissions. Some types of resources have a relatively standardised access control configuration language, for example, most database systems support the access control specification instructions specified in the ANSI SQL92 standard [ansi92]. But even those resources which do have a standardised configuration language often have vendor-specific, or even version-specific details which require translation of instructions between different implementations. In other areas, such as most dedicated network routing equipment, the configuration languages are entirely different for each vendor (standardised low-level network management protocols such as Simple Network Management Protocol (SNMP) do not govern the semantic content of the configuration messages which they transmit).

This creates serious inefficiencies for administrators who must be able to specify rules in several different languages and be familiar with multiple different access control enforcement approaches. Furthermore, it means that implementation changes, or even version upgrades can require the entire set of configuration rules to be completely re-written. Research into policy specification languages is partially motivated by the desire to avoid this necessity by allowing administrators to specify rules in target-independent, semantically-consistent specification languages. There have been several practical attempts to develop such general-purpose policy specification languages [damianou02b, ribeiro01]. Most industry standardisation attempts have focused on languages designed for more restricted application domains. For example, the IETF Policy Core Information Model (PCIM) [moore01] and the DMTF Common Information Model Simplified Policy Language (CIM-SPL) [dtmf07] standards both focus on network policies, while OASIS have concentrated their efforts on standardising the eXtensible Access Control Markup Language (XACML) [oasis03, oasis05].

Since one of the key goals of PBM approaches is to provide a means of handling the increasing complexity of information system administration, a key requirement of policy specification is to provide convenient grouping abstractions to enable much more efficient management and maintenance of policy sets. For example, it is much more convenient for IT managers to be able to specify rules that apply to all databases or all programmers rather than having to configure system-wide rules for each database and each programmer individually. The use of grouping abstractions is not unique to policy systems, and is common to many types of traditional access control management system. For example, discretionary access control models, commonly used by the UNIX family of operating systems to control access to the file system, allow permissions to be specified for groups as well as for individual users [doeppner83]. Firewall configuration languages allow rules to be applied to classes of traffic. However, PBM aims to extend the scope of the grouping abstractions used in such systems, enabling rules to be specified to classes of entities which apply across heterogeneous systems and networks and are applicable and meaningful on multiple managed information resources [sloman94a]. Policy specification languages use abstractions which group similar resources, such as network devices, together [boros00], or which group individuals together on the basis of them having similar functions in an organisation [sandhu00, lupu98] or which divide organisations into management domains [sloman94b]. These

grouping abstractions and the relationships between them that are defined within policy specification languages, when applied to an organisation's information systems, embody elements of an organisational model in that they group together subjects, objects and administrators and relate them to each other.

2.3.3 Mapping Management Goals to Concrete Policies

Although it might seem like a simple task, where management can simply translate their goals directly into a policy specification language, the mapping of management goals to policy specifications turns out to be an extremely complex process. Management goals tend to be expressed in high level language, in a way that can not be mapped directly into a policy specification. For example, the ultimate goal of a publicly listed corporate is normally "to maximise shareholder value" and it is unlikely that PBM systems will, in the near future, have the ability to translate such high-level goals into concrete policies that can be implemented by network devices. Organisational theory recognises a distinction between governance (what is to be done) and management (how it is to be done). Governance generally applies to high level goals, while management is concerned with implementation. In PBM, governance rules can be referred to as 'abstract policies' while management rules are termed 'concrete policies' [damianou02a, lupu99, moffett93]. Policy refinement is the process of transforming high-level, abstract policy specification into a low-level, concrete one that can be directly enacted. In [moffett93], the concept of policy hierarchies is introduced, whereby higher level policies are refined into lower level policies.

In [damianou02a] the main objectives of a policy refinement process are identified:

- To determine the resources needed to satisfy the requirements of the policy.
- To translate high-level policies into operational policies that can be enforced.
- To verify that the concrete policies meet the requirements of high-level policy.

The first of these objectives involves mapping abstract entities defined as part of a high-level policy to the physical objects/devices that make up the underlying system. The second specifies the need to ensure that any policies derived by the refinement process, be refined to operations that are supported by the underlying system. The final objective requires that there be a process for incrementally decomposing abstract requirements into successively more concrete ones, ensuring that at each stage the decomposition is correct and consistent.

The use of logic is important in the area of policy refinement. As an example, Michael et al. [michael01] describe a suite of tools, which serve as an expert database management system to automate the process of mapping natural language policy statements into equivalent first-order predicate calculus. Other approaches use event calculus [bandara03, bandara04] to formally describe policies in order to make them amenable for analysis. Van Lamsweerde [vanlamsweerde95, vanlamsweerde96, vanlamsweerde99] proposes a formal technique for elaborating goals grounded in Temporal Logic. This approach, called KAOS¹, represents each goal as a Temporal Logic rule and then makes use of refinement patterns to decompose these goals into a set of sub-goals that logically entail the original goal [darimont96]. Another method that has been used relies on finite state machines in order to decompose goals into implementable sub-goals [lewis07]. In [dealbuquerque05] an approach to automatic derivation of low-level network security policies, which depends on modelling managed systems as abstract sub-systems, is proposed.

Strassner has developed a policy continuum model, shown in figure 2.2, designed to allow multiple constituencies to co-define and develop policies, from different views and at different levels of abstraction [vandermeer06]. It recognises the fact that policies are specified at many different levels, but that these policies can be interconnected and that different views are appropriate for different roles within the organisation.

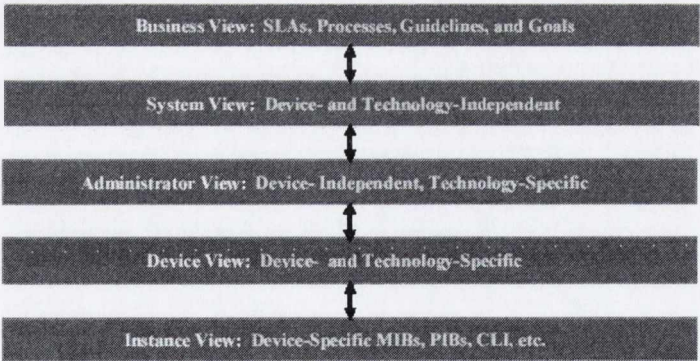


Figure 2.2 Strassner’s Policy Continuum Model

2.3.4 Policy Conflicts

Another problem encountered in mapping management goals to policy specifications is that policy conflicts can arise in the resultant policy set. Policy Conflicts occur when “the conditions of two or more policy rules that apply to the same set of managed

¹ Not to be confused with the semantic policy language KAoS

objects are simultaneously satisfied, but the actions of two or more of these policy rules conflict with each other” [strassner97]. Essentially, this means that the policy set contains two conflicting decisions which specify that different choices should be taken in the same context. The PBM system needs either to ensure that such policy conflicts cannot arise, by ensuring that conflicting policies cannot be specified, or it needs to have access to information which allows it to resolve the policy conflict, for example, by deciding that one or other of the policies is more important for the attainment of the organisation’s goals.

The problem of specifying policies so that there are no such conflicts within the set is not simple. In particular, the identification of what constitutes a conflicting action is dependant on the semantics of the resource or application being managed. For example, when dealing with a packet router, two policies which have true conditions in a given context and actions that place the same packet on two different queues will be in conflict, whereas in other application domains, there will be no conflict if two different actions are taken on the same resource. Thus, it is essentially impossible to adopt a simple generic solution to eliminating all policy conflicts from a policy system – an understanding of the application semantics is required in order to identify conflicting actions. However, in many cases the application semantics can be simple enough to make the task of identifying policy conflicts amenable to analysis. For example, in dealing with access control policies, in most cases, policy specification languages are designed in such a way as to enable the identification of situations where two policies will apply to the same access control request (by matching subjects, actions and targets for example). Policy conflicts can easily be identified when two or more policies have opposite effects – i.e. they return negative and positive authorisations. This class of policy conflict is known as a *modal conflict* [lupu99].

A classification of policy conflicts is presented in [lupu99], which discusses both modality conflicts and application specific conflicts. Modality conflicts can be categorised into 3 distinct types:

1. Authorisation conflicts arise when a positive and a negative authorisation policy is defined for the overlapping subjects, targets and actions.
2. Obligation conflicts arise when one policy obliges a subject to perform an action whilst at the same time another policy forbids the action from being performed.

3. Unauthorised obligation conflicts arises when a subject is obliged to perform an action that it does not have the authorisation to do. In a system with a default negative authorisation policy in which actions have to be explicitly authorised, this could occur if an obligation policy is defined without an associated authorisation policy.

Application specific conflicts are those that arise because of constraints that exist but may not be defined for the particular application in which the policies are being used. For example, a system that enforces the principle of separation of duties would define a conflict if the person who submits an expenses report is also allowed to approve it. Furthermore, outside of access control policies, many actions specified in policy rules do not have a modality with respect to each other, making the identification of policy conflicts application-dependant. There has been some work in defining semantic models for some domains to allow the automated identification of application specific policy conflicts. In [charalambides05] a hierarchical conflict classification tree is introduced, targeted at the network QoS domain. In [baliosian04], a technique for analysing policy conflicts in a technology neutral way using an adapted subset of Finite State Traducers to create a formal model of the policy system in an efficient manner in linear time is described. Other work has proposed the use of decision tables to minimise the set of policies to be evaluated for conflict analysis when a policy rule is updated [graham04].

Policy conflicts can also be sub-divided into static and dynamic conflicts [jajodia97]. The distinction is that analysing the syntax of a policy statement can identify static conflicts. These conflicts will occur irrespective of the state of the system enforcing the policies, as is often the case for simple modality conflicts. Dynamic conflicts are those that occur at run-time and arise because a particular state of the system results in a conflict. Dynamic conflicts are harder to detect in advance given that it may be necessary to analyse the system in all possible states to do so. For example, if a condition in a policy rule depends on the data contained in an external database – all the possible different states of that data would have to be analysed in order to identify potential conflicts. In general, the more expressive the policy language, the greater the number of dynamic conflicts that may arise and the greater the difficulty in analysing policies to detect conflicts.

Rather than attempting to eliminate policy conflicts, some PBM systems allow administrators to specify automatic policy conflict resolution algorithms, which allow

policy conflicts to be resolved even in dynamic situations - for example, by ensuring that negative authorisation policies override positive authorisation policies in access control systems, or by applying priority levels to policy rules. The XACML concept of a *policy combining algorithm* and *rule combining algorithm* being specified with each set of policies generalises this approach and allows different combining algorithms to be specified for different groups of policies throughout the system [oasis03].

2.3.5 From Management Goals to Organisational Goals

It can be assumed that most PBM systems will be deployed within organisations in order to fulfil organisational goals rather than being deployed to achieve the goals of the individuals who manage them. There may be many individuals with various different roles within the organisation who act as human managers of information systems and their governance requirements may be contradictory. Furthermore, the organisational goals at the highest level may contradict the goals of individual managers, who are operating within different areas and working to achieve the local goals of their unit rather than the global goals of the organisation. Although this area has been acknowledged as problematic [damianou02a], most research into the problem of mapping management goals to concrete policies has implicitly assumed that management goals are themselves non-contradictory and that it is possible to capture a set of management goals which can be refined into a set of non-conflicting policies. The discussion of organisational theory above suggests that this is not the case. Thus, rather than trying to derive policies from management goals, it makes more sense to aim to derive policies from organisational goals – since the purpose of the system is to help to achieve the goals of the entire organisation and management requirements will vary across the organisation and the goals of some organisational units will conflict. In such cases, it is only by referencing the higher level goals of the organisation that a resolution can be reached. For example, two departments might run revenue-generating services which require network bandwidth. When the network is saturated, the best allocation of bandwidth, from the organisation's point of view, will maximise revenue across both services. It is not possible to achieve an optimal solution by looking at the local goals of the competing units in isolation, since both departments are correctly attempting to achieve their local goals by maximising revenue for their particular services.

Another major problem for PBM systems stems from the complexity of deployment. Most current approaches depend upon a difficult and intensive phase of requirements

analysis prior to deployment, using methodologies borrowed from software requirements engineering, but there is little evidence of their utility in practice. In [crook02] an attempt is made to develop a role-modelling framework which captures much of the complexity of real world organisations, by modelling roles under a number of different criteria, based on seniority, qualification, function, work process, market and environmental factors. However, there is no evidence that this works in practice and the complexity of this engineering only serves to underline the complexity of the problem. The complexity of this task is not surprising, however, when the approach is looked at through the lens of the organisational theory discussed in section 2.2. The definition of policy is carried out by a group of experts who attempt to express the various management goals of the organisation into an exhaustive set of concrete policies. This policy set is thereafter maintained by expert administrators who can modify it, but significant modifications will themselves require extensive analysis before they can be deployed. This is a decidedly Taylorist approach to organisational modelling and it is no surprise that such systems are difficult to integrate into real world organisations. In one of the few published articles that examined the problems of deploying PBM solutions in practice, Michael Jude noted that:

"Policy-based network management (PBNM) turned out to be difficult to put into practice. Early adopters have found that developing and deploying policies is not simple, cheap or quick. Instead, PBNM has been a time-intensive, complex, expensive process. Additionally, it has demanded that the enterprise organisation mutate to match the technology-rather than the technology meeting the enterprise's management needs." [jude01]

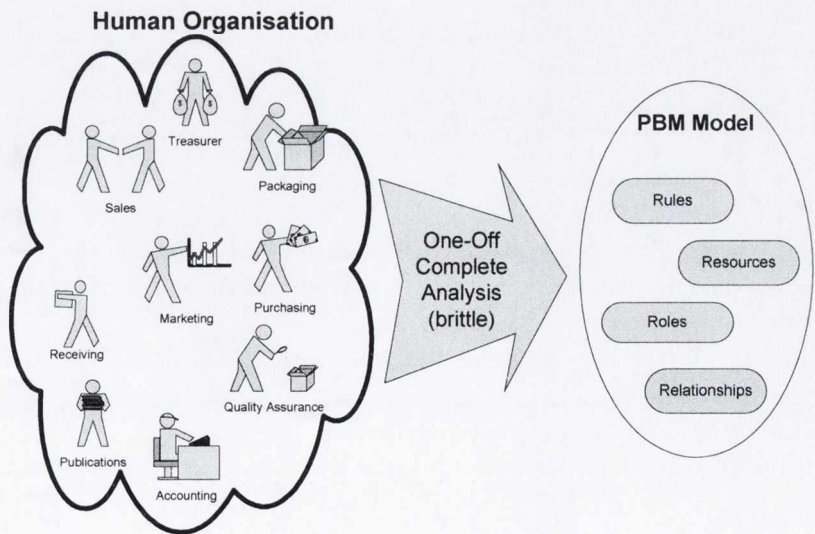


Figure 2.3 The Brittle Nature of One-Off Requirements Engineering for PBM

PBM systems should be able to incorporate changes in the real world organisations which without requiring the re-programming of the system. Changes to the requirements of the organisation should be accommodated by changes to policy rules rather than changes to software. If goals, membership, relationships with other organisations, business rules or any other aspect of the organisation that is modelled by the system changes, this should be incorporated as choices in the behaviour of the system and should not require changing the functionality of the system. Organisations are dynamic entities with changing requirements and policy based management systems should be able to model these changes and enforce any of the consequent requirements through updating rules rather than modifying the functionality of components. By accommodating as much of the variability of requirements as changes to rules rather than changes to the software, the cost of managing and integrating changes into the system should be minimised. However, using current approaches, the cost of making extensive changes to the policy system can be great, since it may require extensive re-analysis and the derivation of a new exhaustive set of policy rules to capture the new situation.

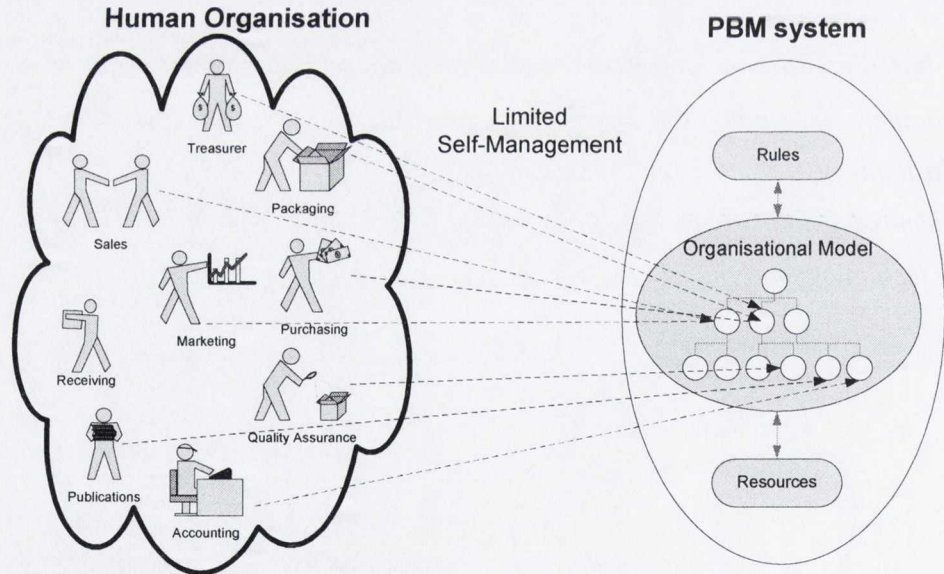


Figure 2.4 Using a dynamic organisational model to enable limited self-management

By incorporating a model of the organisation into the PBM system, as illustrated in figure 2.4, many of these problems can be addressed. Organisational goals can be broken down into lower level management goals, covering the goals of units within the organisation and these goals can be broken down further into goals of sub-units. By linking goals to their organisational context, a clear policy refinement path can be

identified, linking higher level goals of the entire organisation to lower level management goals of specific units. Policy conflicts which result from conflicts between organisational units could be addressed by reference to higher level policies. The complexity of requirements engineering and change management could be reduced by decentralising the modelling process and making groups within the organisation responsible for managing their own model within the system – which mirrors the situation in real world organisations where, as has been shown above, there is always some level of autonomy and decentralised decision making. By incorporating a model of the organisation into the PBM system, and linking policy specifications and resources to specific organisational contexts, the system can reason about policies (which can be considered to be decisions) based on factors such as: “*who made the decision?*” “*With whose authority?*” and “*what is the scope of their authority within the organisation?*” These are important considerations for people in real world organisation when they decide how to act, particularly when decisions conflict.

2.3.6 Evaluation Criteria

From the point of view of a policy system, three distinct areas where organisational models are employed can be identified, at least implicitly. Firstly, people can be the subject of rules – that is to say that rules can define what a person can or must do. Typically, policy systems support the definition of access control policies which specify what human users of a system are authorised to do. These are generally known as *permissions* or *authorisation policies* [moffett90]. Some AC systems also allow rules to be expressed as *negative authorisation* [lupu99] or *prohibition policies* [dunlop03], which are policy rules that express what users of a system may not do. People can also be the subject of *obligation* [gama05] or *refrain* [damianou01] policies, which specify what they must and must not do respectively.

In order to minimise the number of rules within the system and to make the task of specifying and maintaining the set of rules more tractable, abstractions such as *roles* [ferraiolo92], *groups* [wilde05] and *organisations* [elkalam03] are employed by policy systems and these comprise some of the elements of the system’s organisational model. Such organisational constructs can be equally applied to software agents acting as subjects in a system, as discussed in the section on Multi-Agent Systems in section 2.2.1. From the point of view of a policy based system, it makes little difference if the entity that is defined to be the subject of an action is a human user or a software agent.

A second area where human organisations interact with policy systems relates to the authoring of access rules and the definition of whatever structural elements are utilised by the system. This is most commonly described as the administration of the system and although some systems assume that the administrator is an individual or a small group which is sufficiently well co-ordinated, mutually trusted and well-organised in the real world not to require that the system should contain any organisational model of the users who carry out these administrative tasks. However, many systems do support the definition of elements of an organisational model as it relates to the system's administration, using constructs such as *administrative roles* [sandhu99a]. Many policy systems also support some form of delegation of permissions, whereby users can temporarily grant a subset of their permissions to other users and this can be considered as a means of decentralising the administration of the system, albeit to a limited extent.

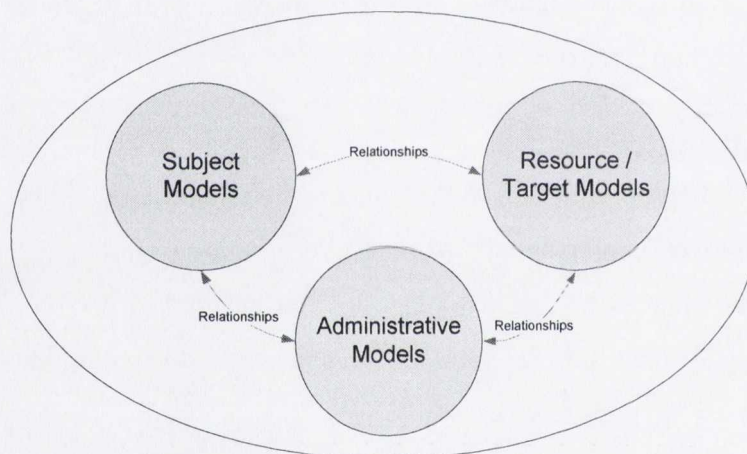


Figure 2.5 Organisational Models Embodied in AC Systems

The third area where policy systems utilise organisation models, of a sort, is in modelling the information resources managed by the system. These resources can be viewed as essentially passive elements in the system, which are acted upon by actors [lampson71]. They most commonly appear in policy systems as the target of actions in rules. Once again, in order to be able to author rules which apply to the use of sets of entities rather than individuals, policy systems generally provide grouping abstractions which can be used to group together managed resources and which provide them with some basic organisational structure. For example, UNIX directories serve to group together file system objects for the application of AC policies. Rather than forming an organisational model in their own right, it is the relationship of the resource groupings to the organisational models describing actors and administrators of the system that embodies the overall organisational model.

A set of evaluation criteria can now be defined which will be used to evaluate the current state of the art in PBM systems.

- 1 What model of the organisation can be derived from the PBM model? What grouping abstractions are used? What are their relationships to each other?
- 2 How does decision making reflect knowledge of how decisions are made in real world organisations: autonomy, decentralisation, delegation, etc?
- 3 What support is there for dynamic evolution of the model to reflect changes in the real world organisation without a need for extensive remodelling?
- 4 Can complex relationships and conditions be expressed such that the PBM system can capture the complexity of real world systems?

2.4 Organisational Models in Access Control Systems

Although they very much pre-date the emergence of PBM as a specific field of research, Access Control (AC) systems provide a good starting point. Despite being generally narrower in scope and defined at a lower-level than many modern PBM approaches, AC systems deal with the same problem of managing the Organisation-Computer Interface as do modern PBM systems and they embody similar elements of the organisational model. Although many AC systems only support the specification of simple positive *permissions*, a permission can be considered as a simple policy, written in a specification language which has only a single word vocabulary (i.e. *permit*). Appendix 1 presents a detailed discussion of the major variations of AC models that have been developed and the organisational models embodied within them. A brief summary and overview of these findings is presented here.

Research into access control has generally concentrated on identifying useful AC models, which provide a formal representation of security policies and allow properties of the system to be proved through formal reasoning. This places inherent limitations upon the extensibility and the practical expressiveness of the models produced, since the task of introducing extra factors, with their own semantics, into the formal model is complex. AC systems have traditionally been divided into *discretionary* [osborne00, solworth04] and *mandatory* [bell76] models. Discretionary policies are concerned with the specification of authorisation rules to govern the access of users to information contained within a system, whereas mandatory policies are mostly concerned with controlling information flow between the objects of a system. Information flow policies

are often described as a separate type of policy, and are directly related to the issue of data confidentiality. More recently variations on *Role-Based Access Control* (RBAC) models have attracted increasing attention; they allow more flexible specification of rules and are expressive and flexible enough to serve as a superset of the traditional approaches.

2.4.1 Mandatory & Discretionary Access Control

In Mandatory Access Control (MAC), users are grouped together by their classification levels and the categories that they belong to, but these are fairly unwieldy and rigid grouping methods. The classification level of objects is the only way in which the managed resources are related to the human organisation, meaning that every object is related to all the subjects in the system in one of only 3 possible ways. No aspects of the functional nature of the organisation to which the subjects belong is modelled. All administration of the system is carried out by a security officer whose access rights are external to the model. Furthermore, the model is fundamentally restricted in terms of its expressiveness, since there is no facility for specifying conditional access and only four different means of accessing each object are supported.

Discretionary access control (DAC) has been used extensively in commercial applications, particularly in operating systems. The central idea of DAC is that the owner of an object, who is usually its creator, has discretionary authority over who else can access that object. DAC, in other words, involves owner-based administration of access policy. This provides considerable flexibility and decentralisation of the system's administration by devolving many access control decisions about an object to the user, rather than being the responsibility of a dedicated security administrator. There are many variations of DAC models, such as *strict DAC* [graham72] and *Liberal DAC* [osborn00] and the major distinguishing feature between them concerns how the owner's rights can be delegated to other users and how such delegations can be revoked. The most basic and fundamental approach to the specification of DAC policies uses an access matrix model [lampson74]. Such models are severely limited in their expressiveness. Subjects are mapped directly to access rights and thus many classes of constraints (such as temporal or history-based constraints) on authorisations cannot be expressed [samarati00]. Various formal extensions have been proposed to DAC models, such as temporal constraints [bertino03] and negative permissions [samarati00]. Many DAC systems support some form of *group* construct, which allows users to

delegate rights over the objects that they own to groups of users. Although DAC is most commonly based on the concept that each object has an owner who is responsible for administering access to that object, in practice such systems generally also include administrators who can grant and revoke rights. A variety of schema for specifying administrative policies exist, classified into major categories of centralised, hierarchical, cooperative, ownership and decentralised in [sandhu94].

In most DAC models, resources are strongly related to users in the system through the ownership relationship. Support for groups which have a relationship with resources augments this to give some representation of organisational units and collective functions; however this generally remains secondary to the dominant ownership relationship. Thus, the model of the organisation that is contained is fairly atomised, with strong relationships between individual users and resources, but weak relationships between users and between groups of users and resources. DAC's centralised administration essentially positions the administration of the system outside the user model. In most of these schemas, the dynamism of the system, whether driven by an administrator or by an object's owner, is very difficult to constrain. Limitations can be placed on delegations and administration rights, but other than that the organisational model can evolve in any direction. In general, the limitations of these schemas means that it is very difficult to derive the functional aspects of the organisation from the models embodied in a DAC configuration. Traditional DAC models also generally lack expressiveness and flexibility in terms of how they can constrain access to objects.

2.4.2 Role Based Access Control

Most recent research into access control models has focused on the role as the fundamental grouping abstraction¹. In 1996, Sandhu et al. [sandhu96] proposed a set of conceptual RBAC models, which were eventually approved in February 2004 as the American National Standard ANSI INCITS 359-2004 [sandhu00]. Meanwhile, an associated administrative model was developed in a series of papers known as ARBAC97 (Administrative Role Based Access Control), ARBAC99 [sandhu99a, sandhu99b] and ARBAC02 [oh02]. Although the RBAC / ARBAC models represent the only real standardisation in the field, they are very limited in their modelling capability. Only positive permissions are supported. Hierarchical inheritance relationships are the

¹ Roles are grouping abstractions in that they group together policies from the point of view of the policy administrator. They generally can't, however, be considered to represent a group of users with a collective function.

only way that roles are related to one another and such hierarchies have been shown to be insufficient for modelling the complex interrelationships between individuals within human organisations [moffett98, goh98]. Role inheritance, normally based upon the concept of specialisation and generalisation as used in object oriented programming, is not well suited for modelling structural elements, supervision hierarchies or other hierarchies that exist within organisations. Furthermore, both ANSI RBAC and ARBAC lack support for delegation of permissions and assume that the organisational model will be constructed in advance through a complex requirements analysis phase and will be thereafter maintained by specialist administrators.

As a result of the limited nature of the RBAC standard, a large number of extended, or alternative models have been proposed. Many of the extended models have sacrificed formal provability for expressiveness and include negative permissions, which give rise to the possibility of conflicts [lupu99, alkahtani04, miege05] and further complicate the problems of role-inheritance [miego05]. Some extended RBAC models also contain obligations [miego05] and integrate a general purpose concept of *contexts* into their models [miego05, georgiadis01], or base the assignment of users to roles on attributes, rather than depending on static assignment by security officers (e.g. Rule-Based RBAC (RB-BAC) [alkahtani04]).

Several of the extended RBAC models add new types of grouping abstractions to the standard role. For example, Team Based Access Control (TMAC) is an extension to RBAC, first proposed in [thomas97a] and later extended with contexts as C-TMAC [georgiadis01] and later updated to TMAC04 [alotaiby04]. These models add a concept of the *team* to the RBAC model which supports RBAC's use in collaborative working environments. Organisation Based Access Control (OrBAC) [miego05] is designed for situations in which users have different permissions depending on the *organisational context* within which they find themselves and where the policy system manages resources that refer to multiple organisations and sub-organisations. Coalition Based Access Control (CBAC) [cohen02] incorporates teams, organisations and tasks from other models and augments them with a range of new entities such as coalitions and partner organisations.

Several extended models have also been proposed to support delegation and finer granularity of administrative policies. For example, Scoped Administration of Role-Based Access Control (SARBAC) [crampton02] is an extension of RHA4 [crampton03]

and an alternative to ARBAC97, that provides fine-grained limits to the scope of administrative rights. In [barka00a], a role-based delegation model called RBDM0 is introduced. In RBDM0, if a role is delegated then all the associated privileges are granted. Delegation is limited to a single step. Later work by the same authors [barka00b] identifies cases of role-based delegation that are useful in practice, and in particular extends the model to include cascading delegation. RDM2000 [zhang03a] is another RBAC role-delegation model which allows reasoning over role-histories. Other work has looked at providing greater granularity of delegation, rather than being confined to delegating membership of entire roles, such as the Permission Based Delegation Model (PBDM) [zhang03b] or has provided more fine-grained control over the semantics of delegation [crampton06].

Summary of Organisational Models in RBAC

The organisational models embodied within role-based AC systems focus primarily on defining the permissions for individual users. In all the RBAC-derived models, role membership remains the core determinant of access control decisions – even when other variables are significant. Roles are a relatively low-level grouping abstraction; they are simply collections of permissions which are mapped to sets of users. Thus, RBAC models use a bottom-up approach to constructing a model of the organisation. The overall organisational model is built up from the set of roles and their relationships to each other, rather than by decomposing the organisation into component structures and functions in a top-down manner. As a result of this approach, role-based models exhibit several limitations when attempting to model the collective aspects of organisations.

Structural & Functional Units

The various higher-level structural and functional units of real-world organisations – divisions, departments, regional branches, teams, committees, boards, teams and so on – are not captured by the standard RBAC models. Although TMAC, CBAC and OrBAC do provide support for structural grouping abstractions which can also have a functional significance, they do not define how the policies defined within these abstractions relate to one another, nor how they relate to the whole. In TMAC teams are modelled in isolation from one another and there is nothing within the model which allows them to be directly related to one another or to the system's global function. The OrBAC model includes the *sub_organization* relation between organisations, but specifies no particular semantics for inheritance of roles across organisations and leaves it up to security

administrators to define whether roles are inherited up, down or not-at-all between organisations and their sub-organisations. Similarly, the relationships between the various entities in the CBAC model are not well defined and it is unclear how policies defined in different organisational contexts interact. Finally, few of the models define abstractions which relate managed objects to one another or which allow users, or groups of users, to be tightly coupled to groups of objects.

Model Construction and Administration

Few of the RBAC models attempt to integrate the model of how the system is administered into the user model to build a unitary model of the organisation. Administration is either specified as a distinct administrative hierarchy, as in ARBAC97, or partially related to the user model through the organisational scope, as in AdOrBAC, or left undefined altogether. While the role abstraction may represent a bottom-up approach to modelling the organisation, the modelling process in RBAC-derived models generally takes a decidedly top down approach. Permissions are assigned to roles by specialist security administrators and, although many delegation models have been proposed, they are normally concerned with temporary transfers of permissions, with all permanent changes to the system's state carried out by administrators. Thus RBAC models are built from the top-down, using low-level abstractions. This contrasts strongly with how structures and policies in real-world organisations tend to be created. As they grow, organisations divide themselves into top-down abstractions – creating divisions, branches, teams and so on as they are required – and assign sub-goals to these organisational units. The actual rules governing the functions of the units is often defined through a bottom-up process whereby policies are added through progressive grounding as problematic situations are encountered and, as they evolve, authority and resources are frequently redistributed amongst the organisational units in a semi-permanent but never entirely static way.

2.5 Practical Policy Based Management Systems

Having examined the high-level motivations driving PBM research and the theoretical models developed for modelling human organisations within AC systems, this section will summarise the most important research work that has aimed to produce practical implementations of PBM systems. In general, research in the area commonly known as *Policy Based Management* originated in the field of network management and practical PBM system research has been, by and large, less concerned with the definition of

formal models and formal provability than the AC models described in section 2.4, most of which emerged from the security research community. PBM system research has been more concerned with implementation architectures, language expressiveness, probabilistic analysis of network events and addressing of managed resources. Once again, the PBM systems are analysed with particular concentration on the organisational models embodied within them.

2.5.1 IETF Policy Model

The Internet Engineering Task Force (IETF) policy model was developed primarily in order to specify network policies [moore01a]. It considers policies as rules that specify actions to be performed in response to defined conditions:

```
if <condition(s)> then <action(s)>
```

The condition-part of the rule can be a simple or compound expression specified in either conjunctive or disjunctive normal form. The action-part of the rule can be a set of actions that must be executed when the conditions are true. An example of the type of rules that administrators can specify is the following policy which defines the traffic priority level between two nodes in the network.

```
if ((sourceIPAddress = 171.11.12.1 AND destinationIPAddress =  
172.3.3.2) OR (sourceIPAddress = 172.3.3.2 AND destinationIPAddress =  
171.11.12.1)) then set Priority := 6
```

The IETF defines a basic policy based management architecture, pictured in figure 2.6, which forms the basis of many commercial implementations.

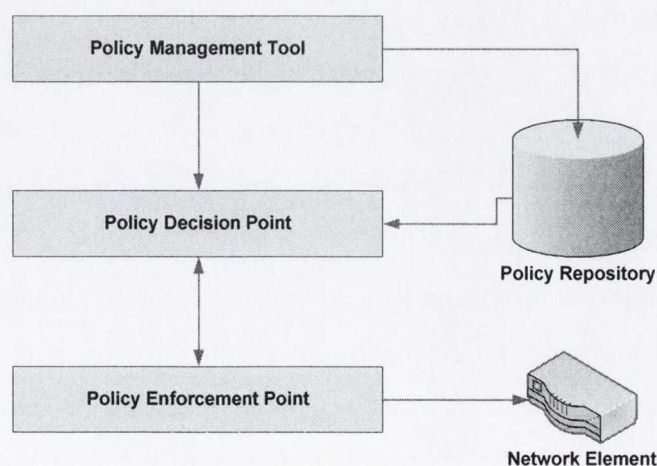


Figure 2.6 IETF Policy Architecture

A *Policy Decision Point* (PDP), also referred to as a policy consumer, retrieves policies from the policy repository, interprets the policies and sends them to *Policy Enforcement*

Points (PEP) (e.g. routers, bridges) to enforce them. A PDP may need to translate the set of rules it receives from the repository to a format that is understood by the corresponding PEPs. Policy rules can be tagged with one or more roles. A role represents a functional characteristic or capability of a resource to which policies are applied, such as backbone interface, frame relay interface, BGP-capable router, web-server, firewall, etc. The PDP uses the *policy role attribute* of a policy rule to identify the policy enforcement points it needs to send the rule to. Other functions of the PDP include receiving policy decision requests from PEPs and returning policy decisions to them. PDPs can also send asynchronous policy decisions based on state updates or external requests. Policy enforcement involves the PEP applying actions according to the PDP's decisions, based on current network conditions. These conditions can be based on static parameters (e.g. source/destination IP address) or on dynamic parameters (e.g. current bandwidth availability, time of the day). A PEP enforces the policy by, for example, permitting or forbidding requests or allocating packets from a connection to a particular queue or changing message priorities or message tags. A PEP and PDP can be combined into a single component.

The IETF do not define a specific language to express network policies, instead they define a generic object-oriented information model for representing policy information following the rule-based approach described above. Early attempts at defining a language by the IETF [strassner97] have been abandoned. The Policy Core Information Model (PCIM) [moore01] extends the Common Information Model (CIM) [dmtf99], defined by the Distributed Management Task Force (DMTF), with classes to represent policy information. The CIM defines generic objects such as managed system elements, logical and physical elements, systems, service, users, etc, and provides abstractions and representations of the entities involved in a managed environment including their properties, operation and relationships. The PCIM defines a set of classes and relationships that provide an extensible means for defining policy control of managed objects. These classes and relationships represent the structure, not the contents, of a policy. The actual policy content, in whatever specification language is used, is provided by sub-classing classes to derive technology and vendor-specific conditions, actions, and other elements. One example of a concrete specification language based upon PCIM is the Simplified Policy Language (CIM-SPL) standard proposed by the DMTF [dmtf 07].

PCIM Policy rules can be grouped into nested policy groups to define policies that are related in any application specific way, although no mechanism exists for parameterising rules or policy groups. Both actions and conditions can be stored separately in a policy repository and reused in many policy rules. Policy rules can be associated with a priority value to resolve conflicts between conflicting rules. However, this approach is not scalable in large networks with a large number of rules specified by a number of different administrators since different administrators will tend to apply different priority ratings to the various classes of policy rules that they author. It is often impossible for a policy author to accurately judge the priority level that should be applied to a policy rule relative to the priority of an existing policy rule. For this approach to scale properly, each administrator would require an accurate picture of the relative priority of all policy rules with respect to the overall goals of the system. In reality, administrators tend to have focus especially on a particular aspect of the system, which can lead them to over-emphasise the importance of those rules which govern the resources which are their responsibility. For example, if an administrator authors a policy rule with a certain priority level and subsequently finds that this rule is ignored in favour of another with a higher priority, the administrator may simply increase the priority level of the new policy, leading to a cycle of priority escalation. Several extensions and modifications to the model have since been proposed by the IETF [moore03] including administrator-specified decision strategies for rule evaluation and conflict resolution, although these, once again, depend on the administrators who define these strategies having an accurate system-wide view of policy priorities.

Apart from the PCIM, the IETF policy framework working group have proposed an information model to represent policies that administer, manage, and control access to network QoS resources for integrated and differentiated services QoS management [snir03], network device QoS datapath mechanisms [moore04] as well as an LDAP policy schema [strassner04]. The philosophy of the IETF is that business policies expressed in high-level languages, combined with the network topology and the QoS methodology to be followed, will be refined to the policy information model, which can then be mapped to a number of different network device configurations. Vendors following the IETF approach are using graphical tools to specify policy in a tabular format and automate the translation to PCIM.

The IETF definition of a policy is not particularly suitable for the specification of access control policies. However, it is possible to include the particular *subject-target-action* for which policies are specified as part of the IETF policy condition as follows:

```
if <subject=s AND target=t AND action=a> AND <condition(s)> then <action(s)>
```

However this is not a good way to specify policies – by embedding the information about the subject, target and action to which the policy applies within the policy condition, the problems of maintaining and administering the policy set are greatly increased. Furthermore, the PCIM and much of the other work that has been done in the IETF is much more focused on organising managed resources and it does not possess grouping abstractions that are suitable for modelling human organisations with their departments, teams and many different types of groups. Finally, the administration of the system is not managed by the policy system itself and it is assumed that it is carried out by a network administrator or other technical officer, rather than being distributed throughout the organisation.

2.5.2 Policy Description Language

The policy description language (PDL) is an event-based language originating at the network computing research department of Bell-Labs [lobo99]. PDL policies are specified using the paradigm of active databases [dayal95] to define a policy as a function that maps a series of events into a set of actions:

```
on <event> if <condition> take <action>
```

This differs from the IETF definition of policy by virtue of the explicit specification of the event, although if one considers that the event is merely a special clause of the condition, the two forms reduce to the same fundamental form. However, by specifying the event explicitly, the requirement that the managed elements must be capable of detecting relevant events is removed. The language can be described as a real-time specialised production rule system to define policies. The syntax of PDL is simple and policies are described by a collection of two types of expressions: policy rules and policy defined event propositions. Policy rules are expressed in the form:

```
event causes action if condition
```

Policy defined event propositions are expressed of the form:

```
event triggers policy-defined-event if condition
```

Events can be primitive or complex. There are two types of primitive events: policy defined events, which are only generated by the policy defined event propositions, and system events, which are generated by the environment. Primitive event classes can define attributes. Instances of classes take actual values for those attributes that can be referenced by other events, actions or conditions within the same rule. Primitive events can be composed to form complex events that enable policies to be enforced under any of the following situations:

- If two events e1 and e2 occur simultaneously.
- If an event e does not occur.
- If an event e2 immediately follows an event e1.
- If an event e2 occurs after an event e1.

The language has clearly defined semantics and an architecture has been specified for enforcing PDL policies. Work on conflict resolution for policies written in PDL is described in [chomicki00]; extensions to the language to specify workflows for network management can be found in [kohli99]; later extensions to introduce monitors - constraints on the actions that the network manager can execute simultaneously - are described in [chomicki01]. More recently, an extension of PDL, PPDL has been proposed, which allows the specification of user-defined preferences on how to apply monitors [bertino05]. The PDL language has been used to program Lucent switching products [virmani00] and proves to be powerful in a variety of network operations and management scenarios.

Despite its expressiveness, PDL does not easily support access control policies, nor does it support the composition of policy rules into roles, or other grouping structures. Access control policies could be represented as a PDL policy if the *subject-action-target* accesses were mapped to particular events and the actions were limited to the set {permit, deny}. Then authorisation policies could be specified as follows:

```
subject-action-target causes permit|deny if condition
```

Or, a more sophisticated mapping from combinations of subject, target and action to events could be employed. However, this is not a convenient way to specify access control policies and the language was not intended for the purpose and thus lacks useful grouping abstractions for the domain. Nor does it provide anything in the way of support for modelling the system's administration.

2.5.3 Open Distributed Programming Reference Model (ODP-RM)

The group working on the International Standards Organisation (ISO) Open Distributed Processing Reference model (RM-ODP) have defined an enterprise language as part of the RM-ODP Enterprise Viewpoint [iso00], which incorporates concepts such as policies and roles within a community. A *community* in RM-ODP terminology is defined as a configuration of objects formed to meet an objective. The objective is expressed as a *contract*, which specifies how the objective can be met, and a *configuration* is a collection of objects with defined relationships between them. The community is defined in terms of the following elements:

- The enterprise objects comprising the community,
- The roles fulfilled by each of those objects and the relationships between them,
- The policies governing the interactions between enterprise objects fulfilling roles,
- The policies governing the creation, usage and deletion of resources,
- The policies governing the configuration of enterprise objects and assignment of roles to enterprise objects,
- The policies relating to the environment contract governing the system.

Policies constrain the behaviour of enterprise objects that fulfil actor roles in communities and are designed to meet the objective of the community. Policy specifications specify what behaviour is allowed or not and often contain prescriptions of what to do in case some rule is violated. Policies in the ODP enterprise language thus cover the concepts of obligation, permission and prohibition. The ODP enterprise language is an abstract language in the sense that it does not prescribe the use of any particular notation. Recently, there have been a number of attempts to define precise languages that implement the abstract concepts of the enterprise language. These approaches concentrate on using UML to graphically depict the static structure of the enterprise viewpoint language as demonstrated by [steen00], as well as languages to express policies based on those UML models. Steen et al. [steen99, steen00] propose a language to support the enterprise viewpoint where policy statements apply to a role, the subject of the policy, and such statements represent either a permission, an obligation or a prohibition for that role. Logical conditions are expressed using the Object Constraint Language (OCL) [omg99] in the before-, if- and where- clauses defined in the grammar.

```
[R?] A <role> is (permitted | obliged | forbidden) to (do <action> [before  
<condition>] | satisfy <condition>)[, if <condition>][, where <condition>][,  
otherwise see <number>]
```

The expressiveness of the language is limited in a number of ways. It does not allow composition of policies or constraints for groups of policies. Constraints cannot be specified to restrict the activation/deactivation of roles or the assignment of users and permissions in roles. The role is the only grouping abstraction employed within the community. In general, the RM-ODP Enterprise Viewpoint provides a flexible but relatively structureless means of specifying a community's internal organisation. While the fact that it is defined at a relatively low level of abstraction allows a great deal of flexibility in describing the interactions within an organisation, the problem of defining grouping abstractions and the relationships between them in order to create a model of the organisation is largely beyond its scope. An organisation can be modelled, but it must be built, bottom-up, using low level constructs.

2.5.4 PONDER

Ponder [damianou02b] is a complete PBM framework which includes a specification language¹. It is one of the more expressive policy languages, addressing both access control (authorisation policies) and management (obligation policies). The Ponder language also includes a wealth of grouping abstractions and relationships which allow administrators to group together entities in a variety of ways in order to simplify the specification of policies which have a subject or target scope that incorporates multiple entities. Due to the fact that it is a complete policy management framework with a rich collection of grouping abstractions, Ponder is examined in particularly close detail here as it provides a good illustration of the complex requirements faced by a PBM system in capturing a model of the organisation. As Ponder allows both obligation policies and authorisation policies to be applied to roles, Ponder is an example of Role Based Management (RBM), a superset of RBAC. Ponder authorisation policies use the standard subject-action-target form.

```
Inst auth+ switchProfileOps {
  subject /NetworkAdmin ;
  target <ProfileT> /Nregion/switches ;
  action load(), remove(), enable(), disable() ;
}
```

Ponder Authorisation Policy: Members of the NetworkAdmin domain are authorised to load, remove, enable or disable objects of type ProfileT in the Nregion/switches domain.)

A subset of the Object Constraint Language (OCL) [omg99] is used to specify Ponder constraints. Constraints are equivalent to conditions in the `on <event> if <condition>`

¹ All examples of policy specifications in this section are taken from [Damianou02b]. The explanations and discussions are, however, original.

take <action> policy form. Constraints limit the conditions under which the action specified in the policy will be taken (which will be permit or deny if it is an authorisation policy, depending on its modality). Ponder constraints can relate to:

- Subject/target state – constraints based on the object state as reflected in terms of attributes at the object interface.
- Action/event parameters – constraints based on event parameter values in obligations or action parameter values in authorisations or refrains.
- Time – constraints which specify the validity periods for the policy.

Policy constraints can be considered as conjunctions of basic constraints, which can be either time or state based. Time-based constraints can be compared for possible overlaps and state based constraints can be identified by Ponder as either simultaneously satisfied or mutually exclusive if they relate to states of the same system component. An example of a Ponder obligation policy with constraints is as follows:

```
Type oblig printJob (set S, domain T, int maxpages) {
  on print(job, sender);
  subject S ^ {sender};
  target T->select(t | t.state = 'idle');
  do print(job) -> sender.mail("job
  re-directed");
  when job.pages > maxpages;
}
```

Example of a Ponder Obligation Policy. The sender (of the print job), executes the job on targets (printers within domain T) which are idle and sends an email indicating that the job has been redirected when the number of pages in the job is greater than maxpages.

In addition to the positive and negative authorisation policies and obligation policies, Ponder introduces a *refrain policy* which is a negative obligation policy. It specifies what a subject *must not* do. Although this is semantically equivalent to a negative authorisation policy – it is required by Ponder to specify instructions to automated subjects when the target is not trusted to enforce a negative authorisation policy or when the target being acted upon is internal to the subject and its invocation does not involve a method call on a target object. The combination of these 4 types of policy with Ponder's flexible constraint specification language and the ability to define types of policies which can be instantiated with parameters make Ponder very expressive.

Domains

Domains are the fundamental grouping abstraction used by Ponder in order to provide structure on the set of subjects and the set of targets within the system. Domains also serve to group policies together through the *role* and *group* structures of the language, described below. The use of domains as a grouping abstraction by researchers in

Imperial College London for policy based management precedes the development of Ponder [sloman94b]. Domains provide a general-purpose hierarchical means of grouping objects and they have the special property that a policy applying to a domain will propagate to its sub-domains, meaning that policies can be specified which will apply to large numbers of objects in a scalable way. Membership of domains is explicit – a Ponder domain simply holds a list of references to the objects that are contained by the domain and sub-domains are treated as objects when referenced from a parent domain. A sub-domain is not a subset of the parent domain, in that an object included in a sub-domain is not a direct member of the parent domain, but is an indirect member. A sub-domain or object may have multiple parent domains.

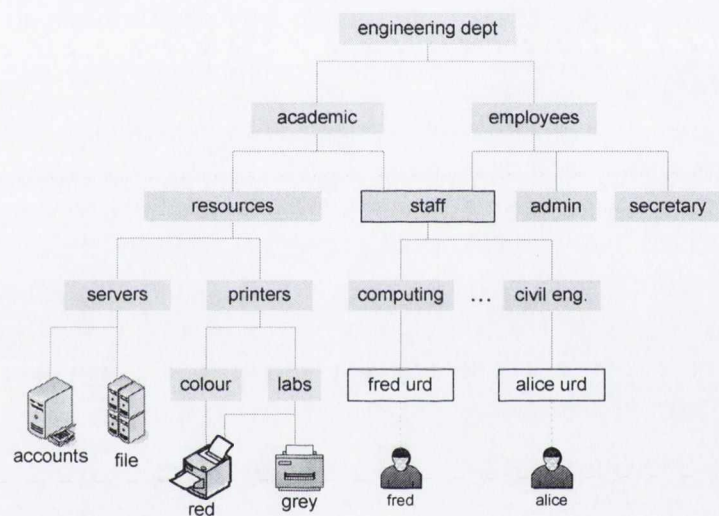


Figure 2.7 Ponder Domain Structure Example

Domains can be referenced by path-names, with ‘/’ being used as a delimiter. For example, in figure 2.7, the *staff* domain can be equally referenced as */engineering-dept/academic/staff* or */engineering-dept/employees/staff*. An object may have different local names with multiple parent domains, where ‘/’ is used as a delimiter for domain path names. Policies normally propagate to members of sub-domains, so a policy applying to domain */engineering-dept/academic/resources* will also apply to members of domains */engineering-dept/academic/resources/servers* and */engineering-dept/academic/resources/printers*. Objects may be moved from one domain to another and any policies which apply to the new domain will automatically apply to the moved object replacing the policies from the old domain – this provides one means of managing the policies that apply to objects.

The subject and target components of Ponder policies are made up of *Domain Scope Expressions* (DSE). DSEs allow the combination of domains to flexibly describe the sets of objects to which policies apply. DSEs were originally introduced in [yialelis96], with semantics described by [marriott97]. They include operators for set union, set difference, set intersection (using the ‘+’, ‘^’ and ‘-’ operators respectively). They also include unary operators which specify different sub-sets of the objects in a domain (the ‘@’ and ‘*’ unary operators return all non-domain members of a domain, and all members of a domain including sub-domains respectively). Ponder extends DSEs to support calls to Ponder’s domain-object library.

Groups & Roles

In Ponder a *group* is simply a packaging construct which groups an arbitrary collection of policies together. A Ponder *role* is simply a group that is associated to a domain, by virtue of having the domain as a common subject of the policies in the group. Since domains are used by Ponder as the fundamental grouping abstraction, a role merely needs to be specified as a domain scope expression and a set of policies. Users are mapped to the role through their membership of the domain. Like policies, Ponder roles can be specified as parameterised types, objects of these types can then be instantiated by specifying parameter values.

```
inst role /managementInfo/roles/securityAdminT
{
  inst auth+ userMgmtAC {
    target ds = /system/servers/domainServer;
    action ds.addUser(), ds.deleteUser();
  }

  inst auth+ userBackupMgmtAC {
    target bs = /system/servers/backup;
    action bs.createUserAcc(), bs.deleteUserAcc(),
    bs.setPreferences();
  }

  inst auth+ backupShutdown
  {
    target t = /system/serv/backup;
    action t.shutdown;
  }
} @ /managementInfo/roles/securityAdmin
```

Example of a Ponder role. The Security Admin role is authorised to add and delete users, on both the domain server and the backup server as well as shut down the backup server. All non-domain objects that are members of the /managementInfo/roles/securityAdmin domain are members of this role

Role Inheritance & Role Hierarchies

Ponder’s *type specialisation* relationship relates roles through the mechanism of inheritance, using the keyword *extends*. The semantics of this relationship are directly

borrowed from object oriented programming, as is the syntax. Type specialisation creates a role hierarchy which corresponds to the *isa* hierarchy identified by Moffett [moffett98]. Inherited roles can add new policies and override policies with the same name. Ponder supports multiple inheritance. In the cases where two of the base-classes have policies with the same name – which is one significant problem in multiple inheritance schemas in general [lupu98] – Ponder can generate a warning to the policy author who has specified the role inheritance. Alternatively, the names of policies can be prefixed with the name of the class from which they are inherited in the derived class to eliminate ambiguity.

Role Relationships

A Ponder *role relationship* is a grouping abstraction used to collect a set of policies which relate to the rights and duties of roles towards each other. It can also include policies related to resources that are shared by the roles. Relationships were introduced in [lupu98] along with a syntax for defining the dialogic aspects of role relationships (the sequences of messages that roles are permitted to exchange). However, this has not been implemented in the Ponder language and thus Ponder's role relationship structure is not expressive enough to define many of the interaction protocols that might be required by applications. Role relationships are collections of policies (and other elements of the Ponder language) where roles are used in the place of subject and target domain-scope expressions.

Management Structures

The final grouping abstraction defined in Ponder is the *Management Structure*. A management structure is a composite policy containing definitions of roles, relationships and other nested management structures. Basic policies, which refer to the roles and relationships defined in the management structure, as well as constraints in the form of meta-policies can also be included. Management structures are intended to allow the modelling of large organisational structures such as branches, departments, divisions and whatever other structural units are required by the application domain. Management structures are structuring mechanisms used to compose policy specifications based on organisational or network structure, in order to cope with large-scale policy specifications. Management structures are purely grouping mechanisms and Ponder does not allow the specification of relationships or interaction protocols between them.

Administration and Meta-Policies

Ponder policy objects, roles and other composite policies are managed objects like any other. References to any of them can be added to Ponder's domains, just like any of the other objects within the system. This enables the creation of policies whose targets are other policies, roles or relationships and provides the ability to specify, through policy, which roles are permitted to add, delete and edit specific sets of policies. Obligation policies can be defined to specify what actions must be performed on policy objects when certain events occur. In addition, meta-policies allow for constraints on the final form of edited policies. Meta-policies specify constraints over a set of policies, on the permitted types of policies or their policy elements. These constraints apply to policies within a specific scope, and limit the permitted policies in the system, or disallow the simultaneous execution of conflicting policies. The body of a meta-policy specifies the constraint as a series of OCL expressions separated by semicolons. The expressions can be Boolean or navigation expressions. If any of the Boolean expressions evaluates to true, execution stops, and the action following the raises-clause is executed. This way, a series of related constraints can be specified within the same meta-policy.

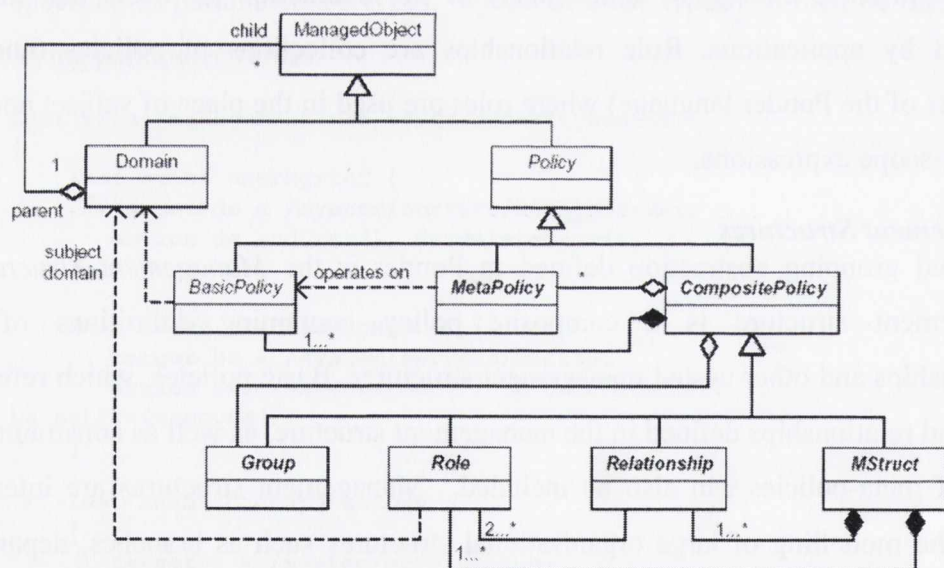


Figure 2.8 Ponder Composite Policy Object Class Relationships

Delegation Policies

Another element of the Ponder language which relates to the administration of the system is the *Delegation Policy*. Delegation policies specify under what conditions a user may temporarily transfer her access rights to another user in the system (called a *grantee*) to perform actions on the user's behalf. A Ponder delegation policy is always

associated with an authorisation policy, which specifies the access rights that can be delegated. Negative delegation policies forbid delegation of certain actions. Delegation policies are not meant to be used for assignment of rights by security administrators, but for the temporary transfer of a particular access right. They allow a limited decentralisation of administration of the system, albeit a decentralisation that is limited to immediate tasks and not organisational functions. Delegation policies can be constrained by temporal conditions or a condition based upon the state of system / subject / target / grantee / action attributes, or on the maximum number of cascaded delegations.

Group Decision Making

The above features combine to provide a set of tools for a potentially powerful, self-managed system. There is no need for specialised administration roles, since any normal role can be granted permission to update policies - in exactly the same way that roles can be granted permission to access non-policy resources. Ponder's expressiveness also allows group decision making processes to be specified within policies. For example, authorisation and obligation policies can be used to specify "backing rules" [varadharajan96]. Backing rules specify a condition that must be true to ensure that the individual carrying out the action has the backing of the group. It is assumed that the backing condition can be specified and monitored by the underlying monitoring service, and then specified as an event to trigger obligation policies. Although Ponder can express certain backing conditions, it is acknowledged that to express arbitrary backing policies, a scripting language would be required [damianou02b] and the problem of integrating Ponder with a voting system has not been addressed.

2.5.4.1 Ponder's Organisational Model

Although the Ponder framework does provide the building blocks for enabling a self-managed organisation, it is only a framework. Ponder is extremely expressive and applicable to a large subset of the typical requirements of policy based management systems. It provides a wealth of grouping abstractions, many of which can group together virtually any of the structural units of the language into administrative units. Ponder can be used to construct a rich structural model of the organisation, made up of management structures, groups of policies and roles. Role relationships provide a means of relating some of the structural elements together, while the nesting of

management structures provides another means of relating the structural elements together into a model of the organisation – a bottom-up and top-down model respectively. Meta-policies and policies which apply to policy objects allow a precise definition of administrative authority, while delegation allows a certain measure of decentralisation of administration. The roles responsible for the system's administration can be integrated into the general organisational model, providing a more complete model of the organisation.

Flexibility Comes at the Cost of Well Defined Semantics

Despite and because of Ponder's flexibility, it provides little in the way of answers to the question of how a policy based management system can best model a human organisation. Ponder allows the creation of virtually any mapping between the human organisation and authority vis-à-vis the policy system. The Ponder domain construct is the source of this flexibility. By modelling every Ponder entity as an object, and modelling domains as hierarchical sets of object references and by allowing objects to be referenced by multiple domains and by allowing arbitrary domain scope expressions to be the targets and/or subjects of policy rules, a great deal of flexibility is gained, but many of the problems of organisational modelling are pushed beyond the scope of the system. Much of the organisational model is dependent on how domains are populated. For example, the choice of which domains newly created objects are added to – including all of Ponder's structural elements – has a significant bearing on how the organisation works. The use of roles is at the discretion of the policy author and policies can be specified for arbitrary sets of subjects, as well as for roles. Furthermore, using extension through inheritance can tend to freeze policy after deployment rather than allowing it to be modified progressively. Although meta-policies can be authored to constrain policies, this once more pushes the problem of organisational modelling onto the administrators. Similarly, the targets of policies can contain arbitrary domain scope expressions – meaning that designing and maintaining organisational schemas that map sets of users to resources is dependant on human administrators devising an organisational model and specifying it with meta-policies.

Flexibility Increases Complexity of Policy Processing and Comprehension

The reliance upon the flexibility of domain structures also creates several problems for administration. If objects can appear in multiple domains and policies apply in a cascading manner to domains and their sub-domains, the affects of updating policies, or of adding object references to domains, soon become difficult to predict. In order to

analyse the affects of an update to a policy in the system, to check for policy conflicts and functional correctness, all of the objects affected by the updated policy and all of the other policies that affect them must be identified and considered. Any updated policy can affect every object in a domain and any of these objects may be referenced by an arbitrary number of other domains any of which can have policies applied to them. Furthermore, any policies which have subject or target domain-scope expressions which resolve to include any of the objects affected by the updated policy must also be considered. Finally, any meta-policies which relate to the updated policies, or to the sets of policies which reference any of the affected objects must be considered. This presents a potentially large search space in order to identify the policies which must be evaluated in order to identify policy conflicts, and, depending on the sophistication of the algorithm that identifies policies that are potentially relevant and the rigour of the organisational schema defined by the administrators, many computationally expensive function calls may have to be performed in order to evaluate whether policies which contain arbitrary constraints conflict with the updated policy set for each object that is referenced by the updated policy. Although the rapid advances in the speed of modern computing hardware means that such potentially computationally expensive processes may not be problematic, PBM systems require high levels of responsivity and experiences of their deployment in industry show that policy sets tend to grow to large sizes quickly once deployed, when they are required to deal with the quirks of real world organisations [zhang05], thus minimising the size of the policy set to search will always have a beneficial effect on performance.

More problematic than the complexity of analysing policy updates is the problem faced by the human administrator in mapping updates to the policy set into functional requirements of the system. Human administrators generally have a functional requirement in mind when they make updates to the policy set. In order for them to author policies which accurately reflect this function, they need to be able to understand the consequences of the update, or to put it another way, the greater the complexity of the consequences, which can be expressed as the set of the policies that need to be considered, and is equal to the set of policies that apply to the set of objects referenced by the updated policy, the greater the difficulty in verifying that an updated policy will fulfil the administrator's functional requirement. In many cases, there is no way to automatically eliminate many policies from this functional verification which are

concerned with non-related constraints, which may still have a combinatorial effect which is undesirable from a functional point of view. Thus, while Ponder does provide abstractions which allow great flexibility in modelling an organisation, in order to create a system that can more easily map a functional requirement to a policy specification, there is a need to devise an organisational schema which implements a more structured organisational model and limits the scope of the consequences of policy changes to a much more restricted set.

2.5.5 Open Architecture for Secure Interworking Services

The Open Architecture for Secure Interworking Services (OASIS), developed at Cambridge University [bacon02] is a role-based access control architecture for achieving secure interoperation of services in an open, distributed environment. OASIS Services define roles and implement formally specified policy for role activation and service use. Clients present credentials to named OASIS services which are then checked against the policy specification in order to activate a role or invoke a service. OASIS uses the role definition language (RDL) [hayton98], as its policy specification language. RDL is a formal language based on Horn clauses, where sets of rules indicate the conditions under which a client may obtain a role. The conditions for entry to a role are described in a role membership rule, expressed in terms of the credentials that a user must present to enter the role, together with constraints on the parameters of those credentials. Roles in RDL are also considered to be credentials; a client that successfully presents credentials to gain entry to a role is issued a role membership certificate (RMC), which is presented by the client with subsequent service invocations. OASIS supports parameterised roles, giving the ability to add attributes which are relevant to access control to each role instance, and then make decisions based upon these attributes. For example, the role *paidUpSubscriber* role may have an attribute *rate* to indicate the level of subscription payment. OASIS rules may contain conditions that are monitored throughout the time active roles are dependent on them, if these conditions become false, the role is automatically revoked.

OASIS is unusual in that it eschews the notion of role hierarchies. Its creators argue that: “one of the main reasons for using RBAC is that it provides a natural way to model constraints such as separation of duties [and] role hierarchies complicate the specification and enforcement of these constraints”. They believe that the use of such

hierarchies in RBAC “arises mainly through the influence of object oriented modelling” and “are not convinced of their utility” in this domain [hine00].

Instead of utilising role hierarchies, OASIS uses role activation dependency as the fundamental role-role relationship. Individuals must meet a set of conditions in order to activate roles before being granted the permissions and obligations associated with the role. Role activation is governed by a set of rules, specified in logic, which are parameterised and dynamic. The conditions for role activation may include prerequisite roles that must have been activated by the user prior to activating the current role. Prerequisite roles offer much of the same ability to reuse policies that are used by role hierarchies – the major difference being that policies from pre-requisite roles cannot be over-ridden, as they can be in inherited roles.

The following is an example of a role membership rule in RDL, which specifies that a user x is a member of the *SeniorHaematologist* role if she is also a member of the *Haematologist* and the *SeniorDoctor* roles.

$$\text{SeniorHaematologist}(x) \leftarrow \text{Haematologist}(x) \wedge \text{SeniorDoctor}(x)$$

Like ARBAC97, OASIS uses a separate administrative set of roles to administer the system [belokosztolszki02]. These roles are constrained by meta-policies, which ease the task of policy administrators since they constrain policy design and allow administrators to adjust policies without breaking the fundamental requirements of the organisation. OASIS allows limited self-management by users. Users can manage a subset of the resources in the system, including the access control policies for the resources that they manage - for example their personal files. OASIS introduces the notion of appointment to replace delegation. A user in a role acting as an *appointer* grants another user, the *appointee*, a credential which may be used to activate one or more roles. Roles activated on the basis of an appointment are usually associated with some tasks or responsibilities, and encapsulate the privileges granted by the appointer. Appointment differs from delegation in that an appointer can grant a credential which allows the appointee to activate roles that the appointer can not activate. This removes several of the problems typically associated with delegation in DAC models, such as cascaded delegation (since the appointee will typically not be in a role which permits him to be the appointer of the appointed certificate). Appointment also surmounts the requirement for partial delegations in role based delegation. Partial delegation refers to

the situation where only a subset of the permissions associated with a role are delegated. This is frequently desirable according to the principle of least privilege, but it breaks the formal semantics of RBAC. In OASIS, the precise roles which can be appointed can be specified, as well as specifying rules for who can activate the appointee role. When an appointment is made, a revocation certificate is issued. Appointments can be revoked in three different ways depending on requirements: *appointer-only* appointments can only be revoked by the appointer; *appointer-role* appointments can be revoked by anybody who can activate the appointer role, and are designed to allow appointments to be made in such a way as to limit the spread of damage if a misbehaving party makes inappropriate appointments by increasing the number of people who can revoke it; finally *system-managed* appointments are revoked automatically if certain conditions are met which might be related to time, tasks or sessions.

Appointment allows the decentralisation of administration to a certain extent, by introducing an appointee role. Members of that role can distribute privileges through appointment. Appointees can only re-delegate appointed privileges if they are also a member of the appointer role. However, this remains a relatively centralised system of administration compared to DAC models, where each user in the system can delegate privileges. The appointee role in OASIS is best viewed as a special case of an administration role. As well as roles, OASIS contains the notion of management domains to support large organisations which contain many interoperating domains, such as the health service [eyers06]. Each OASIS service is defined within a domain, and these domains can be administered autonomously. Although these domains are not related to one another through the model, OASIS's credential-based service architecture allows credentials from one domain (such as a role membership certificate) to be specified as requirements for entering a role in a different domain. However, such relations, defined within role activation conditions merely link together roles within the domains and do not capture how the various domains are related to one another.

2.5.6 XACML

The Extensible Access Control Markup Language (XACML) has been defined by the Organisation for the Advancement of Structured Information Standards (OASIS - not related to the Cambridge RBAC system above). In February of 2003, the OASIS membership at-large approved XACML version 1.0 as an OASIS Standard [oasis03]. In February of 2005, XACML version 2.0 was approved as an OASIS Standard

[oasis05]. XACML is designed to be a common language for defining security policy across an enterprise and is based upon XML and XML-Schema, due to the ease with which XML can be extended and the widespread support for XML based applications. XACML defines the basic requirements of a policy language for expressing information system security policy as the following:

- To provide a method for combining individual rules and policies into a single policy set that applies to a particular decision request.
- To provide a method for flexible definition of the procedure by which rules and policies are combined.
- To provide a method for dealing with multiple subjects acting in different capacities.
- To provide a method for basing an authorization decision on attributes of the subject and resource.
- To provide a method for dealing with multi-valued attributes.
- To provide a method for basing an authorization decision on the contents of an information resource.
- To provide a set of logical and mathematical operators on attributes of the subject, resource and environment.
- To provide a method for handling a distributed set of policy components, while abstracting the method for locating, retrieving and authenticating the policy components.
- To provide a method for rapidly identifying the policy that applies to a given action, based upon the values of attributes of the subjects, resource and action.
- To provide an abstraction-layer that insulates the policy-writer from the details of the application environment.
- To provide a method for specifying a set of actions that must be performed in conjunction with policy enforcement.

The XACML specification includes a data-flow model which describes the basic components of an XACML implementation architecture (an extension of the IETF policy architecture) and the sequence of messages that pass between them, as illustrated in figure 2.9.

In addition to the PEP and the PDP of the IETF architecture, XACML includes the notion of a Policy Administration Point (PAP) and a Policy Information Point (PIP). Access control policies can be considered to have the general form:

<subject> - <action> - <target> - <condition> - <effect>

This can be translated into an IETF policy by including the *subject-action-target* part as a clause of the condition, or translated into a PDL policy by mapping it to an event, and defining the IETF *action* as an effect, which can be either *permit* or *deny*. Broadly speaking this is the same model of policy specification as that utilised by most RBAC and other security-policy specification languages. In XACML *policy sets* contain collections of policies, which can themselves contain multiple rules and each of these elements can define its own *subject-action-target* matching rules, which enables PDPs to efficiently filter policies with respect to their relevance to the *subject-action-target* specified in the policy request.

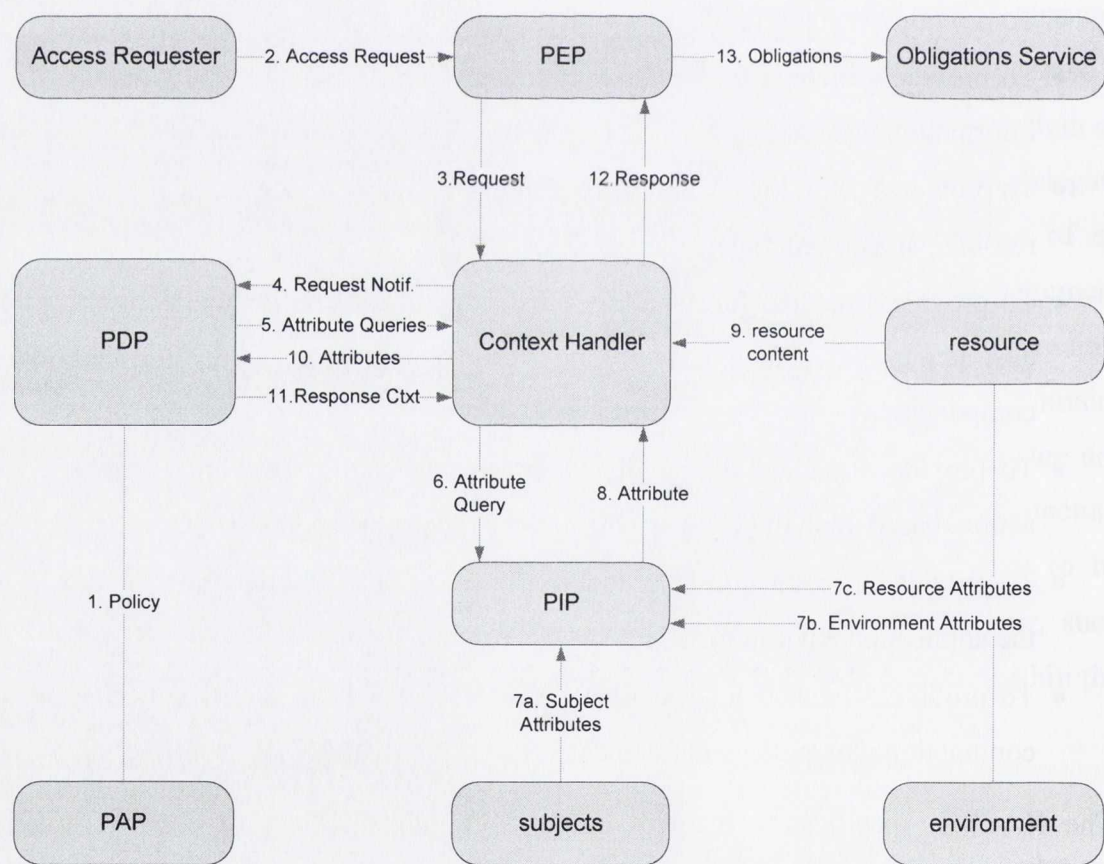


Figure 2.9 XACML Data-Flow Model

Rather than concentrating on providing a formal model for security policy specification, XACML adopts a pragmatic approach focusing on standardising the specification of conditions and the interchange formats of policy information. XACML supports the

concept of *contexts*, similarly to the C-TMAC and OrBAC models described in appendix 1, which provide the PDP with attribute information so that it can evaluate policies within a particular context. XACML divides the context into four sub-spaces, namely contextual information relating to the subject, the environment, the resource and the action and specifies the communication sequence by which contextual information related to each element can be discovered by the context handler. One of the most significant innovations of the OASIS XACML standard is that, rather than providing a method for detecting and correcting policy conflicts, XACML policy sets and policies are associated with a *policy combining algorithm* and *rule combining algorithm* respectively. This allows policies and rules to be specified in such a way that policy conflicts are resolved according to an algorithm specified by the administrator.

