

8-31-2011

A framework for usage management

Pramod Jamkhedkar

Follow this and additional works at: https://digitalrepository.unm.edu/ece_etds

Recommended Citation

Jamkhedkar, Pramod. "A framework for usage management." (2011). https://digitalrepository.unm.edu/ece_etds/123

This Dissertation is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Electrical and Computer Engineering ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

Pramod A. Jamkhedkar

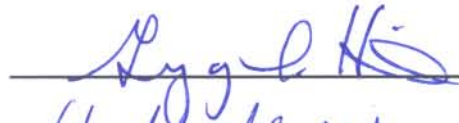
Candidate

Electrical and Computer Engineering

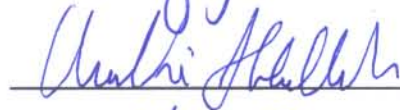
Department

This dissertation is approved, and it is acceptable in quality and form for publication:

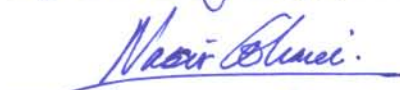
Approved by the Dissertation Committee:



Dr. Gregory L. Heileman, Chairperson



Dr. Chaouki T. Abdallah



Dr. Nasir Ghani



Dr. Jedidiah R. Crandall

A Framework for Usage Management

by

Pramod Jamkhedkar

B.E., Computer Engineering, University of Mumbai, 2002

M.S., Computer Science, University of New Mexico, 2005

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Engineering

The University of New Mexico

Albuquerque, New Mexico

July, 2011

©2011, Pramod Jamkhedkar

Dedication

To my parents, for their continued support and inspiration.

Acknowledgments

First and foremost I would like to thank my advisor, Prof. Gregory Heileman, for his unwavering support, continued motivation, guidance and inspiration. Working and collaborating with him over the course of my doctoral studies has given me an opportunity learn from not only his professional qualities, but also his personal character.

I would also like to thank Prof. Chaouki Abdallah, Prof. Nasir Ghani for their support and guidance during my doctoral studies. I would also like to thank them, and Prof. Jedidiah Crandall for agreeing to be on the committee and reviewing this work. Additionally, I also thank professors Sudharman Jayaweera and Deepak Kapoor for helping me in the areas of their expertise.

Finally, I would like to thank my family and friends whose support and motivation has helped me through the difficult times.

A Framework for Usage Management

by

Pramod Jamkhedkar

ABSTRACT OF DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Engineering

The University of New Mexico

Albuquerque, New Mexico

July, 2011

A Framework for Usage Management

by

Pramod Jamkhedkar

B.E., Computer Engineering, University of Mumbai, 2002

M.S., Computer Science, University of New Mexico, 2005

Ph.D., Engineering, University of New Mexico, 2011

Abstract

This thesis proposes a formal framework for usage management in distributed systems. The principles of system design are applied in order to standardize certain features of the framework, such as the operational semantics, and leave free of standards areas that necessitate choice and innovation. The framework enables use of multiple policy languages, and dynamic interpretation of usage policies in different computing environments. In addition, the framework provides formal semantics to reason about interoperability of policies with respect to computing environments. The use of this framework in different usage management scenarios is demonstrated including multi-level security, cloud computing and digital rights management (DRM) systems. Furthermore, DRM is cast in a setting that allows the modeling of a number of current approaches within a game theoretic setting. Current strategies that attempt to influence the outcome of such games are analyzed, and a new type of architectural infrastructure that makes novel use of a trust authority is considered in order to create a suitable environment for constructing DRM games that may prove useful in the future.

Contents

List of Figures	xi
List of Tables	xiv
Glossary	xv
1 Introduction	1
2 Background	7
2.1 Access Control	7
2.2 Digital Rights Management	8
2.3 Scope of Usage Management	9
2.4 Challenges in Usage Management	11
3 Usage Management Framework	19
3.1 Meta-model	20
3.2 Objects and Interfaces	24

Contents

3.2.1	Context Object	24
3.2.2	Policy Object	25
3.3	Deployment and Operation	27
3.3.1	Setup Stage	28
3.3.2	Working Stage	29
3.4	Framework Analysis	32
3.4.1	Interoperability Semantics	32
3.4.2	Context and Policy Hierarchies	36
3.4.3	Dynamic Interpretation	39
3.5	Mathematical Model	42
3.5.1	Model Description	43
3.5.2	Dynamic Interpretation	53
3.5.3	Operational Semantics	54
3.5.4	Interoperability Semantics	56
4	DRM Game	58
4.1	The Baseline Game	61
4.1.1	Perfect DRM	63
4.1.2	Imperfect DRM	65
4.2	Important Subgames	68
4.3	Repeated Games	74

Contents

4.3.1	Content Spreading	76
4.3.2	Customer Influence	78
4.4	Conclusions	79
5	Applications	81
5.1	Multi-level Security	81
5.1.1	MLS Overview	82
5.1.2	Context Modeling	82
5.1.3	Policy Specification	84
5.1.4	System Operation	86
5.2	Cloud Computing	87
5.2.1	Usage Management Requirements in Clouds	89
5.2.2	Overlaying Usage Management Framework on Cloud Infrastructure	92
6	Conclusions	99
7	Future Work	101
	References	102

List of Figures

2.1	The primary elements in a usage management system.	10
2.2	The challenge of managing multiple policy languages in a distributed system.	12
2.3	Policy management in closed systems	13
2.4	The approach where content moves across different computing environments, and each computing environment having a policy interpreter. . . .	14
3.1	The Metamodel for the Usage Management Framework	21
3.2	A context object	25
3.3	A policy object	26
3.4	Setup stage for usage management.	28
3.5	Working stage for usage management.	30
3.6	Multiple policy objects used on a single client system	34
3.7	Single policy object used on multiple client systems.	35
3.8	A hierarchy of policy types	37

List of Figures

3.9	A hierarchy of context types	38
3.10	Dynamic interpretation of activities in terms of identifiable actions in the computing domain.. . . .	40
3.11	Operational semantics of the calculus.	55
4.1	(a) An abstract model of the baseline DRM game in extensive form, along with two examples of its strategic form, (b) one in which $\pi_c(dl) \geq \pi_c(p, tp)$, and (c) the other in which $\pi_c(p, tp) \geq \pi_c(dl)$	62
4.2	(a) A general model in extensive form of the post-content-acquisition subgame that flows from the outcomes of the game shown in Figure 4.1a, where node i can assume the values 1–3. (b) An example of this subgame in strategic form that models RIAA efforts, and (c) a model that incorporates rewards.	69
4.3	The complete DRM constituent game that consists of the baseline content acquisition subgame of Figure 4.1, followed by the post content acquisition subgames of Figure 4.2.	75
5.1	A multi-level security environment.	82
5.2	A multi-level security mashup policy server	87
5.3	Usage management in existing cloud environment.	90
5.4	Usage management in cloud.	91
5.5	A distributed cloud infrastructure consisting of multiple cloud services and virtualization.	92
5.6	Persistence of policies across data aggregations.	93

List of Figures

5.7	Cloud usage management operating with SLA QoS monitoring.	94
5.8	Setup phase usage management that include context and license generation.	95
5.9	Operation of usage management in cloud environment.	97

List of Tables

3.1	An example structure of context.	44
3.2	An example structure of context instance.	46
3.3	An example description of functions.	46
5.1	Context for multi-level security.	83
5.2	Resource table	84
5.3	Resource-Policy table	84
5.4	Policy table for mashups	86

Glossary

\mathcal{F}_k	Set of functions used by an object k
\mathcal{I}_k	Set of functions provided by the interface of object k
pol	Policy object
con, C	Context object
umm	Usage management mechanism
$pol \bowtie^c umm$	Policy pol is context compatible with usage management mechanism umm
$pol \bowtie^p umm$	Usage management mechanism umm is policy compatible with policy pol
$pol \bowtie umm$	Policy pol is interoperable with usage management mechanism umm
$o_2 \geq o_1$	Object o_2 inherits object o_1
cr	Constraints
E	Environment entity
S	Subject entity
R	Resource entity

Glossary

iE, iS and iR	Instances of entities E, S and R
p	Property
D_p	Domain set of a property p
F_p	Set of functions provided by property p
f_p	Function provided by property p
$iC \models cr_C$	Instance of context C satisfied constraint cr expressed over C
RA	Set of restricted activities
f	Function provided by property p
ε	Policy usage expression over RA
av	Activity
rv	Restricted activity
Acv	Set of activities
act	Action
Act	Set of Actions
iav	Activity instance
$iav \propto rv$	Activity instance iav satisfied restricted activity rv
\mathcal{P}	Set of permissions
O	Set of obligations

Chapter 1

Introduction

Usage management is management of usage of resources (and data) across and within computing environments. In the coming years, data and resources will be increasingly used in innovative ways in open, distributed environments such as the Internet.

In most policy management systems deployed, use of resources (and data) has generally been restricted to closed computing environments that are managed by a single entity. However, with the explosion of Web 2.0 applications on the Internet, these assumptions no longer hold. Data is now increasingly used across loosely coupled distributed systems, where data owners and data users are a part of separate computing environments. The separation of resource owner and the owner of the computing environments within which resources are consumed is more pronounced. Moreover, computing environments within which resources are used are no longer fixed, and resources often move across multiple computing environments. This has led to a tussle, where resource owners demand that they hold the right to express the resource usage policy, rather than the owners of the computing environments.

This trend is increasingly observed in usage management of commercial content, medical information and financial data, military data, social data, etc. Piracy of digital con-

Chapter 1. Introduction

tent, such as movies and music, has been one of the most difficult problems to solve in the recent years. For technological and commercial reasons, the computing community has been unable to control the massive amount of piracy of commercial entertainment content. Digital rights management (DRM) systems have always struggled with issues including interoperability and enforcement of rights policies [35, 51]. The DRM problem is a business-to-consumer, where industries owning content have been worried about violation of copyright by individual consumers.

The consumer-to-business side of this problem has been equally challenging in the recent years. Private individuals and institutions employing computing services are getting increasingly worried about the manner in which their data is handled by service providers. For example, the use of private information by online mail and social networking services service providers has been a matter of serious concern. Similarly, online personal health resources management systems such as Google Health and Microsoft Health Vault, has consumers worried about the privacy and use of their personal health information.

More recently, cloud computing solutions have gained increasing attention in their ability to deliver computing services as the fifth utility. In cloud computing, applications, systems software, and hardware are offered as utility services to consumers over the Internet. In cloud computing, cloud users' data reside in the cloud for a finite amount of time, that these data are handled by multiple cloud services, and that data fractions may be stored, processed, transformed, mashed, and routed across a geographically distributed cloud infrastructure. These activities occur “behind-the-scenes”, within the cloud, while giving cloud users an impression of a single virtual machine. In current cloud implementations, cloud users have very little control over the manner in which data are handled by cloud providers, once they are pushed into the cloud. As consumers start aggressively using cloud services, this limitation will become a matter of serious concern.

In all these situations, it is necessary that owners of data are able to express the terms of usage of their data once they pass on their data to other users over the Internet. In order to

Chapter 1. Introduction

address this problem, it is necessary to identify the common set of characteristics that are exhibited by all the above mentioned scenarios. The primary distinguishing characteristic is that all these scenarios deal with “usage” of data rather than just access to data. In many of these cases, data access is often implied, and the focus is on how a given user uses the data after it has been handed over. The second characteristic is that data are used not within the confines of a single system managed by a single entity, but rather it is used within a distributed system-of-systems, where each sub-system may be independently managed by a different entity. The third characteristic is that data owners has no *a priori* knowledge or control over the system or systems within which their data may be used.

A solution to address these problems requires an open framework that enables three capabilities, namely, policy expression, interpretation and enforcement. First, the framework must provide a mechanism that allows expression of different types of usage policies over resources. Such a mechanism must allow users to express various usage semantics including permissions, obligations, constraints, partial dependencies, authorizations, to name a few. Second, since the data in usage management travels across a distributed system, it must be possible to dynamically interpret a policy across different systems whose exact characteristics may not be known *a priori*. Finally, it must provide mechanisms to ensure that the data are used in accordance with the policy on the client system. The reason the framework needs to be open is because usage management occurs over a distributed open system such as the Internet, and it is therefore necessary to accommodate different types of existing systems and policy languages.

Research in this area has largely followed independent paths in the areas of access control frameworks [12, 13], and digital rights management systems [46]. There have been proposals to combine these two areas into a more comprehensive framework for usage control [59]. Most of the current research in this area has been directed towards developing more expressive language by using either a different type of mathematical logic, or a formalism with a greater reasoning capability [10, 11, 23, 36, 37, 64, 75]. Such

Chapter 1. Introduction

advancements, even though useful in closed systems cannot be effectively deployed in a distributed system without a large overhead of deploying policy interpreters across the different systems. Similarly, there are frameworks for policy management in distributed systems, however they still require a pre-agreement on the characteristics of different constituent systems and provide no means for addressing the issue of interoperability, dynamic interpretation and accommodation for multiple policy languages [28].

To address the problem of interoperability two approaches have been followed, namely, translation mechanisms and standardization. The solutions that have tried to address interoperability by means of translations have often proposed translation mechanisms that translate a given policy from one language to another [38, 62, 69]. Such translations are infeasible, and difficult to carry out for most of the policy languages [51, 67]. Other approaches have led to the development of complex policy specification languages that have tried to establish themselves as the universal standard [1, 2, 73, 76]. Such an approach requires standardization of the complete policy language, which stifles innovation and flexibility [38, 46, 47, 49]. The other disadvantages of adoption of a universal, highly expressive policy language is that such languages are bloated and extremely difficult to formalize and reason over [36, 37].

Enforcement of usage policies across a distributed framework has always been a big challenge. This problem has mostly been approached by means of trusted computing platforms that have tried to control user actions in order to ensure that usage is in accordance to the policy. However, most of these approaches have either failed because of security breaches, or the users have rejected these solutions as they have prevented users from ease of use and fair use. Other successful solutions to enforcement have either been in closed systems where system administrators have complete control over the hardware [6, 21], or in case of solutions such as Apple's Fairplay technology, where content owners own the complete vertical chain from application to hardware. However, even such solutions have faced resistance from users due to lack of interoperability.

Chapter 1. Introduction

In this thesis, an open interoperable framework for usage management in distributed systems is proposed, along with game theoretic approaches that incentivize users to use content in accordance with usage policies. In the proposed approach, the principles of system design are applied to develop an open framework for usage management that supports interoperability. These principles have been used by researchers in Internet design, to achieve a balance between interoperability and open, flexible architectures [7, 17, 25]. In order to achieve this goal, certain features of the framework such as the operational semantics are standardized, and features that necessitate choice and innovation are left free of standards. Similarly with respect to enforcement, existing scenarios for DRM are modeled in game theoretic setting to reason about possible incentives that can be created for users to use content respecting copyright.

The framework exhibits a set of features that address the problems encountered with usage management of resources in distributed systems. First, the operational semantics of the framework are independent of syntax and semantics of any particular type of policy languages or logics that are currently used, or may be used in the future for policy specification. The framework provides a scaffolding upon which different policy languages or logics can be used to express usage policies. This is achieved by creating a design space where policy languages can be used or introduced to operate within the framework without affecting the operational semantics of the framework. Second, the framework allows policies to be interpreted dynamically in different systems. The framework tries to achieve a maximum degree of independence between the process of policy expression and interpretation of policies on client systems. This allows, on the one hand, for policies to be expressed with minimal knowledge of client systems, and on the other hand it allows client systems to interpret and enforce usage rules with minimal knowledge about policies. Finally, the framework provides formal semantics for interoperability to reason whether or not it is possible to interpret a usage policy within a client system. With these features, the framework provides an open actionable platform upon which different types of independent systems and policy languages can interoperate with minimal overhead.

Chapter 1. Introduction

To address the problem of policy enforcement in DRM content usage, DRM is cast in a setting that allows to model a number of current approaches as games. The DRM game is partitioned into two subgames, one that considers the game associated with content acquisition, and a second that considers how a consumer uses the content, along with a vendors response to this usage. Examples are provided in order to demonstrate how these subgames correspond to real situations associated with content industries, and the conditions under which Nash equilibria will exist. These subgames form the primary stage of a repeated game that models a number of important long-term interactions between consumers and vendors. Current strategies that attempt to influence the outcome of the repeated game are analyzed, and a new type of architectural infrastructure is considered that makes novel use of a trust authority in order to create a suitable environment for constructing DRM games that may prove useful in the future.

The rest of the thesis structured as follows. Chapter 2 provides the background and the motivation for the problem of usage management in distributed systems. This chapter introduces the term *usage management* and challenges posed by usage management systems, followed by the limitations of existing approaches to address these issues. In Chapter 3, usage management framework described along with its constituent elements, the operation of the framework, along with formal semantics for interoperability. In Chapter 4, a game theoretic approach to the problem of enforcement in to a class of usage management systems is provided, in particular DRM systems for commercial content. Chapter 5 provides a set examples on how the framework would play a major role to enable usage management in the areas of multi-level security and cloud computing. In Chapter 6 and 7 useful conclusions and future work necessary to make this framework more applicable and acceptable in research and industry are discussed.

Chapter 2

Background

In this section the historical development of usage management is explained, followed by the scope and constituent elements of usage management, tracing its origins to access control and digital rights management . Following this, the challenges that result from the evolving nature in which information is being used across systems are discussed. This is followed by a discussion on how existing approaches are used in access control and DRM, and how they have been unable to address the challenges in usage management.

2.1 Access Control

Access control mechanisms are systems that manage controlled access to resources. The central idea behind access control is that access to a resource is granted depending upon subject attributes, resource attributes, and system attributes. The central component of any access control system, is an access control language (ACL) that is used to express rules for granting access to different resources in the system.

Access control policies can be categorized into two types, namely, discretionary access

Chapter 2. Background

control (DAC) and mandatory access control (MAC) [41]. DAC policies are policies that are specified by the owner of the resource, based on the users' attributes. On the other hand MAC policies are made by a central authority and apply to the whole system. A number of access control models have been developed that allow different types of access control in these two modes. The most successful ones being the role-based access control model (RBAC), and the Bell and LaPadula model [12, 13]. The focus of access control models is to capture the different types of relationships between and among a set of resources and a set of users, and express access rules based on those relationships.

The use of an access control language or an access control framework within a system includes a significant overhead. This typically includes overlaying the framework on the system where it is to be deployed. It also includes a pre-agreement on the characteristics and ontologies used to model the system. Access control mechanisms are generally tightly coupled with the system in which they are deployed. Access control systems focus primarily on the rules for granting access to different users within the system, and generally do not deal with the problem of usage.

2.2 Digital Rights Management

Digital rights management (DRM) consists of mechanisms that manage controlled usage of digital resources. The central idea behind DRM is that usage rules for a given resource are specified for a particular user or group of users, and the use of the resource is subsequently managed for a finite period of time. Usage rules generally include a set of permissions and obligations, along with rules specifying how the permissions may be exercised over a period of time and under what circumstances. DRM also includes mechanisms such as trusted computing that ensure the enforcement of rights on the user machines [6]. The most well known, but unsuccessful, approaches to address this problem are IBM's Cryptolope and Microsoft's Palladium technologies [22, 26, 52]. In Chapter 4, a game theoretic

Chapter 2. Background

approach is proposed, in which instead of enforcement, users are incentivized to use content in accordance with usage policies [39, 78]. The central component of DRM systems is a rights expression language (REL) that is used to express usage rules (or rights) in the form of a license that is generated by a resource owner for a user or a group of users that will use a resource.

Some of the earliest attempts at DRM RELs date back to 1980's, and involve the development a formal language for legal discourse [18, 55, 54]. At present, creative commons, along with two XML-based RELs, namely, XrML and ODRL, are most commonly used [77, 44]. Both XrML and ODRL have formed alliances with major players in the industry and standards bodies. Semantics of these XML-based languages are informal. A number of approaches using various types of formalisms and approaches have been used to develop formal RELs. Gunter et al. [34] and Pucella et al. [64] have used trace-based semantics to develop formal RELs. Other formalisms such as first-order logic and CafeOBJ have been used to develop RELs [10, 23, 75]. Many researchers have attempted to provide formal semantics for existing XML based RELs [36, 66, 37]. It is however a general agreement that popular XML-based RELs are difficult to formalize in their entirety [36, 37, 49].

More recently, researchers have tried to expand the concept of DRM to propose concepts including usage rights management and usage control. The idea of usage rights management is developed along the lines of making users aware of how a resource is supposed to be used [42]. A more formal framework, $UCON_{ABC}$ encompassing usage rules and access control is proposed by Park et al. [59].

2.3 Scope of Usage Management

Before discussing scope and components of usage management, it is important to note the difference between ACLs and RELs. Even though the goals of these two types of

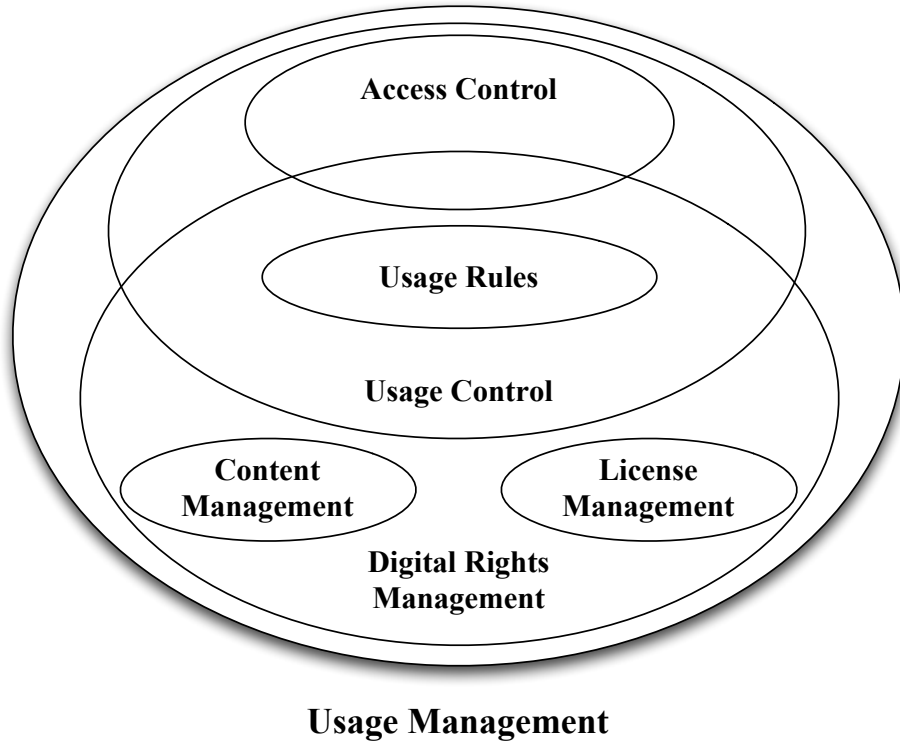


Figure 2.1: The primary elements in a usage management system.

languages overlap, the focus of research in ACLs and RELs is significantly different. ACLs focus on defining access rules in terms of relationships between set of resources and set of users. In DRM systems, once a user obtains a license for a resource, access to that resource is implicit, and what matters is how that resource is used from that point onwards. Therefore, RELs focus on defining different types of usage rules for a given user (or group of users) over a given resource (group of resources).

We introduce the concept of usage management that is built upon the term *usage control* introduced by Park et al. [59]. The usage control model, called $UCON_{ABC}$, is based on *Authorizations, obligations and Conditions* [59]. $UCON_{ABC}$ combines access control and permissions and obligations in a single model.

Chapter 2. Background

Usage management components, and their relation to one another is shown in Figure 2.1. Usage management is a combination of usage control and DRM. Usage control is a combination of access control and usage rules. Digital rights management includes content management, license management, specification of usage rules and a simplified subset of access control. Content and license management include processes that manage how content and license are bundled, encrypted and distributed or managed across multiple clients. These processes include encryption mechanisms, trust management, trusted computing platforms and other such management techniques. Many RELs, including XrML and ODRL, have tried to incorporate these functionalities. Thus, usage management is the collective set of processes and mechanisms that enables one to manage and control how data are used within a system. This encompasses policy specification languages, licensing mechanisms, policy expression and reasoning mechanisms, policy enforcement mechanisms, usage tracking, along with supporting authentication and encryption mechanisms [59, 48].