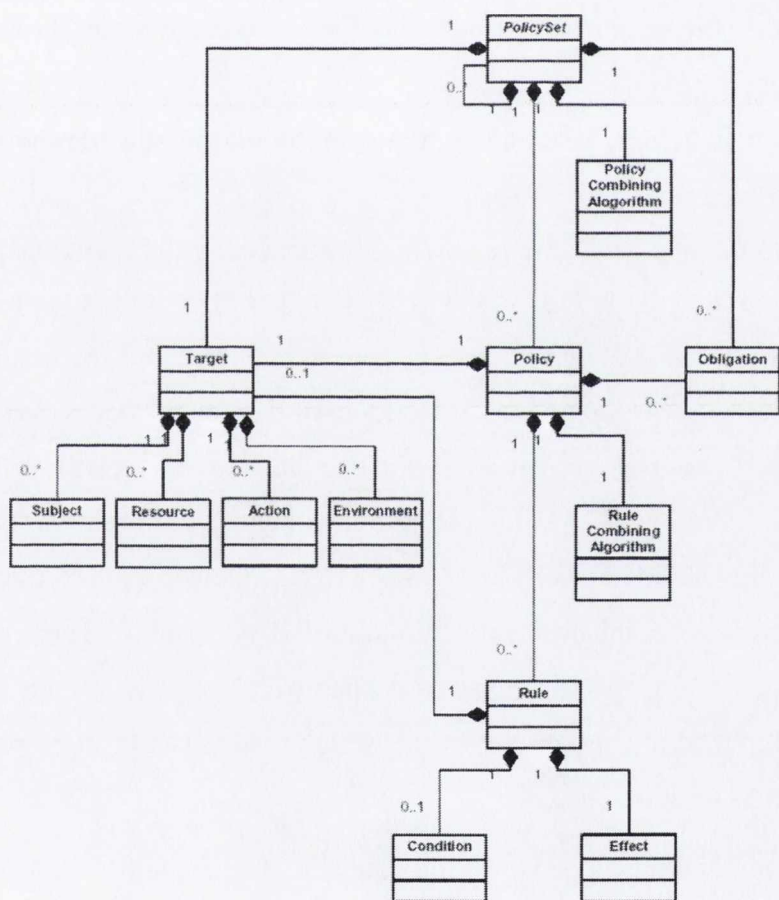


Figure 2.10 XACML Policy Model

The XACML specification itself is designed to be extensible. The core specification for XACML 2.0 does not provide any support for grouping mechanisms such as roles or hierarchically organised resources. However, six profiles which extend XACML were released simultaneously with the core specification, namely: Security Assertion Markup

Language (SAML) 2.0, XML Digital Signature, Privacy Policy, Hierarchical Resource, Multiple Resource, and Core and Hierarchical Role Based Access Control (RBAC). These add implementations of the standard RBAC models to the XACML standard as well as providing a range of methods for grouping resources together for convenience. Similarly, XACML does not mandate any particular model for administration, but its flexibility and expressiveness means that it can support ARBAC and many other administrative models. Thus, XACML provides solutions to many of the engineering, interchange and formatting problems which occur in policy based management, but it is neutral to the particular organisational model employed, being flexible enough to incorporate a wide range of grouping abstractions and schema.

2.5.7 Other PBM systems.

In addition to the PBM systems described thus far, there are a wide variety of other approaches to policy based management that have been developed for research and production systems. However, the systems presented above provide a reasonable cross-section of policy based management systems from an organisational modelling point of view. Nevertheless, some of the other research in the area is briefly mentioned below.

Credential Based Trust Specification Languages

A common solution to the problem of managing access control over the Internet, where entities may be unknown to each other, is based upon the use of digital certificates, or credentials which represent information about the client that is certified by trusted entities. The client presents its credentials to the server and access control decisions are made based upon these credentials. The OASIS RBAC framework, described above, is an example of such a mechanism integrated into an RBAC system. There are various other access control policy specification approaches which rely largely on this simple mechanism and allow administrators to specify which credentials must be presented in order for an action to be authorised.

Trust management frameworks combine authentication with credentials-based authorisation [grandison00] and are used for applications such as signed email, active networks and e-commerce. In [blaze98, blaze99], two trust management applications are presented: the *PolicyMaker* and its successor *KeyNote*. Both of these applications are used to answer signed queries of the form “*does a set of requested actions r , supported by credential set C , comply with policy P ?*” where the credentials can be public key certificates with anonymous identity. Both policies and credentials are

predicates specified as simple C-like and regular expressions. In this context a policy is a trust assertion that is made by the local system and is unconditionally trusted by the system. IBM's trust policy language (TPL) [herzberg00] provides a clearer separation between the authentication of subjects based on certificates and the assignment of authorisations to those subjects which have been successfully authenticated. With TPL, the credentials result in a client being assigned to a role which specifies what the client is permitted to do, where a role is a group of entities that can represent specific organisational units (e.g. employees, managers, auditors). The assignment of access rights to roles is outside the scope of TPL which focuses on extending RBAC mechanisms by mapping unknown users to well defined roles.

Commercial Policy Based Management Systems

Most of the systems described at length in this chapter exist primarily or exclusively within the realms of academia and research. Recent years have seen a steady increase in the number of commercial management systems which are based upon a policy based management approach. Most of these products are situated in the field of network / configuration management, with a particular concentration on QoS management and are based on the IETF policy model. Although the number of systems in existence is far too great to mention them all, some of the better known products are IBM's *Tivoli Suite*, Cisco's *CiscoAssure Policy Manager*, HP's *Openview PolicyXpert* and Nortel's *Enterprise Policy Manager*. IBM's *Workplace* is an example of a well known product which supports role based access control. Due to the closed nature of these products, and the limited technical information publicly available, it is difficult to say much about the underlying organisational models that they employ.

2.6 Human Interaction with policy based management

The field of policy based management is relatively young and thus there has been little in the way of specific research into human interaction with such systems. In the words of one researcher in the field: "little is known about policy-based interaction among people and between people and systems" [maglio05]. However, there are a few notable exceptions. Dijker [dijker96] and Barrett [barrett04, campbell05] have produced studies of how humans traditionally interact with policy systems for the purpose of identifying requirements for PBM systems and identified some major requirements.

Several of these requirements closely reflect the features of PBM that are generally accepted: policies should be specified in a human-friendly vocabulary; specifications of

policy rules should separate goals from methods; policies should embody the accumulated expertise of the organisation and should make that expertise available [barrett04]. In addition to these basic features, they identified several requirements which are not commonly viewed as being an essential part of PBM systems but are important in the context of deploying policy based systems within human organisations.

- The formation of policy should include input from the stakeholders in policy decisions as well as domain experts and the author of the policy. As Dijker notes, “there are too many potential consequences of establishing a policy to leave its development to a single individual” [dijker96]. This means that PBM systems need to be able to identify who these stakeholders are in any decision and provide a means for them to have input into the policy formation.
- It should be possible to specify policy rules with different levels of generality and specificity. Thus, it should be possible to create general rules which will serve as behavioural boundaries of the managed system. Within these boundaries, it should be possible to apply more specific rules to govern specific actions, operations or events.
- Policy rule sets should be adaptable over time to add more specific details to rules. Barrett refers to this as “*progressive grounding*”. This reflects the fact that organisations are complex entities and many of the policies are implicit and form part of the members’ ‘common sense’. So, rather than assuming that it is possible to completely translate an organisation’s operations into a set of policies at one time, it is necessary to allow policy rules to be built up over time, by adding exceptions and more specific rules as problems become apparent.

Another piece of work to note is an analysis of the interaction between humans and policy systems, described in [maglio05]. This research conducted ethnographic studies focusing on decision-making in IT organisations. It concluded that human interactions with policy systems tend to follow a pattern of negotiation and that this negotiation was particularly valuable in trying to debug or fix problems such as policy conflicts. It further concluded that policy systems need to be flexible, lightweight, and support reversible and incremental updates.

2.7 Summary

This chapter has analysed policy based management systems based upon their capacity to model human organisations, with particular focus on their support for modelling the

dynamic, decentralised, evolutionary aspects of organisations. The most important findings of this analysis can be summarised as the following.

- **Lack of Focus on Collective Goals and Functions.**

Although there has been some research into the automatic refinement of high level goals into concrete policies, there has been little examination of the problem of doing this on an organisational level. Large organisations are composed of multiple structural units with different goals, functions and relationships to each other. In order to capture an integrated model of the organisation within the policy based system, there is a need to capture the relationships between these various functions and the structures and policies within the system. Although some PBM systems support the specification of some types of organisational units, they provide little assistance in mapping the goals of these units to the goals of other units and the overall goals of the organisation.

- **Flexibility and Expressiveness Comes at the Expense of Complexity and Sacrifices Well-Defined Semantics**

Most PBM systems do not focus on incorporating integrated and accurate models of organisations. Although some of the formal models, such as CBAM, and policy frameworks, such as Ponder, do have the ability to incorporate complex organisational models, by specifying various grouping abstractions and relating them to each other, the flexibility is a consequence of the abstractions having user-defined semantics. The problem of mapping an organisation's structure to these constructs remains a complex task. There are many different ways in which an organisation's requirements can be mapped to the various abstractions in these models and most of the mappings will be extremely complex and brittle to change. In order to model a particular organisation in a maintainable way, all administrators will need to limit themselves to using the grouping abstractions and relationships in particular consistent ways or else the organisational model will quickly become divorced from any mapping of the real organisation and difficult to understand and maintain. Flexibility does not in itself solve the problem of modelling the organisation, it merely facilitates it. Furthermore, flexibility comes at a cost in terms of management complexity - the more flexible the specification language, the greater the difficulty in analysing and comprehending policy state changes.

- **Limited Support for Decentralised Modelling and Administration**

A major requirement that has been identified for an accurate organisational model is that it should support limited autonomy of groups and decentralisation of decision

making throughout the organisation. In order to prevent the need for an expensive and complex process of requirements engineering aiming to devise an exhaustive “Taylorist” recipe for the organisation, the modelling of the organisation and the specification of policies should be decentralised, allowing units within the organisation to progressively create their own model. Most PBM systems use a separate administrative hierarchy and where delegation is used, it is as a temporary transfer of access rights from one individual to another. This is not suitable for decentralised modelling of an organisation, where the structure itself is created autonomously and decentralisation is built into the structure as a permanent fixture. Furthermore, most PBM systems which support delegation require the authoring of meta-policies and/or delegation policies which specify precisely how a right can be delegated. Since the concept of delegation depends upon the assumption that the grantee is best placed to make decisions about the managed resource, the specification of precise parameters for delegation by the security administrator goes some way towards defeating the purpose. Also, as the number of policy rules in a system increases, the complexity of managing the policy system approaches the complexity of managing the managed resources. Therefore, by requiring that all delegations are explicitly authorised by policy rules, the net effect may be to increase the complexity of administration of the system.

A consequence of a model which supports radical decentralisation of decision making is that it must be able to do so in such a way as to minimise the complexity of implementing decisions on the system. Thus, human policy authors must be able to understand the potential consequences of updates to the policy system. In order to achieve this comprehensibility, a means of limiting the potential consequences of policy updates is required, by minimising the number of managed resources that might be potentially affected, and by minimising the number of policies that must be considered in any given context.

- **Large Number of Competing Policy Languages with Few Standards**

Another problem that can be noted from this survey is that there are a large number of different policy specification languages which are in use in different domains. Although there has been an emergence of standards (e.g. PCIM, XACML), these are generally targeted towards a particular policy application domain (e.g. Network management and access control) and there appears to be little impetus behind the proposed universal policy specification languages such as Ponder. Furthermore, even

where there are emerging standards, the nature of PBM is such that implementations tend to support different constraint semantics depending on the particular application: while the IETF may define a policy as if $\langle \text{condition}(s) \rangle$ then $\langle \text{action}(s) \rangle$, the set of conditions supported for any resource will be different.

Finally, the more expressive policy specification approaches are not required for all applications and thus it makes sense to adopt less expressive formal models for these applications in order to take advantage of their relative speed and ease of analysis. From the point of view of defining an interface for Organisation Computer Interaction (OCI) between the human organisation and the policy system, one problem that may very well be encountered is the deployment of multiple mutually incompatible policy specification languages and management frameworks within the same organisation.

Chapter 3. Community Based Policy Management – Theoretical Model

3.1 Introduction & Background

In this chapter, the basic elements of the Community Based Policy Management (CBPM) model and their semantics are described. The chapter starts by presenting a brief outline of the insights and inspiration that guided this approach and the evolution of the model that was adopted. A discursive summary of its essential features follows, introducing the terminology describing the conceptual space within which the work is situated. The third section of the chapter provides a formal description of the most important aspects of the CBPM model, including the decision-making algorithms employed. The full formal descriptions of the decision algorithm is presented in appendix 2, while its formal operational semantics are detailed in appendix 3.

The CBPM model is, most fundamentally, an attempt to provide a framework which can empower human organisations to structure and organise their interactions with information systems in such a way as to achieve the high-level goals of the organisation as effectively as possible. The key principles of this system are derived from the objectives and requirements identified in the preceding chapter. However, rather than being based on a purely theoretical analysis of the problem space, this approach emerged from the practical problems of modelling particular forms of organisation drawn from the real world which were not easily addressed by existing systems.

The initial work on this model was directed towards the problem of modelling Internet communities for the purpose of applying policy based management solutions [feeney04a, lewis04a]. Internet communities are differentiated from traditional bureaucratic organisations in a number of ways. They mostly depend upon volunteer labour. Their membership and goals can be fluid. They are widely distributed and often all community interaction is electronic. They tend towards flat hierarchies and often have wide membership involvement in decision making. They can be composed of multiple autonomous sub-communities with independent decision making mechanisms and different internal organisations who frequently split (or *fork*) or merge together. There are several well-known examples of such communities, such as the open-source software-development community that develops the Linux operating system. The success of the model has been proved by the widespread adoption of Linux by the IT

industry. Although this type of community has flourished in the past decade, there remain several widespread problems to their development. The vast majority of projects have remained very small and have had problems in expanding, lacking any means of regulating access to project resources and management responsibilities in a controlled way [krishnamurthy02]. They have often remained centred on a single individual maintainer, responsible for all decision making, who becomes a bottleneck to management, thus restricting the expansion of the project. Larger projects, which have succeeded in attracting greater numbers of members, have had problems in integrating the various sub-systems, which work in practice as autonomous projects, require a large amount of manual negotiation and ultimately often depend on a single maintainer to carry out management functions. For these reasons, a PBM approach has the potential to be particularly attractive to Internet communities.

However, the problems of modelling such communities within existing policy based management systems turns out to be problematic. The highly fluid nature of these communities creates a requirement for an organisational model that can be constructed entirely dynamically by participants. This structure evolves through a process of *specialisation, forking* and *incorporation* [feeney04a]. The *working group* is normally the fundamental grouping abstraction rather than the role. Responsibilities and duties are allocated to working groups and the individual's rights are dependant upon their memberships of working groups and their role within the working group. Working groups are also frequently self-managed, rather than falling under the authority of a central administrator. Many of the resources of the communities are not owned by the community. They are often owned by individual participants, or by participating sub-groups and donated to the community at the discretion of the owner. Therefore, a core assumption of most existing policy based management solutions – that the security administrator is authorised to author policy rules over all the managed resources – does not hold.

The initial goal behind the development of CBPM was to devise a means of modelling the way in which internet communities worked in practice. In analysing this problem, inspiration was found in two distinct areas.

- Neuro-biological models of organisation. Most particularly the neuro-biological processes whereby the brain operates through the training of zombie agents which operate as sub-conscious processes, thereby freeing up the consciousness

to deal with higher level processes [koch04]. In organisational terms, the consciousness can be considered to create specialised sub-units and delegate responsibility for specific functions to them, freeing up the higher decision making areas to concentrate on higher level goals.

- The Viable System Method (VSM). The VSM is itself a biologically inspired approach to organisational analysis [beer85]. It is based upon modelling an organisation's operations as a recursively defined hierarchy of viable systems. *"The principle applies at all levels, throughout all Viable Systems. Wherever you look, the Operation is composed of (smaller) Viable Systems, and is embedded in a (larger) Viable System. They are all nested one within the other, like Russian dolls. Recursion means that at all levels, the same principles of organisation "recur," and so it doesn't matter if you are diagnosing a two-person business or the global economy - exactly the same principles apply."* [walker06]

The CBPM approach relies upon modelling the organisation independently of its decision making structure. The organisation is conceived as an inverted pyramid, with each level representing a different unit, recursively defined, within the organisation and the entire organisation forming the base of the pyramid, as in figure 3.1.

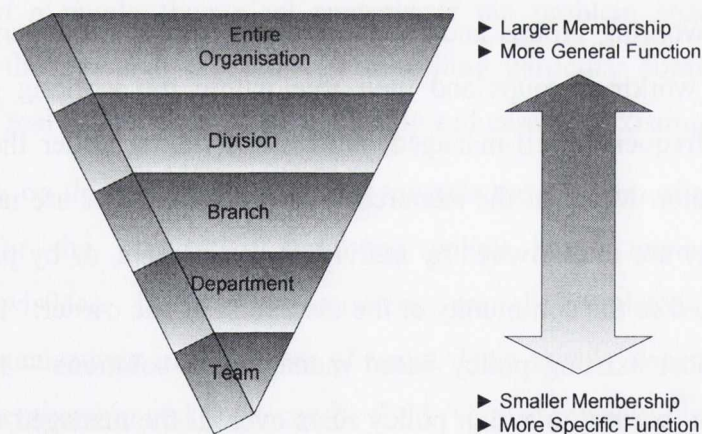


Figure 3.1 Organisation as a Hierarchy of Authority

Descending the pyramid, each level is more specific in function than the level above and has a subset of its membership. By *specific*, it is meant that the function of the entity is to carry out a specific function which has been identified as assisting in the attainment of a higher level function. This pyramid is termed the *hierarchy of authority*. Although the diagram above only shows a single path through the organisation, real

organisations will have multiple divisions, branches, departments, teams and will have different structures at different levels and different branches of their hierarchy.

This hierarchy of authority is used as the basis of an *authority map* of the organisation. Each organisational unit, whether it be a structural or functional unit in the organisation, is associated with a decision making method. An example, of this mapping is shown in figure 3.2, where the hierarchy of authority is mapped to the equivalent groups and individuals that make decisions on behalf of the organisational units. It should be noted that policies are considered to be “decisions about choices in the behaviour of a system” and are merely a means of specifying decisions.

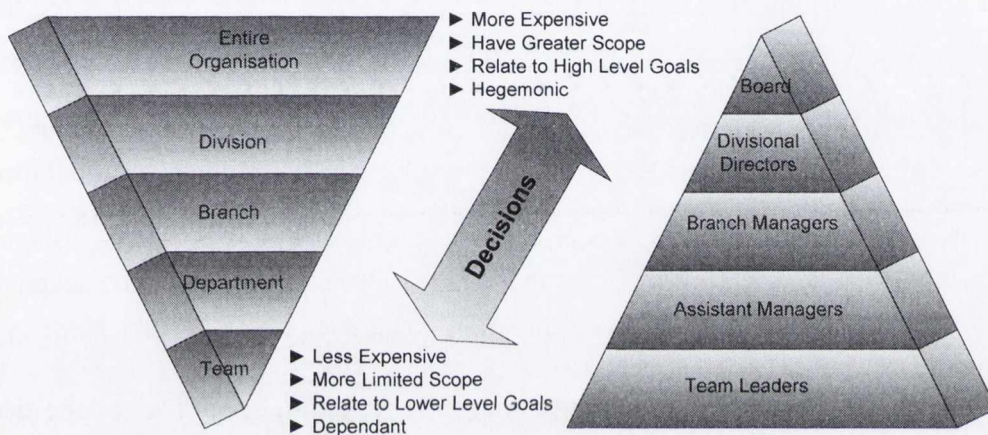


Figure 3.2 Decision Making Hierarchy

Each level in the pyramid on the right represents the decision making methods of the organisational units in the pyramid on the left. In moving up the hierarchy of authority, it can be observed that decisions:

- **Have Greater Scope.** Decisions of the organisation’s board may be binding on the entire membership of the organisation, and may relate to any of the organisation’s resources – they have global scope. Meanwhile, decisions made by a team leader normally only apply to the members of the team and to the resources that have been allocated to it – their scope is limited to their team.
- **Relate to Higher Level Goals.** Normally, the decisions of the organisation’s board will relate to the high level goals of the organisation - they will be defining strategies for the entire organisation to follow, rather than concerning themselves with the low level workings of the various teams that make up the organisation. Similarly, the team leaders’ decisions will reflect the goals of the team and may not be directly derivable from the organisation’s high level goals.

- **Are More Expensive.** Decisions at the higher levels of the organisation are considered more expensive in general than those further down the hierarchy of authority in two ways. Firstly, it is normally the case that the higher in the hierarchy one goes, the slower and more time-consuming the decisions are. For example, there may be several weeks between meetings of the board and these meetings may have crowded agendas, making their decisions relatively precious. Meanwhile, a team leader can often take an instantaneous decision without consulting anybody. Secondly, the cost of mistakes in decisions generally becomes more expensive as one ascends the hierarchy.
- **Can Overrule Decisions at Lower Levels.** Finally, it should be noted that the further up the hierarchy one goes, the greater the weight of the decisions. If the board makes a decision on behalf of the entire organisation, all of the units further down the hierarchy are bound to follow it. Individual divisional directors or team leaders require the explicit permission of the board before they can take decisions which conflict with the board's decisions. Decisions taken at the higher levels are *hegemonic* while those taken further down are *dependant*.

It should be noted that these characteristics generally hold, regardless of the decision making hierarchy that is associated with the organisation's hierarchy of authority. For example, if an organisation is run according to the principles of direct democracy, where every person who is affected by a decision has a vote in the decision, this can be modelled by configuring communities so that a decision within any unit of the organisation requires a referendum of all the members of that unit. Decisions made for the team will still be relatively inexpensive, since the team will have few enough members working together to be able to take a vote quickly, and the decisions will still have the same scope and relationship to the goals of the organisation as if the organisation was managed by a traditional hierarchical bureaucracy. Conversely, in any sizeable organisation, it can be expected that the time and effort involved in organising a referendum of the entire organisation will be considerable. However, once a decision has been taken by universal referendum, all of the members of the organisation, and all of the units within the organisation will be bound to follow it according to the general principles of this organisational model.

Due to the increasing expensiveness of decisions as one ascends the hierarchy, it is desirable to distribute decision making authority to the lowest possible point – it makes

much more sense for the details of a team’s operations to be decided by the team leader, or an assistant branch manager, than by the board. In order to fulfil this principle, the CBPM model is designed to inherently support progressive devolution of management responsibility. Each unit in the organisation aims to push decision making responsibility downwards wherever possible and units can be created whenever existing units manage to decompose their own structure and function into sub-functions. The process of pushing decision making down the hierarchy is based on *delegation*. However, this notion of delegation differs from that used by most PBM systems. Rather than the temporary transfer of rights from one individual to another, this delegation is a persistent transfer of authority between organisational units.

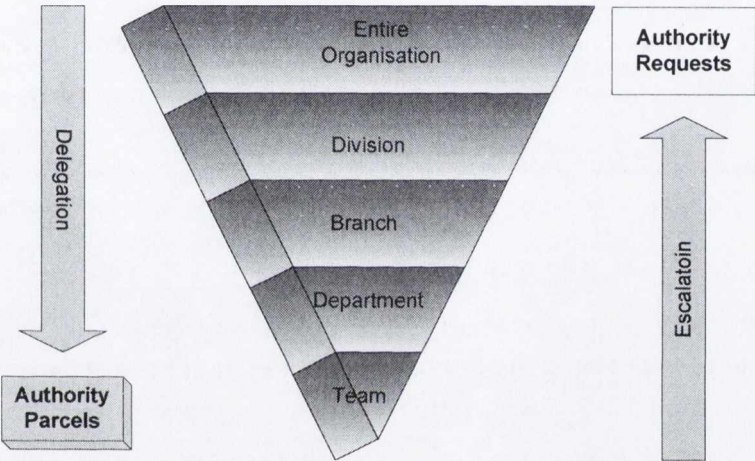


Figure 3.3 Delegation / Escalation Cycle

Figure 3.3 illustrates the basic dynamic of the CBPM model. Higher level units in the hierarchy of authority delegate *Parcels of Authority* to the lower level units. A parcel of authority refers to a set of resources whose use is constrained in various ways. Through delegation, the *grantee* unit gains authority to make decisions about the resources that have been granted, but those decisions are bound by the constraints specified in the parcel of authority. Receipt of an authority parcel amounts to being given responsibility to make decisions about the management of certain resources in order to fulfil the goals of the unit. This notion of the distribution of authority in constrained bundles to units which serve as a representation of the stakeholders of the potential decision reflects research into human interactions with policy based systems [barrett04].

In order to adhere to the principle of least privilege, the parcels of authority should be constructed in such a way so as to minimise the scope of authority to that which is strictly required. The dialectical goals of respecting this principle, while maximising the

amount of responsibility that is pushed downwards, provide one of the motors which drive the evolution of the system. This is a manifestation of the fundamental tension in all management systems between the certainty of centralised control and the convenience of decentralisation. The other factor that serves to shape the dynamic evolution of the system is *escalation*. Escalation deals with situations where an organisational unit needs to make decisions about a resource which it has not been delegated sufficient authority for, or where it is prevented from accessing the object due to a higher-level policy. It amounts to a lower unit asking a higher unit to delegate it authority, or to take a decision on its behalf with regards to the relevant resource.

3.2 Informal Description

For the purpose of this description, a PBM system is considered to be a system that receives *policy decision requests* and returns *policy decisions*. In the case of access control policy, it is assumed that the requests include information which allows the policy system to identify a *subject*, an *action* and a *target* and that the decisions contain information which allows the receiving process to either *permit* or *deny* the requested access to the resources.

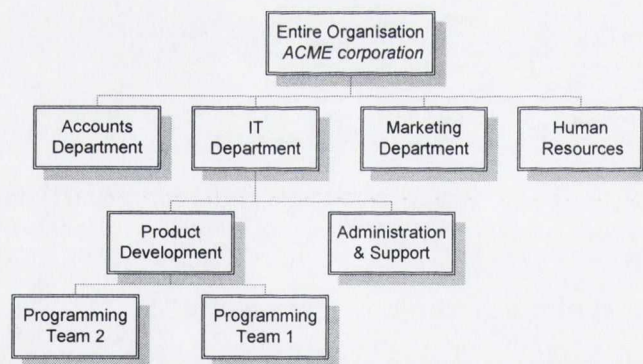


Figure 3.4 Segment of Simple Organisation Chart in terms of Organisational Units

The most basic element of the CBPM model, which differentiates it fundamentally from existing role-based PBM approaches, is the fact that the model of the organisation is primarily based on the organisational units that constitute it, such as divisions, departments, teams and working groups, and their relationship to the resources controlled by the organisation, rather than focusing on the positions of individuals within the organisation and their relationships to resources. In the CBPM approach, an organisation is considered to be composed of a recursive hierarchy of organisational units, with the entire organisation at the root of the tree and the most specific units within the organisation as the leaves. Each of these units has a goal with respect to its

role within the organisation, although this goal or function may not be fully known – it is progressively assembled.

3.2.1 The Community

A unit of an organisation is a composite, complex entity. Although it can be spoken of as having a high-level function with respect to the organisation to which it belongs, the successful performance of this function is dependant on the actions of the individuals who are members of it. However, the function of the unit cannot necessarily be derived from the functions of the individuals who make it up; the behaviour or properties of the unit are not a property of any single individual member, nor can they easily be predicted or deduced from the behaviour of the members. Models of organisations which are built up from the properties and behaviours of the individual actors within the system, such as role-based models, cannot accommodate such aspects of organisations without introducing further abstractions and are then still faced with the problem of how to relate these abstractions to the underlying role-abstraction. Although TBAC [georgiadis01] and OrBAC [elkalam03] models both address this issue, they are both essentially extensions of RBAC, extensions which include an extra variable – team or organisational membership – in the assignment of roles and do not attempt to model the properties of the organisational unit. The CBPM model is differentiated from such approaches as it considers the organisation entirely from the point of view of the composite organisational units that can be identified within it.

The term *community* is adopted to describe an organisational unit. This terminology was chosen firstly as it reflects the composite nature of units within human organisations, being made up of many human actors, and secondly due to the fact that the term comes as close as possible to being a general purpose grouping abstraction which can encompass any type of human organisation with a common high-level goal. Or, to put it in plain English, communities are groups of individual people and they are generally groups with some interest in common. The use of the term community in the CBPM model is similar to its use in the RM-ODP Enterprise Language [steen00]. In both instances a grouping and its relationships to other groupings is analysed as a unit. However, while the RM-ODP Enterprise Model uses the community concept to analyse a wide range of relationships and interactions between an organisation and other external organisations, and provides only low-level abstractions for describing its internal configuration, in CBPM, the concept is used primarily to analyse the

relationship between the sub-units within an organisation. Communities can be part of bigger communities and have smaller communities that are part of them – the organisational structure is recursively defined.

A community within an organisation is considered to have certain properties, independent of its membership:

1. It has a relationship to the resources of the organisation, autonomously of its membership. So, for example, the statement that *the marketing department has a budget of \$100 million* signifies a relationship between the organisational unit denoted by *marketing department* and the financial resources of the organisation. Modelling an organisation in terms of organisational units and their relationships to resources allows this information – which is not derivable from the properties of any individual actor in the system – to be captured by the model.
2. It is the subject of actions within the organisation, autonomously of its membership. Thus, for example, it might be said that *the IT Department hired 20 employees*. The action taken by the IT Department and the acquisition of the resources is not dependent on the properties of any of the members of the IT department and can be considered as an action taken by the unit.
3. In order to consider a unit within an organisation as being the subject of actions, it must also be considered as being capable of taking decisions. Thus, the *decision making methods* of each organisational unit are also explicitly modelled. So, for example, an organisation's operations might be defined by saying that the approval of the Chief Technical Officer is the decision making method of the IT department, or that a vote of the board of governors is the decision making method of the entire organisation.

A recursive model of the organisation in terms of its organisational units possesses an inherent hierarchy, which is termed a *hierarchy of authority*. This hierarchy is an effect of the compositional nature of organisations and the communities within them. On each level of the hierarchy, the communities are components of the communities above. Thus, in the example shown in figure 3.4, the four departments are components of the overall organisation and the two programming teams are components of the product development group. This hierarchy imposes several conditions on the communities. Firstly, their membership, in terms of the individual actors who make up the community, is a subset of the membership of the communities above them. So, for

example, for an individual to be a member of the IT department, she must also be a member of the organisation. Secondly, the authority over resources possessed by a community is subordinate to the authority exercised in the community above it. Exactly what is meant by this *subordinate authority* is fully explained in section 3.3, however, for now this can be simply considered to mean that decisions made by the higher communities will over-rule those decisions made by the lower units, when they conflict.

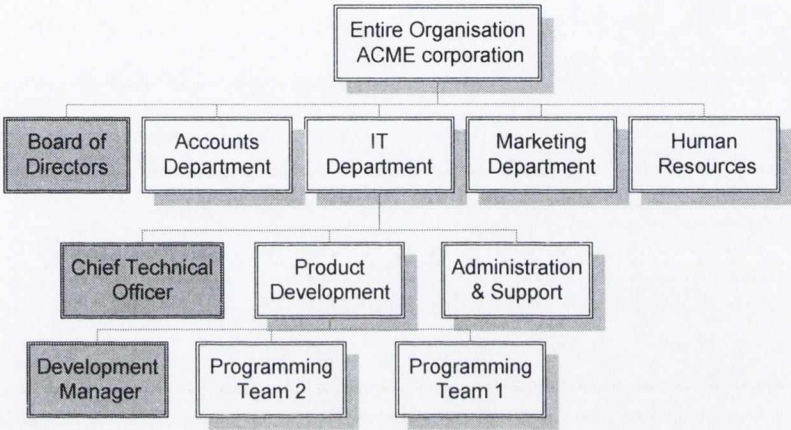


Figure 3.5 Hierarchy of Authority with Decision Making Units Highlighted

A particular feature of the community model is that communities which make decision on behalf of other communities are modelled as communities in their own right and are positioned in the hierarchy of authority beneath the community on whose behalf they exercise authority. In figure 3.5, some of the communities which make decisions on behalf of larger units have been added to the earlier example. Modelling decision making units as simply another community in the hierarchy allows the incorporation of an organisation’s specific decision making hierarchy into the hierarchy of communities without assuming any particular organisation-wide model of decision making. So, for example, while most of the units may have dedicated subordinate units which handle decision making, the two programming teams in figure 3.5 are presented without any such sub-units – they are self-managed and may even be democratically run, using voting or consensus in order to reach decisions.

The *board of directors* is viewed as a community which makes decisions on behalf of the entire organisation. The decision making methods at the level of the entire organisation are those that reflect the goals of the organisation as a whole. So, for example, the board of directors of a publicly listed company takes decisions on behalf of the entire company and these decisions are supposed to be aimed at maximising shareholder value. These decisions have an inherent precedence over decisions taken by

organisational units further down the organisation's hierarchy. The *IT department* cannot veto the board's decision to outsource IT services. Similarly, the *product development group* cannot ignore the *IT department's* decision to cease a product's development. This hierarchy of authority springs naturally from the fact that the lower communities are components of the higher communities and their goals are, by definition, sub-goals of the higher communities' goals.

Another aspect of this hierarchy that is important for the purposes of the organisational model is the relationship between the communities within the organisation and the resources controlled by the organisation (i.e. the resources to which the organisation will apply policies). The board of directors might decide, on behalf of the entire organisation and in their capacity as the decision makers for that community, that all accounts must be audited. The CTO does not have authority to over-ride this decision on behalf of the IT department with respect to that department's accounts. This is described by saying that authority for the financial resource is *rooted in the root community* of the organisational hierarchy, or to put it another way, the community representing the entire organisation possesses *ultimate authority* over the resource. In general, ultimate authority for all resources owned by the organisation will normally be possessed by the root community, representing the entire organisation. This reflects the fact that the organisation is the legal owner of the resources. Other communities within the organisation may possess *limited autonomy* with respect to these resources – their authority is bounded by decisions taken further up the community hierarchy. In some organisational models, however, it is possible for other communities in the hierarchy to possess ultimate authority over resources. This type of arrangement is inherently volatile – as the resource is managed autonomously from the overall organisation, despite the fact that the managing community is an integral part of the organisation. However, such arrangements exist in practice and can be modelled by the CBPM. They are referred to as *hybrid organisations*.

Sub-communities can possess authority to take decisions on a subset of the resources controlled by the entire organisation, an authority which they may be able to exercise autonomously of their parent in the community hierarchy. As was noted above, each community has authority over a subset of its parent's resources and this authority is bounded by the decisions of its parent. Thus, for example, the marketing department may have considerable latitude in deciding what to spend its budget allocation on, but

be constrained by any policy decisions that the board of directors may have taken committing it to spending money in a particular way. For example, the board of directors could take a decision that only approved suppliers will be used. Any decisions taken by the marketing department will be constrained by this decision. Similarly, the advertising team may have considerable decision making power when it comes to concepts, but be constrained to using printing contractors specified in a central contract between the company and a partner.

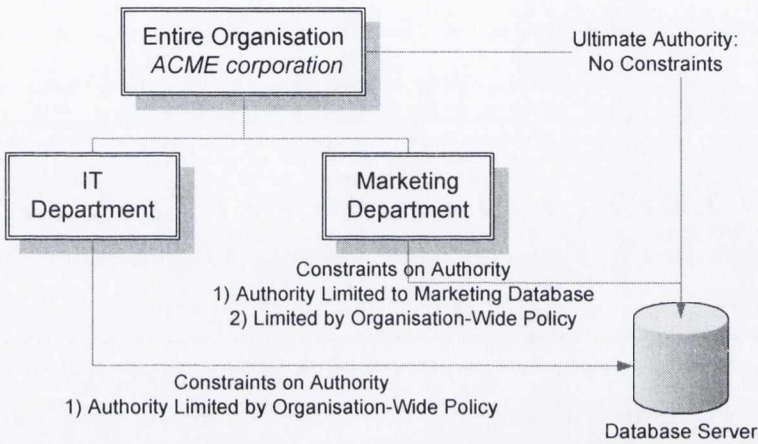


Figure 3.6 Authority Relationship of Organisational Units to Resource

In order to understand the concept of communities possessing authority over resources, it is useful to consider the relationship of each of the communities in a simple organisation model to a particular information resource, such as a database server, as depicted in Figure 3.6. Since the server is the property of the entire organisation, there are no limits on the authority of the root community over it. That is to say that, from an access control point of view, when the entire organisation is considered as an agent, all operations are permitted. If the decision making mechanism for the root community, whatever it may be, decides that the community wishes to access the resource in a particular way it must be permitted by the access control system, otherwise the organisation would not enjoy full control over its own resources. Other units of the organisation may also possess authority over the resource, but this authority is constrained by any policies about its use that may be defined by the root node of the organisational hierarchy. So, for example, while the IT department may possess authority to carry out a broad range of operations on the database server, creating and deleting databases without reference to the root node, they may be compelled to support database services for other departments and thus their authority over the resource is limited. They may not possess authority to delete the data stored in a marketing

database hosted on the server. Other departments may have even further limited authority over the resource. The marketing department may have authority to read and update data in a certain database, but have no authority to define storage schemas.

From the point of view of the CBPM model, the hierarchical tree of communities is considered to be a resource itself, also managed by the organisation. This resource represents the structure of the organisation and, similarly to any other managed resource, authority over this resource can be distributed throughout the organisation. Thus, organisational restructuring can be considered to be simply another form of resource management. The community which represents the entire organisation has authority to take any possible action on this resource – for example, by creating new communities beneath itself, or removing them. Authority over a subset of this resource can be possessed by any of the communities within the organisation. So, for example, the IT department may possess authority to create new communities as subsets of itself – or, to put it another way, it may be autonomous from the rest of the organisation with respect to its own internal organisation. In practice, communities will often possess limited autonomy with respect to their own structure, with some latitude to establish new internal communities and operational rules, while other aspects of their internal organisation and structure will be dictated by the overall organisation. Thus, the IT department may be compelled to have a community within itself that is responsible for quality control, while being free to decide how it will divide itself into project teams.

Thus, it is stated that a model of an organisation based on communities, as they have been described above, contains an inherent hierarchy of authority. At the apex of this hierarchy is the community representing the entire organisation, which has general authority over all decisions taken by all the individuals and units within the organisation. As we descend the hierarchy, the communities have authority in more specific domains. This hierarchy runs from the general to the specific. Rather than attempting to prove that this is a universally correct method of modelling organisations, it is adopted as a working definition of an organisation for the purpose of constructing the CBPM model.

Definition: *An **organisation** is a hierarchy of communities, a hierarchy that runs from the general to the specific, where decisions taken by communities with respect to the organisation's resources have a precedence that mirrors the hierarchy. This is called the organisation's **hierarchy of authority**.*

Community Federations

However, this model is insufficient to describe all forms of organisation. There are many examples of human organisation where the hierarchy of authority of the units with respect to resources does not hold – more general functional units do not always possess authority over more specific ones. For example, a *supply chain* has been defined as a “coordinated system of entities, activities, information and resources involved in moving a product or service from supplier to customer. The entities of a supply chain typically consist of manufacturers, service providers, distributors, and retail outlets. Supply chain activities transform raw materials and components into a finished product” [wikipedia06]. Supply chains are often formed by a collection of autonomous organisations, each with its own hierarchy of units and related hierarchy of authority. However, due to the highly integrated nature of many modern supply chains, it is frequently the case that certain resources are not exclusively controlled by any of these organisations. For example, a purchasing database might be shared between all of the partners in the chain. It may be the case that none of the participating organisations has authority to take decisions relating to this resource without consulting the other partners. Thus, in order to model the relationship between organisational units and resources for the purpose of applying the community model, it is the community that represents the entire supply-chain that has authority over this database resource. The individual organisations that constitute the chain are now communities positioned beneath the community that represents the entire supply chain.

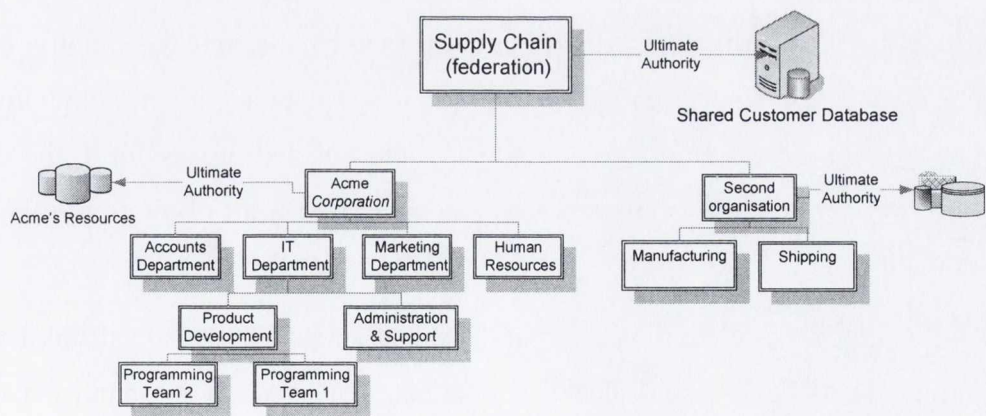


Figure 3.7 Simple Schematic of Organisational Units' Authority over Resources in a Federation

In this case, the hierarchies of authority reflect the community model of the organisation only with respect to those resources over which the overall supply chain possesses ultimate authority. With respect to all other resources, including the resources representing the structures of the constituent organisations, ultimate authority remains

with the communities representing the autonomous organisations that make up the supply chain. Happily, there is little problem in incorporating this type of organisational arrangement into the CBPM model. The hierarchy of authority still applies, but the hierarchy of authority for each resource is rooted in the community that possesses ultimate authority for that resource. Thus, the supply chain community does not possess any authority over the non-shared resources of the autonomous organisations that constitute it, since ultimate authority for them is possessed by the communities representing the individual organisations. It is only with respect to the shared resources that the root community is at the top of the hierarchy of authority. Thus the location of the root community in the hierarchy of authority is equal to the community which possesses ultimate authority for that resource.

The type of organisational model described above in relation to the supply chain example, is referred to as a *federation*. It is distinguished from the definition of the *organisation* above by the fact that the root community possesses ultimate authority for only a subset of the resources managed by the overall community hierarchy. By allowing ultimate authority for resources to be distributed among the communities in the hierarchy and by making the community with ultimate authority for a particular resource the root node in the hierarchy of authority for that resource, the CBPM model is generalised to incorporate a very wide range of organisational forms. The *organisation* described above – the traditional concept of the organisation – becomes a special case of an organisation, distinguished by the fact that ultimate authority for all resources controlled by the organisation is possessed by the root community. So, for example, considering a situation where the various autonomous organisations involved in the supply chain were to merge and adopt a traditional company form, the change would be simply modelled by moving ultimate authority for all of the resources to the root community.

In the CBPM model, a federation is generally modelled as a multi-lateral link between communities to an independent federal community. This federal community acts as a dependant organisation for managing resources shared between the organisations. Any organisation can pass a parcel of authority to the federal community through this link. The federal community can in turn pass parcels of authority to any of the communities linked to it. Federal Policies can be applied to any of the resources delegated to the federation, governing how the resources are utilised by the federation.

3.2.3 Modelling Resources

In order to model the authority over resources possessed by the communities within an organisation's hierarchy of authority, and in particular to allow communities to possess limited authority over resources, for example possessing authority to carry out a limited set of actions on a specific database table, a means of modelling resources so that they can be sub-divided into smaller units of authority, or parcels of authority, is required. This is achieved by creating a *resource authority model* to describe the resources controlled by the community, a model which defines a resource in terms of sets of actions and targets. This model defines the actions that are defined on the particular resource, how the resource can be sub-divided into distinct targets and how these targets and actions are related to each other by the *implies authority* relation.

From the CBPM point of view, a resource is any entity or object that the organisation wishes to apply policies to. For example, a database, a file system, a network router and network bandwidth are all resources. Each of these resources has a finite number of valid actions that may be taken upon it. So, for example, if the resource is a database server, the actions will include *create database*, *delete database*, *create table*, *insert data*, *select data* and so on. Many resources represent much too coarse a granularity for many of the common requirements of a PBM system. So, for example, if the policy system is managing access control to a database server, a means of controlling access to individual tables, individual rows of tables and even particular fields within a database is normally required. Many resources are organised hierarchically and share largely identical sets of possible actions that may be taken upon them. Therefore, CBPM resource models consist of hierarchical *target trees* and hierarchical *action trees*.

The hierarchical models of the resource represents a containment relationship that is defined from the point of view of an application's access control requirements rather than using the native hierarchical structure of the resource. So, for example, a target tree of file system objects could be constructed to mirror the directory structure of a file system. However, in many cases this hierarchy does not reflect the access control requirements of a particular application. For example, the traditional DAC access control model used with UNIX file systems do not generally mirror the hierarchical directory structure that is used to organise the file system. The owner of a directory may not own the files in the directory and although she has permission to read and write to

the directory, this does not imply that she also has permission to read and write to the files in the directory.

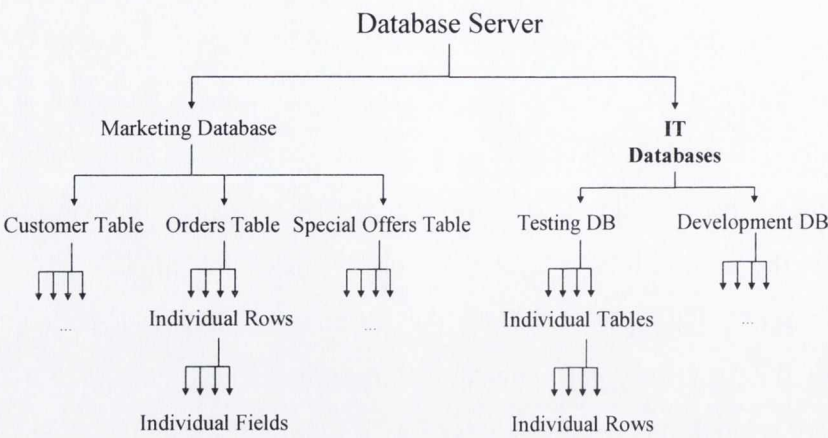


Figure 3.8 Example of Partial CBPM Resource Target Tree Representing a Database Server

Each node on the target tree represents a subset of the possible targets for which an access control decision may be requested on the resource. The edges of this tree relate these nodes according to the *implies authority* relation. The nodes on this tree do not necessarily correspond to any physical resource and can include arbitrary grouping nodes according to what is useful for the particular access control requirements of the system. So, for example, there is a node called *IT databases* on the target tree in figure 3.7 and all the IT databases are grouped under this node. Thus, a hierarchical target tree for each managed resource in the system is constructed, tailored to the particular access control requirements of the real-world system. This target tree may have little or no relation to the native hierarchy employed internally by the resource; the nodes are logical entities, which do not necessarily correspond with any physical resource and a particular node can appear in multiple places on the target tree.

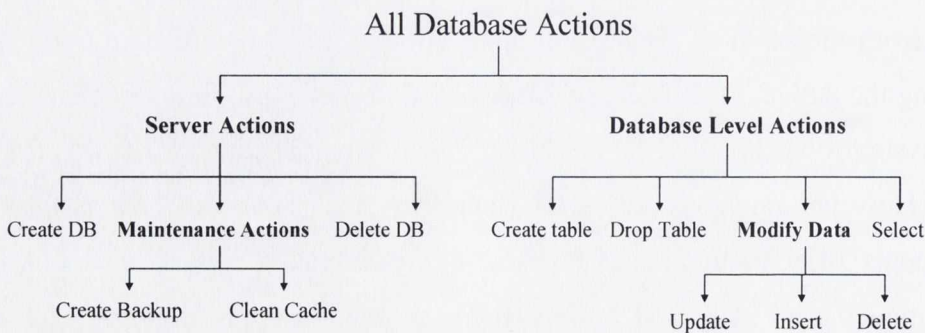


Figure 3.9 Example Resource Action Tree for Database Resource

As stated above, there is a finite set of actions supported by the resource for each node on the target tree. A hierarchical tree of these actions is created according to the *implies*

authority for relation. This tree is known as the resource's *action tree*. A community that possesses authority for a certain action on the tree also implicitly possesses authority for those actions beneath it in the action tree. Thus, for example, a database server resource could be defined to have the action tree pictured in figure 3.9.

This hierarchical model of resources has some similarity to Ponder's domain model (discussed in section 2.5.4 above) in that both are hierarchical representations which allow policy rules to be conveniently applied to subsets of managed resources. The main difference is that here the resource models are two-dimensional, incorporating actions as well as targets, and that this hierarchy is defined to represent a particular authority relationship, which places a semantic constraint upon the construction of the hierarchy that is absent in the domain model.

3.2.4 Community Membership

Having derived a model of the organisation in terms of its organisational units, described as communities, and their relationships to the resources of the organisation in terms of the *authority* that they exercise with respect to each of the managed resources, there remains the matter of relating this model to individual members of the organisation. For, while it may be the communities within the organisation which possess authority to take actions on resources, this authority is necessarily enacted by individuals since the communities are abstract, composite entities.

The communities within an organisation include individuals in a *membership* capacity. Each individual who belongs to the organisation is a member of the community that represents the entire organisation: the root community. As one descends the organisational tree, a subset of the individuals who are members of the organisation are members of each community directly beneath it in the community hierarchy. A *membership policy* can be defined for each community, a policy which can be evaluated to map a subset of the members of the parent community to the members of the child community. This mapping of individuals to communities in terms of membership also incorporates an inherent hierarchy. An individual can only be a member of a community if she is also a member of its parent. It is impossible for an individual to be a member of the IT department without also being a member of the overall organisation. Similarly, it is impossible for an individual to be a member of a project team within the IT department, without also being a member of the IT department. This rule is a

requirement of the CBPM model which springs from the decomposition of the organisation into recursive organisational units.

By successively defining membership policies for each community, as we descend the hierarchical tree of the organisation, all of the members of the organisation are mapped to the communities to which they belong. If the community membership rule returns a positive result – however that is interpreted - when a particular individual actor's identity is passed as a parameter to the rule, and if the membership rule for all of the community's parent communities also returns a positive result, until the root of the tree is reached, then the individual actor is said to be a member of the community.

3.2.5 Community Policy

One further element is required before the CBPM model is rich enough to be applied to the practical problem of applying PBM to resources. Thus far a model of the organisation in terms of its communities has been presented. Further, this incorporates a model of the authority that the organisational units possess over the resources of the organisation. A means of mapping the individuals that make up the organisation to these communities as members has been added. However, it cannot be assumed that each of the individual members can exercise the authority over the resources possessed by those communities that they are members of. The final addition to the community model, which addresses this issue, is the *community policy set*. Each community possesses a set of policies which dictate how the community's authority over resources can be acted upon by the community membership. The community policy set is considered to be the set of decisions made by the community. When a community policy permits an individual member to exercise the authority of the community to act upon a resource, it is stated that she *acts on behalf* of that community and she is considered to be acting *as an agent* of the community which has approved her actions through its policies.

Community policies can be seen as the decisions of the community governing how the individual members of the community can access the community's resources, decisions which are intended to allow a community to fulfil its function and goals. For the moment, nothing is defined about the nature of the policies in the model. It is sufficient to consider policies as rules that can be evaluated to generate a meaningful decision. The properties of policies in the CBPM model are described in section 3.3 below.

3.2.6 Steps Required to Model an Organisation

The outline of the CBPM model of the organisation has been presented above. The steps involved in creating this model can be summarised as follows:

1. A hierarchical tree of units within the organisation which are called *communities* is derived from the real world organisation(s).
2. A *resource authority* model is defined for each of the resources to which policy based management will be applied.
3. The *authority possessed* by each of these communities with respect to the resources managed by the entire organisation is specified, an authority that is subject to the community hierarchy defined in the first step.
4. Individual actors are mapped to communities, by defining *membership rules* for each community, again utilising the hierarchy of communities which constitutes the organisation to ensure that if an individual is a member of a community, he must also be a member of each of the units further up the hierarchy.
5. A *community policy set* is defined for each community which maps the authority possessed by the community to authority that the individual members of the community may exercise on behalf of the community.

3.2.7 Dynamic Aspects – Spawning, Delegation and Escalation

Rather than requiring the community hierarchy and the authority possessed by each of the communities in the model to be defined in full in a pre-operation phase, the CBPM model is designed to be dynamically constructed during operation through the dynamic formation of communities and the distribution of authority to them. The idea being that each community can create sub-communities of itself whenever it identifies a function that can be sub-divided from the overall community. The dynamic aspects of the CBPMS enable the progressive creation of a rich model of the organisation and a means of distributing authority throughout the organisation and resolving conflicts within the organisation through the natural hierarchy upon which the model is based.

The development of the community hierarchy is conceived as a dynamic process of communities *spawning* sub-communities. Rather than attempting to model the organisation in its entirety in advance, the structure evolves by a community spawning a *sub-community* directly beneath it in the hierarchy and delegating resource authorities to that sub-community. The ability for a community to dynamically spawn a new community directly beneath it in the community hierarchy and to delegate authority

over resources to this *sub-community* or *child-community* enables an organisation to flexibly adapt itself to the changing requirements of its environment.

The dynamic distribution of authority and the resolution of policy conflicts are both based on a simple concept. The communities higher up the hierarchy can *delegate authority* over the resources that they possess authority over to communities further down the organisational tree. When communities further down the tree find that they have insufficient authority to carry out their function, they can *escalate* decision requests to the communities further up the tree, who, in turn, can use their authority to perform the requested function or, alternatively, delegate the authority for the function to the community who escalated the decision. This is termed the *delegation-escalation* cycle and it provides the mechanism for the dynamic distribution of authority throughout the organisation. Rather than relying on a static model defined in a pre-operation phase, this approach allows the incorporation of the distribution of authority into the operation of the organisation and incorporates it into that operation as an ongoing process of negotiation, refinement and evolution, allowing the model to respond to changes in the organisation's requirements.

If a *resource authority* is delegated to a sub-community, the sub-community then possesses that resource authority until the delegation is revoked or the sub-community is destroyed by its parent community. The sub-community which possesses the resource authority can define policies for this resource authority or any resource authorities implied by it. According to the hierarchical model of the organisation, the decisions taken by the parent community have precedence over the decisions taken by the child community and thus the policies defined by the parent community have greater precedence than those defined by the child community when there is a conflict. A sub-community is a specialisation of its parent community with a subset of its authority. Any policies that are defined for delegated resource authorities in the sub-community have a lower precedence than those defined in the parent community according to the natural hierarchy of the organisation which flows from the general to the specific. In general, a community will spawn a new sub-community when it identifies a subset of its members who require different policies to be applied to their actions on a subset of the community's resources than the rest of the members of the community. This allows a community structure to develop organically. For example, if the number of members of the community grows to incorporate different actors who routinely carry out different

tasks, the community can spawn new sub-communities, delegate the resources necessary for the different tasks to each one and define membership rules to allocate the existing membership to the newly spawned communities. Thus, for example, if there is a community of software developers who are all working on the same task and, over time, the number of members of this community grows through expansion of the organisation and the members of this community find themselves working on two different development projects. According to the principle of least privilege, policies should be defined so that actors only have access to those resources which are required for them to perform their tasks on the projects on which they are working. Two new sub-communities are spawned to reflect the two project groups, membership rules for these sub-communities are defined to allocate the appropriate actors to the appropriate project groups. Then those resource authorities which have different policy requirements depending on the project group are delegated to the new sub-communities and community policies are defined within the scope of the sub-communities to enforce the desired rules. Thus, the policies defined in the parent, *developers*, community are those that apply to all developers and the policies that are defined in the sub-communities representing the project groups are defined to reflect the access requirements for members of those specific communities. Thus, the principle of least privilege is applied by creating two specialised sub-communities, delegating the appropriate resource authorities to those communities and dividing the membership between them as is required.

3.3 Formal Definition

In order to formalise the above description, the starting point is to define exactly what is meant by the terms that are used in this discussion. In this section, a formal specification for the CBPM model is described and the precise meaning of the various concepts used is defined. In particular what is meant by an *organisation* and a *federation* are defined in terms of the community construct. Then the decision making algorithm which implements the community based policy model is defined. Finally the operational semantics of the community management primitives which modify the state of the system are specified. All the formal definitions use based on first order logic.

3.3.1 Communities and their relationships

The first thing that must be defined about the CBPM model is that it is composed of a set of communities and a hierarchy defined on those communities. Let C_{Σ} be the set of all communities defined within the system: $C_{\Sigma} = \{C_1, C_2, \dots, C_n\}$.

Parent Function

The CBPM model defines a function *parent*, p , which maps a community to its parent community in the community hierarchy: $\text{parent} : C_{\Sigma} \rightarrow C_{\Sigma}$. A rule is defined specifying that, in the CBPM model, *all communities must have a parent community*:

$\forall c \in C_{\Sigma} : \exists y \in C_{\Sigma} : p(c) = y$. A root community is defined as a community which is its own parent since root communities represent distinct organic entities and are not spawned by any other community in the system. From the point of view of the conceptual model they are a new organism which spawned itself. The set of root communities is thus defined as the set of communities which are their own parents:

$$\text{Roots}_{\Sigma} \subseteq C_{\Sigma} = \{c \in C_{\Sigma} : p(c) = c\}$$

Since the domain and co-domain of the *parent* function are identical, and since the function is complete on the domain of C_{Σ} (from the parenthood rule above), the function can be composed: $\forall x \in C_{\Sigma} : \exists y \in C_{\Sigma} : p(p(x)) = y$. Thus the common notation to signify successive composition of a function can be used: $p(p(x)) \Leftrightarrow p^2(x)$, and this notation is used to state the rule that the parent function can be compositionally applied any number of times to any community and will always map the community to another community: $\forall c \in C_{\Sigma}, n \in \mathbb{N}^* : \exists y \in C_{\Sigma} : p^n(c) = y$. Thus the notation $p^n(c)$ is used to signify the n th parent of community c , where n can be any positive integer.

Has Authority Relation

The *parent* function, defined on the universe of communities, is used to construct a partial order on the set of communities according to the hierarchy of authority, on the basis that a community always has lesser authority than its parent. The binary relation *Has Greater Authority Than*, is defined, using the symbol $>$, on the set of Communities: $\forall x, y \in C_{\Sigma} \ x > y$ iff $(\exists n \in \mathbb{N}^* : p^n(y) = x) \wedge x \neq y$. Since a root community is defined above as being its own parent, the definition of this relation must exclude the case where x and y are the same community. According to the conceptual model, the only community which has equal authority with any given community is the community itself, thus the relation *Has Greater or Equal Authority Than*, can be defined as follows,

using the symbol $\geq$: $\forall x, y \in C_\Sigma \ x \geq y$ iff $x > y \vee x = y$. The relation $\geq$ produces a partial order on the set U . That is to say that $\forall a, b, c \in C_\Sigma$, the relation is: reflexive: $a \geq a$, anti-symmetric: $(a \geq b \wedge a \neq b) \Rightarrow \neg(b \geq a)$ and transitive: $(a \geq b \wedge b \geq c) \Rightarrow (a \geq c)$. This relation is used to define a rule designed to forbid cycles in the community hierarchy. This rule states that *a community cannot have greater authority than a second community which has greater authority than itself*: $\forall x, y \in C_\Sigma, x > y \Rightarrow \neg y > x$

Organisation

An organisation is defined, according to the CBPM model, as a set of communities springing from a single root community. An organisation is any subset of the set of defined communities that contains a community which is its own root and has greater or equal authority than all communities in the set. The definition of the organisation must also stipulate that the organisation includes all communities which the root community has greater or equal authority than. It should be noted that this definition of a community permits only a single root community in an organisation.

Definition : $O \subseteq C_\Sigma$ is an *Organisation*

iff $\exists x \in O : (p(x) = x) \wedge (\forall y \in O : x \geq y) \wedge \forall c \in C_\Sigma : x \geq c \Rightarrow c \in O$

Federations and Federal Communities

The CBPM model requires the definition of a second function on the set of defined communities, namely the *federations* function, denoted as f , which maps a community to the set of communities which it is federated to. Its domain is C_Σ , the set of defined communities and its co-domain is the power set of C_Σ .

$f : C_\Sigma \rightarrow P(C_\Sigma)$

where $f(c) = \{c_a, c_b, \dots, c_n\}$ is the set of communities that community c is federated to

A *federal community* is defined as a community to which any other community is federated and the set of federal communities can be defined as follows:

Definition : $c \in C_\Sigma$ is a *Federal Community*

iff $\exists x \in C_\Sigma : c \in f(x)$

$Federals_\Sigma \subseteq C_\Sigma = \{c \in C_\Sigma : \exists x \in C_\Sigma : f(x) = c\}$

A *federation*, which is the set of communities federated to a federal community and their sub-communities is defined as follows:

Definition : $F \subseteq C_\Sigma$ is a *Federation*

iff $\exists x \in F : (x \in Roots_\Sigma) \wedge (x \in Federals_\Sigma) \wedge$

$(\forall y \in F : (x \geq y) \vee (x \in f(y))) \wedge \forall c \in C_\Sigma : \exists z \in F : z \geq c \Rightarrow c \in F$

That is to say that a federation is composed of a federal root community and all communities that are federated to this community, and all communities which those communities have greater authority than. Federations do not have to be disjoint. They do not have to have a single root community within them. Since a federation is a composite entity, federations of federations can be formed, creating a situation where a community is a member of multiple federations.

Having defined the basic concepts that the CBPM uses in order to describe organisational forms, it should be noted that the state of the system in the CBPM model can be described as the triple (C_{Σ}, p, f) where C_{Σ} is the set of defined communities, p is the *parent* function defined on C_{Σ} and f is the *federations* function defined on C_{Σ} . The variable $\Omega = (C_{\Sigma}, p, f)$ is introduced to represent the state of the CBPM system.

3.3.2 Resource Specification

In addition to specifying the communities and their relationships, the concept of the hierarchical resource model that was introduced in the previous section must be formalised. Let R_{Σ} be the set of all resources managed by and modelled within the system. $R_{\Sigma} = \{R_1, R_2 \dots R_n\}$. Each resource is defined by two partially ordered sets $\forall R_i \in R_{\Sigma} : R_i = ((T_i, \geq), (A_i, \geq))$ where T_i is the set of all *targets* defined for resource i and A_i is the set of all distinct *actions* defined for it. The relation $\geq$ orders the set of targets according to the *implies authority for target* relation. Thus, the statement $t_1 \geq t_2$ signifies *authority to take a given action on target t_1 implies authority to take that action on target t_2* . Similarly, the relation $\geq$ orders the set of *actions* according to the *implies authority for action* relation. The statement $a_1 \geq a_2$ means that *authority to take action a_1 on a given target implies authority to take action a_2 on the same target*. For the purpose of this discussion the language can be simplified by stating the two implies authority for relations, for targets and for actions, as *implies authority for*. There is no ambiguity in using the same symbol or statement to specify “implies authority for action” and “implies authority for target”; the relation can only be used with two elements that are from the same set and thus its meaning is unambiguous and can be simplified as above.

Resource Authorities

The *resource authority* is a very important construct in the CBPM model. As mentioned above, a resource authority is a triple (r, t, a) where r is a resource model, t

and a are a target and an action node respectively defined for that resource model. The following rule defines a valid resource authority:

$$ra = (r, t, a) \in RA_U \Rightarrow r \in R_\Sigma, r = ((T, \geq), (A, \geq)), t \in T, a \in A$$

The binary relation *implies authority* is defined on the set of valid resource authorities:

Let $RA_U = \{ra_1, ra_2, \dots, ra_n\}$ be the set of all possible valid resource authorities

$$\forall a, b \in RA_U : a = (r_a, t_a, a_a), b = (r_b, t_b, a_b)$$

$$a \geq b \text{ iff } (r_a = r_b) \wedge (t_a \geq t_b) \wedge (a_a \geq a_b)$$

Resource authorities are the basic unit of authority in the community based model. If a community possesses a resource authority, it also possesses authority for any resource authorities that are implied by that resource authority in the resource model. This relation allows a comparison of the authority that is possessed with the authority needed to perform an action on a given target. As shall be seen, this is a key requirement of the community model, the construction of which depends upon organisational units dividing the authority that the community possesses and delegating subsets of it to sub-communities to allow them to concentrate on more specialised components of the overall function.

3.3.3 Community State Specification

From the discussion in section 3.2, recall that a community C is composed of the following elements:

1. A set of delegated authorities; these are resource authorities that have been delegated to the community. This set is denoted as DA
2. A set of community policies specifying the rules of the community, denoted P
3. A membership policy, denoted m

The state of a community is defined as the triple (DA, P, m) . This triple fully specifies the state of the community and there is no other state information required by the formal model. The structure and semantics of each element of this triple is described below.

Delegated Authorities

Each community c possesses a set of delegated authorities $DA_c = \{da_{c1}, da_{c2}, \dots, da_{cn}\}$. Each delegated authority is a pair (c, ra) where c is a community and ra is a valid resource authority: $da_{ci} = (a, b), a \in C_\Sigma, b \in RA_U$. As the policy decision algorithm requires that the system can identify the community that delegated the authority, each resource authority that the community has been delegated must include a reference to the community that

delegated the authority. In accordance with the community model, a community can be delegated authority for a resource by its parent community, or by a community that it is federated to, or else it may possess ultimate authority over a resource itself. When a community possesses ultimate authority over a resource, there is no community that can be said to have delegated that authority, so this situation is defined as being one in which a community has delegated authority over the resource to itself, which once again is a good match to the conceptual model. The rule for validating a delegated authority in a community c is now defined.

Let DA_{cU} be the set of all possible valid delegated authorities in a community c

$DA_{cU} = \{da_{c1}, da_{c2}, \dots, da_{cn}\}, da_{ci} = (c_{ci}, ra_{ci}), ra_{ci} = (r_{ci}, t_{ci}, a_{ci})$

$da_x \in DA_{cU}$ iff $(\exists da_x \in DA_x : ra_y \geq ra_x) \wedge ((c_x = c) \vee (c_x = p(c)) \vee (c_x \in f(c)))$

Meaning that a resource authority can only be delegated to a community by its parent, or by a federation that it belongs to, or by itself, and that the community that delegates the authority must itself possess a resource authority that implies authority for the delegated resource authority. To put it simply, it can be said that a community can only delegate authority that is less than what it possesses and that a community can only grant authority to a child or to a community that it is federated to.

Taking the range of the set of delegated authorities of a community, this set of resource authorities defines the *domain of authority* of the community. If the community has been delegated a resource authority, it can define policies that correspond to resource authorities implied by that authority and members of the community will be bound by these policies. However, when the resource authorities have been delegated by another community, any policies that the delegating community has defined which conflict with this community's policies will override them.

3.3.4 Community Policies, Policy Reasoning and Context

The community policy set, P , is a set of pairs, (ra, ps) where ra is a resource authority, and ps is a policy specification. No particular approach to policy specification is mandated. From the point of view of the CBPM model, a policy is an entity that can be evaluated in a given context and will return a meaningful result. Policy specifications are simply messages that are passed to a Policy Reasoning Service (PRS) for evaluation. Only the results of this evaluation are of interest to the algorithm upon which the CBPM Service (CBPMS) depends. From the point of view of the CBPMS, there are only two possible results of a policy evaluation. A policy decision either evaluates to *not*

applicable or to some other value which is passed back to the requestor and is interpreted by the system as being *applicable*. This allows the integration of a wide variety of approaches to policy specification and evaluation into the CBPMS. A policy is viewed as any entity that can be evaluated by a PRS in a given context to return a result and the form of the results that the policy evaluation should return is not specified, it is simply required that a subset of the results returned should be mapped to the *applicable* value of the CBPMS, with the rest of the return values being treated as *not applicable*. The CBPMS decision algorithm does not need to understand the semantics of the policy decisions returned by the PRS, it merely needs to be able to identify which policy decision should be interpreted as signifying that the policy specification is not applicable in the given context. The only other requirements for the form of the policy decision response that is returned by the PRS to the CBPMS is that the CBPMS must be able to detect policy results which are semantically equal from the point of view of the policy decision consumer and that it must be able to generate a “not applicable” message.

From the point of view of the PRS, the policy decision request that the CBPMS sends to it will generally require information about the context in which the policy is being evaluated. Once again, the community model does not concern itself with the contents or format of this contextual information, it is merely passed from the policy request to the PRS for evaluation. The specification of the functions provided by the PRS allow for the transmission of a context vector (a very common method of incorporating constraints into PBM systems [georgiadis01, oasis03, elkalam03]).

Let PS_U be the set of all valid policy specifications $= \{ps_1, ps_2, \dots, ps_n\}$ supported by the Policy Reasoning Service (PRS).

Let PD_U be the set of all valid and meaningful decisions of policy evaluations $= \{pd_1, pd_2, \dots, pd_n\}$ returned by the PRS

Let Con_U be the set of all valid and meaningful context vectors $= \{con_1, con_2, \dots, con_n\}$. It is assumed that the current state of the context is encoded in a context vector: $con_i = (con_{i1}, con_{i2}, \dots, con_{in})$ where each entry represents the state of some variable that is potentially meaningful to the Policy Reasoning Service. The policy specification language can be defined as the triple (PS_U, PD_U, Con_U)

PRS functions: evaluate, applicable, semantically_equal

The PRS maps policy specifications to policy decisions. The *evaluate* function, denoted e , is defined as a function that maps from the Cartesian product of the set of valid policy specifications and the set of valid contexts, to the set of valid policy decisions. It is also specified that this function is defined for all combinations of valid policy specifications and contexts:

evaluate

$$e : PS_U \times Con_U \rightarrow PD_U$$

$$\forall ps \in PS_U, \forall con \in Con_U : \exists pd \in PD_U : e(ps, con) = pd$$

The *applicable* function, denoted a , which maps from the set of valid policy decisions to a Boolean domain is also defined. This function is required by the CBPMS policy decision algorithm as policy decisions which are not applicable in the current context must be identified. In general, it can not be assumed that the policy specification language will enable the identification of policies that are applicable to a given context without evaluation. Therefore, this function is defined on the domain of valid policy decisions rather than that of valid policy specifications.

applicable

$$a : PD_U \rightarrow B, B = \{0,1\}$$

$$\forall pd \in PD_U : a(pd) = b : b \in B$$

A further binary relation on the set of valid policy decisions is defined. This is the *semantically equal* relation which relates policy decisions to other policy decisions that are semantically equal to them. The semantically equal relation is denoted by the symbol $\cong$ and the relation is defined as follows:

$$\cong : \subseteq PD_U \times PD_U$$

This relation must exist in order to allow the detection of policy decision conflicts. Other than requiring the ability to test policy decisions for this relation, and to test policy decisions for applicability, the CBPMS does not require any semantic understanding of policy decisions. Like policy specifications, policy decisions are treated as opaque messages which are passed back to the policy decision consumer for interpretation and their contents are not otherwise of concern to the algorithm.

Finally, although there is generally no need to be concerned with the contents of policy decisions from the point of view of the CBPMS algorithm, there is a requirement that it must be able to generate a message that signifies *not applicable* to the policy decision consumer in a language that is understood by that consumer. This signifies a situation

where the CBPMS was not able to generate a policy decision for the request for some reason (no applicable policies, invalid data, etc). Therefore a function *generate not applicable*, denoted n , is introduced and defined on the PRS. The notation pd_0 is used to indicate this *not applicable* policy decision which should be a constant for any policy specification language. $n: P(PD_U) \rightarrow PD_U, n(PD_U) = pd_0$. This rule states that any policy language must have a policy decision which is interpreted as *not applicable* by the PRS.

PRS Specification

Thus, the requirements for the policy reasoning service have been specified. A Policy Reasoning Service is defined as the septuple: $PRS = (PS_U, PD_U, Con_U, e, a, \cong, n)$. Or, to put it another way, the CBPMS supports any policy specification language and framework which fulfils the following requirements.

- All valid policy specifications can be evaluated to give a valid policy decision in a given context.
- All policy decisions can be checked to see if they are applicable in a given context and can be checked for semantic equality with other policy decisions.
- The PRS can generate a policy decision which will be interpreted by the policy decision consumer as *not applicable*.

Consider an example of a policy framework for access control, where the set of valid policy decisions is $\{Accept, Deny, None\}$ (*None* specifies that no policy decision is generated). If the a function and the semantic equality relation are defined as follows:

$$\begin{aligned} a: PD_U \rightarrow B: PD_U &= \{Accept, Deny, None\}, B = \{0,1\} \\ a(Accept) &= 1, a(Deny) = 1, a(None) = 0 \\ \forall a, b \in PD_U, a &\cong b \text{ iff } (a = b) \vee (a = Deny \wedge b = None) \vee (b = Deny \wedge a = None) \end{aligned}$$

This definition creates a situation with a default Deny rule, where no policy decision is interpreted as being semantically equivalent to a *Deny* policy decision. Note that this can be easily changed to a default Permit rule by changing the definition of the *semantic equality* relation as follows:

$$\forall a, b \in PD_U, a \cong b \text{ iff } (a = b) \vee (a = Accept \wedge b = None) \vee (b = Accept \wedge a = None)$$

Recall, from above, that the community policy set is a set of pairs (ra, ps) where ra is a valid resource authority and ps is a valid policy specification. To put it another way, all policy specifications are associated with a resource authority. In order to understand the

function of this resource authority, the concept and definition of a policy decision request will first be briefly examined.

From the point of view of its interface to clients the CBPMS performs a single, simple function. It receives *Policy Decision Requests* and it returns *Policy Decisions*. In order for the CBPMS algorithm to work, it is required that policy requests contain certain information. Thus, a *Policy Decision Request* is defined as the triple $pr = (ra, c, con)$ where ra is a valid resource authority, c is a valid community and con is a valid context vector: $ra \in RA_U, c \in C_\Sigma, con \in Con_U$. Each element of the triple is now explained in turn.

The resource authority element of the triple specifies what action on what target within which resource that access is sought for. This does not have to correspond with a specific action on the physical resource. Rather, each specific action and specific target on the physical resource is mapped to a resource authority – which represents a node on the resource authority tree for that particular resource. Section 3.5.1 presents one example of how a physical resource can be mapped to a resource authority model, chapter 5 provides several more and the question is examined in detail in appendix 4. For the moment, it can be assumed that the entity which produces policy decision requests accurately maps physical access attempts to resource authorities.

Since the CBPM model sees communities as being the subject of all actions, all requests must be associated with a community which will serve as the subject of that action. Or, to put it another way, the reference to a community that is included in the policy decision request, situates the request in the context of the organisation or federation for which the request is being made. It can be assumed that the entity which generates policy decision requests knows which community, organisation or federation that policy requests should be referred to for particular requests.

Context Vectors

Finally, it has been specified that a policy decision request contains a context vector, Con representing any state information that the PRS may require in order to evaluate policy. The only requirement for the context vector is that it should be a valid and meaningful description of the context from the point of view of the PRS. The CBPMS algorithm and community based model do not require any semantic information to be derivable from the context vector, it is a black box that is passed from the Policy Decision Request to the PRS. It is assumed that the entity which produces policy

decision requests will correctly generate context vectors which accurately reflect the context as understood by the PRS. In the following chapter, much greater detail is provided in describing how this policy vector can be formed in the CBPMS.

As an example of a possible policy decision request, consider the case where a user tries to remove a table from the *Testing Database*, using the resource model illustrated in figure 3.7 and figure 3.8. A policy decision request could be formulated as follows:

$pr = ((\text{Database Resource}, \text{Testing Database}, \text{Drop Table}), \text{IT Department}, \text{con})$

Having defined the policy decision requests, the significance of the resource authorities which are associated with the policy specifications in the community policy set can be explained. Each resource authority in the community policy set defines the scope of the policy specification. That is to say that whenever a policy decision request arrives to the CBPMS, the policy specification is only evaluated if the resource authority that is associated with it *implies authority* for the resource authority in the policy decision request. Thus, the resource authority in the community policy set acts as an event filter on the associated policy specification. It essentially defines a set of events – encoded as resource authorities – that will ‘fire’ the policy specification and cause it to be evaluated. Thus, within each community, all policies are given a simple scope by their associated resource authority which indicates which events they will be evaluated for.

Policy Scoping Rule

Let $p = (ra_p, ps)$ be an element in the community policy set for community C

Let $pr = (ra_{pr}, c, con)$ be a Policy Decision Request for this community

The policy p will be considered to be in scope for the request iff $ra_p \geq ra_{pr}$

Exactly how this policy scoping principle is utilised by the CBPMS is defined in the description of the policy decision algorithm below. To understand the concept of policy scoping, the resource authority model illustrated in figures 3.7 and 3.8 is used again.

There are two elements in the community policy set:

$p_1 = ((\text{Database Resource}, \text{Database Server}, \text{All Database Actions}), ps_1)$

$p_2 = ((\text{Database Resource}, \text{Special Offers Table}, \text{Modify Data}), ps_2)$

a Policy Decision Request $pr = (ra_{pr} = (r_{pr}, t_{pr}, a_{pr}), c_{pr}, con_{pr})$ arrives

According to the policy scoping rule, the first community policy p_1 , will be within scope whenever any policy decision request arrives where $r_{pr} = \text{DatabaseServer}$. Since the resource authority associated with the policy is defined as the root node of the resource authority tree, the policy specification has the scope of the entire resource tree. This policy will always be evaluated by this community whenever any type of policy

decision request arrives that is concerned with this resource. In the case of the second policy specification, the policy rule will only be evaluated for policy decision requests where $ra_p \geq ra_{pr} \leftrightarrow (r_{pr} = \text{Database Server}) \wedge (\text{Special Offers Table} \geq t_{pr}) \wedge (\text{Modify Data} \geq a_{pr})$,

So, for example, a policy decision request with the resource authority:

$$ra_{pr} = (\text{Database Resource}, \text{Special Offers Table}, \text{Update})$$

will be within the scope of policy p_2 , but a request with the resource authority:

$$ra_{pr} = (\text{Database Resource}, \text{Testing Databases}, \text{Drop Table})$$

will not, since the relation $\text{Special Offers Table} \geq \text{Testing Database}$ is not defined and thus the relation $ra_p \geq ra_{pr}$ is also not defined. Policy specifications may be Obligation Policies, as well as Authorisation Policies and this policy scoping mechanism also allows definition of trigger events for obligation policies.

The final requirement of the CBPM model is that a membership policy be defined for each community in the system. As outlined in section 3.2, the membership policy maps individual actors to communities that they are members of. However, it will have been noted that thus far the specification includes no notion of an actor. This is not accidental. The CBPM model does not contain an abstraction to model the identity of the actor who is making a request. It is assumed that such information is included in the context if it is needed. Thus, for example, there might be a variable in the context vector which represents an actor's identity, or alternatively, there could be a variable containing a certificate from an authentication server – whatever is required by the application and is supported by the PRS policy specification language. From the point of view of the CBPMS, a membership policy is simply a policy specification which is evaluated in the given context, the result of which is tested for applicability and in the case of the decision being *applicable*, the actor making the request is deemed to be a member of the community, otherwise the actor is not deemed to be a member. The details about exactly how the actor's identity, or security tokens or certificates are transmitted and how the membership policy specification is formulated do not concern the CBPMS. It should be noted that the membership policy differs from a community policy in that there is no resource authority associated with it; it is simply a policy specification. The evaluation of the membership policy is a part of the policy decision algorithm and thus the event with which this policy is associated is implicit.

Membership Policy: $m \in PS_U, pr \in PR_U, pr = (ra, c, con)$

The actor making the request, pr , is considered to be a member of c iff $a(e(m, con)) = 1$

3.3.4 Overview of the CBPM Model

All the elements of the CBPM model have now been defined. To briefly recap: at the highest level, the state of the CBPMS is defined by the triple and decomposed below. The various static elements and their relationships are defined in the UML diagram beneath. The state of the system can be summarised as:

$$\begin{aligned}
 CBPMS &= (\Omega, R_\Sigma, PRS), \Omega = (C_\Sigma, p, f), C_\Sigma = \{c_1, c_2, \dots, c_n\}, c_i = (DA_i, P_i, p_i), p: C_\Sigma \rightarrow C_\Sigma, f: C_\Sigma \rightarrow P(C_\Sigma) \\
 DA_i &= \{(c_a, ra_a), (c_b, ra_b), \dots\}, P_i = \{(ra_x, ps_x), (ra_y, ps_y), \dots\}, m_i \in PS_U, ra_i = (r_i, t_i, a_i) \\
 R_\Sigma &= \{r_1, r_2, \dots, r_n\}, r_i = ((T_i, \geq), (A_i, \geq)), T_i = \{t_{i1}, t_{i2}, \dots, t_{in}\}, A_i = \{a_{i1}, a_{i2}, \dots, a_{in}\}, PRS = (PS_U, PD_U, e, a, \cong, n)
 \end{aligned}$$

It should be noted that almost everything in the model is essentially composed of other entities. The exceptions are: the *target edge*, *target node*, *action edge* and *action node* elements. They are defined as part of the resource model and are primitive atomic entities as far as the CBPM model is concerned. The other exceptions are the Policy Specification, the Context and the Policy Decision as these are defined within the PRS. Figure 3.10 presents a complete UML diagram of the classes in the model.

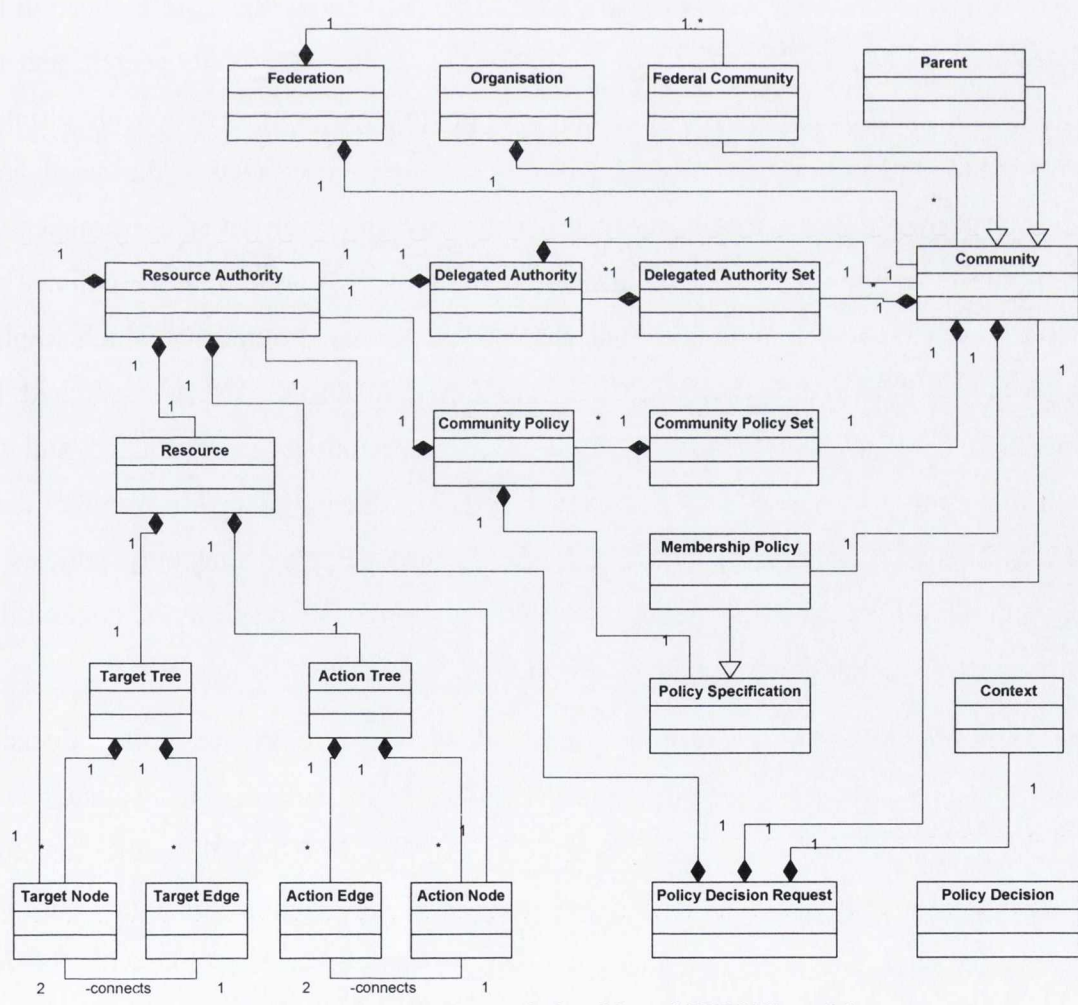


Figure 3.10 UML class relationships of CBPMS model

3.3.5 Policy Decision Algorithm

Having specified the required elements of the CBPM model, the policy decision algorithm applied by the CBPMS can be formally defined. There are three basic steps:

1. Locate the root community in the organisation with respect to the resource being accessed (i.e. the community that possesses ultimate authority for the resource).
2. Evaluate the membership rule and any policies that are in scope for the request.
3. If no policies are applicable, search any children which have been delegated authority within scope for the request resource authority (return to step 2 recursively for each child).

The hierarchical model of the organisation has been constructed in such a way that there is an inherent hierarchy of authority. When this is translated into a policy evaluation algorithm, it means that the first task is to find the community which exercises ultimate authority over the resource to which the policy applies. In a centralised organisation, ultimate authority for all resources is rooted in the root community. For federations and hybrid organisations, this may not be the case. The Community that is referenced in the policy decision request is used as the starting point for the community search, and the community hierarchy is ascended, step by step, until the community with ultimate authority for the resource is reached. Any policies that are defined at this level have precedence over policies further down the hierarchy, thus if any of these policies are applicable to the given context, their results are returned. If none of these policies are applicable, any child communities that have been delegated authority which implies authority for the resource authority in the policy decision request are searched for applicable policies. The policies in these child communities are evaluated and the algorithm continues, descending the tree until an applicable policy is discovered or there are no remaining policies to evaluate. It should also be noted that only policies in communities where the membership rule returns an applicable result in the context that is associated with the policy decision request must be evaluated.

The hierarchy of authority between communities is innate in this policy decision algorithm since policies defined further down the hierarchy will never be evaluated if an applicable policy is found in one of their parent communities. This helps policy searches by eliminating many policies that are in scope. Thus, the potential for policy conflicts between policies defined in communities that are directly comparable with the *has greater authority than* relation is eliminated. There is still the potential for policy

conflicts when conflicting policies are defined within the same community policy set, or when communities which are not related in terms of authority define conflicting policies. These cases are dealt with in the section on policy conflicts below.

The *Get Policy Decision* function, denoted *decision*, is now defined. This is the basic function that the CBPMS offers to the outside world and it is the purpose of the system to provide this service. This function takes policy decision requests and returns policy decisions to a policy decision consumer. The language used for specifying the policy decisions is defined as part of the specification of the Policy Reasoning Service (PRS) and it is assumed that the policy decision consumer understands this language (which could be achieved through the use of translation proxies). As part of its definition, this function produces a valid policy decision for all valid policy decision requests. The policy request is denoted as: $pr \in PR_U, pr = (ra_{pr}, c_{pr}, con)$. For reasons of space, the algorithm followed by the *decision(pr)* function is merely summarised here. In order to specify the algorithm in a structured way, a set of ‘helper’ functions is first defined, as shown in table 3.1 (whose definitions are included in Appendix 2).

Function	Description
<i>rootcom(c, ra)</i>	Returns the community in the community hierarchy to which <i>c</i> belongs that possesses ultimate authority for <i>ra</i>
<i>member(c, pr)</i>	Returns true if the policy request <i>pr</i> passes the membership rule of community <i>c</i>
<i>rootmember(c, pr)</i>	Returns true if the policy request <i>pr</i> passes the membership rule of all the parent communities of community <i>c</i>
<i>children(c)</i>	Returns the set of all the children of community <i>c</i>
<i>delegated(c, ra)</i>	Returns the set of all the child communities of community <i>c</i> to which resource authorities with greater scope than <i>ra</i> have been delegated
<i>scoped(c, ra)</i>	Returns the set of all policies defined for community <i>c</i> that are within the authority scope of resource authority <i>ra</i>
<i>combine(PD₁, PD₂,...)</i>	Combines a set of policy decisions to return a single policy decision (according to the conflict resolution strategy (see section 3.4 below))

Table 3.1 Functions Defined as Part of the CBPMS *decision* algorithm

Having defined all of the above functions, the algorithm for the decision function can now be succinctly defined. This has 3 simple steps. First the root community for the resource authority of the policy decision request is located, then the actor making the request is validated as a member of that community by invoking the *rootmember* function. Finally the policy in the root community is evaluated – which launches the recursive *evalpol* function below, which will search the community hierarchy beneath the root community until it finds an applicable decision, or until there are no more communities to search which the actor is a member of or which have been delegated

authority with greater scope than that of the request. Finally, all the decisions returned are combined with the *combine* function to return a single policy decision:

decision : $PR_U \rightarrow PD_U$, $decision(pr) = pd$, $pr = (ra_{pr}, c_{pr}, con)$, $c_{root} = rootcom(ra_{pr}, c_{pr})$,

$(rootmember(c_{root}, pr) = 0) \Rightarrow pd = n()$

$(rootmember(c_{root}, pr) = 1) \Rightarrow pd = evalpol(c_{root}, ra_{pr})$

evalpol : $C_\Sigma \times PR_U \rightarrow PD_U$, $evalpol(c, pr) = pd$, $pd \in PD_U$, $c = (DA, P_c, m)$, $P_c = \{p_1, p_2, \dots, p_n\}$, $p_i = (ra_i, ps_i)$,

$pr = (ra_{pr}, c_{pr}, con)$, $Q = scoped(c, ra_{pr})$, $PD_{applicable} \subseteq PD_U$, $\forall p \in Q : e(p) \in Z$ iff $a(e(p, con)) = 1$, $PD_{applicable} = \emptyset \Rightarrow$

$(G = delegated(c, ra_{pr}))$,

$PD_{applicable} \subseteq PD_U$, $\forall pd \in PD_U : pd \in PD_{applicable}$ iff $\exists g \in G : (member(g, ra_{pr}) = 1) \wedge ((pd = evalpol(g, ra_{pr})))$

$pd = combine(PD_{applicable})$

3.4 Conflict Resolution

This section deals only with policy conflicts that are detected at runtime and is not concerned with the detection of conflicts between policies at ‘definition-time’ – i.e. when they are first entered into the system. Firstly, it is worthwhile to briefly discuss the different classes of runtime policy conflict that may arise in a community based system. This classification is useful since the relationship between the communities that were the source of the conflicting policy decisions contains semantic information that can be leveraged in order to identify the source of the conflict. There are three basic classes of conflict:

1. A policy of a parent community conflicts with a policy of one of its descendants

According to the CBPM conceptual model of the organisation, such conflicts can always be resolved in favour of the policy from the parent community. Or, to put it another way, given a certain context, if there is a semantic difference in the policy decision generated by an applicable policy of a parent community and the policy decision generated by an applicable policy of a child community, the conflict should be resolved in favour of the former. This type of conflict is handled automatically by the CBPMS policy decision algorithm above. The search for applicable policies stops descending the community tree when it finds an applicable policy in a community, this decision result will be returned immediately and any conflicting policies that may be defined in child communities will not even be evaluated.

2. A policy of a community conflicts with a policy defined in the same community

If there are two conflicting policy decisions generated by policies in the same community and the policy specification language provides no information that identifies an algorithm to apply to produce a single policy decision from the conflicting policies,

then there is insufficient information to automate the conflict resolution. According to the CBPM conceptual model, this is viewed as an internal problem in the community which requires analysis of the community's policy set and the re-authoring of one of the conflicting policies so that the community's policy is clearly specified. Since the conflicting policies represent the decisions of the same community, and are thus made with equal authority, it is not possible to automatically resolve such a conflict and all that can be done is to bring the administrators' attention to the existence of the problem.

A second strategy for such conflict resolution, and one that does not require administrators to re-engineer community policy sets when conflict is found between policies within the same community, is to simply adopt a policy specification approach which includes metadata which the CBPMS can use to map policy conflicts to automatic conflict resolution algorithms. For example, the XACML specification [oasis03] allows policies to be specified in policy sets which support a standard and extensible set of conflict resolution algorithms, known as *Policy Combination Algorithms*. The basic algorithms defined in the standard are: *DenyOverrides*, *PermitOverrides* and *FirstApplicable*. By including such meta-data within policy specifications, the community policy sets can be specified in different ways depending on the requirements of the community. If the *DenyOverrides* policy combination algorithm is used, policies are specified in such a way to make sure that policy decisions which have "Deny" semantics according to the XACML specification should always override conflicting policy decisions within a single community. By using this algorithm, access can be denied in specified contexts to specific actions on specific targets within a resource with the guarantee that any other policy which returns a different decision will be ignored in favour of the Deny result. *PermitOverrides* works in the same way, except that the policy decisions with "Permit" semantics will override all others. This allows the specification of policies in such a way that those actions which are explicitly permitted will always be permitted regardless of any applicable policies which would return a different result. Finally, the *FirstApplicable* algorithm depends on the community policies being evaluated in order of priority. This requires that the set of community policies is stored along with information which allows the set to be ordered as a priority list. Each policy will be evaluated in order and the algorithm will return the first policy to return an applicable decision.

3. A policy of a community conflicts with a policy of a community that is not directly comparable to it in terms of the hierarchy of authority.

This situation arises when a parent community grants overlapping resource authorities to two of its children. If the children communities, or any of the communities beneath them in the community hierarchy, create policy rules which return conflicting policy decisions given the same context with a resource authority which is implied by the root node of the subset of the resource authority tree formed by the overlap of the granted resource authorities, then there is a potential for policy conflicts. The conflict occurs when an actor is simultaneously a member of two communities which contain applicable policies for a request and the communities are not directly comparable. It should be noted that, according to the decision algorithm above, such policy conflicts can only occur in a situation where an individual actor is a member of all of the communities which contain the conflicting policies simultaneously.

In this case, any combination of the two basic strategies outlined above for dealing with conflicting policy decisions can again be adopted. However, conceptually, the policy conflict is not viewed as stemming from a problem in the specification of the conflicting policies themselves. It is interpreted as a problem of resource competition between organisational units. Since the two communities are not directly related, it cannot be assumed that there is any overlap in terms of their function and goals. Both communities may be components of completely different divisions playing important roles in attaining the high level goals of the organisation. In such a situation the community tree must be ascended to the lowest community which is higher than both of the conflicting communities in the tree, or to put it another way, the lowest common ancestor of the conflicting communities is identified as the community within which the problem must be resolved. It is only at this higher level that the goal of the community will be general enough to allow the evaluation of the claims of the two sub-communities that are competing to supply the policy decision in response to a request.

Thus, if a conflict resolution strategy is adopted which depends upon policy re-engineering rather than automatic meta-data based resolution, the CBPMS can support this by generating a *conflictNotification* event which automatically brings the existence of the conflict to the attention of the policy administrators of community that is identified as the source of the conflicting policies, the event is routed to the administrators of the lowest common ancestor of the conflicting communities.

In order to automate the resolution of policy conflicts in situations of resource competition, metadata must be added to the system. However, rather than being metadata associated with policies, this metadata is a property of the community that is the lowest common ancestor of the conflicting communities. If the CBPMS has access to information which associates the set of children of the community with an algorithm for resolving conflicts between them, the communities can be specified to be ordered in terms of priority by using the *FirstApplicable* algorithm, or any other algorithm that can be applied to the resolution of policy conflicts.

3.5 CBPMS Administration

The CBPMS policy decision algorithm and conflict detection have now been defined to show how policy decisions are generated in response to requests, given a particular state of the CBPMS. However, thus far a static model has been assumed, a model which remains in a constant state throughout these algorithms. The CBPMS specification, however, does not only define a policy decision algorithm, it also defines rules for how the state of the CBPMS may change in order to enforce the validity of the community based model of the organisation while allowing the community structure to evolve in a controlled way over time. This is achieved by defining a set of functions which change the state of the system in a precisely specified manner and defining a rule which specifies that the state of the system can only change in response to these functions. These functions are termed the *CBPMS management primitives*, they provide an interface which allows the organisations using the CBPMS to dynamically create the model of their organisations and distribute authority throughout them in such a way as to ensure that the system remains in a state that is well defined and consistent with the requirements of the CBPM model.

3.5.1 CBPMS Management Primitives

The CBPMS management primitives define, in a precise way, how the state of the CBPMS may change. Their operational semantics are defined in appendix 3. Figure 3.11 presents a graphical view of these primitives, while table 3.2 lists the primitives along with their arguments.

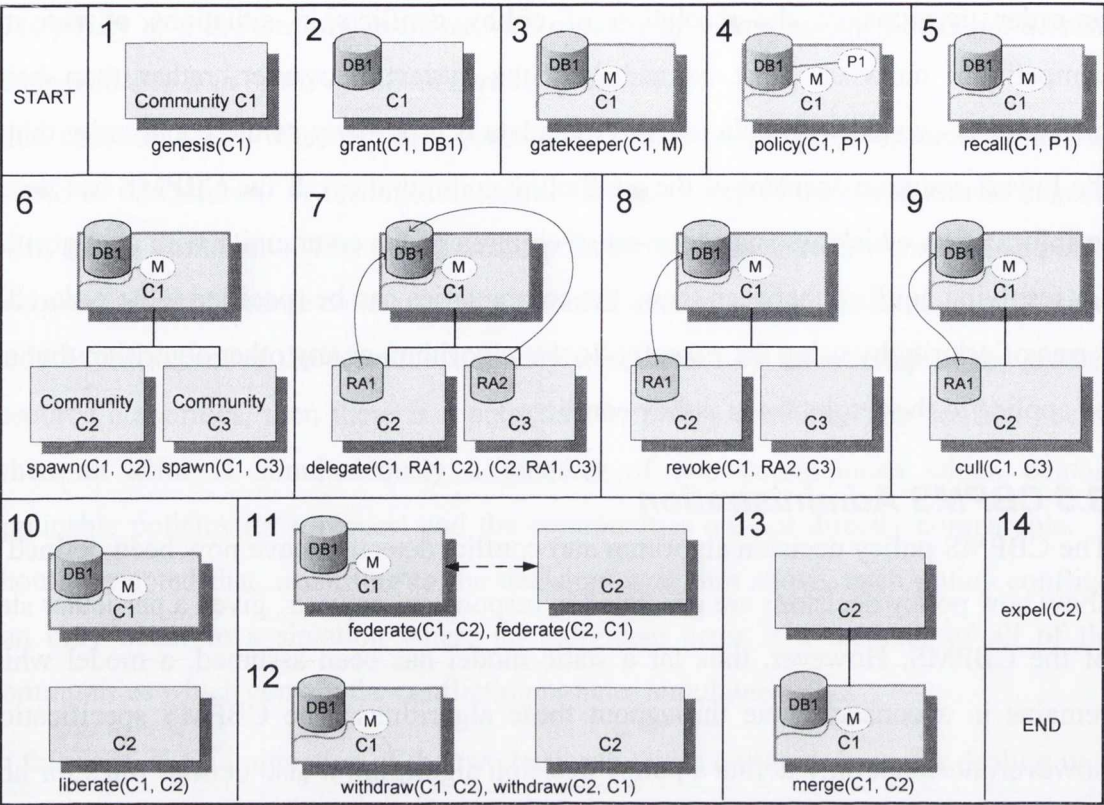


Figure 3.11 CBPMS Management Primitives Invoked in Sequence

The primitives can be divided into two groups. The first group is related to events that occur outside of a community context, namely the creation and deletion of organisations within the system and the assignment of ultimate ownership of resources to communities. These situations are related to real world events outside of the domain of the community based model – an organisation signing up to use the service, or an organisation introducing community based modelling to a new area of their managed resources are the types of events that trigger such actions and since they refer to relationships and entities that are unknown to the system, they can not be modelled as an action of an existing community. In these cases the CBPMS manager is viewed as being the subject of the action or primitive. There are three such CBPMS *system primitives* and there are eleven CBPMS *community oriented primitives*. This means that a community is seen as being the subject of each operation. The first argument to each of these primitives is the community that is the subject. So, for example, the operation `cull(c1,c2)` is equivalent to saying *c1 culls c2*.

Primitive	Parameters / types	Effect
<i>genesis</i>	<i>None</i>	<i>Creates a new root community</i>
<i>expel</i>	Community ID <i>c</i>	<i>Remove community c from system</i>
<i>grant</i>	Community ID <i>c</i> Resource Authority <i>ra</i>	<i>Grants ultimate authority over ra to c</i>

<i>spawn</i>	Community ID <i>c</i>	<i>Community c creates a new sub-community</i>
<i>cull</i>	Community ID <i>c1</i> Community ID <i>c2</i>	<i>Community c1 removes sub-community c2</i>
<i>delegate</i>	Community ID <i>c1</i> Resource Authority <i>ra</i> Community ID <i>c2</i>	<i>Community c1 delegates authority over ra to sub-community c2</i>
<i>revoke</i>	Community ID <i>c1</i> Resource Authority <i>ra</i> Community ID <i>c2</i>	<i>Community c1 revokes authority over ra from sub-community c2</i>
<i>policy</i>	Community ID <i>c</i> Policy <i>p</i>	<i>Community c defines policy p which will apply to members of c</i>
<i>recall</i>	Community ID <i>c</i> , Policy <i>p</i>	<i>Community c removes policy p which no longer applies to members of c</i>
<i>gatekeeper</i>	Community ID <i>c</i> Policy Specification <i>ps</i>	<i>Community c defines its membership rule (if the policy specification ps is evaluated and found to be applicable in a given context, the requesting context is considered to represent a member of c)</i>
<i>federate</i>	Community ID <i>c1</i> Community ID <i>c2</i>	<i>Community c1 joins the federal community c2</i>
<i>withdraw</i>	Community ID <i>c1</i> Community ID <i>c2</i>	<i>Community c1 leaves the federal community c2</i>
<i>liberate</i>	Community ID <i>c1</i> Community ID <i>c2</i>	<i>Community c1 liberates sub-community c2 which becomes an autonomous root community</i>
<i>merge</i>	Community ID <i>c1</i> Community ID <i>c2</i>	<i>Community c1 liberates sub-community c2 which becomes an autonomous root community</i>

Table 3.2 CBPMS Management Primitives

3.5.2 Community Hierarchy Resource Model

One of the main motivations behind the development of the CBPMS was to provide a PBM system which would support community self management. This would allow the members of particular communities to manage the model themselves – through the invocation of the primitives above, for example. Although the various primitives have been introduced, providing the ability to dynamically create an organisation’s structure and distribute authority across it in a controlled way that allows precisely scoped policy management, how they might be integrated into a practical approach to self-management has not been discussed. How to ensure that the primitives are invoked only when the community’s decision making mechanisms dictate that they should? The CBPMS answers this question by simply considering the community hierarchy itself to be a hierarchical resource, authority for which can be distributed throughout the community hierarchy like any other resource. There is no separate administration model. This requires the construction of an action tree and a target tree to represent the community hierarchy. Figure 3.12 shows a simple target and action tree which represent

a simple community structure, with an action tree consisting of the primitives (with some grouping nodes added for convenience of delegation) and a target tree which simply mirrors the community hierarchy. Modelling the community hierarchy in such a way allows the community to distribute authority for managing the community structures throughout the organisation, by delegating nodes on this resource authority tree to the desired communities – allowing them to self-manage, or manage other communities. This approach also means that administration of the CBPMS is managed by the CBPMS itself – meaning that the system is self-managing in two senses.

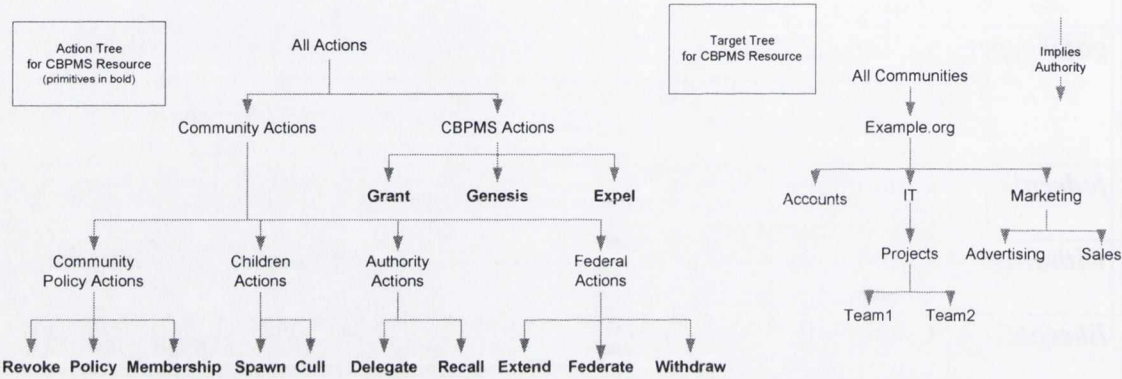


Figure 3.12 CBPMS Hierarchical Resource Model Representing the Community Hierarchy Itself

It should be noted that there are many possible ways in which the community hierarchy resource model could be designed. Action nodes do not need to directly map to CBPMS primitives, nor does the target tree have to resemble the community hierarchy – there are some situations where a flat target tree structure is preferable. The questions surrounding the construction of community resource models and how they relate to self-management are addressed in appendix 4.

3.6 Conclusion

This chapter has introduced the theoretical model underlying the Community Based Policy Management System through an informal and formal description of the entities and concepts involved. The CBPMS is designed to facilitate the modelling of dynamic organisations, with decentralised decision making structures. The organisational model, defined through CBPMS primitive invocations, is used by the CBPMS’s decision algorithm to assist it in resolving conflicts, based upon the organisation’s hierarchy of authority. It is important to bear in mind what the CBPMS is and what it is not. It is a system for managing the interactions between an organisation and their policy based systems. It is not a policy specification language or a complete access control model –

its expressiveness and flexibility are dependant upon the policy specification language adopted. It can equally as well be applied to obligation policies, configuration policies, or any policy specification approach that provides the basic semantic requirements of the Policy Reasoning Service, described in section 3.3.4. The CBPMS has no interest in the contents of policies and policy decisions beyond those requirements.

Therefore, this work does not overly concern itself with demonstrating that the CBPM model can express the various textbook properties (e.g. separation of duties or controls on delegations) that are generally required in order to validate AC models or policy specification languages. The validation of this work concentrates on demonstrating that the CBPMS model can be applied to a wide range of real-world organisations and chapter 5 is devoted to that task. Appendix 5 presents a series of brief explanations as to how the CBPMS can be used to address the most commonly cited requirements of policy systems.

Chapter 4. CBPMS Architecture and Implementation

Having defined the various components and semantics of the community based model, the problem of turning the theoretical solution into a practical system which human organisations can utilise to govern their interactions with policy based systems remains. In essence, this requires an implementation of the model as a software system. This is described in several stages. First an abstract conceptual architecture is outlined, an architecture formed by dividing the system into its logical components and defining the basic functions, input-output characteristics and the interactions between the components. This conceptual architecture is designed in such a way as to allow it to support as wide a variety as possible in terms of policy specification languages and policy enforcement systems. A set of components and a definition of their APIs emerges from this analysis, describing the basic characteristics that a CBPMS should exhibit. In the second part of the chapter, the practical implementation of the CBPMS is described, illustrating how particular features of a policy specification language can be integrated into this conceptual architecture to form a fully featured PBM system. The policy management platform and its user interface, which provides a convenient mechanism that allows communities of users to manage the policy of their organisation through the community based paradigm, is also introduced.

4.1 Conceptual Architecture

At the highest level, the workings of the CBPMS can be divided into two different functions. Firstly, it serves as what is termed a Policy Decision Point (PDP) in the IETF architecture (see figure 2.6). In the most general terms, this function can be described as responding to events by evaluating the appropriate policies and returning the result of those policy evaluations if required. Secondly, it serves as a policy management system, corresponding with the Policy Management Tool of the IETF architecture, the function of which can be described, again in the most general terms, as responding to management requests by updating the state of the system in accordance with the rules for the operational semantics of the CBPMS and returning the result of that management request if desired.

A service oriented paradigm is adopted as the basic architectural approach. That is to say that the system is divided into a number of discrete providers of well-defined and stateless services and these services are orchestrated in order to provide the required

functionality of the system. A service-oriented architecture eases the integration of the CBPMS with existing policy systems and network architectures, as well as allowing other systems to gain selective access to information about the state of the policy system. A top down approach to the definition of the services that make up the system is adopted, defining each component in the system in terms of simple services that it provides and the functions that it provides to the consumers of the service. This strategy for defining service based architectures closely follows the OASIS reference methodology for service architectures [oasis06]. It should be noted that such a top-down functional analysis mirrors the approach to the analysis of the organisation in the CBPM model which decomposes an organisation progressively into simple units.

Community Policy Decision Service

The first level of analysis of the functioning of the CBPMS - as a service which acts as a PDP, denoted the Community Policy Decision Service (CPDS), pictured in figure 4.1.

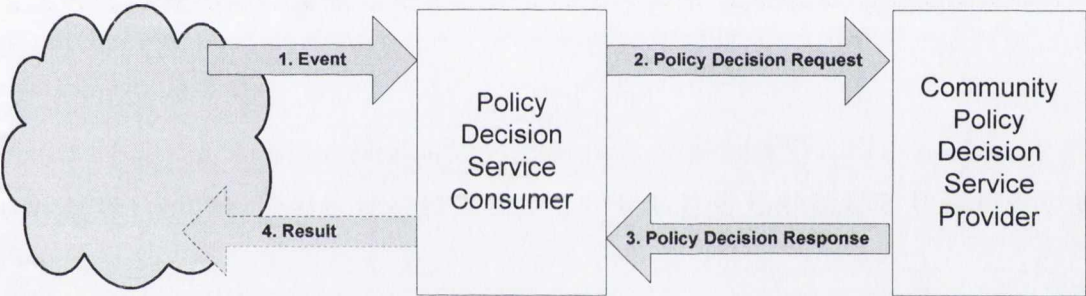


Figure 4.1 CBPMS Community Policy Decision Service Architecture

This works as follows. An event occurs, causing the consumer to issue an appropriate policy decision request to the policy decision service provider. The policy decision service provider evaluates the appropriate policies for the request and returns the result of the evaluation to the consumer in a form that is understood by that consumer.

Community Policy Management Service

The high level architecture for the Community Policy Management Service (CPMS) is pictured in figure 4.2 and is almost identical to the CPDS above. Again it works as follows. An event occurs somewhere in the universe, causing the policy management consumer to send a policy management request to the Policy Management Service Provider. The Policy Management Service Provider updates the state of the system and sends a response to the consumer if that is appropriate in either case. Although the architecture remains abstract, some details can already be commented upon. Firstly, the general question of how events are generated within the system can be ignored.

Secondly, how the results of policy decisions are used by policy consumers is beyond the scope of the CBPMS. Those details are, to a very large extent, application specific. The CBPMS is concerned with the management of an organisation’s policies and it aims to deliver an architecture and community based model of management that can be used as a simple service by any type of policy-aware system.

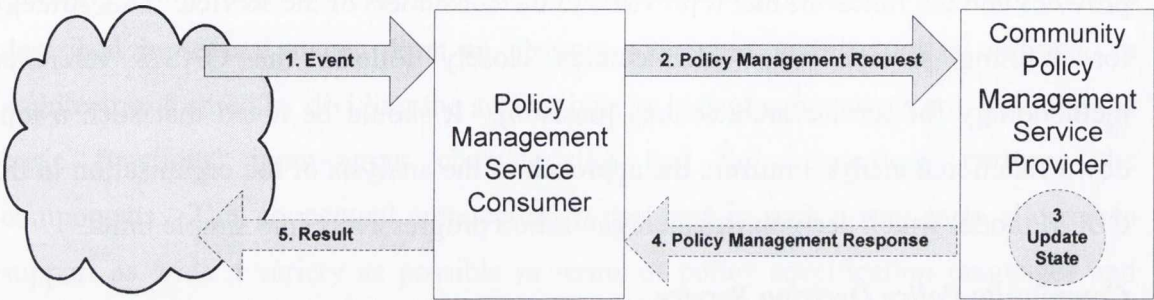


Figure 4.2 CBPMS Policy Management Service Architecture

4.1.1 State Management – Community & Resource Repositories

Having defined the two fundamental services of the system, it can be observed that they depend on the same state information. Thus, in order to further decompose the system, the data from the service logic is separated out and encapsulated in a service, which provides only those functions which are most closely bound to the retrieval and modification of that data. This allows the modelling of both the decision service and the management service above so that they act as consumers of this state management service. In fact, better is possible. The state of the CBPMS is defined as the triple $CBPMS = (\Omega, R_{\Sigma}, PRS)$. Leaving aside the PRS for the moment, since it is not tightly coupled to the rest, being only related through resource authorities, the required interactions with the community structure and the set of resource models can be modelled as separate services. Thus, responsibility for the management of state information regarding the system is divided into two separate services, the Community Record Management Service (CRMS) which manages the community structure, and the Resource Authority Information Service (RAIS) which provides access to information about the resource models representing managed resources.

4.1.2 Serialised Community Structure

For convenience, consider that both the RAIS and the CRMS essentially consist of a repository of resource models and a repository of community structures respectively, along with functions for accessing the entries in that repository. Since the specification

for the community structure is $\Omega = (C_{\Sigma}, p, f)$, this triple must be translated into a set of community structures.

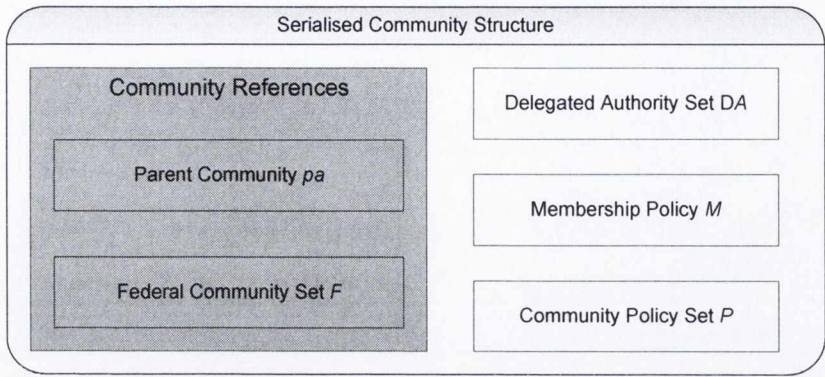


Figure 4.3 CBPMS Community Record Structure

This can be done by incorporating these functions into the data structure by including a reference to a parent community with each community and a set of references to federal communities which could be empty. This serialisation allows the community structure Ω to be treated as a set of community structures which are described by $c_i = (DA, P, m, pa, F)$ as pictured in figure 4.3

This structure represents the minimum amount of information required to represent a community within the community based policy management model. To consider this model from the point of view of policy organisation within a policy based management system; in order to provide the system with the necessary information to capture the functional relationships of the organisation, the policies are associated with metadata which situates each set of policies within the community hierarchy. Ignoring the conceptual model altogether and simply considering the problem of minimising the number of policies which must be evaluated in any policy based management system, all of the meta-data is useful in eliminating sets of policies from searches when making policy decisions. Thus, by leveraging the knowledge about the organisation embodied in the community hierarchy, the policies in the system are effectively partitioned into sets, many of which can be excluded from any particular runtime policy search.

When it comes to the resource model repository, there is no such translation to do as there are no functions which relate the set of resource models to one another. Thus, the resource model repository can be considered to be a set of resource structures, where each resource structure consists of 2 trees (or more precisely partially ordered sets), one representing the action authority tree, the other representing the target authority tree.

4.1.3 CBPMS Service Based Architecture

The conceptual architecture of the Community Based Policy Management System, decomposes the system into four basic component services. The Resource Authority Management Service (RAIS) and the Community Record Management Service (CRMS) both provide access to the repositories containing the records of the resources and the communities managed by the system respectively. The Community Policy Decision Service (CPDS) and Community Policy Management Service (CPMS) correspond to the PDP and the Policy Management Tool respectively in the IETF architecture. By decomposing the system into four services in this manner, much of the system can be treated as a set of ‘black boxes’ and the specification can be simplified when describing the functionality required to support the CBMPS organisational modelling approach.

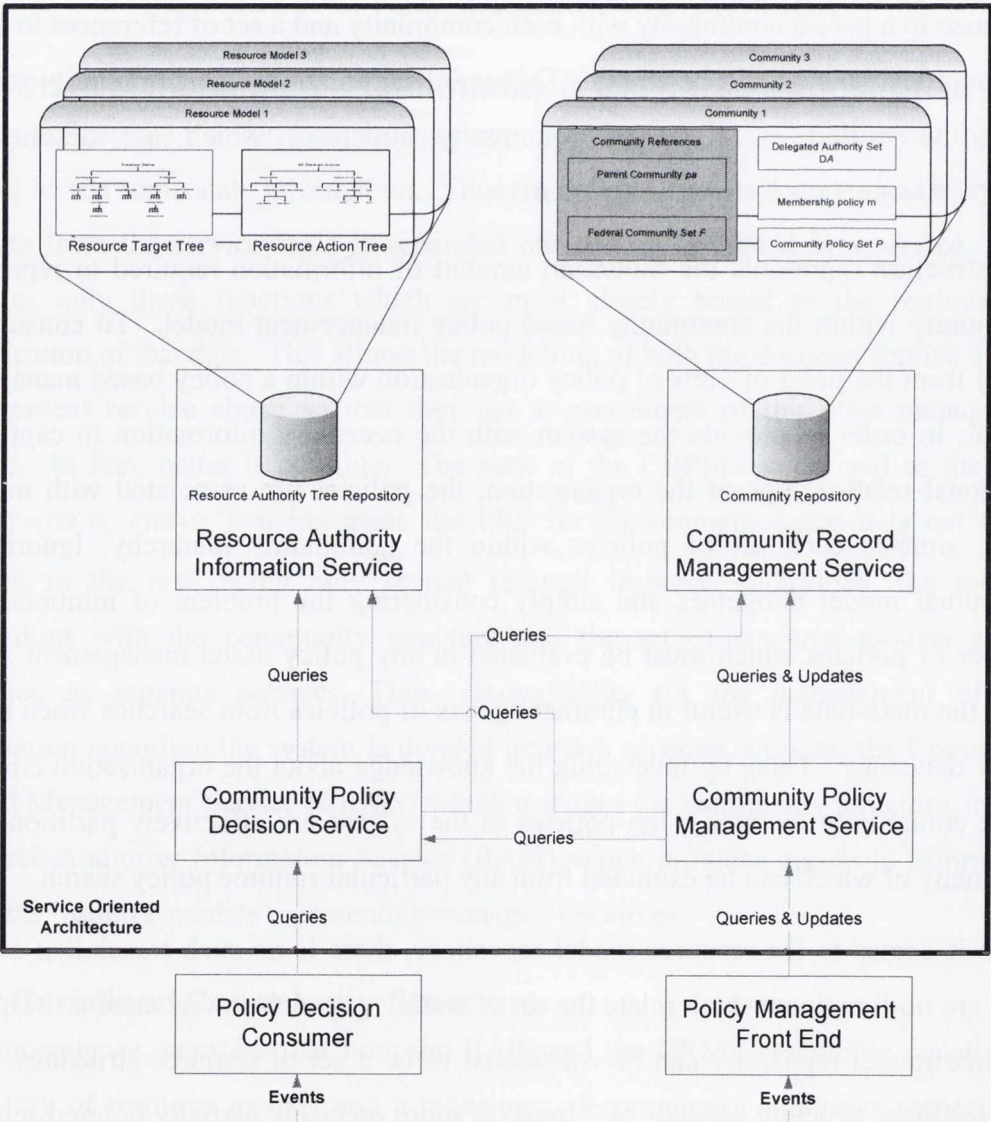


Figure 4.4 Community Based Policy Management System (CBPMS) – Service Architecture

Figure 4.4 shows the basic architecture of the CBPMS. The arrows between the services show where services within the system are consumers of other services. Thus, for example, the CPMS uses the CPDS to manage access control to its own functionality, as described in the section on self-management in the previous chapter, and it also uses the RAIS and CRMS services. Those arrows which are marked as *queries* indicate that the interaction between the services does not cause any state information to change, while those arrows marked as *queries & updates* indicate that the use of this service may cause the state of the system to be modified. It should be noted that, although the Policy Decision Consumer and the Policy Management Front End are strictly speaking outside the Service Oriented Architecture, they are considered to be part of the system and considerable detail is provided of the problems involved in the implementation of both of these elements of the system.

The Policy Decision Consumer (PDC) is a generic description for any process which uses the policy decision service in order to evaluate policies pertaining to its operation. The PDC detects events which require policy decisions and formulates them as policy decision requests. A PDC might be a standalone process, which detects events and maps them to policy decision requests, or it could be embedded in an application as the access control policy enforcement point, in which case the events which trigger the policy decision requests would be attempts by the user to carry out actions on resources and the policy decision would instruct the enforcement point to permit or deny the action, depending on the result of the policy evaluation. The PDC maps relevant events to policy decision requests and passes them to the CPDS. In the following material, the implementations of several different policy decision consumers are described.

4.2 RAIS - Resource Authority Information Service

The Resource Authority Information Service (RAIS) is the component of the CBPMS which has responsibility for managing access to the resource authority trees that serve as models of the resources managed by the system. Although any practical implementation of a CBPMS will need to include facilities for adding, removing and modifying resource authority trees in a controlled manner, the specification of such a tool is not relevant to the service provided to the other components of the CBPMS and can thus be safely ignored for the moment. The assumption that the resource authority repository contains an accurate and secure model of the resources managed by the system is adopted. Nor does this discussion concern itself with how the data is

physically stored, how the resource authority trees are referenced or how they are modelled in memory as there are many well known approaches to solving such problems. It is assumed that the RAIS maintains identifiers for the resource authority trees which it manages and that it makes these available to the other components in the system, allowing them to reference particular trees and particular nodes on those trees.

Implies Authority Function

From the point of view of the CPDS, the RAIS must provide a single, simple function, specified here using a pseudo-code language which includes the types of each argument. A similar abstract notation is used to describe the signatures of the basic functions provided by all services in the system in the CBPMS conceptual architecture.

```
Boolean result = impliesAuthority(ResourceAuthority ra1, ResourceAuthority ra2)
```

ra1 and *ra2* are resource authorities, with each resource authority being composed of a resource identifier, which refers to a resource model in the resource repository; a target node which refers to a node in the target node in the resource's action tree and an action node which refers to a node in the resource's action tree. The *impliesAuthority(ra1, ra2)* function returns true or false depending on whether resource authority *ra1* *implies authority for* resource authority *ra2* according to the hierarchical authority-centric model of the resource. Each resource authority is composed of a resource reference, a reference to a target node and a reference to an action node (*r,t,a*). The algorithm is extremely simple and can be stated as follows:

```
Boolean function impliesAuthority(ra1, ra2)
    ra1=(r1,t1,a1), ra2=(r2,t2,a2)
    IF r1 = r2 AND the repository contains a resource with that ID
    AND t1 >= t2 AND t1 and t2 are both valid target nodes
    AND a1 >= a2 AND a1 and a2 are both valid action nodes
    THEN
        Return true
    ELSE
        Return false
    END IF
END Function
```

The *impliesAuthority* function is the sole required interface between the CPDS and the RAIS in the CBPMS architecture. The CPDS is a consumer of the RAIS.

Other Resource Authority Information Functions

In addition to providing a service interface for the *impliesAuthority* relation on managed resources, the RAIS must also provide mechanisms to allow consumers of the service to discover details of the resource authority models contained in the repository, in

particular to allow the Policy Management User Interface (PMUI) to provide a visualisation of the resource authority trees to enable users of the system to conveniently delegate authority over particular resources to the communities that make up the organisation. However, the discussion of what specific information querying functions the RAIS should provide will be postponed, as this is highly dependant on the specific requirements of the PMUI. For the moment, this interface is left abstract and all that is noted is that, in addition to providing the *impliesAuthority* function, the RAIS provides functions which allow consumers to get information about the contents of the resource model repository and to update these models.

4.3 CRMS - Community Record Management Service

The CRMS is the component of the CBPMS which manages the updating and retrieval of the community records and elements within them, including their policy sets. It is assumed that the CRMS takes care of the mapping between identifiers for communities and their elements and the records stored in the repository and this discussion does not concern itself with the details of how such a mapping may be carried out as it there are many well known solutions to this problem.

4.3.1 Interactions between CRMS and CPDS

From the point of view of the CPDS, the CRMS must simply provide an interface which allows the CPDS to retrieve whatever elements of any community record that it requires – the well known *get* style of function. The basic functions are specified as follows. Each function is passed a community ID which is the CRMS identifier for a specific record. In addition, the *getPolicy* function is passed a policy ID, which is the CRMS identifier for a specific policy within a community policy set. The type of the data returned by the function is indicated to the left of the function name. If extra meta-data must be added to the community structure, extra functions can be easily added to support them in a similar manner.

```
CommunityID parent = getParent(CommunityID child)
Array[CommunityID] children = getChildren(CommunityID parent)
Array[CommunityID] federations = getFederations(CommunityID federated)
Array[PolicyID] policies = getPolicies(CommunityID com)
Policy pid = getPolicy(CommunityID com, PolicyID com)
PolicySpecification rule = getMembership(CommunityID com)
Array[DelegatedAuthority] authority = getDelegatedAuthorities(CommunityID com)
```

In addition to these basic data retrieval functions, on the principle that functions that are tightly bound to the data within the same service should be encompassed by it, and

thinking ahead in terms of the desire to minimise network traffic in a distributed implementation, these basic functions are expanded to add the following ones which require the employment of some basic logic.

```
Boolean result = isAncestor(CommunityID c1, CommunityID c2)
Boolean result = hasAuthority(CommunityID c, ResourceAuthority ra)
Array[PolicyID] policyIDs = getRelevantAuthorities(CommunityID c, ResourceAuthority ra)
```

The *isAncestor* function returns a Boolean value based on whether or not c1, the first community ID passed, refers to an “ancestor” of the second community passed. By ancestor, it is meant the first community is higher than the second community in the community hierarchy and that by composing calls of the parent function of the second community, the first community will be reached eventually. The algorithm is described in pseudo-code in Appendix 6.

The *hasAuthority* function returns true if the community in the first argument possesses a resource authority that implies authority for the resource authority in the argument. It tells us whether the community has authority to define policies that are relevant to the resource authority.

The *getRelevantPolicies* function returns an array of policy IDs, corresponding to the policies in the community policy set of the passed community ID, c, that are potentially relevant to the passed resource authority. By potentially relevant, it is meant that the policies are within scope as defined by comparing the resource authorities that are contained in each policy to the passed resource authority.

4.3.2 Interactions between CRMS and CPMS

The interaction between the CRMS and the CPMS is distinguished from other consumers of the CRMS service as the CPMS requires an interface which allows it to change the data stored in the Community Repository – thereby changing the state of the CBPMS. Here a design decision is required, an interface could be implemented which allows general purpose updating of community records, allowing consumers of the service to update community records with arbitrary data of the right type, trusting the CPMS to use the functions in such a way as to ensure that the community structure remained valid as the state changes. The second possibility is to implement the community primitives as the interface, thereby putting the burden of ensuring a valid community structure onto the CPMS.

Due to the fact that the management primitives described in the pervious chapter should be the only operations which update the state of the system, these primitives are tightly coupled with the data in the repository. Therefore, it makes sense to directly translate the management primitives into an interface definition for this service. The following functions serve as the service interface and again indicate the type of response that they return before the function names.

```
CommunityID new_community = genesis()
void grant(CommunityID c, ResourceAuthority ra)
void expel(CommunityID c)
CommunityID child = spawn(CommunityID parent)
void cull(CommunityID child)
void delegate(CommunityID parent, ResourceAuthority ra, CommunityID child)
void recall(CommunityID parent, GrantedAuthority ga, CommunityID child)
PolicyID pid = policy(CommunityID c, Policy p)
void revoke(CommunityID c, PolicyID pid)
void federate(CommunityID c, CommunityID federal)
void withdraw(CommunityID c, CommunityID federal)
void gatekeeper(CommunityID c, PolicySpecification ps)
void liberate(CommunityID parent, CommunityID child)
void merge(CommunityID child, CommunityID parent)
```

The semantics of these primitives are precisely defined in appendix 3. Note that the only functions which return information (aside from error codes and success codes which are omitted from this discussion) are those which create a new community and create a new policy, all the rest are specified as having a void return type. In order to make this interface more convenient for the policy management service, a function is added to these basic primitives which allows policies to be updated rather than having to revoke the old policy and specify a new one. This has exactly the same semantics as a sequential call to *delegate* to define the new policy followed by a call to revoke to delete the old policy. The only difference is that the policy retains its Policy ID across the operation, something that can not be guaranteed with the alternative since the generation of policy identifiers is an internal matter for the CRMS. This completes the basic interface definition for the CRMS.

```
void updatePolicy(PolicyID pid, Policy p)
```


4.4 CPDS Community Policy Decision Service

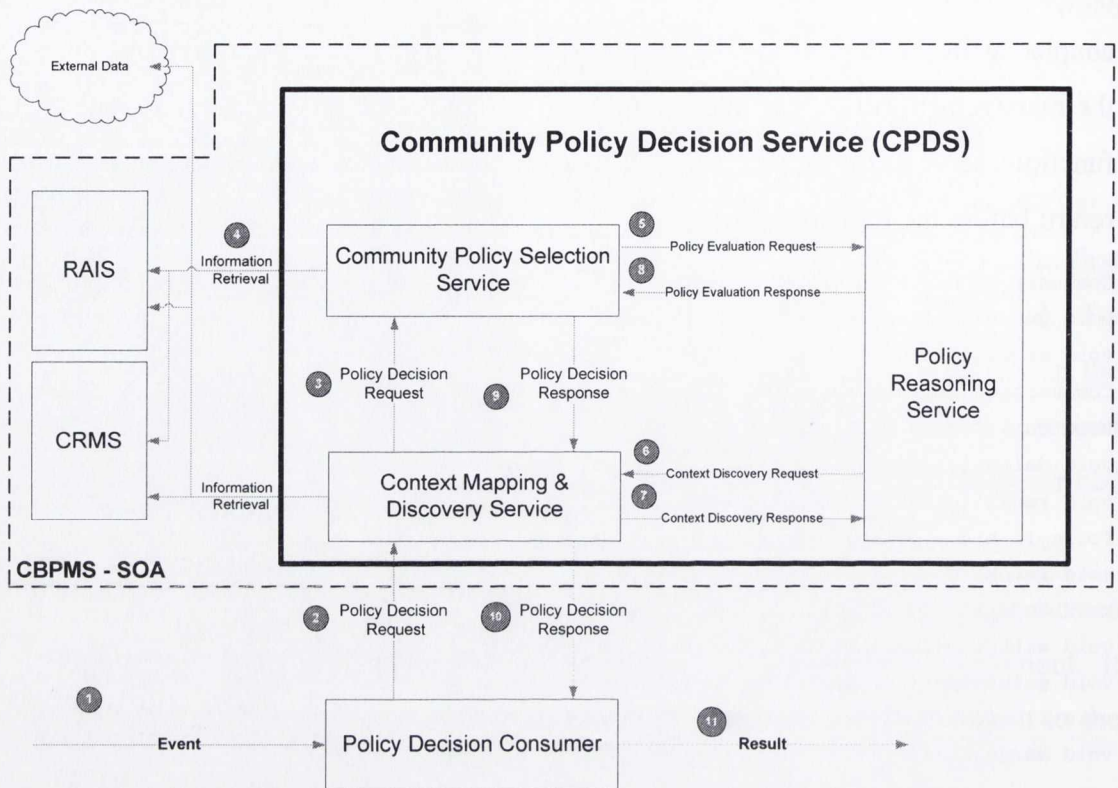


Figure 4.5 CPDS architecture

The Community Policy Decision Service is the component of the CBPMS which interacts with policy decision consumers. The CPDS is the Policy Decision Point (PDP) of the IETF policy architecture and it serves the function of retrieving and evaluating the policies that correspond with particular policy requests. In order to structure this discussion and to allow it to focus on those features of the system which are critical to the operation of the CBPMS, a conceptual architecture of the CPDS is constructed and divided into components, which are themselves modelled as services. In figure 4.5, the services involved and their interactions are graphically illustrated. In brief the operation of the CPDS can be described as follows: an event occurs which prompts a policy decision consumer to send a policy decision request to the CPDS. This context of the request is mapped into the space of the CBPMS by the Context Mapping and Discovery Service (CMDS) with whatever contextual information the application requires regarding the context in which the request is made and the state of the world. This updated policy decision request is then sent to the community policy selection service (CPSS). The CPSS uses the *decision* function, which implements the policy decision algorithm described in chapter 3, to select those policies to evaluate in response to the

request (according to the CPSS algorithm below) and repeatedly sends them to the Policy Reasoning Service (PRS) for evaluation. The PRS may use the CMDS to discover further contextual information that might be required in order to evaluate policy specifications. After evaluating each policy specification, the PRS returns a policy evaluation response to the CMDS. This continues until the policy evaluation Response evaluates as *applicable*, whereupon it sends a Policy Decision response back to the CMDS, which, if required, maps the response back into the space of the policy consumer. Depending on the contents of this response the policy consumer, when being deployed to provide responses to access control decisions, may then perform the requested action on the relevant resource. The API of the CPDS is comprised of a single function whose signature is as follows:

```
PolicyDecision pd = decision(PolicyDecisionRequest pdr)
```

4.4.1 CMDS - Context Mapping & Discovery Service

The Context Mapping and Discovery Service (CMDS) receives access control requests from a Policy Decision Consumer and returns policy decisions. It is a component that is tightly coupled with the Policy Decision Consumer that it interacts with. Its role in the function of the CPDS is twofold. Firstly, it maps the context of the policy decision request to the internal context of the CBPMS. That is to say that it translates the information contained in the policy decision request into the format required for policy evaluation. Depending on the PDC, this might involve the identification of a correct resource reference and target and action nodes on a resource model, the identification of the community to whom the request should be addressed (i.e. the community with ultimate authority over the resource) and the modification of the context included in the request to add information that might be required for policy evaluation. For example, the CMDS can be responsible for session management by adding a session key to the context of each incoming request – allowing policy evaluations to make use of session based information despite the stateless nature of the service.

A policy decision request, as defined in the previous chapter, consists of three components. It contains a resource authority, a community reference and a context data structure. The exact steps involved in context mapping are very much application dependant and the specification of the input-output characteristics of this component from the point of view of the PDC, is simply that it should accept a policy decision request structure and should return a policy decision response in the policy specification

language that the PDC understands. From the point of view of the interactions between the CMDS and the Community Policy Selection Service, the interface is the very same, the difference being that in this case the CMDS acts as the policy decision consumer. Thus, it can be stated that the CMDS acts as a proxy which transforms the incoming request into the internal variable space. By including the CMDS in the conceptual architecture in such a role, the PDC is able to only transmit a simplified version of the policy decision request and a small subset of the contextual information that is required in order to evaluate policies. This allows the minimisation of network traffic between the PDCs and the CBPMS as well as allowing the incorporation of changes in the basic resource mapping into the system without affecting the coding of the PDCs.

In terms of the Resource Authority that is received by the CMDS, there is only one requirement: that the CMDS can map it to a resource identifier, a target node and an action node. The PDC could be programmed so that it sends policy decision requests with this information already embedded in them, or the CBPMS could be integrated with existing PDCs which generate requests in different formats, by introducing the CMDS service which maps the native requests to ones that are suitably formatted. There is no need to completely specify the form of the context in such a request, Any information that the PDC wishes to pass to the CBPMS to enable it to make this decision is allowed, for example the parameters of the action that a user wishes to carry out. However, this information is not directly relevant to the operation of the CBPMS algorithm, so it is grouped together as context.

The specific nature of the translations that are carried out in the CMDS depend on the model of the resource and the capabilities of the PEP. From the point of view of the conceptual architecture, it can be treated as a black box and it can be assumed that the appropriate community identifier, resource identifier, target and action can be derived from the policy decision request or from state information that it has stored from previous requests.

The second role of the CMDS in this architecture is to provide contextual information on demand to the Policy Reasoning Service (PRS). The PRS evaluates policy specifications in a given context. However, there are potentially a very large number of contextual variables which could be utilised by a PRS and it can thus be impractical to transmit each and every single context variable from the CBPMS into the PRS. The context discovery aspect of the CMDS, the *discover* function, allows the PRS to

discover contextual information pertaining to a particular request in order for it to evaluate a policy specification. This function is naturally dependant on the PRS and the underlying policy evaluation engine being equipped with the ability to make such queries. In this conceptual architecture the form of the context variable or the returned context data is not defined –that detail is left to the implementation.

The API of the CMDS is thus composed of two different functions with the following signatures:

```
policyDecision decision(PolicyDecisionRequest)
ContextData discover(ContextVariable)
```

4.4.2 PRS - Policy Reasoning Service

From the point of view of the CBPMS, the choice of policy specification approach, or the policy evaluation engine used is irrelevant. Policies are simply messages that are passed to a Policy Reasoning Service (PRS) for evaluation. Only the results of this evaluation are of interest to the policy selection algorithm upon which the CBPMS depends. From the point of view of the CPSS, which implements the key algorithm of the CBPMS, there are only two possible results of a policy evaluation. A policy decision either evaluates to *not applicable* or to some other value which is passed back to the intact and is evaluated by the CPSS as being *applicable*. This allows the integration of various approaches to policy specification and evaluation into the CBPMS. Policies are viewed as anything that can be evaluated to return a result with a given context and the set of results that the policy evaluation should return are not specified, it is simply required that a subset of the results returned should be mapped to the *applicable* value of the CPSS, with the rest of the return values being treated as *not applicable* by the CPSS. The CPSS algorithm does not need to understand the semantics of the policy decisions returned by the PRS, it merely needs to be able to identify which policies are not applicable in the given context. The only other requirement for the form of the policy decision response that is returned by the PRS to the CPSS is that the CPSS must be able to detect policy results which are semantically equal from the point of view of the CBPMS access control decision consumer (i.e. the PEP).

The syntax or the protocol used for the policy decision request or the policy decision response sent between the CPSS and the PRS is not defined. It is merely required that the policy decision request includes a policy, denoted in whatever policy specification approach is in use (which is the policy specification language used by the PRS), and that

it also includes whatever contextual information is available about the access control request, in a *context* data structure. Once again the syntax or semantic content of this *context* structure is left undefined, leaving it up to the particular implementation of the CPSS and the PRS to dictate how contextual information is represented. Thus, the requirements for the PRS are defined so that it must support the following function:

```
PolicyDecision pd = evaluatePolicy(Policy p, Context con)
```

where *p* is a policy specified in a form that is understood by the PRS, *con* is a data structure containing contextual information in a form that is understood by the PRS. The requirement for the returned result is simply that the CPSS can map it to either *applicable* or *not applicable* and that it can identify semantically equal results. Thus, it is also required that the PRS should support three additional functions for evaluating the results of policy decisions as explained in the previous chapter

```
Boolean isApplicable = evaluateResult(PolicyDecisionResult pdr)
Boolean isEqual = isSemanticallyEqual(PolicyDecisionResult pdr1, PolicyDecisionResult pdr2)
PolicyDecision pd = notApplicable()
```

Exactly how the PRS carries out the mapping required in these functions depends on the semantics of the policy language being used. By adopting a generic approach to policy and policy evaluation results and by separating out the responsibility for semantic interpretation of policy rules and policy decisions into a separate service which can be viewed as a black box by the CPSS (which performs the policy selection algorithm), the integration of a wide range of policy specification approaches and policy evaluation engines into the CBPMS is supported. In the following section it will be shown, in detail, how a CBPMS can be implemented to support different policy specification approaches, and how it can even support multiple different policy specification languages within the same system.

4.4.3 CPSS - Community Policy Selection Service

The CPSS is responsible for performing the key algorithm that is required in order to implement a Community Based Policy Management System. It receives policy decision requests, selects policies to evaluate and interprets the results of the policy evaluations in accordance with an algorithm. This service is required to expose a single function to consumers, namely:

```
policyDecision pd = decision(PolicyDecisionRequest pdr)
```

It should be recalled, from above, that a Policy Decision Request is defined as the triple (*Resource Authority, Community, Context*). Since a CBPMS may manage the policy of

multiple distinct organisations, the request must specify which from which community the access is requested from. The CDPS uses the Community identifier to fetch the community record from the CRMS. The algorithm which the CPDS executes when this function is called implements the decision function specified in chapter 3. It controls which policies in the system are evaluated in response to a policy request and how the results of these evaluations are combined.

The CPSS first needs to locate the community or federation which has ultimate authority over the resource to which access is being requested since the policies of this community are paramount over all others according to the conceptual model. No policies defined for the resource in any other community can over-ride policies defined by the community with ultimate authority over the resource and so there is no point in evaluating policies from elsewhere in the system until it is known that there is no applicable access control policy defined by its owner. Thus, the community tree is traversed from the starting community until we reach the community which either possesses ultimate authority for the resource, or to the federation node which has been delegated the authority by one of its children (since the policies applied to the resource by the community that possesses ultimate authority over the resource do not apply to external organisations which have been delegated the resource by a federation delegations from federal communities to their children do not have to be followed in order to find all potentially relevant policies).

Each policy is stored with a resource authority which defines the scope of the policy. This resource authority can be interpreted as the definition of the set of events which cause this policy to be fired, where the events are access control requests that the policy's resource authority implies authority for. So, a policy stored as (policy, RA) can be understood to mean "this policy will apply to any requests for this resource which have actions $\leq$ RA(action) and targets $\leq$ RA(target) on the action and target trees respectively.

Those policies that are relevant to the policy decision request are evaluated and tested for applicability in the given context. If any of the policies are applicable in the given context, the results of the applicable policies are tested to make sure that they are all semantically equivalent. If they are not semantically equivalent, then a policy conflict has been identified and the output of the policy conflict resolution algorithm is returned.

If they are all semantically equivalent, then the policy decision result is returned to the consumer of the CDPS.

If none of the policies in the community record are relevant to the access control request, or if none of the policies are applicable in the given context, then the search for relevant policies in other communities begins. The community records for the child communities which have been delegated authority over the requested resource are loaded and searched for relevant policies. The algorithm recursively runs the policy search on the new community. Once each child community has returned its policy decision the conflict detection algorithm is run once again to generate the community's policy decision that will be returned to the consumer.

4.4.4 Community Policy Management Service & User Interface

The Community Policy Management Service (CPMS) acts as the Policy Enforcement Point for the community model itself and in particular it enforces access control policies to the Community Record Management Service (CRMS). The API of the CPMS is dependant on the particular resource model of the CBPMS resource that is used. Essentially, each non-composite node on the action tree for the CBPMS resource should correspond to an API call. One option for the CPMS is to simply use the CRMS primitives and directly enforce access to these. However, it is much more convenient to support higher level functionality in the CPMS and map these to CRMS API calls in the CPMS. Thus, for example, function calls such as *setMandate (Community, Policy)* could be supported which would be de-composed by the CPMS into two CBPMS primitive calls as detailed in the previous chapter. Similarly, a *createFederation (Community)* API function to create a new root community and federate the community passed in the argument to it could be created. In addition to its function in managing access control to the actions on the resource action tree and decomposing these actions into primitive calls, the CPMS is the service which provides an API to the User Interface to allow it to provide visualisations of the community structure and the policies contained therein. The practical problem of integrating a user interface into the community management system is described in section 4.8.

4.5 Full Services API

Figure 4.6 represents the interactions and the co-dependencies between the services which make up the CPDS. Having defined the functions and inter-dependencies

between each service, the entire API of the system together is shown in figure 4.7. The arrows between services indicate that the service pointed towards acts as a provider. The dotted line presents the public interface of the CBPMS service.