Next, the challenges in implementing usage management systems, and the need for an actionable framework for addressing the challenges are explained.

2.4 Challenges in Usage Management

Figure 2.2 shows a distributed system composed of multiple subsystems that are owned and managed by different entities. This figure shows the problem of usage management of resources in a distributed system. The cloud represents a distributed system composed of individual subsystems. Each subsystem is different, with different characteristics, owned and maintained with different entities. The content shown in the figure is accompanied by a policy. In a generalized version of this setup, the resource can be either data, information, content or a physical resource such as a computing resource, storing resource, or a networking resource. A policy may be attached to resource either directly by embedding

Chapter 2. Background

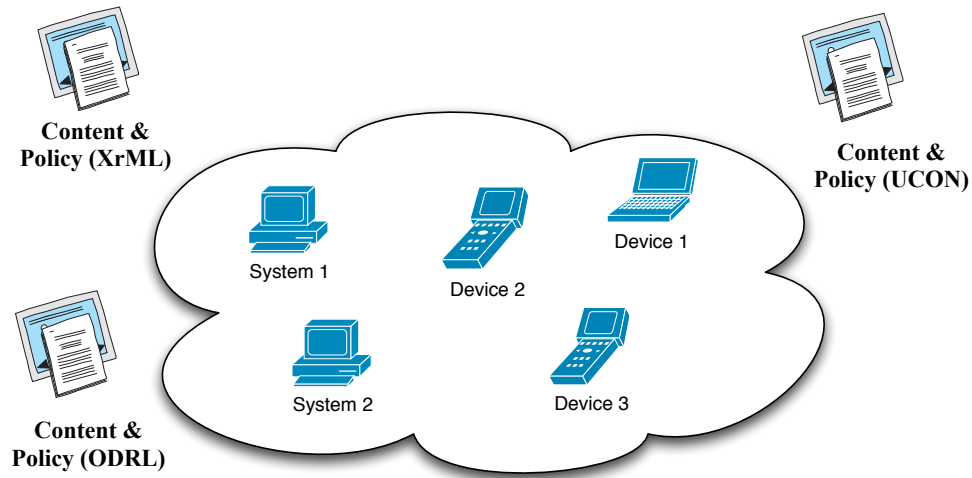


Figure 2.2: The challenge of managing multiple policy languages in a distributed system.

in the data, or they can be connected to each other indirectly via identification mechanisms such as the Handle system [71].

In classical access control closed system, policies are tightly coupled with the environment with respect to which they are expressed, interpreted and enforced. The working of such systems is shown in Figure 2.3. In this scenario policies are expressed with respect to a pre-agreed computing environment. Hence, every policy expressed can be correctly interpreted within the given computing environment by means of an interpreter, which is a software running within the computing environment.

Unlike classical access control systems, information is increasingly used across highly networked, distributed computing environments that are not a part of a single centrally managed system. In addition, digital information is increasingly used in innovative ways in which it is transformed, processed or merged with other information while being used across computing environments. One such example is the mashup process where information from two separate sources is merged to generate a new information source. This necessitates usage management policies to be tightly coupled with the resource, rather than the system. They are then interpreted and enforced as the resource moves across different

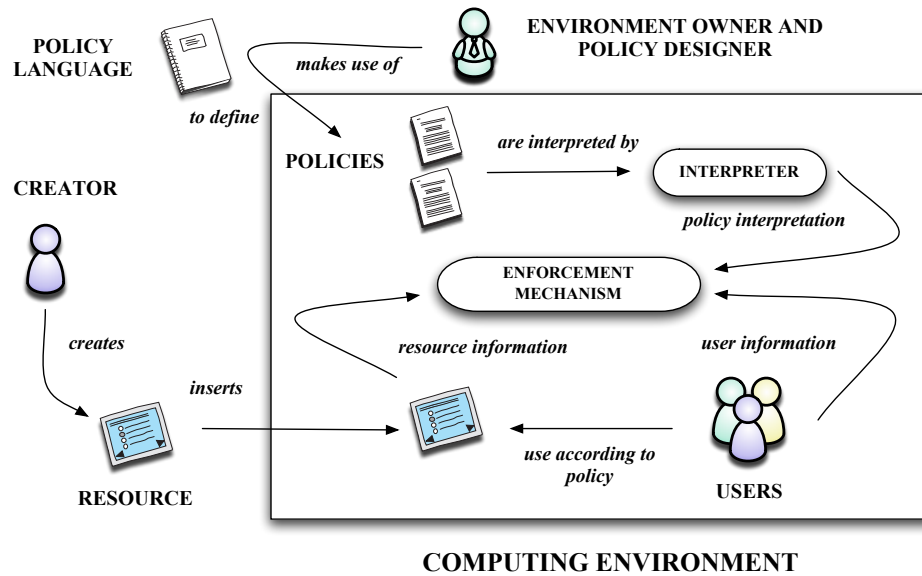


Figure 2.3: Policy management in closed systems

computing environments.

A logical approach to this problem is to build an interpreter for the policy language that runs on the computing environment (or the client), and enforces the policy on the client. This approach has been very successful for access control systems, because access control policies are tightly coupled with computing environments whose nature is known *a priori*. However, in usage management scenarios where resources move across environments that are not known *a priori*, such an approach leads to a number of problems. To address such a situation, each of the computing environments must incorporate an interpreter and enforcement mechanism that is custom built for different policy languages. This situation is shown in Figure 2.4. However, as shown in the figure, in the presence of multiple policy languages, interoperability becomes a major problem. When the data moves to a computing environment that runs an interpreter for a different policy language, the current policy can no longer be interpreted.

If a given computing platform intends to be a part of multiple information ecosystems,

Chapter 2. Background

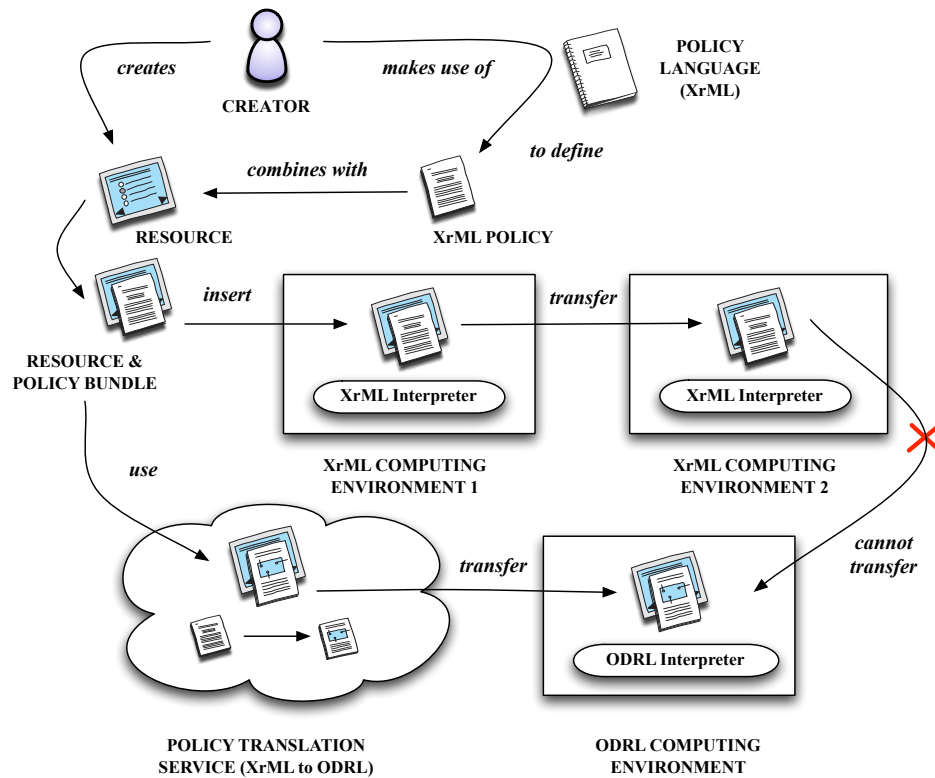


Figure 2.4: The approach where content moves across different computing environments, and each computing environment having a policy interpreter.

it must support the policy languages used by each of these ecosystems. This means that policy language interpreters for each of these policy languages need to be custom built for that particular computing platform. Furthermore, any advances or changes that are made in these policy languages will require corresponding updates in the interpreters used in all computing platforms.

The second disadvantage is that even though a given computing platform may support multiple information ecosystems, each of these information ecosystems still operate in complete isolation from one another. Since different ecosystems use different policy languages, licenses expressed within one ecosystem cannot be interpreted in another ecosystem. This prevents resources from moving freely across different ecosystems. To enable

Chapter 2. Background

free movement of data across different systems, it is necessary that interoperability among multiple policy languages is supported.

A number of recent papers have provided interesting solutions addressing interoperability at this level [3, 27, 51, 67, 69]. Major approaches have been used to address interoperability include translation mechanisms, architectural solutions and standardization.

Researchers have addressed the interoperability problem by developing translation mechanisms between policy languages [38, 62, 69]. However, such translations are feasible only for simple policy languages or a small subset of complex policy languages, and have been largely unsuccessful for practical purposes [51, 67].

The Coral and Marlin initiatives have provided architectural solutions to DRM interoperability [3, 27]. In the Coral approach, different licenses for different DRM systems are generated from a common token in accordance with a common ecosystem [27]. In the Marlin approach, licenses are expressed programmatically in the form of control objects to prevent dependence on any one particular REL [3]. Both Coral and Marlin architectures focus on the management of licenses across multiple systems.

In order to avoid the problem of interoperability, many policy languages have tried to establish themselves as the universal standard [1, 73, 77]. There are several problems with this approach. First, a policy language that is universal must be able to handle all types of policy expressions. Policy languages that have tried to achieve this have become highly complex and bulky [49]. A bulky, complex policy language creates problems of choice, formalism, and translation. The problem of choice means that a complex standard language forces every application to incorporate and support the complexity even though the requirements of the application are modest. Some languages, such as XrML, have tried to address this issue by means of extensions to a basic core language [8]. As the complexity of a language increases, it becomes more difficult to formalize the language. This leads to a situation where a language that is trying to establish itself as a universal standard

Chapter 2. Background

does not have any formal basis. Research has shown that it is not possible to formalize all the features of XrML, which is an XML-based language [36, 37]. Another problem that arises because of increased complexity of policy languages is the difficulty of translating the language to some other language. More complex a given language the more difficult it becomes to translate it to some other policy language [67]. The second problem that arises because of standardization of a language is its adaptability in different environments. Since the language is supposed to be a universal standard, to be used in all types of different computing environments, the language must maintain a certain level of abstraction. It is not possible for such languages to express policies that are too specific to a particular environment type. This problem is usually solved by means of providing extensions to the standard language that are customized for one particular type of environment [8, 76].

Due to lack of standards and interoperability, numerous formal logic-based rights expression languages have evolved. These languages use different types of mathematical logics to express and reason over various types of usage semantics. At the same time, XML-based languages such as XrML and ODRL have developed and continue to develop independent of these formal languages, and have not been able to incorporate their expression and reasoning power. Hence, unlike access control languages, the formal logic-based languages continue to remain outside the radar of industry alliances.

Hence, none of these languages are likely to become the *defacto* industry standard in the future. Different information ecosystems will continue to use different policy languages according to the policy expression requirements. Such a fragmented use of policy languages will remain the biggest obstacle to achieving usage management along with unhindered flow of information across highly distributed computing environments. Existence of multiple policy languages in such scenarios pose two problems, namely, difficulty supporting multiple languages and lack of interoperability.

The proposed framework provides an infrastructure that can accommodate existing languages and support interoperability among these languages. The framework, unlike

Chapter 2. Background

Coral and Marlin architectures, provides a formal calculus to reason about the relationship between a license, a computing environment, and interoperability between them. The framework incorporates concepts such as programmable policies and common ecosystems used by Coral [27] and Marlin [3] architectures respectively, and hierarchical composable policies used by Ponder policy language [28]. The design of the framework is based on the principles of *design for choice*, eloquently described by Clarke et al. with reference to “tussles” in cyberspace [25]. They explain the importance of identify the locations in the architecture where standards need to be introduced to enable interoperability, and locations where they should *not* be applied to enable innovation and differentiation. The design choices for the framework is based upon the following set of assumptions:

1. Information ecosystems will operate across highly networked, distributed, diverse computing environments managed by independent entities.
2. Resources will move across these computing environments as well as different information ecosystems. We define information ecosystem to be the set of rules and rights models that develop around particular content types (e.g., music, ebooks, software etc.)
3. Multiple information ecosystems and computing environments will continue to use different policy languages, depending on the types of rules and rights models required for expressing their respective policies.
4. No single policy language will be able address the policy expression requirements of different information ecosystems. Policy languages will continue to change and evolve using different logics to express various usage semantics.

These assumptions combined with the *design for choice* approach proposed by Clarke et al., form the basis for the design of the usage management framework proposed in this thesis.

Chapter 2. Background

In the next chapter the list of underlying principles, design and formal specification of the framework along with formal semantics for interoperability are explained.

Chapter 3

Usage Management Framework

This chapter provides a detailed description of the usage management framework. First, in Section 3.1, a meta-model that lays the foundation for the framework is introduced, followed by the principles that drive the design of the framework. The meta-model is based on two entities, namely policies and computing domains over which policies are defined, along with the interaction between the two. It serves the purpose of providing an abstraction for different types of policy languages that the framework intends to support.

Following this, Section 3.2 introduces the notion of executable policies and defines the core elements of the framework. These elements include policy objects that express usage rules, context objects that represent computing domains within which policies are interpreted. Section 3.3 explains the deployment and operation of the usage management framework. The process is divided into two stages, namely, the Setup stage and the Working stage. During the Setup stage, policy objects and context objects are generated from a common ontology representing the computing domain. The Working stage represents the interpretation and enforcement of a policy object within a client system.

Next, in Section 3.4, the features of the framework are analyzed, along with the advantages the framework offers in terms of flexibility, interoperability, and openness. Here,

three prominent features are introduced, namely, formal interoperability semantics, hierarchical organization of policies and contexts, and dynamic interpretation, followed with a discussion on the advantages offered by each of these features.

Finally, Section 3.5 provides a mathematical model that can be used to express policies and model computing domains. It must be noted that the proposed mathematical model is *not* a policy language. Rather, it is a scaffolding that provides a design space to introduce different types of mathematical logics to generate different types of policy languages that can be used within the framework. For the purpose of this thesis, a simple language for expressing permissions and obligations is introduced along with a demonstration of how such a language is used within the larger setting of the usage management framework.

3.1 Meta-model

The previous chapter explained the underlying assumptions of future information management systems, and motivated the need for an open interoperable usage management framework and the goals such a framework must achieve. This section explains the underlying meta-model of the framework, and the key principles and features that drive the design of the framework in achieving the goals.

Figure 3.1 shows the meta-model underlying the proposed usage management framework. The meta-model consists of policies and computing domain.

- **Computing Domain:** A computing domain represents the computing environment with respect to which policies are expressed, interpreted and enforced. For example, consider the event of opening of a file. In this case, the computing domain captures information such as the properties of the agent who carried out the act, the properties of the resource (a file in this case) over which the act was carried out, and the environmental circumstances including the time, date, location, etc., when the action

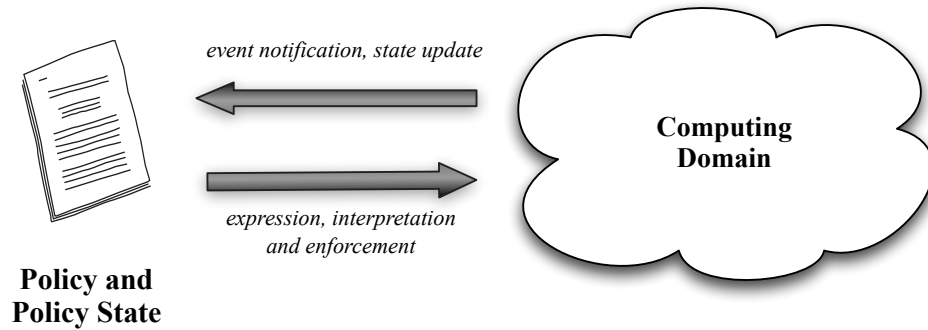


Figure 3.1: The Metamodel for the Usage Management Framework

occurs.

- **Policy and Policy State:** A policy describes the usage rules that govern the manner in which usage of content must be carried out within the computing domain. A policy may have many responsibilities, however, the primary responsibility of a policy is to determine whether or not a set of actions can be allowed to occur within a computing domain. A policy makes this decision based on three parameters, namely, the type of actions carried out on the content, the current state of the policy (which may record previous relevant actions that have been carried out), and the conditions of the computing domain under which the actions are carried out.

Over the years, numerous policy languages have been proposed in the areas of access control, usage control and DRM. Most of these languages fit the high-level meta-model shown in Figure 3.1. Policy languages differentiate themselves from each other based on the following two sets of characteristics, namely, 1) expression and reasoning semantics, and 2) operational semantics and deployment mechanisms.

The expression and reasoning semantics of policy languages depend on the type mathematical formalism used to express the language. Access control languages and models such as Bell and LaPadula [12, 13], focus on different types of access semantics and divide the computing environment in subjects and object attributes. Similarly role based

Chapter 3. Usage Management Framework

access control models allow modeling of subjects based on different role structures and their relationships, and grant access, authorizations and delegations based on these relationships [68]. Most of the policy languages have included entities such as subject, object and computation platform for modeling computing domains [59]. Languages that deal with DRM have a more complex structure in order include additional entities such as licensor, rights holder, licensee, and other rights management terms [77, 44]. These languages also include semantics such as permissions, obligations, penalties, and payments to name a few. The structure of the state is another aspect on the basis of which languages differentiate themselves. Different models and policy languages follow different ways of maintaining policy state. In the UCON model, state is maintained in terms of variables that represent mutable subject and resource attributes [59]. In XrML, the same functionality is achieved by means of a StatefulCondition extension, which maintains external state variables that allow one to maintain the state of the license [76]. Logic-based RELs proposed by Gunter et al. and Pucella et al., maintain policy state as a sequence of event traces [34, 64].

The operational semantics of policy languages determine the manner in which conditions are checked, policy state is updated and policy rules are validated. For example, whether the environment conditions should be updated before or during policy validation check? Similarly, should the state be updated before, during, or after the policy validation is carried out? The order in which these actions are carried out by the system dictates the operational characteristics of the model. The classification of different modes of operation in usage control is explained in the UCON model [59]. The deployment mechanisms for policy languages reflect the manner in which the policy model or language is implemented within the system. These include interpreter-based mechanisms, executable policies, server-client-based or web-based mechanisms. In interpreter based systems such as XrML and ODRL, an interpreter for the policy language runs on the client system [2, 76]. In executable policies, policies are executable pieces of code that travel along with the data across a system. Such an approach is followed by Ponder policy language, Mar-

Chapter 3. Usage Management Framework

lin and Coral [3, 27, 28]. The server-client deployment and the web-service deployment follow the same mechanism where the client system queries the server for validation of policies. One of the goals of the framework is to accommodate these different types of policy languages, operation modes and deployment mechanisms

As shown in the Figure 3.1, there is a strong coupling between a policy and a computing domain. Policies are expressed in terms of a computing domain, and policies are interpreted and enforced within a computing domain. This means the process of policy expression requires knowledge of the domain with respect to which policies are expressed. This knowledge includes the names of domain properties over which restrictions are expressed, the values over which the properties are defined, and the names of the identifiable actions that need to be controlled in the computing domain. Similarly, the process of policy interpretation and enforcement requires knowledge of policy syntax and semantics in order for the policy to be appropriately interpreted and enforced within the computing domain.

Such a tight coupling between policies and computing domains poses problems when policies are to be interpreted in multiple different computing domains in a distributed system. The proposed usage management framework aims to enable policies to be expressed, interpreted and enforced across multiple computing domains across a distributed system. To enable this, the framework adheres to the following set of principles mentioned below:

1. A maximum degree of separation between policies and computing domains.
2. A mechanism to reason about interoperability of policies with respect to computing domains.
3. Dynamic interpretation of policies across computing domains.
4. Accommodation of multiple policy languages within the framework.

Based on these principles, primary entities of the framework are described, followed by the deployment and operational semantics.

3.2 Objects and Interfaces

In order to achieve a high degree of separation between policies and computing domains, in the proposed framework both policies and computing domains are represented as objects. A policy is represented by a policy object, and a computing domain is represented by a context object.

3.2.1 Context Object

A context object is an instantiation of a context that captures the structure and the state of a computing domain. A context is defined according to the structure of the computing domain it represents. A context represents the entities within a computation domain, and the relationships among these entities. For a given computation domain, each entity is defined by a set of properties, and the context maintains the current values for each of those properties. The primary goal of a context object is to capture the conditions under which actions are carried out within a computation environment. A context maintains an interface that includes the following functions:

1. Update and retrieve relationships among the entities.
2. Update, retrieve and compare the attribute values for a given entity.

A context object is constantly updated by the system with appropriate values using the update interface. The interface of a context object depends on the type and complexity of the computing domain it represents.

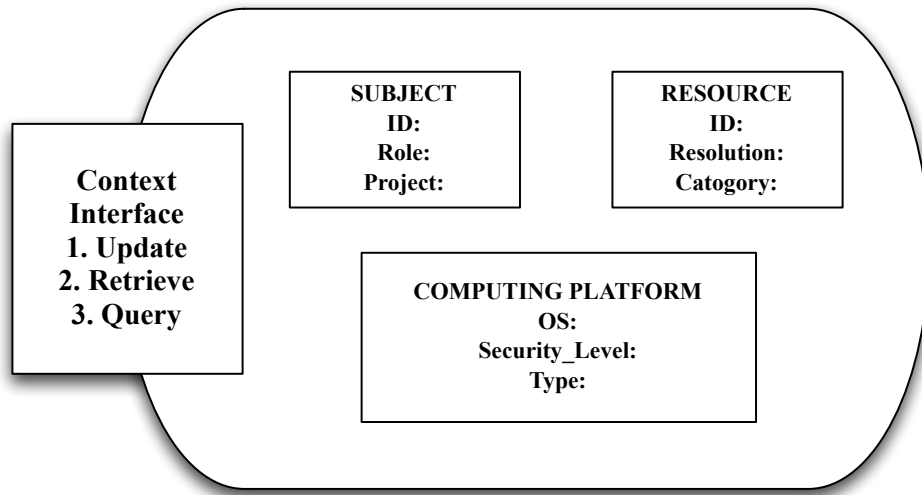


Figure 3.2: A context object

Figure 3.2 shows a context object. It consists of three entities, namely, Subject, Resource and Computing Platform. Each of these entities has a set of properties as shown in the figure. The context object maintains appropriate values for these properties and makes them available for policy validation. These properties are queried, updated, and retrieved by means of an interface provided by the context object.

3.2.2 Policy Object

A policy object is an executable object that has a behavior, a state, and an interface, as shown in Figure 3.3. Policy objects are used within the framework by means of the interface they expose to other systems. The concept of representing policy as an object has been used in a number of previous approaches for managing policies in a distributed environment [3, 27, 28].

The behavior of a policy object represents the policy rules that are expressed in a program logic. Different types of policy objects can make use of different policy languages

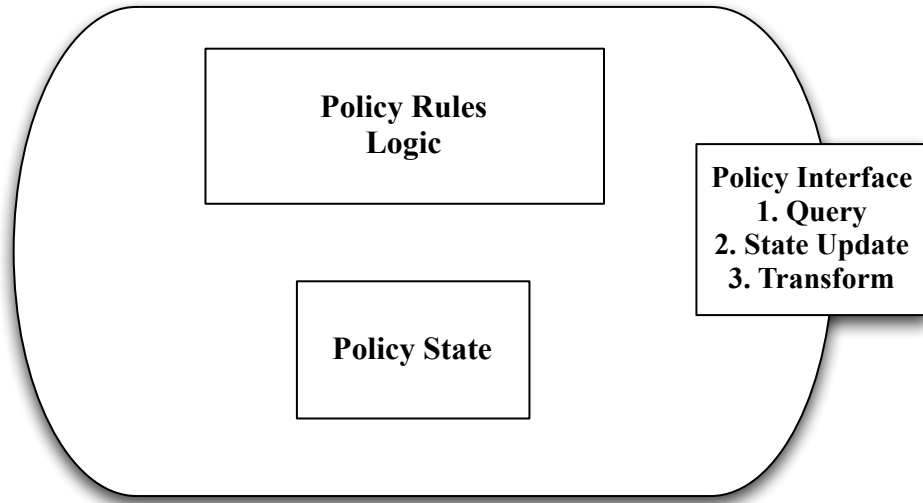


Figure 3.3: A policy object

with varying policy expression capabilities. Various usage semantics, such as, permissions, obligations, temporal dependencies, partial dependencies, and interleaving semantics can be captured by different types of policy objects. The operation of the framework is agnostic to the behavior of a policy object which is hidden from the other systems. This means, that policy descriptions from different policy languages can be transformed into policy objects, and still be used within the framework.

The state of a policy object captures the history of usage associated with the policy. Every time usage associated with a policy object is performed within a computing domain, policy object updates its state to record the history. As mentioned earlier, different policy languages have different ways of maintaining policy state. The manner in which event and usage histories are maintained within a policy object are hidden from the outside world. Thus policy languages that maintain usage histories in different ways such as state variables [59], or event traces [64], can be used within the framework to create policy objects.

The interface provides gateway that allows client systems to query policy objects. It is

Chapter 3. Usage Management Framework

necessary for systems that make use of a policy object to know how to use the interface provided by the policy object it intends to query. Depending on its type, a policy object will expose an appropriate interface to other systems. The type of interface exposed by a policy object will depend on the type of queries that the policy object can handle, which in turn, depends on the type of policy language underlying the policy object. A powerful underlying policy language will be able to support a richer interface, and simpler policy language will support a simpler interface. The functions maintained by license object interfaces can be categorized as follows:

1. **Query interface:** Query a license regarding decisions on usage.
2. **License state interface:** Update, retrieve or reset the state of a license.
3. **License processing interface:** Check compatibility with another license or merge a license with another license.

The next section provides a description of the working of the usage management framework, and how policy and context objects are used within the the framework.

3.3 Deployment and Operation

The working of the usage management framework is divided into two stages, namely, the Setup stage and the Working stage. The Setup stage includes the process of creation of a context object that corresponds to the underlying computing domain, and creation of a policy object. The Working stage includes the process of policy interpretation and policy enforcement within the computation domain.

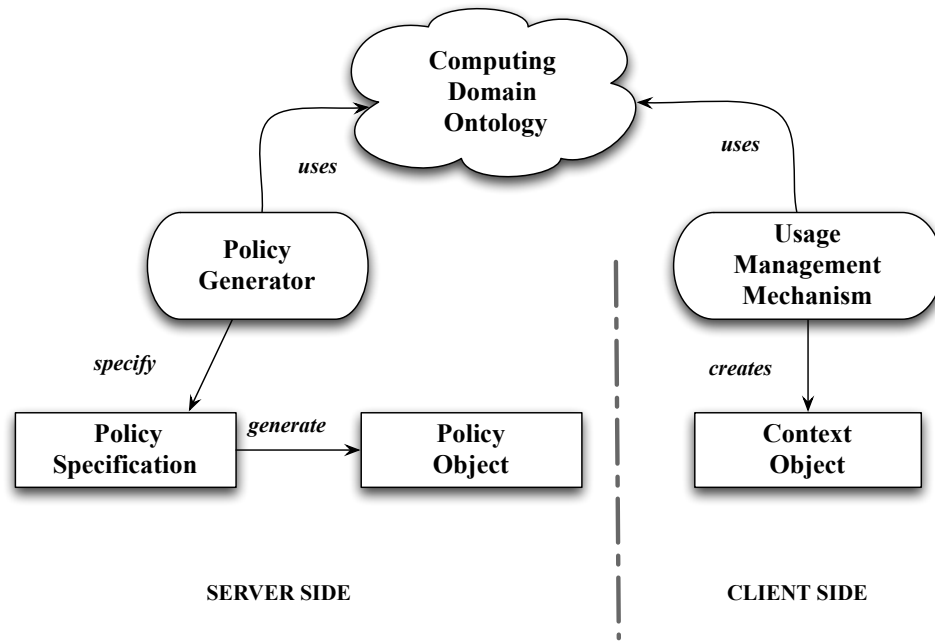


Figure 3.4: Setup stage for usage management.

3.3.1 Setup Stage

The Setup stage involves the creation of a policy object and a context object as shown in Figure 3.4. The policy object is generally created by the agent who defines the policy. This entity may be the owner of the resource or the system administrator. In case of a client-server system, a policy object may be created on the server side. A policy is defined by means of a policy specification language or some kind of a user friendly policy specification system. A policy generator transforms this policy into a policy object. In case policies are resource specific, a policy object may be attached to a resource by means of indirection using an identity resolution mechanism such as the Handle system [71].

The client system in which policies are to be interpreted and enforced creates a context object that captures the structure of the computing domain of the client system. Usage Management Mechanism (UMM) is a program running on the client system that acts as a

Chapter 3. Usage Management Framework

controller between context and policy objects. The working of the UMM is explained in the next section.

A context object is generated from the computing domain ontology that is representative of the client system. An ontology defines a common vocabulary for agents or programs that need to share information in a domain, and includes machine-interpretable definitions of basic concepts in the domain and relations among them. The same ontology is used by the policy specification mechanism to express policy constraints. A common ontology is necessary for an agreement over the vocabulary and structure of computation domain. The existence of a common ontology used for the creation of policy and context objects the basis for interoperability support within the framework. For example, an authorization policy expressed in terms of the context object shown in Figure 3.2 can be defined as follows:

“The ‘view’ action on a Resource with SecurityClassification greater than ‘Secret’ can be carried out only by a Subject working on Project ‘Alpha’ assuming the Role of a ‘Manager’ on a ComputingPlatform having SecurityType equal to ‘Secure’

In this case, the context object maintains the current state of computing domain when the view action is being carried out on a given resource. The next stage is the Working stage where a policy is interpreted and enforced within a client system.

3.3.2 Working Stage

The working stage, shown in Figure 3.5, shows the interactions among relevant components on a client system where policy is intended to be interpreted and enforced. At the end of the Setup stage, policy object and context object are generated from a common computing domain ontology that is representative of the client system.

UMM is the primary component of Working stage that coordinates the interactions

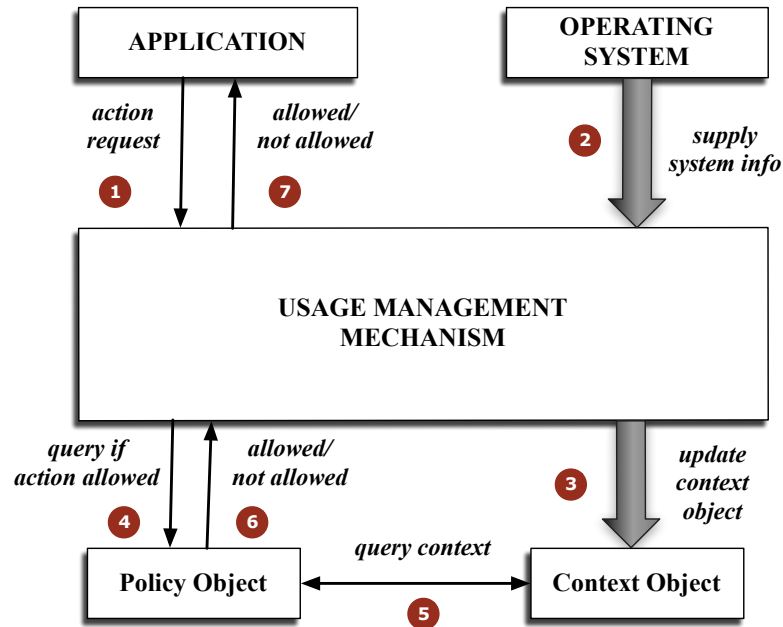


Figure 3.5: Working stage for usage management.

among all the other components. The UMM takes usage requests from applications and returns an appropriate result or answer to the requesting application after querying the policy object. UMM generates and maintains a context object that stores the values of system properties. It is also the responsibility of the UMM to obtain system values from the operating system of the underlying platform and update the context object with appropriate values corresponding to the present request. The application shown in the figure refers to any application that intends to use the resource in question. The UMM provides a standard interface for applications to query usage validity.

The information flow in a typical query for usage of a resource in system is as follows:

1. In this step, the application requests the UMM for a check on the validity of the action being carried out on a given resource. In this step, the application provides information to UMM regarding the properties of the resource, and that of the subject or the agent who is performing the intended action on the resource.

Chapter 3. Usage Management Framework

2. In Step 2, the UMM collects the information from the operating system to determine the conditions under which the action is intended to be carried out.
3. In Step 3, the UMM updates the state of the context object with information collected from the operating system and the application. It is the responsibility of the UMM to determine what type of information is necessary for updating the state of the context object.
4. In Step 4, after updating the context object, the UMM queries the policy object regarding the validity of the action requested by the application. In this step, the UMM running on the client system must know *a priori* how to use the interface provided by the policy object.
5. In Step 5, the policy object requests the context object with information necessary to evaluate the policy. It must be noted here that the policy object must know *a priori* how to call the functions provided by the interface of the context object. Accordingly, the context object provides the policy object with the necessary information to evaluate the policy.
6. In Step 6, the policy object evaluates the validity of action requested by the application, and returns an appropriate value to the UMM.
7. In Step 7, the UMM notifies the application whether or not the requested action is valid or invalid.

Next, analysis of the features of the framework, and how they help achieve the goals of usage management systems mentioned earlier is discussed.

3.4 Framework Analysis

This section provides an analysis of the features of the framework including interoperability, context and policy hierarchies and dynamic interpretation.

3.4.1 Interoperability Semantics

In traditional policy management systems, the syntax and semantics of policy languages is tightly coupled with the system within which the policy is to be interpreted. This approach generally requires an interpreter running on the client system that can interpret the policy. In the proposed framework the goal is to achieve maximum degree of separation between policies and computing domains.

The approach taken in this framework is to represent policies and computing domains in terms of objects with standardized interfaces that hide the internal workings from the other systems. Such a separation not only enables interoperability, it also allows the framework to accommodate multiple policy languages and different types of computing domains.

In order to analyze the interoperability characteristics of the framework, the interactions that take place in the framework are first explained. In the proposed framework, a UMM runs on every client system that provides a policy query interface to the higher-level applications that use the concerned resources. In order to process these queries, the UMM calls the functions provided by policy objects. Policy objects, in order to process usage queries, call the functions provided by the context object interface. Context objects also provide an interface to the UMM to update its state with the most recent values.

The interactions among the operating system, the UMM and the context object are internal to the client system. These interactions and interfaces can be unique to every client system. The interactions between a UMM and applications are also internal to

Chapter 3. Usage Management Framework

the client system, without affecting the operation of the framework. If the client system intends to support multiple third party applications, then it is necessary that this interface is standardized.

The important interfaces that determine the interoperability between policies and computing domains are: 1) Interface provided by policy objects to UMMs on client systems, and 2) Interface provided by context objects to policy objects.

UMM — Policy Object Interface. A UMM queries the validation of an action by querying a policy object by invoking functions provided by the policy object interface. Since the policy object is an executable program, the UMM need not understand the policy logic or the behavior of the policy object, and the manner in which it maintains its internal state. However, in order to support multiple policy objects of different types, it is necessary that the UMM knows *a priori* the set of functions provided by the interface of a given policy object. Hence, in order to support interoperability between UMMs and policy objects it is necessary the interface provided by policy objects is standardized. The interface provided by a policy object determines the type of queries that the policy object will support. As mentioned earlier, policy objects created from complex policy languages will support a richer interface compared to policy objects created from simpler policy languages. Hence a standard interface for all types of policy objects will limit the flexibility of the framework. To address this, policy objects are grouped in the form of a hierarchy as explained in the next section.

Policy Object — Context Object Interface. A policy object queries context object to obtain the current values of the properties of the computing domain. For a given policy object to query context objects residing in different computing domains, it is necessary that the interface provided by the different context objects standardized. This is one of the reasons why both policy and context objects need to be generated from a common computing domain ontology. Similar to policy objects, its is infeasible to expect all computing domains, and subsequently all context objects, to be identical and share the same

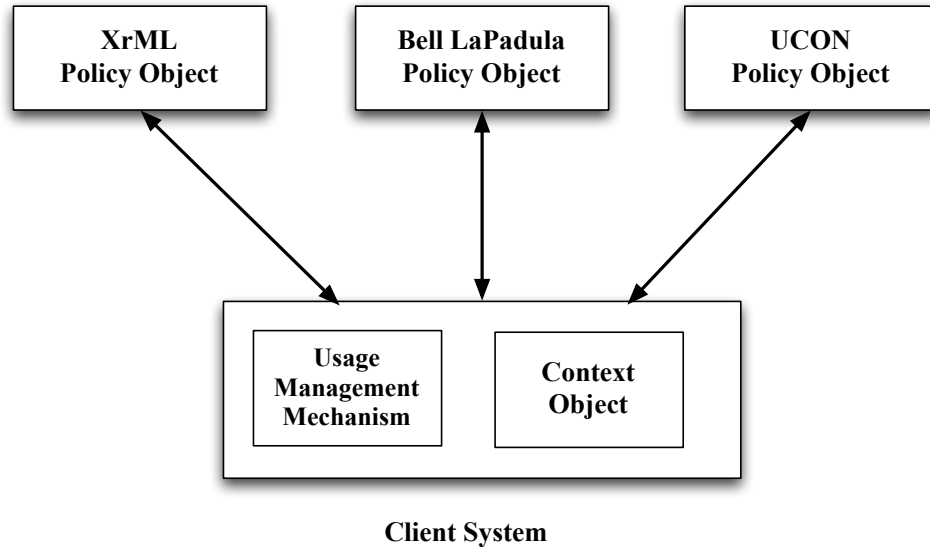


Figure 3.6: Multiple policy objects used on a single client system

entities and properties. In order to address this issue, contexts are grouped in the form of a hierarchy as explained in the next section.

Based on the above mentioned characteristics, it is possible to formally express interoperability between a policy and a UMM in terms of the interface and set of queries used by UMM, policy object and context object. Formal semantics for interoperability enables to determine on the fly whether or not a policy can be interpreted within a given computing domain. Let,

- pol denote a policy object,
- umm denote a usage management mechanism,
- con denote a context object maintained by umm ,
- \mathcal{F}_{umm} denote the set of functions used by umm to query policy objects,
- \mathcal{I}_{pol} denote the set of functions provided by the interface of the policy object pol ,

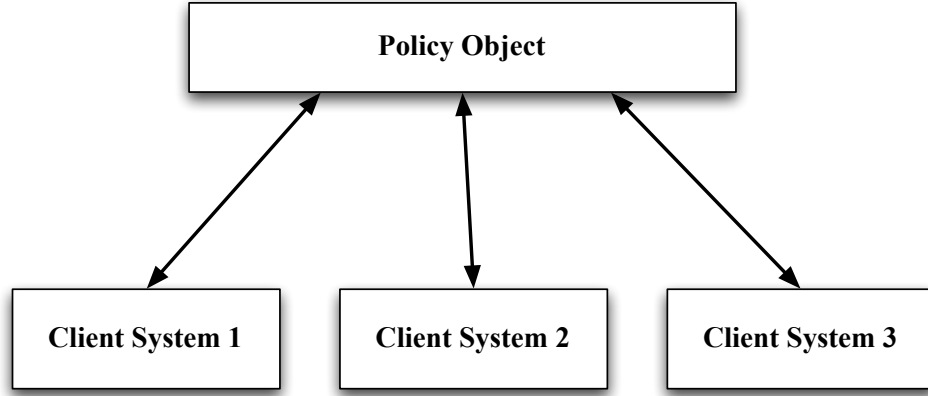


Figure 3.7: Single policy object used on multiple client systems.

- \mathcal{F}_{pol} denote the set of functions used by the policy object pol to query a context objects, and
- \mathcal{I}_{con} denote the set of functions provided by the interface of the context object pol

Based on these definitions, interoperability semantics are expressed in terms of context compatibility and policy compatibility. These compatibilities are expressed between policy objects and UMMs. Context compatibility between a policy object and a UMM denotes the ability of the policy object to query the context object maintained by the UMM. Context compatibility between pol and umm , denoted by $pol \bowtie^c umm$, holds if $\mathcal{F}_{pol} \subseteq \mathcal{I}_{con}$. Policy compatibility between a policy object and a UMM denotes the ability of the UMM to query the policy object. Policy compatibility between pol and umm , denoted by $pol \bowtie^p umm$, hold if $\mathcal{F}_{umm} \subseteq \mathcal{I}_{pol}$. We say pol is interoperable with umm , denoted by $pol \bowtie umm$ if and only if $pol \bowtie^c umm \wedge pol \bowtie^p umm$

Since the interactions take place only via interfaces, the semantics and logic of the policy language used to create policy objects is hidden from the client system. This creates a design space for accommodation of policy languages without affecting the operational semantics of the framework. This allows the use of different policy languages to be used

to create policy objects that can be interpreted and enforced in the same client system as shown in Figure 3.6. Similarly a single policy object can be interpreted and enforced in different client systems as shown in Figure 3.7.

Next section explains how policies and contexts can be organized in a hierarchical manner and how it is possible to reason about interoperability in terms of these hierarchical relationships.

3.4.2 Context and Policy Hierarchies

Compatibility between the structure of context and policy objects is necessary for interoperability in this framework. One solution is to standardize the interfaces supported by both policy and context objects. However, fixing these two entities will severely limit the flexibility of the model. No single context can cover all different types of computing domains. Similarly policy structure is a design space that is free for innovation and differentiation. Complex policy languages making use of sophisticated logics will have greater capabilities of expression and reasoning, and therefore it is necessary that they support a much larger and richer interface.

To address this, policy and context type are grouped in a hierarchical manner via inheritance relationships as shown in Figures 3.8 and 3.9. The most abstract entities at the top of the hierarchy will include features that are common to all entities. It must be noted that the inheritance relationships are primarily based on the interface supported by different policy and context types.

As shown in Figure 3.8, policy types are categorized in a hierarchical manner. The *RootPolicy* has an interface that provides the core set of functions that are common to all policies. This includes the function *allowed?()*, which all policy types must provide. Some of the basic categorizations of policies is shown in Figure 3.8. A primary categorization would be stateful and stateless policy types. A stateful policy type will require to pro-

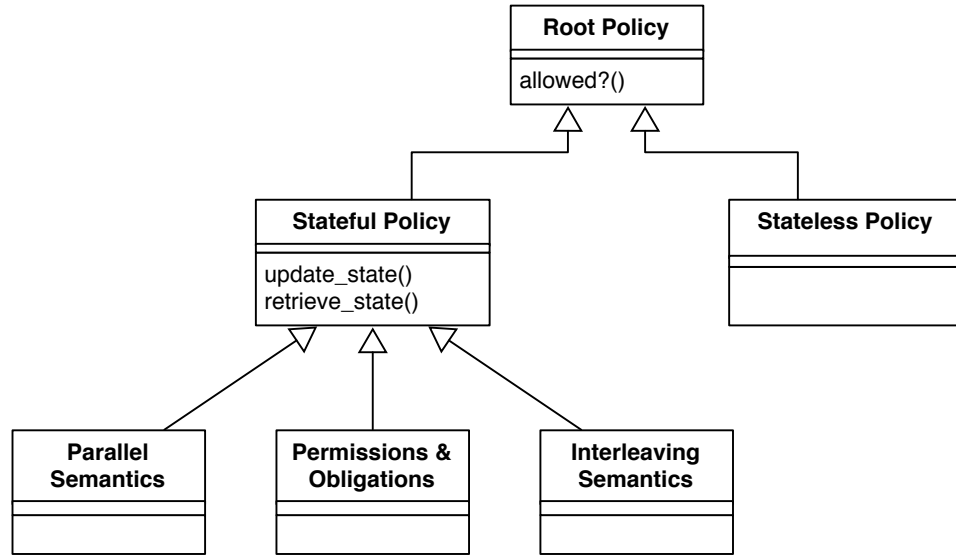


Figure 3.8: A hierarchy of policy types

vide an interface for update, retrieve, and reset of the policy state. As mentioned earlier, most policy languages allow state maintenance by means of state variable or event traces. However it should be noted that the manner in which a policy object maintains its state is hidden from UMMs. Stateful policies can be further categorized into the types of usage semantics such as parallel actions, interleaving semantics and deontic semantics such as permissions and obligations supported by the policy.

Similar to policy hierarchy, contexts can be categorized in a hierarchical manner as shown in Figure 3.9. For example the top-level contexts will include properties such as location, time, date etc., that are common to all contexts. Once these are defined, newer contexts can be inherited from them to create a context hierarchy. Using this approach, properties that are appropriate for different computing environments can be used to develop different context types. For example, a generic context type *network* can be created, and can be used to further develop different types of network contexts such as wired network, wireless network, home network and sensor network, etc. that inherit the properties of network context. Similarly, a generic context of type *device* can be constructed with

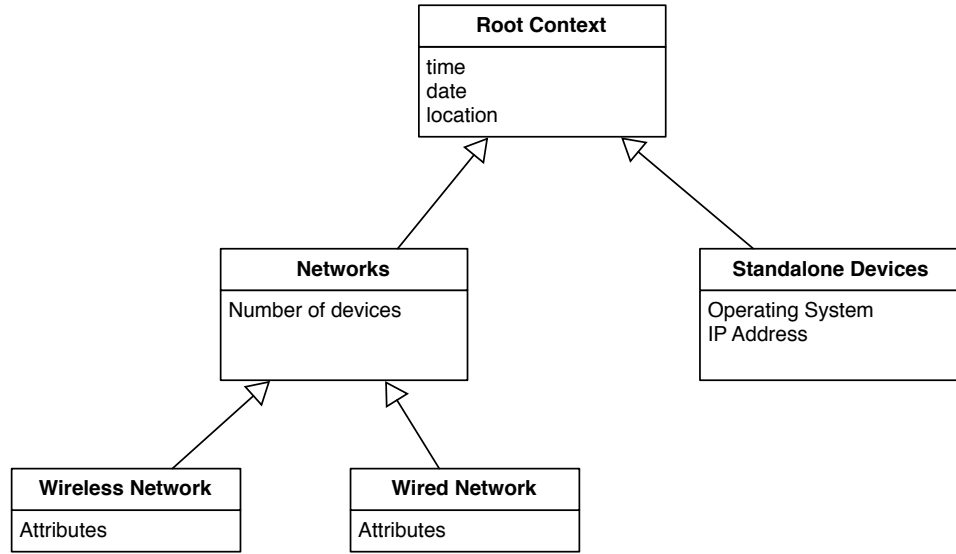


Figure 3.9: A hierarchy of context types

properties that are inherited by contexts representing different types of devices such as hand held devices, laptops, smart phones, etc.