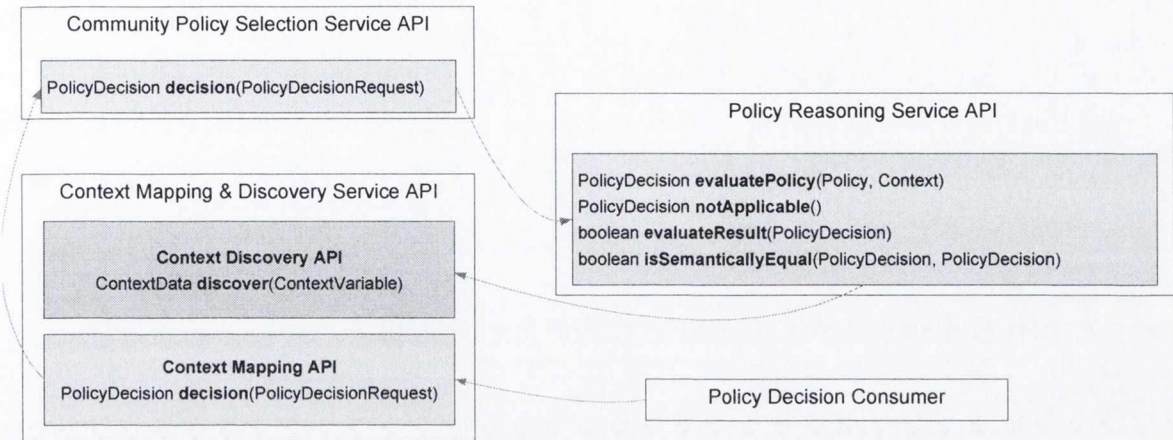


Figure 4.6 APIs & Interactions between Component Services of Community Policy Decision Service

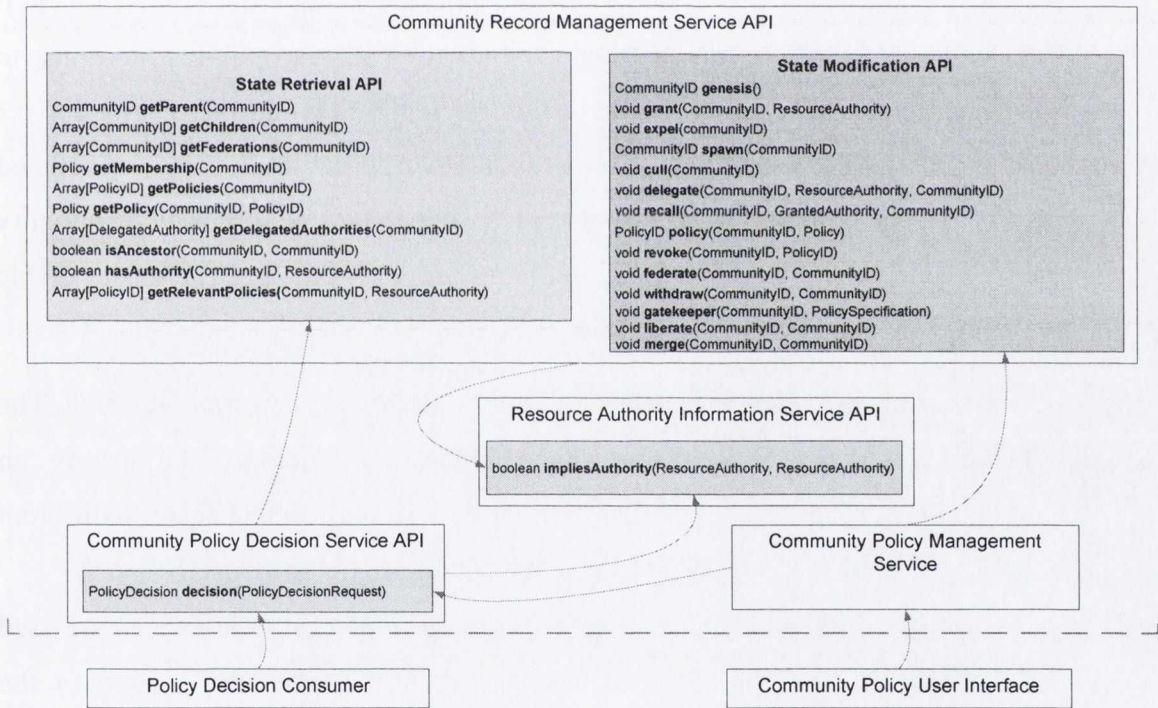


Figure 4.7 Interactions between Component Services of CBPMS

4.6 Full Implementation with XACML

The XACML 2.0 draft specification [oasis05] was selected as a suitable policy specification language for the initial implementation. XACML has a number of features that make it particularly attractive to the goals of this research. It is specified as a base system with a number of optional extensions which cover such areas as role based access control and hierarchically organised resources. This means that many of the

common features of policy specification languages, such as roles, are not embedded in the core specification and can thus be dispensed with in this case where they are not required. XACML also includes many convenient features which can be used by the CBPMS system to enable automatic conflict resolution through policy combining algorithms and other useful facilities such as textual descriptions of policy specifications. The following section describes how each component of the architecture has been implemented with regards to this system. The implementation is based upon a combination of the Java and PHP programming languages.

4.6.1 Community Record Management Service

The first step in the implementation of the CRMS is to define a concrete syntax for the encoding of community records. In order to support interoperability and open standards, XML was used as the underlying markup language. XML is an obvious choice due to the existence of a large number of libraries and applications which can process it, and it provides natural support for the basic tree-structure which is required in order to describe the community hierarchy. The community definition and the other elements that make up the CBPMS XML namespace are defined by the XML schema, included in appendix 7. The community structure follows the design of the serialised community structure in section 4.1.2 and most other elements are very simple. The most notable features of the schema are the following

- Each community element contains a *Child Combining Algorithm* attribute. The *Child Combining Algorithm* attribute allows a community to specify an automatic conflict resolution algorithm for policy conflicts that occur in different branches of the community hierarchy beneath the community itself.
- The schema contains mandatory *description* and *title* attributes – which are used to provide a high level natural language description of the function of the element, in order to assist human comprehension of the system and to provide an audit trail in terms of the translation of high level goals into concrete policies.
- The *GrantedAuthoritySet* element contains a record of all the authority delegated by the community. This is the converse of the *DelegatedAuthoritySet* element, which contains a record of all the authority delegated to the community.
- Resource IDs, Policy IDs and DelegatedAuthorityIDs and CommunityIDs are all specified as URIs.

4.6.2 CRMS Distributed Architecture

The *GrantedAuthoritySet* element, mentioned above, was added to the CBPMS schema to allow the CRMS to be more securely distributed across multiple independently managed domains and helps to optimise the implementation of the policy search algorithm, by limiting the number of community structures that must be retrieved from storage. This works since the policy search algorithm can simply retrieve a single community structure and if there are no applicable community policies, it can determine which further community structures to search by simply checking which communities have been granted relevant authorities by this community without having to retrieve all the children and federal community structures. From a security point of view it means that, in an arrangement where the community record management and storage is distributed across multiple administrative or organisational domains, a community can guarantee that only the authority that this community has legitimately delegated – as recorded in the community record itself – will be visible to the algorithm and that corruption of community records in other domains cannot provide access to authority that this community has not granted. In particular this means that the CRMS algorithm guarantees that the organisation which is the owner of the resource can host its CMRS locally and limit all authority granted to other organisations strictly to that legitimately delegated to them, even if their community records become corrupted.

The architecture depends on the following semantics being obeyed by the CRMS services in respect to the URIs contained in the *CommunityIds* that are passed to it. If the URI contained in these elements refers to a remote system, the CRMS forwards the request to that system by using its Web Service Interface. In general the problem of mapping URIs to web service API calls is trivial for almost all functions supported by the CRMS service. The only cases that need special consideration for a distributed architecture are the *create()* and *spawn()* primitives since which CRMS community record repository should store the new community structure must be known. This is resolved by adopting the convention that the *spawn* primitive always creates a new community stored in the same community repository that houses its parent community structure and the *create* primitive uses that community that is specified in the policy decision request. It can be assumed that all other elements of a community, including community policies are managed by the same service provider that manages the community which they apply to. Although it should be noted that these are merely

conventions which will ensure that an organisation's entire specification will remain managed by the same service provider in the distributed network for security reasons and they are not a requirement of the model.

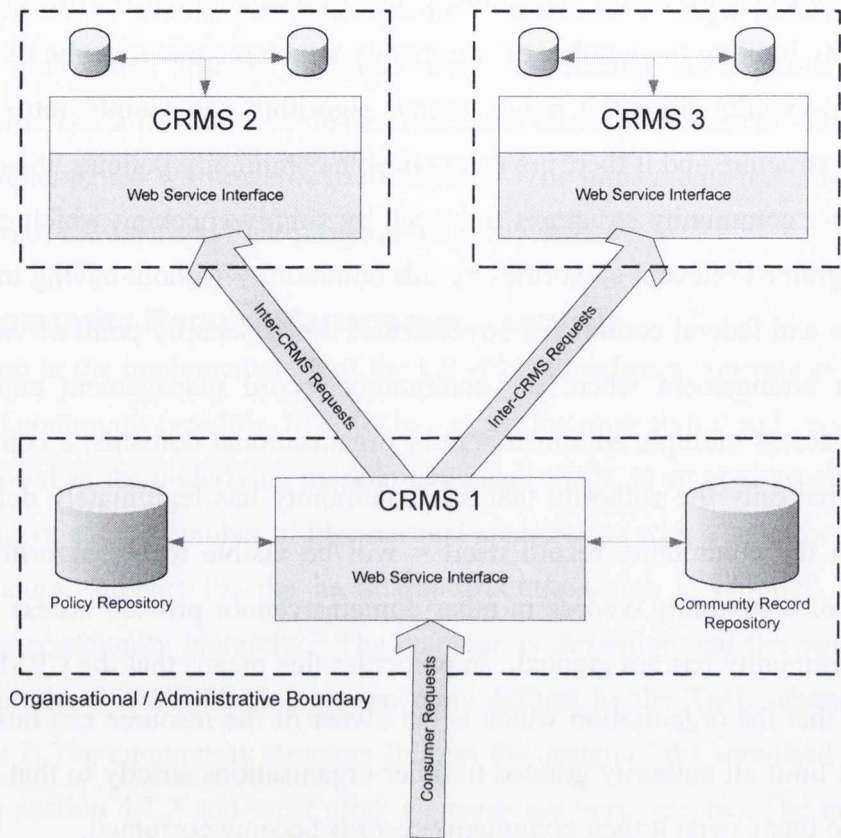


Figure 4.8 Distributed CRMS Architecture

4.6.3 Resource Authority Information Service

The RAIS is implemented as a simple service. Within a CBPMS system, multiple RAIS providers can be employed independently of each other without any requirement for inter-operability or co-ordination between them. Each resource model is managed by a single RAIS (the distribution of a single resource model across multiple services is forbidden) and the server location information is embedded in the URI of resource references. Since resource models are self-contained structures which do not contain references to other resource models or other external data, the task of distributing the models across a distributed network of RAIS providers is trivial – any arbitrary RAIS can be chosen to host any resource model. Once again, and for much the same reasons as above, XML and XML-Schema are used to describe the resource structure. Rather than defining a full specification for an authority tree, several basic standard types of authority tree have been implemented and an interface which allows the definition of

arbitrary new trees has been provided. Recall that the service API of the RAIS consists of a single function:

```
boolean result = impliesAuthority (ResourceAuthority ra1, ResourceAuthority ra2)
```

For each distinct type of authority tree that is implemented, this class is extended to provide an implementation for the abstract functions in this base class. This inheritance based mechanism for supporting a variety of different authority tree types has the disadvantage of requiring a small amount of programming in order to provide support for resources with new authority models. However, this disadvantage is offset by the flexibility that such a mechanism provides. Several generic types of authority tree, suitable for describing a wide variety of resources can be constructed. Only in cases where there is a reason to take advantage of some particular aspect of a resource's semantic structure will new classes be needed. Thus authority trees can be created which are parameterised, dynamically constructed at run-time or use highly specific algorithms in order to create their partial order on the authority nodes.

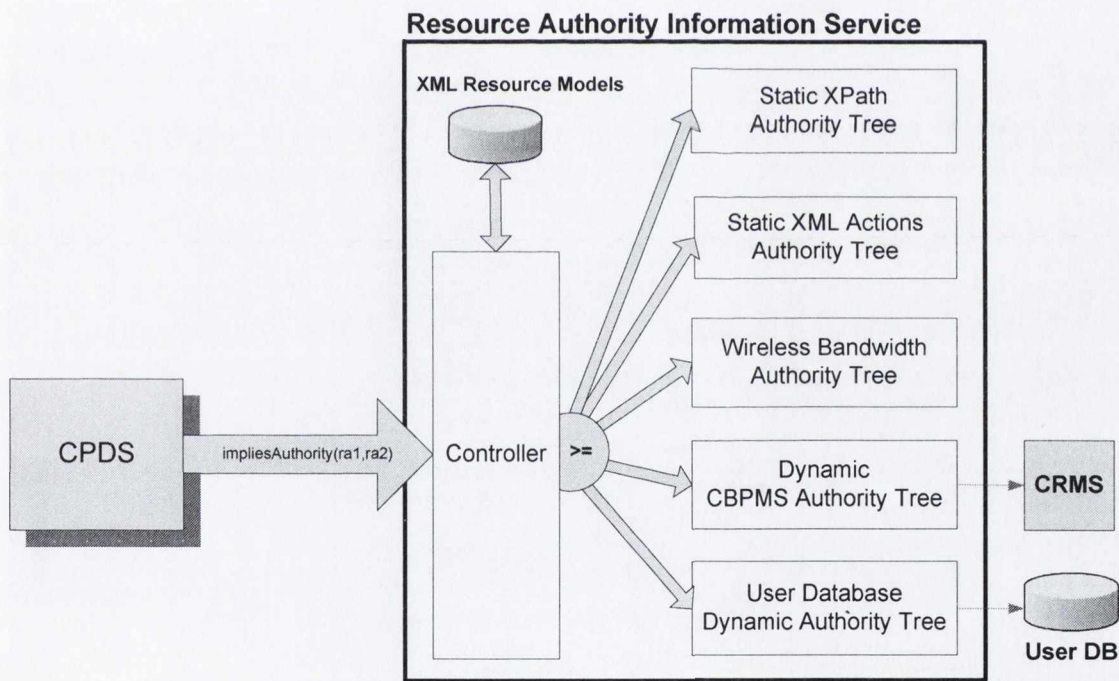


Figure 4.9 Resource Authority Information Service Design

The design of this service is shown in figure 4.9. The controller receives requests through its XPL-RPC web-service interface. It retrieves the XML resource model that is referenced in the resource authorities that have been passed to it. It loads objects of the classes specified for the target and the action tree respectively and uses their $\geq$ operator to compare the target and action tree respectively and returns true if both of

these also return true. Note that some of the authority tree classes pictured in the diagram interact with external services and data sources.

4.6.4 Context Mapping & Discovery Service

The CMDS performs two functions. Firstly, it serves as a decision request proxy, mapping the incoming policy request into the CBPMS context space. Because the goal is to support a wide variety of policy decision consumers, which might be very simple devices with limited processing capacity, no standard name space or data structures are imposed upon the contents of the incoming policy decision requests. Instead, the policy consumers are allowed to specify their context in whatever format is most convenient for them and this context is mapped to the CBPMS internal context format in the CMDS. In order to do this, the CMDS requires access to information which will allow it to perform the mapping for each individual policy decision consumer.

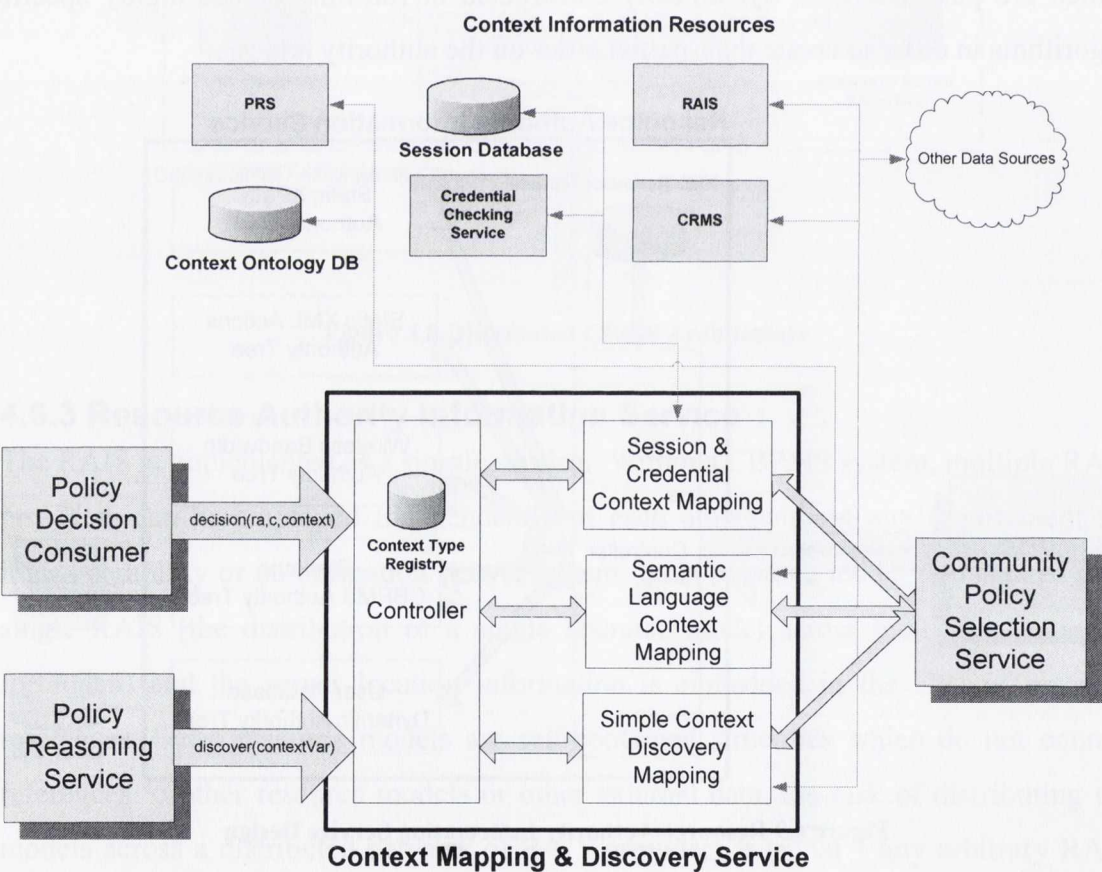


Figure 4.10 Context Mapping & Discovery Service Design

The CMDS must be able to map each policy decision request to a context mapping function. This is achieved using the same approach that was applied to the problem of policy modelling. Each request includes within it a reference to a context type. If the

CMDS supports this context type it launches a context mapping handler of the appropriate type which carries out the required context mapping. Once again, although this necessitates a certain amount of programming in order to integrate policy decision consumers which use new formats and semantics for the contexts that they transmit, generic handlers can be used to deal with a wide variety of incoming context models. Appendix 8 presents a detailed example of how the CMDS can be used to implement session and credential-handling functionality.

4.6.5 Community Policy Selection Service

The implementation of the CPSS is fairly straightforward. It implements the *decision* function formally described in section 4.3 and in appendix 2. The one variation on the basic algorithm that is introduced into this implementation concerns conflict detection and resolution. The community element contains an attribute which can be used to specify an automatic conflict resolution algorithm for conflicts that may arise between children on different branches of the community hierarchy beneath the community. This implementation supports the same conflict resolution algorithms and definition format as are supported by XACML 2.0 for policy conflicts.

In addition to providing support for algorithmic automatic conflict resolution of policy decisions, the CPSS also includes support for notification actions being attached to policy conflict events. If the policy specification language does not support automatic conflict resolution, the CPSS can still go some way towards resolving the problem. The community which must take responsibility for re-engineering their policies can always be identified. There are only two situations in which a policy conflict can arise in the system. The first is when the conflicting policies are defined within the same community and in this case it is that community which must re-engineer its policies in order to resolve the conflict. The second is where the conflict arises due to policies specified in two communities which do not have a direct hierarchical relationship with each other. In this case, there are a number of ways to solve the problem. One or other of the communities can change their conflicting policies in order to eliminate the conflict, or can change their membership policies so that actors will not hold simultaneous memberships of the two communities when acting upon the resource in question. However, the conflict may not be a consequence of a problem with policy specifications. It may signal a resource conflict between two groups within the same organisation, with mutually contradictory requirements towards a resource which they

share authority over. Thus, the policy engineering problem must be properly situated in the lowest common parent of the conflicting communities. It is only at that level of functional abstraction that a decision can be made to adjudicate between the competing and contradictory demands of the communities.

This implementation thus incorporates a facility for a policy decision conflict notification event to be generated whenever a policy conflict is encountered, in addition to whatever conflict resolution approach is in place. A message is sent to the relevant community's management, through the CPMS, indicating the nature of the conflict, from which a list of resolution options can be automatically generated. This functionality is described in the CPMS implementation section below.

4.6.6 Policy Reasoning Service

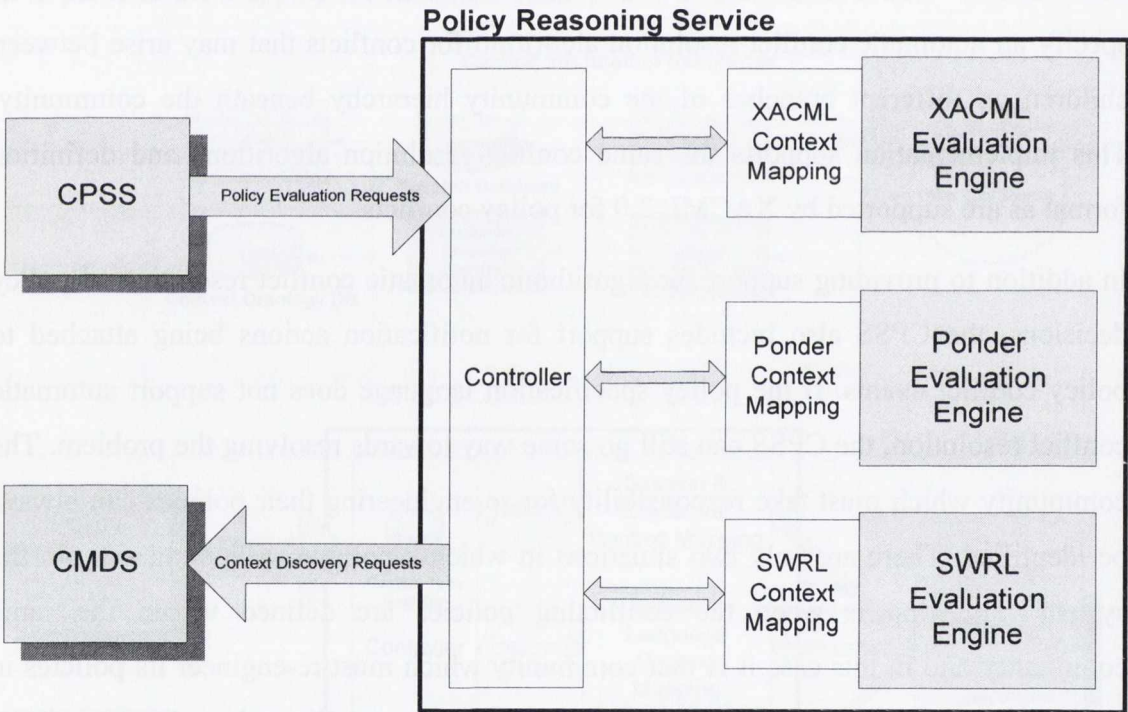


Figure 4.11 Policy Reasoning Service (PRS) Implementation

The design of the PRS closely mirrors that of the RAIS and CMDS. The PRS implements the functionality required by the community based model and its service interface is those four functions which were described in chapter 3. The one modification to the service interface is that this implementation requires an extra argument to each function, a string representing the policy specification language that is being used. This allows multiple policy evaluation engines to be incorporated into the same PRS, and different policy specifications can be mixed together in a community's

policy set. This implementation again uses a controller piece of code to receive the requests from the service consumer – in this case the CPSS. The controller passes the request to the appropriate policy evaluation engine for the policy specification language of the request. Again, this implementation is based upon an object oriented inheritance based mechanism. An abstract base class is defined to represent a generic policy evaluation engine and for every policy evaluation engine that is used, an inherited class which provides a concrete implementation of four basic methods is created. The implementation of this class for each policy evaluation engine must take care of invoking the appropriate functions within the policy specification engine and of populating the context-space of the engine with whatever data it requires from the CBPMS context in order to evaluate the policy. The nature of these tasks is entirely dependant on the policy evaluation software and the problems lie more in the realms of engineering than research, so they will not be dwelt upon.

4.6.7 Community Policy Management System

The final component of this implementation comprises the CPMS and the CPUI. Due to the nature of user interfaces this is also the most complicated component to specify. Although the CPMS and the CPUI are logically separate, the CPMS is embedded within the user interface. Logically, it can be said that a CPMS is a service which offers an interface to the CBPMS management primitives. Rather than being defined as part of the architectural specification, the functions that make up the CPMS interface are derived from the resource model of the CBPMS resource itself. That is to say that the functions of the CBPMS should correspond to nodes on the CBPMS resource action tree, and the node on the CBPMS target tree should be derivable from the function or its arguments.

The CPMS is responsible for enforcing access control policy relating to the CBPMS. Thus it acts as the policy enforcement point, or the policy decision consumer, for access attempts to the data which defines the state of the CBPMS. Whenever a user attempts to carry out a management action on the CBPMS resource, the CPMS generates a policy decision request and invoke the *decision* function of the CPDS, from which it receives a policy decision. Depending on the contents of this policy decision, the CPDS may then translate the function into a sequence of calls to the CRMS primitives. Thus, how the CPMS generates the policy decision request and the semantics that it will obey in relation to interpreting policy decisions must be specified. Since XACML is being used

as the access control policy specification language within this system's implementation, this encoding is chosen as the native policy decision language of the CPMS. This does not limit the CBPMS to this language, however, since a context mapping class can be implemented which translates policy decisions from other languages into the XACML format.

The CPMS can further be implemented to provide extra services to the communities in the system. That is to say that extra information pertaining to the communities managed by the CRMS can be placed within the CPMS. For example, this implementation of the CPMS adds two basic elements to each community structure – a message queue and a decision queue and voting module. These services and the information that they contain support a number of useful user interface applications which greatly assist in the collective management of communities, but they do not, in any way, affect the state of the communities in the CRMS, thus they are maintained separately, within the CPMS.

The CPMS is an XACML Deny-biased PEP which does not support any obligations. If a policy decision is "Permit", access is permitted, otherwise access is denied. The very simplest implementation of the CPMS provides access to the CBPMS management primitives and nothing else. It simply performs an access control check each time one of the functions is invoked and performs the action if the policy decision returned is equal to "*permit*". This simple CPMS implementation relies upon the CPUI to do generate meaningful values with which to populate the arguments of the management primitives. While it might work perfectly well, a practical implementation requires the provision of much more to the user than this simple interface. In order to design such a CPMS implementation, some of the requirements of a policy management system need to be examined from a user-focused point of view.

CPMS Requirements

The CPMS provides access control to the community records stored in the system. Since these records form the model which dictates access control to all of the resources managed by the system, it is particularly important to provide a secure implementation of the CPMS. The implementation of the CRMS, which is ultimately responsible for changing the state of the system needs to merely include security mechanisms to ensure that the state updating functions which it provides must be only invoked by a valid CPMS. Several different CPMS providers can be implemented, which may all use different models of the underlying CBPMS resource, at the same time, but it must be

ensured that only certified CPMS providers can access the CRMS primitives. There are many well-known ways to secure such interfaces which use point-to-point communication between well-defined and persistent entities and there is no point in going into them here. It suffices to say that the CRMS contains a list of valid CPMS consumers of its services and these registered consumers are the only entities permitted to invoke the CBPMS primitives. The CRMS places no restrictions on the invocation of primitives by registered CPMS providers, since it is the CPMS which enforces access control policy, the CRMS merely implements state updates defined by the primitives.

Thus, the CPMS must provide access control to the CBPMS services itself. In order to support a flexible, powerful and general purpose user interface for policy management, the CPMS must be able to provide fine-grained access control over the invocation of functions which cause the state of the system to be updated. Furthermore, for reasons of confidentiality it should also be able to provide fine grained access control over who can view elements in the community structure as well as for updates. It would be ideal if the access control had a granularity which allowed users to individually specify access control policies for each individual element within a community structure. There are many conceivable situations where a community might want to keep certain policies, certain sub-communities or certain authority delegations confidential from other users, while allowing other examples of similar elements to be viewed. Similarly, it would be ideal to be able to stipulate specific policies for updating specific elements of a community structure. For example, this would allow communities to delegate authority for updating a single policy to a sub-community and, similarly, specify policies which apply to single policies within the system, thus allowing policies to be defined that will constrain the form of other policies and act as meta-policies. All of this can be achieved by means of conditions within policy specifications. However, in order to take advantage of the hierarchical resource model and to allow a more flexible distribution of rights without having to rely on such meta-polices, as well as to minimise the number of policy specifications that must be evaluated for any given request, this fine granularity should be included in the resource model.

4.7 Community Policy Management - User Interface

The basic requirements for the CPUI were identified as the following:

- It should provide the ability for users to view any part of the community structures, their policies and other elements, all subject to access control policy.

- It should provide an indication of which communities a user is a member of
- It should provide a convenient way for users to generate resource authorities in such a way that only valid resource authorities can be generated, while granting the user full control over the granularity of the resource authorities produced.
- It should provide a visualisation of the hierarchically modelled resources
- It should provide a convenient interface for those functions which update the state of the CRMS. This includes, the creation of new communities, the distribution of authority to those communities through delegation and the specification of community policies.
- It should provide integration with a user-authentication service in order to ensure basic security.

The implementation is presented in sections, illustrated by screenshots from a working CBPMS system. The user interface is a web-based system, programmed in the PHP programming language.

4.7.1 Authentication and community membership control.

This CPUI implementation uses a simple authentication service for authentication management. The system presents the user with a login screen and when the user inputs a valid username and password into the system, the authentication server issues the user with a certificate of authentication. This certificate is added by the CPUI to the user's context that will be passed to the CPMS and onward to the CPDS for policy evaluations. Each and every action in this system is connected to a particular community in a particular organisation, but there may be many organisations modelled within the CBPMS, hence before taking any management actions, the user must connect themselves to a particular organisation to give their actions an organisational context.

4.7.2 Community Management Overview

Once the user has been authenticated and has chosen a community to connect to, they can enter the community management system as displayed in figure 4.12. An authentication certificate and the root community which they are connected to is transmitted with their context each time that a policy decision request is generated from their actions.

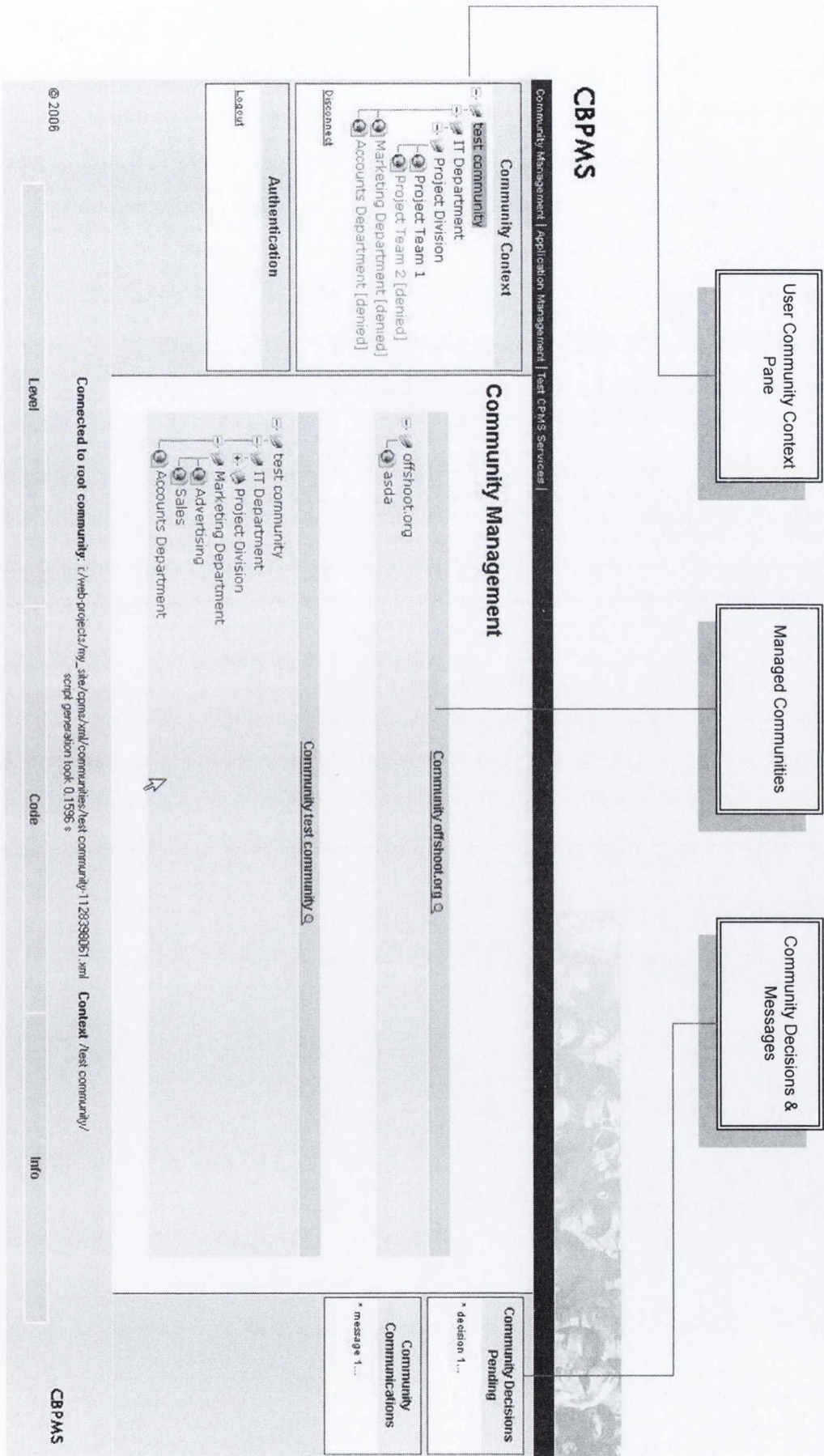


Figure 4.12 CBPMS Community Management System

This *community context pane* displays those communities which the user has successfully joined. In general, this pane allows the user to view those parts of the community hierarchy which she or he is a member of as well as those communities which he or she is not a member of – but not their sub-communities. The *managed communities pane* shows the community hierarchies of all those communities that are managed by the system which the user has permission to view according to the policies of the communities that the user is a member of. For a user to view a community, there must be some community that the user is a member of which possesses authority to view the community in question and there must be a community policy defined for that community which permits its members to view that community. In the case of this example, the *offshoot.org* community is owned by the *test community*, while the *test community* is self-managing in that it owns the resource that represents itself. A policy is defined in the *test community* which allows all members of the community to view all its CBPMS resources, hence the entire hierarchies of both organisations can be viewed.

4.7.3 Community Management Screen

When a user selects a community to manage, they are brought to the basic community management screen, shown in figure 4.13. This screen is the portal for all the CBPMS community actions and is divided into 3 areas. At the top is the *community connections pane*, beneath it is the *resource management pane*, and finally there is the *policy management pane*.

Community Connections

Root Community

Children		
Name	Manage	Delete
IT Department		
Marketing Department		
Accounts Department		

Create New Child Community

Federations	
Name	Leave
No federal memberships	

Join Federal Community

Resource Management

Resource Tree

Community test community

Presence Management (CPMS)

Community Management System

Door Access Management

Policy Management

Membership Rules	Rules: 2	Obligations: 0
Community Policies	Policies: 4	
Mandates From Parent	Policies: 0	
Mandates Set For Children	Policies: 1	

Figure 4.13 CBPMS Community Management System – Community Management Screen

The community connections pane shows the other communities that this community is related to in the hierarchy, its parent, children and federal connections. From this pane, the user can choose to delete any child communities, create new children or join federations. These map directly to the spawn, cull and federate CBPMS primitives. The resource management pane displays a resource tree with entries for each resource that the community possesses some authority over. Selecting one of these resources brings the user to the management screen for that particular resource. The resource management sub-system is responsible for displaying the hierarchical model of the resource in such a manner as to allow the user to select a valid resource authority in a convenient way. Thus, when a user selects a resource in the community’s resource tree, the system needs to provide a visualisation of the resource authorities contained within that resource. In fact, rather than displaying a visualisation of the entire set of resource authorities in this pane, only that subset of resource authorities that the community possesses authority over is displayed – since these are the only resource authorities that the community can use in any way, either in delegating authority or specifying policies.

However, due to the flexible nature of the concept of a hierarchy of authority within a resource, it is not possible to provide a single consistent visualisation of a hierarchy of authority for all resources. Although all authority trees can be represented as a hierarchical tree, many of these trees will have an infinite number of nodes – for example if the real numbers are being used as a partial order of an authority tree. What can be done, however, is to provide a general purpose framework and a number of generic visualisation classes which allow new resource visualisation types to be easily incorporated into the interface. The basic rule is that the user must first choose a node on the resource’s target tree and, then select a node on the resource’s action tree. Once these nodes have been selected, the user is invited to create a new policy with this resource authority, or to delegate the resource authority to a child community.

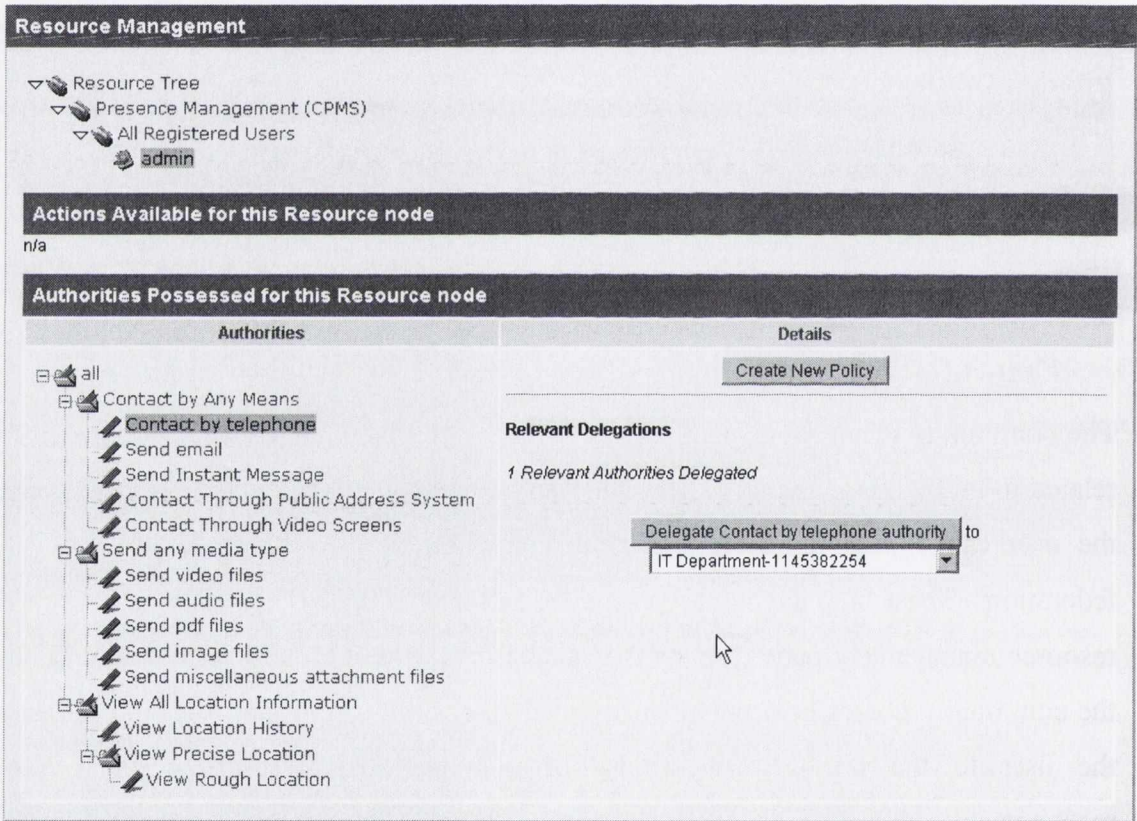


Figure 4.14 CBPMS Community Management System – Resource Authority Trees

Figure 4.14 shows a resource model for a presence management system. The tree of targets is situated at the top of the screen and here represents the tree of users in the system, the action tree, beneath it, represents actions other users can take with respect to the presence information of the target user. Both of these hierarchical trees are constructed by the standard hierarchical resource representation class, but this class can be easily extended in order to provide other visualisations within the respective areas of

the screen. It should be noted that this implementation of the tree visualisation does not show the entire target tree. In order to minimise network bandwidth, only those nodes that are directly beneath the selected node are expanded, all others only show the path to the selected node. The resource management screen allows users to delegate authority to sub-communities. This screen also provides a link to a page where all delegated authorities which relate to a particular authority are listed – allowing delegations to be recalled from sub-communities. These functions map directly to the delegate and recall CBPMS primitives. Thus, the resource management sub-system takes care of the distribution of authority throughout the organisation.

4.7.5 Policy Management

The policy management sub-system, shown in figure 4.15, allows users to specify community policies as well as the community’s membership policy.

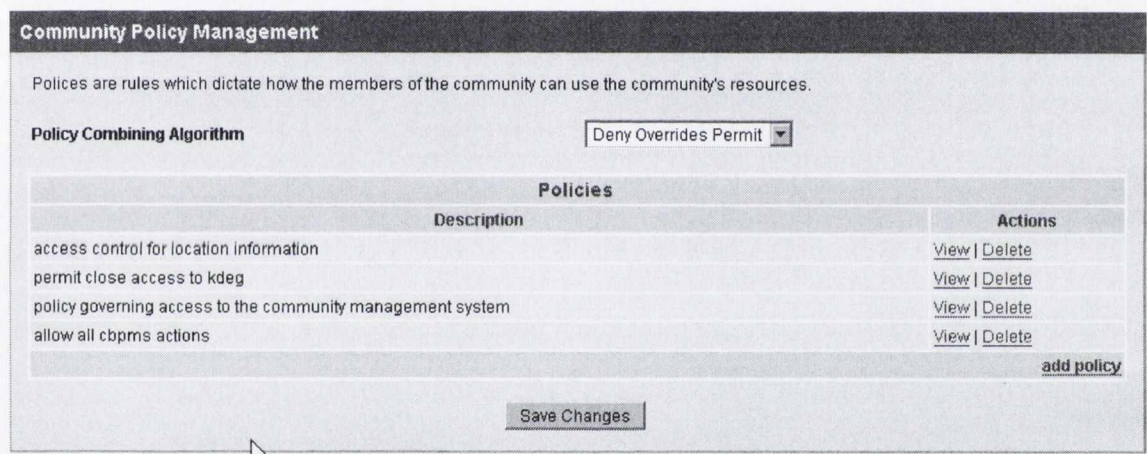


Figure 4.15 CBPMS Policy Management Screen

In addition to showing descriptions of the policies that the user is permitted to view, this screen provides links to various actions that users can take on the community policy set. The user can delete any of the policies – which is mapped directly to the *revoke* CBPMS primitive, they can view the contents of an existing policy or they can add a new policy. Finally a user can modify the policy combining algorithm for the community policy set. If the user selects a policy to view, or decides to add a policy, they are brought to the policy authoring screen. The precise form of this page depends on the policy specification language being used.

policy, a node on both the action tree on the right and the target tree on the left of the target pane must be selected.

The conditions evaluated and the decisions returned by XACML policies are defined in XACML rules within the policies and an XACML policy can contain multiple rules. This is different than most concepts of policies as policies are often assumed to be rules themselves. However, that does not present a real problem. A rule pane is provided at the bottom of the policy authoring screen which allows users to create new rules, edit or delete existing ones and change the rule combining algorithm employed by the policy. The rule authoring screen is very similar to the policy authoring screen. One difference is that the scope of the resource authority for the rule must be contained by the scope of the resource authority for the policy within which the rule is located. This allows the optimisation of the policy search algorithm – if the resource authority of a policy is not in scope for a particular request, then the resource authorities for the rules within the policy do not have to be evaluated, as they will also be out of scope by definition.

Another difference between the policy authoring screen and the rule authoring screen is that a rule is defined in XACML as having an effect, either “Permit” or “Deny” rather than having a combining algorithm. By changing this value, whether the rule is a negative authorisation (deny) rule, or a positive authorisation (permit) rule is chosen. Finally, there is a rule condition. In XACML, rules can have an optional condition. Rules without conditions will always return the rule effect whenever they are evaluated. By adding a condition to a rule, any arbitrary algorithmic condition can be added to a rule – if the condition evaluates to true, the rule returns its effect when evaluated, if the condition evaluates to false, the rule is considered to be *not applicable*.

The CBPMS system provides two different types of condition authoring screen, each of which supports a different degree of flexibility. A screen which produces XACML specified policies, using a pre-defined set of variables within the four basic maps in the XACML model, is provided. A screen which allows the user to input conditions in PHP, which will be embedded directly into the XACML, is also provided. By supporting PHP conditions, using a standardised mapping to the XACML data maps, policy conditions can be specified much more rapidly and less verbosely than XACML allows and the full flexibility of a programming language, with all of its libraries, is at the user’s disposal in authoring conditions. In terms of building a policy specification language which can be reasoned about, this is a bad choice, since the flexibility of a

programming language greatly increases the complexity of policy analysis. However, while developing an organisational paradigm for PBM, it is much more useful to provide a rapid development environment to allow the flexible testing of the framework. Hence, such a facility has been included, as shown in figure 4.17.

Community Policy Management - View Policy - View Rule - View Condition

Edit the condition below. When a relevant access attempt causes the rule containing this condition to be triggered, this condition will be evaluated. If it does not evaluate to true, the rule will be ignored

You can specify a condition in PHP in the box below. If you want to use attributes relating to the subject making the request, the action that the subject is requesting access for, the target resource that the request is being made to, or the environment from which the request is being made, you can use the special variables below

- TARGET(X) where X is one of the following: resource, target_path
- ACTION(X) where X is one of the following: id
- SUBJECT(X) where X is one of the following: id, location
- ENVIRONMENT(X) where X is one of the following: time, date

Example Condition

SUBJECT(id) == 'john' or SUBJECT(id) == 'jane'

This condition would mean that this rule only applies if the subject's id is john or jane

Expression

SUBJECT(id) == 'chekov'

Save Changes

Figure 4.17 CBPMS XACML / PHP Condition Definition Screen

Mandates & membership policies

The main CPUI community management screen also allows users to view and modify the community membership policy. Since the community membership policy has been defined as a specification which returns either a *not-applicable* result or an *applicable* result, this is most easily translated into XACML as a condition. Hence, the membership policy authoring screen is exactly the same as the condition authoring screen. If the condition evaluates as true when passed a user's context, the user is deemed a member of the community, otherwise they are not considered to be a member. The CPUI community management screen also allows users to view and modify mandates that have been set for this community, and those that this community has defined for its sub-communities. From the previous chapter, recall that mandates are simply policy rules which are defined for a sub-community by its parent and are then 'fixed' by the parent by means of a policy rule in its own community policy set. The set of mandated policies is managed using exactly the same screens as the set of community policies, the only difference being that the CPMS maps user actions on mandated policies to the sequence of actions required to set a mandate, rather than the simple primitive required to update, delete and create policies in the community policy set.

4.7.6 Community Structure Visualisation

The three basic sub-systems which allow the management of the community based system have been described. The ability to spawn, cull, federate and withdraw is provided through the connections management sub-system. The resource management sub-system allows users to delegate and recall and policies can be created, deleted and updated through the policy management system. To understand how the CPMS manages access to these capabilities, briefly consider how the CBPMS resource is modelled and how it relates to access control.

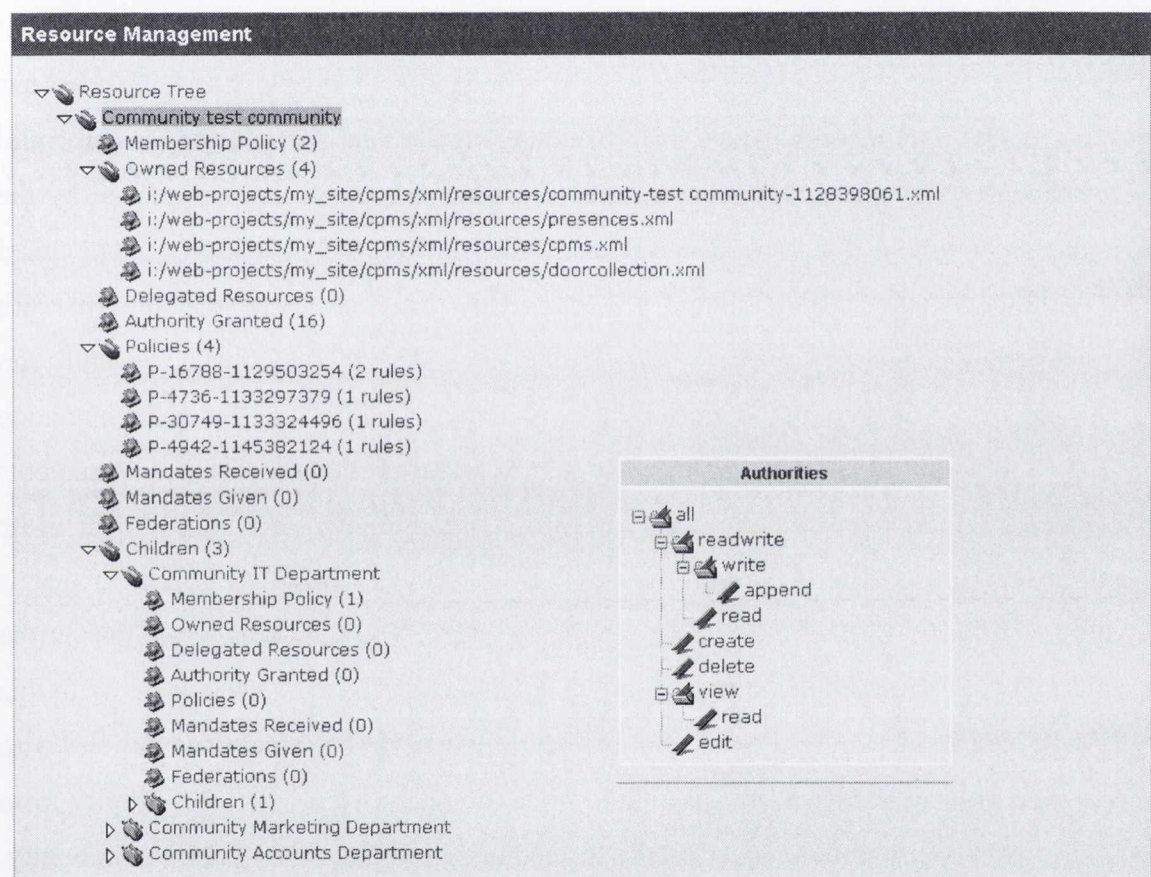


Figure 4.18 CBPMS Community Management System – Community Resource Model

As discussed in section 3.5, the CBPMS is a resource itself. In fact, this application models it as a set of resources. Each root community in the system is a resource, while there is a separate resource which represents those CBPMS actions which no community is the subject of, namely the *genesis*, *expel* and *grant* primitives. This allows these global-operator-driven CBPMS functions to be distributed separately to the community-driven functions and allows self-management authority to be delegated to a community without also delegating any authority to take actions which affect the global state of the system, outside the boundaries of the organisation to which the community

belongs. Figure 4.18 shows a portion of the resource authority tree for the community resource for *test community*, as seen by that self-managing community. Note that the target tree is very detailed and allows the selection of resource authorities which are particular policies. In fact, taking advantage of XACML's format, these nodes can be expanded further and policies can be attached to particular rules within policies. Also note that the actions are relatively low-level. This is a reflection of the fine-grained nature of the target tree. Rather than having an action for "update membership policy" and another for "update policy" this difference is expressed through the target reference and the same "edit" action is used in both cases. Not all possible combinations of actions and target nodes are valid and this application indicates when the user has selected a non-valid combination – for example only the read and view actions are valid on mandates that the community has received since they can only be updated by the parent community. Otherwise the resource authorities are interpreted in a hierarchical cascading manner. So, for example, if a community possesses a resource authority with target node = *community test community* and action node = *all*, then that community can take any action on any component of the community as well as any action on any node of the community's children. If the community has a resource authority with target = *Authority Granted* and action = *create* then the community is allowed to delegate authority to its children. This illustrates how a resource model can be used that is entirely different to the underlying resource structure. The only real constraint on the model of the community resource is that it should be intelligible for the user. With this fine-grained and low-level resource model authority can be distributed and policies with very fine-grained scope can be specified. For example, after delegating authority from the *test community* to the *IT community* for updating a single policy in that community, the view of the IT community of the community resource is shown in figure 4.19.

Although the nodes in the resource target tree directly above the delegated authority are visible, they are greyed out and the IT department is limited to creating resource authorities for the single policy that it has been delegated. Similarly, the action tree for this community consists of a single node – the edit action which is the only action that this community has been delegated authority for. By delegating this authority, the test community has granted a very limited degree of self-management to the IT department, as the IT department can now specify policies which would permit the community's members to update this particular policy (or the rules within it).

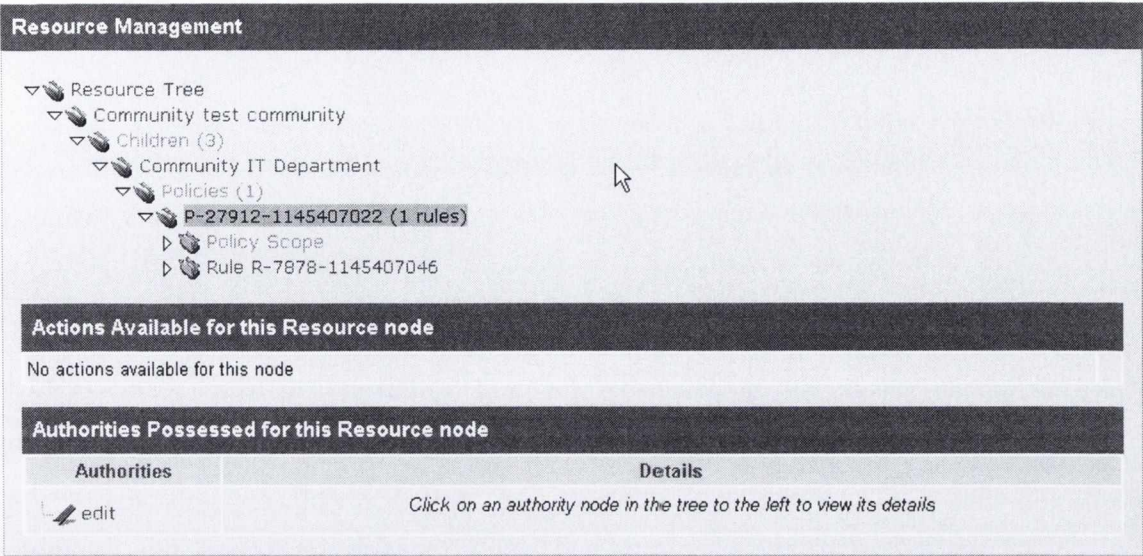


Figure 4.19 CBPMS Community Management System – Limited Delegation of Self-Management

4.8 Summary

This chapter has described the architecture and implementation of the CBPMS. An abstract service oriented architecture for a policy based system which operates according to the semantics developed in the previous chapter has been described. The service interfaces of the components which make up the system have also been defined along with their various interactions with each other. No characteristics of the policy specification languages have been assumed beyond those basic capabilities which are a requirement of the CBPM model. An extensible web-based implementation of this architecture, using the Java and PHP programming languages and an XML-RPC protocol for the provision of web-services, has been described. The implementation of a management system which allows users to conveniently use the system to structure their organisational model dynamically, distribute authority over resources throughout the organisation and define policies for those resources which they have authority over, has also been presented. The following chapter describes several case-studies and experiments that were carried out using this system in order to evaluate whether it can meet the goals of this research.

Chapter 5 Case Studies, Experiments and Results

5.1 Introduction

This chapter sets out to validate the basic goals of this thesis, defined in chapter 1. The development of the CBPM model has been described in detail in chapter 3, while its implementation has been described in chapter 4. It remains, however, to be demonstrated that this software system can be easily integrated into an organisation's IT systems to deliver a practical PBM capability. The first experiment, E1 described in section 5.2, applied the CBPMS to the management of a ubiquitous computing environment and was designed primarily to evaluate to what extent the system had achieved the first goal of this research and could be viably integrated into a complex management architecture.

Evaluation of the second goal required a demonstration of the CBPMS's ability to model a wide range of organisational forms, incorporating decentralised and fluid decision making structures, and its capacity at efficiently capturing organisational change. The final goal was to demonstrate that there are some practical advantages to using the CBPM model when compared to current role-based models, in particular that it reduces the number of policies that the system must evaluate for any context (goal 3a) and reduces the frequency of conflicts (goal 3b). Two further experiments (E2 & E3) and one extensive and detailed case-study (C1), were carried out in order to validate these remaining goals.

The case study, C1, described in section 5.2, uses the CBPMS to model a political system and shows how the basic institutions of a state (in this case the Republic of Ireland) can be developed through it in an evolutionary fashion. Starting from a single, undifferentiated polity, where everybody votes on all decisions, the case study charts the gradual evolution of a constitutional model based upon the institutions of the Irish state and its relations with other entities. Using this constitutional model, several scenarios are modelled by the system. This case study illustrates the CBPMS's capacity to support both self-management and progressive evolution of the model from a very simple structure to a very complex one, without the need for extensive re-modelling and analysis and contributes to demonstrating the second research goal.

The second experiment, E2, described in section 5.3, is based upon the modelling of a real world Internet community. The community's organisation and operations are

observed over time and modelled within the CBPMS. The CBPMS is then integrated into the community's collaborative software management system in order to allow the community's operations to be encoded as a CBPM configuration. The CBPM model is then evaluated by the community's administrators and is used to track changes to the community's structures over a period of a year. This experiment was designed to evaluate the capacity of the CBPMS to flexibly and robustly model a dynamic organisation and decentralised administration (goal 2) and provide some firm evidence that the system could model an evolving and decentralised real-world organisation, with results verified by the membership.

The third and final experiment, E3, described in section 5.4, concentrates on comparing various aspects of the organisational model embodied in the CBPMS with a role-based model of an organisation through a series of simulations. The goals of this comparison are two-fold. Firstly, it shows that the number of policy rules that must be evaluated for any given context is consistently less by an order of magnitude when the CBPM is applied (goal 3a). Secondly, it demonstrates that the frequency of conflicts is much smaller when the CBPM model is employed (goal 3b). The overall comparison serves to highlight the essential differences between the two approaches and illustrates that elements of the organisation's model which are not captured within the role-based model, can be easily incorporated into the CPBM system.

5.2 Experiment 1: Physical Security Management in a Ubiquitous Computing Environment

5.2.1 Scenario

The first experiment involved utilising the software implementation of the CBPMS with the XACML policy language, as described in chapter 4, to specify access control policies to manage the physical security of a simulated environment and to manage access to the presence information of users within that environment. This experiment was primarily designed to test the viability of the CBPMS implementation in practice in a demanding and complex application environment, specifically the aim was to show that the system would display the following characteristics:

- That the CBPMS architecture would allow it to be reasonably easily integrated into a heterogeneous real-world network.
- That the CBPMS could effectively deliver policy decisions in a demanding, real-time environment, requiring sub-second response times.

- That the organisational model could be modified dynamically without any interruption to the application.

A second aim of this experiment was to verify that the CBPMS software system was an accurate implementation of the CBPM model and would faithfully apply its *decision* algorithm for resolving conflicts. Finally, the experiment compiled statistics regarding the number of policies that had to be evaluated by the CBPMS in order for it to reach a decision. This data was gathered, from this realistic simulation, to provide a comparative benchmark for the statistics that were later gathered from the final experiment, which used comparatively unrealistic structures and policies.

The experiment involves modelling the management of a building used as a research institute within a University, housing several teams of scientists working on different projects, some of which are of a confidential nature. The institute houses their research laboratories, their personal offices and common areas such as the lobby, stairways and canteen. The building is a ubiquitous computing environment, where the location of each person is tracked by sensors. Doors automatically open for those who have appropriate access rights and one's location is available on an instant messaging application. The task was to use the CBPMS to capture the requirements of a realistic scenario in this domain. The following were the basic requirements:

- There is a variety of legislation and various rules of the University which govern physical access to buildings and this must be guaranteed by the system. For example, fire doors must allow access in emergencies.
- All of the research groups working within the building consult in the formation of policy governing common areas.
- Each research group decides who is allowed access to their laboratories.
- Individual researchers decide who is allowed to access their personal offices, except in exceptional circumstances.
- Individual researchers will be allowed to control access to their personal location information, except in exceptional situations, as decided by University rules.
- All of the above is managed by authoring policies. It is assumed that all of the users of the institute are capable of authoring policies.

5.2.2 Experiment Environment

PUDECAS is a 3D simulation environment designed to test context-aware adaptive services in the wireless, mobile and context aware markets. It was chosen for this

experiment as it provides a realistic environment for testing ubiquitous computing systems without the expense of fitting an entire building with sensors and activators. It provides service developers with a toolkit for creating 3D simulations of physical spaces which allow events within the simulator to trigger calls to external, networked services. It is based upon the well-known Valve Source Engine from the popular Half-Life 2 computer game. The PUDECAS application framework has been introduced in detail in [quinn06]. For the purpose of this experiment, however, it suffices to know that the framework can be programmed to send messages to a service in response to events or actions by characters in the simulation. In this experiment, the physical environment is a simulation of the Lloyd Building in Trinity College Dublin as pictured in figure 5.1.

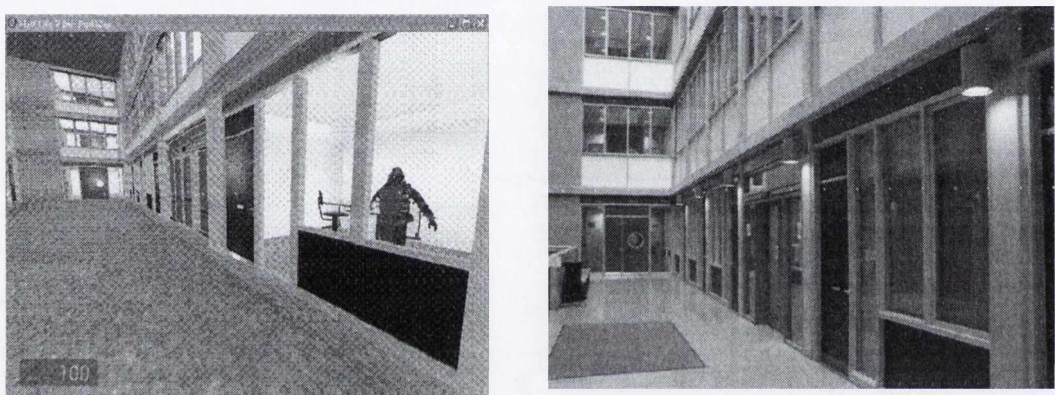


Figure 5.1 Screenshot from Simulation of Trinity’s Lloyd Building and actual photo

The environment was set up so that whenever a user approached a door, the PUDECAS environment would generate an event notice including the identity of the user and the id of the door that the user had approached. This event notice was sent, through the PUDECAS service interface, to a CBPMS proxy, acting as the policy decision consumer according to the CBPMS architecture described in section 4.1. These CBPMS event detection proxies are simple templated objects which translated network communication semantics from an application into the policy decision request format required by the CBPM Decision Service (CPDS) web-service interface. They translate network communications in a variety of protocols and application domains into policy decision requests and responses and can be configured to work in both synchronous and asynchronous modes of communication (see network architecture in figure 5.3). Due to the simplicity of the CPDS service interface, which consists of a single decision function, the event detection in this experiment required little more than a protocol translation. Other application domains may require more complicated event detection which maintains state.

Whenever the CBPMS proxy received this event notification, it formulated a policy decision request which it forwarded to the CPDS. The policy decision request was implemented as a simple name-value array, describing the user's context; it was populated in accordance with the schema shown in table 5.1. The CPDS returned a policy decision according to the XACML language, either *Permit*, *Deny*, *Indeterminate*, or *NotApplicable*. If the policy decision was *Permit* then the CBPMS proxy sent the *door_id* to the PUDECAS simulator, causing the simulation engine to open the door, otherwise no message was returned and the door was not opened. The Pudecas simulator also sent a simple periodic message to the CBPMS proxy every second for each user who was currently active in the simulator, indicating their location. These location messages were not propagated beyond the Context Mapping and Discovery Service (CMDS) which, in this case, was configured to provide session management and recording services for the application. This session management feature meant that policy decision requests could have various attributes added to them within the CPDS, including the location history of the user – allowing policy rules to be authored which permitted or denied access depending on the current and past locations of the user – enabling rules to specify that a user is only permitted to open a door from one direction, or, for example, cannot open a door if they have recently been in a particular location.

Name	Value	Name	Value
USER	[user_id]	USER	[user_id]
COMMUNITY	lloyd_users	COMMUNITY	lloyd_users
TARGET	[door_id]	TARGET	[user2_id]
ACTION	open door	ACTION	view location

Table 5.1 Two Examples of Policy Decision Requests Generated by the CBPMS in response to Simulator Events. The example on the right is generated when a user approaches a door, the example on the left when a user attempts to view the location of another user

The second element of the experiment involved an Instant Messaging (IM) application. The IM client has a roster of buddies, grouped by the user of the client into lists e.g. friends, work colleagues and family and allows each user to request presence information of all the other users of the simulator – in particular location information and ‘buddy-list’ information.

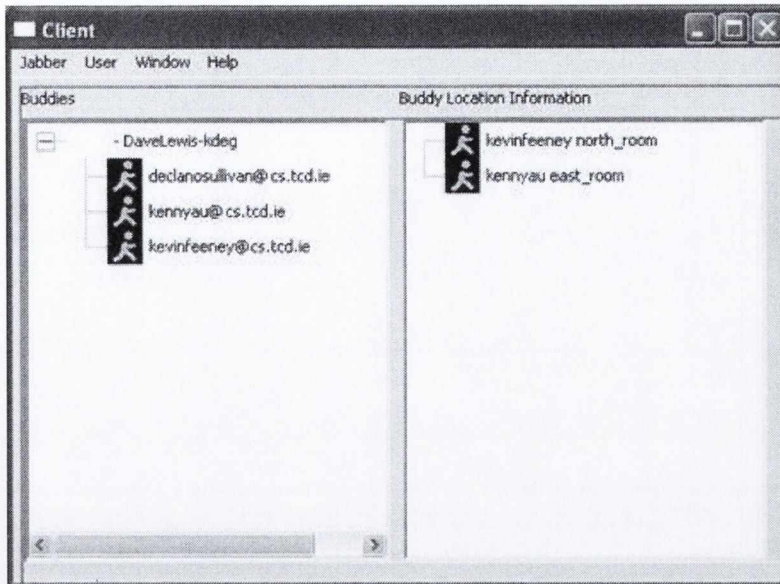


Figure 5.2 Screenshot of IM Application

In this experiment, whenever a user requested such information, a CBPMS Event Detection Proxy subscribed to these messages and responded by instigating a policy decision request to the CBPMS. If the requesting user was permitted to access the location information of the target user, the IM application displayed the location information of the target user within the application pane as shown in figure 5.2. The CBPMS Event Detection Proxy also returned a list of all users who should be in the user's buddy list. This list was compiled by creating a simple database-interrogating CMDS service class which iterated through all the users in the JABBER user-database and formulated policy decision requests for each user to see if they were members of the relevant community. The full architecture employed for this experiment is pictured in figure 5.3. It was evaluated to see whether it could support multi-user experiments simulating policy-managed ubiquitous computing environments.

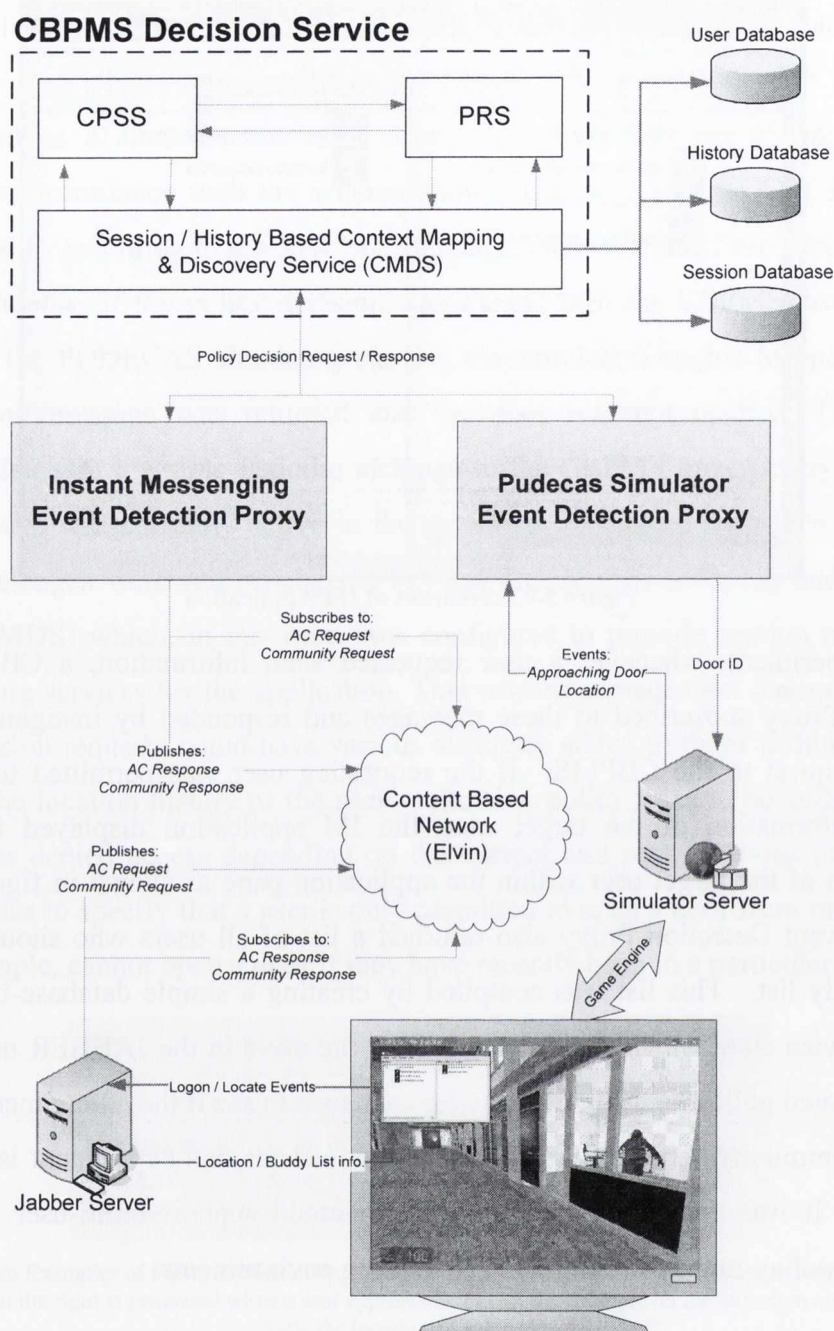


Figure 5.3 Architecture for CBPMS integration experiment

5.2.3 Creating the CBPMS Models

In this experiment, access to all of the doors, the population of people's buddy lists, and access to their location information is to be managed by the CBPMS. The first task is to create suitable models of these resources. This is a straightforward task as the resources are extremely simple. The doors in the building are modelled as a target tree consisting of a simple hierarchical tree of doors, gathered together into convenient grouping nodes, to allow policies to be specified for groups of doors more easily, for example, by room

and by floor. The action tree for this resource consists of a single node – *open door* – since this is the only action that can be performed by users on the resource; the simulator is programmed to automatically close doors after the user passes through them. The doors are modelled as a one-dimensional tree, where each node represents a door, or collection of doors. The only other resource that is to be managed by the CBPMS is the location information of each user. Once again, this is very simple as there is only a single action that users can take vis-à-vis another user – to view their location. Rather than building a static representation of the users in the system, however, the resource model merely provided an interface into the user database utilised by the IM interlocutor. This option was taken in order to allow the system to be easier to maintain, since users only had to be registered in a single, system-wide database from which the resource model was dynamically constructed. However, in order to provide support for new ubiquitous features and future modifications to the simulator software, the action trees of both resource models were extended to include various actions which, while not being supported by the current software, might be supported by future ubiquitous services using the simulator. Thus the resource models constructed, pictured in figure 5.4, extended the action trees to include various extra nodes.

Having constructed the resource models, a basic community model was designed, based on the organisational structure of the building's management, as derived from the requirements in section 5.2.1 above. The model is pictured in figure 5.5 and is loosely based upon a cross-section of those groups within the department of computer science in Trinity College Dublin who currently use the building. It was created by recursively breaking down the users of the building into more specific and smaller units.

Resource Management

Resource Tree

Door Access Management

All Doors

First Floor

KDEG Lab

Door To Corridor

Door 2 To Corridor

Door To Meeting Room

Door To Small Office

Actions Available for this Resource node

n/a

Authorities Possessed for this Resource node

Authorities	Details
all open door close door	Click on an authority node in the tree to the left to view its details

Resource Management

Resource Tree

Presence Management (CPMS)

All Registered Users

admin

k.arl.quinn@cs.tcd.ie

dave.lewis@cs.tcd.ie

vinnie.wade@cs.tcd.ie

declan.osullivan@cs.tcd.ie

elleanor@cs.tcd.ie

austin@cs.tcd.ie

kevin.foley@cs.tcd.ie

kevin.carey@cs.tcd.ie

Actions Available for this Resource node

n/a

Authorities Possessed for this Resource node

Authorities	Details
all Contact by Any Means Contact by telephone Send email Send Instant Message Contact Through Public Address System Contact Through Video Screens Send any media type Send video files Send audio files Send pdf files Send image files Send miscellaneous attachment files View All Location Information View Location History View Precise Location View Rough Location	Create New Policy Relevant Delegations No Relevant Authority Delegated Delegate View Precise Location authority to IT Department-1145382254

Figure 5.4 Resource Models Employed (screenshots from CBPMS GUI)

Having created the initial basic model of the communities, a sequence of CBPMS primitives are invoked in order to seed the system to reflect the organisation’s requirements. Firstly, ultimate authority for the resources representing the building’s doors, the presence information of users and the community structure itself are granted

153

to the root community. Then, this authority was delegated to the *staff* community and from there to the management community, who thus act as a control community for both the *staff* community and the root community – they decide global policy. In order to cede self-management to the various research groups, authority for their communities was delegated from the root community to the staff community and from there to the academic community and from there to each of the research groups and projects. This self-management allows them to specify themselves the policies that apply within the groups to the resources that are delegated to them and allows them to change their structure as they deem fit. Having distributed authority for the resource representing the community structure to the various communities, authority for the other resources, representing the buildings doors and the presence information of users was distributed in a similar fashion. For example, authority to author policies about the doors leading to lifts and common areas was delegated to the *academics* community, while authority for the doors leading to laboratories was further delegated to the research groups and the projects that used them.