If the relationship among a group of policy types and context types respectively can be established, then then it is possible to reason about interoperability based on these relationships. Let,

1. Let umm_1 maintain context con_1 , and use policy query functions denoted by set \mathcal{F}_{umm_1}
2. Let umm_2 maintain context con_2 , and use policy query functions denoted by set \mathcal{F}_{umm_2}
3. Let con_1 provide an interface \mathcal{I}_{con_1} , and con_2 provide an interface \mathcal{I}_{con_2}
4. Let policy type pol_1 maintain policy interface \mathcal{I}_{pol_1} and use context functions \mathcal{F}_{pol_1} , and policy type pol_2 maintain policy interface \mathcal{I}_{pol_2} and use context functions \mathcal{F}_{pol_2}

Chapter 3. Usage Management Framework

We say that context con_2 inherits context con_1 , denoted by, $con_2 \supseteq con_1$ if $\mathcal{I}_{con_1} \subseteq \mathcal{I}_{con_2}$. Similarly, policy pol_2 inherits policy pol_1 , denoted by, $pol_2 \supseteq pol_1$ if $\mathcal{I}_{pol_1} \subseteq \mathcal{I}_{pol_2}$.

Based on the inheritance relationships, it is now possible to reason about interoperability of policies with UMMs with following inferences,

1. $(pol_1 \bowtie^p umm) \wedge (pol_2 \supseteq pol_1) \implies pol_2 \bowtie^p umm,$
2. $(pol \bowtie^c umm_1) \wedge (con_2 \supseteq con_1) \implies pol \bowtie^c umm_2,$
3. $(pol_1 \bowtie umm) \wedge (pol_2 \supseteq pol_1) \wedge (\mathcal{F}_{pol_2} \subseteq \mathcal{F}_{pol_1}) \implies pol_2 \bowtie umm$
4. $(pol \bowtie umm_1) \wedge (con_2 \supseteq con_1) \wedge (\mathcal{F}_{umm_2} \subseteq \mathcal{F}_{umm_1}) \implies pol \bowtie umm_2$

Next, the notion of dynamic interpretation introduced that allows policies to be interpreted in different ways in different or same computing domains.

3.4.3 Dynamic Interpretation

As mentioned earlier, the one of the design goals of the proposed framework is to achieve maximum degree of separation between policy definitions and the computing domains within which policies are interpreted and enforced. This separation is essential in a distributed environment where policies are expressed with minimal prior knowledge of the computing domains where they might be interpreted and enforced. One important aspect of usage policies are the usage verbs used to define usage in policies. Different types of usages can be expressed using verbs such as “play”, “view”, “print”, and “loan”, etc. When a policy creator defines the usage policy, he/she uses this vocabulary to express the policy terms. Each of these verbs has a specific interpretation depending upon the computing domain within which it is interpreted. The dynamic interpretation feature allows these verbs to be interpreted dynamically depending on the computing domain within which a given policy is interpreted.

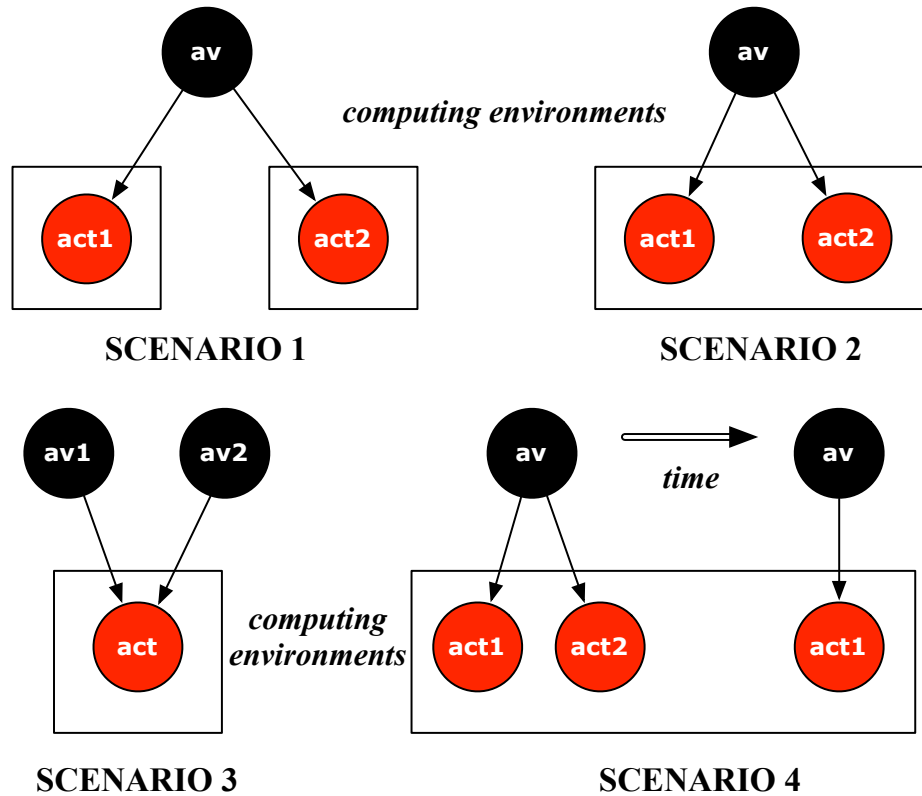


Figure 3.10: Dynamic interpretation of activities in terms of identifiable actions in the computing domain..

In every computing domain, actions are carried out as discrete identifiable events. These events are different for each computing environment, and may depend on the type of computing platform, operating system, etc. Hence, every verb can be interpreted within a computing environment in terms of either a single event or a sequence of such events. From here on, the term “activity” is used for the verbs used in a *policy*, and the term *action* is used for identifiable events in a computing domain.

The separation of the verbs used in licenses and their interpretation within computing environments allows *dynamic interpretation* of policies. The interpretation of activities in terms of identifiable actions within a given computing domain is carried out by means of

Chapter 3. Usage Management Framework

an interpretation map called Action Map. This interpretation map is independent of the rest of the functioning of the framework. This design choice leads to powerful capabilities that are not possible systems that rely on static interpretation.

Figure 3.10 shows the different ways in which activities in a policy can be dynamically interpreted. Scenario 1 shows that the same activity in a given policy can be interpreted differently in two different computing environments. For example, the activity *play* can have one interpretation in an iPhone environment, and a different interpretation in a personal computer environment. Scenario 2 shows how a given activity can be interpreted as multiple actions. For example an activity such as *view* can be interpreted as both *openfile* and *printfile* action in a given computing domain. Scenario 3 depicts the situation where two different activities are interpreted as the same action within a computing domain. For example, a license expressing two activities, *view* and *play* can be interpreted to imply the same action *openfile* in a given computing environment. Scenario 4 shows an interesting use case where the interpretation of a changes depending on external factors such as time, security threats, known attacks on the system, etc. External changes in computing environments often require a change in the interpretation of a policy. For example, in case of a security breach, it may be necessary to interpret a policy more strictly or the technological changes in computing devices may require a policy to be interpreted in a different manner. Dynamic interpretation is a powerful design feature. The fact that interpretation of a policy can be dynamically changed without affecting other parts is a distinguishing feature of this framework. Thus enabling interpretation of policies modulated by external events is a powerful design feature of this framework.

Next, a mathematical framework is introduced that can be used to model policies and contexts. Based on this model, policy languages can be generated by incorporating different types of mathematical logics.

3.5 Mathematical Model

In this section a formal model to structure policies and context is introduced. It must be noted that the policy framework explained in the previous section is independent of the mathematical model described here. This model simply provides one of the ways to structure usage policies. The model provides a design space created for usage semantics, where different logics can be used to express different types of usage semantics. An overview of the model is provided below.

A context object consists of three entities, namely, subject, resource and environment, where each entity has a set of attributes. A policy object consists of a set of activities, where each activity is wrapped in access rules expressed in terms of the attributes of subject, resource and environment. Access rules specify the circumstances (determined by the values taken by subject, resource, and environment attributes) under which a particular activity can be performed. An activity wrapped in access rules is called *restricted activity*. A usage policy specifying usage rules, referred to as *policy expression* in the model description, is then defined over a set of restricted activities. A usage policy specifies semantics such as permissions, obligations, partial ordering, interleaving semantics, count limits, and other such usage rules. Usage policy semantics and validation are dependent only on the policy state, and are therefore independent of the context object. An activity is permitted, given that it satisfies both the usage policy and the access policy. In the proposed mode, a first order language for access policy specification, and a simple usage policy language is defined that capable of expressing only permissions and obligations is defined. Both these choices are made only for the purpose of demonstrating the capabilities such a policy structure offers. Both these design spaces can be substituted with different types of logics to create policy types with advanced expression and reasoning capabilities.

The proposed model provides two advantages, namely, separation of access and usage rules, and formal semantics for interoperability. In the policy structure, a design space

Chapter 3. Usage Management Framework

created for access and usage rules. In currently used policy languages, these two aspects of usage control specification are incorporated within a single policy language. However, it is difficult to create a universal policy language that is able to express all types of usage semantics, and also incorporate access rules. There does not exist a policy specification language that is able to express all types of usage and access semantics.

The separation of these two aspects of a policy allows an independent choice of usage semantics and access semantics to be used within a policy. Sophisticated access control policies such as the one proposed by Halpern et al. [37] can be used in the design space created for generation of restricted activities. There exist well-developed mathematical logics such as deontic logic, linear temporal logic, dynamic logic, etc., that can express the usage semantics [43]. Policy languages have been developed to express usage semantics using these logics [64, 65]. It is therefore possible to develop usage policy languages or use existing ones with modifications in this design space. The ability to leverage the use of existing languages for usage rules expression and access rules expression independent of one another is an important advantage of the proposed model.

3.5.1 Model Description

This section describes the primary entities in the usage management model, namely, context, policy and usage management mechanism. As mentioned earlier, a context defines the computing domain within which usage management is carried out. A context models the computing environment within which usage management is carried out, agents or subjects operating in the system, and resources within the system. A policy consists of usage rules that determine in what manner the resources in the system must be used by the agents. A usage management mechanism interprets policies and coordinates the communication between policy and context objects. Each of the entities are described in detail below.

Table 3.1: An example structure of context.

Context			
Entity	Property (p)	Domain (D_p)	Functions (F_{D_p})
Environment (E)	Date	{01/01/0000, ..., 12/31/9999}	{on, before, after, between}
	Location IPAddress	{Set of all countries} {0.0.0.0, ..., 255.255.255.255}	{equals} {equals, between}
Subject (S)	SubjectID	{000-00-0000, ..., 999-99-9999}	{equals}
	SecurityClearance	{A, B, C, D, E, F}	{equals, lesser, greater, between}
Resource (R)	ResourceID	{000-000, ..., 999-999}	{equals}
	SensitivityLevel	{1, 2, 3, 4, 5, 6, 7}	{equals, lesser, greater, between}

Context

A context is defined by the tuple $C = \langle E, S, R \rangle$, where E represents the set of system environment properties, S represents the set of subject properties and R represents the set of resource properties. Each property is a place holder for a set of values. The values taken by a property p range over the elements in its unique respective domain D_p .

The sets E , S and R represent the type of environment, subject, and resource, respectively, uniquely defined by the set of properties contained in each set. An instance of an environment type E , denoted by iE , is defined by the tuple $\langle p_1 = k_1, \dots, p_n = k_n \rangle$, where $E = \{p_1, \dots, p_n\}$ and $k_i \in D_{p_i}$. Subject and resource instances, denoted by iS and iR are defined similarly. A context instance is defined by the tuple $iC = \langle iE, iS, iR \rangle$, with the corresponding context type $C = \langle E, S, R \rangle$

Every property p has a non-empty set of boolean functions or predicates defined over the domain represented by the set F_p . The boolean functions allow to compare the values of properties of the entities defined.

A constraint provides restrictions on the context within which usage is carried out. An environment constraint provides restrictions on the properties of the computing environment; a subject constraint provides restrictions on the properties of the subject performing the usage; and a resource constraint provides restrictions on the properties of the resource which is being used.

Chapter 3. Usage Management Framework

A constraint is defined as a set of restrictions over a given context. Constraints for a given property are expressed in terms of boolean functions defined for the domain of the property. A constraint for a given context $C = \langle E, S, R \rangle$ is defined by the set $cr_C = \{cr_E, cr_S, cr_R\}$, where cr_E , cr_S and cr_R are the constraint for environment E , constraint for subject S , and constraint for resource R respectively. Constraints cr_E , cr_S and cr_R are defined as follows:

$$\begin{aligned} cr_E &= \bigwedge |f| \neg cr_E | cr_E \wedge cr_E | cr_E \vee cr_E, \text{ where, } f \in F_p, p \in E, \\ cr_S &= \bigwedge |f| \neg cr_S | cr_S \wedge cr_S | cr_S \vee cr_S, \text{ where, } f \in F_p, p \in S, \text{ and} \\ cr_R &= \bigwedge |f| \neg cr_R | cr_R \wedge cr_R | cr_R \vee cr_R, \text{ where, } f \in F_p, p \in R. \end{aligned}$$

A context instance, iC , satisfies a context constraint cr_C , denoted by $iC \models cr_C$, if and only if $iE \models cr_E \wedge iS \models cr_S \wedge iR \models cr_R$, where,

$$\begin{aligned} iE &\models cr_E \text{ if } cr_E \text{ evaluates to true under the values taken by } iE \\ iS &\models cr_S \text{ if } cr_S \text{ evaluates to true under the values taken by } iS \\ iR &\models cr_R \text{ if } cr_R \text{ evaluates to true under the values taken by } iR \end{aligned}$$

The context, context instance, and context constraint defined above provide the basis for the structure of policy and usage management mechanism.

Example of a Context —

A sample of a context structure, for context C , is shown in Table 3.1, which shows the properties of the set of Environment (E), Subject (S) and Resource (R), and their respective domains. As shown in the table, every domain has a set of boolean functions. Table 3.2 shows the instance of the context, denoted by iC , defined in Table 3.1, where each of the properties are assigned values from its respective domain.

Table 3.2: An example structure of context instance.

Context instance 1 (i_1C)	
Entity	Property (p)
Environment instance (iE)	iDate = 04/05/2010 iLocation = France iIPAddress = 127.0.0.1
Subject instance (iS)	iSubjectID = 876-76-7896 iSecurityClearance = B
Resource instance (iR)	iResourceID = 789-455 iSensitivityLevel = 3

Context instance 2 (i_2C)	
Entity	Property (p)
Environment instance (iE)	iDate = 01/21/2010 iLocation = USA iIPAddress = 127.0.0.1
Subject instance (iS)	iSubjectID = 876-76-7896 iSecurityClearance = F
Resource instance (iR)	iResourceID = 789-455 iSensitivityLevel = 3

Table 3.3: An example description of functions.

Boolean Function	Description
$E.Date.between(d1, d2, iDate)$	True if $iDate$ lies between dates $d1$ and $d2$.
$E.Location.equals(loc, iLocation)$	True if $iLocation$ equals the location loc .
$E.IPAddress.between(ip1, ip2, iIPAddress)$	True if $iIPAddress$ lies between $ip1$ and $ip2$.
$S.SubjectID.equals(id, iSubjectID)$	True if $iSubjectID$ equals id .
$S.SecurityClearance.greater(sc, iSecurityClearance)$	True if $iSecurityClearance$ is greater than the value of sc .
$R.ResourceID.equals(id, iResourceID)$	True if $iResourceID$ equals the value of id .
$R.SensitivityLevel.between(sl1, sl2, iSensitivityLevel)$	True if $iSensitivityLevel$ lies between $sl1$ and $sl2$.

The domain functions are shown in the last column of Table 3.1. Each function can be used by the policy to express restrictions over the values of a given property. To uniquely identify functions, they are represented in the form of *Entity.Property.Function()*. The function $E.Date.between(01/01/2010, 31/01/2010, iDate)$, determines if the value of $iDate$ in the context iC lies between the dates 01/01/2010 and 31/01/2010. Some of the functions from Table 3.1, and their intended meaning are described in Table 3.3.

Boolean functions are used to express constraints over a given context. For example constraints for the usage of a given resource in terms of the context described in Table 3.1 can be expressed as follows:

Chapter 3. Usage Management Framework

“The usage must be carried out between January, 1, 2010 and January 31, 2010, only in USA and Canada. The usage can be carried out only on a resource with sensitivity level between 2 and 5. The usage must be carried out only by subjects with security clearance greater than level A.”

A context constraint for the above mentioned policy is expressed as follows:

$$cr_C = \langle cr_E, cr_S, cr_R \rangle, \text{ where,} \quad (3.1)$$

$$\begin{aligned} cr_E = & E.Date.between(01/01/2010, 31/01/2010, iDate) \wedge \\ & (E.Location.equals(USA, iLocation) \\ & \vee E.Location.equals(Canada, iLocation)), \end{aligned}$$

$$cr_S = S.SecurityClearance.greater(A, iSecurityClearance), \text{ and}$$

$$cr_R = R.SensitivityLevel.between(2, 5, iSensitivityLevel).$$

From the examples shown in Table 3.2, it can be seen that $i_1C \not\models cr_C$, because the date and location constraints are not satisfied. However, it can be seen that $i_2C \models cr_C$

Policy

A policy, denoted by pol , is defined by the 3-tuple $pol = \langle RA, \varepsilon, \mathcal{I}_{pol} \rangle$

1. RA defines the set of restricted activities in a policy.
2. ε defines the policy usage expression over RA .
3. The interface \mathcal{I}_{pol} defines the set of functions supported by the policy pol .

We now explain each of these terms in detail.

Chapter 3. Usage Management Framework

Restricted Activity. As mentioned earlier, an activity is a policy abstraction for different operations that may be carried out in a system. For example, verbs such as *use*, *play*, *pay* and *operate* represent specific set of operations that are performed by agents in a given system. Such verbs when used in a policy are called activities, denoted by av . An activity may be accompanied by a set of constraints over the context within which the activity is to be carried out. Activities accompanied with constraints are called restricted activities.

A restricted activity, rv , is defined by the tuple $rv = \langle av, cr_C \rangle$, where av is an activity, and cr_C is a context constraint. Restricted activities are the building blocks for specifying usage policies.

An activity instance is an activity av that is being performed with respect to a context instance iC , is denoted by $iav = \langle av, iC \rangle$.

An activity instance $iav = \langle av_1, iC \rangle$ conforms to a restricted activity $rv = \langle av_2, cr_C \rangle$, denoted by $iav \propto rv$, if and only if $av_1 = av_2$ and $iC \models cr_C$.

The conformance of a restricted activity can only be determined with respect to an activity instance which contains complete information about state of the context when the activity is being performed. This information must be provided to a policy by the usage management mechanism.

Restricted Activity Example —

Consider a restricted activity $rv = \langle view, cr_C \rangle$, where $view$ is the activity and cr_C is the constraint defined over context C by equation 1. The context C is described in Table 3.1. We now define three activity instances, namely, $iav_1 = \langle view, i_1C \rangle$, $iav_2 = \langle view, i_2C \rangle$ and $iav_3 = \langle print, i_2C \rangle$, where i_1C and i_2C are described in Table 3.2. The following relationships hold:

- $iav_1 \not\propto rv$, since $i_1C \not\models cr_C$.

Chapter 3. Usage Management Framework

- $iav_2 \propto rv$, since $i_2C \models cr_C$, and $view == view$.
- $iav_3 \not\propto rv$, because even though $i_2C \models cr_C$, the activity defined in iav_3 is *print*, and that defined in rv is *view*.

Policy Expression and Policy Interface. A policy expression provides usage semantics over a set of restricted activities. These may include permissions, obligations, parallel actions, partial dependancies, interleaving semantics and count limit to name a few. For the purpose of this discussion, a simple usage policy language that is capable of expressing permissions and obligations is introduced. It must be however noted that policy expression provides a design space where different types of languages capable of expressing and reasoning about different types of usage semantics can be used. Hence it is possible to replace this language by a different language capable of a greater expression power.

In this simple usage policy language, a user can specify the set of permissions \mathcal{P} , the set of obligations \mathcal{O} , what set of obligations are associated with a given permission, and a mechanism to record which of the obligations have been fulfilled. The set of permissions and obligations are mutually exclusive to avoid cyclic dependancies. Mathematically, the policy expression will include the following set of definitions:

1. Set of permissions $\mathcal{P} \subseteq RA$,
2. Set of obligations $\mathcal{O} \subseteq RA$, s.t. $\mathcal{P} \cap \mathcal{O} = \phi$,
3. Function $f_{\mathcal{PO}} : \mathcal{P} \rightarrow 2^{\mathcal{O}}$, returns the set of obligations associated with a permission.
4. Function $f_O \rightarrow \{true, false\}$, is set to *true* if the obligation is carried out, and is set to *false* otherwise.

The final element in the policy definition is the policy interface I_{pol} . Policy interface reflects the type of queries that a usage management mechanism or a user can ask a policy.

Chapter 3. Usage Management Framework

For the usage policy language described here, the policy interface I_{pol} may contain the following functions.

1. **allowed** (**av**, **iC**) — determines whether or not a given activity performed under the given context state can be carried out.
2. **permissions**() — returns the set of permissions.
3. **obligations**(**rv**) — returns the set of obligations for a given permission.
4. **remaining_obligations**(**rv**) — returns the set of obligations for a given permission that are not yet fulfilled.
5. **update_obligation**(**av**, **iC**) — updates the state of the policy specifying that the obligation is being fulfilled.
6. **reset**() — resets the license by setting all the obligation states to false.

The algorithms for the following functions are as follows:

allowed ($iav = \langle av_1, iC \rangle$)

```
1  constraint_satisfied_permissions = []
2  for all  $rv = \langle av, cr_C \rangle \in \mathcal{P}$  do
3      if ( $av == av_1$ )  $\wedge$  ( $iav \propto cr_C$ ) then
4          add  $rv$  to constraint_satisfied_permissions
5  valid_permissions = []
6  for all  $rv \in \text{constraint\_satisfied\_permissions}$  do
7      obligations = ( $f_{PO}(rv)$ )
8      if (for all  $ob \in \text{obligations}$ ,  $f_O(ob) == \text{true}$ ) then
9          add  $rv$  to valid_permissions
10 if valid_permissions  $\neq \text{empty}$  then
```

Chapter 3. Usage Management Framework

```
11    return true
12 else
13    return false
```

The **allowed** algorithm works as follows. Lines 2-4 selects all restricted activities in the set \mathcal{P} that are satisfied by the context values in iC , and have the same activity as supplied by the usage management mechanism. These are the set of permissions that satisfy the constraints, and are stored in the array *constraint_satisfied_permissions*. Lines 6-8 checks whether or not all the obligations for a given permission are satisfied. Permissions whose obligations are satisfied are stored in the array *valid_permissions*. Finally, if the array *valid_permissions* is not empty, the function returns *true* else returns *false*. The algorithms **permissions** and **obligations** are self explanatory.

permissions ()

```
1  return  $\mathcal{P}$ 
```

obligations(*rv*)

```
1  return  $f_{OP}(rv)$ 
```

remaining_obligations(*rv*)

```
1  rem_obligations = []
2  for all  $ob \in f_{OP}(rv)$  do
3      if  $f_O(ob) == false$  then
4          add ob to rem_obligations
5  return rem_obligations
```

Chapter 3. Usage Management Framework

The **remaining_obligations** algorithm works as follows. It takes as an input a permission in the form of a restricted activity. Lines 2-4 selects all obligations for that permission whose values are set to false. and returns the set of these obligations.

```
update_obligation( $iav = \langle av_1, iC \rangle$ )
1   $matching\_obligations = []$ 
2  for all  $ob = \langle av, cr_C \rangle \in O$  do
3      if  $(av == av_1) \wedge (iav \propto cr_C)$  then
4          add  $ob$  to  $matching\_obligations$ 
5  for all  $ob \in matching\_obligations$  do
6       $f_O(ob) = true$ 
```

The **update_obligations** algorithm works as follows. In this algorithm, the *umm* provides an activity instance set $iav = \langle av_1, iC \rangle$. It tells what activity av_1 is performed and under what conditions iC . Lines 2-4 selects all obligations that match the description, and sets values of those obligations to true on Line 6 indicating that these obligations are fulfilled. The **reset** algorithm is self explanatory.

```
reset()
1  for all  $ob \in O$  do
2       $f_O(ob) = false$ 
```

Usage Management Mechanism

As explained in Section 3.3, a usage management mechanism is an entity that operates within a computing domain, and is responsible for interpreting and enforcing policies within the computing domain. In this mathematical framework a UMM, denoted by *umm*, is defined by the 3-tuple $umm = \langle \mathcal{F}_{umm}, Act, iC \rangle$, where,

Chapter 3. Usage Management Framework

1. \mathcal{F}_{umm} defines the set of functions that the UMM uses to query policies. A policy interface must support all the functions defined in the the set \mathcal{F}_{umm} .
2. Act is the set of identifiable actions provided in the computing domain. The actions defined in the set Act are the actions that are performed by the users, and can be uniquely identified within the computing environment. These actions are the application-level or operating system-level commands that can be identified within the computing environment. For example such actions can be “*openfile*”, *writefile*, *copyfile*, etc. Activities defined in the license are mapped to these actions. This mapping is described in Section 3.5.2.
3. iC is the context instance maintained by the usage management mechanism. Whenever an action is requested within the computing environment, the UMM updates the context instance iC to record the state of the context under which the action is to be carried out.

The UMM on the computing platform is used by the higher level applications and the security mechanisms. If the computing platform undergoes changes in technology, the sets \mathcal{I}_{umm} , Act and iC are changed accordingly.

3.5.2 Dynamic Interpretation

The activities or the verbs used in license expression are abstract terms, such as *use*, *copy*, *transfer*, *play*, etc. It is possible to agree upon and fix the activities that will be used to express licenses for a particular type of usage management ecosystem. On the other hand, the computing environments deal with actions that are identifiable and closely relate to the ones supported by operating systems. A usage management mechanism is agnostic to the activities defined in a license, and a license is agnostic to the actions defined in the usage management mechanism. An interpretation function is a map that maps the activities in a

Chapter 3. Usage Management Framework

license to the actions defined in the usage management mechanism. For the simplicity of the discussion it is assumed that the map is one-to-one.

Formally, an interpretation function $Int : Acv \rightarrow Act$, where Act is the set of activities, and Act is the set of actions. For example:

$Acv = \{view, transfer, modify\}$, and

$Act = \{openfile, writefile, sendfile\}$, then

$Int(view) = openfile, Int(transfer) = sendfile$, and

$Int(modify) = writefile$.

Next section describes the operational semantics for this framework similar to the ones described in Section 3.3

3.5.3 Operational Semantics

The operational semantics of the calculus are shown in Figure 3.11. The entities shown in the figure are similar to the ones discussed earlier, with the addition of the interpreter. A typical usage management operation takes place as follows.

The process of policy interpretation and enforcement takes place in eight steps as shown in the figure. In *Step 1* the application requests the usage management mechanism to determine whether a particular action (*act*) on a given resource by a given user is valid or invalid. As a part of the request, the application provides all the information about the user, the resource and the action that is to be performed. In *Step 2* the usage management mechanism obtains the current state of the computing environment from the operating system. The type of information obtained by the usage management mechanism depends on the manner in which the context is modeled. Such information may include, current location, day, date, time, ip address, etc. In *Step 3* the usage management mech-

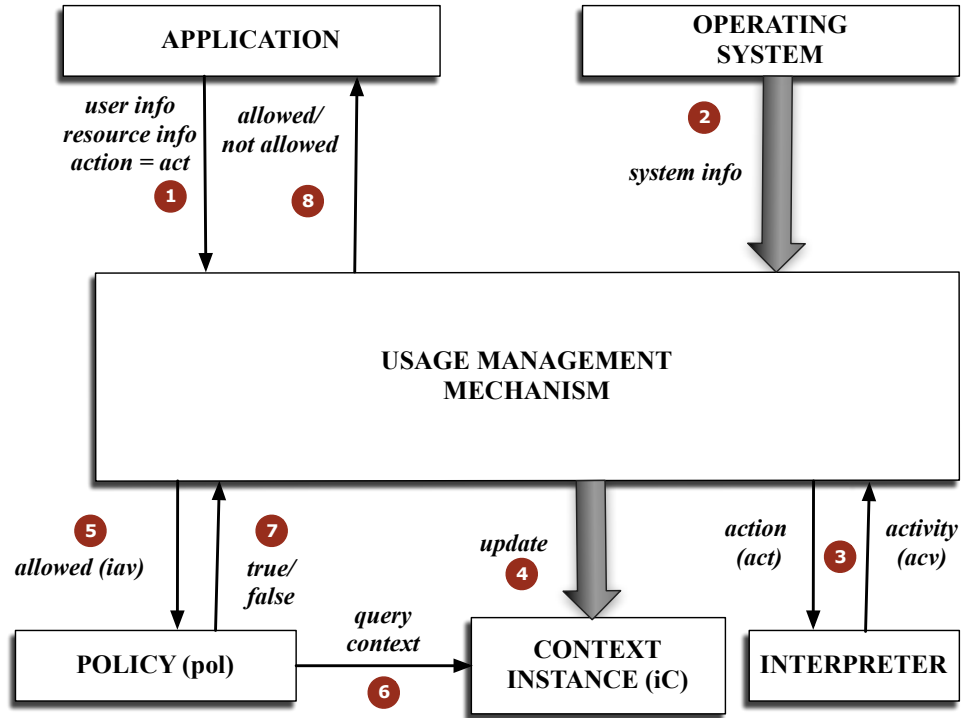


Figure 3.11: Operational semantics of the calculus.

anism consults the interpreter for the activity (*acv*) that corresponds to the action (*act*) requested by the application. The interpreter is a map as explained in the earlier section, and provides the corresponding activity (*acv*). In *step 4*, the usage management mechanism updates the context instance *iC* by using the current values of system parameters, user information and resource information. This is followed by *step 5*, where the usage management mechanism queries the policy whether or not the specified activity is valid. In *step 6* the policy queries the context instance *iC* regarding the system values and subject and resource properties while the activity is to be performed. Based on these values, the policy determines the validity of the action and provides an appropriate answer in *Step 7*. In *Step 8*, the usage management mechanism provides the result of the query to the application.

In the next section the interoperability semantics of this model are defined.

3.5.4 Interoperability Semantics

The interoperability semantics and results explained in Section 3.4.1 hold for this mathematical model for policies. However, given the well defined internal structure for context, policies and UMM, it is possible to define these semantics in more detail. In Section 3.4, the following definitions were introduced: $\mathcal{I}_{pol}, \mathcal{F}_{pol}, \mathcal{I}_{con}, \mathcal{F}_{umm}$. Based on these sets, interoperability relationships between policies and UMMs are defined. Let a usage management system have a context $con = \langle E, S, R \rangle$, a policy $pol = \langle RA, \varepsilon, \mathcal{I}_{pol} \rangle$ and a usage management mechanism $umm = \langle \mathcal{F}_{umm}, Act, icon \rangle$. Based on these definitions, it is possible to calculate the values for $\mathcal{I}_{pol}, \mathcal{F}_{pol}, \mathcal{I}_{con}, \mathcal{F}_{umm}$ as follows.

The values for \mathcal{F}_{umm} and \mathcal{I}_{pol} are provided in the definitions of umm and pol . Hence policy compatibility definition $\mathcal{F}_{umm} \subseteq \mathcal{I}_{pol} \implies pol \bowtie^p umm$, provided in Section 3.4.1, holds.

\mathcal{I}_{con} , the interface provided by context con is calculated as follows. $\mathcal{I}_{con} = \bigcup_p F_p$, where $p \in E \cup S \cup R$, and F_p is the set of all functions provided by property p .¹

The set \mathcal{F}_{pol} is calculated as follows. Let the functions used in a given constraint be denoted by f_{cr} . Hence, for a given restricted activity $rv = \langle av, \{cr_E, cr_S, cr_R\} \rangle$, the set of functions used for restricted activity rv , denoted by, f_{rv} , is equal to $f_{cr_E} \cup f_{cr_S} \cup f_{cr_R}$. Thus, the set of functions used by policy, $pol = \langle RA, \varepsilon, \mathcal{I}_{pol} \rangle$, is given by $\mathcal{F}_{pol} = \bigcup_{rv \in RA} f_{rv}$. Based on these definitions of \mathcal{F}_{pol} and \mathcal{I}_{con} , the definition $\mathcal{F}_{pol} \subseteq \mathcal{I}_{con} \implies pol \bowtie^c umm$, provided in Section 3.4.1 holds.

Hierarchical relationships among contexts and policies can be similarly calculated based on policy and context structures can be calculated. Given two policies $pol_1 = \langle RA_1, \varepsilon_1, \mathcal{I}_{pol_1} \rangle$ and $pol_2 = \langle RA_2, \varepsilon_2, \mathcal{I}_{pol_2} \rangle$, policy pol_2 inherits policy pol_1 , denoted by, $pol_2 \supseteq pol_1$ if $\mathcal{I}_{pol_1} \subseteq \mathcal{I}_{pol_2}$. Similarly, given two contexts, $con_1 = \langle E_1, S_1, R_1 \rangle$, and $con_2 = \langle E_2, S_2, R_2 \rangle$, $con_2 \supseteq con_1$ if $E_1 \subseteq E_2 \wedge S_1 \subseteq S_2 \wedge R_1 \subseteq R_2$. Based on these def-

¹It is assumed here that properties do not share function names.

Chapter 3. Usage Management Framework

initions for inheritance, the interoperability results defined in Section 3.4.2 hold for this mathematical model.

This chapter provided a detailed description of the proposed usage management framework. First, a meta-model that provided the core for the framework was introduced. Following this, principles of system design were applied to develop a framework for usage management that enables interoperability, provides formal semantics for interoperability and reason about it. In this framework, focal points were identified where it is necessary to apply standards, and areas that needed to be free of standards. The internal structure of policies and contexts are left free of standards, so that they provide a design space to accommodate different types of computing domains and policy languages. Whereas, policy interface, context interface and operational semantics of the framework are standardized, and provide the glue that holds the framework together. The framework is extensible that allows creation of new policy types and context types structured by means of hierarchical relationships.

Chapter 4

DRM Game

One of the biggest challenges in usage management is enforcement of usage policies on client systems over which the resource owner has little or no control. This problem plays a critical role in DRM system to control piracy. Researchers have addressed this problem by developing trusted computing solutions where user actions on resources is controlled [6, 21]. However, the problem is that even if a small number of users are able to break the security, the content in unprotected form can be made available everywhere by means of content distribution networks and file sharing sites [35, 70]. Also trusted computing solutions are usually seen by users as too draconian and have been consistently rejected by users. Hence, the term “DRM” currently has a very negative connotation associated with draconian restrictions on how legitimate users can make use of content. This may be attributed to the fact that most DRM systems to date have in fact performed in this way. For this reason, some have suggested that more apt names for DRM are “Digital Restrictions Management” or “Digital Restrictions Malware”[33], while others have referred to the restrictions that exist in the Apple, Microsoft and Sony music platforms using a more colorful acronym [16]. In contrast, an open architectural framework for DRM should be pursued, one that allows for DRM business models and practices vastly different from what we have today, and strikes a more reasonable balance between content vendors

Chapter 4. DRM Game

and consumers. Numerous research efforts over the past few years have considered how DRM systems might be organized in order to better support desirable properties such as reusability, portability, standardization and interoperability [38, 46, 49, 63, 67, 69]. Future DRM systems must involve flexibility in decision making, and the ability to negotiate at all levels within the architecture [9, 15, 50]. In order for these and other changes to occur, however, a number of fundamental changes must occur within the architectural infrastructure of these systems. In this chapter, one means of analyzing existing and future DRM approaches is provided. The proposed approach, which makes use of game theory, allows us to take into account the strategic situations that exist at the heart of most DRM environments. With this approach in mind, a new architectural component, namely a *trust authority* is introduced, that can be used to construct a different type of DRM environment. More specifically, an environment in which it is possible to influence a consumer's actions more by rewards, and less by the punishments that vendors attempt to attach to file sharing.

This chapter starts by describing a simple baseline game in Section 4.1 that models the strategies associated with the acquisition of content. This game captures the choices that exist between a vendor and consumer for most forms of “entertainment” content. That is, for this type of content, the consumer has the choice of either purchasing the content from a vendor, or downloading it from a file sharing site. The vendor also has the choice of supplying purchased content to the consumer with or without technical protection measures. These measures are primarily copy protection technologies. It is interesting to note that it is these technologies that many now simply refer to as “DRM”, even though the full scope of DRM is much broader than copy protection. Indeed, when you investigate the colorful acronyms for DRM described in the previous paragraph, it is invariably the case that the complaints are actually with technical protection measures.

Once a baseline game for content acquisition is established in Section 4.1, it is described how this game should be played in the case of perfect DRM (actually perfect technical protection) and imperfect DRM. Specifically, the situations under which Nash

Chapter 4. DRM Game

equilibria will exist for both cases are investigated.

An important consideration that affects the utilities of both customers and vendors is how a customer uses a piece of content once he has obtained it. Thus, content acquisition game developed in Section 4.1 is treated as a subgame, and an additional subgame is developed in Section 4.2 in order to model these post content acquisition strategies. The important consideration on the part of the consumer involves whether or not to share content, and for the vendor the responses that can be made to the consumer's decision are considered. Once again, investigations on how the conditions under which an equilibrium condition can be established for this subgame are carried out.

Next, in Section 4.3, an additional level of reality to the subgames developed in the previous sections is added. Specifically, it is considered, what happens if the overall game is played repeatedly. In game theory, these repeated games aim to capture the logic behind a long-term interaction among the players, in the case presented here, consumers and vendors. In repeated games, short-term gains, long-term gains, strategies, rewards, punishments, and cooperation can be effectively studied along with their effects on achieving equilibria mutually beneficial to all players. The primary goal of the theory of repeated games is to isolate the strategies that support mutually desirable outcomes [58]. In Section 4.3, two types of repeated games are considered, one that models the current approach which is primarily focused on punishment, and the other that makes use of a trust authority in an attempt to create a win-win situation for both players.

Others have considered the use of game theory in the context of DRM. Most notably, the Secret Incentives-based Escrow System (SPIES) applies game theory to DRM systems where content protection is of interest [53]. This system is intended for applications where a secret must be protected for a limited time period and shared between two parties. The SPIES system requires the content consumer to place an amount of money in an escrow account, after which he receives access to a secret (or content). Anyone who has a copy of the secret can send a commitment to their content to the escrow service. Once a time

period has expired, anyone who has a copy of the content can receive a share of the money from the escrow account. Thus, the incentive of the legitimate possessor is not to share the secret or they will lose the security deposit. In order for this to work within a game theory setting, SPIES also requires the incorporation of charitable organizations within the framework. There is another system architecture that uses economic incentives to motivate users to keep content within a subscription community [40]. This system also makes use of an escrow authority, but in this case the escrow service pays users for sharing content with authorized users. These payments are intended to motivate users to keep content within a subscription community, but they will not have any influence on those whose receive the content outside of the subscription community. The work presented here differs in that it first attempts to use game theory to analyze existing DRM scenarios, and then extends the resulting game in order to introduce an architectural refinement aimed at building incentives into the DRM game.

4.1 The Baseline Game

A baseline version of the DRM game is depicted in Figure 4.1a. The players consist of a vendor, V , selling a content object and a potential consumer, C , who may purchase the content object. If the content object is available for downloading via an alternative (presumably illegitimate) site, then the consumer has the choice of either paying for and obtaining the content from the vendor's site, p , or downloading it from an alternative site, dl . Furthermore, when selling the content, the vendor has the choice of either providing the content with or without technical protection measures, denoted tp and \overline{tp} , respectively.

It is important to note that this game purposefully initiated with the consumer making the first decision. This allows the consumer to signal an intention to purchase content, and for the vendor to respond accordingly given any information that is available about the consumer. Notice also that the game has three possible outcomes, labeled as nodes 1–

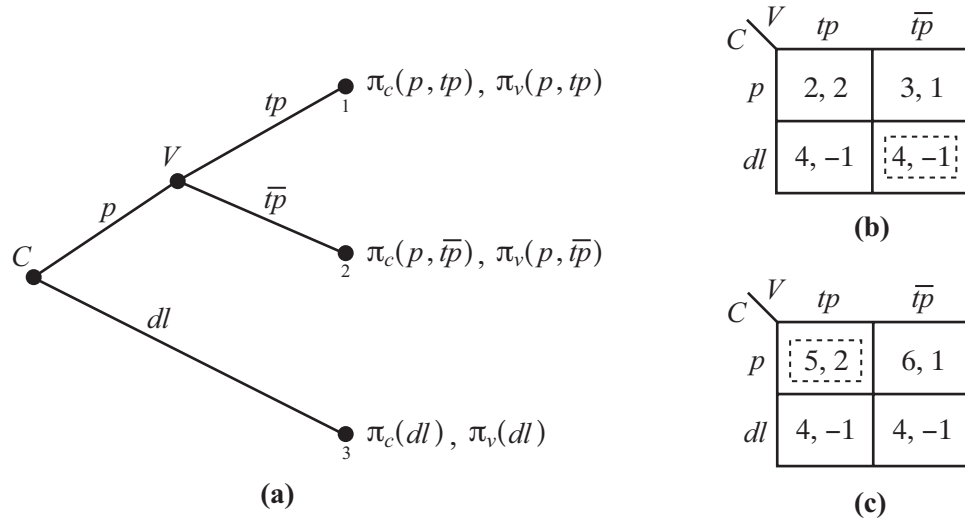


Figure 4.1: (a) An abstract model of the baseline DRM game in extensive form, along with two examples of its strategic form, (b) one in which $\pi_c(dl) \geq \pi_c(p, tp)$, and (c) the other in which $\pi_c(p, tp) \geq \pi_c(dl)$.

3 in the figure, and that the utilities the consumer and vendor receive for a particular outcome are denoted using π_c and π_v , respectively. The three outcomes correspond to (p, tp) at node 1, a consumer that pays for content delivered using technical protection by the vendor; (p, \bar{tp}) at node 2, a consumer that pays for content delivered without technical protection; and (dl) at node 3, a consumer that downloads content without paying for it. Apple's iPod/iTunes/Fairplay arrangement or Microsoft's Zune are examples of systems that attempt to reach the outcome corresponding to node 1. Apple has also modified iTunes in a way that allows it to reach node 2.¹ There are numerous file sharing sites that allow consumers to reach the outcome corresponding to node 3.

Although the game shown in Figure 4.1a is extremely simple, ignoring entities such as content aggregators, content producers, secondary markets, etc., it serves to motivate the reasons behind the current strategies used by DRM vendors, and also leads to more realistic

¹Starting in May 2007, songs from the EMI Group distributed via Apple iTunes have been delivered without FairPlay DRM. These files cost more, but also have a higher audio quality, with an increase in bit-rate from 128 kbit/s to 256 kbit/s.

DRM game scenarios that point to the need for new DRM architectural infrastructures. To see why, consider the following two cases.

4.1.1 Perfect DRM

Let us start by making some general assumptions about the utilities associated with the three outcomes in Figure 4.1a. If moral issues associated with file sharing are put aside, and it is assumed that there is no difference between the content obtained from downloading or through purchase, then the consumer's utilities will generally satisfy $\pi_c(dl) \geq \pi_c(p, \overline{tp}) \geq \pi_c(p, tp)$. That is, C would prefer to obtain the content at no cost via downloading rather than purchasing it, and if purchasing the content, C prefers no technical protection to technical protection.

Now, assume at time $t = \tau$ a content item is first made available from a vendor in electronic form. At this point consumer C has no choice of downloading the content for free. That is, since the download option is not available, the only decision on the part of player C is whether or not to purchase the content. The game collapses to a very simple situation that depends only on the utilities associated with purchasing the content versus not purchasing it, and V may adjust the price of the content in order to influence these utilities. If a price can be found where both C and V receive sufficient utility, a sale may occur, depending upon one other factor. Specifically, the decision of whether or not to use technical protection will also influence the players' utilities. If the technical protection measures associated with DRM are perfect, circumvention of these measures is not possible. In this case, if the utility of C is not reduced too much by the technical protection measures, a sale may still occur. Furthermore, if V does *not* offer the \overline{tp} option, subsequent plays of this game at times $t > \tau$ will be identical to the game played at time $t = \tau$. In this case, in every instance of the game going forward, the vendor is in complete control, with the desirability of the content offered with technical protection, along with

Chapter 4. DRM Game

its price, being the only determining factors.

The only reason to bring this situation up is that it seems to correspond to what many vendors were striving for in the early days of DRM. In those days, it was not uncommon for suppliers of DRM solutions to make claims that their technology was “100% copy-proof”. If these claims were indeed true, then the logical thing for V to do would be to use such a DRM system in order to enforce the game described above. That is, if the choice of tp meant that no copying could occur, then V should never choose \overline{tp} , thereby ensuring that the dl option would never become available to C in subsequent games. It seems that many of the early adopters of DRM technologies were in fact striving for this ideal.

There are numerous reasons why perfect DRM is generally not possible. First, every widely-used technical protection measure that has ever been built has also been circumvented in a manner that is easily transferable to others, and there is no reason to believe that this situation will ever change.² Second, in many cases the same content is offered through multiple channels, one of which may be more susceptible to “misuse”. For instance, in the case of music, CDs are generally not protected. Thus, even if online music could be provided in the manner of the game described above, it would still be possible to obtain the content via file sharing if the same music was released on CD. Using the popular terminology, the former will be referred to as “cracking”, and the latter as “ripping”.

In actuality, C has a third choice, not shown in Figure 4.1a, of not purchasing the content. We have not shown this choice because the focus of the remainder of the chapter is on those factors that influence consumers’ decisions once they have already decided to obtain a particular content object. Furthermore, after V ’s decision regarding technical protection, C may choose to download content. In the game, this situation is treated as if C had initially decided to download the content. Since a game of perfect information is assumed, this makes sense.

²In spite of this, it is still possible to find DRM vendors claiming to have infallible technical protection technology.

In summary, with this case the situation is quite simple, and the content vendor has a clearly understood strategy for maximizing utility. It is assumed that many of the DRM systems that have been created by those in the content industry to date have this model as the underlying ideal. In other words, although most of these vendors recognize that technical protection cannot be perfect, they strive for ever stronger technical protections, as well as laws such as the DMCA [30] that criminalize the circumvention of technical protections, in the hopes that some close approximation to this game can be achieved. Below it is describe why this is likely a losing proposition in the long run.

4.1.2 Imperfect DRM

Now let us consider the case where DRM is not perfect, and content may “leak” through the technical protections that are in place via cracking or enter into file sharing sites via ripping. This game more realistically models the current situation with DRM systems.

The vendor’s preferences, under the assumption that technical protection measures will reduce but not eliminate file sharing, are generally given by $\pi_v(p, tp) \geq \pi(p, \overline{tp}) \geq \pi_v(dl)$. That is, they are diametrically opposed to those of C described previously. Specifically, C prefers \overline{tp} over tp , and may in fact be willing to pay more for the \overline{tp} option, as in the Apple/EMI model described previously. However, because future sales may be negatively impacted by the \overline{tp} option, particularly if it leads to large-scale file sharing, V generally prefers tp over \overline{tp} .

Let us consider a game that is played with the aforementioned preferences in mind, along with the imperfect technical protection assumption. An example game in strategic form is given in Figure 4.2b. This figure shows a payoff matrix that satisfies the preferences, where it is assumed the content is initially available from both a downloading site, as well as from an online vendor. The first element in any pair shown in the matrix corresponds to C ’s utility, and the second entry to V ’s utility. For example, in Figure 4.1b, if

Chapter 4. DRM Game

the outcome of the game is (p, \overline{tp}) , then C 's utility would be 3, and V 's would be 1. Notice that in the case of dl , V has a negative utility that corresponds to a loss of sales. Once again, moral considerations are put aside, and the consumer has equal ease in obtaining the content through either downloading or purchase, then no matter how player V sets the price (assuming nonnegativity) he will not be able to make $\pi_c(dl) \geq \pi_c(p, tp)$. I.e., under these conditions the consumer has no incentive to purchase the content, and will instead download it.