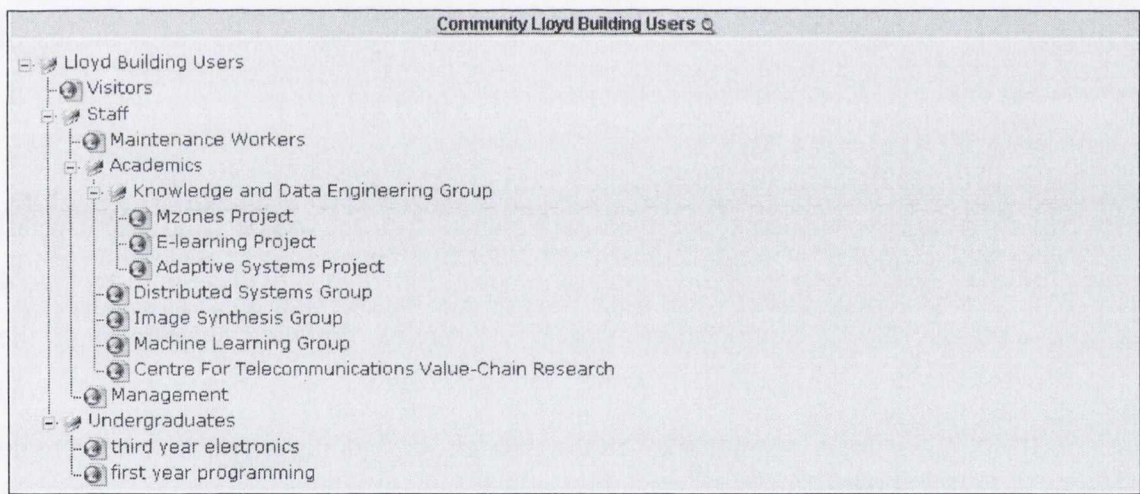


Figure 5.5 Segment of Community Model for Integration Experiment

Next, having delegated the appropriate authority to the various groups, policies were put in place to enforce the management requirements. The management community specified policies in the root community which implemented the various legal codes and building regulations governing access. Thus, for example, a policy was defined for all of the access doors to the building specifying that access should be permitted during emergencies. The *staff* community defined a policy which specified that all office-holders could specify their own policies about their personal office doors (this had to be specified in the *staff* community so that it would be scoped to apply to all staff). The

various research groups specified policies which should apply to the doors that were delegated to them. The *management* community defined a policy specifying that the location information of all the building users would be available to management when they were in the public areas of the building (the lobby, the canteen). The *academics* community defined a policy which specified that each member had permission to specify policies about who could access their personal presence information. All of these policies were specified by attaching a resource authority to a simple, unconditional 'permit' or 'deny' policy.

In total approximately 200 policy rules were specified in the system to enforce the requirements. It should be noted that the various communities in the model did not specify their own policies. The self-management element of the model was tested by the designers of the experiment, by logging into the CBPMS GUI and specifying policies in the guise of a member of the various communities. The experiment did not attempt to show that groups of non-expert users could use the CBPMS to autonomously manage the simulated environment, just that the CBPMS would have allowed such groups to do so had the users been capable of authoring policy-rules as a means of validating the CBPMS implementation.

5.2.4 Running the Experiment

Several different experiments were carried out to test the model and the overall integration of the CBPMS with the simulator environment. Each experiment involved 4 users, none of whom were familiar with PBM, navigating their way through the simulated environment, using the user-interface pictured in figure 5.3. Each user was assigned to a set of communities in the model, by adding them to the simple membership rules of the appropriate communities and these community memberships were allocated so that each user shared membership of a research project with another user. In each iteration of the experiment user's were assigned various tasks, such as attempting to gain access to a room or a corridor in the building, or attempting to track each other's locations as they moved through the environment.

The first run of the experiment was used to familiarise the experimental subjects with the system, and to fine-tune the policy set and the software implementation. The success or failure (e.g. because they could not enter a door) of the users in their tasks in each of their allocated tasks was observed and compared to the results expected. Metrics were also gathered from the experiment, as well as from subsequent

experiments, which recorded the number of policy rules that had to be evaluated for each policy decision request and the time that it took to return policy decisions. The second run of the experiment involved testing the modifications that had been made in the previous run and introducing new tasks to check for any undiscovered specification problems. The third run of the experiment involved modifying the community set to purposely introduce several conflicting policies into the system and setting users tasks designed to trigger these conflicts in order to test the CBPMS's ability to automatically resolve conflicting policies as well as to highlight the organisational source of irresolvable conflicts. The fourth and final run of the experiment involved testing the systems ability to respond in real time to modifications in the policy set and community models. As the users navigated through the environment, policy rules were modified in real time, sub-communities were created and destroyed, and membership rules were changed and the observed behaviours were compared with the expected behaviours.

5.2.5 Results

The first two experiments revealed several areas where both the software integration and the policy set needed to be refined. The most serious problem identified was the fact that, as the experiment progressed, the latency between a user approaching a door and the door opening grew larger. The underlying problem was that the time between the simulator engine sending a request to the CPDS and it receiving a policy decision in response grew as the simulation progressed. After 10 minutes of running the simulation, the lag had increased to as much as 30 seconds, which made the application difficult to use since it became difficult for users to correlate their approaching of doors with their eventual opening as they tended to move quickly around the environment. By examining the CBPMS logs, the problem was diagnosed and the following causes were identified:

- The simulator engine generated multiple requests on most occasions when users approached a door. This was due to the fact that the users tended to run around the environment, stepping quickly in and out of sensor areas and trying to enter multiple doors in quick succession.
- When four users were simultaneously generating a large volume of decision requests in quick succession, the CPDS was sometimes receiving over a hundred requests per second. At a certain rate of traffic, estimated to be roughly 20

requests per second, the CPDS became saturated and its performance dropped off massively. While the normal time for generating and sending a response was approximately 0.005 seconds, the logs showed that this would sometimes suddenly rise to a value between 1.2 and 1.5 seconds.

- When the lag between a user approaching a door and the door opening became several seconds, users tended to respond by re-approaching the doors several times while waiting for them to be opened, or approaching a series of doors. Thus, at times when the CPDS was saturated, the traffic tended to increase.
- Although the time taken by the CPDS to generate and send a response never became greater than 1.5 seconds, the fact that the server had become saturated caused many messages to become queued on the network layer for substantial periods of time. The assumption adopted was that the server's memory had become saturated during these lags. This theory was never fully tested, but all the behaviour observed tended to support it.
- When several identical requests were sent to the CPDS, it responded to each of them and the simulator engine would repeatedly open and close the same door, even when all of the responses were received as the door was already opening.

This problem was resolved in two ways. Firstly the polling frequency of the simulator was decreased, meaning that the simulator would send requests to the CPDS at less frequent intervals. Secondly the CBPMS proxy was modified to make it discard any requests which arrived within two seconds of an identical requests. Other than this problem, the software integration was deemed successful. Doors in the building automatically opened and people's location was visible in accordance with the policies specified. Once the solution was implemented, the problem of a significant lag between the simulator events and the simulator receiving a response disappeared as the maximum rate of requests dropped to less than 5 per second in normal usage.

The policy set proved generally robust; however, some small problems were identified when doors did not open as they were expected to. 3 policy rules had to be removed and 15 policy rules added to the policy set in order to correct these specification problems. The problems generally involved forgetting to specify rules for certain doors, or specifying a rule in a community that was too far up the tree which caused conflicting policies, defined further down the tree to be ignored. For example, in one case, a deny

policy was specified for a door, but a permit policy applied to the entire floor in a parent community, meaning that the deny policy was never invoked – a problem solved by moving the deny policy up one level in the community hierarchy.

During the experiments, several of the tasks were designed to validate the fact that the CBPMS system served as an accurate implementation of the CBPM decision algorithm, in particular that it would react correctly when it encountered conflicts. Table 5.2 details the conflict-scenarios that were tested, how they were tested and the observed results.

Conflict	Test	Observed Result
Conflict Resolution within a community	<i>A policy was specified in the root community denying access to the fire doors and the stairways in all circumstances, in conflict with the policy already defined specifying that access was allowed in an emergency.</i>	<i>When the administrator defined that an emergency was occurring, the fire doors opened according to the algorithm defined in the XACML policy set specification. Changing this algorithm from 'denyOverrides' to 'permitOverrides' correctly changed the door's behaviour.</i>
Conflict Resolution Based Upon Hierarchy of Authority	<i>A Policy was specified for the staff community denying access to the fire doors in all circumstances, as above in conflict with the policy defined in the root community.</i>	<i>When the administrator defined that an emergency was occurring, the fire doors opened when staff approached.</i>
Conflict Resolution Between non-Hierarchically related communities	<i>A Policy was defined for the KDEG community permitting access to a door and a policy was defined for the DSG community denying access to the same door. A user was assigned to both of these groups simultaneously.</i>	<i>The door opened according to the child combining algorithm specified in the academics community (the nearest common parent of the conflicting communities) which specified KDEG policies had higher precedence. Changing the relative priority of the two groups, changed the door's behaviour</i>
Non-Resolvable Conflicts	<i>The above test was repeated, but no child combining algorithm was specified for the academics community.</i>	<i>A conflict event was generated for the academic community informing it that authority for the door in question had been delegated to two groups who had overlapping memberships and that this would have to be resolved by either refining the delegations or by specifying a combining algorithm.</i>

Table 5.2 CBPMS Conflict-Resolution Tests and Results

The final run of the experiment involved an administrator updating the policies of the communities, changing the community structure and changing the membership rules of communities. These changes were tested by asking users to attempt to access doors, making changes and then asking them to attempt to access the doors again and

observing whether the behaviour of the doors changed to reflect the changes to the model. All of these changes were successful. The users were able to access doors according to the organisational model that existed in the instant that they approached the door and triggered the event. The opening of doors reflected the dynamic model of the organisation and how the user fitted into it. In no cases was it possible to identify any discrepancies between the organisational model as it changed and the expected behaviour observed, from careful analysis of the server logs.

However, although this experiment provided some evidence of the ability of the CBPMS to model organisations with evolving and flexible structures, it also revealed several shortcomings in the user interface with respect to such a dynamic environment. The relative complexity of the policy authoring user interface, in particular the composition of multiple rules into a single XACML policy, meant that several ‘clicks’ were required for each modification. The fact that the interface was web-based and was running on the same server as the rest of the CBPMS services caused each page to take up to 5 seconds to load. Since most changes involved several different actions (e.g. creating a new community, specifying a membership rule, specifying a policy, specifying policy rules, specifying a condition), each modification took up to a minute to carry out, and while this is quick in most domains, in the world of simulators based upon computer game engines, whose users are accustomed to highly responsive environments, one minute waiting for a change to be made before carrying out a task is an unacceptable length of time, evidenced by the restlessness of the users. Therefore, while the experiment showed that the CBPMS can indeed model rapidly changing dynamic organisational structures in real time, the web-based management interface prove insufficient for real time policy authoring in such a dynamic environment. However, since the experiment was not designed to test this aspect of the system, and due to the fact that it was at least partially due to the fact that the server in question was under serious load, no general conclusions could be drawn from this observation.

Finally, statistics were gathered from the third and fourth run of the experiment regarding the cost of the policy evaluations, both in terms of evaluations carried out and in terms of the time taken. The results of these measurements are presented in table 5.3. These will be referred to in section 5.5, where they will provide a benchmark for comparing the results of a more artificial simulation and evaluating whether those metrics are likely to deviate strongly depending on the context.

Number of Policy Rules	193-255 (depending on test)
Average Time to Generate Policy Decision	0.017 seconds
Average Community Membership Tests per Request	12.34
Average Rule Evaluations per Request	1.43
Average Authority Comparisons per Request	45.63

Table 5.3 CBPMS Metrics Collected From Experiment 1

5.3 Constitutional Modelling Case Study

As a case study of how the CBPMS can be used to dynamically model a wide range of organisational forms, it is useful to look at how the model could be applied to the problem of modelling the structures of human political states. The CBPMS approach aims to provide a general purpose approach to the modelling of organisations and their management of resources and thus such a case, which represents an organisation with a very broad form of participation, presents a good challenge. In particular, the fact that a state’s structure and rules are generally formally defined in a constitution and legislative statute book provides the opportunity to compare the community models with the legal forms of modern states. Therefore, the problem of modelling a state’s legal structures was chosen as a good case study to showcase the ability of the CBPMS to capture complex and dynamic organisational forms in an efficient and accurate way.

Rather than simply choosing a constitutional model and constructing it as a community structure, it is more informative to build it up from a more simple organisational form through the invocation of CBPMS primitives – highlighting the potential of the model for dealing with dynamic and evolving organisations. Thus, the starting point is chosen as something akin to a Hobbesian “state of nature” and the end point is *Bunreacht na h'Eireann* – the constitution of the Republic Ireland. This case study illustrates how a polity could gradually construct a state infrastructure in small steps, starting from a very simple initial state. Initially, there is a single community, called *citizens*. This community manages two resources, namely the community resource itself and a voting system, whose resource model is shown in figure 5.6. By delegating resource authorities – pairs of nodes from these trees - to communities and specifying policies for these resource authorities within the relevant communities, authority for which communities can vote in which elections / referendums is distributed throughout the system. It should be noted that this case study uses a very simple model of a voting resource – which supports representative elections and direct referendums but does not

discriminate between the wide variety of ways in which elections and referendums can be decided, for a discussion of voting systems, see [ferscha99].

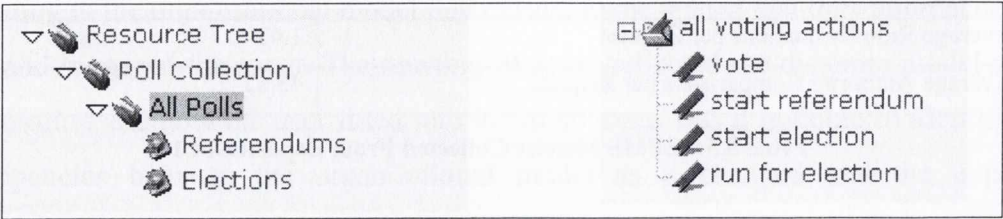


Figure 5.6 Voting Resource Authority Tree Model

In addition to the voting resource and the community resource, each of the citizens is also modelled as a resource (as a state claims the right to do things to a citizen against their will such as to imprison them). From considering the semantics of the community resource and how it maps to the political structure, it is immediately obvious that the policies of the *citizens* community are the state’s legislation. A policy language is selected which allows for the specification of obligations: what a citizen must do in certain situations; and for authorisations: what they may do. These specify the rights and duties of citizens within the state. It should be recalled that each policy, in the CBPMS, is associated with a resource authority. Thus, in order to specify a policy in this case study, a policy type (permit, deny, oblige), a resource authority (surrounded by square brackets in the following) and an optional condition are required. For example, the law that requires all citizens to vote in elections could be specified as follows:

```
Policy.compulsoryVotes = oblige [voting resource, all polls, vote]
```

Whereas the law that permits all citizens to enter elections can be specified as:

```
Policy.Candidature = permit [voting resource, elections, run for election]
```

In order to enter these policies into the legislation governing the citizens community, the policy primitive is invoked for that community:

```
policy(citizens, permit [voting resource, elections, run for election])
```

In order to provide this abstract case study with some grounding in concrete terms, in the following discussion, each step in the narrative is accompanied by the concrete primitive invocations required to model the required changes within the CBPMS.

5.3.1 Direct Democracy

To start from a very simple base, the incipient state is first configured as a centralised direct democracy, similar to the political system of many Greek city states, such as Athens, in the Hellenic era [rhodes04]. This form of political organisation is

characterised by the fact that there is a single set of laws which apply to everybody in the system which is decided by a centralised referendum (hence centralised) and that each individual gets to vote directly on any modifications to the laws (hence direct democracy). In order to establish this type of organisation, the following sequence of primitives is invoked:

1. **genesis**(Citizens)
2. **grant**(voting resource, Citizens)
3. **grant**(community resource, Citizens)
4. **grant**(citizens resource, Citizens)
5. **Policy**(Citizens, permit[voting resource, all polls, all voting actions])
6. **Policy**(Citizens, permit[community resource, Citizens, all] if referendum.pass)

The first policy invocation in this sequence specifies that all citizens are permitted to take all voting actions on all polls. The second policy specifies that any citizen can invoke a CBPMS primitive (including enacting a change in the policies governing the community), only if that change has been passed by a referendum. Having carried out these 6 primitive invocations, the political organisation of the community is a direct democracy. Any policy can be changed by means of a referendum and any citizen can inaugurate a referendum.

5.3.2 Formation of the Executive and Judicial Branches

The direct democracy described above has several obvious problems. For a start, the laws which govern the affairs of humanity are rarely directly enforceable by electronic agents, or at least the lack of electronic judges and magistrates would suggest as much. Furthermore, citizens are wont to ignore the obligations which have been defined for them. Therefore, this proto-state needs a means of enforcement. Therefore, a sub-community is defined which is given the responsibility of enforcing the laws of the state and ensuring that citizens follow the law. This community is called the executive. It is delegated control over the citizens and, in particular, is given the power of arrest. This is accomplished simply by means of the following primitives:

7. **spawn**(Citizens, Executive)
8. **gatekeeper**(Executive, if elected.executive)
9. **delegate**(Citizens, [citizens resource, all citizens, all], Executive)

It is assumed that the voting system issues a certificate to those citizens who have been elected to the executive and that the CBPMS can check for the presence of this certificate in order to check the membership rule of this new community.

However, acknowledging the fact that there is a great deal of ambiguity in the specification of laws, the citizens are worried in case the *executive* should become too powerful – judge, jury and executioner. Thus, rather than specifying a policy which allows all members of the executive to utilise the power over the citizenry which they have been delegated, a second sub-community is created, whose purpose is to decide when a particular law applies. This community is known as the *judiciary*. In order to model its function properly, another new resource is introduced – the *litigation resource* which models the judicial actions that a citizen can take. The following primitives establish a situation where citizens are allowed to prosecute cases and members of the judiciary are allowed to adjudicate on them.

10. **grant**(Citizens, litigation resource)
11. **spawn**(Citizens, Judiciary)
12. **gatekeeper**(Judiciary, if elected.judiciary)
13. **delegate**(Citizens, [litigation resource, all cases, all actions], Judiciary)
14. **Policy**(Citizens, permit[litigation resource, all cases, prosecute])
15. **Policy**(Judiciary, permit[litigation resource, all cases, adjudicate])

Having established the judiciary and defined its workings, the policy which permits the executive to use its powers can be established

16. **Policy**(Executive, permit[citizens resource, all citizens, all actions] if judicial.decision)

This policy dictates that members of the executive are only allowed to take actions on citizens when this has been decided by the judiciary. This represents, in simple form, the concept of separation of powers that is common in constitutional law. Finally, in order to allow the two new communities to regulate their internal organisation without the need for micro-management by the entire judiciary, they are delegated self management authority within certain boundaries and policies are specified in order to allow them to utilise this authority.

17. **delegate**(Citizens, [community resource, Executive, all], Executive)
18. **policy**(Executive, permit[community resource, Executive, all] if executive.decision)
19. **delegate**(Citizens, [community resource, Judiciary, all], Judiciary)
20. **policy**(Executive, permit[community resource, Judiciary, all] if judicial.decision)

It should be noted that these autonomy policies allow the respective sub-communities to specify their own policies, but that these policies are limited to the resources that have been delegated to them. In the above case this would allow the executive to author a

policy for itself which granted itself extra powers over the citizenry, without judicial accountability. To ensure that this cannot be achieved, and to prevent any emergence of a tyrannical executive, a policy can be authored in the root community which explicitly forbids any citizen from taking an action on another citizen without judicial approval. Since this policy is specified higher up the hierarchy than the executive community, it will always have precedence and thus, no matter what policies are defined by the executive, they will never be permitted to take such actions without judicial oversight:

```
21. Policy(Citizens, deny[citizens resource, all citizens, all actions] unless  
judicial.decision)
```

It should be noted that, although all citizens are forbidden from taking action against another citizen without judicial approval, only members of the executive are permitted to do so with such approval.

5.3.3 The Constitution and Separation of Civil and Criminal Law

The judiciary finds itself dealing with two distinct sets of problems. One set of decisions relates to members of the executive prosecuting citizens for breaking the law so that they can have permission to take action against them. The other set of decisions concern adjudicating on the law in situations where two citizens disagree on its interpretation in a given context. In order to deal with these situations, the judiciary forms two sub-communities, one to enforce policies, one to rule on disputes and conflicts. These become the criminal and civil branches of the law and they evolve different decision making methods.

However, the proliferation of legislation, some of which has been agreed upon without sufficient reflection, means that the problem of conflicting laws is frequent and there is no way of resolving these conflicts in general. Therefore, in order to decide precedence, a subset of the laws which cover the most fundamental rules of the system are separated from the rest and these rules are defined within the root community as having a higher precedence than all other policies. If a policy contradicts any of these policies, it is overridden. These policies are the *constitution* of the emerging state. Henceforth, whenever a new policy is encoded for the root community, it is either entered into the constitution or the book of statutes – the term applied to those rules which are of less precedence but apply to all citizens.

5.3.4 Legislative Bodies

Rather than the citizens authoring their own policy, they decide to elect an expert group to author these policies on their behalf. The large volume of decisions which each individual must make is deemed to be better dealt with by a dedicated group. Therefore, the citizens decide to create a sub-community, called the *legislature* which shall have responsibility for the definition of the laws which will apply to the citizens. However, they decide that to give this group carte-blanche to amend the laws as they see fit would be a dangerous move. Therefore, they only allow the legislature to modify policies which are not part of the constitution.

```
22. spawn(Citizens, Legislature)
```

```
23. gatekeeper(Legislature, if elected.legislature)
```

```
24. delegate(Citizens, [community resource, policies, all actions],  
Legislature)
```

```
25. policy(Legislature, permit[community resource, policies, all actions])
```

In order to prevent the legislature from amending the constitution, a policy is entered into the constitution itself which specifies that the constitution can not be modified without a referendum.

```
26. policy(Citizens, deny[community resource, policies.constitution, all actions]  
unless referendum.decision)
```

Thus, the 3 distinctive branches of the state have evolved from the needs of the citizens. A judiciary, executive and legislature have been defined and their rights and responsibilities have been defined. However, the fact that authority for the management of all non-constitutional policies has been delegated to the legislature means that the citizens must be certain that the fundamental rules ensuring democracy have been adequately protected, since the scope of authority of the legislature is so wide and if any of the important rules have been defined outside of the constitution (which is protected by the *constitutional democracy* policy, primitive no 26 above) then the legislature has a path towards dominance.

5.3.5 Legislative Dominance

After a period of time, the elected members of the legislature come to the conclusion that the current policies on elections allow vexatious and frivolous individuals to call referendums and elections. In order to do this, they decide to revoke the existing elections policy and to replace it with a policy which only allows citizens to vote and run in elections. Since the original policy was part of the constitution changes to it

require a referendum. Once this referendum is passed, the power to call elections and to declare referendums is henceforth exclusively the preserve of the legislature.

```
27. revoke(Citizens, Policy.elections)
28. policy(Citizens, permit[voting resource, all polls, vote])
29. policy(Citizens, permit[voting resource, elections, run for election])
30. policy(Legislature, permit[voting resource, all polls, all voting actions] if
legislature.decision)
```

Furthermore, the legislature goes on to enact further reforms with the further goal of limiting the direct involvement of the population in decision making and abrogating most of this responsibility to the representatives in the legislature. The legislature thus amends the membership rules of the executive and the judiciary in order to make them appointed by the legislature rather than elected by the citizenry. This makes the legislature effectively the dominant decision making body in all aspects of the state's life. The original direct democracy has been transformed into representative democracy, where although power is vested in the citizenry, it is largely wielded by the elected legislators and the citizens' rights in terms of political decision making are limited to voting in elections and referendums and putting themselves forward for elections.

At this stage, the structure of the state has become a simple, but largely accurate, model of the structure of the Republic of Ireland, which can be verified by perusal of the state's constitution, which describes, for example, the exclusive right of the parliament to call referendums: *"Every proposal for an amendment of this Constitution shall be initiated in Dáil Éireann as a Bill, and shall upon having been passed or deemed to have been passed by both Houses of the Oireachtas, be submitted by Referendum to the decision of the people in accordance with the law for the time being in force relating to the Referendum."* (article 46.2 Bunreacht na h'Eireann). It should be noted that this is not a universal feature of states – other states allow referendums with a sufficient number of signatures in support to be put to the public – it is the particular configuration of the Irish state.

5.3.6 Regional and Local Organisation

In addition to the central institutions of the state, there exist a large number of regional and local decision making bodies which exist in order to make the burden of policy making less concentrated in the legislature and more decentralised. In order to model these bodies within the CBPMS, regional organisations are created and various resources are delegated to them. These regional organisations are designed in much the

same way as the state – with elected regional legislatures which can create local policies. However these local policies are constrained by national legislation and are also bounded by the resources delegated to them from above. They may also come into conflict with the executive who have also been delegated authority by the legislature. The legislature deals with these problems by establishing a priority scale which differentiates between decisions made by the various branches of the state. This is modelled in the CBPMS by specifying a child combining algorithm which specifies that the executive has precedence over the local when there are policy conflicts, although any of these conflicts can be referred to the judiciary for a second opinion. A good example of this conflict resolution in action occurred in 2004 in the Irish capital, Dublin. The city manager, a centrally appointed executive role, informed Dublin Corporation, a locally elected body, that he would ignore their decisions on spending estimates if they were not in line with his requirements [molony02]. As the legislature made it clear that they would resolve the dispute in favour of the executive, the conflict was resolved without court action. Having created these regional organisations, the high-level structure of the state is pictured in figure 5.7 below.

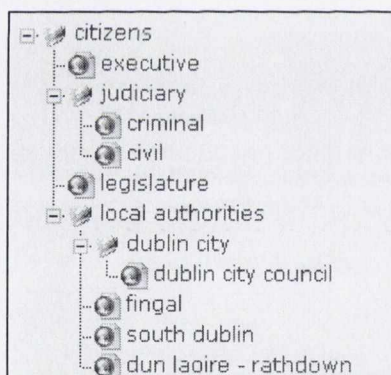


Figure 5.7 Segment of Sample State Community Structure

5.3.7 Federal Organisations

The final part of this case study involves an example of the modelling of federations and the relatively complicated organisational requirements that such situations throw up. In this case, there is a ready-made example to look at, in the form of the European Union (EU). Although the EU is famed for the arcane nature of its statutes, its core structure is relatively simple. It is a federation of states and although the member states are nominally independent and sovereign, they are bound to obey the decisions of certain federal institutions or else face disciplinary action – from fines to expulsion. Such requirements, where there are overlapping spheres of responsibility and something of a

grey area in terms of ultimate decision making power, is extremely difficult to model through conventional role-based approaches. However, it turns out to be a relatively straightforward task using community based modelling.

The first step in the model is to establish a root community to represent the EU. It is a root community since it has its own goals, structures and decision making bodies which are relatively independent of those of the member states. It possesses property in its own right and has its own body of legislation. Once this federal community has been formed, each of the states which wish to join the federation can simply use the federate primitive to join the federation.

```
31. genesis(EU)
32. federate(Citizens, EU)
```

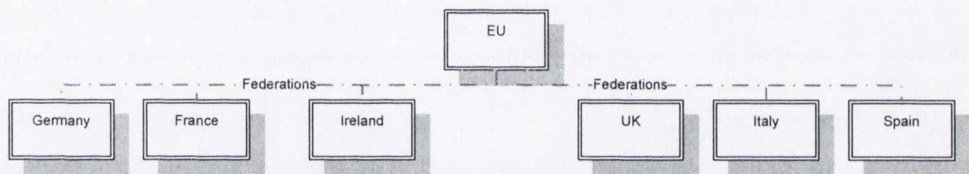


Figure 5.8 Segment of EU Federal Structure

The EU also possesses its own institutions independent of the member states. A sample of these institutions is picture in figure 5.9. This model works in the following way. The member states can share resources by delegating them to the EU community. The policies for how these shared resources are used are decided by the institutions of the EU. What is particularly notable about this structure is that it accurately captures the notion of national sovereignty within boundaries. Each federated country still owns its own resources; it is only those resources that are explicitly shared that the EU has ultimate authority over. The EU sits at the top of the community hierarchy for all shared resources and thus becomes the community in which conflicts over shared resources must be resolved. Any country can choose to leave the federation (or be expelled) and this simply means that it loses access to any shared resources.

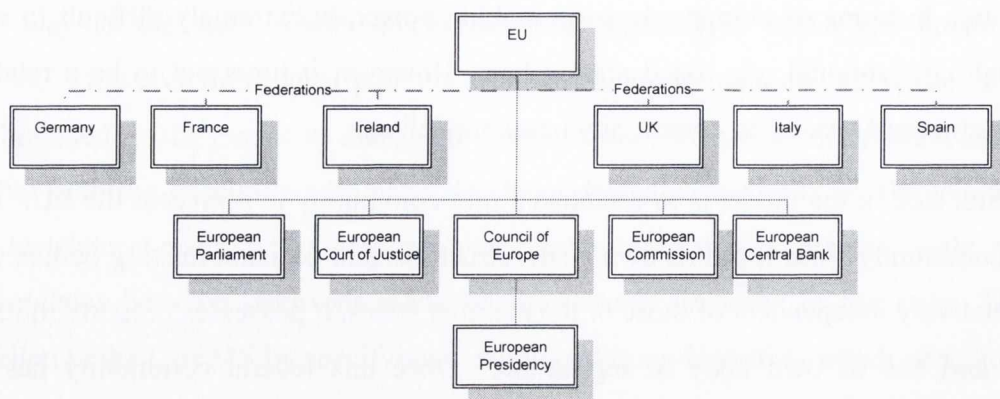


Figure 5.9 EU structure with institutions

Finally, it is worth looking at the way that such a structure can evolve. The policies of the federal community are specified so that any of the member communities may be expelled from the federation if they ignore its policy decisions. However, it is the state's prerogative to enact these policy decisions as local law and they are not directly enforceable within the state until they have been legislated locally or until they have been adjudicated on by the federal judiciary. Therefore, the state remains sovereign within those bounds – although if the state wishes to remain part of the federation, this sovereignty is limited by the federal policies.

5.3.8 Summary

This case study has focused on the CBPMS support for an evolutionary approach to building up a complex organisational structure. The relative simplicity of the modelling process illustrates the power of the community based model to capture complex and precise organisational requirements in dynamic environments. Although this example has been largely abstract, the provision of CBPMS primitives illustrates precisely how such structures can be built up over time.

5.4 Experiment 2 – Modelling Internet Communities

5.4.1 Oscailt

Oscailt, with means 'open' in the Irish language, is an open source content management system, designed for open publishing Internet news sites, partially written by the author of this thesis. It uses a basic role based system for mapping permissions to users. Each user is assigned to a set of roles and each role contains a set of positive permissions, with the user being allocated the union of all of the permissions which are contained by the roles to which they are assigned – a basic implementation of the core RBAC

standard [ansi04]. Each page, or page element (such as a menu, a list, a graphic, etc) is an object and permissions can be applied to individual objects. Objects are collected together into *sub-sites* and roles can be assigned on a per-site basis. This allows a relatively fine-grained specification of access rights, compared to most modern community management systems, with certain individuals being assigned to certain roles within a particular *sub-site* or being allocated permissions over a specific object.

Oscailt was initially developed to serve the needs of the Indymedia community, a network of inter-linked open publishing news sites. The Indymedia community was deemed a particularly suitable subject for community based modelling since its structure is characterised by a fluid membership, a high level of decentralisation and is composed of a large number of autonomous communities collaborating in the management of a shared technical infrastructure [feeney04a]. The author of this thesis has been an administrator of the Irish node of the network since 2002 and his observation of the evolution of organisational forms in such communities provided much of the material on which the CBPM theoretical model was initially built. Due to the consequent understanding of the domain and the relationships with the members of the community, the Irish Indymedia group was selected as a suitable candidate for this experiment.

The Oscailt codebase was modified to provide support for the CBPMS. This involved two distinct steps in development. Firstly, the array of permissions which is populated for each administrator was populated by calls to the CBPM Decision Service (CPDS) rather than to the permission database. Secondly, a plug-in was developed which allowed the CPMS (Community Policy Management System) to be integrated into the management interface. This allowed all access control to be managed and enforced through the CBPMS. In order to achieve as close a comparison as possible, an exceedingly simple policy specification language was deployed, consisting of nothing more than simple permissions which could be associated with resource authorities from the Oscailt resource tree as picture in figure 5.11 below. This limited the expressiveness of policies to the assignment of permissions to all members of a community without any constraints beyond those provided by the resource models. By combining such policies with delegation of resource authorities, a fine-grained management of access control was achievable, equivalent to that supported by the existing RBAC system. Finally, the membership rule definition for each community were limited to defining a subset of users who should be members of each community, which allowed the very same

interface to be used for the assignment of users to communities as had been used for the assignment of users to roles.

5.4.2 Community Model

Having integrated the CBPMS system into an experimental version of the Oscailt software, the challenge remained as to whether a community model of the organisation could be constructed which would reflect the participants' views as to how the organisation was structured. The fact that the majority of members came predominantly from a journalistic or editorial background rather than a technical one, and their observed resistance to technological change, made the process more difficult, albeit more realistic. After several weeks of discussion on the project's editorial email list¹, a community based structure for the group was published on June 15th 2005 [imc05]. The ensuing debate saw several minor modifications to the precise definition of mandates and decision making procedures described in the original proposal, however the fundamental configuration of communities was not questioned. The structure was formally adopted by a meeting of the indymedia editorial collective and the indymedia collective on the 30th of June 2005 by consensus. All 19 editors agreed that the model provided a more accurate description of the organisation's function than the former role-based model had allowed. In particular, the fact that the model associated privileges with functions, allowed limited autonomy and made the hierarchy of decision making clear were highlighted as advantages over the flat role-based structure – since these were features of the way that the organisation worked in practice that were not captured by the existing role configuration.

The community based model adopted is illustrated in figure 5.10. The boxes with thick borders represent communities, the blue boxes represent membership rules, the red boxes represent high-level obligation policies, while the yellow boxes represent authorisation policies. Each community is defined as self-managing and the distribution of privileges is left at the discretion of each community. The goal of each community is to distribute its privileges in such a way so as to best fulfil the goals of the community.

¹ Archived at <https://lists.indymedia.org/mailman/private/imc-ireland-editorial/2005-June/date.html>

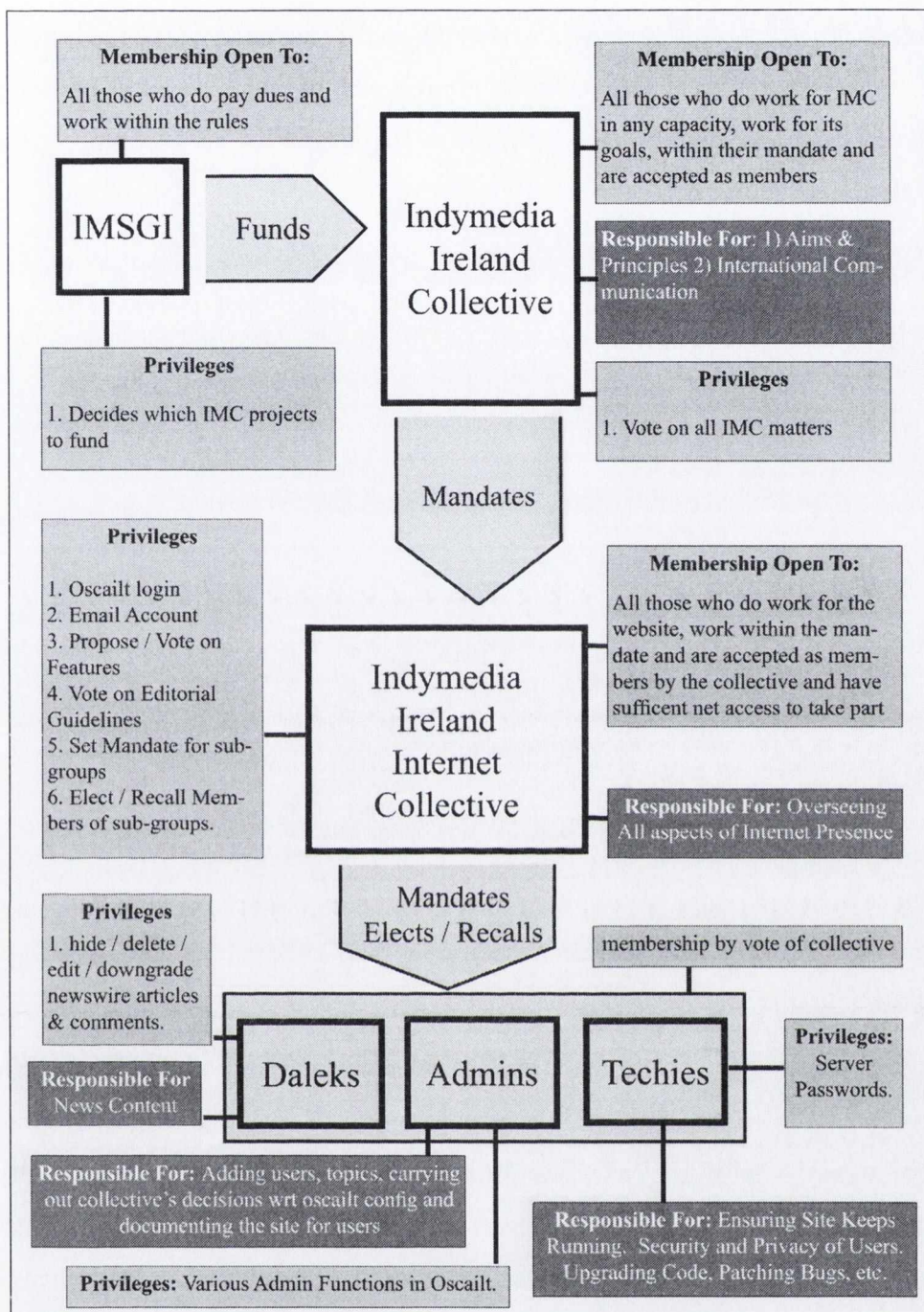


Figure 5.10 Indymedia Community Model

5.4.4 Community Model Implementation

Having formally adopted the community based model of the Indymedia Ireland group, the next step challenge was to map this model into a CBPMS implementation and to test the usability of this model. The formal description above was first modelled as a CBPMS community structure and the resources managed by the system (the Oscailt resource and the Community resource itself) were modelled as hierarchical authority trees, as shown in figure 5.11. Note that an extra community has been created

unregistered users in order to allow the specification of policies which apply to the non-logged in users of the site. The design of the resource authority tree allows delegations of authority and the specification of policies to be applied to groupings of content managed within the system.

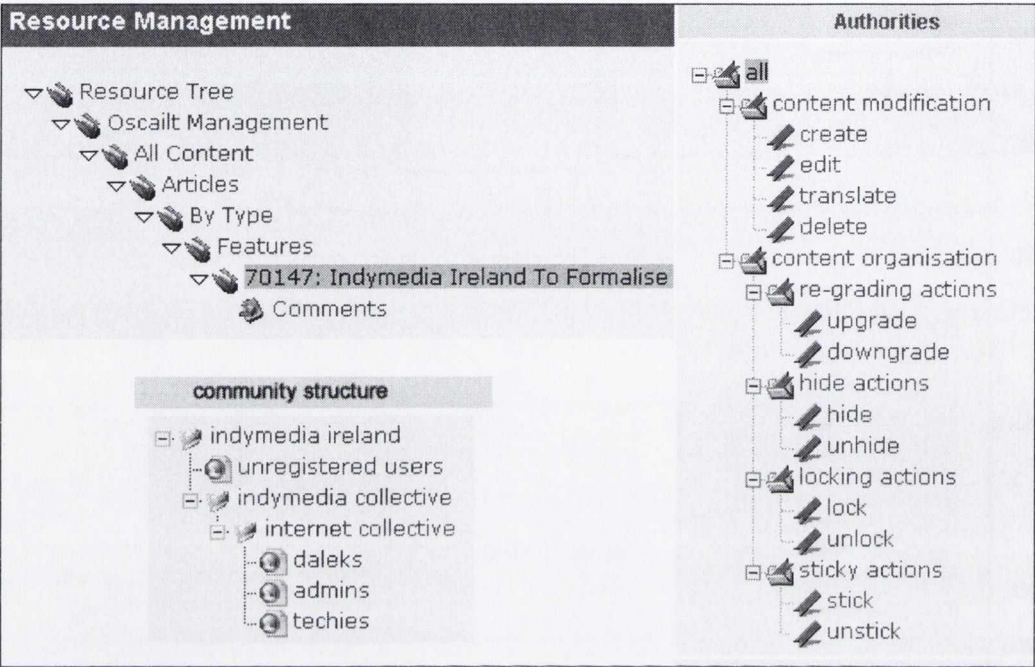


Figure 5.11 Oscailt Community Structure and Resource Target & Action Trees (montage of screenshots from GUI)

Having designed the resource model and the initial community structure, resource authorities were distributed throughout the community structure in order to reflect the requirements as defined in figure 5.10. For example, the resource authority representing each community, with the exception of the *unregistered users* community, was delegated to the community itself. This makes each community self-managing within the bounds established by communities further up the hierarchy. The responsibility for the management of all content in the system is delegated to the *internet collective*, the community representing all members active on the site, and from there to the *daleks* (a name which stems from the fact that the predominant editorial activity is the “extermination” of postings which contravene the editorial guidelines). It should be noted that many of the policies defined in this system are not automatically enforceable – for example the *internet collective* is responsible for the definition of editorial guidelines which define the function of the *daleks*; these guidelines are encoded as natural language statements stored within the system. Although the guidelines are not directly enforceable, the CBPMS system enforces the rules about who can access them

– in this case by stipulating that they may only be modified when the *internet collective* has voted to do so.

Having created the model and distributed the resources among the communities, it was demonstrated to 4 technical administrators of the existing role-based system and they were invited to test it in order to verify whether it fulfilled the requirements of the group. This was achieved by allowing the subjects to log in as any user of the system and to attempt to carry out actions within the system, in order to locate false positives and false negatives. In the course of this test two erroneous specifications were discovered, which necessitated modifications of the policies and an extra delegation in one case. Having tested the accuracy of the model in such a way, they were subjected to an interview comparing the community based approach with the role based system. Unfortunately, there were only 4 administrators who felt themselves sufficiently familiar with the workings of the role based system to be in a position to make this evaluation and therefore, the results from this experiment are of limited scope. Nevertheless, these administrators should have been in an ideal situation in which to evaluate the system, being very familiar with its requirements. Details of the tests and the subsequent interview are reproduced in Appendix 9.

The results revealed the following

- All of the administrators felt that the community based model presented a more accurate and complete model of the organisation and captured more of its operational rules.
- In particular, the ability to delegate self-management to sub-communities was highlighted as a key advantage over the existing role-based model as this was identified as being a core aspect of how the group functioned.
- The hierarchical nature of policy enforcement was identified as a desirable feature which allowed administrators to specify policies without having to worry about ‘breaking the whole thing’.
- The greatest problem encountered was with the complexity of policy specification through the CPMS GUI. None of the administrators had any experience with using policy based management systems and thus found it difficult to master. The GUI design was also highlighted as problematic in several places and proved somewhat time-consuming to navigate at times.

- On the other hand, the formation of a basic community model was seen as straightforward and simple to grasp, although delegation and forming federations were seen as somewhat more difficult.
- The difficulty of specifying policies was identified as a serious problem in implementing self-management. Each self-managed group would need to have at least one member who was reasonably familiar with policy based management and this would require significant training.

Unfortunately, due to Intellectual Property rights issues, it was not possible to deploy the Oscailt-CBPMS system for the management of the community. The Oscailt software is produced under the GNU-GPL license and the copyright for the CBPMS system is owned by Trinity College Dublin. Thus, the CBPMS modifications could not be distributed or deployed with the software. Therefore, the results from the experiments and questionnaires were not able to be validated by any extensive trial. Nevertheless, the information gathered from this experiment is undoubtedly valuable in illustrating the ability of the CBPMS to model a real world internet community.

5.4.5 Community Evolution

Although the CBPMS was not used as the practical management system for the reasons mentioned above, it continued to be used to track the structure of the community in order to test whether it was capable of modelling the evolution of a real world organisation. In September 2006, several major changes were adopted by the group [imc06]. The most significant structural changes were the following:

- The internet collective became an autonomous group – rather than being a sub-community of the overall collective. This meant that it could take all decisions pertaining to the site's management, without the necessity of calling a real-world meeting and getting approval from the overall collective.
- This newly autonomous collective then affiliated to its erstwhile parent through a federal relationship.
- Several new working groups were formed with distinct areas of responsibility. Each of these also enjoyed increased levels of autonomy in setting their rules – for example by changing the number of votes required before material was featured on the site's front page.

Modelling these changes might be expected to be a complex task as it involves the separation of an organisation. However, the CBPMS offers an easy path to capturing such changes through its simple primitive invocations. In order to transform the original model to reflect the new situation, the following series of steps was required.

- Ultimate authority for the Oscailt resource (and some other simpler resources which represented documentation) was granted to the internet collective (the parent collective previously had ultimate authority for it), using the *grant* primitive.
- The membership rule of the internet collective was modified to reflect the new method for approving members, using the *gatekeeper* primitive.
- The parent community invoked the *liberate* primitive to grant autonomy to the internet collective community.
- The newly liberated community invoked the *federate* primitive to federate to the Indymedia Ireland community, which had previously been its parent.

Since the function of the Indymedia Ireland collective had largely been decision making about high level principles – or abstract policies – there were no policies specified in that community for the resources that had been granted to the newly autonomous sub-community. That meant that no policy updates had to be made – all of the policies were already specified in the internet collective community and its sub-communities and these were unaffected by the change. If there had been any policies defined in the root community, these would have to have been transferred to the sub-community before the *grant* primitive was invoked. As, once this is invoked, there is no longer any authority for the resource in the root community and thus all of the policies would be out of scope and would not be enforced. Once the above primitives were invoked, the community structure was as pictured in figure 5.12.

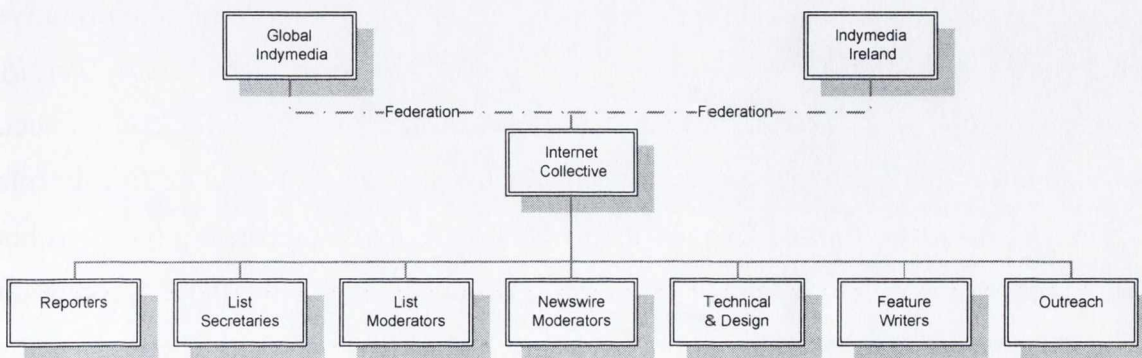


Figure 5.12 Updated Indymedia Community Model

5.4.6 Summary

An existing Internet community was modelled through the CBPMS and this model was validated by presenting it to the community's administrators who confirmed that it captured a more complete view of the organisation than their existing role-based solution, in particular that it incorporated notions of collective decision making and autonomy better which had been absent. Although the management system was not deployed in practice, it was used to track the changes to the organisational structure in a year-long period. The relative ease with which the model was adapted to deal with major structural changes strengthens the argument that the CBPMS provides a flexible and efficient way of modelling dynamic organisations with decentralised decision making hierarchies.

5.5 Experiment 3 - Comparing Communities and Roles

5.5.1 Scenario

In order to paint a clear picture of the essential differences between the standard *role* construct used in most policy management systems and the *community* construct used in the CBPM model, it is instructive to perform an experiment which reveals the differences in detail. In order to do this, the CBPMS and a simple Role hierarchy are both used to construct a model of an organisation. The particular organisational example used is taken from the paper which formally defines the ARBAC model [sandhu99a], to ensure that there is no bias towards the CBPMS from the choice of organisation. The organisation is a simple software engineering company. A number of roles are identified within the organisation, namely *Director*, *Employee*, *Engineer*, *Production Engineer*, *Quality Engineer* and *Project Lead*. In addition to these basic roles, several project-based roles are required, since the permissions of each individual are dependant on which project they have been assigned to. For simplicity, only two projects are modelled as this illustrates the principle adequately. Finally, a set of administrative roles are also required for the management of the system, namely *Senior Security Officer*, *Department Security Officer* and a *Project Security Officer* for each project. These roles are organised into the role hierarchy pictured in figure 5.13. This role hierarchy is intended to model a situation where there are two separate projects within an organisation which require different access control policies for those working on them, and there are a variety of different roles within each project. There are many problems associated with the use of role hierarchies to model organisations in policy

based systems, as described in chapter 2 of this thesis. The multiplicity of distinct role hierarchies within organisations identified by [moffet98] means that in order for role hierarchies to be applied to real-world organisations while respecting important security principles such as the principle of least privilege, the hierarchical structure must be adapted to include complex constructs such as private roles and hierarchies and limited inheritance [sandhu96].

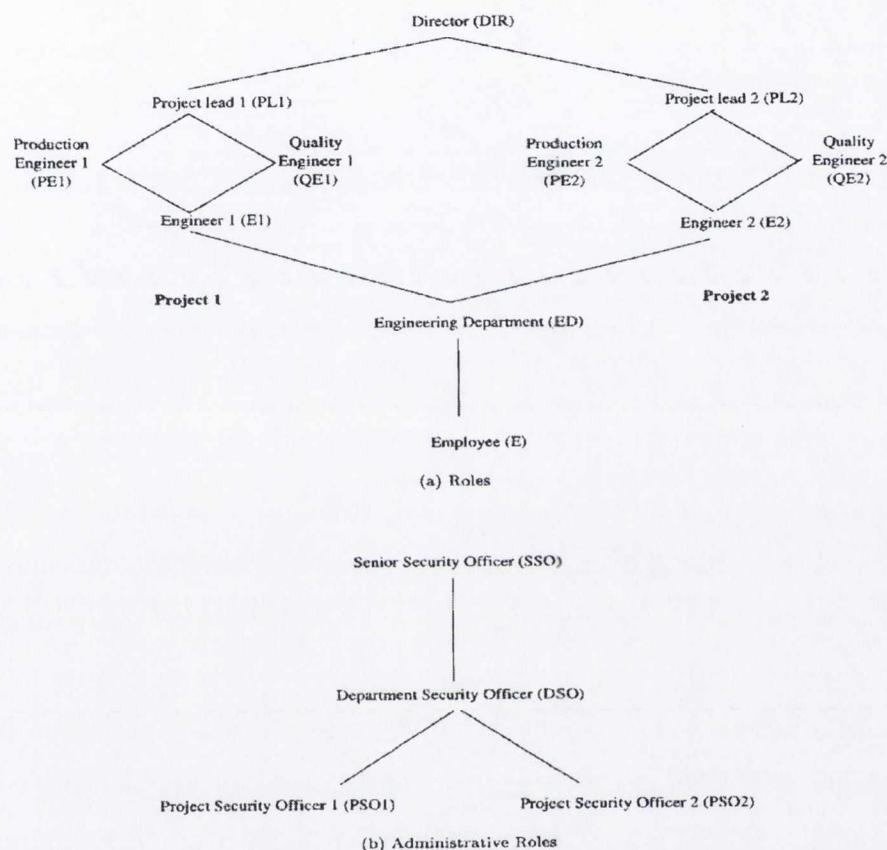


Figure 5.13 RBAC / ARBAC role hierarchy (higher roles inherit lower roles)

Thus, for example, in the above hierarchy, it is generally not desirable for the Director to inherit all the permissions of every engineer in the organisation, and thus private versions of the various engineering roles will have to be created. The need for these complex role-models stems from the fact that the hierarchical relationships in the model express different relationships. While a production engineer and a quality engineer are genuinely specialisations of the engineer role – in accordance with the notion of class hierarchies in object oriented programming – the Director and Project Lead roles are not specialisations of the roles that they inherit. Their relationship to the roles beneath them is a relationship of authority, not of specialisation. The roles beneath the director must follow her decisions and that is what these relations represent. Furthermore, from a

glance at the role hierarchy, it is immediately apparent that the organisational model is divided in two – administrators of the system and users are modelled in separate hierarchies. This combines with the earlier problem to create an organisational model which is at once overly complex and also fundamentally fragmented.

5.5.2 Constructing the Community Model

By contrast, the CBPM model is designed in order to incorporate a naturalistic model of the organisation. Rather than simply representing atomised roles, communities also contain a collective identity. This collective identity is expressed in the authority possessed by the community (through delegations) and can also be expressed through policies which contain collective decision making conditions. Furthermore, since the community structure itself is simply another resource, there is no need for a separate administrative hierarchy. From this basis, the role hierarchy above can be mapped to a CBPM model that takes advantage of these features without sacrificing its ability to represent the organisation.

Figure 5.14 shows the equivalent community hierarchy. In general, this follows the CBPMS approach to modelling organisations, where a community is positioned as an immediate sub-community of that community which is its smallest super-set in terms of membership and authority. In this case, the *Employees* community is the smallest super-set of the *Director* community in the hierarchy, so it belongs as an immediate sub-community of *Employees*. In general, this rule holds true for the community hierarchy – communities representing positions of authority over other communities are positioned as immediate sub-communities of that community. This introduces an important divergence from the hierarchical RBAC paradigm. In the RBAC example cited, the director role inherits all permissions or policies specified for the other non-administrative roles in the organisation. Thus, if a policy is added to the *Project 1 Engineers* role, this policy will automatically apply to the *Director* role too. In the modified community model, this is no longer the case. If a policy is added to the *Project 1 Engineers* community this will not automatically apply to the member(s) of the *Director* community since it is not a sub-community of *Project 1 Engineers*. In order to make the new policy apply to the members of the *Director* community, the policy must also be added to that community, or must be specified in their closest common ancestor – in this case the *Employees* community.

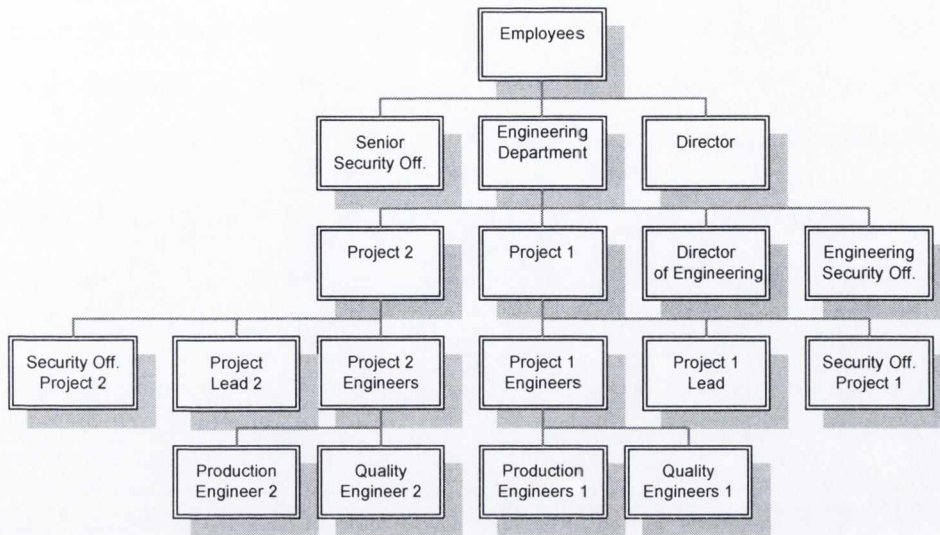


Figure 5.14 Equivalent Community Hierarchy

Meanwhile, the administrative hierarchy has been integrated into the organisational model. Thus, for example, the *Security Officer* of each project is a sub-community of the project itself and the *Senior Security Officer* is a sub-community of the *Employees* community. In order to use this community hierarchy to capture the requirements of the organisation, and in particular to model the administration of the community structure itself, resource authorities representing the community structure resource are distributed throughout the organisation in the following way.

- The community resource representing the employees, engineering department, and the project communities are delegated to the communities themselves – giving each of these communities limited self management. Then this authority is further delegated to the relevant administrators, who will carry out the self-management on behalf of the community above.
- A policy is created for each one of these administrative communities which specifies that the administrator is permitted to carry out any changes that are approved by the community that manages the community above – thus, for example, the *Senior Security Officer* requires the approval of the director for changes to the policies which affect all employees, while the approval of the *Project 1 Lead* is required for changes to the policies which affect *Project 1*.

Decision making authority is now distributed so that the following rules apply:

- The *Senior Security Officer* can define policies, with the approval of the director, which will apply to all employees in the organisation. Due to the fact that

policies are evaluated according to the hierarchy of authority of the organisation, policies defined at this level can not be over-written by policies defined further down the organisation.

- The *Engineering Department's Security Officer* can define policies, with the approval of the director of engineering, which will apply to all members of that department – once they do not conflict with policies which apply to *Employees*.
- The *Security Officers* for both projects can define policies, with the approval of the *Project Leads*, which will only apply to members of their respective projects and these policies will be bound by any policies defined above.

This hierarchical distribution of decision making authority captures the administrative requirements of the organisation in a much more structured way than the administrative hierarchy in the role-based version. The community structure tightly couples each organisational unit with its decision making process and the limited delegation of authority over the community resource ensures that administrators are limited to specifying policies which apply to the units for which they are responsible. An important feature supported by the community model which is absent from the role-based model is the concept of self-management. Since the community structure is modelled as a resource like any other, authority for community management can be distributed throughout the organisation like any other resource. Thus, a community may have authority to author its own policy rules and to spawn sub-communities and delegate authority to them. However, any policy rules or delegations which target resources which have not already been delegated to that community from its parent will not be accepted by the system. This effectively allows the organisation to be modelled in such a way as to support bounded self-management. The delegations define the boundaries, and within those boundaries the community is self-managed. By contrast, role based models offer none of these features.

In addition to the communities which have been more or less directly mapped from the role hierarchy into the community structure in figure 5.14, several new communities have been introduced. In particular a community represents each project. These are grouping communities which may not have any policies specified within them and may merely serve as a conduit for distributing authority throughout the organisation. The individual permissions of users are defined by policies set in sub-communities, while

the grouping communities merely serve as delegation distribution points for resources, gathering together all the authority that is required by all the communities underneath them. Hierarchical roles are not suitable for such grouping and authority distribution constructs. This feature of the CBPM model facilitates a much more organic model of the organisation, by grouping related communities together, even in cases where they share no policies, and facilitating the principle of minimum delegation of authority.

Another advantage that the community model provides is the ability to delegate a limited sub-set of authority to a sub-community, thereby limiting the target-scope of policies defined within that sub-community. So, for example, rather than delegating all authority for all resources from the *Engineering Department* community to the *Project 2 Engineer* community, the community model operates on the basis that only that authority which is required to fulfil a community's function is delegated. For example, if the only managed resource used by *Project 2* is a particular database, then authority for the particular database is delegated rather than authority for all resources. This principle of minimum delegation provides several advantages over role based models which do not constrain the policies that could be specified in particular roles.

1. The danger of erroneous policy specification granting excessive access rights to individuals is reduced since any policy within a community which has a target that has not been delegated to the community will be ignored. In a role based system, there is little protection against a stray policy specification which grants un-envisaged access to the holders of a role which should only grant minor privileges. The CBPMS guards against this by requiring an explicit delegation before any policies can be authored with the resource as their target.
2. The distribution of authority over resources – which resources groups are allowed to use and in what ways they can use them – is mapped across the organisation. By following the chain of delegations throughout the community model, an overview of resource usage and requirements is contained within the model which can prove useful for comprehending how the organisation's structure is linked to its function.
3. The CBPMS policy search algorithm utilises delegations of authorities to eliminate branches of the community hierarchy from the policy search. When a policy decision request arrives at the service, referencing a particular target resource, the CBPMS only needs to search for applicable policies within

communities which have been delegated the required authority over the relevant resource. In a role based system, each role that the subject is a member of must be searched for relevant policies since there is no way of knowing if a role might contain an applicable policy for a particular decision request without examining the policies within it. Furthermore, the CBPMS conflict resolution approach allows all sub-communities to be discarded from the search as soon as an applicable policy is discovered in a parent community. In complex organisational models, this can vastly reduce the size of the policy search space.

Although the community model's support for automatic conflict detection based on the community hierarchy has been touched upon above, it is worth making a few final comments on this feature. For a start, it might seem somewhat counter-intuitive that the conflict resolution algorithm always follows the community hierarchy. This means that policy rules specified in the *Employees* community will have precedence over those specified in the *Director* community which inverts the normal concept of an organisation's hierarchy. However, it should be recalled that when a policy is specified in a community, this signifies that the policy should apply to **all members of the community without exception**. Thus, care must be taken so that any policies specified for a community should really apply to all members of the community. In many cases where a policy should only apply to a subset of a community, but there is no community to represent this subset, this signifies an incomplete community model and an intermediate community is required in order to incorporate the required subjects.

In addition to the simple parent-child automatic conflict resolution mechanism, the CBPMS can also detect and resolve conflicts between communities which have no direct hierarchical relationship by ascending the community tree to the nearest common parent of the conflicting communities and applying that community's specified child combination algorithm (as described in chapter 3). This detailed conflict detection and resolution mechanism can also be employed at policy-specification time in order to highlight situations in which specified policies will be overruled by others defined elsewhere in the community hierarchy.

5.5.3 Simulation Design

In order to evaluate the claimed benefits of the CBPM model in terms of policy search efficiency and conflict detection, the models described above were used to run a series of simulations, measuring the number of policies that must be evaluated and the number

of conflicts that can be automatically resolved in any given context using a role-based model versus a community based model. The experiment was conducted by randomly distributing policies through the CBPM model, generating sequences of random requests and measuring the various numbers of conflicts that were resolved and the number of policies that had to be evaluated for each request by each system. For the purpose of the experiment, the policy specification language consisted of simple positive and negative authorisations, including no constraints or conditions, with the following form: `<action> <target> [permit | deny]`. Due to the fact that there are several different ways in which Role-Based Policy Management Systems (RBPMS) handle permission inheritance [miege05], an approach was adopted that would tend to minimise the amount of potentially relevant policies in the RBPMS configuration and the frequency that conflicts could be expected. Users were not simply assigned all the policies in all the roles to which they belonged – it was assumed that some policies defined in senior roles might be redefinitions of policies defined further down the hierarchy (for example, the Ponder concept of role hierarchies using the *extends* keyword can be used to redefine policies in such a way). Thus, the experiment adopts the assumption that there is a certain number, denoted $p(\text{overwrite})$, which represents the probability that any policy in an inherited role will over-write a policy in a base role, allowing the base-role policy to be excluded from the policy search algorithm. What this probability is in practice is unknown. Chances are that it will turn out to be rather small in most instances, but it might vary greatly depending on the application and the tastes of the administrator.

With this corrective in use, a simple algorithm was constructed for calculating the number of policies that must be evaluated in a given context using the role-based model. This algorithm, implemented by CBPMS role emulation, starts at the bottom of the role hierarchy, checks to see whether the user is a member of the role, and if so, adds any policies which are not identified as over-writes to the set of policies that must be evaluated. If the user is not a member of the role, or there are no more roles higher up the hierarchy to try, the search ends in that branch. In order to make the role-model as close as possible to the CBPMS model to which it is being compared, it is assumed that the role-membership checks are based upon attributes of the subject, rather than being assigned by a security officer (i.e. that a model like Rule Based-RBAC is used [alkahtani04]).

The community-based policy search algorithm is slightly more complex. The search starts at the root community and descends the hierarchy progressively to those sub-communities to which relevant authority has been delegated, until the community membership rule fails or there are no further communities or an applicable policy is discovered. Only those policies which are associated with resource authorities of greater scope than the required access need to be evaluated.

Thus, with the above configurations in place, the following experiment was constructed. A model of a simple target resource was adopted as in figure 5.15. A series of simulated policy decision requests were generated, using nodes randomly selected from this resource model. Thus, for example, the generated policy decision request *write file1*, represents a user's attempt to update this file. The simulation involved running a number of such randomly generated requests and evaluating the average number of policy evaluations that had to be carried out in order to arrive at a policy decision. While the Role-based model has no knowledge of the hierarchical nature of the resource, the resource tree still served as a way of identifying conflicting policies.

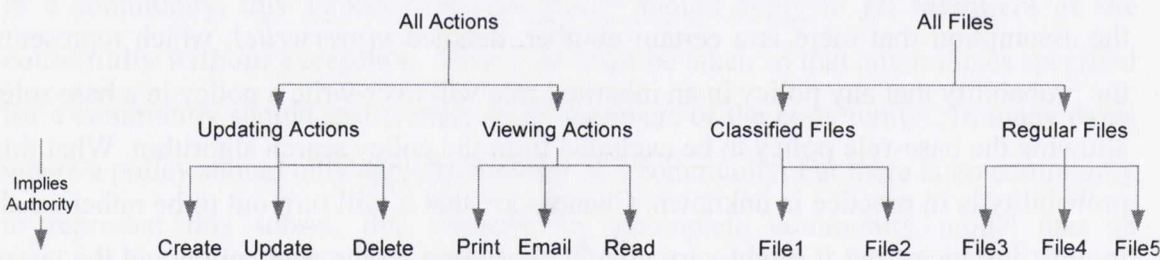


Figure 5.15 Simple Resource Model used in simulation experiment

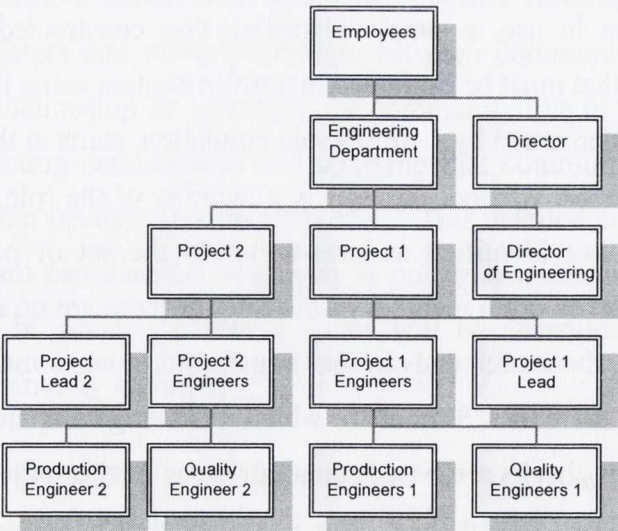


Figure 5.16 Community Hierarchy for Comparison

The Role model was constructed according to the schema pictured in figure 5.13(a) and a community model was constructed to mirror it, as shown in figure 5.16. Note that the integrated administrative communities are omitted, as are the other extra communities added above. This ensures that the comparison is between two models which are based on the same organisational structure. In the cases of both models, a number of policy rules were generated and randomly distributed among the various roles or communities in the structure. In the case of the CBPM model, each of these policies was further randomly associated with a resource authority from the resource authority tree, defining the scope of the policy. Furthermore, in order to model the distribution of authority within the CBPM model, the minimum authority required to ensure that these policies were within scope was then distributed to the relevant sub-communities.

A probability, $p(member)$ was assigned to the chance that any particular subject passed the membership test of any particular role or community. Since a user can only be a member of a role or a community if she is a member of the base roles, or communities further up the hierarchy, this effectively means that the probability of a subject being a member of each role decreased as the community tree was descended and as the role hierarchy was descended. Therefore, the probability that a particular subject was a member of a community was equal to $p(member)^n$. With $n=0$ in the role at the bottom of the hierarchy and in the community at the top of the community hierarchy, and increasing by one with each step away from this base.

A further probability, $p(overwrite)$, was assigned to the chance that any particular policy in the role-based model overwrote a policy in a base class. Finally, a probability, $p(applicable)$, was assigned to the chance that any particular policy within scope would be evaluated as applicable by the CBPMS and would return a valid policy decision. Thus, the size of the search space in the competing models could be compared.

For each policy request, the role-based system merely has to do two things. First map the request to the appropriate roles, then evaluate the policies within the roles. The CBPMS also carries out these evaluations, and in a way that is, to all intents and purposes, identical. There is no particular reason why such evaluations should be any more or less expensive when carried out by either the CBPMS or the role-based system. The difference is that the CBPMS also carries out authority comparisons, comparing the resource authorities associated with requests with the scope of policies and the authority that has been delegated to communities. It is this use of scope which allows the

CBPMS to radically reduce the number of calculations that must be carried out. However, this leaves the question of how expensive are these comparisons themselves? The answer is that it depends on the complexity of the resource model. Most resource models should allow authorities to be compared in highly optimised ways – for example by merely comparing two real numbers, or comparing nodes on a hierarchical tree. However, there is no limit on the algorithms employed by the RAIS to calculate relationships between resource authorities and this could therefore become complex and expensive. In any case, it should be virtually impossible for this evaluation to be as expensive as a full policy evaluation, since comparing targets and actions is but a small part of the work that needs to be done to evaluate policy rules in most cases. Thus, the worst case scenario is taken to be that this evaluation is as expensive as a policy rule evaluation, while a more reasonable scenario is that it is 25% as expensive. Using these mappings, and also equating membership evaluations with policy evaluations, a direct comparison of the two models can be created. In the results below, the RBPMS system is compared to the CBPMS using both these mappings, 1:1 and 0.25:1, in order to express a worst-case scenario for the CBPMS as well as a normal one.

Finally, in each run of the simulation, the policies evaluated are checked for conflicts. The simple policy specification language used can produce two decisions, *permit* and *deny*. A conflict is indicated if a two applicable policy have the same target and action and opposite results (i.e. a modal conflict). The number of policy conflicts that arise per 1000 requests is counted for each model, in order to compare the systems' capacity for automatic conflict resolution. The simulation is run several times, using different algorithms for distributing the randomly generated policies among the roles and communities, to investigate how they might affect the outcome.

5.5.4 Simulation – First Run

Each simulation was run in the following manner. A number, P , of policies were generated, with randomly generated resource authorities associated with them. These policies were then randomly distributed, amongst the roles and the communities in the two models. Fifty random policy decision requests were then generated. Then another set of policies was generated and randomly distributed again, replacing the originals. This was carried out 20 times, meaning that a total of 1,000 different policy decision requests were evaluated in total for each value of P . For the first simulation $p(\text{membership})$ was set at 0.5, $p(\text{applicable})$ was set at 0.5 and $p(\text{overwritten})$ was set at

a number between 0 and 0.5 depending on how many policies were present in inherited roles – if there were no policies available to be overwritten, the probability that a new policy would overwrite one in an inherited role became 0, if there were more than 100 policies in inherited roles, the probability rose to 0.5. Although these values are arbitrary, they seem reasonable. The simulation was first run with the values of $P = 10, 20, 50, 100, 200, 500, 1000, 2000, 5000$) and returned the values shown in tables 5.4 and 5.5, while figures 5.17 and 5.18 show them in graph form, with the left axes using a logarithmic scale.

<i>Policies in System</i>	<i>Role Membership Tests</i>	<i>RBAC policies evaluated</i>	<i>Estimated expense of RBAC evaluations</i>	<i>Conflicts per 1,000 requests</i>
10	5.47	2.17	7.64	85
20	5.5	4.57	10.07	100
50	5.51	12.18	17.69	421
100	5.51	24.86	30.37	743
200	5.6	49.78	55.38	953
500	5.37	123.31	128.68	1000
1000	5.56	250.03	255.59	1000
2000	5.63	513.79	519.42	1000
5000	5.42	1231.47	1236.89	1000

Table 5.4 RBPMS Results from first simulation

<i>Policies in System</i>	<i>Community Membership Tests</i>	<i>Community policies evaluated</i>	<i>Authority Comparisons</i>	<i>Estimated expense of CBPMS evaluations (worst-case)</i>	<i>Estimated expense of CBPMS evaluations (normal)</i>	<i>Conflicts per 1,000 requests</i>
10	2.14	0.04	4.34	6.52	3.26	6
20	2.44	0.14	5.26	7.84	3.89	0
50	3.1	0.35	10.03	13.48	5.95	15
100	3.27	0.82	17.32	21.41	8.42	28
200	3	1.33	30.62	34.95	11.98	70
500	1.84	3.37	55.93	61.14	19.19	301
1000	1.37	6.53	90.44	98.34	30.51	624
2000	1.07	12.8	158.95	172.82	53.60	894
5000	1.01	33.18	382.4	416.59	129.79	987

Table 5.5 Community-Based Results from first simulation

From these results, some observations about the experiment can be made. The number of community membership tests that had to be carried out tended towards 1. This is a consequence of the community semantics. As the number of policies increases, there are more policies defined for the root community and thus, at a certain point (5000 policies in this example), a point is reached where the probability that an applicable policy will be found in the root community approaches 1. This feature is a consequence of the random distribution of policies throughout the model. In real applications, this distribution is definitely not random and the policies defined in the root community define rules which all users must abide by. Another feature worth noting is the fact that,

in most cases, when the number of policies in the system is small, the CBPMS will not evaluate any policies. This is a consequence of the fact that only policies that are in scope and in communities which the user is a member of need to be evaluated. Even when the number of policies in the system grows very large, a relatively small number of them need to be evaluated, as most are not in scope and it only requires a relatively simple authority comparison to check whether a policy is in scope.

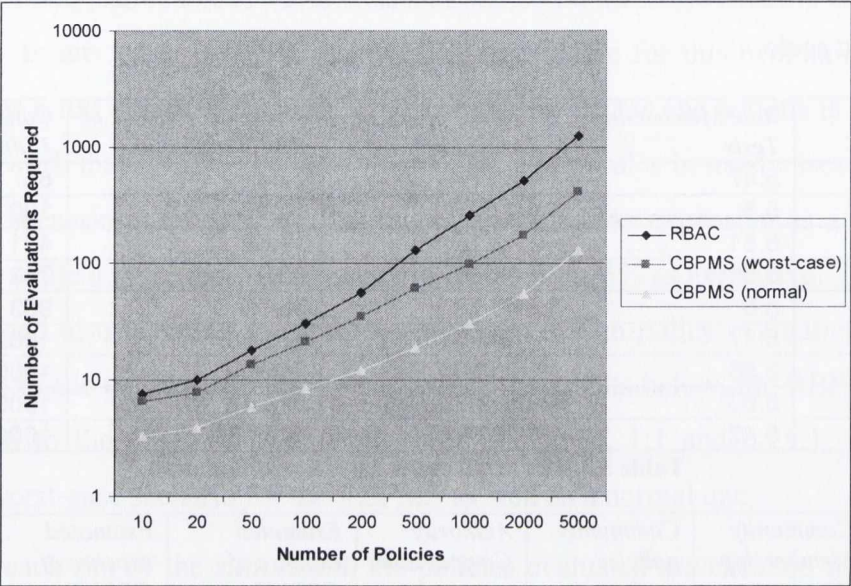


Figure 5.17 Estimated expensiveness of CBPMS and RBPMS with respect to the number of policies

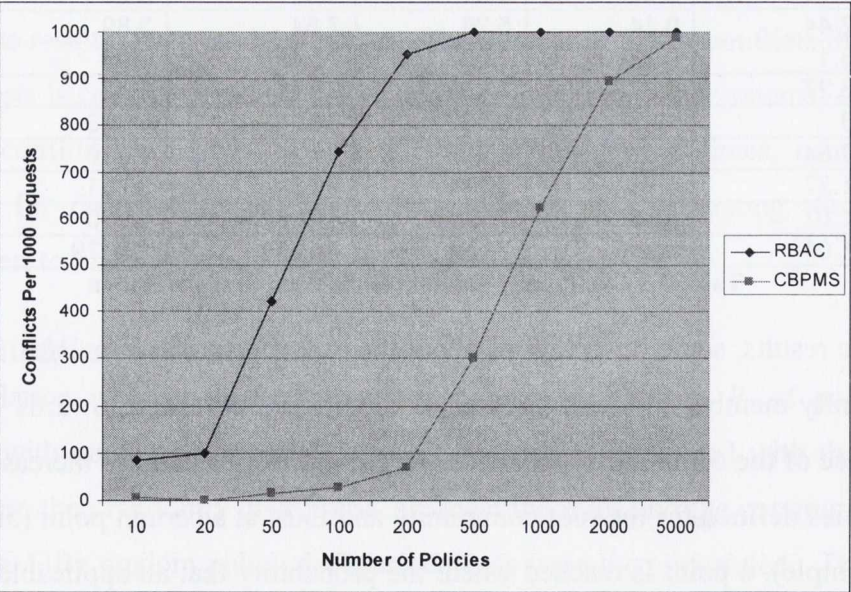


Figure 5.18 Simulation 1: Frequency of policy conflicts in RBPMS and CBPMS

It is also worth noting that the probability of the RBPMS encountering an irresolvable conflict increased significantly as soon as there were 20 policies in the system and once there were 500 or more policies in the system, the probability became 1. By contrast, the

CBPMS conflict probability was still less than 0.1 when there were 200 policies in the system, and did not reach 1 until there were 5000 policies (at which stage there were so many policies about the simple resource present in the root community alone, that a conflict was almost certain).

5.5.5 Simulation – Second Run

For the second simulation, the following changes were made. Rather than being constant, the value of $p(\text{applicable})$ was modified for each community in the model. The closer to the root community, and the greater the number of policies defined in the community, the smaller $p(\text{applicable})$ is. The value was designed to give a probability of less than 0.5 that there is an applicable policy in each community. This represents an extremely pessimistic estimate. In general, in the CBPMS, policies that are within scope are applicable except when they include a conditional clause that is evaluated and found to be not applicable. This assumption means that, if there are 1000 policies defined for the root community, each of them has a probability of less than 0.0005 of being applicable even when in scope. Having made this adjustment, the simulation was re-run to test this worst case scenario comparison, with results below.

<i>Policies in System</i>	<i>Role Membership Tests</i>	<i>RBAC policies evaluated</i>	<i>Estimated expense of RBAC evaluations</i>	<i>Conflicts per 1,000 requests</i>
10	5.47	2.22	7.69	3
20	5.51	4.85	10.36	26
50	5.58	11.13	16.71	42
100	5.67	21.2	26.87	39
200	5.42	37.86	43.28	30
500	5.44	93.67	99.11	49
1000	5.57	192.18	197.75	37
2000	5.37	360.71	366.08	23
5000	5.54	970.99	976.53	33

Table 5.6 RBPMS Results from second simulation

<i>Policies in System</i>	<i>Community Membership Tests</i>	<i>Community policies evaluated</i>	<i>Authority Comparisons</i>	<i>Estimated expense of CBPMS evaluations (worst-case)</i>	<i>Estimated expense of CBPMS evaluations (normal)</i>	<i>Conflicts per 1,000 requests</i>
10	2	0.05	4.42	6.47	3.15	0
20	2.57	0.17	5.68	8.42	4.16	0
50	3.48	0.16	9.16	12.8	5.93	0
100	4.23	0.17	16.9	21.3	8.62	0
200	4.57	0.19	31.95	36.71	12.74	0
500	5.17	0.12	88.4	93.69	27.39	0
1000	5.41	0.17	185.97	191.55	52.07	1
2000	5.16	0.16	342.45	347.77	90.93	0
5000	5.34	0.18	927.29	932.81	237.34	2

Table 5.7 CBPMS Results from second simulation

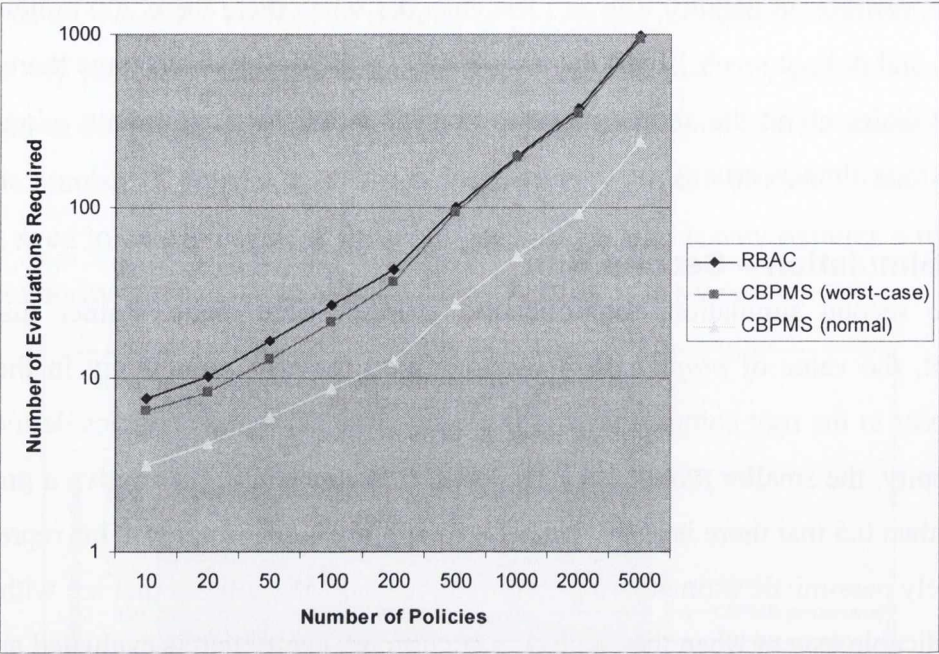


Figure 5.19 Results of Second Simulation – policy evaluations required

The results show that, even when these pessimistic assumptions are adopted, the CBPMS still requires significantly less policy evaluations (by a factor of 4 when resource authority comparisons are evaluated according to “normal” values) than the RBPMS. The probability of conflicts in the CBPMS is also significantly less than for the RBPMS, although both are predictably low compared to the first simulation run due to the fact that very few of the evaluated policies are considered applicable.

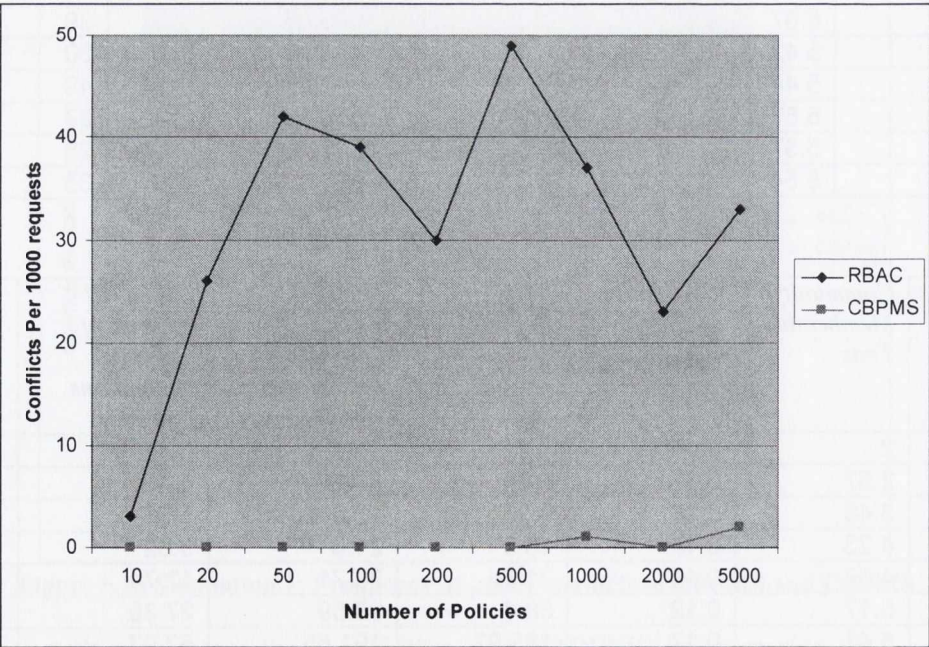


Figure 5.20 Frequency of Policy Conflicts in Second Simulation

5.5.6 Simulation – Third Run

A third and final run of the simulation was performed. This time, the assumptions about random policy distribution in the community model was changed so that most of the policies were defined within leaf communities. This is intended to reflect a situation where there are relatively few policies defined in the higher level communities, since there are few rules that apply uniformly across the entire organisation. Most of the rules about a particular function are fairly closely linked with the specific function and most of the policies defined in leaf communities or other nodes relatively far down the hierarchy. To model such a scenario, the following distribution algorithm was used. The probability of any given policy being assigned to a particular community or role was weighted by powers of two depending on its distance from the root community or role. Thus, the probability of a policy being assigned to a community or role on level 2 was twice that of it being assigned to level 1 and the probability of it being assigned to a community or role in level 3 was 8 times that of level 1 and so on. Having adopted this algorithm for policy distribution, the simulation was re-run. The results are shown in table 5.8 and 5.9.

Note that by the time that there were 5,000 policies in the system about the simple resource, the probability that one of them will be applicable in the root community still approaches one, despite the weighted distribution. This is simply a consequence of the simplicity of the resource model and the large number of policies about the resource. However, the probability of conflicts still increases much quicker, and with a much smaller number of policies in the system for the RBPMS than for the CBPMS.

<i>Policies in System</i>	<i>Role Membership Tests</i>	<i>RBAC policies evaluated</i>	<i>Estimated expense of RBAC evaluations</i>	<i>Conflicts per 1,000 requests</i>
10	5.5	1.19	6.69	0
20	5.47	2.54	8.01	0
50	5.51	6.78	12.29	13
100	5.31	11.82	17.13	15
200	5.52	25.52	31.04	125
500	5.46	63.67	69.13	430
1000	5.42	127.54	132.96	850
2000	5.41	247.57	252.98	980
5000	5.53	652.05	657.58	1000

Table 5.8 RBPMS Results from third simulation

<i>Policies in System</i>	<i>Community Membership Tests</i>	<i>Community policies evaluated</i>	<i>Authority Comparisons</i>	<i>Estimated expense of CBPMS evaluations (worst-case)</i>	<i>Estimated expense of CBPMS evaluations (normal)</i>	<i>Conflicts per 1,000 requests</i>
10	2.15	0	3.71	5.86	3.07	0
20	2.75	0.01	3.71	6.47	3.68	0
50	3.52	0.03	5.81	9.36	5.00	0
100	3.87	0.09	9.95	13.91	6.44	0
200	4.48	0.14	22.15	26.77	10.15	0
500	4.16	0.48	49.43	54.07	16.99	10
1000	3.69	0.84	85.2	89.73	25.83	25
2000	2.8	1.75	117.02	121.57	33.80	128
5000	1.5	4.43	115.02	120.95	34.68	467

Table 5.9 CBPMS Results from third simulation

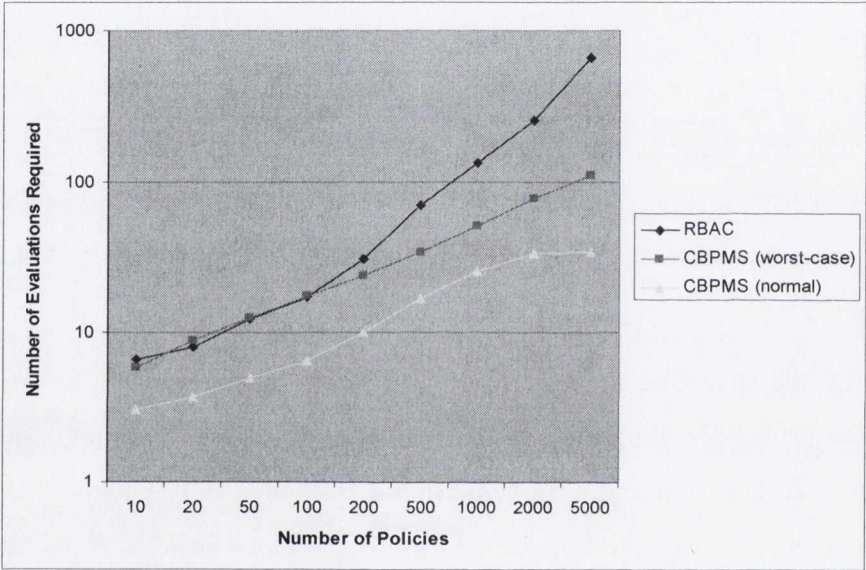


Figure 5.21 Results for third simulation – policy evaluations required

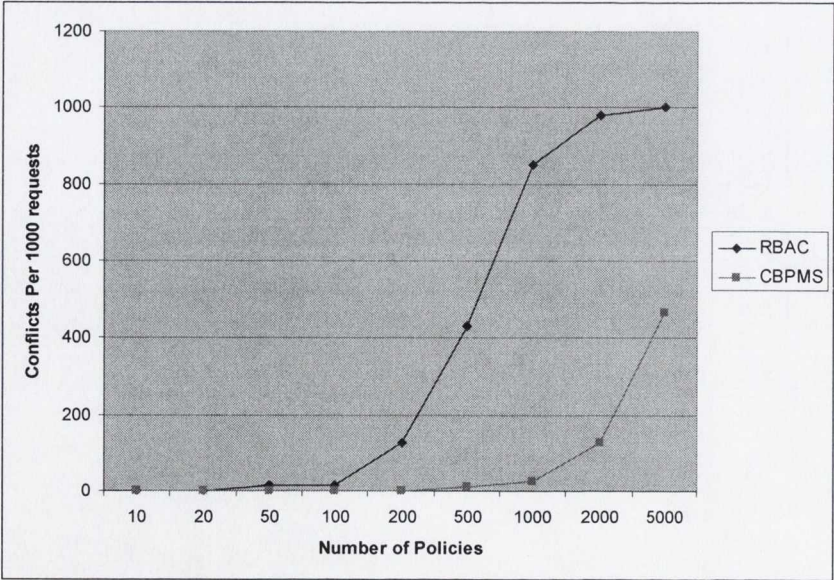


Figure 5.22 Frequency of Policy Conflicts in third simulation

5.5.7 Simulations – Summary

As can be seen from the graphs of the results, even when the worst assumptions are adopted with respect to the distribution of policies and the expensiveness of authority comparisons, the number of evaluations required by the RBPMS model increase at a greater rate than the CBPMS model as the number of policies increases. When normal assumptions are adopted, the number of evaluations required for a policy set of 5,000 policies is reduced by a factor of almost 20, compared with the RBPMS model under test. This is a very substantial performance gain and it also serves to limit the cognitive load on policy authors since, whenever a new policy is authored, it is only the policies which have to be evaluated that need to be considered by the author as all other policies are out of scope and cannot conflict with the new policy (as long as the resource models are constructed correctly). The marked reduction in policy conflicts when using the CBPMS as compared to the RBPMS model also highlights the CBPMS's capacity for automatically resolving conflicts based on the hierarchy of authority.

These benefits are not entirely without costs. They depend upon the construction of a hierarchical resource model. They also depend upon the fact that the CBPMS constrains the specification of policies in accordance with the community and resource models. Thus, the reduction in the size of the policy search space and the frequency of conflicts comes at the cost of the requirement to create a hierarchical resource model and the constraints that the CBPMS model imposes upon policy authoring.

Comparing these results with the benchmark metrics derived in the first experiment (section 5.2) it appears that the simulation bears a reasonably close relationship to what was experienced in that more-realistic setting. Although the numbers of evaluations is less than when the size of the policy set is considered, this can be considered a consequence of the fact that the experiment's policy rules were not randomly generated or distributed. They were authored for almost every resource node possible, in order to model all the requirements and were located in specific communities where they were likely to be applicable. Furthermore, both the community model and the resource model used in the simulation were much larger and more complex. Taking these considerations into effect, the results are broadly consistent with those returned by the simulation. In particular, the relative numbers of the different types of evaluations is quite consistent.

Chapter 6. Conclusions and Further Work

This chapter summarises the findings of this work, compares the results achieved in the previous chapter with the goals of this research, as outlined in section 1.2, and evaluates to what extent those goals have been reached. It then goes on to assess the overall impact of the work, and the major areas in which further research is suggested.

6.1 Analysis of Research Goals

Goal 1: CBPMS Development

The first goal of this research was to *“devise and develop a formal organisational model for PBM, defined in non-ambiguous terms, and to demonstrate that the model can be integrated into an organisation’s IT systems to deliver a practical PBM capability.”* The basic structures of the community and the significant groupings of communities (organisations and federations) in the model were defined, in full, using First Order Logic (FOL), in Chapter 3. Appendix 2 contains a formal definition of the decision algorithm used by the CBPMS decision service while appendix 3 contains a full definition of the operational semantics of the CBPMS primitives. Although the CBPM model was designed to be policy language agnostic, in order for a policy language and reasoner to be supported by the system, it must provide a minimum set of functions to the CBPMS – these functions and their semantics are also described in full in Chapter 3. Together, these elements provide a formal mathematical definition of the state of the CBPMS, precise constraints on how that state might change (through invocation of the management primitives), formal requirements for integrating policy reasoning engines, and a formal definition of how the CBPMS produces policy decisions dependant on its state.

To allow the CBPMS to deliver a practical PBM capability to any particular domain, first an implementation has to be designed and constructed; the applications that serve as Policy Enforcement Points (PEPs) have to be modified to make them policy aware; the CBPMS has to be integrated into the network infrastructure to allow it to accept requests and return decisions to the PEP in a timeframe and with a reliability that meet the application’s requirements; and finally a hierarchical model of the resources that are to be managed has to be constructed – so as to provide useful sub-divisions of resources for the purpose of policy specifications and authority delegations.

The architecture, design and development of the CBPMS system was fully described in chapter 4, based on a modular service oriented architecture, and an implementation of the system with a XACML policy language was outlined. Chapter 5 described two experiments in which the CBPMS was integrated into realistic IT networks, allowing several applications to become policy-driven. The first of these experiments involved the management of services in a simulated ubiquitous computing environment and was designed primarily to test the CBPMS's capacity to be integrated into a complex network and application environment with a requirement for near real-time responses to policy requests. The second of these experiments involved using the CBPMS to model the management of an Internet news site.

In both cases the modular nature of the CBPMS design allowed the quick construction of proxy clients to serve as the CBPMS's PDP interface for communicating with the applications. The simplicity of the CBPMS decision making interface – a single *decision* function – and the fact that it uses the well-known and widely supported XML-RPC web-service standard, greatly eased the task of modifying applications to make them policy-aware. The inclusion in the CBPMS of a generic session management proxy also proved invaluable for minimising network bandwidth and allowing the easy collation of state information from across several data sources. The first experiment required the CBPMS to deliver policy decisions in less than a second to several different client applications, two of which used an asynchronous Content Based Network, while the others communicated synchronously through TCP/IP connections. The CBPMS had to aggregate context information from across these clients in order to build a context vector and it had to deliver responses to requests in less than a second – according to the requirements of the application. Despite the small number of minor problems and teething difficulties encountered, once corrected, the CBPMS, running on a simple shared web-server account, with approximately 200 policy rules configured, was able to deliver a sustained rate of over 10 policy decisions per second, with an average response time of less than 20ms. This configuration proved capable of efficiently managing the physical security and presence management requirements of a realistic model of the community of users of a large building in a research institute.

However, it is important to note that policy sets in large, complex organisations may grow several orders of magnitude larger than that used in this experiment. From experiment 3, described in section 5.5, it is also clear that the efficiency of the system is

strongly affected by the distribution of policy rules amongst communities and the choices of policies in the experiments are not necessarily representative of the situations that one might see in practice. Real deployments might be expected to contain more exceptions and non-paradigmatic cases needing special rules, due to reality's well known tendency to be messy. The system's architecture has been designed with the assumption that most rules will not be defined in or near the root community of large organisations. If this assumption turns out to be false, the potential would arise for the root community servers to become bottlenecks in a distributed deployment. However, adding processing power and network bandwidth to those nodes is likely to be sufficient for this as there is no indication of a super-linear growth of processing complexity, or bandwidth as policy sets grow. It is impossible to know for sure whether the model and architecture will scale adequately to deal with the largest, most complex and demanding policy applications, without applying it in practice over a protracted period. However, there is also no evidence that anything within the model is likely to prove an insurmountable bottleneck to its large-scale deployment.

In addition to the network integration, each application of the CBPMS to a new domain required the construction of hierarchical resource models to represent the resources of the domain. In all of the experiments described here, it proved easy to identify useful hierarchical structures that allowed relatively efficient distribution of authority and specification of policies. As well as the experiments described in this thesis, the CBPMS has been applied to several other domains, including the management of very different types of resource, such as the radio-frequency spectrum [feeney07], in every case the task of constructing the resource models and configuring the Proxies, took no more than a few hours to achieve. However, while useful hierarchical sub-divisions could be readily identified, when modelling a complex resource such as the content embodied in a Content Management System (see section 5.4), multiple useful overlapping hierarchies could be identified, while in other cases, such as the models of physical security, more or less dimensions would have been preferable. The problem of constructing more flexible and expressive resource models is one of the areas identified as requiring further research, in section 6.3.1 below.

Goal 2: Model Generality, Flexibility, Dynamism and Decentralisation

The second goal of this research required a demonstration that the CBPM *“model is capable of capturing a wide range of organisational forms and that it is capable of*

efficiently capturing organisational change as well as being capable of modelling the self-managed groups and decentralised decision making hierarchies that exist in real world organisations”. The first two experiments and the case study, described in chapter 5, were designed to validate this goal. The examples were chosen so as to provide a wide range in terms of the size, formality, decentralisation of decision making, dynamism and complexity of organisational forms and distribution of authority, as depicted in table 6.1.

	Size	Formality	Decentralisation	Dynamism	Complexity
E1	1000s of Actors, ~50 communities, ~50 policy authors	Mixed: Mix of formal (e.g. health and safety legislation) and informal – working group norms.	Medium: limited self-management of Research Teams, subject to constraints defined by multiple higher authorities (legislation, university management, building management)	Stable: Most rules, structures and groups defined in advance and stable throughout, only lower level groups (research teams) exhibited structural changes	Medium: Simple hierarchy for managed doors; distributed control over presence information.
C1	Millions of Actors, 100's of communities, handful of policy authors	Formal: exhaustive set of rules, encoded as constitution, legislation and case history.	Hierarchical: power concentrated in the higher institutions and only very limited self-management and user input into policy formation.	Dynamic: although state structures are normally fairly stable over time, the case-study looked at their evolution	Complex: federations; multiple institutions with over-lapping and co-dependant authority;
E2	About 20 actors, 6 communities, 20 policy authors	Informal: rules based on informal norms established within groups	Non-hierarchical: Extensive self-management of working groups and universal input of stakeholders into decisions.	Punctuated Equilibrium: Stable structures over a period of time, with episode of extensive remodelling	Simple: flat hierarchy with few limits on authority.

Table 6.1 Dimensions of Variation in Organisational Examples Examined

The first experiment, E1, a simulation of the management of physical security and presence information in a ubiquitous computing environment, required all of the building’s potential users and stakeholders to be included in the organisational model. It required a variety of different decision making methods and structures to be implemented in different communities – with several bodies exercising overlapping authority over several of the resources . The organisational form in this experiment was relatively stable at higher levels and much more fluid at lower levels (i.e. research groups) – with numerous changes being applied to it to reflect changes in the simulated scenario. There was little difficulty in adapting the model to reflect these various organisational changes and, what’s more, all of the users of the system were easily able to comprehend the organisational model that was being applied.

The organisational scenario chosen for the case study, C1 presented in section 5.3 involving the creation of a community based model of the major political institutions of the Irish state, was selected as it provided a good example of a highly different type of organisation – one that was much larger and more complex than anything that could be reasonably studied experimentally; one that was more formal and more hierarchical, more bureaucratic and traditional in structure, with less devolution of authority and dynamism, and with structures defined in accessible and transparent documents; and finally one that the author was familiar with. Choosing a real state, with real institutions, also allows the model to be verified by others, since the constitution, statute books and legal precedents of the organisation, which between them describe the basic structures and policies of the state, are in the public domain. The case-study did not set out to model every aspect of the state, instead it limited itself to encoding the high-level relationships between the major institutions, in terms of their decision making authority, the scope of their authority and the hierarchical authority relationships which existed between them. The case study assumed, in line with the basic approach of this work that the details of each community's operations could be worked out at lower levels of the organisation, after deployment, once the basic structures and their relationships had been established.

Furthermore, rather than simply modelling the institutions, the case-study illustrated how the CBPMS could be used to build up such a model over time – how the various political changes that might come about within a group might be encoded as changes to the model. The seamless and efficient way in which the major changes were accommodated in the model, whereby 30 primitive invocations were sufficient to model the high-level changes that took place in a narrative which moved the group from a Hobbesian state of nature into a basic model of the Irish state, illustrates the capacity for the model to handle change efficiently. Furthermore, as these changes were engineered to a fabricated narrative, the dynamism of the model was further tested by applying it to actual changes that had taken place in the state – namely its entry into the EU (or the EEC as it was in 1973) and how the institutions of this federal arrangement could be accommodated by the model while preserving an accurate model of the authority relationships across the existing organisation.

The second experiment, C2, again focused on the modelling of a very different organisation – a voluntary Internet community. As the previous experiments mostly

used change-scenarios proposed by the author, they do not provide an objective change narrative against which to assess the efficiency with which the CBPM manages change, this example looked at the modelling of a real world community during its actual operation. The relatively small size of the community, its dynamic nature, fluid membership and the fact that almost all of its decision making took place online, on public mailing lists, as well as the fact that the community's administrators were happy to take part in the research, meant that this proved a very good source of data. Furthermore, since the administrators had previously been using a role-based access control model, it proved possible to compare their perceptions of the two models and how successfully they captured the nature of the organisation. It should be noted, however, that these results only related to the nature of the organisational models and did not compare other aspects of the two systems (since they were so different in other ways that comparisons would not be valid) and that the sample size was small. Nevertheless, given the relative rarity of people who have some familiarity with role-based policy management, finding a cooperative real world organisation that could provide even this small number of people with some experience was fortunate. In addition to modelling the organisation and its various rules over a period of this year, the models produced were also validated by the administrators ensuring that the CBPMS model fulfilled their requirements and successfully enforced their policies. These challenges were met with few problems and the system experienced little problem in efficiently modelling a fairly major structural change to the organisation in a controlled way with only localised disruption to the model.

It would be impossible to prove that every conceivable organisational structure, or that every conceivable organisational mutation could be captured by any particular model. Furthermore, the experiments focused, by necessity, on the high-level aspects of the organisational model, the distribution of authority, rather than the precise details of policy authoring. Thus, it is possible that deployment in large real-world organisations, with complex policy requirements could reveal that certain requirements could not be adequately modelled, or that the organisational structure imposed by the system might prove unwieldy and inefficient for the specification of important requirements. Nevertheless, the fact that the experiments looked at a diverse a range of organisational forms, differing widely in size, dynamism, formality, centralisation, hierarchy and complexity, and the fact that they were drawn from real-world or realistic scenarios and

the fact that one of them was examined over a protracted period and validated by the community's administrators meant that the CBPMS was presented with a reasonable sample of the organisational forms that exist in real world organisations. In addition to the experiments described in this thesis, the CBPMS has been deployed to model organisational forms in several other contexts, including, the over-lapping spheres of authority of telecommunications regulators and service providers [feeney07], collaborative working environments [feeney04b] and "trust-based organisations" [feeney05]. Furthermore, appendix 5 includes an explanation of how the CBPMS can be used to capture some of the more commonly cited properties that are required of PBM systems. The fact that the CBPMS proved able to efficiently capture and model the organisational structures, distribution of authority and operational rules, in all these domains suggests that it is likely to be capable of handling a wide range of organisational forms.

Goal 3: CBPMS Advantages

The third and final goal of this research was to demonstrate that the community model in the CBPMS could provide certain concrete advantages over current role-based policy systems. The purpose of the fourth and final experiment was to evaluate this goal. A model of an organisational structure, expressed as a role hierarchy, was taken from the document containing the only proposed standard in the area and used as the basis on which to form the comparison, avoiding the potential problem of constructing a model to suit the thesis. The algorithm followed by the role-based model was constructed in order to provide as fair a representation of the state of the art as possible and as accurate a comparison, wherever probability values were arbitrarily assigned to variables, they were varied over the simulation runs and a broad range of different assumptions was adopted for the various simulations. In each case, even when the worst assumptions possible for the CBPMS were adopted, it out-performed the role-based model in terms of the expense of its policy searching and in terms of the probability of a newly authored policy conflicting with a policy already present in the policy set.

These particular criteria were chosen for comparison because they are low-level enough to be measurable and because they are also obviously advantageous. Minimising the size of the policy set that the PBM must evaluate in order to return a decision for any given context is important firstly because it minimizes the amount of processing that the system must do – and although modern machines are fast, policy sets are known to

grow quickly once deployed and PBMs may be deployed to manage real-time systems where response time is critical, so this remains an important metric. However, perhaps more importantly, the reduction in the size of the policy search space is also experienced by the policy author. This limits the cognitive load of policy authors (cognitive load being, in this case, equal to the number of facts and the size and complexity of the ontology that must be understood in order to successfully apply changes). Since the complexity of policy authoring and analysis has proved to be one of the impediments to the widespread deployment of PBM in practice [jude01], a system which has the ability to minimise the number of policies that might interfere with a new policy, and thus must be considered by a policy author, could prove tremendously useful in overcoming that challenge. The results of the final experiment show that the CBPMS might be expected to reduce the number of policies that need to be considered by an average policy author by a factor of more than ten. This feature, therefore, has the potential to prove of major practical benefit to PBM technologies.

The second concrete measure used was the probability of conflicts. Again the results showed that the CBPMS was consistently much less likely to experience irresolvable run-time policy conflicts than the role-based model. Once again, this is a consequence of the distribution of resource authorities and the hierarchical nature of the *decision* algorithm. Not only was the frequency of conflicts significantly reduced, but the CBPMS was shown to be able to identify the organisational source of the problem, and even to analyse and suggest possible solutions. Considering the fact that handling policy conflicts has been one of the major problems that researchers in the area have grappled with, any system which has the capacity to eliminate large numbers of conflicts and suggest possible solutions to the remaining ones, has a clear advantage over its competitors.

Finally, in constructing this model the number of steps and the complexity of the procedures required to construct the role based hierarchy and that required to construct the community based hierarchy was compared. There was no difference between the two – the only procedure in the CBPMS which does not have an equivalent in the RBPMS was the delegation of authority. Other than that, they are exactly as complex to configure and specify, the semantics of the CBPMS depend upon the internal state that it maintains in terms of the authorities it has delegated and that have been delegated to it. However, the CBPMS can be configured to make delegation implicit in the definition

of policy – as was carried out in the random distribution used in the third experiment. This meant that the two systems were identical in terms of configuration and specification, the extra step of explicit delegation only needing to be included if its semantics are required. Therefore, this suggests that the practical advantages introduced by the CBPMS are not off-set by a concomitant increase in complexity of specification.

6.2 Contribution

CBPM Model

The CBPM model is a major contribution of this work. It represents an attempt to define a formal system which allows an organisation to create a model of itself for the purpose of applying PBM solutions. It consists of a set of community structures, a decision algorithm and a set of 14 primitives which change the state of the system. It is capable of modelling a wide range of organisational changes, from community formation, specialisation, merging and splitting to changes to distribution of authority over resources as well as the definition of policy rules, all with well-defined semantics. Although there is no guarantee that the CBPM model will prove sufficient to model all existing organisational forms, and the model will undoubtedly have to be refined over time, it represents a significant first-step in formalising a top-down organisational modelling approach that allows for the rules to be defined in a bottom-up, decentralised manner.

Analysis of Modelling Approaches

The second major contribution of this work is an analysis which suggests that current approaches to organisational modelling in PBM may be limited in their capacity to express the dynamic, decentralised nature of many real world organisations. It argues that role-based approaches are based on an abstraction that is too simple and too low-level for the task of effectively mapping an organisation's high-level goals into a configuration of their PBM system. It further suggests that approaches to modelling organisations which rely too much on centralised planning and analysis of policies and which do not incorporate sufficient support for flexibility, self-management, progressive accumulation of policy sets and decentralised decision making into their systems will continue to face great, even insurmountable difficulties, in successfully maintaining a configuration which represents a good match to the organisation's goals. Rather than using low-level bottom-up abstractions in a centralised, top-down modelling process,

this work suggests that a top-down recursive analysis of the organisation into units linked to function, with the operations of the units defined through a distributed, bottom-up progressive grounding process, would be an approach that is both more likely to lead to a policy configuration that will be a good match to the organisation's goals, and will also tend to be a more accurate model of how the organisation develops in practice.

6.2.1 Minor Contributions

Novel Conception of Delegation

The concept of delegation employed by the CBPMS is unique. Rather than being a temporary transfer of permissions, from one individual to another, CBPMS delegation involves a *bundle of authority* being transferred from one organisational unit to another. This *bundle of authority* consists of a *resource authority*, which is a reference to a node on a hierarchical model of the managed resource, and a set of policies which are bound to the delegated resource. This concept of delegation, defines boundaries on how the resource may be used, without specifying exactly how it is to be used, leaving that up to the community to which the authority has been delegated. This *delegation of authority* allows the organisation to decentralise its policy authoring process in a controlled way and it means that the CBPM model contains a map of how authority over resources is distributed throughout the various communities that make up the organisation. This concept of delegation is counter-posed with the concept of escalation, to form the delegation-escalation cycle, a novel approach to identifying the appropriate decision making unit within an organisation – allowing policy conflicts to feed information back into the organisational model providing an impetus towards progressive refinement of the rules.

Distributed Service Oriented Policy Architecture

The service oriented architecture of the CBPMS is based upon a decentralised, federated, network of simple services. It can be distributed across servers controlled by multiple independent yet inter-related organisations in a controlled way, allowing each organisation to distribute authority over its resources to other organisations, while ensuring that it maintains the authority to specify mandatory rules over how its resources are used. It represents an innovative offering in terms of an architecture

allowing multiple autonomous organisations and federations can exercise joint control over shared resources through a PBM system.

PBM Metrics

This thesis has also proposed a novel set of metrics that should be considered of importance to PBM systems, namely the probability of irresolvable policy conflicts and the size of the policy set that needs to be evaluated in order for the system to arrive at a decision. These metrics and their importance are explained in sections 5.5 and 6.1.

6.3 Further Work

There remain significant challenges to realising the promise of the decentralised approach to policy-authoring that this thesis proposes. There also remain several novel features of the CBPM model which have yet to be fully evaluated in empirical tests to see how they perform and whether they do exhibit the advantages that are hypothesised for them. Lastly, there are several areas where the CBPMS might be expanded upon in order to make its modelling capability more powerful and easier to apply to novel domains. The following sections mention some of the more important areas for further research in the field.

6.3.1 Resource Models and Ontologies

The CBPMS, as it has been described in this thesis uses a simple partial ordering of ordered pairs to model resources, based upon a hierarchical tree of action nodes and a tree of target nodes. However, this is a relatively simple resource model and while it can be applied to virtually any resource, some resources map poorly to a pair of simple hierarchical trees. Although it is relatively trivial to adapt the resource model by adding new dimensions (or removing them) it is not certain that this will be sufficient for modelling the requirements and the axes upon which all resources might be divided up for administration and indeed it is highly likely that a simple comparison of nodes will not be sufficient to capture the essential dimensions along which some resources are divided up into “parcels of authority”. Therefore, there is a need to investigate ways of expressing more complex relationships between nodes on resource authority trees, and more importantly, clear semantics to allow the system to ensure that policies with non-overlapping scopes will not interfere with each other.

One promising area of research when considering the semantic richness of resource models, covers the various proposals for semantic policy languages. These are policy

languages, such as KAoS [bradshaw03, uszok03] and Rei [kagal03], based upon the DAML and RDF-S semantic web languages respectively. Policies and resources described through such languages would allow for a tremendous amount of automated reasoning to be applied to the interaction of resources, policies and contexts. However, on the other hand, the task of constructing such ontologies is not simple and remains to be done for almost all conceivable applications.

6.3.2 Integration of Multiple Policy Languages

One of the features of the CBPM model is that it does not mandate any particular policy language; this means that it could be used to manage policies specified in multiple different application specific policy languages. As long as all the languages obey the required semantics of the CBPM model, they can be accommodated. However, in order for the CBPM algorithm to work correctly, the Policy Reasoning Service (PRS) must be able to tell whether two policy decisions conflict with each other, and when the policy decisions are specified according to a different language with different semantics the PRS will need to understand the semantics of both languages well enough to correctly decide whether they policies conflict. Although the CBPMS has been configured to use multiple specification languages, these were very simple examples, consisting of XACML policies alongside a simple permission specification language and a simple obligation specification language where the task of analysing policy decisions for conflicts was relatively trivial. Therefore, this is another area that requires research and once again the use of ontologies to analyse the various complex mappings between concepts in policy languages suggests itself as a useful approach.

6.3.3 Policy Authoring Dialectics

One of the hypothesized benefits of the CBPMS is its support for what is called the *policy authoring dialectic*. This concept refers to the hypothesis that basing the organisational model on delegating authority from higher levels of the organisation, and then escalating requests for authority from lower levels is likely to help the organisation to reach an optimal policy configuration over time. This hypothesis is based upon the idea that human organisations build up their policy set over time in response to problematic situations. If members of a community are forced to escalate too many of their routine requests to administrators of higher level for approval, the higher level administrators will be motivated to delegate the required authority, or relax the relevant

policy, in order to reduce the number of routine tasks that he must carry out as part of his responsibilities. On the other hand, delegating authority too liberally will lead to more conflicts between unrelated communities which must be solved by the higher-level administrator, once again meaning that the administrator's job will be easier, the more precisely he defines the mandates of the communities beneath him – delegate too much or too little and his workload increases. The policy dialectic hypothesis holds that these twin pressures, which will operate on all policy authors, will tend to mean that the policy set will tend towards more complete and more stable configurations.

The PBM metrics discussed in section 6.2.4 also suggest a possible way of evaluating the hypothesis. Another approach that has been proposed by researchers in Trinity College Dublin [lewis07] involves using promise theory to measure the stability of various policy configurations, while using genetic algorithms to search the space of policy configurations to evaluate their stability. However, such work is still in its infancy.

6.3.4 Realizing Decentralised Policy Authoring

Although this thesis has demonstrated that the CBPMS can support the modelling of self-managing groups with decentralised decision making processes, it has not demonstrated that it is possible in practice for policy authoring to be distributed throughout organisations. Traditionally, policy authoring has been seen as very much confined to specialist security administrators with the knowledge and skills to analyse the potential consequences of changes to policy sets in minute detail. If it is to be decentralised, then this requires that more people, with generally lower level of skills than senior security officers would be expected to possess, would be authoring policies (note that these do not have to be non-technical end-users, decentralising policy authoring to include local technical administrators would represent a significant broadening of policy authoring from most current approaches).

The CBPM model does go some way towards solving the problem. By limiting the size of the policy search space, as explained above, the complexity of the potential consequences that policy authors have to consider is greatly reduced. Furthermore, there is no potential for a policy to impact upon a resource that has not been delegated to the community, so the administrators of each community would not have to worry about stray policy specifications interfering with mission-critical systems. Still, beyond

this lessening of the cognitive load, there is still a need for tools, techniques and visualisations schemes which allow administrators the ability to master the inherent complexity of the domain. There is ongoing research in Trinity College Dublin into visualisation schemes and information flows between humans and policy systems, and a prototype community visualisation scheme has been produced [christos].

6.4 Conclusion

Recent years have seen explosive growth in online communities, through social networking sites, blogs, wikis, instant messaging applications and many other new technologies that profit from what is known as ‘user-generated content’. Meanwhile, traditional organisations have started to embrace many of these technologies in an effort to move towards the vision of the agile organisation. In general, these developments illustrate the benefits of empowering users and decentralising decision making. However, in order for this technological trend to be integrated into the core business activities of organisations, there is a need for a means of controlling the interactions and collaborations so that they help to achieve the high-level goals of the organisation. In order to take full advantage of the participants’ creativity and intelligence, these controls and constraints can also profit from maximising user-involvement in decision making, incorporating greater democracy, participation and a sense of collective ownership into the collaboration. Furthermore, they will need to be applied in a way that is flexible to change, organisational diversity and is highly responsive to changes in high level goals.

The Community Based Policy Management approach developed in this thesis represents a promising initial step in addressing this challenge. It is a system designed to facilitate flexible, robust, decentralised management of collaborative organisations. Although the CBPMS implementation, described in this thesis, remains a prototype, there are concrete plans underway for the development of a commercially viable implementation. Proposals for commercialisation funding have been submitted to Enterprise Ireland and several research projects which aim to develop its principles further are already underway at Trinity College Dublin, with particular focus on policy visualisation approaches and resource modelling. Finally, a proposal has been short-listed by the European Research Council for funding to perform a large-scale longitudinal study of the operations of real world communities through the CBPM model.

Bibliography

- [abadi93] Abadi, M., Burrows, M., Lampson B., Plotkin G. (1993) *A Calculus for Access Control in Distributed Systems*. ACM Transactions on Programming Languages and Systems, vol. 4(15), pp. 706-734, September 1993.
- [abrams93] Abrams M. D. (1993) *Renewed Understanding of Access Control Policies* in Proceedings of the 16th National Computer Security Conference, Baltimore, Maryland, U.S.A., pp. 87-96, 20-23 September 1993
- [ahn99] Ahn, G., Sandhu R. (1999), *The RSL99 Language for Role-Based Separation of Duty Constraints*, in Proceedings of the Fourth ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, ACM Press, pp. 43-54, 28-29 October 1999.
- [alkahtani03] Al-Kahtani, M. A., Sandhu, R. (2003), *Induced Role Hierarchies with Attribute-Based RBAC*, in Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT), Villa Gallia, Como, Italy, June 2003
- [alkahtani04] Al-Kahtani, M. A., Sandhu, R. (2004), *Rule-Based RBAC with Negative Authorization*, 20th Annual IEEE Computer Security Applications Conference (ACSAC'04), 2004
- [alotaiby04] Alotaiby, F. T., Chen J. X. (2004), *A Model for Team-based Access Control (TMAC 2004)*, in Proceedings of the international Conference on information Technology: Coding and Computing (Itcc'04) Volume 2 - Volume 2 April 05 - 07, 2004.
- [ansi92] ANSI, *ANSI Database Language SQL*, (1992) ANSI X3.135-1992 American National Standards Institute, 1992
- [ansi04] ANSI/INCITS, (2004) *Role Based Access Control*, ANSI/INCITS Standard 359-2004
- [anderson01] Anderson R., Lee, J.-H, Stajano, F. (2001) *Security Policies* in Advances in Computers, Academic Press. vol. 55, 2001.
- [bacon02] Bacon, J., Moody, K., Yao, W. (2002), *A model of OASIS role-based access control and its support for active security*, in ACM Transactions on Information and System. Security. 5, 4 (Nov. 2002), pp 492-540.
- [baliosian04] Baliosian J., Serrat, J. (2004) *Finite State Transducers for Policy Evaluation and Conflict Resolution*, in Fifth IEEE International Workshop on Policies for Distributed Systems and Networks 2004.
- [bandara03] Bandara A. K., Lupu, E, Russo, A. (2003), *Using Event Calculus to Formalise Policy Specification and Analysis*, in Fourth IEEE International Workshop on Policies for Distributed Systems and Networks 2003.
- [bandara04] Bandara, A. K., Lupu, E, Russo, A., Moffett, J. (2004), *A Goal-based Approach to Policy Refinement* in Fifth IEEE International Workshop on Policies for Distributed Systems and Networks 2004.
- [bandara05] Bandara A. K., Lupu E., Russo A., Dulay N., Sloman M., Flegkas P., Charalambides M., Pavlou G., (2005), *Policy Refinement for DiffServ Quality Of*

Service Management in proceedings of the IEEE/IFIP Integrated Management Symposium (IM'2005), Nice, France, IEEE, May 2005.

[barka00a] Barka E., Sandhu. R., (2000a), *A role-based delegation model and some extensions* in Proceedings of the 23rd National Information Systems Security Conference, Baltimore, MD, October 2000.

[barka00b] Barka E., Sandhu. R. (2000b), *Framework for role-based delegation models* in Proceedings of the 16th Annual Computer Security Applications Conference, New Orleans, Louisiana, December 2000.

[barker00] Barker, S. (2000), *Security Policy Specification in Logic*, in Proceedings of the International Conference on Artificial Intelligence (ICAI00), Las Vegas, Nevada, USA, pp. 143-148, 26-29 June 2000.

[barrett04] Barrett, R. (2004) *People and Policies: Transforming the Human-Computer Partnership* in 5th IEEE International Workshop on Policies and Distributed Systems and Networks, IEEE, 2004

[beer85] Beer, S. (1985), *Diagnosing the System for Organizations*, Wiley, Chichester, US, 1985

[bell76] Bell, D. E., LaPadula, L. J. (1976) *Secure Computer System: Unified Exposition and Multics Corporation*, Mitre Corporation, Massachusetts, USA, March 1976

[belokosztoloszki02] Belokosztoloszki, A., Moody, K. (2002), *Meta-Policies for Distributed Role-Based Access Control Systems*, in Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, California, USA, 106-115, 2002.

[bertino98] Bertino, E., Bettini, C., Ferrari, E., Samarati P., (1998) *An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning*, in ACM Transactions on Database Systems, vol. 23(3), pp. 231-285, September 1998.

[bertino01] Bertino, E., Bonatti, P. A., Ferrari, E., (2001) *TRBAC: A Temporal Role-based Access Control Model*, in ACM Transactions on Information and System Security, 4(3), 191-233. 2001

[bertino03] Bertino, E., Catania, B., Ferrari, E., Perlasca, P., (2003) *A logical framework for reasoning about access control models*, in ACM Transactions on Information and System Security (TISSEC), 6 (1): 71-127 2003

[bertino05] Bertino, E., Mileo, A., Proveti, A, (2005) *PDL with Preferences*, in Proceedings of the Sixth IEEE international Workshop on Policies For Distributed Systems and Networks (Policy'05) - June 2005.

[blau55] Blau P. (1955), *The Dynamics of Bureaucracy*, University of Chicago Press, Chicago

[blaze98] Blaze, M., Feigenbaum, J., Keromytis, A., (1998) *Keynote: Trust Management for Public-Key Infrastructures*, in Proceedings of the Security Protocols International Workshop, Cambridge, England, Springer-Verlag LNCS, pp. 59 - 63, April 1998.

[blaze99] Blaze, M., Feigenbaum, J. Ioannidis, J., Keromytis, A.D, (1999) *The Role of Trust Management in Distributed Systems Security*, in Secure Internet Programming:

Security Issues for Mobile and Distributed Objects. New York, NY, USA, Springer-Verlag, pp. 185 – 210. 1999.

[boje93] Boje, D., Winsor, R. (1993), *The Resurrection of Taylorism: Total Quality Management's Hidden Agenda*, Journal of Organizational Change Management, Volume: 6 Number: 4, 1993

[boros00] Boros, S. (2000), *Policy-based network management with SNMP*, in Scientific Output of the University of Twente, The Netherlands, 2000.

[boudaoud01] Boudaoud, K., Guessoum, Z., McCathieNeville C., Dubois P., (2001) *Policy-based Security Management using a Multi-agent System*. In HP Openview University Association Plenary Workshop, Berlin Germany, 2001.

[bradshaw03] Bradshaw, J. M., Uszok, A., Jeffers, R. Suri, N. (2003), *Representation and reasoning for DAML-based policy and domain services in KAoS and Nomads*, in the Proceedings of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2003). Melbourne, Australia, New York, NY: ACM Press, 2003

[braverman] Braverman, H. (1974) *Labour and Monopoly Capital: The Degradation of Work in the 20th century*, Monthly Review Press, New York, 1974.

[campbell] Campbell, C.S., Kandogan, E.; November, A.; Barrett, R.; Maglio, P.P. (2005), *Policy: an experimental evaluation of policy-based administration in a city simulation* in Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, Page(s):119 – 122, 6-8 June 2005

[charalambides05] Charalambides, M., Flegkas, P., Pavlou, G., Bandara, A. K., Lupu, E. C., Russo, A., Dulay, N., Sloman, M., Rubio-Loyola, J. (2005), *Policy Conflict Analysis for Quality of Service Management*, in Proceedings of the Sixth IEEE international Workshop on Policies For Distributed Systems and Networks (Policy'05) June 2005.

[chen95] Chen, F., Sandhu R. (1995), *Constraints for Role-Based Access Control*, in Proceedings of the First ACM/NIST Role Based Access Control Workshop, Gaithersburg, Maryland, USA, ACM Press, November 1995.

[chomicki00] Chomicki J., Lobo. J. (2000), *A Logic Programming Approach to Conflict Resolution in Policy Management*. In 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'2000), Breckenridge, Colorado, April 2000

[chomicki01] Chomicki J., Lobo J., (2001) *Monitors for History-Based Policies*, in Proceedings of second IEEE International Workshop on Policies for Distributed Systems and Networks. Springer-Verlag, LNCS, pp. 57–72. 2001,

[clark87] Clark, D. D., Wilson, D.R. (1987), *A Comparison of Commercial and Military Computer Security Policies*, in Proceedings of the IEEE Symposium on Security and Privacy, 1987.

[cobb96] Cobb, L., (1996) *Minimalist Modeling Methodology*, Aetheling Consultants 1996. available at: <http://www.aetheling.com/docs/Minimal.html> accessed June 2006

[cohen02] Cohen, E., Thomas, R. K., Winsborough, W., Shands, D. (2002), *Models for coalition-based access control (CBAC)*, in Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies (Monterey, California, USA, June 3-4, 2002). (SACMAT '02)

- [coutinho05] Coutinho, L., Sichman J. M., Boissier, O. (2005), *Modeling organization in MAS: a comparison of models*, in Proceedings of the 1st. Workshop on Software Engineering for Agent-Oriented Systems (SEAS'05) Uberlândia, MG, October 2005.
- [crampton02] Crampton, J., Loizou, G. (2002) *SARBAC: A New Model for Role-Based Administration*. Technical Report BBKCS-02-09, Birkbeck College, University of London, July 2002.
- [crampton03] Crampton, J., Loizou, G. (2003), *Administrative scope: A foundation for role-based administrative models*, in ACM Transactions on Information and System Security (TISSEC), 6(2):201–231, May 2003.
- [crampton06] Crampton, J., Khambhammettu, H. (2006), *Delegation in role-based access control*, in Proceedings of 11th European Symposium on Research in Computer Security, 174-191, 2006.
- [crook02] Crook, R., Ince, D., Nuseibeh, B. (2002), *Towards an Analytical Role Modelling Framework for Security Requirements*, in Proceedings of 8th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ-02), Essen, Germany, 9-10 September 2002.
- [cuppens04] Cuppens, F., Miège, A. (2004), *AdOrBAC: an administration model for Or-BAC*. Computer Systems: Science & Engineering 19(3): CRL Publishing, Leicester UK
- [damianou01] Damianou, N., Dulay, N., Lupu, E., Sloman, M. (2001), *The Ponder Policy Specification Language*, in Proceedings of the Policy Workshop 2001, HP Labs, Bristol, UK, Springer-Verlag, 29-31 January 2001.
- [damianou02a] Damianou, N., Bandara, A., Sloman, M., Lupu E. (2002a) A Survey of Policy Specification Approaches, Department of Computing, Imperial College of Science Technology and Medicine, London, 2002.
- [damianou02b] Damianou, N., (2002b) *A Policy Framework for Management of Distributed Systems*, PhD Thesis, Department of Computing, Imperial College. London, U. K. 2002.
- [darimont96] Darimont, R. Van Lamsweerde, A. (1996), *Formal Refinement Patterns for Goal-Driven Requirements Elaboration*, in 4th ACM Symposium on the Foundations of Software Engineering (FSE4): 179-190. 1996.
- [dayal95] Dayal, U., Hanson, E., Widom, J. (1995), *Active database systems*, in Modern Database Systems: the Object Model, interoperability, and Beyond, W. Kim, Ed. ACM Press/Addison-Wesley Publishing Co., New York, NY, 434-456, 1995.
- [dealbuquerque05] de Albuquerque, J. P., Krumm, H., de Geus, P. L. (2005), *Policy Modeling and Refinement for Network Security Systems*, in Proceedings of Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 24-33, 2005.
- [debruijn05] de Bruijn, J., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L., Kifer, M., Fensel, D. (2005) *The Web Service Modeling Language WSML*. WSML Deliverable D16.1v0.2, 2005. Available online at: <http://www.wsmo.org/TR/d16/d16.1/v0.2/>
- [dignum02] Dignum, V., Meyer, J.-J., Weigand, H., Dignum, F. (2002), *An organizational oriented model for agent societies*, in Proceedings of International

Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02), at AAMAS, Bologna, Italy, 16 July 2002.

[dignum05] Dignum, V., Vazquez-Salceda, J., Dignum, F. (2005). *OMNI: Introducing social structure, norms and ontologies into agent organizations*. In Bordini, R. H., Dastani, M., Dix, J., and Seghrouchni, A. E. F., editors, *Programming Multi-Agent Systems: Second International Workshop ProMAS 2004*; New York, NY, July 20, 2004; Selected Revised and Invited Papers, volume 3346 of LNAI, pages 181-198, Berlin Heidelberg. Springer.

[dijker96] Dijker, B. (Ed) (1996), *A Guide to Developing Computer Policy Documents*, SAGE Short Topics in Systems Administration book 2, 1996.

[dmtf07] Distributed Management Task Force Inc. (DMTF) (2007), *Common Information Model (CIM) Specification, version 2.2*, available online: <http://www.dmtf.org/spec/cims.html>, 14 June 1999. accessed July 2007

[dodgson58] Dodgson, C. L. (1958) *A method for taking votes on more than two issues*. Clarendon Press, Oxford, 1876. Reprinted in (Black, 1958) with discussion

[doeppner83] Doeppner, W., Glacalone, T. (1983), *A formal description of the UNIX operating system*, in *Proceedings of the second annual ACM symposium on Principles of distributed computing*, 1983.

[dulay01] Dulay, N., Lupu, E., Sloman, M., Damianou, N. (2001), *A policy deployment model for the Ponder language*, in *Proceedings of IEEE/IFIP International Symposium on Integrated Network Management*, 2001 Seattle.

[dunlop03] Dunlop, N., Indulska, J., Raymond, K., (2003) *Methods for Conflict Resolution in Policy-Based Management Systems*, in *Proceedings of the Seventh International Enterprise Distributed Object Computing Conference (EDOC'03)*, 2003

[dym02] Dym, C.L., Wood, W.H., Scott, M.J. (2002), *Rank Ordering Engineering Designs: Pairwise Comparison Charts and Borda Counts*, *Research in Engineering Design*, 13, 236-242, 2002.

[edwards79] Edwards R. (1979) *Contested Terrain: the Transformation of the Workplace in the 20th century*, Heinmann, London 1979.

[elkalam03] El-Kalam, A., Benferhat, S., Mieke, A., El-Baida, R., Cuppens, F., Saurel, C., Balbiani, P., Deswarte, Y., Trouessin, G. (2003), *Organization based access control*, in *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, page 120, Washington, DC, USA, 2003.

[esteva04] Esteva, M., Rosell, B., Rodriguez-Aguilar, J. A., Arcos, J. L. (2004), *AMELI: an agent-based middleware for electronic institutions*, in *Proceedings of The International Conference on Autonomous Agents and Multiagent Systems*, pages 236-243, 2004

[eyers06] Eyers, D.M., Bacon, J., Moody, K. (2006), *OASIS role-based access control for electronic health records*, in *IEE Proceedings of Software*, February 2006

[feeney04a] Feeney, K., Lewis, D., Wade, V. (2004a), *Policy Based Management for Internet Communities*, in *Proceeding of IEEE 5th International Workshop on Policies for Distributed Systems and Networks*, IBM Thomas J Watson Research Center, New York, USA, pp 23-34, June 2004

- [feeney04b] Feeney, K., Lewis, D. (2004b), *Community Based Policy Management for Smart Spaces*, in Proceedings of 2nd International Workshop on Managing Ubiquitous Communications and Services (MUCS), Dublin, Ireland, December 13-14, 2004
- [feeney05] Feeney, K., Quinn, K., Lewis, D., O'Sullivan, D., Wade, V. (2005), *Relationship-Driven Policy Engineering for Autonomic Organisations*, in Proceedings of 6th IEEE International Workshop on Policies for Distributed Systems, Stockholm, Sweden, pp 89-98, June 2005,
- [feeney07a] Feeney, K., Lewis, D., Argyroudis, P., Nolan, K., O'Sullivan, D. (2007a), *Grouping Abstraction and Authority Control in Policy-based Spectrum Management*, in Proceedings of the IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN'07), April 2007. [feeney07b] Feeney, K., Lewis, D., Wade, V., (2007b) Roles Considered Harmful in Policy-based Management for Dynamic Organisations, in Proceedings of the IEEE International Symposium on Integrated Network Management, (IM 07), May 2007
- [felleman91] Felleman, D.J., Van Essen, D.C. (1991), *Distributed hierarchical processing in the primate cerebral cortex*, in Cerebral Cortex 1:1-47, 1991
- [ferber98] Ferber, J., Gutknecht, O. (1998), *A meta-model for the analysis and design of organizations in multi-agent systems*, in Proceedings of the Third International Conference on Multi Agent Systems, Paris, France, pages 128-135. IEEE Press. July 1998.
- [ferber03] Ferber, J., Gutknecht, O., Michel, F. (2003), *From agents to organizations: an organizational view of multi-agent systems*, in Giorgini, P., Meuller, J. P., Odell, J., editors, Agent-Oriented Software Engineering IV: 4th International Workshop, AOSE 2003, Melbourne, Australia, Revised Papers, volume 2935 of LNCS, pages 214-230, Berlin Heidelberg. Springer. July 15, 2003.
- [ferraiolo92] Ferraiolo, D.F., Kuhn, D.R. (1992), *Role Based Access Control* 15th National Computer Security Conference, 1992
- [ferscha99] Ferscha, A., Scheiner, C. (1999), *Collective Choice in Virtual Teams*, in Proceedings of the 8th Workshop on Enabling Technologies on infrastructure For Collaborative Enterprises. WETICE. IEEE Computer Society, Washington, DC, 96-101. June 1999.
- [freeman72] Freeman, J., (1972) *The Tyranny of Structurelessness*, Berkeley Journal of Sociology, pp. 151-165, Vol. 17, 1972-73
- [gama05] Gama, P., Ferreira, P. (2005), *Obligation Policies: An Enforcement Platform*, Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 203-212.
- [georgiadis01] Georgiadis, C.K., Mavridis, I., Pangalos, G., Thomas, R.K. (2001), *Flexible Team-based Access Control Using Contexts*, in Proceedings of the 6 ACM Symposium on Access control models and technologies, pages 21-27, May 2001
- [goh98] Goh, C., Baldwin, A. (1998), *Towards a More Complete Model of Role*, in Third ACM Workshop on Role-Based Access Control, pages 55-61, Fairfax, Virginia, October 1998.
- [golbeck03] Golbeck, J., Hendler, J., Parsia, B. (2003), *Trust Networks on the Semantic Web*, in Proceedings of the 12th International Web Conference (WWW03), Budapest, Hungary, May 2003.

- [golbeck04] Golbeck, J., Hendler, J. (2004), *Inferring Reputation on the Semantic Web*, in Proceedings of the 13th International Web Conference (WWW2004), New York, NY, USA, May 2004.
- [graham72] Graham, G., Denning, P. (1972), *Protection - principles and practice*, in AFIPS Spring Joint Computer Conference, pp. 40:417-429, 1972.
- [graham04] Graham, A., Radhakrishnan, T., Grossner, C., (2004) *Incremental Validation of Policy-Based Systems*, in Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004.
- [grandison00] Grandison, T., Sloman, M. (2000), *A Survey of Trust in Internet Applications*. IEEE Communications Surveys and Tutorials, vol. 3(4), Oct-Dec 2000.
- [grosof03] Grosof B., Poon, T. (2003), *SweetDeal: representing agent contracts with exceptions using xml rules, ontologies, and process descriptions*. In In Proceedings of the 12th World Wide Web Conference, Budapest, Hungary, May 2003
- [grosof07] Grosof, B., Kifer, M., Martin, D. L. (2007), *Rules in the Semantic Web Services Language (SWSL): An Overview for Standardization Directions*. Position Paper for W3C Workshop on Rule Languages for Interoperability. Available at: <http://www.w3.org/2004/12/rules-ws/paper/124/>. Accessed July 2007.
- [hawke05a] Hawke, S., Tabet, S., de Sainte Marie, C. (2005a), *Rule Language Standardization Report from the W3C Workshop on Rule Languages for Interoperability*, April 2005 <http://www.w3.org/2004/12/rules-ws/report/> Accessed January 2007.
- [hawke05b] Hawke, S. (2005b), *W3C Rule Interchange Format Working Group Charter* <http://www.w3.org/2005/rules/wg/charter> 2005, Accessed January 2007
- [hayton98] Hayton, R. J., Bacon, J. M., Moody, K. (1998), *Access control in an open distributed environment* in Proceedings of IEEE Symposium on Security and Privacy, pages 3--14, May 1998.
- [heckscher94] Heckscher, C. (1994), *Defining the post-bureaucratic type*, in Heckscher, C., Donnellon, A. (Eds), *The Post Bureaucratic Organization: New Perspectives on Organizational Change*, Sage, Thousand Oaks, CA, 1994.
- [herzberg00] Herzberg, A., Mass, Y., Michaeli, J., Naor D., Ravid, Y. (2000), *Access Control Meets Public Key Infrastructure, or: Assigning Roles to Strangers*, in Proceedings of the IEEE Symposium on Security and Privacy, Oakland, California, USA, 14-17 May 2000.
- [hine00] Hine, J., Yao, W., Bacon, J., Moody, K. (2000), *An Architecture for Distributed OASIS Services*, in Proceedings of Middleware 2000, New York, USA, Lecture Notes in Computer Science, Springer-Verlag, pp 107-123, April 2000.
- [hoagland98] Hoagland, J. A., Pandey, R., Levitt, K. N. (1998), *Security Policy Specification Using a Graphical Approach*. Technical report CSE-98-3, UC Davis Computer Science Department, 22 July 1998.
- [fan06] Hong Fan, Zhu Xian, Xing Guang-lin (2006), *A distributed role-based access control model for multi-domain environments*, Journal of Shanghai University (English Edition), Shanghai University Press, pp 134-141, vol. 10, no. 2 April 2006
- [hopfl06] Höpfl, H.M. (2006), *Post-bureaucracy and Weber's "modern" bureaucrat*, Journal of Organizational Change Management, Volume: 19 Issue: 1 Page: 8 – 21, 2006

- [horling04] Horling, B., Lesser, V. (2004), *Quantitative organizational models for large-scale agent systems*, in Proceedings of the International Workshop on Massively Multi-Agent Systems, pages 297–312, 2004.
- [horrocks04] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosz, B., Dean, M. (2004), *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission 21 May 2004. Available from <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>. Accessed July 2007.
- [hubner02] Hubner, J. F., Sichman, J. S., Boissier, O. (2002), *A model for the structural, functional, and deontic specification of organizations in multiagent systems*, in Bittencourt, G. and Ramalho, G. L., editors, *Advances in Artificial Intelligence: 16th Brazilian Symposium on Artificial Intelligence, SBIA November 2002*
- [ietf00] IETF Network Working Group (2000), *The COPS (Common Open Policy Service) Protocol*, IETF Request for Comments 2748. Available online at: <http://www.ietf.org/rfc/rfc2748.txt>.
- [imc05] IMC, Indymedia Ireland: (2005), *Indymedia Ireland to formalise its operating rules*. <http://www.indymedia.ie/article/70147> (accessed August 2006)
- [imc06] IMC Indymedia Ireland (2006), *Indymedia's Fifth Birthday – New Structures Adopted, Your Help Wanted*, November 26, 2006. Available at: <http://www.indymedia.ie/article/79150> (accessed September 2007)
- [iso00] ISO/IEC (2000) JTC1/SC7/WG17 N0127, *Information Technology – Open Distributed Processing – Reference Model: Enterprise Viewpoint Language*, July 2000
- [iso06] ISO/IEC (2006) JTC 1/SC 32 N xxxx 2006-06-01 ISO/FDIS 24707 ISO/JTC 1/SC 32/WG2 *ANSI Information technology - Draft Standard - Common Logic (CL) - A framework for a family of logic-based languages*.
- [jaeger00] Jaeger, T., Tidswell, J. (2000), *Rebuttal to the NIST RBAC model proposal*, in Proceedings of the fifth ACM workshop on Role-based access control, pp 65 – 66, 2000.
- [jajodia97] Jajodia, S., Samarati, B., Subrahmanian, V. S. (1997), *A Logical Language for Expressing Authorisations*, in Proceedings of the IEEE Symposium on Security and Privacy, pp. 31-42, May 4-7, 1997.
- [jajodia01] Jajodia, S., Samarati, P., Sapino, M. L., Subrahmanian, V. S. (2001), *Flexible Support for Multiple Access Control Policies*. ACM Transactions on Database Systems, vol. 26(2), pp. 214-260, June 2001.
- [jeannerod97] Jeannerod, M. (1997), *The Cognitive Neuroscience of Action*, Oxford, Blackwell 1997
- [jude01] Jude, M. (2001), *Policy-based Management: Beyond the Hype*. In Business Communications Review, available from <http://www.bcr.com/bcrrmag/2001/03/p52.asp>, pp 52-56, March 2001. Accessed July 2005.
- [kagal01] Kagal, L., Finin, T., Peng, Y. (2001), *A Delegation Based Model for Distributed Trust*, in Proceedings of the Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents, International Joint Conferences on Artificial Intelligence, August 2001.

- [kagal03] Kagal, L., Finin, T., Joshi, A. (2003), *A Policy Language for A Pervasive Computing Environment*, in Proceedings of the IEEE 4th International Workshop on Policies for Distributed Systems and Networks, June 2003
- [keeney05] Keeney, J., Carey, K., Lewis, D., O'Sullivan, D., Wade, V. (2005), *Ontology-based Semantics for Composability of Autonomic Elements*, in Proceedings of the Workshop on AI in Autonomic Communications at 19th International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, 30 July – 5th Aug 2005
- [koch04] Koch, C. (2004), *The Quest for Consciousness: A Neurobiological Approach*, Englewood, CO: Roberts and Co, 2004
- [kohler64] Kohler, I. (1964), *The formation and transformation of the perceptual world*, in Psychological Issues, 3, 1–173. 1964
- [kohli99] Kohli, M. and J. Lobo (1999). Policy Based Management of Telecommunication Networks. In Proceedings of the Policy Workshop 1999, HP Labs, Bristol, UK, 15-17 November 1999.
- [krishnamurthy02] Krishnamurthy, S. (2002), *Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects*, in First Monday, vol. 7, no. 6, June 2002, Available from: http://firstmonday.org/issues/issue7_6/krishnamurthy/ (accessed June 2004)
- [lampson71] Lampson, B. W. (1971), *Protection* in Proceedings of the. 5th Princeton Conference on Information Sciences and Systems, Princeton, 1971
- [lawrence67] Lawrence, P. Lorsch, J. (1967), *Organisation and Environment*, Harvard University Press, Cambridge, Mass, 1967
- [lee95] Lee, T. (1995), *A novel approach to asynchronous, decentralized decision-making in military command and control*. In Proceedings of the Seventh international Workshop on Computer-Aided Software Engineering. CASE. IEEE Computer Society, Washington, DC, 1995.
- [lewis04a] Lewis, D., Feeney, K., Carey, K., Tiropanis, T., Courtenage, S. (2004a), *Semantic-based Policy Engineering for Autonomic Systems*, in Proceedings of 1st IFIP WG6.6 International Workshop on Autonomic Communication - Autonomic Communication Principles, Berlin 18-19 October 2004
- [lewis04b] Lewis, D., Feeney, K., Tiropanis, T., Courtenage, S. (2004b), *An Active, Ontology-driven Network Service for Internet Collaboration*, in Proceedings of Workshop on the Application of Semantic Web Technologies to Web Communities at ECAI'04, Valencia, Spain, 22nd-27th Aug 2004.
- [lewis04c] Lewis, D., Brady, A., Carey, K., Conlan, O., Feeney, K., Higel, S., O'Donnell, T., O'Sullivan, D., Quinn, K., Wade V. (2004c), *Managed Person-centric Adaptive Services for Smart Spaces*, in Proceedings of eChallenges, e2004 conference, Vienna, Austria, IOS Press, Part 1, pp 42-49, 27th – 29th October 2004.
- [lewis04d] Lewis, D., O'Donnell, T., K. Feeney, Brady, A., Wade, V. (2004d), *Managing User-centric Adaptive Services for Pervasive Computing*, in Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'04), May 17th-18th 2004, New York City, USA, pp 248-255

- [lewis07] Lewis, D. Keeney, J. Carey, K. O'Sullivan, D. Wade, V. (2007), *Semantics for Composable Autonomic Elements*, *Journal of Autonomic and Trusted Computing*, 2007
- [li05] Li, N., Byun, J. W., Bertino. E. (2005) *A Critique of the ANSI Standard on Role-Based Access Control*. CERIAS TR 2005-29, Purdue University.
- [linington98] Linington, P., Milosevic, Z., Raymond, K. (1998), *Policies in Communities: Extending the ODP Enterprise Viewpoint*, in proceedings of the 2nd IEEE Enterprise Distributed Object Computing Workshop, San-Diego, Nov.1998.
- [lobo99] Lobo, J., Bhatia R., Naqvi, S. (1999), *A Policy Description Language*. In Proceedings of the Sixteenth National Conference on Artificial Intelligence, Eleventh Innovative Applications of AI Conference, Orlando, Florida, USA, 18-22 July 1999.
- [luisi03] Luisi, P. L. (2003) *Autopoiesis: a review and a reappraisal*. *Naturwissenschaften* **90** 49–59, 2003
- [lupu98] Lupu, E. (1998), *A Role-Based Framework for Distributed Systems Management*. PhD Thesis, Department of Computing, Imperial College. London, U. K., July 1998.
- [lupu99] Lupu, E. C., Sloman, M. S. (1999) *Conflicts in Policy-Based Distributed Systems Management*. In IEEE Transactions on Software Engineering - Special Issue on Inconsistency Management, vol. 25(6), pp. 852-869, November 1999.
- [mackenzie07] Mackenzie, M. L. (2007), *Leadership in the Information Age: A Culture of Continual Change*, *Bulletin of the American Society for Information Science & Technology*, April/May 2007
- [maglio05] Maglio, P. Campbell S. Kandogan E. (2005), *On the Need for Negotiation in Policy-based Interaction*, in Proceedings of the Second International Conference on Autonomic Computing, Seattle, WA, USA, 13-16 June 2005
- [mao05] Mao, X., Wang, J., Chen, J. (2005), *Modeling Organization Structure of Multi-Agent System*, pp. 116-119, IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2005.
- [marriot96] Marriot, D., Sloman, M. (1996), *Implementation of a Management Agent for Interpreting Obligation Policy*, IEEE/IFIP Distributed Systems Operations and Management (DSOM 96), L'Aquila (Italy), 1996.
- [marriott97] Marriott, D. (1997), *Management Policy for Distributed Systems*, Ph.D. Dissertation, Imperial College, Department of Computing, London, UK, July 1997.
- [maturana80] Maturana, H., Varela, F. (1980) *Autopoiesis and Cognition: the Realization of the Living*. Robert S. Cohen and Marx W. Wartofsky (Eds.), Boston Studies in the Philosophy of Science 42, [1st edition 1973] 1980.
- [mayo46] Mayo, E. (1946), *Human problems of an Industrial Civilization*, Macmillan, New York. 1946
- [mcknight96] McKnight, H.D., Chervany, N.L. (1996), *The Meanings of Trust*, Technical Report 94-04, Carlson School of Management, University of Minnesota, 1996.
- [mcsweeney06] McSweeney, B. (2006), *Are we living in a post-bureaucratic epoch?* *Journal of Organizational Change Management*, Volume: 19 Issue: 1 Page: 22 – 37, 2006.