The critical relationship in terms of establishing a Nash equilibrium is between $\pi_c(dl)$ and $\pi_c(p, tp)$. If $\pi_c(dl) \geq \pi_c(p, tp)$, then a Nash equilibrium exists for the case of C downloading rather than purchasing content, denoted by the dashed box in Figure 4.1b. Specifically, once C plays the strategy dl , V 's strategy becomes irrelevant. Therefore, the solution or equilibrium for the game will involve C downloading content, and as long as the aforementioned preference relationship holds, no change in strategy by either C or V can cause the game to move to a solution that is more profitable to either player. Thus, if a vendor endeavors to create a situation of perfect DRM, but fails, consumers will be driven towards downloading if the downloaded and purchased content are equally desirable to C .

The only way to move the game from the Nash equilibrium described above is to create a situation where $\pi_c(p, tp) \geq \pi_c(dl)$. If this can be accomplished, then a Nash equilibrium will be established for the (p, tp) outcome, which corresponds to C paying for content delivered with technical protection. This situation is depicted in Figure 4.1c, and no change in strategy by either C or V can cause the game to move to a solution that is more profitable to either player. Notice that even though C would obtain a higher payoff from the (p, \overline{tp}) outcome, the solution of the game will not move there because the decision between tp and \overline{tp} is not C 's.

Thus, the issue becomes one of establishing business models and other policies that can create a situation whereby $\pi_c(p, tp) \geq \pi_c(dl)$. We will also see that $\pi_c(p, \overline{tp}) \geq \pi_c(dl)$ may provide a suitable solution, if V 's utility can be made large enough. On the surface,

Chapter 4. DRM Game

since one strategy involves C receiving content for free, and the other involves C paying for it, it seems that either of these preference relationship would be difficult to create. There are, however, a number of ways to create situations whereby $\pi_c(p, tp) \geq \pi_c(dl)$ or $\pi_c(p, \overline{tp}) \geq \pi_c(dl)$. These must include strategies for lowering utility $\pi_c(dl)$, and this in fact seems to be the primary focus of vendor activities. In particular, the two main ways of reducing player C 's utility involve *legal* attacks and *technological* attacks.

With legal attacks, the goal is to prosecute illegal file sharing, and then widely publicize any successes in this area. This, in effect, reduces $\pi_c(dl)$, but only for consumers who believe they could be subject to similar prosecution. There are a variety of technological attacks that can be deployed. One method, called *poisoning* involves modifying content so as to make it less desirable to C [24]. E.g., this may involve adding high amplitude white noise to a music file, or removing large chunks of scenes from a movie file. These files are then uploaded to file sharing sites. Thus, by making it more difficult for C to obtain “good” content, $\pi_c(dl)$ is effectively lowered. Another technological attack involves inserting malware or forms of malicious software into the clients used for downloading. All of the forms of attack described above are in widespread use. Simulations indicate that poisoning is one of the most effective techniques for reducing $\pi_c(dl)$ [29]. These attacks are ingenious in that they take the strength of peer-to-peer networks, namely their power-law connectivity, and turn it against them. In other words, poisoned content can be made to spread nearly as rapidly as good content in these environments.

It seems that less effort has been applied towards methods that increase the utilities associated with either $\pi_c(p, tp)$ or $\pi_c(p, \overline{tp})$. This can be accomplished by somehow adding value to purchased content relative to downloaded content. E.g., the reliability and ease with which content can be obtained from Apple's iTunes certainly adds value to the content, and has contributed to the success of this system. In Section 4.3 another means of achieving this result though the use of a trust authority is considered.

For either of the situations just described, there are in fact numerous additional deci-

sions on the part of both the consumer C and the vendor V that must be made in order to implement specific strategies. In the next section these are cast in a game-theoretic setting that will allow them to be attached to the baseline game described in this section.

4.2 Important Subgames

We now describe three important subgames, g_1 – g_3 , that can be played starting from each of the three outcomes identified in Figure 4.1a. That is, each of these subgames corresponds to strategies that can be played post content acquisition. These subgames are concerned with how C uses content, and how V responds to this. The first subgame, g_1 , corresponds to the game played after content has been purchased with technical protection; the second subgame, g_2 , corresponds to the game played after content has been purchased without technical protection; and the third subgame, g_3 , corresponds to the game played after content is obtained via file sharing.

In each subgame, since C has obtained the content, he now has a choice of either sharing the content, strategy s , or not sharing it, strategy \bar{s} , as shown in Figure 4.2. Next, V has the decision of whether or not to take some actions based on this file sharing (or lack thereof). These are denoted by a and \bar{a} , respectively, in Figure 4.2. The types of actions that can be taken will depend upon the subgame being played, and will be discussed in more detail shortly.

In order to justify the importance of this model, let us point out how various nodes in this game correspond to specific outcomes that currently exist in the marketplace. We will also comment on how certain strategies that exist in this game have not yet been exploited. Finally, it will be described how the actions taken at the last step of the game by the vendor may influence the utilities associated with various nodes that occur earlier in the game.

First, consider the outcome $(1, s, a)$, which corresponds to a consumer paying for con-

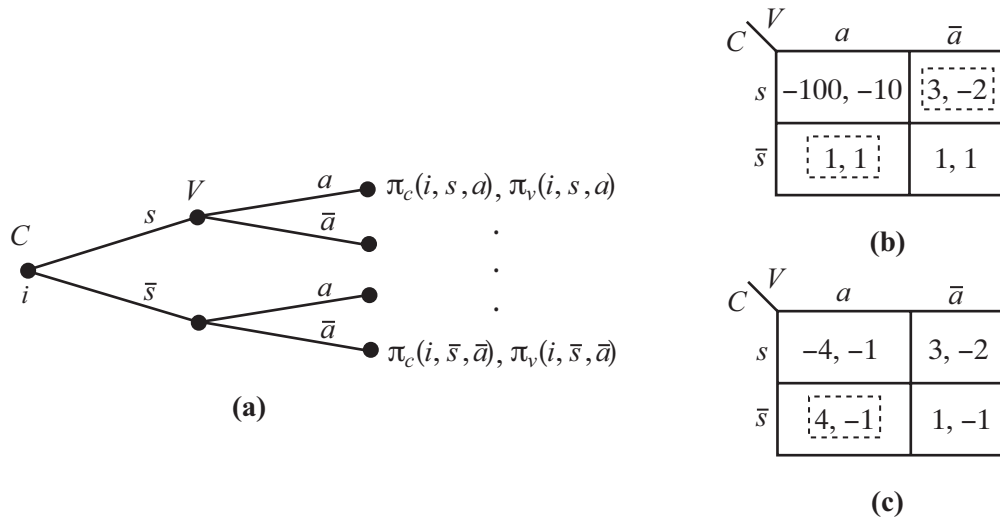


Figure 4.2: (a) A general model in extensive form of the post-content-acquisition subgame that flows from the outcomes of the game shown in Figure 4.1a, where node i can assume the values 1–3. (b) An example of this subgame in strategic form that models RIAA efforts, and (c) a model that incorporates rewards.

tent that is technically protected, then subsequently sharing this content, and finally, the vendor taking some action. Since the content was supplied with technical protection, it must be the case that it was cracked by C , and therefore, since provisions such as the DMCA may apply, the action taken by V may be a legal attack.

Another situation to consider occurs with respect to the outcomes $(3, \bar{s})$ and $(3, s)$. The former corresponds to what are commonly referred to as “freeloaders” or “leeches”, consumers that download but do not share content. It is interesting to note that most file sharing sites have constructed various mechanisms that penalize this outcome. Thus, they are trying to create a situation whereby C ’s utility is increased by following $(3, s)$ rather than $(3, \bar{s})$. Nevertheless, historically a large percentage of downloading corresponds to consumers that will follow the freeloading outcome [5]. It is also important to recognize that to date, the Recording Industry Association of America (RIAA) has only prosecuted those outcomes that involve sharing. That is, the main risk associated with the use of file sharing sites involves sharing and not downloading, which may decrease the utility of the

Chapter 4. DRM Game

$(3, s)$ outcome for some consumers, namely those that believe they could be subject to RIAA actions.

Let us consider a concrete example of a vendor that is seeking to penalize file sharing behavior that emanates from node 3, and where specific payoffs are provided to the players. This, for example, models the action that the RIAA has taken against file sharing on college campuses [72]. In this case, if V takes action, the penalty applied is very severe, e.g., college students were told that a typical settlement with the RIAA is \$3,000. In Figure 4.2b, this is modeled as a payoff of -100 for C if V takes action, and 3 if V does not. In general, since this is a severe penalty, it must be that case that $\pi_c(3, s, a) \ll \pi_c(3, s, \bar{a})$; however, the legal costs to V associated with pursuing this action are also much greater than the loss associated with this one instance of file sharing. That is, it is expected that $\pi_v(3, s, a) \ll \pi_v(3, s, \bar{a})$. Thus, it is possible to think of this as the “nuclear option”, as it is very effective when applied, but V is loathe to use it. Notice in Figure 4.2b that C receives the highest payoff for sharing when no action is taken, and a smaller payoff for not sharing, i.e., for freeloading. As discussed above, file sharing sites attempt to reward the former, and penalize the latter. Finally, since the RIAA does not take action if file sharing does not occur, the payoffs associated with those outcomes are 1 for both C and V .

Given the game shown in Figure 4.2b, two pure strategy Nash equilibria exist, one for the case of not sharing when action is taken, and one for the case of sharing when no action is taken. Thinking about these equilibria from the perspective of the extensive form game, it can be seen that an action is taken by V only after the decision by C of whether or not to share. Because C knows this, he can reason as follows: “If I play \bar{s} , then it does not matter to me what V plays; however, if I play s , then I know that V will play \bar{a} . Therefore since I receive a higher payoff in the latter situation, I will share the file”. What this means is that only the Nash equilibrium at (s, \bar{a}) is sequentially rational, and therefore it will end up being the solution of the game.

The problem with the game in Figure 4.2b is that since it only involves pure strategies,

Chapter 4. DRM Game

C is certain about what action V will take. In order to make this game more realistic, uncertainties are introduced in V 's actions through a lottery. That is, in the mixed strategy case, V may select to play strategy a with some probability. To see how this affects the play, let us assume that C plays s for sure, but that V plays a with probability q , and \bar{a} with probability $1 - q$. In this case, the expected payoff to C is given by:

$$\mathbb{E}[\pi_c] = (-100q + 3(1 - q))$$

and it is possible to make $\mathbb{E}[\pi_c] < 0$ by choosing $q > 4/103$. In other words, if V takes action only 4% of the time, he can make C 's expected utility negative.

We should not expect C to remain idle in the case that V is playing a mixed strategy. Thus, let us consider the case when C plays a mixed strategy that involves sharing with probability p . Assuming that V still only wishes to take action with probability q , what level of penalty P should V apply in these cases in order to ensure that C 's expected utility is negative? In this case

$$\mathbb{E}[\pi_c] = p [-(P + 3)q + 3] + (1 - p).$$

Thus, if C shared content 50% of the time, and V wanted to keep his frequency of action at 4%, then

$$\mathbb{E}[\pi_c] = 0.5 [-(P + 3)0.04 + 3] + 0.5$$

and now in order to ensure that $\mathbb{E}[\pi_c] \leq 0$, a penalty of at least $P = 97$ must be applied whenever action is taken, rather than $P = 100$ as it was in the case when C always chose s . Thus, it can be seen that C playing a mixed strategy has very little effect on the game. It is also easy to see that if V were able to increase the penalty to say $P = 1,000$, then he would only need to take action 0.4% of the time in order to make C 's expected utility negative. Note, however, that there may be some maximum penalty that V is able to apply—this would happen for example if the legal system sets a threshold on the amount that V can penalize C . This appears to provide a good model for the behavior of the RIAA with

Chapter 4. DRM Game

regards to file sharing. That is, they rarely take action against file sharing, but when they do, it is severe, and with as much publicity as possible.

Finally, consider a game in which V seeks to reward good behavior, rather than focusing all of his attention on punishing bad behavior. Furthermore, this game attempts to take fair use into account. Specifically, consider a game that emanates from node 2, when purchased content has been supplied without technical protection. We assume that a certain amount of file sharing will take place, but that this may take place within the bounds of fair use. Determining acceptable fair use has always been a gray area, often requiring a judge's opinion to settle specific cases. One of the problems with DRM solutions to date is that it does not allow for these gray areas [32]. Given this, consider the following scenario. A musical artist uploads a song she has just produced, to an online music vending site with which she has a contractual agreement. In addition, she provides the terms under which she would like the song sold (or perhaps these are part of the previously established contract). In this case, she specifies that she would like to offer the song, along with certain bonus materials, to all members of the site that have established a given level of trust, at a price this is specified as A . In addition, for those members of the site that have a lower trust rating, she would like to offer the song alone at a higher price, call it B . In fact, she may establish additional offerings depending upon the characteristics of the users she would like to service. Indeed, for users that do not rise above a certain trust threshold, she may deny the ability to purchase the content in digital form, or only allow for streaming but not downloading rights. Users of this vendor's site have the option of registering, and the choice revealing certain information that will be used to establish their ID at the site. Songs delivered to the users are fingerprinted with their ID, and a third party trust authority monitors file sharing sites in order establish a trust measure for the user. For example, the trust authority may classify a user as having low, medium, or high trust, depending upon their file sharing habits.³ Furthermore, fair use doctrines can be incorporated into

³This is quite similar to how a credit agency establishes a credit rating for a person depending upon their payment habits.

this ranking. If an attempted transaction at the site meets the criteria established by the artist, the song may be downloaded, and the user is free to play it on any device.

Notice that in the situation just described, there are no traditional DRM content restrictions. Once users have obtained content, they can use it however they wish, knowing that the manner in which they use and share the content will have a direct bearing on how they will be trusted in the future. Notice also that the essential component in this DRM scenario, the trust authority, exists as part of the infrastructure, possibly as a middleware service. Finally, because of quantitative measures of trust, it becomes possible for economists to relate behavior to price, and attempt to solve for the optimal values of A , B , etc. A related work, in that it does not require new technology on the consumer side, analyzes the optimal design of flexibility in the presence of unlicensed distribution channels, and in particular, how adding value to licensed content affects equilibrium policies [15]. It is also interesting to note that researchers at Philips Research are working on consumer-side hardware that modifies access according to user behavior [50]. This technology could also make use of trust authority middleware.

It has been said that we should not expect DRM to ever be smart enough to distinguish fair use from copyright infringement [32]. Note, however, that in the scenario described above, DRM is not expected to do this. Rather, the DRM technology must only collect information about how content is being used.⁴ This information can then be used, by a judge for example, to determine if fair use doctrines have been violated, thereby putting us back to a more balanced situation with respect to fair use that is similar to what existed prior to the Internet.

In any event, let us cast the above situations that make use of a trust authority into the game-theoretic setting of Figure 4.2a. Specifically, assume that according to the rating

⁴It is assumed that use of fingerprints embedded as watermarks in a manner similar to what those in the cryptography community refer to as *traitor tracing*. This approach is susceptible to collusion attacks [56], but just like other types of cryptography, it does not have to be perfect in order to be useful.

provided by the trust authority, V supplies a content object to C , and that the quality of this content object (e.g., its resolution, the amount of bonus materials, etc.) is determined by this rating. In this case, a reward is tied to the \bar{s} action, as well as a punishment to the s action. An example is shown in Figure 4.2c. In this example, the reward in the case of the (\bar{s}, a) strategy is not significantly different from the game of Figure 4.2b; however, the punishment associated with the strategy (s, a) is far different. Specifically, in the case of (s, a) , the punishment is now much smaller, but so is the cost to V . This is due to the fact that in this case, C is receiving lower-value content (a penalty not nearly as severe as getting sued by the RIAA), and it costs V very little in order to pursue this action (certainly far less than the legal fees associated with a law suit). Now, even if the cost of taking no action remains the same, relative to the game in Figure 4.2b, the solution is far different. Specifically, for the pure strategy case, a Nash equilibrium now exists for the strategy (\bar{s}, a) , which corresponds to a win-win situation for both the consumer and vendor.

It is certainly possible for both players to play mixed strategies in this game as well, but assume these are implemented so that they do not change the fundamental win-win nature of this game. In this case, the more important question is how current decisions on the part of both C and V will affect their utilities in the future. That is, with this game, if C decides to share too much content, this will negatively impact his utility in the future, and a similar situation arises if V does not provide sufficient rewards for good behavior. The game shown in Figure 4.2c does not adequately capture this aspect unless the game is repeated. This is the focus of the next section.

4.3 Repeated Games

Repeatedly playing the DRM game shown in Figure 4.3, opens up the possibility of changing the players' future behaviors. That is, the notion of reciprocity is introduced into the game, whereby one player's decision at one point in time can affect the decision made by

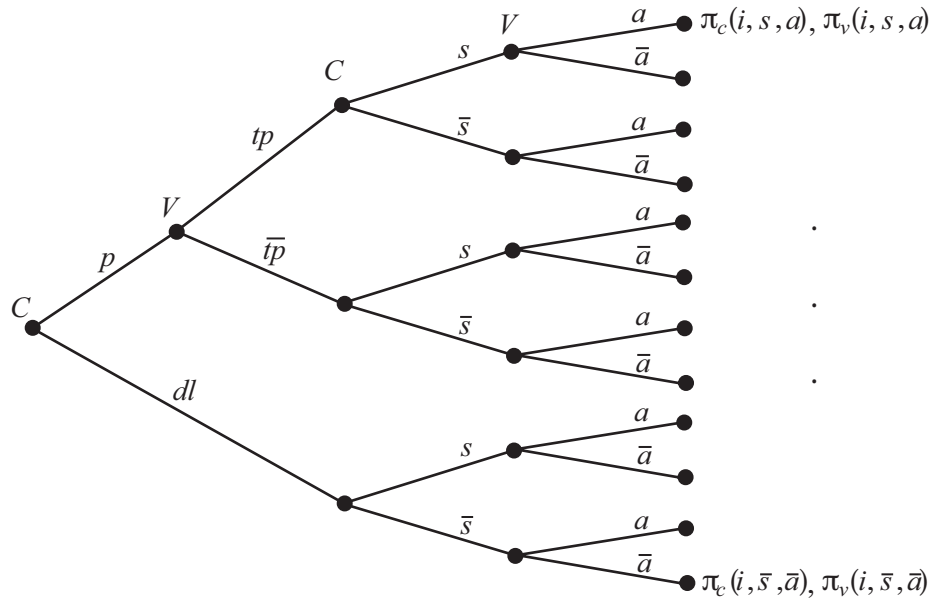


Figure 4.3: The complete DRM constituent game that consists of the baseline content acquisition subgame of Figure 4.1, followed by the post content acquisition subgames of Figure 4.2.

the other player at a future point in time. A player may be willing to sacrifice short-term gains if he is convinced that by doing so he will be provided with a reciprocal reward in the future. There are two important forms of reciprocity that can be studied with respect to the DRM game, that of punishing “bad” behavior, and that of rewarding “good” behavior. It seems that to date, vendors have focused primarily on the former. In this section an infrastructure that would allow them to also exploit the latter is described.

First, however, it should be noted that it is possible to view good and bad behavior from the perspective of either player. For instance, in the early days of electronic music distribution, consumers punished vendors for offering high-priced music accompanied with cumbersome technical protection. The punishment in this case was large-scale downloading. Vendors responded with punishments of their own—primarily the legal and technological attacks already described in Section 4.2. In addition, vendors have sought more aggressive technical protection measures. Recently, for example, consumers have viewed

Chapter 4. DRM Game

a SONY BMG technical protection measure that ends up installing a root kit as bad behavior, leading to a large amount of negative publicity for the vendor, and a subsequent recall of media containing the technology. Indeed, these types of incidents are one of the primary reasons behind the development of the negative view towards DRM that was discussed in Section ??.

In order to create a repeated game out of the stage game shown in Figure 4.3, it is assumed that at each terminal node in the constituent game, a new game is played with that terminal node as the root. This leads to the next iteration of the repeated game. There are two ways of thinking about this repeated game. By fixing the content object, and applying a mixed strategy to C 's decisions, it is possible to model how a population of customers might treat a single content object. In this case, each iteration of the constituent game corresponds to the interactions between a single customer, whose average behavior is captured by the mixed strategy, and the vendor. The other situation that can be modeled involves fixing the customer, and allowing the content to vary at each stage of the game. Each type of repeated game is explained below in more detail.

4.3.1 Content Spreading

For the repeated game in which the content is held constant, the game is focused on how this content is acquired by each customer. In other words, how the content spreads among the customers, via downloading and purchase, and how the vendor's actions can influence this spreading is analyzed. It is the goal of V to repeat the game in the upper part of the Figure 4.3 constituent game tree as much as possible. That is, V would like to play the subgames g_1 and g_2 as much as possible, as these result from sales; however, in both of these subgames there is an s option that will directly affect the probability of dl in subsequent games.

In order to bias these games towards g_1 , g_2 , and the \bar{s} decision, V can play tp . An anal-

Chapter 4. DRM Game

ysis of this in the context of content acquisition was provided in Section 4.1. Now reality is added as to how C will use this content, and what V 's response will be. One strategy that resembles the RIAA's actions involves V tolerating a certain amount of sharing, and then taking legal action against C for a short period of time when this amount is exceeded, followed again by a toleration of sharing. In the game theory literature, this strategy, where a deviation from desired behavior is met with a punishment phase, after which all is forgiven, is called a *forgiving trigger* [31]. In reality, V does not know exactly how many rounds of the game have been played, as V does not know with certainty how much file sharing is taking place. Therefore, this parameter must be estimated from available data, e.g., from observed file sharing or from measured popularity of a content object versus its actual sales. Time also plays an important role in this repeated game, as vendors are usually more aggressive in protecting new releases, and consumers are also more interested in acquiring them. This can be accounted for by applying a discount factor, $0 < \delta \leq 1$, at each stage of the game.

If it is assumed that V receives a profit of a from each play of the constituent game if C chooses p , then after n stages of this game, where C always plays p , V would receive a total discounted utility of $\frac{an(\delta^n + \delta)}{2}$. Thus, If V monitors this utility, a punishment (legal attack) corresponding to one time period can be triggered whenever some fraction α of the expected utility for V is not met. If it is assumed that the penalty has a cost to V of $-P$, then V 's utility for one no punishment/punishment cycle is

$$\frac{\alpha an(\delta^n + \delta)}{2} - P.$$

The folk theorems associated with infinitely repeated games tell us that a subgame perfect equilibrium can be established for this behavior cycle only if the above quantity can be made positive [58]. The difficulty in this approach is that V has little control over the first term in the equation. It is easy to conceive of situations where the first term never exceeds P , and the cost of taking legal action is never warranted.

There are a number of additional strategies that can be investigate with respect to this

repeated game. For instance, it is more realistic to assume that V would also apply punishment in the form of technological attacks. Note that the trigger for a technological attack can include either the amount of sharing and/or the lack of sales; a legal attack, however, can only be applied in response to illicit sharing. We might also consider more elaborate strategies that involve offering a mix of content with and without technical protection, along with triggers for determining when punishment should be applied. All of these strategies, however, share the fundamental property that they mainly focus on punishment side of the equation in attempting to modify the future behavior of a population of customers.

4.3.2 Customer Influence

The high-level strategy associated with the vendor's actions in the previous repeated game can be summarized as “no bad deed goes unpunished”. That is, the primary focus is on finding ways to punish customers for bad behavior. We now consider the general strategies that can be associated with the “no good deed goes unrewarded” approach. With current DRM infrastructures, it is difficult to reward good behavior, making it less likely that this type of behavior will be seen.