- [michael01] Michael, J. B., Ong, V. L., Rowe, N. C. (2001), *Natural-Language Processing Support for Developing Policy-Governed Software Systems*. In Proceedings of the 39th International Conference on Technology for Object-Oriented Languages and Systems, Santa Barbara, California, IEEE Computer Society Press, pp. 263-274, July 2001.
- [miege05] Mieke, A. (2005), *Definition of a formal framework for specifying security policies. The Or-BAC model and extensions*. Ph.D. thesis, Télécom Paris (Ecole Nationale Supérieure des Télécommunications), June 2005
- [moffett90] Moffett, J., Sloman, M., Twidle, K. (1990), *Specifying discretionary access control policy for distributed systems*, Computer Communications, vol. 13, pp. 571-80, 1990,
- [moffett93] Moffett, J., Sloman, M. (1993), *Policy Hierarchies for Distributed Systems Management*. IEEE JSAC Special Issue on Network Management. vol. 11(9), pp. 1404-1414, December 1993.
- [moffett98] Moffett, J. D. (1998), *Control Principles and Role Hierarchies*, in Proceedings of the Third ACM/NIST Role Based Access Control Workshop, Fairfax, Virginia, USA, ACM Press, October 1998. pp 22-23
- [molony02] Molony, S. (2002), *Councillors face sack in new refuse charges row*, The Irish Independent, Dec 31, 2002. Available online at: http://www.unison.ie/irish_independent/stories.php3?ca=9&si=895373&issue_id=8556 Accessed June 2004.
- [mont99] Mont, M. C., Baldwin A., Goh, C. (1999), *Role of Policies in Distributed Trust Framework* Hewlett Packard Laboratories – Bristol, UK, Technical Report HPL-1999-104, 1999.
- [moore01] Moore, B., Ellessen, E., Strassner J., Westerinen, A. (2001), *Policy Core Information Model - Version 1 Specification*, RFC 3060, available from <http://www.ietf.org>, February 2001.
- [moore03] Moore, B., Ellessen, E., Strassner, J., Westerinen, A. (2003), *Policy Core Information Model Extensions*, IETF RFC 3460, February 2003.
- [moore04] Moore B., Durham D., Strassner J., Westerinen A., Weiss W. (2004), *Information Model for Describing Network Device QoS Datapath Mechanisms*, IETF RFC 3670, January 2004
- [nyanchama99] Nyanchama, M., Osborn, S.L. (1999), *The Role Graph Model and Conflict of Interest*, in ACM Transactions on Information and System Security 2(1): 3-33
- [oasis03] OASIS (Organization for the Advancement of Structured Information Standards) (2003), *Core Specification: eXtensible Access Control Markup Language (XACML) Version 1.0*, available from <http://www.oasis-open.org/committees/xacml>, Accessed June 2005.
- [oasis05] OASIS (2005), *XACML 2.0 Core: eXtensible Access Control Markup Language (XACML) Version 2.0*, available from <http://www.oasis-open.org/committees/xacml>, 2005. Accessed July 2006.

- [oasis06] OASIS (2006), *Reference Model for Service Oriented Architecture 1.0*, OASIS Standard, 12 October 2006, available at <http://docs.oasis-open.org/soa-rm/v1.0/> (accessed July 2007)
- [oh02] Oh S., Sandhu R. (2002), *A Model for Role Administration Using Organization Structure*, in Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002), pages 155-162, Monterey, California, June 2002.
- [omg99] OMG (Object Management Group) (1999), *Object Constraint Language Specification, version 1.3*, Chapter 7 in OMG Unified Modelling Language Version 1.3, June 1999.
- [ortalo98] Ortalo, R. (1998), *A Flexible Method for Information System Security Policy Specification*, in Proceedings of the 5th European Symposium on Research in Computer Security (ESORICS 98), Louvain-la-Neuve, Belgium, Springer-Verlag, September 1998.
- [osborn00] Osborn, S., Sandhu, R., Munawer, Q. (2000), *Configuring role-based access control to enforce mandatory and discretionary access control policies*, ACM Transactions on Information and System Security (TISSEC) Vol 3, (2), Pages: 85 – 106, May 2000
- [parunak01] Parunak, H. V. D., Odell, J. J. (2001) *Representing social structures in UML*, in Wooldridge, M. J., Weifl, G., and Ciancarini, P., editors, Agent-Oriented Software Engineering II: Second International Workshop, AOSE 2001, Montreal, Canada.
- [patwardhan04] Patwardhan, A., Korolev, V., Kagal, L., Joshi, A. (2004), *Enforcing Policies in Pervasive Environments*, in Proceedings, International Conference on Mobile and Ubiquitous Systems: Networking and Services, August 2004
- [quinn04] Quinn, K., O'Sullivan, D., Lewis, D., Wade, V.P. (2004), *Composition of Trustworthy Web Services*, in Proceedings of Information Technology & Telecommunications Conference (IT&T 2004), Limerick, Ireland, October 2004.
- [quinn05] Quinn, K., O'Sullivan, D., Lewis, D., Brennan, R., Wade, V.P. (2005), *deepTrust Management Application for Discovery, Selection, and Composition of Trustworthy Services* in Proceedings of IDIP/IEEE 9th International Symposium on Integrated Network Management (IM 2005), Nice, France, May 2005.
- [quinn06] Quinn, K., Kenny, A. Feeney, K., Lewis, D., O'Sullivan, D., Wade, V. (2006), *A Framework for the Decentralisation and Management of Collaborative Applications in Ubiquitous Computing Environments*, in Proceedings of IFIP/IEEE Network Operations and Management Systems, Vancouver 2006
- [rhodes04] Rhodes P.J.(Ed) (2004), *Athenian democracy*. New York: Oxford University Press, 2004
- [ribeiro01] Ribeiro, C., Zuquete, A., Ferreira, P. (2001), *SPL: An access control language for security policies with complex constraints*, in Proceedings of the Network and Distributed System Security Symposium (NDSS'01), San Diego, California, February 2001.
- [roethlisberger64] Roethlisberger, F., Dickson, W. (1964), *Management and the Worker*, Wiley, New York.

- [rossetti98] Rossetti, Y.(1998), *Implicit short-lived motor representations of space in brain-damaged and healthy subjects*, in *Consciousness & Cognition* 7, 520-558, 1998
- [saintandre04] Saint-Andre, P. (Ed.) (2004), *Extensible Messaging and Presence Protocol (XMPP)*, IETF RFC 3920, October 2004
- [samarati00] Samarati, P., Vimercati S. (2000), *Access Control: Policies, Models, and Mechanisms*, in *Foundations of Security Analysis and Design (Tutorial Lectures)*. R. Focardi and R. Gorrieri (eds), Springer -Verlag, pp. 137- 196, September 2000.
- [sandhu93] Sandhu, R. S. (1993), *Lattice-Based Access Control Models* Computer, vol. 26, no. 11, pp. 9-19, Nov., 1993.
- [sandhu94] Sandhu, R., Samarati, P. (1994), *Access Control: Principles and Practice*. IEEE Communications Magazine, vol. 32(9), pp. 40-48. 1994
- [sandhu96] Sandhu, R., Coyne, E., Feinstein, H., Youman, C. (1996), *Role-Based Access Control Models* Computer, vol. 29, no. 2, pp. 38-47, Feb., 1996.
- [sandhu99a] Sandhu, R., Bhamidipati, V., Munawer., Q. (1999a), *The ARBAC97 model for role based administration of roles*. ACM Transactions on Information and System Security, 2(1):105--135, 1999.
- [sandhu99b] Sandhu R., Munawer. Q. (1999b), *The ARBAC99 Model For Administration of Role*, in *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC'99)*, Phoenix, Arizona, December 1999.
- [sandhu00] Sandhu, R., Ferraiolo D., Kuhn, R. (2000), *The NIST Model for Role-Based Access Control: Towards A Unified Standard*. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, Berlin, Germany, pp. 47-61, 26-28 July 2000.
- [schoenberger97] Schoenberger, E. (1997), *The cultural crisis of the firm*, Oxford, Blackwell, 1997.
- [segall97] Segall, B., Arnold. D. (1997), *Elvin has left the building: A publish/subscribe notification service with quenching*. In *Proceedings of the Queensland AUUG Summer Technical Conference*, Brisbane, Australia, 1997. Available online at <http://www.dstc.edu.au/Elvin/doc/papers/auug97/AUUG97.html>. Accessed February 2007.
- [shadbolt02] Shadbolt, N. (2002), *A Matter of Trust*, IEEE Intelligent Systems, pp. 2-3 January/February 2002.
- [shen92] Shen H., Dewan, P. (1992), *Access control for collaborative environments*, in *Proceedings of ACM Conference on Computer-Supported Cooperative Work (CSCW'92)*, Building Real-Time Groupware, pages 51-58. ACM Press, 1992.
- [silva04] Silva, V., Choren, R., Lucena, C. (2004), *A UML based approach for modeling and implementing multi-agent systems*, in *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, pages 914-921, 2004.
- [simon97] Simon, R., Zurko, M. E. (1997), *Separation of Duty in Role-based Environments*, in *Proceedings of the 10th IEEE Workshop on Computer Security Foundations* (June 10 - 12, 1997).
- [sintek02] Sintek, M., Decker, S. (2002), *TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web*, in *Proceedings of First International Semantic Web Conference*, pp 364-378. 2002

- [sloman94a] Sloman, M. (1994a), *Policy Driven Management for Distributed Systems*. Journal of Network and Systems Management, Plenum Press. Vol.2 No. 4, 1994
- [sloman94b] Sloman, M., Twidle, K. (1994b), *Domains: A Framework for Structuring Management Policy*. Chapter 16 of Network and Distributed Systems Management (Sloman, ed.), pp. 433-453. 1994
- [sloman02] Sloman, M., Lupu, E. (2002), *Security and Management Policy Specification*, IEEE Network Special issue on Policy Based Networking March 2002
- [solworth04] Solworth, J.A. Sloan, R.H. (2004), *A layered design of discretionary access controls with decidable safety properties*, in Proceedings of IEEE Symposium on Security and Privacy, pages 56- 67, May 2004
- [snir03] Snir, Y., Ramberg, Y., Strassner, J., Cohen, R., Moore, B. (2003), *Policy Quality of Service (QoS) Information Model*, IETF RFC 3644, November 2003
- [steen99] Steen, M. W. A., Derrick J. (1999), *Formalising ODP Enterprise Policies*. In Proceedings of the 3rd International Enterprise Distributed Object Computing Conference (EDOC '99), University of Mannheim, Germany, IEEE Publishing, September 1999.
- [steen00] Steen, M. W. A., Derrick J. (2000), *ODP Enterprise Viewpoint Specification*. Computer Standards and Interfaces, vol. 22, pp. 65-189, September 2000.
- [strassner97] Strassner, J., S. Schleimer, *Policy Framework Definition Language (version 00)*, IETF Internet draft work in progress, available from <http://www.ietf.org>, 17 November 1997.
- [strassner04] Strassner, J., Moore, B. Moats, R. Elleson, E. (2004), *Policy Core Lightweight Directory Access Protocol (LDAP) Schema*, IETF RFC 3703, February 2004.
- [taylor11] Taylor, F. W. (1911), *The principles of scientific management*, Harper & Brothers, New York, 1911
- [taylor62] Taylor, J. G. (1962), *The behavioral basis of perception*. New Haven: Yale University Press. 1962
- [thomas97a] Thomas, R., Sandhu, R. (1997a), *Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management*, in 11th IFIP Working Conference on Database Security, Lake Tahoe, California, USA, 1997.
- [thomas97b] Thomas R., (1997b), *TMAC: A primitive for Applying RBAC in collaborative environments*, in 2nd ACM Workshop on RBAC, Fairfax, Virginia, USA, pp. 13-19. November 6-7 1997.
- [thompson84] Thompson, K. (1984), *Reflections on Trusting Trust*, *Communication of the ACM*, Vol. 27, No. 8, August 1984, pp. 761-763
- [thompson02] Thompson P., McHugh, D. (2002), *Work Organisations*, Palgrave, New York, 2002.
- [tideman06] Tideman, N. (2006), *Collective Decisions And Voting: The Potential for Public Choice*, Ashgate Publishing Ltd, Aldershot, 2006
- [ting88] Ting, T.C. (1988), *A User-Role Based Data Security Approach*, Database Security: Status and Prospects, (C.E. Landwehr, Ed.). Elsevier, 1988.

- [tonti03] Tonti, G., Bradshaw, J.M., Jeffers, R. Montanari, R. Suri, N., Uszok, A. (2003), *Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder*. In International Semantic Web Conference, pp. 419--437, 2003.
- [uszok03] Uszok, J. Bradshaw, R. Jeffers, N. Suri (2003), *KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement*, in proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), Lake Como, Italy
- [vandermeer06] Van Der Meer, S., Davy, A., Davy, S., Carroll, R., Jennings, B., Strassner, J. (2006), *Autonomic Networking: Prototype Implementation of the Policy Continuum*, in Proceedings of the 2006 IEEE Workshop on Broadband Convergence Networks.
- [vanlamsweerde95] Van Lamsweerde, A., Darimont, R. (1995), *Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt*, in 2nd IEEE Symposium on Requirements Engineering (RE '95), York, UK, IEEE Computer Society Press.
- [vanlamsweerde96] Van Lamsweerde, A. (1996) *Divergent Views in Goal-Driven Requirements Engineering*. ACM SIGSOFT Workshop on Viewpoints in Software Development, San Francisco.
- [vanlamsweerde99] Van Lamsweerde, A. (1999), *Goal-Oriented Requirements Analysis with KAOS* Presentation to. Policy Workshop 1999, HP-Laboratories, Bristol, UK.
- [varadharajan96] Varadharajan, V. (1996), Allen, P., *Joint actions based authorization schemes*. Operating Systems Review 30, 3, Jul. 1996, 32-45
- [varela74] Varela, F. J., Maturana, Humberto, R., Uribe, R. (1974), *Autopoiesis: the organization of living systems, its characterization and a model*. Biosystems 5 187–196. 1974
- [verma00] Verma, D. C. (2000), *Policy-Based Networking: Architecture and Algorithms*, New Riders Publishing, 2000.
- [virmani00] Virmani, A., Lobo, J., Madhur K. (2000), *Netmon: network management for the SARAS softswitch*, in Proceedings of IEEE Network Operations and Management Symposium, 2000.
- [walker06] Walker, J. (2006), *A introduction to the Viable System Model as a diagnostic & design tool for co-operatives & federations Jon Walker Version 3.0*, 2006, available online at: http://www.esrad.org.uk/resources/vsmg_3/screen.php?page=home, accessed July 2007
- [weiskrantz97] Weiskrantz, L. (1997), *Consciousness Lost and Found A Neuropsychological Exploration*, Oxford University Press, New York, Oxford, 1997.
- [wickham04] Wickham, P. A. (2004), *Management Consulting: Delivering an Effective Project*, Prentice Hall, 2nd Edition 2004.
- [wikipedia06] Wikipedia (2006), *Supply Chain Definition*, available online at: http://en.wikipedia.org/wiki/Supply_chain (accessed June 2006)
- [wilde05] Wilde, E., Nabholz, N. (2005), *Access Control for Shared Resources*, In: Proceedings of the International Conference on Intelligent Agents, Web Technology

and Internet Commerce, IEEE Computer Society Press, Vienna, Austria, November 2005.

[yialelis96] Yialelis, N., Sloman, M. (1996), *A Security Framework Supporting Domain Based Access Control in Distributed Systems*, Proceedings of Internet Society Symposium on Network and Distributed System Security, San Diego, Feb. 1996, IEEE, pp. 26-39.

[zhang03a] Zhang, L., Ahn, G., Chu, B. (2003a), *A rule-based framework for role-based delegation and revocation* in ACM Transactions on Information Systems. Security 6, 3 (Aug. 2003).

[zhang03b] Zhang, X., Oh, S., Sandhu, R. (2003b), *PBDM: a flexible delegation model in RBAC*, in Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (Como, Italy, June 02 - 03, 2003). SACMAT '03.

[zhang05] Zhang, Y., Liu, X., Wang, W. (2005), *Policy Lifecycle Model for Systems Management*, IT Professional, vol. 07, no. 2, pp. 50-54, Mar/Apr, 2005

Prior Publications

The work described in this thesis has been developed through a series of peer-reviewed publications. This list contains the publications which relate directly to this work.

Lewis, D., O'Donnell, T., Feeney, K., Brady, A. (2003) *Person Centric Service Adaptation*. First International Workshop on Management of Ubiquitous Communications and Services, MUCS 2003, Waterford, Ireland, December 11, 2003

Feeney, K., Lewis, D., Wade, V. (2004), *Policy Based Management for Internet Communities*, in Proceeding of IEEE 5th International Workshop on Policies for Distributed Systems and Networks, IBM Thomas J Watson Research Center, New York, USA, pp 23-34, June 2004

Feeney, K., Lewis, D. (2004), *Community Based Policy Management for Smart Spaces*, in Proceedings of 2nd International Workshop on Managing Ubiquitous Communications and Services (MUCS), Dublin, Ireland, December 13-14, 2004

Feeney, K., Lewis, D., Wade, V., (2007) Roles Considered Harmful in Policy-based Management for Dynamic Organisations, in Proceedings of the IEEE International Symposium on Integrated Network Management, (IM 07), May 2007

Lewis, D., Feeney, K., Carey, K., Tiropanis, T., Courtenage, S. (2004), *Semantic-based Policy Engineering for Autonomic Systems*, in Proceedings of 1st IFIP WG6.6 International Workshop on Autonomic Communication - Autonomic Communication Principles, Berlin 18-19 October 2004

Lewis, D., Feeney, K., Tiropanis, T., Courtenage, S. (2004), *An Active, Ontology-driven Network Service for Internet Collaboration*, in Proceedings of Workshop on the Application of Semantic Web Technologies to Web Communities at ECAI'04, Valencia, Spain, 22nd-27th Aug 2004.

Lewis, D., Feeney, K., Tiropanis, T., Courtenage, S. (2004), *An Active, Ontology-driven Network Service for Internet Collaboration*, in Proceedings of the 1st IFIP TC6 WG6.6 International Workshop on Autonomic Communication, WAC, Berlin, Germany, October 18-19, 2004

Lewis, D., Brady, A., Carey, K., Conlan, O., Feeney, K., Higel, S., O'Donnell, T., O'Sullivan, D., Quinn, K., Wade V. (2004), *Managed Person-centric Adaptive Services for Smart Spaces*, in *Proceedings of eChallenges*, e2004 conference, Vienna, Austria, IOS Press, Part 1, pp 42-49, 27th – 29th October 2004.

Lewis, D., O'Donnell, T., K. Feeney, Brady, A., Wade, V. (2004), *Managing User-centric Adaptive Services for Pervasive Computing*, in Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'04), May 17th-18th 2004, New York City, USA, pp 248-255

Feeney, K., Quinn, K., Lewis, D., O'Sullivan, D., Wade, V. (2005), *Relationship-Driven Policy Engineering for Autonomic Organisations*, in Proceedings of 6th IEEE International Workshop on Policies for Distributed Systems, Stockholm, Sweden, pp 89-98, June 2005

Lewis, D. O'Sullivan, D., Feeney, K. Keeney, J. Power, R. (2006), *Ontology-based Engineering for Self-Managing Communications*, 1st IEEE International Workshop on Modelling Autonomic Communications Environments (MACE 2006), Dublin, Ireland,

25-26 October 2006, edited by W. Donnelly, R. Popescu-Zeletin, J. Strassner, B. Jennings, S. van der Meer, multicon verlag, 2006, pp81 - 100

Quinn, K., Kenny, A. Feeney, K., Lewis, D., O'Sullivan, D., Wade, V. (2006), *A Framework for the Decentralisation and Management of Collaborative Applications in Ubiquitous Computing Environments*, in Proceedings of IFIP/IEEE Network Operations and Management Systems, Vancouver 2006

Lewis, D., Feeney, K., Foley, K., Doyle, L., Forde, T., Argyroudis, P., Keeney, J., O'Sullivan, D., *Managing Policies for Dynamic Spectrum Access*, 1st International IFIP TC6 Conference on Autonomic Networking (AN2006), Paris, France, September, 2006, LNCS 4195, Springer, 2006, pp285 - 297

Feeney, K., Lewis, D., Argyroudis, P., Nolan, K., O'Sullivan, D. (2007), *Grouping Abstraction and Authority Control in Policy-based Spectrum Management*, in Proceedings of the IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN'07), April 2007.

Feeney, K., Lewis, D., Wade, V., *Handling Dynamic Organizational Change with Community-Based Policy Management*, (2007) 8th IEEE International Workshop on Policy (POLICY 2007), Bologna, Italy, 13-15 June 2007, 2007

Lewis, D., Feeney, K., O'Sullivan, D., *Integrating the Policy Dialectic into Dynamic Spectrum Management*, IEEE Dynamic Spectrum Access Networks (DySpan 2007), Dublin, Ireland, 17-20 April 2007, 2007

Appendix 1: Organisational Modelling & Access Control

Access control is concerned with limiting the activity of legitimate, authenticated users. Its objective is to ensure that every access to the system and its resources is controlled and that only those accesses that are authorised are permitted. There are three basic components to an access control system: the subjects, the target resources and the rules which specify the ways in which the subjects can access the target resources. The set of high-level rules according to which access control must be regulated are traditionally called access control policy [samarati00]. Research into access control has generally concentrated on identifying useful access control models, which provide a formal representation of security policies and allow properties of the system to be proven through formal reasoning.

Access control systems have been traditionally divided into *discretionary* and *mandatory* access control models. Discretionary policies are concerned with the specification of authorisation rules to govern the access of users to information contained within a system, whereas mandatory policies are mostly concerned with controlling information flow between the objects of a system. Information flow policies are often described as a separate type of policy, and are directly related to the issue of data confidentiality. More recently *Role-Based Access Control* (RBAC) policies have attracted increasing attention, they allow more flexible specification of rules and are expressive enough to be seen as a superset of the traditional approaches. This appendix analyses the organisational models embedded in the most common access control systems.

A1.1 Mandatory Access Control

Mandatory access control (MAC) enforces access control on the basis of fixed regulations mandated by a central authority. Mandatory security policies are typified by the Bell-LaPadula lattice-based model and are often called Lattice Based Access Control (LBAC) [sandhu93]. Lattice-based models were defined to deal with the issue of data confidentiality, and concentrate on restricting information flow in computer systems. This is achieved by assigning a security classification to each subject (an active entity that can execute actions) and each object (a passive entity storing information) in the system. Subjects and objects form a lattice based on their classification, which is used to enforce some fixed mandatory policies regarding the actions that subjects can

execute on objects. The Bell-LaPadula identifies two sets: objects (O) and subjects (S), with S being a subset of O, and defines the following operations that subjects can perform on objects:

- Execute (no observation, no alteration)
- Read (observation, no alteration)
- Append (no observation, alteration)
- Write (observation, alteration)

The model defines a set of totally ordered classifications C (Top-secret, Secret, Classified, Unclassified), and a set of categories K (e.g. NATO, NUCLEAR etc), partially ordered by set inclusion. Each object *o* is assigned exactly one level, *L(o)* called the classification of the object. A level consists of a classification from the set C and a subset of the set of categories K. Each subject is defined as having two levels, a clearance level and a current level. The system enforces mandatory access control rules on all operations, known as the *Simple Security Property* and the **-Property* (star property) which ensure that only authorised operations are permitted [bell76].

In terms of the model of the human organisation embodied within it, LBAC models generally provide very little. Users are grouped together by their classification levels and the categories that they belong to, but these are fairly unwieldy and rigid grouping methods. The classification level of objects is the only way in which the managed resources are related to the human organisation, meaning that every object is related to all the subjects in the system in one of only 3 possible ways. No aspects of the functional nature of the organisation to which the subjects belong is modelled. All administration of the system is carried out by a security officer whose access rights are external to the model. Furthermore, the model is fundamentally restricted in terms of its expressiveness, since there is no facility for specifying conditional access and only four different means of accessing each object are supported.

A1.2 Discretionary Access Control

Discretionary access control (DAC) has been used extensively in commercial applications, particularly in operating systems and relational database systems. The central idea of DAC is that the owner of an object, who is usually its creator, has discretionary authority over who else can access that object. DAC, in other words,

involves owner-based administration of access policy. This provides considerable flexibility and decentralisation of the system's administration by devolving many access control decisions about an object to the user, rather than being the responsibility of a dedicated security administrator. DACs enable the owner of an object to specify what types of accesses can be made (privileges) and by whom (which users) on the object [osborne00, solworth04].

There are many variations of DAC models and the major distinguishing feature between them relates to how the owner's discretionary power can be delegated to other users and how such delegations can be revoked. Major variations include *strict DAC* where the owner of an object is the only user with discretionary power to delegate access rights to the object and this ownership cannot be transferred [graham72]. *Liberal DAC* allows the owner to delegate discretionary authority for granting access to an object to other users and can be further sub-divided into variations where such delegations are limited to a number of stages – limiting the amount of times a delegated right can be re-delegated to other users [osborn00]. Similarly, major variations exist in how revocation of access rights is handled by such models. However, such models are generally limited to applying a small number of constraints to delegations (such as the maximum indirection level) and cannot express other conditionalities such as temporal or environment based constraints.

The most basic and fundamental approach to the specification of DAC policies uses an access matrix model [lampson71] (usually implemented as an Access Control List (ACL) or set of Capabilities) as a framework for reasoning about the permitted accesses. In the access matrix model the *protection state* of the system is defined by a triple (S, O, A) , where S is the set of subjects, O is the set of objects and A is the access matrix where rows correspond to subjects, columns correspond to objects and entry $A[s, o]$ reports the privileges of s on o . Such models are severely limited in their expressiveness. Subjects are mapped directly to access rights and thus many constraints (such as temporal, history based or system-state based authorisations) on authorisations cannot be expressed [samarati00]. Various formal extensions have been proposed to DAC models which incorporate such constraints. For example, [bertino03] describes a formal model for extending authorisations with temporal constraints. Furthermore, many researchers identify the need for both positive and negative authorisations as a way to conveniently specify exceptions in authorisation rules [samarati00].

Many DAC systems support some form of *group* construct for allowing users to delegate rights over the objects that they own to groups of users. Broadly speaking, groups allow access rights for objects to be mapped to groups of users, rather than being mapped to individuals. The Posix [posix] specification for UNIX Extended Access Control Lists includes support for subjects with multiple group memberships and the definition for access control rights for each object on a per-group basis. In [abadi93] a calculus for access control is presented (and shown to be decidable for a range of DAC models) where subject's rights are derived from memberships groups, which can be nested. Many traditional DAC systems, such as UNIX operating systems, rely upon central administrators to create groups and set up memberships. In [wilde05], an extension is proposed to DAC to surmount the problems in applying DAC to collaborative environments, by defining two different types of group: named groups and resource-based groups, as well as hierarchical roles. Although this schema allows users greater flexibility in sharing data – resource groups allow the sharing of resources with all those who have access rights to another resource – the definition of *named groups* and their memberships still remains the responsibility of administrators.

Although DAC is most commonly based on the concept that each object has an owner who is responsible for administering the access to that object, in practice such systems generally also include administrators who can grant and revoke rights as well as the object's owner. A variety of schema for specifying administrative policies exist [sandhu94], which cause the model to deviate from the DAC model as noted by [abrams93] who described such models as *Non Discretionary Access Control* (NDAC). NDAC refers to situations where authority is vested in some users but there are various controls on its propagation. NDAC Administrative schema can be divided into broad categories, including the following [sandhu94].

- **Centralised** where a single authoriser (or group) is allowed to grant and revoke authorisations to the users.
- **Hierarchical** where a central authoriser is responsible for assigning administrative responsibilities to other administrators. The administrators can then grant and revoke access authorisations to the users of the system according to the organization chart.

- **Cooperative** where special authorisations on given resources cannot be granted by a single authoriser but needs cooperation of several authorisers.
- **Ownership** where a user is considered the owner of the objects she creates. The owner can grant and revoke access rights for other users to that object
- **Decentralised** where the owner or administrator of an object can also grant other users the privilege of administering authorisations on the object.

Analysing DAC in order to identify the organisational model which it embodies gives the following results. Resources are modelled as a simple collection of objects, and although many DAC implementations do allow objects to represent abstract collections of other objects, there is no standard grouping mechanism for managed resources and thus little consensus on how to relate managed resources to one another. In most DAC models, resources are strongly related to users in the system through the ownership relationship. Support for groups which have a relationship with resources augments this to give some representation of organisational units and collective functions, however this generally remains secondary to the dominant ownership relationship. Thus, the model of the organisation that is contained is fairly atomised, with strong relationships between individual users and resources, but weak relationships between users and other users and between groups of users and resources. DAC's centralised administration essentially positions the administration of the system outside the user model. Hierarchical administrative schema embody a fuller model of organisation, as they contain an authority relationship between users, but this hierarchy is essentially only related to the organisational model of users through the resources that it assigns to them - a very indirect relationship. Cooperative administration mechanisms embody some aspects of collective decision making, but it is unclear how such schemes should be structured and how they should relate to ownership. In all of these schema, the dynamism of the system, whether driven by an administrator or by an object's owner, is very difficult to constrain. Limitations can be placed on delegations and administration rights, but other than that the organisational model can evolve in any direction. Therefore, in general, the various limitations of these schemes means that it is very difficult to derive the functional aspects of the organisation from the models embodied in the DAC models. Finally, traditional DAC models generally lack expressiveness and flexibility in terms of how they can control access to objects.

A1.3 Other Traditional Models

In addition to the traditional DAC and MAC models, there are a large number of more or less traditional access control models which have been developed for particular application domains or for dealing with particular security problems. These models are discussed alongside traditional access control solutions as they do not aim towards the generality and expressiveness of modern policy based management solutions. Such models include various attempts to integrate elements of DAC and MAC models. For example, the Clark-Wilson model [clark87] attempts to present in a formal, abstract way commercial data processing practices. Its main goal is to ensure the integrity of an organisation's accounting system and to improve its robustness against insider fraud. Another example is the Chinese-wall policy, described in [anderson01], which was developed as a formal model of a security policy applicable to financial information systems, to prevent information flows which cause conflict of interest for individual consultants. Due to the domain specific applicability of most of such systems, there is little point in looking for general purpose organisational models within them.

A1.4 Role Based Access Control

The concept of the role can be defined as *“the set of rights and duties associated with a position, which are assigned to person who occupies that position”* [sandhu96]. The use of roles in access control rests upon the observation that in most organisations access control decisions are based upon the role that the individual is acting in, rather than their identity [ferraiolo92]. This observation, and the RBAC approach that springs from it, leads to several extra efficiencies in the specification of a policy-based system; a single policy can apply to all members of a role, rather than needing policies for each individual member of the role; when an individual leaves or joins a role, there is no necessity to change the policies associated with the role; policies that are commonly required in large organisations, such as rules specifying “separation of duties” constraints, can be implemented conveniently through roles, by declaring separate roles with constraints upon individuals concurrently activating these roles. As a result of these obvious advantages, role-based access control models, in the most general sense, have become dominant within the research community and have spawned a large number of extensions and variations, as illustrated by the following sections.

Roles can be seen as an extension of the group concept that exists in most DAC models. RBAC models differ fundamentally from DAC models, however, in that the concept of

an individual owner of each managed resource as the dominant source of authority for that resource is removed. Instead an individual's access rights primarily depend upon their membership of roles. Furthermore, whereas DAC groups are primarily viewed as a collection of users, rather than a collection of permissions, and are generally implemented to allow the easy discovery of the set of users associated in a group rather than the set of permissions associated with the group, roles are viewed and implemented to provide both [sandhu96]. A major consequence of removing the concept of individual ownership which is central to DAC is that permissions to delegate authorisations must be explicitly specified within the model, rather than being derived from the object's ownership.

A1.5 ANSI Standard RBAC Models

Sandhu et al. [sandhu96] proposed a set of conceptual RBAC models for standardisation in an effort to encourage implementations to adopt a standard system. These proposals were originally known as RBAC96. The existence of multiple models is due to the fact that RBAC requirements can range from the very simple to the very sophisticated. The set of models were published in a series of draft papers by the US National Institute of Standards and Technology (NIST). The final version was approved in February 2004 as the American National Standard ANSI INCITS 359-2004 [ansi04]. Plans are underway to improve the standard and move the standard to the International Organization for Standardization (ISO). However, the ANSI standard contains several technical errors and there have been calls for the adoption of an improved standard [li05]. Nonetheless, the conceptual models described in the ANSI standard are retained in these suggested improvements. The various models in the ANSI RBAC family are briefly outlined below.

Core RBAC

Core RBAC, originally called *RBAC0*, contains users, roles, permissions and sessions. Users are assigned to roles, permissions are assigned to roles, and users acquire permissions by being members of roles. The NIST standard requires that user-role and permission-role assignment can be many-to-many. Thus, the same user can be assigned to many roles and a single role can have many users. Similarly, for permissions, a single permission can be assigned to many roles and a single role can be assigned to many permissions. The Core RBAC specification includes requirements for user-role review, whereby the roles assigned to a specific user can be determined as well as users

assigned to a specific role. A similar requirement for permission-role review is imposed as an advanced review function. Core RBAC also includes the concept of user sessions, which allows selective activation and deactivation of roles.

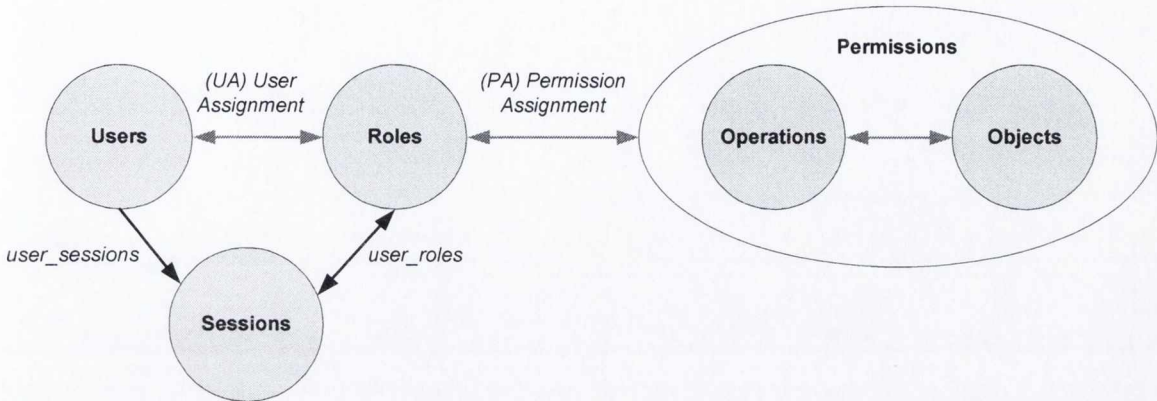


Figure A1.1 Core RBAC model

Hierarchical RBAC

Hierarchical RBAC, formerly known as *RBAC1*, includes *Core RBAC* and introduces the concept of role hierarchies [sandhu98]. Hierarchies are a means for structuring roles to reflect an organisation's lines of authority and responsibility, and are specified using inheritance between roles. The concept of role hierarchies borrows heavily from the concept of inheritance used in object oriented programming. Role inheritance enables reuse of permissions by allowing roles to be specified as junior from which senior roles inherit permissions. A role which inherits a second role has a superset of the permissions of that role. Role hierarchies are most commonly defined in terms of inherited permissions [nyanchama99]. That is to say that a role *r1* is defined as inheriting role *r2* if all privileges of *r2* are also privileges of *r1* [ansi04]. The ANSI standard recognises two classes of role hierarchies, termed *general* and *limited*. In limited role hierarchies, a role may have one or more immediate ascendants, but is restricted to a single immediate descendant. General hierarchies have no such restrictions and thus support multiple inheritance of permissions from two or more role sources and inheritance of user membership from two or more role sources. Limited hierarchies result in a simpler tree structure, while general hierarchies support the composition of senior roles from multiple junior roles with fewer permissions and also enable users to be treated uniformly as a special case of a role – a role with a single member.

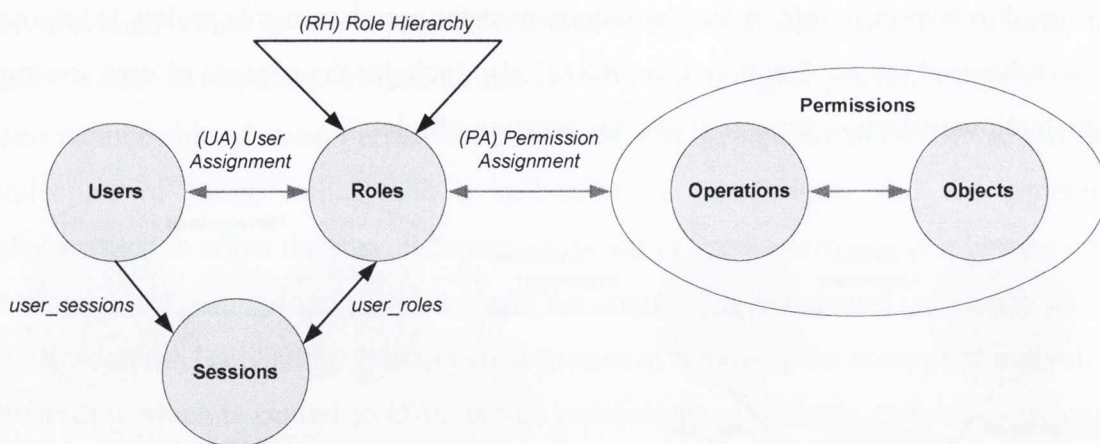


Figure A1.2 Hierarchical RBAC Model

Constrained RBAC

Constrained RBAC extends *Core* and *Hierarchical RBAC* by including constraints to restrict the assignment of users or permissions to roles, or the activation of roles in sessions. Constraints are used to specify application-depended conditions, and satisfy well-defined control principles such as the principles of least-privilege and separation of duties. The standard defines two types of constraints, both of which are designed to support the specification of separation of duties relations. The concept of separation of duties has been widely applied in business, industry, and government and has been extensively studied in the computer security literature [clark87, simon97]. Its purpose is to ensure that failures of omission or commission within an organisation can be caused only as a result of collusion among individuals. Constrained RBAC supports static and dynamic separation of duty relations. Static separation of duties (SSD) enforces constraints on the assignment of users to roles, while dynamic separation of duties (DSD) enforces constraints on the roles that can be activated within or across a user's sessions.

The constraints defined by the NIST standard support the specification of a set of roles and the cardinality of the subset of that set which would constitute a breach of the constraint if they were assigned to a user in SSD, or were activated by the user in DSD. So, for example, an SSD constraint specified as ($\{\text{clerk, supervisor, auditor}\}$, 2) would signify that no user can be assigned to more than 2 of the listed roles at any one time.

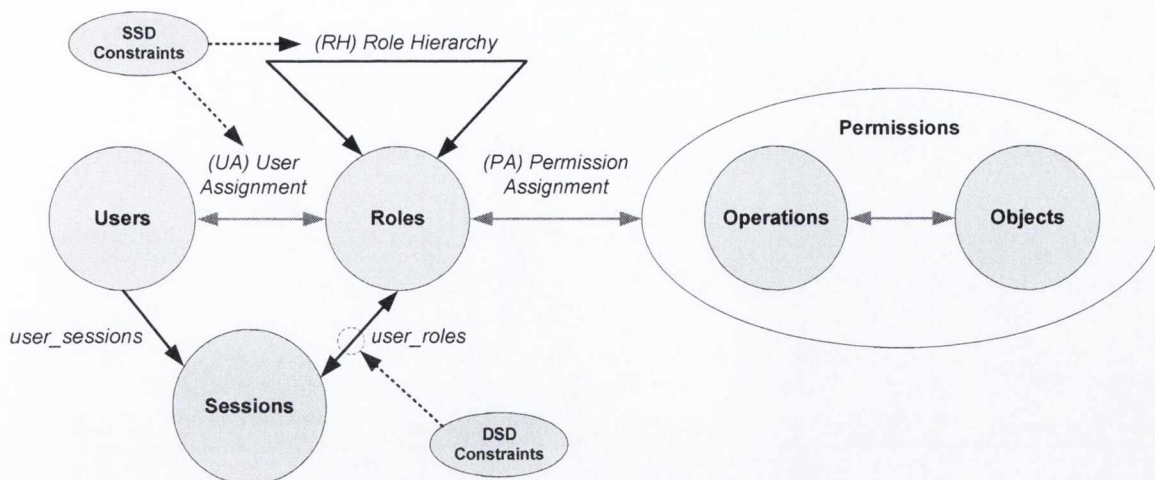


Figure A1.3 Constrained RBAC Model

A1.6 RBAC Administration

It is one of the widely acknowledged shortcomings of the NIST standard for RBAC that there is no provision for the standardisation of administration of RBAC systems [jaeger00]. The ANSI RBAC standard defines a set of administrative functions that must be provided by an RBAC system. However, the problem of managing access to these functions is not addressed by the standard.

At the same time as putting forward the RBAC96 family of models, Sandhu et al proposed the idea of distinct administrative roles with administrative permissions. In a later paper, known as ARBAC97 (Administrative Role Based Access Control), this concept was fleshed out, defining a set of standard administrative actions, grouped into User-Role Assignment (URA97), Permission-Role Assignment (PRA97) and Role-Role Assignment (RRA97) [sandhu99a]. These administrative actions are carried out by users who belong to a separate administrative role hierarchy, which is generally envisaged to consist of the information security officers of the organisation. The ARBAC97 model provides a framework for enforcing administrative policies and delegating responsibility for certain areas of the administration of a RBAC system to roles lower down the administrative hierarchy.

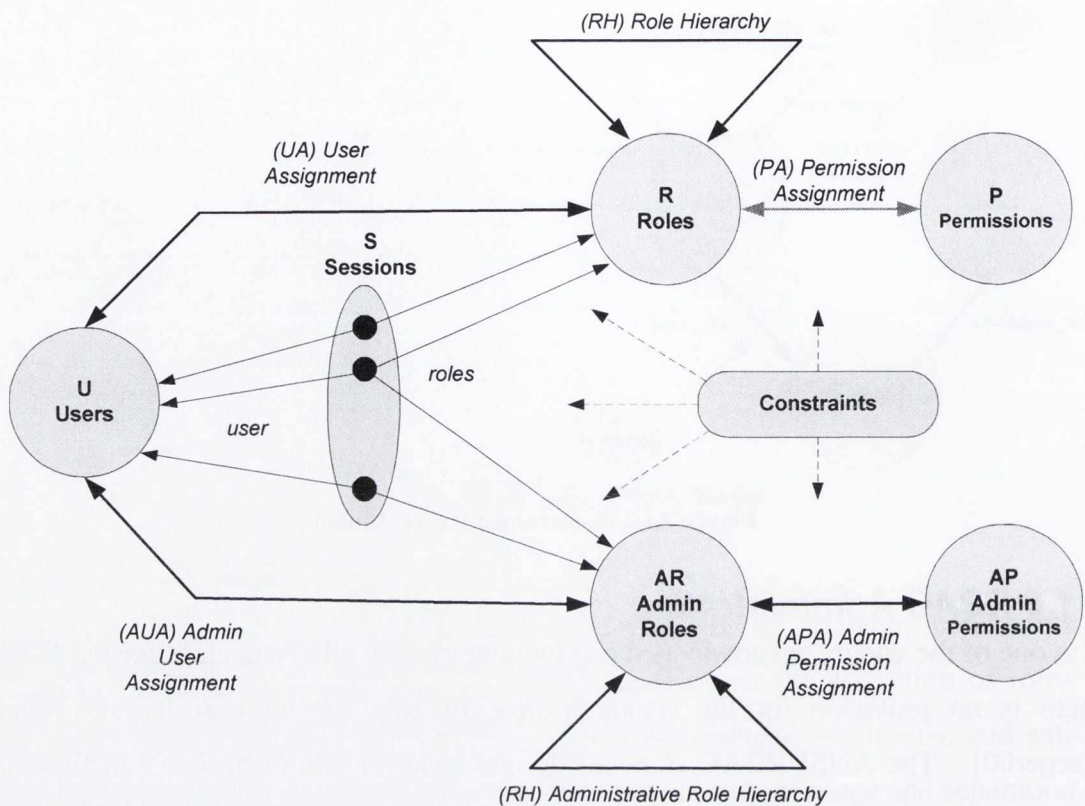


Figure A1.4 ARBAC97 Administrative Model

Two ARBAC extensions have been proposed. ARBAC99 [sandhu99b] presents a way to manage the mobile and immobile users and permissions. Unlike a mobile user, an immobile user can be seen as a non-permanent user such as a user under training, a visitor, a consultant, etc. In this case, the user can be a member of a role and get the corresponding permissions. But an administrative role cannot use this membership to put the immobile user into other roles. That is, an immobile user cannot climb the hierarchy. The same idea is used for the immobile permissions. ARBAC02 [oh02] proposes some improvements to resolve various problems that have been identified with ARBAC97, for example the multi-step user assignments which generate a lot of work for the security officers and which cause redundant tuples in the URA management. The main modification made in ARBAC02 affects the prerequisite conditions for the user and the permission assignment. An organisational structure of user pool and an organisational structure of a permission pool are created. The first one is managed by the human resources group, the second one by the IT group. Two hierarchies independent from the role hierarchy result. User and permission assignment is made by the security officers by picking user and permissions in these pools. This simplifies the assignment processes but transfers the problem of the prerequisite conditions onto the

roles. SARBAC (Scoped Administration of Role-Based Access Control) [crampton02] is an extension of RHA4 [crampton03] and an alternative to ARBAC97. SARBAC relies on administrative scope which changes dynamically as the role hierarchy changes. Thus, update operations over RBAC96 and SARBAC relations become easier and cannot lead to inconsistent rules. In particular, SARBAC makes it possible to delete a role without any restriction. Unlike ARBAC97, it is possible to assign administrative roles to users as SARBAC does not make any distinction between regular and administrative roles.

Role Delegation

Role delegation is an extension to the conventional delegation of privileges, in which one user grants privileges to another through granting them access to roles. In [barka00a], a role-based delegation model called RBDM0 is introduced. In RBDM0, if a role is delegated then all the associated privileges are granted. Delegation is limited to a single step. Later work by the same authors [barka00b] identifies cases of role-based delegation that are useful in practice, and in particular extends the model to include cascading delegation. Role delegation, if deployed with proper controls, can allow some decentralisation of administrative responsibilities through the organisation, both by allowing administrative users to delegate administrative roles and by allowing ordinary users to delegate authorisations, without requiring any intervention from administrators.

A1.7 RBAC's Organisational Model

In analysing the organisational model embodied within an RBAC system, it can be immediately noted that roles provide a fundamental grouping abstraction which relate users of the system together on the basis of them having a common set of permissions with respect to a subset of the managed resources. However, it is not generally possible to map this information to structural or functional information about the organisation. Although roles may serve as abstractions to model structural groupings within the organisation, this is an arbitrary semantic assignment which is dependant on the role being treated as such by administrators. Roles contain no concept of collective functions and it is not generally possible to map a particular role to a particular function of the organisation. Individuals may be members of a role in order to fulfil entirely different, and even contradictory, functions within the organisation. For example, there might be a *Financial Officer* role which must be assumed in order to access an electronic bank account. One user may be assigned to this role in order to pay bills for the organisation,

another user may be assigned to it in order to transfer funds into the account from customers – which are entirely different functions.

A *core RBAC* role can be seen as simply a collection of permissions and a collection of users, and it is a collection selected as being a convenient one for specifying the distribution of permissions throughout an organisation rather than that has an inherent structural or functional significance. The set of individuals who are associated through belonging to the same role is not generally something that can be used in order to derive information regarding the functional or structural characteristics of the organisation. Any such functional or structural significance that is attributed to a role is entirely dependant on the administrators and is not enforced by the model.

Role Hierarchies

Hierarchical RBAC adds the notion of role inheritance and, by extension, role hierarchies to the core RBAC model. It was initially thought that role hierarchies might be sufficient for modelling many of the aspects of a human organisation, and that they provided a natural means for structuring roles to reflect an organisation's lines of authority and responsibility. However, the simple hierarchical role model suffers from various problems; in particular, if superiors automatically inherit the permissions of their inferiors the principle of least privileges is broken. For example, consider the role hierarchy in figure A1.5 (taken from the ANSI standard). By convention more senior roles appear at the top and permissions are inherited upwards.

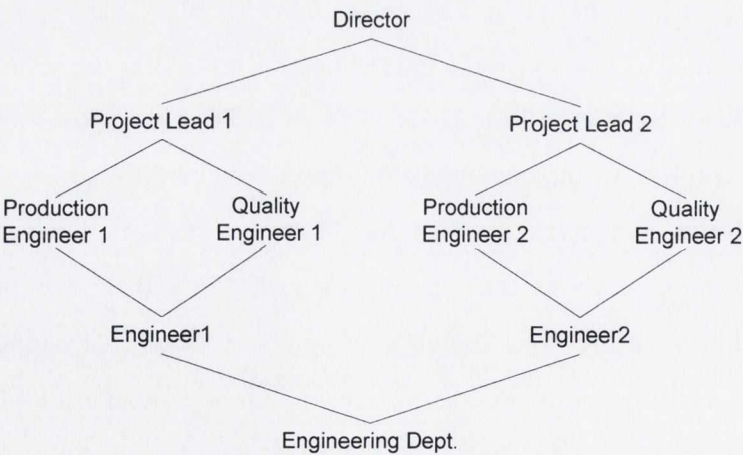


Figure A1.5 Hierarchical RBAC Example

The problem with this hierarchy is that it is not generally desirable for the director of the department to possess all the permissions that are possessed by all of the engineers in the department, since many of these are not relevant to the function of the director role

and thus, the role hierarchy violates the principle of least privilege. To get around this problem, the idea of private roles and limited inheritance has been introduced [sandhu96]. This is achieved by creating a private role as a complement to each inherited role. The private role is used to group the non-inherited permissions together and it is not inherited. In the case where certain superiors in the hierarchy should inherit private permissions, but not others, private sub-hierarchies can be created.

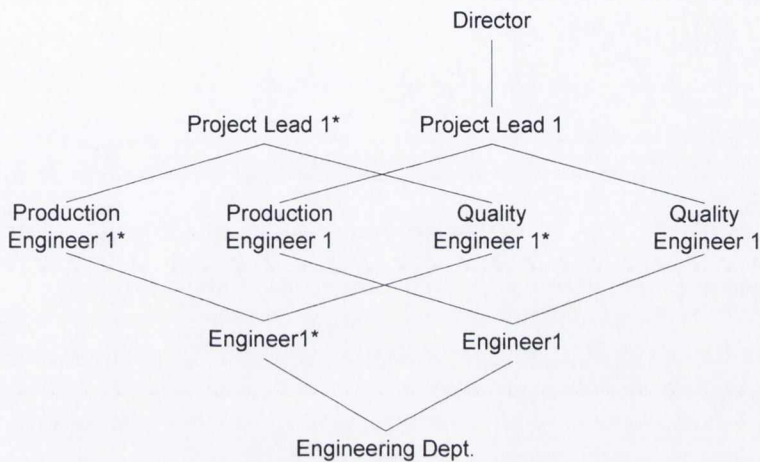


Figure A1.6 RBAC Private Role Hierarchy

Figure A1.6 adds a private hierarchy to a section of the previous role hierarchy. Private roles, indicated by the ‘*’ symbol, collect together permissions that will not be automatically inherited by roles further up the hierarchy. Thus, for example, the *Engineer1** role contains permissions which will not be inherited by the *Quality Engineer 1* role. However, such workarounds can lead to excessively complicated role hierarchy structures that are difficult to engineer and even to comprehend [moffett98]. They make it difficult to ensure that permissions are assigned in such a way as to enforce security requirements. Furthermore, there is nothing within the role-hierarchy itself which links the private roles with the corresponding public counterparts. Any semantic connection must be maintained by the administrators, as it is not captured by the role hierarchy itself.

The complexity of the role hierarchies that are required in order to model the requirements of real world organisations, as illustrated above, is a consequence of the fact that roles represent different aspects of the real-world organisation. In the above example, the *Engineering Dept.* role represents a structural element of the organisation, the *Engineer* role could represent a collection of users who have a particular capability or qualification, the *Production Engineer* and the *Quality Engineer* roles represent

particular functions in the context of a project, while the *Project Lead* and *Director* roles represent supervisory and decision making relations with respect to the roles beneath them.

In [moffett98], three distinct role hierarchies are identified within organisations, based on different relationships between roles, all of which form partial orders, namely:

- Generalisation/specialisation, or the *isa* relationship. This relationship is useful for describing hierarchies based upon specialisation. For example, a neurosurgeon *isa* surgeon.
- Supervision hierarchy, or the *supervises / is supervised* relationship. This hierarchy is based upon the organisational hierarchy. For example, the department manager role *supervises* the team leader role.
- Activity hierarchy, based on aggregation of activities. The hierarchy is based on the *more/less activities than* relationship and the roles are related to activities by the *responsible for* relationship. For example, financial control is responsible for financial forecasting and financial accounts which places the financial controller higher up the hierarchy than the financial forecaster role and the financial accountant role with respect to the more activities than relationship.

However, Moffett does not claim that these three hierarchies form an exhaustive classification. He allows that “*there may be other useful role hierarchies*”. The multiplicity of useful models for defining role hierarchies within organisations creates serious problems for designers of RBAC systems, since if a hierarchy is improperly used, there may be undesirable inheritance of access rights which may break the security constraints on the system. For example, a separation of duty constraint will be broken if the two exclusive roles are derived from any single role higher in the hierarchy. On the other hand, if role activation is limited by overly cautious constraints, deadlocked situations may arise where the security restrictions forbid tasks that are fundamental to the operation of the organisation.

There have been some attempts to derive an analytic role-modelling framework on the part of RBAC researchers, using methodologies borrowed from software requirements engineering, but there is little evidence of their utility in practice. Current approaches in [crook02] go some way towards recognising the complexity of real world organisations, by modelling roles under a number of different criteria, based on seniority,

qualification, function, work process, market and various environmental factors. However, regardless of the sophistication of the analytic process, there are some fundamental limitations to approaches which rely on role hierarchies. For these reasons, some researchers have questioned the utility of all role hierarchies in RBAC systems [moffet98, hine00].

Role hierarchies have a general problem when negative permissions are introduced to the model. The RBAC standard only supports the specification of positive permissions (exceptions are encoded as constraints rather than negative permissions) and role inheritance is defined in such so that a member of a senior role has a superset of the permissions of all inherited roles. Recent research has tended to highlight the usefulness of negative permissions in specifying and maintaining security policy [samarati00] and most modern AC systems support them [e.g. damianou02a, miege05, bacon02]. RBAC inheritance assumes that senior roles have a subset of the permissions of junior roles, but if a senior role includes negative permissions, this assumption is broken. Furthermore, if a senior role includes a positive permission for accessing a particular object and a junior role includes a negative permission for the same object, a conflict arises.

A1.8 RBAC – Extensions & Variations

Although the ANSI RBAC models enjoy several immediate advantages over DAC systems, they are still severely limited in terms of their expressiveness. Users are assigned to roles, permissions are assigned to roles and the only conditionalities that can be expressed by the system are the SSD and DSD constraints which allow administrators to specify cardinality limits on subsets of roles that may be activated or assigned to a user simultaneously. Although this allows many commonly required separation of duties constraints to be expressed, there are many types of conditions which can not be expressed through the standard RBAC models. This has led to the creation of a variety of extensions, modifications and alternatives to the basic RBAC model. Other extensions to the RBAC model have introduced negative permissions, further grouping abstractions and have defined delegation models as well as extending its expressiveness and applying it to new application domains. The following sections briefly analyse the innovations of these extended and alternative models with a focus on the organisational models which they embody.

RB-RBAC – Rule Based RBAC

The Rule-Based RBAC (RB-RBAC) [alkahtani03, alkahtani04] extended family of models was developed based on RBAC to overcome some of its limitations. In particular, RB-RBAC better supports applications with large numbers of users, where manual user-role assignment is impractical, by allowing rule-based assignment of users to roles. One particular model of this family, labelled “*RB-RBAC-ve*”, introduces the concept of negative authorization to the RBAC models. Overall, the rule-based extensions merely add to the expressiveness of the RBAC models and make them more convenient to deploy in practice. The only real impact upon the organisational model is the fact that the negative authorisations contained in RB-RBAC-ve complicate the use of role hierarchies by introducing potential conflicts [alkahtani04]

TBAC – Task Based Access Control

RBAC, DAC and MAC are all based on a *subject – object* view of access control. The basic entities are subjects, objects and the permissions that subjects possess to access the various objects. Thomas and Sandhu have argued that this model is not suitable for many application domains, in particular workflow specification [thomas97a]. They have proposed a family of models for what they call “*active and enterprise oriented*” authorisation management, known collectively as Task-based Authorisation Controls (TBAC). TBAC models depend upon the specification of authorisation controls for tasks which are divided into *authorisation steps*. Each authorisation step has a number of *processing states* and the various TBAC models support the specification of dependencies between authorisation steps. Since TBAC is primarily designed for the specification of access control policies within automated workflow systems, it is only distantly related to RBAC and other models and it has little information pertaining to the organisation model.

TRBAC – Temporal Role-based Access Control Model

A frequent requirement of access control models is the support of temporal constraints. For example, administrators may wish to specify that certain permissions may only be used within a certain time frame. In [bertino01], TRBAC, an extension to the RBAC model is described which adds support for temporal constraints on role activation. TRBAC allows roles to be specified as active in certain time periods and inactive in others. TRBAC supports periodic role enabling and disabling and temporal dependencies, expressed by means of role triggers. Role trigger actions may be either

immediately executed, or deferred by an explicitly specified amount of time. Enabling and disabling actions may be given a priority, which is used to solve conflicting actions. Although TRBAC allows greater expressiveness in the specification of access control policy, it does not extend the structural elements of the standard RBAC model as temporal information is rarely directly related to the organisational model.

TMAC – Team Based Access Control

TMAC is an extension to RBAC, first proposed in [thomas97b] and later proposed as a modified model as TMAC04 [alotaiby04]. It is designed to add features to the RBAC model which support RBAC's use in collaborative working environments. In TMAC, access control revolves around *teams*, where a team is an abstraction that encapsulates a collection of users in specific roles and collaborating with the objective of accomplishing a specific task or goal. Users who belong to a team are given access to resources associated with that team. TMAC has been extended to add the notion of *contexts* to the model, to make C-TMAC [georgiadis01]. In C-TMAC, contextual information concerning collaborative activities is associated with teams and used to dynamically filter user permissions at runtime. This enables greater expressiveness in the specification of rules by allowing permissions that are dependant on particular context states to be specified.

The team abstraction in TMAC adds information to the organisational model embodied in the standard RBAC model. Teams serve both as structural and functional elements as they group together users in structures and these groupings are generally related to a particular collaborative function which the team is designed to carry out. However, teams are not related to one another and are only indirectly related to the role hierarchy. Therefore, although they introduce new information from which a certain amount of knowledge can be derived regarding the organisation being modelled, it is not possible to integrate this information into a single model of the organisation and there is no way to relate the functions of the teams to the functions of individuals or of the entire organisation. Furthermore, teams are designed to support collaborative activities within an RBAC model and they are not a general-purpose structural abstraction which can also be employed to model large, permanent structures within the organisation such as divisions or departments. Therefore, despite the added structural and functional information, TMAC's organisational model represents a partial and atomised view of

the organisation from which it is impossible to derive an integrated model of the organisation's structure, functions or decision making model.

CBAC – Coalition Based Access Control

The Coalition-Based Access Control (CBAC) model [cohen02] is an extension of RBAC which layers a number of grouping abstractions on top of the standard RBAC model, as well as adopting elements of Task Based Access Control. A CBAC organization is composed of *roles*, *teams*, *tasks*, *resources*, *functions* and *users*. In addition, organizations can be related to one another with the *coalition* and *partnerOrganization* abstractions. Unfortunately, the bond between the concepts of user, role, team and organization remains unclear [miege05] and although resources are explicitly modelled is unclear exactly how constraints defined across resources and across organisations are applied [hongfan06]. Furthermore, although it is clear that the model is rich enough to capture a wide range of organisational forms, there is no evidence that it has been successfully applied to any complex real-world organizations in practice. The multiplication of basic entities within the model, the flexibility of their interrelationships, and their interaction with policy renders the model significantly more complex in comparison to the RBAC standards, inevitably requiring significant analysis in order to enact particular configurations. The complexity of the model and the flexibility of the relationships between the entities complicates the task of modelling the policy requirements, rendering it more difficult to enact change to the system, in particular given the lack of a facility for truly decentralised administration

OrBAC – Organisation Based Access Control

The Organisation Based Access Control model, known as OrBAC [elkalam03], is designed for situations in which users have different permissions depending on the *organisational context* within which they find themselves. Like CBAC, OrBAC aims to address situations where the policy system manages resources that are controlled by multiple organisations where individuals may have different roles and permissions depending on which sub-organisation they are acting within. All roles, permissions and the definition of contexts are specified as being associated with a particular *organisation*. Organisations provide a scope for policies and all other entities defined in the system.

Permissions are denoted as *activities* on *views*, where activities and views are abstractions of actions and objects. A permission is expressed as a fact having the form *Permission(org, r, v, a, c)*, where *org* is the organisation, *c* is the context and *r* is the role permitted to perform activity *a* on view *v*. The above declaration can be understood as signifying “*users having context c, who possess role r are permitted to carry out activity a on view v, within organisation org*”. OrBAC also supports the specification of *Prohibitions*, *Obligations* and *Recommendations* in a similar fashion, which, together with the support for context-based specification of permissions, allows a greater expressiveness than that supported by the standard RBAC models. Since proposing the OrBAC model, An associated administrative model, known as AdOrBAC, has been defined [cuppens04]. AdOrBac is largely derived from ARBAC97, but extended by defining all entities within an organisational context..

From the point of view of the organisational model embodied by OrBAC, the model certainly contains a great deal of extra information pertaining to the structure of the organisation, compared to RBAC. The *organisation* abstraction can serve to model structural elements such as administrative domains, and functional units within an organisation as well as grouping sets of users together through their relationship with the organisation. Furthermore, the specification of permissions and other types of access control policies provides information regarding the relationship of the organisation to the resources that are managed by the system. As the administrative permissions are defined with respect to organisations, there is a map of administrative responsibilities to units within the organisation, although administrators are modelled separately from users. OrBac supports role hierarchies, specified with the *sub_role(r1, r2)* relation. All permissions of inherited roles are inherited by sub-roles, although exceptions can be specified. A hierarchy of organisations can also be created, using the *sub_organisation(org1, org2)* relation [miege05]. Roles can be inherited between organisations in the hierarchy, in either direction, up or down the hierarchy, although role inheritance has to be explicitly specified by the Senior Security Officer (SSO) and roles are either inherited in an all or nothing manner.

However, although OrBAC is certainly capable of capturing far more information about an organisation than the RBAC standard models, through the use of a single extra basic entity - which adds much less complexity to the model than is introduced by the large number of abstractions introduced in CBAC – the model remains somewhat limited in

its ability to express organisational configurations. While the *sub_organisation* relation allows organisational units to be related together as subordinate to each other, it does not support the relationship of organisations as partners and there is nothing in the model which can capture the notion of the coalition satisfactorily, nor is the organisation abstraction well suited to modelling more transient, flexible groupings such as project-oriented teams. The semantics of the *sub_organisation* relation are also left undefined, meaning that it is up to the SSO to decide what it means for one organisation to be a sub-organisation of the other, and due to the complexity of policy-engineering, this means that it is likely that the configuration of the system will diverge from the configuration of the real world organisation without any system-enforced constraints to how the relations between organisations and the roles within them are interpreted. The model also lacks a theoretical base which translates into a practical architecture that would support the decentralised management of policy rules which is an important requirements of many real-world multi-organisational situations. It implicitly assumes the existence of a system-wide SSO and a centralised policy repository. In practice organisations may wish to have ultimate control over the policy which applies to their own resources and may not wish to entrust their management to a multi-organisational policy system. Finally, like virtually all of the other access control models dealt with in this survey, there is a lack of evidence of the OrBAC model being applied to the management of complex real world organisations and thus one can have little confidence in its viability in practice.

A1.9 Access Control Policy Specification Languages

RBAC and the various associated access control presented above are generally formally specified as mathematical models. In order to implement them, a means of specifying the policies of the system and an implementation mechanism are required. A large number of different languages for specifying RBAC systems have been proposed. Most RBAC specification languages are closely tied to the formal models and their notation closely resembles formal mathematical logic. These languages generally suffer from a lack of intuitiveness and they often do not easily map onto implementation mechanisms. They require a strong mathematical background in their administrators, which can make them more difficult and time-consuming to use than the higher level policy specification languages described in the following section.

In general, logic-based approaches to policy specification allow formal reasoning about the specified policies, and enable properties of the specified system to be proved, but they are not aimed at human interpretation and do not directly map to an implementation. The authorisation specification language (ASL) [jajodia97] is an example of a formal stratified first order logic language for specifying access control policies. It provides support for role-based access control as a positive or negative requirement for membership of roles can be specified as part of the rule condition. The following is an example of an authorisation rule in ASL, which states that all subjects belonging to group Auditors but not to Accountants are authorised (signified by the *cando* predicate) to write to the audit file:

```
cando(audit_file, s, +write) ? in(s, Auditors) & ¬in(s, Accountants)
```

The *cando* predicate can also be used to specify negative authorisations, by changing the sign in front of the action in the *cando* predicate to a minus. In addition to the *cando* predicate, three other predicates are defined. The *dercando* predicate specifies derived authorisations based on the existence or absence of *cando* rules (i.e. other authorisations in the system). The final two predicates *do* and *done*, can be used to specify history-dependent authorisations based on actions previously executed by a subject. The language includes a form of meta-policies called integrity rules to specify application-dependent conditions that limit the range of permissible access control policies. Targets can be grouped together by type.

Several inadequacies have been identified in the ASL specification [damianou02a]. The language does not scale well to large systems because there is no way of grouping rules into structures for reusability – roles are not specified as collections of permissions, rather being specified as part of the condition of authorisation rules. Furthermore, there is no explicit specification of delegation and no way of specifying authorisation rules for groups of target objects that are not related by type. In a subsequent paper [jajodia01] the language has been extended with predicates which can be used to evaluate hierarchical or other relationships between the elements of a system such as the membership of users in groups, inclusion relationships between objects or supervision relationship between users, all of which add to the organisational model embodied within the system's policy specification by relating groups of users to each other and structuring objects into collections. However, the language still lacks any means to delegate authority and assumes that policy rules will be written by security

administrators, meaning that the organisation's management of the policies themselves are not integrated into the model.

There are various other closely related examples of access control policy specification languages which are based upon formal logic. Barker adopts a similar approach to ASL to express a range of access control policies using stratified clause form logic, with emphasis on RBAC policies [barker00]. Ortalo describes a language to express security policies in information systems based on deontic logic [ortalo98]. Chen et al. describe a language based on set theory for specifying RBAC state-related constraints, which can be translated to a first-order predicate-logic language [chen95]. RSL99 is a formal language, which extends these ideas introduced and can be used for specifying separation of duty properties in role-based systems [ahn99]. The language covers both static and dynamic separation of duty constraints, and its grammar is simple, although the expressions are rather complicated and inelegant. Since time is not defined as part of the state of an RBAC system, these two proposed languages cannot specify temporal constraints, although that is addressed in the TRBAC extension to RBAC described in [bertino01].

A1.10 Conclusion

The preceding survey of organisational models within access control systems is instructive in several ways. A large number of variations and extensions on the role-based standard which augment roles with higher-level abstractions have been proposed and developed by researchers in the last few years. That researchers have been repeatedly motivated to create such models strongly suggests that RBAC models are, by themselves, insufficient to adequately capture many aspects of an organisation's operations for the purpose of applying policy based management solutions. Furthermore, the fact that multiple, mutually exclusive and often contradictory models are the subjects of active research measures suggests that it is unclear which, if any, of the models is likely to be widely adopted or become standardised. Indeed, the lack of experimental data – and in most cases the lack of anything more than a handful of textbook examples – means that it is even unclear whether any of the models is capable of modelling a wide range of real world organisations without becoming unmanageable due to the complexity of maintaining the policy system.

The organisational models embodied within role-based AC systems focus primarily on defining the permissions for individual users. In all the RBAC-derived models, role

membership remains the core determinant of access control decisions – even when other variables are significant. Roles are a relatively low-level grouping abstraction; they are simply collections of permissions which are mapped to sets of users. Thus, RBAC models use a bottom-up approach to constructing a model of the organisation. The overall organisational model is built up from the set of roles and their relationships to each other, rather than by decomposing the organisation into component structures and functions in a top-down manner. As a result of this approach, role-based models exhibit several limitations when attempting to model the collective aspects of organisations.

Structural & Functional Units

The various higher-level structural and functional units of real-world organisations – divisions, departments, regional branches, teams, committees, boards, teams and so on – are not captured by the standard RBAC models. Although TMAC, CBAC and OrBAC do provide support for structural grouping abstractions which can also have a functional significance, they do not define how the policies defined within these abstractions relate to one another, nor how they relate to the whole. In TMAC teams are modelled in isolation from one another and there is nothing within the model which allows them to be directly related to one another or to the system's global function. The OrBAC model includes the *sub_organization* relation between organisations, but specifies no particular semantics for inheritance of roles across organisations and leaves it up to security administrators to define whether roles are inherited up, down or not-at-all between organisations and their sub-organisations. Similarly, the relationships between the various entities in the CBAC model are not well defined and it is unclear how policies defined in different organisational contexts interact. Finally, few of the models define abstractions which relate managed objects to one another or which allow users, or groups of users, to be tightly coupled to groups of objects.

Model Construction and Administration

Few of the RBAC models attempt to integrate the model of how the system is administered into the user model to build a unitary model of the organisation. Administration is either specified as a distinct administrative hierarchy, as in ARBAC97, or partially related to the user model through the organisational scope, as in AdOrBAC, or left undefined altogether. While the role abstraction may represent a bottom-up approach to modelling the organisation, the modelling process in RBAC-derived models generally takes a decidedly top down approach. Permissions are

assigned to roles by specialist security administrators and, although many delegation models have been proposed, they are normally concerned with temporary transfers of permissions, with all permanent changes to the system's state carried out by administrators. Thus RBAC models are built from the top-down, using low-level abstractions. This contrasts strongly with how structures and policies in real-world organisations tend to be created. As they grow, organisations divide themselves into top-down abstractions – creating divisions, branches, teams and so on as they are required – and assign sub-goals to these organisational units. The actual rules governing the functions of the units is often defined through a bottom-up process whereby policies are added through progressive grounding as problematic situations are encountered and, as they evolve, authority and resources are frequently redistributed amongst the organisational units in a semi-permanent but never entirely static way.