Consider again the constituent game shown in Figure 4.3, but now assume the customer C is held constant on every iteration of the repeated game, and that the content is allowed to vary. In this case, the actions taken by V can directly influence a particular customer. In the repeated game of Section 4.3.1, legal action is also generally taken against individual customers; however, the legal attack is actually meant to influence an entire population of customers, primarily those not directly subjected to the legal action. In the current game, it is considered how V can respond individually to each customer in the customer population. As described in Section 4.2, this will be accomplished using a trust authority that allows for the possibility of rewarding good behavior. In the end, we have a DRM

Chapter 4. DRM Game

environment that provides the vendor with strategies that are finer grained, allowing them to create much greater distinctions between the customers they serve.

The folk theorems for infinitely repeated games tell us that it is possible to create an equilibrium around any behavior cycle in the repeated game as long as each player's payoff within the cycle is strictly positive. In the repeated game of Section 4.3.1 this was difficult to guarantee due to the large expense associated with punishment. In this case, however, assuming that it is possible to easily assign a trust rating to each customer, it becomes much easier to create a desired equilibrium. Furthermore, the behavior cycles and associated triggers can be much more sophisticated, depending upon how many customer categories the vendor may want to differentiate between. Note that in the game of Section 4.3.1, V only acted based upon two categories of customers, those that shared, and those that did not.

Finally, it should be noted that the repeated game described in this section and in Section 4.3.1 are not mutually exclusive. In reality, it makes sense for V to play some combination of these games. For example, if C is playing one of subgames g_1 or g_2 shown in Figure 4.3, then V has the opportunity to reward the play of \bar{s} . If, however, C has played s in any of subgames $g_1 - g_3$, then a punishment may be the most effective course of action for influencing the play of later games.

4.4 Conclusions

In this chapter an increasingly detailed version of the DRM game is considered—a game meant to model the strategies associated with various DRM approaches. This was broken down into two subgames, one associated with content acquisition, and the second that dealt with decisions post content acquisition. Subsequently, these two subgames were put together in order to create a constituent game that was then repeated in order to model two

Chapter 4. DRM Game

important situations. The first considered an approach that is prevalent today, with a focus on punishing file sharing. The difficulty of achieving an equilibrium in this game was addressed. Next, an approach that involved rewarding customers for purchasing and not sharing content was presented, along with a discussion of how much easier it would be to establish an equilibrium in such a game. One of the prime components in this latter game was a trust authority middleware infrastructure that was used in order to rate the behavior of customers in this game, and reward them accordingly. The ability to implement such a trust authority is by no means a solved problem. Ongoing research is considering how such an infrastructure can be constructed while at the same time addressing security and privacy concerns. In addition, future research should address more specific strategies associated with the repeated games presented in this chapter.

Chapter 5

Applications

5.1 Multi-level Security

Multi-level security (MLS) environments implemented in security establishments have focussed on access control systems to ensure that resources are accessed by subjects in accordance to the access control policies expressed in terms of security clearances and security classifications. However, recent developments in technologies that enable users to easily collect, integrate and display data from different sources over the Internet, such as Yahoo Pipes, has led to increased interest in enabling such features in MLS systems.

Secure mashups, or smashups, are an ideal case of usage management of resources once they have been granted access to the user. Smashups lead to a number of interesting use cases where not only the act of resource aggregation needs to be controlled, but new content is generated that is governed by a combined set of rules that reflect the usage rules of the constituent resources in the mashed content. In this application, the usage management framework is overlaid on a MLS environment to enable mashups.

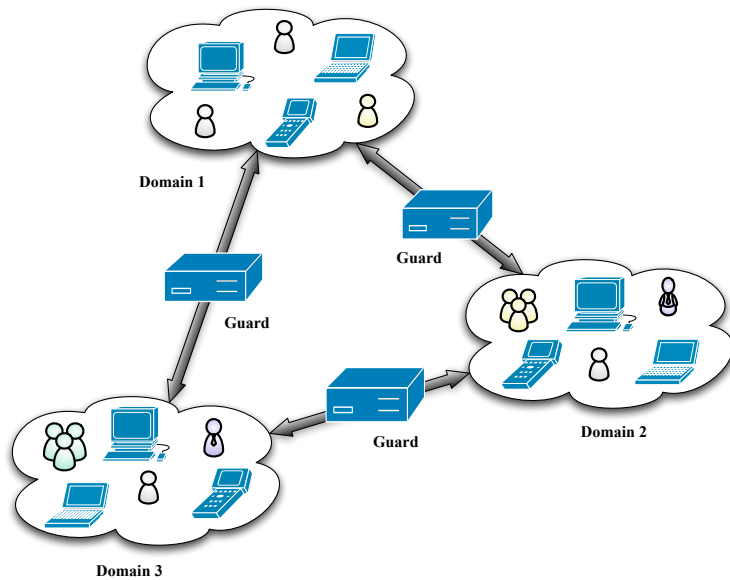


Figure 5.1: A multi-level security environment.

5.1.1 MLS Overview

A multiple domain, distributed MLS system is shown in 5.1. It consists of multiple domains, and each domain consists of users with multiple devices. The information or data in the MLS is classified according to the classification levels, and personnel in the domains are divided according to the security clearance levels. There are *guards* that operate in between two domains that control the flow of information based on the MAC policies. However, usage management that is considered here is resource specific, and focuses on policies that are resource specific.

5.1.2 Context Modeling

The context for the MLS domain contains three entities, namely, Subject, Environment, and Dynamic Environment. The properties of each of these entities, their values and functions supported by the properties are shown in Table 5.1. The Environment entity captures

Table 5.1: Context for multi-level security.

Multi-level Security Context			
Entity	Property	Values	Functions
Environment (E)	Domain	{ABNet, EXNet, TELNet}	Equality
	OS	{Windows, OSX, Linux, Andriod}	Equality
	Device	{Blackberry, iPhone, Desktop, Laptop}	Equality
Subject (S)	Security Clearance	{Top Secret, Secret, Confidential, Restricted}	Comparable
	Role	{Alpha, Beta, Gamma, Delta}	Equality
	Project	{Zen, Yuma, Om}	Equality
Mashup Environment (ME)	Basket	$2^{Set\ of\ Resources}$	{in, not_in}

the properties of the computing environment, the Subject entity captures properties of the subjects in the system, and the Mashup Environment captures the properties of mashup. While Environment and Subject entities are self explanatory, the Mashup Environment needs explanation of the mashup process.

In the mashup process, a user is provided a list of resources that the user can access, along with a mashup area, where the user can drag and combine resources of interest. The Mashup Environment has the Basket property which maintains the list of all the resources that the user has included in the mashup area.

There are three types of functions, namely, Equality, Comparable and Set. Equality type provides functions $==$ and \neq . Comparable type provides functions $==$, \neq , $<$, $>$, \geq , \leq and *between*. The Set type functions are shown are provided to the Basket property. The function *in* returns true if the supplied resource is in the mashup basket. Similarly, function *not_in* returns true if the supplied resource is not in the mashup basket. The rest of the functions are self explanatory.

Every resource in the system is identified with a unique identifier and security classification of the resource. Table 5.2 shows resource properties. Since policies are resource specific, resource is not included as a part of the context.

Table 5.2: Resource table

Resource ID	Name	Description	Security Classification
1	Maps1	Low resolution maps	Unclassified
2	Maps2	High resolution maps	Restricted
3	Weather	Weather data	Unclassified
4	Routes	Routes data	Restricted
5	Troops	Troops locations	Secret
6	Weapons	Weapons locations	Secret
7	Millnt	Military intelligence	Top Secret
8	NavInt	Navy intelligence	Top Secret

Table 5.3: Resource-Policy table

Resource	Domain	OS	Device	Sec. Clear.	Role	Project	Basket
Maps 1	All	All	All	All	All	All	All
Maps 2	ABNet TELNet	All	Desktop Laptop	All	Alpha Beta Gamma	Zen Yuma	All
Weather	All	All	All	All	All	All	All
Routes	TELNet EXNet	All	All	All	Alpha Beta Gamma	Zen Yuma	All
Troops	ExNet	OSX Linux	Desktop Laptop iPhone	\geq Secret	Gamma Delta	Om Yuma	<i>not_in</i> Weapons
Weapons	ExNet TELNet	OSX Linux	Desktop Laptop	\geq Secret	Alpha	Om Yuma	<i>not_in</i> Troops
Millnt	ExNet	OSX	Desktop	\geq Top Secret	Gamma	Yuma	<i>not_in</i> NavInt
Millnt	ExNet	OSX	Desktop	\geq Top Secret	Gamma	Yuma	<i>not_in</i> NavInt

5.1.3 Policy Specification

Each of the resources can now be associated with a usage policy for access and subsequent mashup. In this discussion, a single policy example to show how the mashup rules work is provided. The policies for mashup for all resources are shown in Table 5.3. The rule that resources *Troops* and *Weapons* cannot be mashed together is conveyed in the last column *Basket* entries for these resources respectively

The above mentioned policies can be expressed in terms of a domain specific language (DSL) shown below. A DSL description is transformed into an executable policy object.

Chapter 5. Applications

For example, the expression of policy for the *Weapons* resource is described below.

// Define mashup activity.

a1 = **activity** (:mashup)

// Define constraints.

c1 = **constraint do**

 ((Domain == "EXNet") \vee (Domain == "TELNet")) \wedge

 ((OS == "OSX") \vee (OS == "Linux")) \wedge

 ((Device == "Desktop") \vee (Device == "Laptop")) \wedge

 (Security_Clearance \geq "Secret") \wedge

 (Role == "Gamma") \wedge

 (Project == "Yuma") \wedge

 (Basket not_in "Troops")

end

// Restrict mashup activity with the constraints.

ra1 = **restrict** *a1* **do**

with *c1*

end

// Define permissions with restricted mashup activity.

weapons_policy = **policy** *ra1* **do**

policy_evaluators :standard

constraint_evaluators :predicate

permit *ra1*

end

Table 5.4: Policy table for mashups

Resource	Domain	OS	Device	Sec. Clear.	Role	Project	Basket
Troops & Routes	ExNet	OSX Linux	Desktop Laptop iPhone	\geq Secret	Gamma	Yuma	<i>not.in</i> Weapons
Weapons & Maps 2	TELNet	OSX Linux	Desktop Laptop	\geq Secret	Alpha	Yuma	<i>not.in</i> Troops

During the mashup process, when two resources are mashed up to create a new resource, the policies of the individual resources are logically combined to create the policy of the new resource. Table 5.4, shows policies for mashed up resources.

5.1.4 System Operation

The working of the mashup program is shown in Figure 5.2. The mashup program provides a user with the list of resources that are available for mashup. The program refreshes the list available for mashup whenever the user adds a resource in the mashup space (i.e. the Basket). The list is generated by checking the policies of all the resources against the policies of the resources currently in the mashup space. The check is performed by requesting the policy server that maintains the resource tables and the respective policies for each resource. The policy server can store policies in either their descriptive form or as executable policy objects. The server provides a web interface to query for policy validation. To validate a policy the mashup program provides the current values of the context and the resource in question. The policy server validates the policy of the requested resource against the context provided by the mashup program.

It must be noted that each domain in the MLS is can use its own policy language as long as it conforms to the policy interface provided by the policy server. In this implementation,

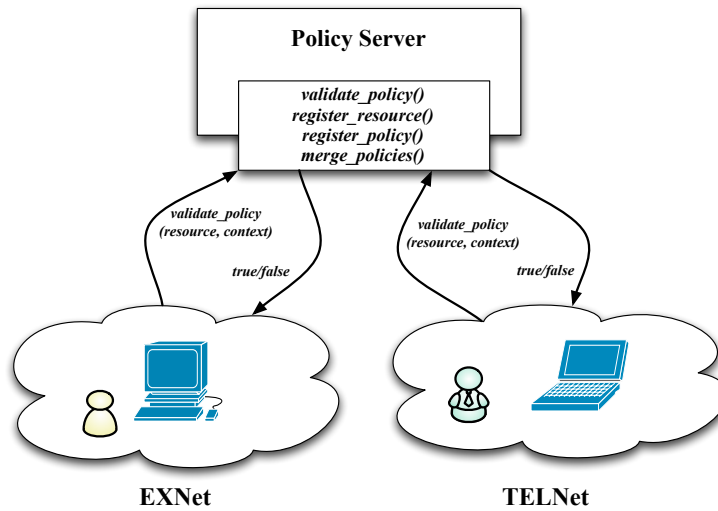


Figure 5.2: A multi-level security mashup policy server

policies are connected to resources by means of a relational table in the policy database on the server. However, another way to implement the mashup mechanism is that instead of a policy server, each resource is embedded with the policy object, and travels with the resource. This is a distributed implementation and can be implemented in situations where a central point of failure can be disadvantageous.

5.2 Cloud Computing

In the recent years, cloud computing has managed to emerge as a computing platform that allows computing services to be consumed as a utility by consumers. In cloud computing, applications, systems software, and hardware are offered as utility services to consumers over the Internet. In service-based architectures, service consumers need to be provided with highly reliable services that meet their expectations. Service consumers indicate these expectations in terms of QoS parameters that are expressed in the form of an SLA negotiated with the service provider.

Chapter 5. Applications

In the realm of computing, there exist multiple service-based paradigms such as web-services, cluster computing, grid computing and cloud computing [19]. Cloud computing is set apart from the other forms of service-based computing paradigms by a collective set of distinguishing characteristics such as market orientation, virtualization, dynamic provisioning of resources and service composition via multiple service providers [20]. These characteristics imply that in cloud computing, cloud users' data reside in the cloud for a finite amount of time, that these data are handled by multiple cloud services, and that data fractions may be stored, processed, transformed and routed across a geographically distributed cloud infrastructure. These activities occur “behind-the-scenes”, within the cloud, while giving cloud users an impression of a single virtual machine. In current cloud implementations, cloud users have very little control over the manner in which data are handled by cloud providers, once they are pushed into the cloud. As consumers start aggressively using cloud services, this limitation will become a matter of serious concern.

The handling or use of cloud users' data within the cloud by different services refers to policies specifying the constraints under which different actions may be carried out on the data. A cloud user may want to limit the manner in which data are stored, routed, or processed, and specify the entities and processes that are authorized to carry out these activities, and under what conditions. As an example, a government agency might want to prevent its data from being stored in one particular country, or prevent routing of the data via a particular set of networks it considers unreliable or insecure. Similarly, a financial company might want to prevent its data from being processed by a particular cloud service, or may want the data to be encrypted before being stored by an untrusted cloud storage service. Usage policies typically consist of a range of semantics such as restrictions on the manner in which data are used, temporal restrictions on usage, spatial or attribute-based restrictions, permissions, obligations, penalties, count-based limits on usage, usage tracking and reporting, and partial dependencies to name a few. Hence, as cloud services become pervasive, cloud users will want to dictate the terms of usage for their data within the cloud in a manner that is expressive enough to capture their concerns.

Existing cloud infrastructures and SLA frameworks are ill-equipped to address the requirements and challenges of usage management in distributed cloud systems. At present, cloud providers enable usage management features via rudimentary techniques involving a one-size-fits-all option, where cloud users have little or no say in expressing usage policies over their data. For example, the Amazon S3 storage service allows a region-based facility where data stored in one region are guaranteed not to leave that particular region [4]. Such options are too coarse and simplistic to address cloud users' concerns.

Existing service-based computing paradigms such as web services, clusters and grid computing use well-established SLA frameworks that enable expression, interpretation, monitoring, control and enforcement of SLA terms [57, 45, 74, 61]. However, the SLAs supported by these frameworks focus on performance metrics such as availability, reliability, bandwidth, response times, instructions per second, etc. [60]. Compared to this, usage management requirements are different since cloud users potentially have a say in every action taken on their data, throughout the lifetime of the data in the cloud, including scenarios where the data are passed on to a third party provider.

Usage management requirements are significantly different from QoS-based metrics, and present a new set challenges that are beyond the capabilities of the existing SLA frameworks. It is therefore necessary to design a separate mechanism for usage management that will enable persistent, automated management data usage control, while allowing seamless flow of data in a distributed cloud infrastructure.

5.2.1 Usage Management Requirements in Clouds

Figure 5.3 shows a high-level model of the manner in which existing systems implement usage management of user data. In this setup usage management policies are statically defined in an SLA by allowing cloud users to choose from a set of options provided by the cloud provider. The usage policy monitoring in existing cloud systems, in many in-

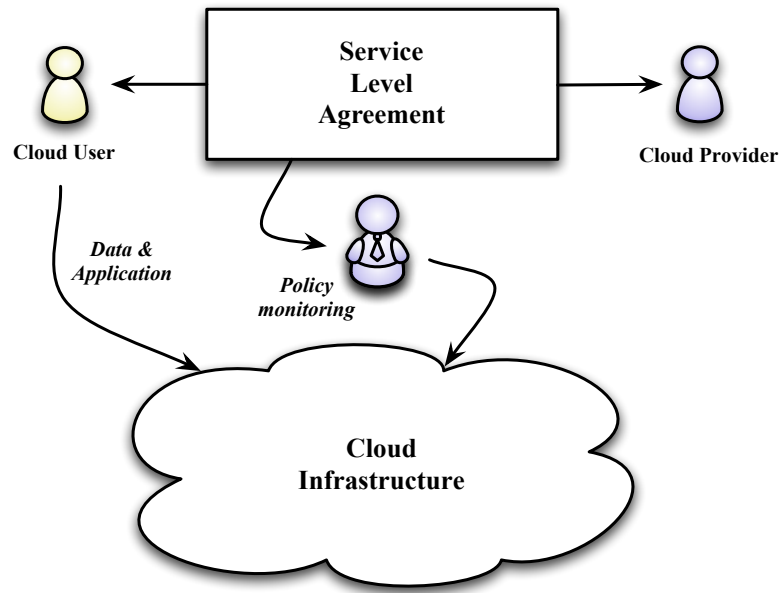


Figure 5.3: Usage management in existing cloud environment.

stances, is carried out by a person (e.g., a lawyer) as shown in Figure 5.3. Such a static approach for handling usage policies presents numerous problems such as lack of fine grained expression of policies, lack of reasoning of policies with respect to underlying cloud infrastructure and under-utilization of resources.

In order to address these concerns, it is necessary that usage policies be managed by means of a usage management framework that will operate over a distributed cloud infrastructure as shown in Figure 5.4. In this figure, a cloud user and a cloud provider negotiate a usage policy and the price associated with the policy in an automated manner via software agents. Existing negotiation frameworks such as the Java Agent Development Framework and the FIPA negotiation protocols can be used for this purpose [14]. Next, the policy is represented in a machine-readable and machine-actionable form. The policy is then interpreted and enforced over a distributed cloud infrastructure making use of existing trusted computing platforms to provide a comprehensive usage management solution. Significant business opportunities become available if these SLAs are easily customizable, allowing

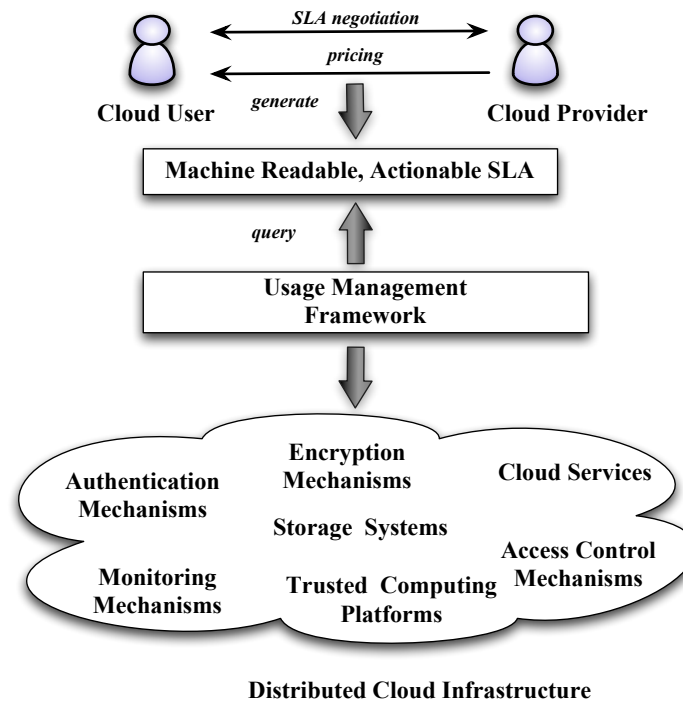


Figure 5.4: Usage management in cloud.

for price differentiation and yield management. The usage management framework can be overlaid on the cloud infrastructure to achieve the following goals.

1. Fine-grained expression of usage policies that can be negotiated between cloud users and cloud providers.
2. Seamless movement of data within the cloud and persistent enforcement of usage policies across different services, at all levels of virtualization, along all data transformations throughout the lifetime of data in the cloud.
3. Actionable policies that enable automated interpretation, reasoning, enforcement, usage tracking and reporting vis-a-vis the underlying cloud infrastructure.

Next, a demonstration of how such a framework can be laid on cloud infrastructure and the capabilities it enables in the cloud is provided.

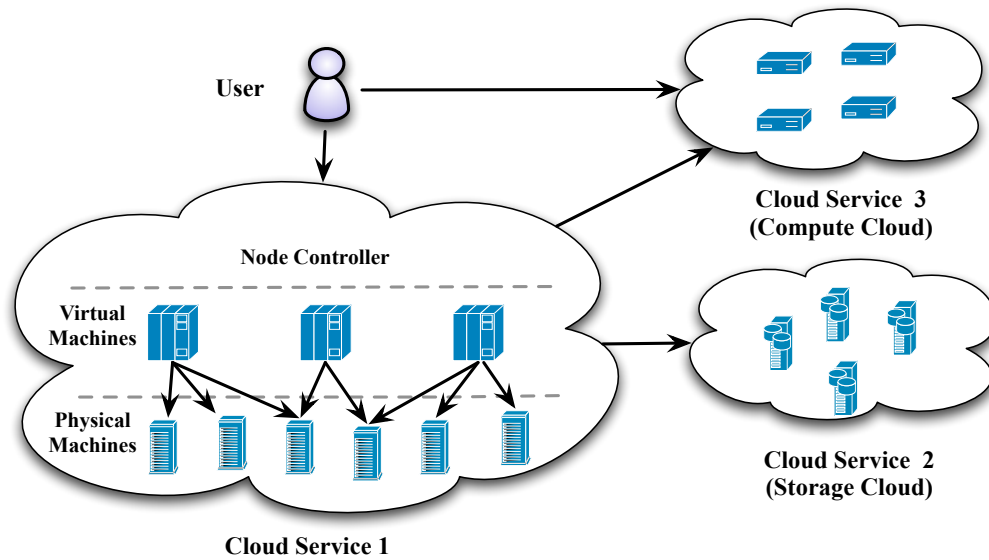


Figure 5.5: A distributed cloud infrastructure consisting of multiple cloud services and virtualization.

5.2.2 Overlaying Usage Management Framework on Cloud Infrastructure

A distributed cloud environment presents a unique set of challenges for the development of a framework that will enable usage management of data that are stored, routed and processed by different services operating within the cloud environment. Figure 5.5 shows the layout of a typical cloud computing environment. A distributed cloud environment may consist of one or more cloud services that are owned by different entities. Services are often composable, and available to users or other services via web-services technologies such as REST and SOAP. The services provide virtualized resources to cloud users that are provisioned on demand to meet the SLA terms that are negotiated between the cloud provider and the user. Within a cloud service implementation, the set of virtual resources are mapped onto a distributed infrastructure of physical resources as shown in Figure 5.5. The allocation of resources for a VM is generally carried out by a controller. Allocation

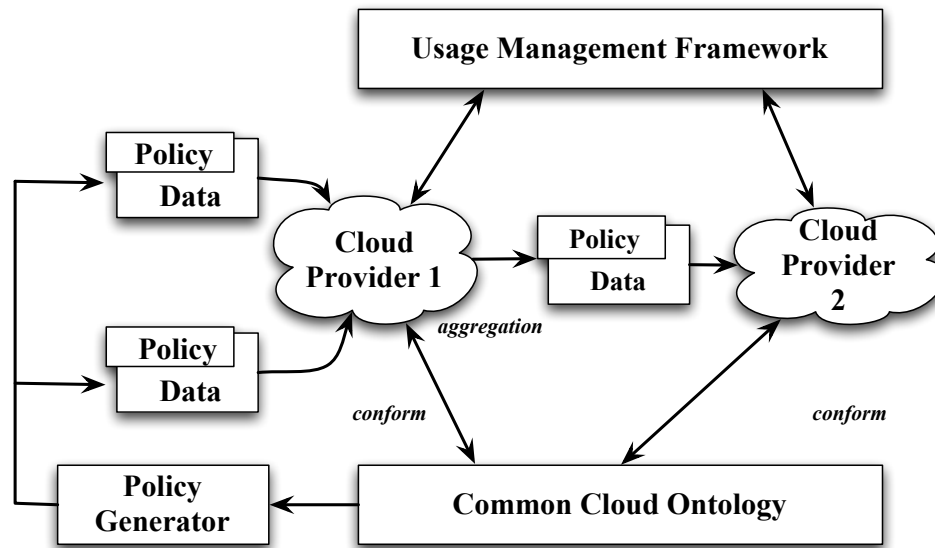


Figure 5.6: Persistence of policies across data aggregations.

of resources is determined by the level of service requested by a user. For example, to maintain a particular level or processing time, a controller will allocate more resources, as the number of request load increases.