Appendix A2. CBPMS Decision Algorithm

This appendix describes in full the CBPMS decision algorithm. It starts by defining a number of functions, which are then used to specify the operation of the decision algorithm.

Get Root Community Function

The first function is *get root community*, denoted *rootcom()*, which, given a request, locates the root community from which policies should be evaluated. In general, it can be expected that for centralised organisations the community specified in the policy decision request will be the root community of the organisation. However, in order to support the concept of hybrid organisations, any community within an organisation must be permitted to possess ultimate authority over a resource. It is assumed that policy decision requests which refer to resources that have been delegated from a federal community will specify this federal community in the policy request. There is a fairly simple algorithm which can deal with the situations where the community in the policy decision request is part of an organisation which has been delegated a resource authority from a federation that is the resource contained in the resource authority of the request. That algorithm is developed fully in the next chapter, for the moment the simpler version with the above assumption is presented.

$$\begin{aligned} \text{rootcom} : C_{\Sigma} \times RA_U &\rightarrow C_{\Sigma} \\ x = (DA_x, P, m), DA_x &= (da_{x1}, da_{x2}, \dots, da_{xn}), da_{xi} = (c_{xi}, ra_{xi}) \\ \text{rootcom}(c, ra) = x, x \in C_{\Sigma} &\Rightarrow (x \geq c) \wedge ((\exists ga_{xi} \in GA_x : c_{xi} = x) \wedge (ra_{xi} \geq ra)) \end{aligned}$$

The root community for a particular combination of a community and a resource authority (associated with a policy decision request) is any community that has greater or equal authority than the community and has granted itself resource authority which implies the requested resource authority. If the request is valid and the CBPMS is in a valid state, there should be exactly one community in the system which matches this rule.

Membership Function

Only those communities where the actor that is the subject of the request is deemed to be a member of the community need be searched for policies. Thus, the *test membership* function which maps onto a binary space is introduced. This function simply evaluates the membership policy with the policy request's context and then

evaluates this result to see if it should be interpreted as *applicable*. If it is *applicable* the actor is considered a member of the community, otherwise the actor is deemed not to be a member.

$$\begin{aligned}
&member : C_{\Sigma} \times PR_U \rightarrow B, B = \{0,1\} \\
&member(c, pr) = b \in B \\
&pr = \{ra, c, con\} \\
&c = \{DA, P, m\} \\
&b = a(e(m, con))
\end{aligned}$$

Root Membership Function

One cannot be sure that the root community for a particular resource is the root community of the organisation, due to the possibility of an organisation with a hybrid structure, a slightly different algorithm is used to check that the actor is a member of the root community that corresponds with the policy decision request. The organisational model demands that an actor must be a member of the root community before she can be a member of its children and so on down the community hierarchy. An actor may not thus be a member of a community without also being a member of its parent community. Therefore, in order to check if the actor is a member of the root community for a particular request, it must be ensured that the actor is a member of all the communities which have greater or equal authority than that community. This is done by defining the set of parent communities for the community and testing membership for all of these.

$$\begin{aligned}
&rootmember : C_{\Sigma} \times PR_U \rightarrow B, B = \{0,1\} \\
&rootmember(c_{root}, pr) = b \in B \\
&c_{root} = \{GA, P, m\} \\
&C_{parents} \subseteq C_{\Sigma} \\
&\forall c \in C_{\Sigma} : c \in C_{parents} \text{ iff } c \geq c_{root} \\
&b = 1 \text{ iff } \forall c \in C_{parents} : member(c, pr) = 1
\end{aligned}$$

Get Children Function

The algorithm also needs to be able to identify the children of a community in a community hierarchy. The *children* function is defined as follows:

$$\begin{aligned}
&children : C_{\Sigma} \rightarrow P(C_{\Sigma}) \\
&children(c) = K \in P(C_{\Sigma}) \\
&\forall x \in C_{\Sigma} : x \in K \text{ iff } (p(x) = c) \wedge (c \in f(x))
\end{aligned}$$

This function maps a community to the set of communities that are considered children of the community. This is defined as all those communities whose parent is the

community or who are federated to the community (i.e. they have the community in their set of federations). Since federal communities can delegate authority to the communities that are federated to them, those federated communities are considered children of the federal community. By considering federations in such a manner, the algorithm is greatly simplified as will be seen below.

Get Communities Delegated Authority Function

Another helpful function is the *communities delegated authority*, denoted *delegated*, function which maps a community and a resource authority to the set of child communities that to which the community has delegated authority that is relevant to the request. Thus function is used by the policy search algorithm to identify only those communities which might contain policies that are relevant to the policy decision request. It is specified as follows.

$$delegated : C_{\Sigma} \times RA_U \rightarrow P(PA_U)$$

$$delegated(c, ra) = G \in P(DA_U)$$

$$\forall x = (DA_x, P_x, m) \in C_{\Sigma} : x \in G \text{ iff } x \in children(c) \wedge \exists da_x \in DA_x, da_x = (c_x, ra_x) : c_x = c \wedge ra \geq ra_x$$

This function uses the children function, defined above, as part of its definition. The set of communities returned contains all communities that are children of community c and have a resource authority granted by c that implies authority for the resource authority that is the argument. Thus, only those resource authorities which are in scope with respect to the request's resource authority are considered.

Get Scoped Policies

Another useful function is the *get scoped policies*, denoted *scoped*, function. To recall, each community policy has a resource authority associated with it, which provides an *event scope* for each policy. This function returns the set of policies within a community that have greater scope than the resource authority in the argument. It is used to find all those community policies that are relevant for a resource authority in a policy decision request.

$$scoped(c, ra_{pr}) : C_{\Sigma} \times RA_U \rightarrow P(P_c)$$

$$scoped(c, ra_{pr}) = S \in P(P_c)$$

$$c = (DA, P_c, m)$$

$$\forall p \in P_c, p = (ra_p, ps) : p \in S \text{ iff } ra_p \geq ra_{pr}$$

Having defined this function, it is applied to the policy request in order to locate the root community from which the policy search starts.

Evaluate Community Policies

Now the function which evaluates a community's policy set with respect to the resource authority in the argument, denoted *evalpol*, is defined. This function follows a simple algorithm. First it identifies the set of policies that are within scope of the resource authority passed to it. Then it evaluates these policies and evaluates the returned decisions for applicability. Any of the returned decisions that are *applicable* in the context are added to a set of applicable policy decisions. If no applicable policy decisions are found in the community, then any sub-communities to which authority with a relevant scope has been delegated are identified. Each of these is tested in turn to search for applicable policy decisions. Note that this function calls itself recursively until a non-null set of applicable policy decisions is found in a community. It is also important to note that, if no applicable policy decisions are found in the community policy set, each and every child of the community must be searched for relevant policy decisions since each child represents a branch whose communities are not directly related in terms of authority and thus policy conflicts cannot be automatically resolved in this algorithm.

evalpol: $C_{\Sigma} \times PR_U \rightarrow PD_U$, $evalpol(c, pr) = pd$, $pd \in PD_U$, $c = (DA, P_c, m)$, $P_c = \{p_1, p_2, \dots, p_n\}$, $p_i = (ra_i, ps_i)$, $pr = (ra_{pr}, c_{pr}, con)$, $Q = scoped(c, ra_{pr})$, $PD_{applicable} \subseteq PD_U$, $\forall p \in Q: e(p) \in Z$ iff $a(e(p, con)) = 1$, $PD_{applicable} = \emptyset \Rightarrow (G = delegated(c, ra_{pr}))$,
 $PD_{applicable} \subseteq PD_U$, $\forall pd \in PD_U: pd \in PD_{applicable}$ iff $\exists g \in G: (member(g, ra_{pr}) = 1) \wedge ((pd = evalpol(g, ra_{pr})))$
 $pd = combine(PD_{applicable})$

The *applicable* and *evaluate* functions defined on the PRS are used in order to specify the contents of the policy decision set that this function builds. If there are no policies in scope, or if there are no applicable policies, the policy decision set will be the null set.

Policy Decision

Having defined all of the above functions, the algorithm for the decision function can now be succinctly defined. This has 3 simple steps. First the root community for the resource authority of the policy decision request is located, then the actor making the request is validated as a member of that community by invoking the *rootmember* function. Finally the policy in the root community is evaluated – which launches the recursive *evalpol* function above which will search the community hierarchy beneath the root community until it finds an applicable decision, or until there are no more communities to search which the actor is a member of or which have been delegated authority with greater scope than that of the request.

decision : $PR_U \rightarrow PD_U$, $decision(pr) = pd$, $pr = (ra_{pr}, c_{pr}, con)$, $c_{root} = rootcom(ra_{pr}, c_{pr})$,
 $(rootmember(c_{root}, pr) = 0) \Rightarrow pd = n()$
 $(rootmember(c_{root}, pr) = 1) \Rightarrow pd = evalpol(c_{root}, ra_{pr})$

Policy Decision Combination

The CBPMS policy decision algorithm, has now been entirely specified, except for one element, the *combine* function which combines the set of applicable policy decisions that have been generated by the algorithm and maps them to the specific policy decision that is the output of the function. This is where policy conflicts are detected. It can be seen from the above algorithm that this function maps a set of policy decisions to a single policy decision. There are three possibilities with respect to the set of policy decisions that is the argument to this function.

1. $PD_{applicable}$ is the null set. In this case the algorithm has failed to find any applicable policy decisions for the request and should return the policy decision which is interpreted as *not applicable* to the policy decision consumer.
2. $PD_{applicable}$ is a non-empty set and all the policy decisions in it are semantically equal according to the PRS specification. In this case any entry in the set is returned.
3. $PD_{applicable}$ is a non-empty set and not all of the policy decisions in it are semantically equal according to the PRS specification. This signifies a policy conflict within the system. The following section shows how meta-data can be integrated into the system, allowing automatic resolution of such conflicts, as well as allowing automatic analysis of the policy system to generate policy refinements with which to re-engineer policies in the event of undesired conflicts.

combine : $P(PD_U) \rightarrow PD_U$, $combine(PD) = pd$
 $PD = \phi \Rightarrow pd = n()$
 $\exists x, y \in PD : \neg x \cong y \Rightarrow pd = n()$
 $PD \neq \phi \wedge \exists x \in PD : \forall y \in PD : x \cong y \Rightarrow pd = x$

Having defined all of the above functions, the algorithm for the decision function can now be succinctly defined. This has 3 simple steps. First the root community for the resource authority of the policy decision request is located, then the actor making the request is validated as a member of that community by invoking the *rootmember* function. Finally the policy in the root community is evaluated – which launches the

recursive *evalpol* function, which will search the community hierarchy beneath the root community until it finds an applicable decision, or until there are no more communities to search which the actor is a member of or which have been delegated authority with greater scope than that of the request.

decision : $PR_U \rightarrow PD_U$, $decision(pr) = pd$, $pr = (ra_{pr}, c_{pr}, con)$, $c_{root} = rootcom(ra_{pr}, c_{pr})$,
 $(rootmember(c_{root}, pr) = 0) \Rightarrow pd = n()$
 $(rootmember(c_{root}, pr) = 1) \Rightarrow pd = evalpol(c_{root}, ra_{pr})$

Appendix A3. CBPMS Primitive Definitions

For the purpose of this discussion, and to simplify the notation, several things are assumed for each of these formal descriptions. Firstly, since these primitives only change the state of the community structure, $\Omega = (C_\Sigma, P, m)$, the state of the system can be considered as simply Ω . Thus, each one of these primitives has the effect of updating the state of Ω , which is denoted by writing $\Omega \rightarrow \Omega'$. The state change of the CBPMS could be added explicitly in each case by including the line: $CBPMS' = (\Omega', PRS, R_\Sigma)$ at the end of each definition. However, since this is present in every primitive, to aid brevity and clarity it is omitted and stated only here. Similarly, to be absolutely strict in logic and notation, it should be stated that the domain of each of these primitives is formed by the Cartesian product of the universe of all possible CBPMS systems, $CBPMS_U$ with the domains of the function arguments. The co-domain of all of the primitives is also $CBPMS_U$. So, for example, the $grant(c_{recipient}, ra)$ primitive below would be formally specified as:

$$grant(CBPMS, c_{recipient}, ra) : CBPMS_U \times C_\Sigma \times RA_U \rightarrow CBPMS_U$$

which can be interpreted as saying that the grant function maps a valid CBPMS state, a community and a resource authority to a valid CBPMS state. However, once again, since this is present in every definition, it will be stated once here and assumed that an argument to each function is the CBPMS itself. The state of the CBPMS can be decomposed as:

$$CBPMS = (\Omega = (C_\Sigma, p, f), PRS = (PS_U, PD_U, e, a, \cong, n), R_\Sigma)$$

and when these symbols are used in the primitives, they refer to the state of the system before the primitive was invoked. In order to avoid confusion, whenever there is any ambiguity about what entities particular variables refer to, subscripts are introduced and exactly what they refer to is defined. In the formal descriptions of the primitives, a *predicate condition* is present, which defines the requirements on the state of the system in order for the system to change state. If the condition is not met, the state of the system does not change.

Create a new Root Community - genesis

The *genesis* primitive, creates a new root community in the CBPMS set of communities C_Σ . The community is created as an empty structure with $DA = P = m = \phi$. Since the community is a root community, it has no external parent and thus the *parent* function must map the community to itself. Conceptually, a genesis operation is seen as the CBPMS manager incorporating a new autonomous organisation into the system. The CBPMS manager creates a community structure, which then goes on to live within the system and carry out actions in its own right. Genesis can be described as follows:

$$\text{genesis} : \Omega \rightarrow \Omega'$$

$$c_{\text{new}} = (\phi, \phi, \phi)$$

$$\Omega' = (C_\Sigma \cup (c_{\text{new}}), p \cup (c_{\text{new}}, c_{\text{new}}), f)$$

Remove a Root Community - expel

The *expel* primitive has the opposite effect and meaning of the genesis primitive. It removes a root community and all of its sub-communities (i.e. an organisation) from the system. Thereafter, the organisation will not be managed by the CBPMS and all of the policies and structures associated with all communities in the organisation will be removed. This reflects situations where the organisation ceases using the CBPMS in order to manage its resources. All references to the removed communities must be removed from the *federations* and *parent* functions.

$$\text{expel}(c_{\text{victim}}) : \Omega \rightarrow \Omega', c_{\text{victim}} \in \text{Roots}_\Sigma$$

$$\forall c \in C_\Sigma : (\neg c_{\text{victim}} \geq c) \Rightarrow c \in C'_\Sigma$$

$$\forall (a, b) \in p : (a \in C'_\Sigma \wedge b \in C'_\Sigma) \Rightarrow (a, b) \in p'$$

$$\forall (a, B) \in f : (a \in C'_\Sigma) \Rightarrow (a, B) \in f'$$

$$\forall (a, B) \in f', \forall b \in B : (b \in C'_\Sigma) \Rightarrow b \in B'$$

$$\forall (a, B) \in f' : (a, B') \in f''$$

$$\Omega' = (C'_\Sigma, p', f'')$$

Grant ultimate authority over a resource to a community – grant.

The grant primitive can introduce new resources into the community management system by assigning ultimate authority over the resource to a community within the system or it can move ultimate authority for a resource from one community to another. Conceptually, this operation is viewed as the CBPMS manager assigning ownership of a resource to a community. It should reflect real world events such as the organisation acquiring a new resource to be managed by the system, or even a resource being sold from one organisation to another.

$$\begin{aligned}
& \text{grant}(c_{\text{recipient}}, ra_{\text{new}}) : \Omega \rightarrow \Omega', ra_{\text{new}} = (r_{\text{new}}, t_{\text{new}}, a_{\text{new}}), r_{\text{new}} = ((T_{\text{new}}, \geq), (A_{\text{new}}, \geq)) \\
& \text{condition} : (c_{\text{recipient}} \in C_{\Sigma}) \wedge (r_{\text{new}} \in R_{\Sigma}) \wedge (t_{\text{new}} \in T_{\text{new}}) \wedge (a_{\text{new}} \in A_{\text{new}}) \in r_{\text{new}} \Rightarrow \\
& \forall c \in C_{\Sigma}, \forall da \in DA_c, da = (c_{da}, ra_{da}) \in DA'_c \text{ iff } \neg(ra_{da} \geq r_{\text{new}} \vee ra_{\text{new}} \geq r_{da}) \\
& \forall c \in C_{\Sigma}, \forall p \in P_c, p \in P'_c \text{ iff } \neg(ra_{da} \geq r_{\text{new}} \vee ra_{\text{new}} \geq r_{da}) \\
& \forall c \in C_{\Sigma} : c' = (DA'_c, P'_c, m) \\
& da_{\text{new}} = (c'_{\text{recipient}}, ra_{\text{new}}) \\
& c'_{\text{recipient}} = (DA'_{\text{recipient}} \cup da_{\text{new}}, P'_{\text{recipient}}, m) \\
& C_{\Sigma} = (c_1, c_2, \dots, c_{\text{recipient}}, \dots, c_n), C'_{\Sigma} = (c'_1, c'_2, \dots, c'_{\text{recipient}}, \dots, c'_n) \\
& \Omega' = (C'_{\Sigma}, p, f)
\end{aligned}$$

One of the consistency rules of the system is that no two communities can possess ultimate authority for resource authorities that are directly comparable. That is to say if there are two different communities, c_1, c_2 with sets of delegated authorities DA_1, DA_2 respectively, then:

$$\forall x = (c_x, ra_x) \in DA_1, \forall y = (c_y, ra_y) \in DA_2 : (c_1 = c_x \wedge c_2 = c_y) \Rightarrow \neg(ra_y \geq ra_x \vee ra_x \geq ra_y)$$

If any two delegated authorities within any community in the system were delegated by the community itself, then they must not be comparable with the *implies authority for* relationship. This is a consequence of the policy decision algorithm needing to know which single community is the root community for the particular resource authority defined in a policy decision request. If there is an existing community in the system which possesses ultimate authority for the resource (i.e. if the resource was already being managed by a community in the system) then this authority must be removed. Furthermore, since this community may have delegated authority for the resource to other communities, all those granted authorities which are implied by the granted resource authority must be removed.

Creating a Sub-Community - spawn

The *spawn* operation creates a new community directly beneath the community in the organisation's hierarchy. This new community is a subset of the creating community in terms of membership and directly beneath it in the hierarchy of authority. After spawning a new sub-community, the creating community can delegate authority it. Organisationally, this occurs when a community wants to separate out some of its functions into a more specialised community. This primitive is the fundamental operation which allows the organisation to define its own structure dynamically by creating new organisational units within existing units.

$$\begin{aligned}
& \text{spawn}(c_{\text{creator}}) : \Omega \rightarrow \Omega' \\
& \text{condition} : c_{\text{creator}} \in C_{\Sigma} \Rightarrow \\
& c_{\text{new}} = (\phi, \phi, \phi) \\
& \Omega' = (C_{\Sigma} \cup (c_{\text{new}}), p \cup (c_{\text{new}}, c_{\text{creator}}), f)
\end{aligned}$$

Sub-Community Removal - cull

The cull operation, the reverse of the spawn operation, removes the specified sub-community and all of its sub-communities from the organisation's community hierarchy. When a community is destroyed all of its components – community references, delegated authorities, the community membership rule and all policies defined for the community are also deleted from the system. Since the parent community is destroyed, all of these components are permanently out of scope and will never be invoked by the system, hence they can be safely removed and the problem of hanging communities no longer exists. Organisationally, this operation might be invoked when a functional unit within a community is no longer needed due to it either being improperly specified or else having been made redundant due to evolutionary changes in the organisation's practices. Conceptually this is seen as the parent culling its child.

$$\begin{aligned}
& \text{cull}(c_{\text{parent}}, c_{\text{child}}) : \Omega \rightarrow \Omega' \\
& \text{condition} : (c_{\text{parent}}, c_{\text{child}} \in C_{\Sigma}) \wedge (p(c_{\text{child}}) = c_{\text{parent}}) \Rightarrow \\
& \forall c \in C_{\Sigma} : c \in C'_{\Sigma} \text{ iff } \neg(c_{\text{child}} \geq c) \\
& \forall (a, b) \in p : (a \in C'_{\Sigma} \wedge b \in C'_{\Sigma}) \Rightarrow (a, b) \in p' \\
& \forall (a, B) \in f : (a \in C'_{\Sigma}) \Rightarrow (a, B) \in f' \\
& \forall (a, B) \in f', \forall b \in B : (b \in C'_{\Sigma}) \Rightarrow b \in B' \\
& \Omega' = (C'_{\Sigma}, p', f'')
\end{aligned}$$

Delegating Authority - delegate

Delegating authority over a resource to a sub-community is the basic operation which enables the dynamic distribution of rights and responsibilities through the community hierarchy which models the organisation. For a community to delegate authority ra , it must possess a resource authority which implies authority for ra . Once a community has been delegated a given authority, it can author policies with an authority scope that is less than or equal to the granted authority. These policies will apply to the membership of the community, effectively mapping the community's authority to individual actors' access rights.

$$\begin{aligned}
& \text{delegate}(c_{parent}, ra_{new}, c_{child}) : \Omega \rightarrow \Omega', c_x = (DA_x, P_x, m_x), DA_x = \{da_1, da_2, \dots, da_n\}, da_x = (c_x, ra_x) \\
& (c_{parent}, c_{child} \in C_\Sigma) \wedge (\exists da_p \in DA_{parent} : ra_p \geq ra_{new}) \wedge (c_{child} \in \text{children}(c_{parent})) \vee (c_{child} \in f(c_{parent})) \Rightarrow \\
& DA'_{child} = DA_{child} \cup ra_{new}, c'_{child} = (DA'_{child}, P_{child}, m_{child}) \\
& C_\Sigma = (c_1, c_2, \dots, c_{child}, \dots, c_n), C'_\Sigma = (c_1, c_2, \dots, c'_{child}, \dots, c_n) \\
& \Omega' = (C'_\Sigma, p, f)
\end{aligned}$$

Note that this function is specified so that a community can only delegate authority to a community that is one of its children (as defined in the children function above) or to a community to which it is federated. In effect this means that authority can only be distributed from a community to its direct children or to a federal community that it is federated to, or, in the case of federal communities, it can delegate authority to the communities that are federated to it. This rule serves to maintain the hierarchy of authority of the organisation and ensures that the system cannot evolve in such a way that it breaks the basic rule for centralised organisations that a parent's authority over a resource is a super-set of its children's authority over that same resource.

Recalling Authority - recall

The recall primitive is the inverse of the delegate primitive. It removes a delegated authority that a community has delegated to a community that is in its set of children. Since the child no longer possesses the resource authority, any delegated authority that the child community has delegated in turn to other communities (or any authority delegations that are purely implied by the removed delegated authority). In the CBPM conceptual model, it is said that the community recalls the authority from its child. Organisationally, this primitive is invoked when a community wishes to re-allocate a right or a responsibility from a sub-community in order to better meet the functional requirements of the parent.

The function works by first removing the delegated authority from the child. Then it removes any policies that are no longer in the scope of authority of the child (which is just for neatness as above). Then it examines any of the delegations that the child has made to its children and any of those that are no longer in scope are removed in turn, and so on recursively down the community hierarchy.

$$\text{recall}(c_{\text{parent}}, ra_{\text{recall}}, c_{\text{child}}) : \Omega \rightarrow \Omega'$$

$$\begin{aligned} \text{condition} : & (c_{\text{parent}}, c_{\text{child}} \in C_{\Sigma}) \wedge ((p(c_{\text{child}}) = c_{\text{parent}})(c_{\text{parent}} \in f(c_{\text{child}}))) \wedge (\exists da_p \in DA_{\text{parent}} : ra_p \geq ra) \wedge \\ & (\exists da_x \in DA_{\text{child}} : (ra_x = ra_{\text{recall}} \wedge c_x = c_{\text{parent}})) \Rightarrow \\ & \forall da_x \in DA_{\text{child}} : da_x \notin DA'_{\text{child}} \text{ iff } (ra_x = ra_{\text{recall}}) \wedge (c_x = c_{\text{parent}}) \\ & \forall p_x \in P_{\text{child}} : p_x \in P'_{\text{child}} \text{ iff } \exists da_y \in DA_{\text{child}} : (ra_y \geq ra_x), c'_{\text{child}} = (DA'_{\text{child}}, P'_{\text{child}}, m_{\text{child}}) \\ & \forall c_{\text{lower}} \in \text{children}(c_{\text{child}}), \forall da_x \in DA_{\text{lower}} : (c_x = c_{\text{child}}) \wedge \neg(\exists da_y \in DA_{\text{child}} : ra_y \geq ra_x) \Rightarrow \text{recall}(c_{\text{child}}, ra_x, c_{\text{lower}}) \\ & C_{\Sigma} = (c_1, c_2, \dots, c_{\text{child}}, \dots, c_n), C'_{\Sigma} = (c_1, c_2, \dots, c'_{\text{child}}, \dots, c_n), \Omega' = (C'_{\Sigma}, p, f) \end{aligned}$$

Creating Policy – policy

By creating a community policy for a particular resource that a community has authority over the community defines a resource authority scope and a policy specification which will be evaluated whenever a policy request arrives. In the case of access control policies, the community defines a mapping from the collective authority of the community to the individual access rights of its members. Since the organisational scope of policies is always confined to the community that specifies it, there are no repercussions for any other communities within the system.

$$\text{policy}(c_d, p_{\text{new}}) : \Omega \rightarrow \Omega'$$

$$\begin{aligned} c_d &= (DA_d, P_d, m), p_{\text{new}} = (ra_{\text{new}}, ps_{\text{new}}) \\ \text{condition} : & (c \in C_{\Sigma}) \wedge (ps_{\text{new}} \in PS_U) \wedge (\exists da_x \in DA_d : ra_x \geq ra_{\text{new}}) \Rightarrow \\ & P'_d = P_d \cup p \\ & c'_d = (DA, P'_d, m) \\ & C_{\Sigma} = (c_1, c_2, \dots, c_d, \dots, c_n), C'_{\Sigma} = (c_1, c_2, \dots, c'_d, \dots, c_n) \\ & \Omega' = (C'_{\Sigma}, p, f) \end{aligned}$$

Deleting Policy – revoke

The *revoke* primitive is the inverse of the policy primitive. It removes a policy that was previously defined by a community.

$$\text{revoke}(c_d, p) : \Omega \rightarrow \Omega'$$

$$\begin{aligned} c_d &= (DA_d, P_d, m), p = (ra, ps) \\ \text{condition} : & (c \in C_{\Sigma}) \wedge (p \in P_d) \Rightarrow \\ & \forall p_x \in P_d : p_x \in P'_d \text{ iff } p_x \neq p \\ & c'_d = (DA, P'_d, m) \\ & C_{\Sigma} = (c_1, c_2, \dots, c_d, \dots, c_n), C'_{\Sigma} = (c_1, c_2, \dots, c'_d, \dots, c_n) \\ & \Omega' = (C'_{\Sigma}, p, f) \end{aligned}$$

Membership Policy – gatekeeper

The *gatekeeper* primitive is invoked by the community to define the policy specification that is evaluated to decide who is a member of the community. In the conceptual model for access control, this is the user who attempts to access the resource. However, more

generally, this is merely a policy specification that is evaluated in the context of a particular policy decision request and is not inherently linked to any notion of individual identities. Recall that individuals (or policy decision requests) are only considered to be members of a community if they are also a member of its parent community. Thus, this operation can be conceptually viewed as the community deciding which members of its parent community are to be members of this community.

$$\begin{aligned}
& \text{gatekeeper}(c_d, ps) : \Omega \rightarrow \Omega' \\
& \text{condition} : (c_d \in C_\Sigma) \wedge (ps \in PS_U) \Rightarrow \\
& c'_d = (DA, P, ps) \\
& C_\Sigma = (c_1, c_2, \dots, c_d, \dots, c_n), C'_\Sigma = (c_1, c_2, \dots, c'_d, \dots, c_n) \\
& \Omega' = (C'_\Sigma, p, f)
\end{aligned}$$

Join Federation - *federate*

The *federate* primitive causes a community to join a federal community. A rule of the system is that a community cannot federate to a community directly related to it on the hierarchy same organisation as this would create a potential loophole which could undermine the hierarchy of authority and allow communities to delegate authority over resources to communities whose parents did not possess that authority. There is also no special requirement about the nature of the community that is being linked to. Communities can federate to any communities that they want, but unless the agreement is reciprocal, the system will not be able to transfer authority in either direction along it. Furthermore, having a separate root community which serves as the federal community allows the cooperation to be managed as an independent entity and greatly eases the specification of policies concerning the cooperation itself.

$$\begin{aligned}
& \text{federate}(c_d, c_f) : \Omega \rightarrow \Omega' \\
& \text{condition} : (c_d, c_f \in C_\Sigma) \wedge \neg(c_d \geq c_f \vee c_f \geq c_d) \Rightarrow \\
& f' = f \cup (c_d, c_f) \\
& \Omega' = (C_\Sigma, p, f')
\end{aligned}$$

Leave Federation – *withdraw*

The *withdraw* function removes a community from a federation. When a community withdraws from a federation, any resource authorities that have been delegated in either direction have to be removed from the system, since the link no longer remains.

$$withdraw(c_d, c_f) : \Omega \rightarrow \Omega'$$

$$condition : (c_d, c_f \in C', c_f \in f(c_d)) \Rightarrow$$

$$\forall c \in f(c_d), c \neq c_f \Rightarrow c \in f'(c_d)$$

$$\forall da_x = (c_x, ra_x) \in DA_d : (c_x = c_f) \Rightarrow recall(c_d, ra_x, c_x)$$

$$\forall da_x = (c_x, ra_x) \in DA_f : (c_x = c_d) \Rightarrow recall(c_f, ra_x, c_x)$$

$$\forall p_x \in P_d : (\exists ga_y \in DA_d : ra_y \geq ra_x) \Rightarrow p_x \in P'_d$$

$$\forall p_x \in P_f : (\exists ga_y \in DA_f : ra_y \geq ra_x) \Rightarrow p_x \in P'_f$$

$$\Omega' = (C'_\Sigma, p, f')$$

Liberate

The *liberate* function turns a sub-community into a root community. It is invoked by a parent community when it wants to liberate a part of itself and make it into an autonomous organisation in the CBPM model. Conceptually, this can be seen as an organisation splitting, or *floating* a part of its operation to make it an independent entity. Any authority that has been delegated to the liberated sub-community is null and void as there is no longer any relationship between the new community and its erstwhile parent.

$$liberate(c_a, c_b) : \Omega \rightarrow \Omega'$$

$$condition : (c_a, c_b \in C_\Sigma) \wedge (p(c_b) = c_a) \Rightarrow$$

$$\forall (a, b) \in p : (a \neq c_b) \Rightarrow (a, b) \in p'$$

$$p'' = p' \cup (c_b, c_b)$$

$$\Omega' = (C_\Sigma, p'', f)$$

Merge

The *merge* function is the reverse of the liberate primitive. It allows a formerly autonomous root community to become a child community of another community. Conceptually it is seen as two organisations merging. A community that merges into another community becomes a unit of that community.

$$merge(c_a, c_b) : \Omega \rightarrow \Omega'$$

$$condition : (c_a, c_b \in C_\Sigma) \wedge (p(c_a) = c_a) \Rightarrow$$

$$\forall (a, b) \in p : (a \neq c_a) \Rightarrow (a, b) \in p'$$

$$p'' = p' \cup (c_b, c_a)$$

$$\Omega' = (C_\Sigma, p'', f)$$

Appendix A4: Community Resource Modelling

There are many different ways to build a target tree from a community structure. There is nothing that dictates any particular rules about how the target tree for the community resource should be structured. Indeed, by constructing target trees in different ways we can facilitate different types of organisational models. For example, the community resource model from section 4.5, pictured in figure A4.1 can be modelled in many other ways.

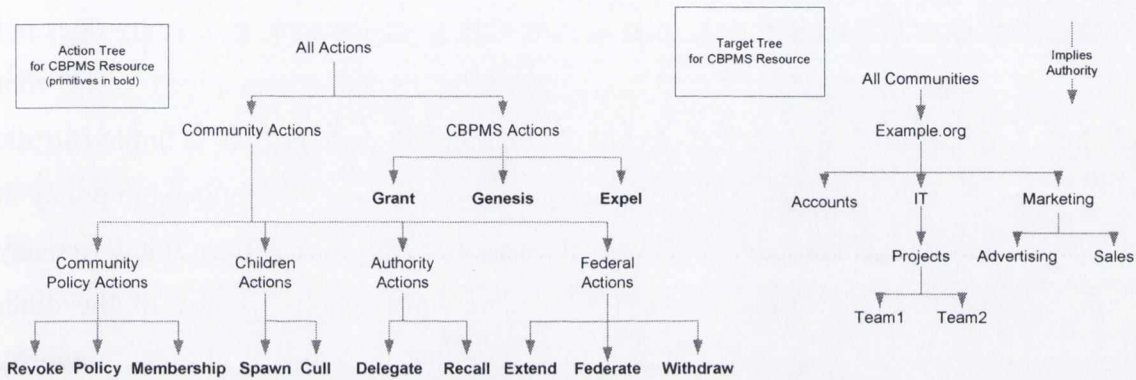


Figure A4.1 CBPMS Resource Tree & Section of Action Tree

For example, a possible alternative model would use the target tree in figure A4.2, where communities are not modelled as being underneath their parents. This model would be perfectly valid and is suitable for a situation where communities are completely self-managed and modelled as unrelated nodes on the target tree.

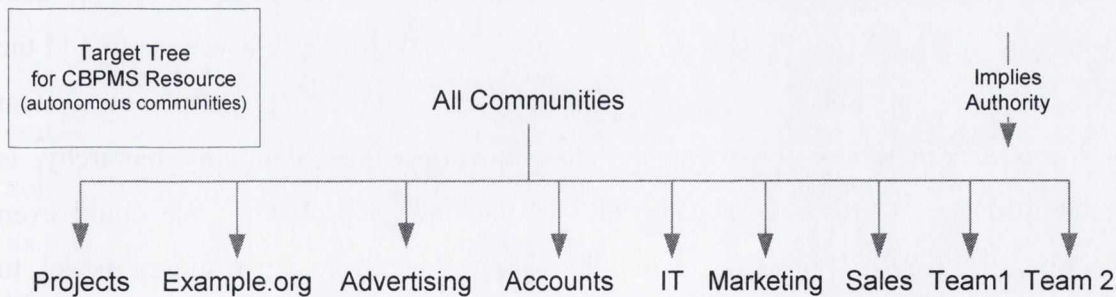


Figure A4.2 Target Tree for Self-Managed Community Example

We should note that, although the target tree has been changed, the community hierarchy remains unchanged. If we were to construct our resource model in such a manner, ultimate authority over each of the communities could be granted to the community itself by means of the grant primitive. For example, using such a resource target model, the following series of primitive invocations would create a new project

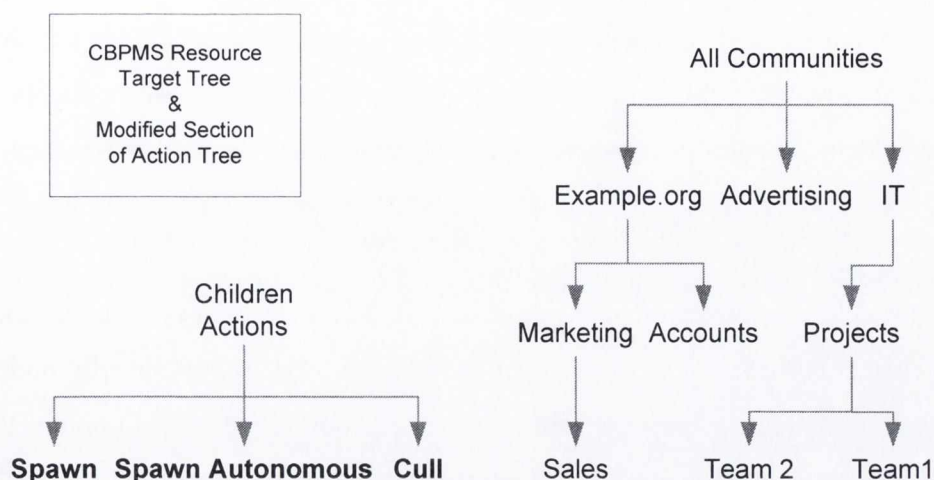
team community, Team 3 in the IT projects team, entirely democratically self-managed and autonomous of its parent.

```
spawn(Projects, Team3) grant(Team3, (CBPMS, Team3, Community Actions)
policy(Team3, p) : p = (ra, ps) ra=(CBPMS, Team3, policy), ps = (auth+
majorityVote{ when vote(majority, context.policy-request )})
```

This creates what we call a hybrid organisation. The root authority for all resources does not reside in the root community of the organisation. There is no way for the root community, representing the entire organisation, to dictate to the sub-communities how they should structure themselves – what policies they will write, who should be a member, what sub-communities they spawn or cull, since the root community does not have any authority over the resources representing its sub-communities. The root community's control over the other communities in the organisation is limited to the ability to cull its immediate children and its ability to re-define its membership policy in such a way as to exclude individuals from the entire organisation. It can not define any policies relating to the use of the CBPMS community actions by any of the other communities, since it possesses no authority over these resources, ultimate authority over which has been granted to the communities themselves.

Resource Model Plasticity

One thing to note about this situation is that the resource model of the CBPMS must change with each successful invocation of one of the management primitives. For example, each time a spawn(c) primitive is successfully invoked a new community is added to the target tree of the CBPMS resource model. This, however, is not a problem as long as the target tree of the resource model is updated in such a way as to add the nodes to their correct place in the target hierarchy. If, as in the first example, the system of locating communities on the target tree, mirroring the community hierarchy, is maintained by any resource updates, we do not have a problem. We could even combine the two different ways of building a target tree from our resource model, to allow communities to spawn sub-communities as autonomous or subordinate entities, leading to a target tree and action tree like that shown in figure A4.2.



In this example resource model, we have added another action node **Spawn Autonomous**. By invoking this operation a community will create a new autonomous sub-community rather than positioning the new sub-community beneath it in the target tree. In the example, the advertising and IT communities are fully self-managing and autonomous from the community representing the entire organisation. Although the spawn autonomous action is not a primitive of the system, as long as the policy management tool can implement it through a sequence of primitive invocations, we can use any such non-primitive operation in our action tree. In this case, the operation would be implemented by invoking the spawn primitive, triggering a mechanism to integrate the new community into our target tree as an independent resource rather than as a lower node, followed by the grant primitive to give the community ultimate authority over the resource representing itself.

In order to do this we need to be able to update the target tree dynamically. To solve this problem would could specify obligation policies which apply to the invocation of certain primitives in order to update the resource model accordingly, but a better solution is to generate the resource model dynamically from the community structure – as we show in our implementation in the following chapter. First, however, we will examine what the implications for the system are when we modify our resource model.

Adding Target and Action Nodes

In general, it is quite possible for us to re-model resources dynamically – our target tree and action tree are merely simply hierarchies that our policy selection algorithm follows, oblivious to the meaning of any of the targets or actions. However, we should be aware of what effect changes to the target or action tree will have on existing delegations of authority and community policy. The situation of adding new nodes

without changing the existing hierarchical relationship between existing nodes is not problematic since all existing delegations of authority and all existing policies will still be valid with unchanged semantics. The only thing to bear in mind is that existing policies with resource authorities which imply the new node will apply to it too.

Removing Target and Action Nodes

Removing nodes is slightly more complicated. If the removed node is a leaf node in the tree, it will simply mean that any delegations of authority for that specific node will be no longer valid and any existing policies whose resource authorities contain that node will also be invalid. However, simply removing those policies from the system will return the system to a fully valid state. If the node is not a leaf node in the tree, we will find ourselves with delegations of authority and policies that are no longer valid, but there is the potential for delegations and policies which depend upon the invalid delegations and are themselves valid since they contain a reference to a node in their resource authority that is beneath the removed node in the hierarchy. To deal with this case we need to replace the invalidated delegations and policies with delegations and policies which list all of the nodes beneath the removed node, thus saving our existing policies and delegations that are still valid, yet returning the system to a valid state.

Modifying the implies authority Relationships

Finally there is the case of changing the resource model in such a way that we change the relative authority relationships of nodes. That is to say that if there are two nodes in the original model which have an authority relationship and the two nodes are also present in the new model, then the authority relationship is either reversed or not present in the authority tree (i.e. they are not directly comparable) or vice versa. Then the basic authority hierarchy of authority has been changed and we encounter a large number of potential problems – policies authorising actions which are suddenly beneath them in the hierarchy which the policy was never intended to do. In these cases, since the basic semantics of the model have changed, it will be necessary to remove all delegations and policies whose resource authorities refer to the re-modelled resource and re-start the process of delegation and policy definition.

We can state this as the following rule for the consequences of a modification to a resource model in terms of the community structure.

A resource model is updated : $r_i \rightarrow r'_i$

We use the symbols r and q to describe the implies authority relation : $r_i = ((T_i, r), (A_i, q)), r'_i = ((T'_i, r'), (A'_i, q'))$

The condition for a modified authority relationship is :

$$\exists t_1, t_2 \in T_i : (t_1, t_2 \in T'_i) \wedge ((t_1, t_2) \in r) \wedge \neg((t_1, t_2) \in r') \vee \exists a_1, a_2 \in A_i : (a_1, a_2 \in A'_i) \wedge ((a_1, a_2) \in q) \wedge \neg((a_1, a_2) \in q')$$

which causes the following change in state in the community structure $\Rightarrow C_\Sigma \rightarrow C'_\Sigma$

$$\forall c \in C_\Sigma, c = (DA, P, m), \forall da = (c_p, (r, t, a)) \in DA : r \neq r_i \Rightarrow da \in DA'$$

$$\forall c \in C_\Sigma, c = (DA, P, m), \forall p = ((r, t, a), ps) \in P : r \neq r_i \Rightarrow p \in P'$$

$$\forall c \in C_\Sigma, c = (DA, P, m) : c' = (DA', P', m)$$

$$C'_\Sigma = \{c'_1, c'_2, \dots, c'_n\}$$

Appendix A5: Expressiveness of CBPMS

It is important to note that Community Based Policy Management is not a complete policy system, designed to be capable of expressing a wide range of conditions, constraints and policy requirements directly. The CBPMS approach considers the construction of the organisational model to be separable from the specification of policy and only concerns itself with the former. Apart from some simple semantic requirements on the policy reasoning engine, the CBPMS is policy-language agnostic, designed to provide an administrative and organisational framework and not to provide policy specification. Thus, in general, questions that take the form “can the CBPMS express this policy” can be answered with “if the underlying policy reasoning engine and specification language can express it and if any contextual information required to evaluate the policy is accessible to the CBPMS context-discovery service (CMDS)”. In general, the CBPMS can be configured to express whatever requirements can be expressed with the policy specification language employed. However, there are certain policy requirements which necessitate that the organisational model interacts closely with the specification of policy and it is worth looking at these a little closer.

Separation of Duty

The ability to enforce separation of duty constraints are a frequently cited requirement of policy systems. Separation of duty constraints reduce the possibility of fraud or damaging errors by partitioning the tasks and associated privileges so cooperation of multiple users is required to complete sensitive tasks [damianou02b]. Authorised users are assigned privileges which do not lead to execution of conflicting tasks. One of the frequently cited benefits of RBAC is that roles provide a convenient abstraction for partitioning such tasks. Communities provide exactly the same facility as roles in this respect – separated duties can be divided into sub-communities with membership rules which prevent users from being a member of conflicting roles simultaneously (this information is available as part of the CBPMS context). Community membership rules can be used to apply a range of similar requirements on sets of communities which an individual can be a member of simultaneously, being used to specify role cardinality limits, for example.

Encoding Exceptions

The CBPM system provides a number of ways in which exceptions to policies can be encoded – in this context exceptions are considered to be certain situations which should be treated differently to the normal policy rule. Most simply, the CBPMS specification requires the PRS to provide a means of distinguishing between policies that are applicable or not. That means that any constraint, or set of constraints, that can be expressed in the policy language can be applied to any policy rule and that rule will be ignored when the constraint is not met. Thus, to take a very simple example, the following policy rule: `permit read fileA`, could have an exception added to it by specifying a constraint such as: `if user != 'john'`.

Although this is a general purpose way of specifying exceptions to rules, it embeds the exceptions directly into the policy rules which, for reasons of maintainability, is a bad idea. It is particularly undesirable to embed information about the organisational model within the rules in such a manner. A better approach to specifying those exceptions which limit the rule's applicability to a subset of the users of a community is to form two new sub-communities and to sub-divide the membership of the new community between them according to requirements. The policy rule is then specified in the non-exceptional community. By sub-dividing communities in such a manner, the exception is encoded as part of the organisational model and the groups of users on either side of the divide are modelled as organisational units – with the potential to evolve and take on a life of their own.

Generality versus Specificity

The CBPMS decision algorithm resolves conflicts in favour of the policies defined further up the community hierarchy. This is a precedence which favours the general over the specific. By contrast, it is often convenient to define policies in such a way that the specific policies have priority over the general policy. Indeed, a progressive grounding approach should seek to start with general rules and flesh them out by adding higher-priority rules which dealt with more specific situations. However, this contrast is deceptive. The CBPMS conflict resolution obeys the hierarchy of authority and thus the generality and specificity is related to hierarchy, not the semantic scope of the policy. CBPMS policies are given priority according to the authority of those who define them – those who have general authority being considered to have a higher priority than those with more limited, specific authority. The employment of this hierarchy does not

prevent, in any way, the development of a policy set where specific policy rules have priority over more general policy rules. It merely constrains the system so that it is impossible for any decision maker to create a higher priority rule than an existing rule defined by decision makers higher up the organisational hierarchy without their agreement or the agreement of their superiors.

Cross-Organisational Collaborations

The CBPMS adopts a model of the organisation where authority and organisational units are recursively sub-divided into smaller units with more specific function. However, there are many situations in which organisational structures do not follow this strict sub-division. For example, project groups, or inter-disciplinary committees, can draw their membership from multiple different divisions of an organisation and groups from entirely different areas of an organisation might collaborate on a particular task. The federate primitive allows the CBPMS to capture such arrangements in general. Any community in the hierarchy can federate to any other community in the organisation that is not a direct ancestor or descendant of the community (which could break the basic hierarchy of authority). The federal link provides an abstraction for managing the cross-organisational collaboration and a controlled means of passing resources between the partners in the federation.

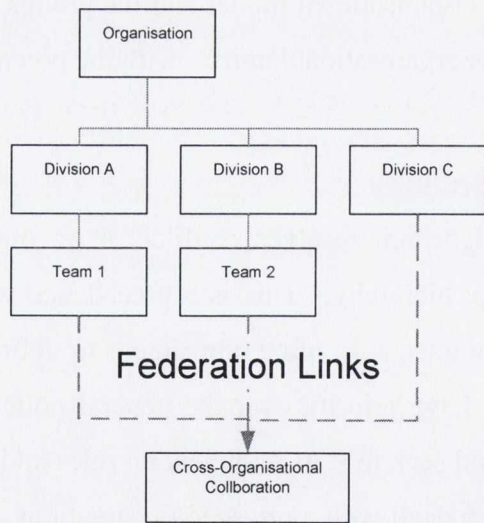


Figure A5.1 Using Federations to Represent Cross-Organisational Links

There are, however, certain situations in which the use of a federation will not capture the organisation’s requirements. The federal community will be limited to specifying policies which apply to resources directly delegated from the constituent communities

and will have subordinate authority to the delegating community. There are situations where, for example, an organisation may wish to instead create a cross-organisational unit that is managed independently of the various smaller collaborating units and may have authority over resources that none of the components enjoy. This is achievable by simply creating a sub-community of the lowest community in the hierarchy which contains a superset of the membership, in terms of people and resources, required for the collaboration. Then the necessary resources can be delegated to this community, and a suitable membership rule defined, to give this collaborative community an independent existence

By combining the two techniques – creating federal links between communities, and creating sub-communities at higher levels of the trees to encapsulate subsets of the organisation, a wide range of complex organisational topologies can be captured.

Collective Decision Making

A detailed discussion of collective decision making mechanisms and the various voting and approval systems in existence is beyond the scope of this thesis. However, there is a large volume of published material on the matter [ferscha99] and a wide variety of systems available which implement various collective decision making strategies such as majority or plurality votes, the Dodgson rule [dodgson76], Borda counting [dym02], the Nanson rule, the Hare rule, the Coomb rule and so on [tideman06]. Whichever particular system, strategy or set of strategies is used is not important from the point of view of the CBPMS. All that is of concern is that a collective decision making system be available to the community and that it can be engineered to obey the semantics required to integrate it into the CBMPS.

Consider a simple collective decision making system which supports several different types of decision making strategy. For example the system might support the following strategies, *majority vote* – where the winner is the majority of votes cast after a certain amount of time, *at least x votes for and at most y votes against* where a binary decision is deemed passed if the formula is met for a specified x and y after a certain amount of time, *x approvals*, where a proposal is deemed passed if it receives x proposals within a certain amount of time. In order to integrate policies which require collective decision making procedures to have been completed before an action is authorized, this information must be expressed in the policy specification language. Exactly the best

way to do this depends upon the flexibility of the policy specification language. However, in general, it amounts to a similar process. An authorization policy is created for the relevant resource authority and a condition is attached to the policy specification which signifies “only authorized after decision making strategy x has approved it”. This condition could be as simple as a check to see if a certificate issued by the decision making system is present in the context of the policy request, leaving the work of populating this context to other parts of the system, or it could be integrated into the policy specification language. So, for example, a Ponder constraint could be defined that looked something like the following in order to associate certain actions on certain targets with required decision making procedures: `when inst auth+{ when vote(majority, context.policy-request)}` to specify a majority decision is required or: `when inst auth+ {when vote(approval(2), context.policy-request)}` to specify that the action requires two approvals before it will be authorized. Being able to specify voting rules within policy specifications in such a manner is dependant on the policy specification language being expressive enough to implement the following. When a resource authority arrives that is within the scope of this policy, the evaluation of the condition will invoke a decision making process that will be associated with the policy decision request (derived from the context). Only after this decision making process has completed will the action be authorised or not depending on the result of the decision making procedure.

The practicalities and problems of integrating collective decision making systems into the CBPMS will be discussed in the following chapter, but for the moment it is assumed that such a facility is available within the policy specification language and PRS. Thus, to return to the discussion of the control communities exercising their authority, the following policies can be specified for the *board* control community:

```
policy(board, p) : p =(ra, ps) ral=(CBPMS, Example.org, Community
Policy          Actions)          ps=(auth+          majorityVote{          when
vote(majority,context.policy-request )}
```

This policy enables the board to take any community policy action on any target community beneath the Example.org community in the community hierarchy after a majority vote of the board. Whenever a board member proposes an action with the community policy management tool, the tool will send a policy request to the CBPMS

which will cause this policy to be evaluated and a voting procedure to be initiated. If the voting procedure is successful, the action will be authorised. A similar policy can now be applied to the control communities of each community in the original hierarchy, enabling each community to be managed by majority vote, or different voting rules could be applied to different communities as required. Indeed, to create an entirely democratic decision making process, the control communities can be eliminated throughout the hierarchy and these policies can be applied to the original communities themselves – thereby making the entire organisation democratic rather than hierarchical.

However, one immediate problem which arises is that, although the board of directors, or the entire membership, may have authority to define the organisation's policies, they are unlikely to always be the same individuals who actually use the policy management tool in order to author the policies. Instead, it is likely that, in many situations, specialists who are familiar with the tool will translate the high level policy decisions of the decision makers into specific community actions and specific policy definitions. This can easily be incorporate this into this model by creating further control communities to control the first layer of control communities and populating these communities with the technical specialists who will implement the decisions of the control community within the system itself. This configuration can be described as a *technocracy*, particularly when the technical sub-communities control the community representing the functional unit rather than the control community for that unit.

Appendix A6. Isancestor Function

The *Isancestor* function recursively climbs the community hierarchy starting from community *c2*, until it either reaches the community *c1*, in which case it returns true, or else until it reaches a root community (identified as being its own parent), whereupon it returns false. The pseudo-code listing is below.

```
Boolean Function isAncestor(CommunityID c1, CommunityID c2)
  IF the repository contains communities with ID c1 and c2 THEN
    IF c1 = getParent(c2) THEN
      return true
    ELSE IF c2 = getParent(c2) THEN
      return false
    ELSE
      return isAncestor(c1,getParent(c2))
    END IF
  ELSE
    return false
  END IF
END Function
```

Appendix A7: CBPMS XML Schema

Community Element

```
<xs:element name="Community" type="cbpms:CommunityType" />
<xs:complexType name="CommunityType">
  <xs:sequence>
    <xs:element ref="cbpms:Title" minOccurs="1"/>
    <xs:element ref="cbpms:Description" minOccurs="0"/>
    <xs:element ref="cbpms:Parent" />
    <xs:element ref="cbpms:Children" />
    <xs:element ref="cbpms:Federations" />
    <xs:element ref="cbpms:GrantedAuthoritySet" />
    <xs:element ref="cbpms:DelegatedAuthoritySet" />
    <xs:element ref="cbpms:CommunityPolicySet" />
    <xs:element ref="cbpms:MembershipPolicy" />
  </xs:sequence>
  <xs:attribute name="CommunityId" type="xs:anyURI" use="required"/>
  <xs:element name="Description" type="xs:string"/>
  <xs:element name="Title" type="xs:string"/>
  <xs:attribute name="ChildCombiningAlgId" type="xs:anyURI"/>
</xs:complexType>
```

Note that this structure mirrors the community data structure described in section 4.1.2 with one exception: a *GrantedAuthoritiesSet* element has been added, which is explained below. The *CommunityID* attribute is a URI, which may point to a community stored on a different CRMS. The Child Combining Algorithm attribute allows a community to specify an automatic conflict resolution algorithm for policy conflicts that occur in different child-branches of the community hierarchy beneath this community, as described below. Also note that each community element contains mandatory description and title elements – which are used to provide a high level natural language description of the function of the community and the community’s natural language title. The *title* and the *description* elements are used throughout the CBPMS schema in order to assist human comprehension of the system and to provide an audit trail in terms of the translation of high level goals into concrete policies. Both are defined as an XML Schema string type and allow free-form content.

Parent, Children, Federations & Community Relation Elements

```
<xs:element name="cbpms:Parent" type="xs:anyURI" />
<xs:element name="cbpms:Children" type="cbpms:CommunitySetType"/>
<xs:element name="cbpms:Federations" type="cbpms:CommunitySetType"/>

<xs:complexType name="CommunitySetType">
  <xs:sequence>
    <xs:element ref="cbpms:CommunityRelation" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:element name="cbpms:CommunityRelation" type="CommunityRelationType" />

<xs:complexType name="CommunityRelationType">
  <xs:sequence>
```



```

        <xs:element ref="cbpms:Description" minOccurs="0"/>
        <xs:choice minOccurs="1">
            <xs:element ref="cbpms:Community"/>
            <xs:element ref="cbpms:CommunityIdRef" />
        </xs:choice>
    </xs:sequence>
</xs:complexType>

<xs:element name="cbpms:CommunityIdRef" type="xs:anyURI" />

```

Each community contains a *parent*, *children* and a *federations* element. The *Parent* element simply contains the URI of the parent community, which can refer to a remote CRMS. If the community is a root community the parent community URI should be identical to the ID of the community. The *Federations* and *Children* elements are sequences of community relation elements. The community relation contains a mandatory description element – again ensuring the recording of a high-level description of the purpose of this federal or child community from the point of view of this community. The *CommunityRelation* element can contain either a reference to a community record or an embedded community element – thus supporting the storage of an entire organisation’s community hierarchy in a single stream of XML. Note that the description element is present in both the community element itself and in the community relation element. When the community set is the *federations* element this allows the inclusion of a description of the federal relationship from the point of view of the federating community as well as from a global point of view. When the community set is the set of children, the CBPMS software can either replicate the information across both, or else use them as distinct descriptions from the different points of view of the parent community and the community itself.

Granted Authorities & Delegated Authorities Elements

```

<xs:element name="DelegatedAuthoritySet" type="cbpms:DelegatedAuthoritySetType" />
<xs:element name="GrantedAuthoritySet" type="cbpms:DelegatedAuthoritySetType" />

<xs:complexType name="DelegatedAuthoritySetType">
    <xs:sequence>
        <xs:element ref="cbpms:DelegatedAuthority" maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>

<xs:element name="DelegatedAuthority" type="cbpms:DelegatedAuthorityType" />
<xs:complexType name="DelegatedAuthorityType">
    <xs:sequence>
        <xs:element ref="cbpms:Title" minOccurs="1"/>
        <xs:element ref="cbpms:Description" minOccurs="0"/>
        <xs:element ref="cbpms:CommunityIdRef" />
        <xs:element ref="cbpms:ResourceAuthority" />
    </xs:sequence>
</xs:complexType>

```


The *DelegatedAuthoritySet* element contains a record of all the resource authorities that this community possesses. Each *DelegatedAuthority* contains a record of the resource authority granted and a reference to the community to which the authority was granted. The *GrantedAuthoritySet* element contains a record of all the authority delegated by this community – the converse of the *DelegatedAuthoritySet* element and thus is replicated data which does not affect the basic model or operational semantics. In the case of the delegated authorities, the community reference points to the community that delegated the authority to this community. In the case of the granted authorities, the community ID points to the community that this community has delegated the authority to. Also note that each delegated authority element also contains a mandatory description field – which is used to provide a high level natural language description of the reason for the delegation.

Adding the *GrantedAuthoritySet* element to the CBPMS schema allows the CRMS to be more securely distributed across multiple independently managed domains and helps to optimise the implementation of the policy search algorithm, by limiting the number of community structures that must be retrieved from storage. This works since the policy search algorithm can simply retrieve a single community structure and if there are no applicable community policies, it can determine which further community structures to search by simply checking which communities have been granted relevant authorities by this community without having to retrieve all the children and federal community structures. From a security point of view it means that, in an arrangement where the community record management and storage is distributed across multiple administrative or organisational domains, a community can guarantee that only the authority that this community has legitimately delegated – as recorded in the community record itself – will be visible to the algorithm and that corruption of community records in other domains cannot provide access to authority that this community has not granted. In particular this means that this algorithm guarantees that the organisation which is the owner of the resource can host its CMRS locally and limit all authority granted to other organisations strictly to that legitimately delegated to them, even if their community records become corrupted.

Community Policies & Membership Policy

```
<xs:element name="cbpms:CommunityPolicySet" type="cbpms:CommunityPolicySetType" />
<xs:element name="cbpms:MembershipPolicy" type="cbpms:PolicySpecType"/>

<xs:complexType name="CommunityPolicySetType">
  <xs:sequence>
```



```

        <xs:element ref="cbpms:CommunityPolicy" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>

    <xs:element name="cbpms:CommunityPolicy" type="cbpms:CommunityPolicyType" />
    <xs:complexType name="CommunityPolicyType">
      <xs:sequence>
        <xs:element ref="cbpms:Title" minOccurs="1"/>
        <xs:element ref="cbpms:Description" minOccurs="0"/>
        <xs:element ref="cbpms:PolicyScope" />
        <xs:element name="cbpms:PolicySpec" type="cbpms:PolicySpecType"/>
      </xs:sequence>
    </xs:complexType>

    <xs:complexType name="PolicySpecType">
      <xs:sequence>
        <xs:element name="PolicyIDRef" type="xs:anyURI"/>
      </xs:sequence>
      <xs:attribute name="PolicySpecLanguage" type="xs:string" use="required"/>
    </xs:complexType>

```

The Community Policy Set is a set of all the policies defined for the community. Each policy consists of a title, a description, a policy scope (encoded as a resource authority) and a policy specification. A policy specification consists of the URI of the policy specification, with an attribute identifying the policy specification language in which the policy is specified.

Resource Authorities

```

<xs:element name="cbpms:ResourceAuthority" type="cbpms:ResourceAuthorityType" />
<xs:element name="cbpms:PolicyScope" type="cbpms:ResourceAuthorityType" />
<xs:complexType name="ResourceAuthorityType">
  <xs:sequence>
    <xs:element ref="cbpms:ResourceRef" minOccurs="1" />
    <xs:element ref="cbpms:TargetRef" minOccurs="1" />
    <xs:element ref="cbpms:ActionRef" minOccurs="1" />
  </xs:sequence>
</xs:complexType>

<xs:element name="cbpms:ResourceRef" type="xs:anyURI" />
<xs:element name="cbpms:TargetRef" type="xs:string" />
<xs:element name="cbpms:ActionRef" type="xs:string" />

```

The Resource authority element consists of 3 elements – a reference to a resource which is a URI, and a target reference and an action reference, both of which are free-form strings, the contents of which are interpreted according to the resource type as identified by the RAIS, as described below.

Full Community Example

Having defined the XML representation of the community structure, an example community encoded according to this schema is shown below:

```

<cbpms:Community communityId="http://www.example.org/cbpms/c123">
  <cbpms:title>An Example Community</cbpms:title>

```



```

<cbpms:Description>The description of this community function</cbpms:Description>
<cbpms:Parent>http://www.example.org/cbpms/c123</cbpms:Parent>
<cbpms:Children>
  <cbpms:CommunityRelation>
    <cbpms:Description>The function of the child</cbpms:Description>
    <cbpms:CommunityIdRef>http://www.example.org/cbpms/c125
    </cbpms:CommunityIdRef>
  </cbpms:CommunityRelation>
  <cbpms:CommunityRelation>
    <cbpms:CommunityIdRef>http://www.example.org/cbpms/c126
    </cbpms:CommunityIdRef>
  </cbpms:CommunityRelation>
</cbpms:Children>
<Federations>
  <cbpms:CommunityRelation>
    <cbpms:Description>Why a member of the federation</cbpms:Description>
    <cbpms:CommunityIdRef>http://www.example.org/cbpms/c999
    </cbpms:CommunityIdRef>
</Federations>
<cbpms:GrantedAuthoritySet>
  <cbpms:DelegatedAuthority>
    <cbpms:Title>Read Secret File</cbpms:Title>
    <cbpms:Description>Access to the secret file</cbpms:Description>
    <cbpms:CommunityIdRef>http://www.example.org/cbpms/c126
    </cbpms:CommunityIdRef>
    <cbpms:ResourceAuthority>
    </cbpms:ResourceAuthority>
  </cbpms:DelegatedAuthority>
</cbpms:GrantedAuthoritySet>
<cbpms:DelegatedAuthoritySet>
  <cbpms:DelegatedAuthority>
    <cbpms:Title>Read Secret File</cbpms:Title>
    <cbpms:Description>Access to the secret file</cbpms:Description>
    <cbpms:CommunityIdRef>http://www.example.org/cbpms/c126
    </cbpms:CommunityIdRef>
    <cbpms:ResourceAuthority>
    </cbpms:ResourceAuthority>
  </cbpms:DelegatedAuthority>
</cbpms:DelegatedAuthoritySet>
<cbpms:CommunityPolicySet>
</cbpms:CommunityPolicySet>
<cbpms:MembershipPolicy>
</cbpms:MembershipPolicy>
</cbpms:Community>

```

XACML Integration

Although the above specification fully describes the community structure, the policy search algorithm can be further optimised and the amount of conflict resolution meta-data that can be access by maximised by adjusting the policy search algorithm slightly in order to take advantage of the policy specification language used. Thus, for example, in order to integrate XACML policies into the system with the greatest degree of flexibility, the URI specified in the PolicyIDRef element can refer to either a XACML PolicySet element or an XACML Policy element, or a reference to either of these types. This will allow multiple related policies to be attached to a single policy scope and grouped together in sets. Since XACML Policy Sets can contain other elements of the same type, sets of policy specifications can be embedded inside each other and fine-grained conflict resolution algorithms can be specified within and between an arbitrary number of sets of policies. Thus, when the *getRelevantPolicies(ResourceAuthority)*

function is invoked, either an XACML PolicySet or an XACML Policy is returned for evaluation.

Going further than this, the XACML specification can be extended to incorporate an extra element into both the XACML Policy element and the XACML Policy Set element which will represent the resource authority of each individual Policy Set and policy rule. XACML policies contain a Target element which operates as a scoping filter in much the same way that the resource authority element scopes a CBPMS policy specification. Furthermore, the hierarchical filtering mechanism that applies to XACML Target elements can also be applied to the Resource Authorities. This means that, if there is a resource authority specified for an embedded Policy or a Policy Set, it is interpreted as giving a greater precision to the scope of the resource authority of a Policy or Policy Set within which it is embedded. By utilising this hierarchical filtering mechanism on XACML policy specifications, many of the policies within a policy specification consisting of one or more policy sets can be pre-filtered before the policy specification is sent to the PRS for evaluation. The policy specification can be effectively trimmed so that only those XACML policies which are within the specific scope of the policy decision request are ever evaluated.

In order to implement this there are a number of choices – simply extending the XACML definition being of the most general purpose use. However, for the purpose of this CBPMS implementation the definition of the `xacml:Target` element was overwritten to turn it into a resource authority type. Although this limits the filtering that the XACML PRS engine can do internally, for the purposes of this work, maximising the efficiency of XACML evaluations internally is not a priority and the very same result can be achieved by specifying suitable conditions for the policy specifications.

```
<xs:element name="xacml:Target" type="cbpms:ResourceAuthorityType" />
```

Finally, one element of the schema has not been described: `ChildCombiningAlgId`. Rather than defining a specification for this anew, the specification and semantics of XACML's Policy Combining Algorithm element is used to define this element since the task of combining the results of multiple policy decisions in a policy set is identical to the task of combining the results of multiple policy decisions from different child communities.

Appendix A8. Sessions and Credentials

Session support, user authentication and credential testing are requirements for many applications which might consume policy decisions, particularly those that consume access control decisions. Although the CBPMS does not concern itself directly with user authentication, supporting access control rules based on authentication is an important requirement. Similarly, the use of sessions, while not required for the service is useful in terms of limiting the size and complexity of context messages transmitted across the network. In order to provide this support a Session and Credential Handling context mapping class was implemented, overriding the decision function of the base class. The basic outline of a Java implementation is presented below.

```
public class SCHContextMapper extends ContextMapper {
    public String decision(ResourceAuthority ra, String com, Map context){
        String cert = context.get("authentication-certificate");
        String uid = context.get("user-id");
        if(cert != null && uid != null) {
            If(SecurityServer.check(uid, cert)) {
                If(SessionServer.sessionExists(uid){
                    SessionServer.updateSession(uid, context);
                }
                else{
                    SessionServer.createSession(uid, context);
                }
                return CPSService.decision(ra, com, context);
            }
        }
        return null; //error indicator
    }
}
```

This context type requires the policy consumer to populate the context with two required values. An authentication certificate – which is a token that has been issued to the user by an authentication server, and a user-id which allows the CMDS to identify the session that will be used in this context mapping. The Session Server class operates according to a very simple algorithm. If a session already exists for the user, any name-value pairs contained in the input context beyond the basic requirements are interpreted as updates to the user context which will be added to the user session as well as being retained in the output context. This context mapping class allows policy decision consumers to optimise their communications with the CBPMS by selectively transmitting information related to the context of user actions to the system, rather than having to transmit a large amount of data every time.

By incorporating a check to the authentication server into this class, basic security checks are handled by the service –such checks can also be specified as policy rules, but for certain applications, such a check is ubiquitous and it is more efficient to encode it in such a manner and minimise the amount of policies that have to be evaluated.

Semantic Language Context Mapping

A second type of context mapping class that can be implemented uses a semantic web encoding of the policy consumer and the internal context in order to map the context to the request. This class overrides the load method and interprets the URI as the URI of an OWL file containing an ontology which maps entities in the policy consumer's context to entities in the CBPMS internal context. The decision method uses this ontological context based mapping to map the passed context variables into the CBPMS context. An extremely limited set of ontologies are used in the implementation as the problems of run-time context mapping are manifold and beyond the scope of this work. Applying semantic web technologies to context mapping allows the support of sophisticated relationships between different contexts using open standards.

Discover function

In addition to providing the proxy functionality which maps the consumer's context to the CBPMS context, the CMDS also provides a second function, the discover function, which according to the architecture, allows the PRS to request values for context variables that it requires in order to evaluate policies. This function is implemented in a way that is identical to that of the function above. The class definition is expanded to include a discover function and classes can override it. Each discovery request contains a reference to a contextMapID and each class can interpret this as is required.

```
public abstract class PolicyEvaluator{
    public abstract String evaluatePolicy(String PolicySpec, Map context)
    public abstract boolean evaluateResult(String result);
    public abstract boolean isSemanticallyEqual(String result1, String result2)
    public abstract String notApplicable();
}
```

The implementation of a class to represent an XACML engine is outlined below. An “indeterminate” result should be treated as semantically equal to a “Deny” result. In accordance with the XACML data-flow model, the XACML PDP is notified of the request before evaluating it. This will cause the engine to return a set of attribute queries to the CBPMS context – encoded as a map of key – empty-value pairs. The

discover function of the CMDS is invoked to populate the values of these attributes. Only after this context discovery can the evaluate method of the engine be invoked.

```
public class XACMLPolicyEvaluator extends PolicyEvaluator{
    public String evaluatePolicy(String PolicySpec, Map context){
        XACML engine = new XACML();
        engine.loadPolicy(PolicySpec);
        XACMLRequest req = new XACMLRequest();
        req.populate(context);
        Map attributeQueries = engine.requestNotify();
        CMDServer.discover(attributeQueries);
        return engine.evaluate(attributeQueries);
    }
    public abstract boolean isApplicable(String result){
        if(result.equals("NotApplicable")){
            return false;
        }
        return true;
    }
    public abstract boolean isSemanticallyEqual(String result1, String result2){
        if(result1.equals(result2) ||
            (result1.equals("Deny") || result1.equals("Indeterminate"))
            && (result2.equals("Deny") || result2.equals("Indeterminate"))){
            return true;
        }
        return false;
    }
    public abstract String notApplicable() {
        return "NotApplicable";
    }
}
```

Due to the fact that this work has little to do with exploring the detailed workings of policy evaluation engines, the PRS implementation is limited to two simple evaluation engines. In addition to the XACML policy evaluation engine, an extremely simple obligation engine is provided. This obligation engine supports arbitrary PHP commands allowing great flexibility in extending its functionality, but providing virtually no possibility of policy analysis. In addition to allowing define obligation policies to be defined that will be triggered by specific policy decision requests, this engine is used to take advantage of XACML 2.0's support for obligation policies that can be attached to access control policies in the specification language.

This is implemented by encoding a special type of XACML obligation policy which is interpreted by the PRS controller to be an obligation to be evaluated by the obligation policy engine. This behaviour is generalised to allow any policy decision result to be interpreted as a policy that is to be interpreted by a different policy evaluation engine.

If a call to a policy evaluation class's *evaluate* function returns a result which is interpreted by the controller as being addressed to a policy evaluation engine, then the result from that evaluation is returned from the service's evaluate function. The algorithm that the controller follows is shown below:

```
public String evaluatePolicy(String policyLang, String policySpec, Map context)
{
    if(factory.hasClass(policyLang)){
        PolicyEvaluator pe = factory.create(policyLang);
        String result = pe.evaluate(policySpec, context);
        if(isOnwardPolicy(result)){
            onwardLang = getPolicyLang(result);
            onwardSpec = getPolicySpec(result);
            return evaluatePolicy(onwardLang, onwardSpec, context);
        }
        return result;
    }
    return null; //error indicator
}
```

The controller merely needs to implement a method for detecting when a result is an onward policy evaluation request and methods for parsing a policy language and policy specification from this result. It recursively calls itself to evaluate the returned policies until it receives a policy decision result which is not an onward policy request and returns this value.

By specifying such obligation requests within XACML policies, logging can be enforced, notification messages and any other type of functionalities that is required can be triggered when the policy is evaluated with the specified result. This deviates from the XACML model where it is the PEP (or policy decision consumer in the terminology of this work) that executes obligation policies. However, this does not break the XACML model since it is only those obligations which are explicitly defined to be enacted by the PRS that are automatically evaluated, all other obligations are passed back as policy decision results to the CPSS and from there back to the policy consumer. By incorporating this special type of obligation policy into this implementation, a broader range of applications can make use of this feature since their obligations can be specified without reference to the capabilities of the policy consumer.

Appendix A9: Oscailt Evaluation

In order to test the oscailt configuration, administrators were asked to assume a number of different identities (i.e. community memberships) and attempt to carry out the following tasks with the content management system, to verify that it correctly captured the community's operations:

- Comment on a story.
- Add a new story
- Add a picture to a story (40K, 400K)
- Hide a story
- View hidden comment
- View hidden story
- Move story to front page.
- View administration page
- Change Policy by adding a constraint

Once policy specification errors were eliminated, the CBPMS configuration was confirmed to have implemented the required policies in each area, with respect to the situations tested.

Interview Questions

- How does the community model correspond to your view of the organisation?
- How does this compare with the previous role-based organisation of users?
- What features of this model do you prefer?
- Which do you dislike?
- Would you be happy to use this system to manage your website?
- If not, what about this system would put you off.
- Describe the organisational steps necessary to give the daleks a new permission or to remove a permission from the techies. In broad outlines, how would you achieve this with the CBPMS?

The data-sheets from this experiment are available as a research white paper from KDEG, Trinity College Dublin.