In order to incorporate usage management in cloud computing it is necessary that resources or data that is inserted into the cloud is accompanied by usage policies specified by data owners. As mentioned earlier, policy objects can either be embedded in the data or policies can be stored on a policy server and attached to resources via indirection, as explained in the MLS system.

Figure 5.6 shows the case when policies are embedded in the data. In this case, cloud service 1 aggregates data from different sources and then passes it on to cloud service 2 for storage. In this case policy objects are embedded inside the data, and when policies are aggregated, a combined policy is generated for the mashed up data.

Since cloud services are loosely distributed and maintain a certain level of independence, using the proposed framework, each service provider can use its own policy lan-

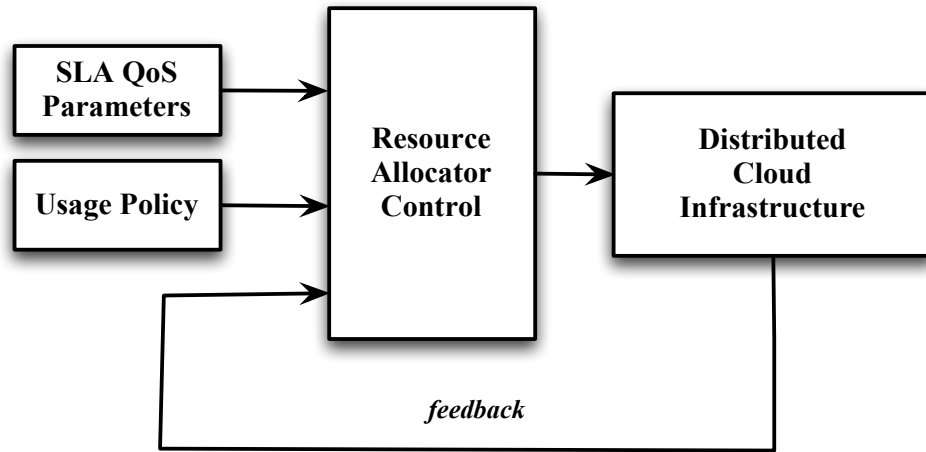


Figure 5.7: Cloud usage management operating with SLA QoS monitoring.

guage as long as it conforms to the cloud ontology and the policy interface. Each cloud service will then have to implement a UMM at the controller level, where allocation of resources takes place. The operation of a resource allocator control is shown in Figure 5.7.

Every resource allocator controller in a cloud service will allocate resources based on two input parameters, namely, the SLA agreed between user and the service provider and usage policy associated with the data for which cloud resources (networking, storage and computing) are to be allocated. A resource allocator control will first consult the SLA, and select a set of resources to be allocated for the data. Once the resources are selected, the allocator will query the policy associated with the data to determine which of the resources are valid to be allocated for the data in question. Hence, whenever data moves through the system or is handled by processes, its handling and movement is controlled by the usage policy that governs it.

The deployment of the usage management framework on cloud infrastructure operates in two phases, similar to the ones described in Chapter 3. There is a Setup stage and a Working Stage explained below.

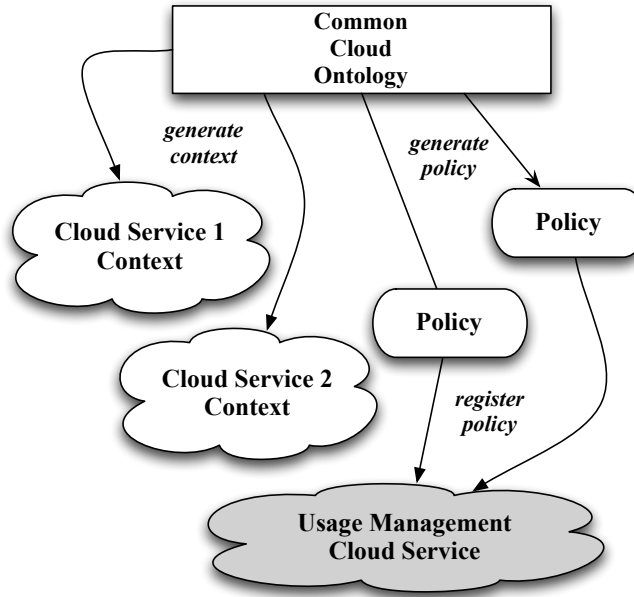


Figure 5.8: Setup phase usage management that include context and license generation.

Setup Phase

The underlying principle driving the design of proposed architecture is that policies are evaluated within a context, and a context is separated from the syntax and semantics of policy languages. A usage policy consists of restrictions over usage expressed in terms of context properties. A context provides a formal representation of the cloud environment within which a policy is interpreted. A context formally represents the entities within the environment, the relations among the entities, properties of the cloud environment, and the current state of the environment.

As an example, consider a context for a cloud service consisting of different data processing sub-services (*dps*). A simplified context may consist of the following parameters: $\{date, dps_location, dps_trust - level\}$, where *date* represents the global date, *dps_location* represents the location of the sub-service, and *dps_trust - level* represents the trust level of the sub-service ranging from 1-5. Usage policies for the data are then expressed as re-

Chapter 5. Applications

strictions over these parameters. For example, a rudimentary usage policy might state that “*Data set X can be processed only by a dps located in the USA, with trust levels greater than 4, and not beyond December 16th, 2013*”.

Even though the example presented above is oversimplified, the point to note is that every cloud service has a representative context whose state (values taken by the parameters in the context) is continually maintained with appropriate values. During the Setup phase, shown in Figure 5.8, cloud services define and generate their own context from a common cloud ontology that is shared among all cloud services. Depending the type and nature of cloud services, services will share certain common context parameters such as time, date and location, whereas differentiate in terms of other parameters. Usage policies over users’ data set are generated using the common cloud ontology. It can be seen that all the parameters, using which policy restrictions are described, must be well-defined in the context of the cloud environment within which the policy is to interpreted.

In the proposed architecture, the policy languages used for expressing the policies are not standardized. Different cloud services may use their choice policy languages depending on their expression and reasoning requirements. However, in order for policies to be interpreted across different cloud environments, it is necessary that all policies make use the terms defined in the common cloud ontology that is shared by all cloud services. The manner in which policies are interpreted in the operational phase is explained next.

Working Phase

The Working phase consists of policy management, policy interpretation and validation via a common usage management cloud service that is shared by different services within the cloud. The operation of usage management cloud service is shown in Figure 5.9. Instead of standardizing a common policy language for all cloud services, in the proposed architecture, the usage management cloud service provides a standard web interface for managing

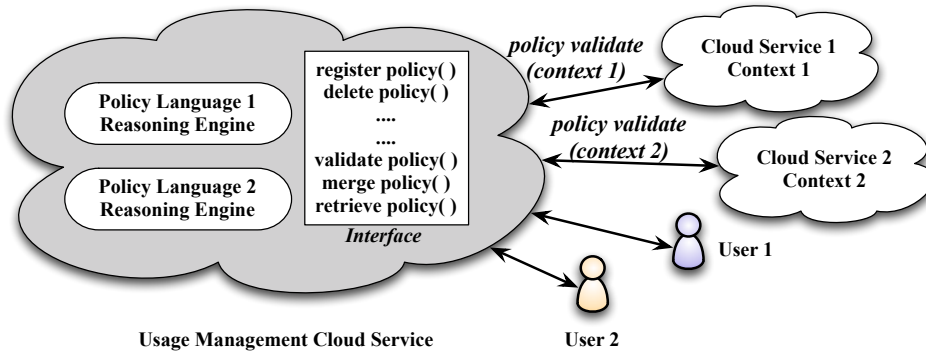


Figure 5.9: Operation of usage management in cloud environment.

policies. This standard interface allows policies to be registered, retrieved, attached to data sets, deleted, validated, interpreted, reasoned, and merged with other policies by different cloud services via an extensible web interface.

In the proposed system, usage policies are validated by cloud services in the following manner. Cloud services query the usage management service for a particular action on a particular data set by providing the context under which the action is intended to be carried out. The usage management service then evaluates whether the policy and the context are interoperable by comparing the terms. If the terms are consistent, then the policy restrictions are evaluated with respect to the current state of the context provided. If the restrictions are satisfied, the action is given a go ahead, otherwise it is denied. Every policy maintains a state or history of the usage performed by different cloud services on the data set. Such a history is commonly maintained in usage control and DRM languages to enforce count limits or obligation semantics on certain actions over a data set. Data usage histories can also be potentially used to validate data use with respect to policy terms, and provide cloud users with usage tracking services for their data.

Cloud services can register policies expressed in different policy languages, however, all the policies are queried by means of a standard web interface. This approach ensures that the syntax and semantics of different policy languages is hidden from the cloud ser-

Chapter 5. Applications

vices that need to query usage policies. This precludes the need for every cloud service provider to deploy an interpreter for different policy languages. In addition, a service provider can introduce a new policy language for its operations, along side its previous policies expressed in the old policy language in a seamless manner. Figure 5.9 shows how multiple policy languages can operate behind a common, standardized web interface for usage policy management.

Chapter 6

Conclusions

In this thesis a motivation is provided for the need for usage management solutions that operate in open distributed environments. First, a meta-model was developed that can be used as a reference for studying different usage management models. Following this, principles of system design were applied to develop a framework for usage management that enables interoperability. In this framework, focal points were identified where it is necessary to apply standards, and areas that needed to be free of standards. The underlying principles of the proposed model is the clear separation of policy expression, policy interpretation and policy enforcement. Such a separation allows policies to be expressed with minimal *a priori* knowledge of computing environments in which the policies will be interpreted. Finally, a mathematical model was developed that provides a specific instance of the proposed framework and provides formal semantics for interoperability and enables dynamic interpretation of policies. It was further shown how policies and contexts can be categorized in a hierarchical manner and reasoned over for interoperability. The work also demonstrates how the framework can be overlaid on distributed systems including multi-level security systems, cloud computing, and DRM systems.

Chapter 6. Conclusions

In distributed systems, such as DRM systems, resource owners have little or no control over client systems, which makes enforcement of usage policies on such system extremely difficult. In such scenarios it has been shown in this thesis that it is possible to model current approaches within a game-theoretic setting. Furthermore, it is demonstrated that the existence of a trust authority middleware infrastructure is necessary to influence the behavior of customers to respect the copyright.

Chapter 7

Future Work

For the future work it is necessary to incorporate this model to enable added functionalities such as mashups, that will allow merging of policies. This needs to be followed by the development of a library of license languages using different types of logics with varying capabilities that can be used within the model. Such a library can be used by license providers to allow easy generation of licenses to average users that can be interpreted across different environments. Finally, building upon the model to develop different types of information ecosystems that can handle different types of resources and data including medical data, commercial content, and financial data, will demonstrate further applications of the proposed framework.

With respect to the DRM games, one of the prime components of these games is a trust authority middleware infrastructure that is used in order to rate the behavior of customers in this game, and reward them accordingly. The ability to implement such a trust authority is by no means a solved problem. Future research must consider how such an infrastructure can be constructed while at the same time addressing security and privacy concerns. In addition, future research should address more specific strategies associated with the repeated games presented in this thesis.

References

- [1] Enabler release definition for DRM V2.0. Technical report, Open Mobile Alliance, 2003.
- [2] Open Digital Rights Language ODRL Version 2 Requirements. ODRL, Feb. 2005.
- [3] Marlin architecture overview. Technical report, 2006.
- [4] Amazon Web Services: Overview of Security Processes, Sep. 2008.
- [5] Eytan Adar and Bernardo A. Huberman. Free riding on a Gnutella. *First Monday*, 5(10), Oct. 2000.
- [6] Sadeghi Ahmad-Reza and Christian Stubble. Taming trusted platforms by operating system design. *Springer-Verlag*, pages 286–302, 2004.
- [7] Harald Alverstrand. The role of the standards process in shaping the internet. *Proceeding of the IEEE*, 92(9):1371–1374, 2004.
- [8] Alapan Arnab and Andrew Hutchison. Extending ODRL to enable bi-directional communication. In *Proceedings of the Second International ODRL Workshop*, Lisbon, Portugal, July 2005.
- [9] Alapan Arnab and Andrew Hutchison. Fairer usage contracts for DRM. In *Proceedings of the Fifth ACM Workshop on Digital Rights Management*, pages 1–7, Alexandria, VA, Nov. 2005.
- [10] Alapan Arnab and Andrew Hutchison. Persistent Access Control: A Formal Model for DRM. In *DRM '07: Proceedings of the 2007 ACM workshop on Digital Rights Management*, pages 41–53, New York, NY, USA, 2007. ACM.
- [11] Adam Barth and John C. Mitchell. Managing digital rights using linear logic. In *LICS '06: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 127–136, Washington, DC, USA, 2006. IEEE Computer Society.

References

- [12] D. Elliott Bell and Leonard J. La Padula. Secure Computer Systems: Mathematical Foundations, MTR-2547, Vol. I. Technical report, The MITRE Corporation, Bedford, MA, March 1, 1973. ESD-TR-73-278-I.
- [13] D. Elliott Bell and Leonard J. La Padula. Secure Computer System: Unified Exposition and Multics Interpretation, MTR-2997, Rev. 1. Technical report, The MITRE Corporation, Bedford, MA, March 1976. ESD-TR-75-306.
- [14] Fabio Bellifemmine, Agostino Poggi, and Giovanni Rimassa. JADE: A FIPA Compliant Agent Framework. In *Proceedings of the Practical Applications of Intelligent Agents*, pages 97–108, London, U.K., 1999.
- [15] D. Bergemann, T. Eisenbach, J. Feigenbaum, and S. Shenker. Flexibility as an instrument in DRM systems. In *Fourth Workshop on the Economics of Information Security*, Kennedy School of Government, Harvard University, June 2–3 2005.
- [16] D. Berlind, Executive Editor. Content restriction annulment and protection (CRAP) video.
- [17] Marjory S. Blumenthal and David D. Clark. Rethinking the design of the Internet: The end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, Aug. 2001.
- [18] A. J. Bonner. A prolog framework for reasoning about permissions and obligations, with applications to contract law. Technical report, Rutgers University, 1988.
- [19] Rajkumar Buyya. Market-Oriented Cloud Computing: Vision, Hype, and Reality of Delivering Computing as the 5th Utility. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09*, Washington, DC, USA, 2009. IEEE Computer Society.
- [20] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.
- [21] Philip L. Campbell, Lyndon G. Pierson, and Edward L. Witzke. Trusted objects. Technical report, Sandia National Laboratories, 2000.
- [22] Amy Carroll, Mario Juarez, Julia Polk, and Tony Leininger. Microsoft palladium: A business overview. Technical report, Microsoft Content Security Business Unit, June 2002.

References

- [23] Cheun Ngen Chong, Ricardo Corin, Sandro Etalle, Pieter Hartel, Willem Jonker, and Yee Wei Law. LicenseScript: A novel digital rights language and its semantics. In *Third International Conference on the Web Delivery of Music*, pages 122–129, Los Alamitos, CA, Sept. 2003.
- [24] N. Christin, A. Weigend, and J. Chuang. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *Proceedings of the 6th ACM Conference on Electronic Commerce*, pages 68–77, New York, NY, 2005.
- [25] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in Cyberspace: Defining Tomorrow’s Internet. In *SIGCOMM*, pages 347–356, Pittsburgh, Pennsylvania, USA, Aug. 2002.
- [26] Tim Clark. IBM closes unit, 2002.
- [27] Coral consortium whitepaper. Technical report, Feb. 2006.
- [28] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The Ponder Policy Specification Language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, POLICY ’01, pages 18–38, London, UK, 2001. Springer-Verlag.
- [29] D. Dhanekula, G. L. Heileman, and B. Horne. Content spreading in peer-to-peer networks. In *Proceedings of IADIS International Conference on e-Commerce 2005*, pages 85–92, Porto, Portugal, Dec. 15–17 2005.
- [30] The Digital Millennium Copyright Act, H.R. 2281, *United State Code*, Pub. L. No. 105-304, 112 Stat. 2860, Oct. 28 1998.
- [31] Prajit K. Dutta. *Strategies and Games: Theory and Practice*. MIT Press, Cambridge, MA, 1999.
- [32] E. W. Felton. A skeptical view of DRM and fair use. *Communications of the ACM*, 46(4):56–59, 2003.
- [33] Free software foundation. some confusing or loaded words and phrases that are worth avoiding. www.gnu.org/philosophy/words-to-avoid.html.
- [34] Carl Gunter, Stephen Weeks, and Andrew Wright. Models and languages for digital rights. In *HICSS ’01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 9*, page 9076, Washington, DC, USA, 2001. IEEE Computer Society.

References

- [35] Stuart Haber, Bill Horne, Joe Pato, Thomas Sander, and Tarjan Robert Endre. If piracy is the problem, is DRM the answer? Technical report, Trusted Systems Laboratory, HP Laboratories Cambridge, 2003.
- [36] Joseph Y. Halpern and Vicky Weissman. A Formal Foundation for XrML Licenses. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, pages 251–265, Asilomar, CA, June 2004.
- [37] Joseph Y. Halpern and Vicky Weissman. A Formal Foundation for XrML. *J. ACM*, 55(1):1–42, 2008.
- [38] Gregory L. Heileman and Pramod A. Jamkhedkar. DRM Interoperability Analysis from the Perspective of a Layered Framework. In *Proceedings of the Fifth ACM Workshop on Digital Rights Management*, pages 17–26, Alexandria, VA, Nov. 2005.
- [39] Gregory L. Heileman, Pramod A. Jamkhedkar, Joud Khoury, and Curtis J. Hrnccir. DRM game. In *Proceedings of the Seventh ACM Workshop on Digital Rights Management*, pages 54–62, Alexandria, VA, Oct. 2007.
- [40] B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, 2001.
- [41] Vincent Hu, David Ferraiolo, and Rick Kuhn. Assessment of Access Control Systems. Technical report, National Institute of Standards and Technology, Sep. 2006.
- [42] Helge Hundacker, Daniel Pahler, and Rudiger Grimm. URM - usage rights management. In *Proceedings of the 7th International Workshop for Technical, Economic and Legal Aspects of Business Models for Virtual Goods*, pages 125–138, Nancy, France, Sep. 2009.
- [43] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, England, 2000.
- [44] Renato Iannella. Open digital rights language (ODRL), Version 0.5, Aug. 2000.
- [45] IBM. *Web Service Level Agreement (WSLA) Language Specification*, Jan. 2003.
- [46] Pramod A. Jamkhedkar and Gregory L. Heileman. DRM as a Layered System. In *Proceedings of the Fourth ACM Workshop on Digital Rights Management*, pages 11–21, Washington, DC, Oct. 2004.
- [47] Pramod A. Jamkhedkar and Gregory L. Heileman. *Handbook of Research on Secure Multimedia Distribution*, chapter Rights Expression Languages. IGI Publishing, 2008.

References

- [48] Pramod A. Jamkhedkar, Gregory L. Heileman, and Chris Lamb. An Interoperable Usage Management Framework. In *Proceedings of the Tenth ACM Workshop on Digital Rights Management*, Chicago, Oct. 2010.
- [49] Pramod A. Jamkhedkar, Gregory L. Heileman, and Ivan Martinez-Ortiz. The Problem with Rights Expression Languages. In *Proceedings of the Sixth ACM Workshop on Digital Rights Management*, pages 59–67, Alexandria, VA, Nov. 2006.
- [50] S. Katzenbeisser, K. Kursawe, and J. Talstra. Graceful infringement reactions in DRM systems. In *Proceedings of the Sixth ACM Workshop on Digital Rights Management*, pages 89–95, Alexandria, VA, Oct. 30 2006.
- [51] Rob H. Koenen, Jack Lacy, Michael MacKay, and Steve Mitchell. The Long March to Interoperable Digital Rights Management. *Proceedings of the IEEE*, 92(6):883–897, 2004.
- [52] Ulrich Kohl, Jeffrey Lotspiech, and Marc A. Kaplan. Safeguarding digital library contents and users: Protecting documents rather than channels. *D-Lib Magazine*, 3(9), Sept. 1997.
- [53] M. B. Margolin, M. K. Wright, and B. N. Levine. Analysis of incentives-based secrets protection system. In *Proceedings of the Fourth ACM Workshop on Digital Rights Management*, pages 22–30, Washington, DC, Oct. 2004.
- [54] L. T. McCarty. A language for legal discourse I. basic features. In *ICAIL '89: Proceedings of the 2nd international conference on Artificial intelligence and law*, pages 180–189, New York, NY, USA, 1989. ACM.
- [55] D. L. McGuinness. Reasoning with permissions and obligations in contract law. Technical report, Rutgers University, 1986.
- [56] P. Moulin and R. Koetter. Data-hiding codes. *Proceedings of the IEEE*, 93(12):2083–2126, Dec. 2005.
- [57] Open Grid Forum. *Web Services Agreement Specification*, Mar. 2007.
- [58] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.
- [59] Jaehong Park and Ravi Sandhu. The UCON_{ABC} Usage Control Model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.
- [60] Adrian Paschke and Elisabeth Schnappinger-Gerull. A Categorization Scheme for SLA Metrics. In *Proceedings of Multi-Conference Information Systems (MKWI06)*, Feb. 2006.

References

- [61] Pankesh Patel, Ajith Ranabahu, and Amit Sheth. Service Level Agreement in Cloud Computing. In *Proceedings of the Workshop on Best Practices in Cloud Computing: Implementation and Operational Implications for the Cloud at ACM International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Orlando, FL, Oct. 2009.
- [62] Josep Polo, Jose Prados, and Jaime Delgado. Interoperability between ODRL and MPEG-21 REL. In *Proceedings of the first international ODRL workshop*, Vienna, Austria, Apr. 2004.
- [63] Bogdan C. Popescu, Frank L. A. J. Kamperman, Burno Crispo, and Andrew S. Tanenbaum. A DRM security architecture for home networks. In *Proceedings of the Fourth ACM Workshop on Digital Rights Management*, pages 1–10, Washington, DC, Oct. 2004.
- [64] Riccardo Pucella and Vicky Weissman. A logic for reasoning about digital rights. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 282–294, Nova Scotia, Canada, June 2002.
- [65] Riccardo Pucella and Vicky Weissman. Reasoning about dynamic policies. In *Proceedings FoSSaCS-7, Springer Lecture Notes in Computer Science 2987*, pages 453–467. Springer-Verlag, 2004.
- [66] Riccardo Pucella and Vicky Weissman. A formal foundation for ODRL, Jan. 2006.
- [67] Reihaneh Safavi-Naini, Nicholas Paul Sheppard, and Takeyuki Uehara. Import/export in digital rights management. In *Proceedings of the Fourth ACM Workshop on Digital Rights Management*, pages 99–110, Washington, DC, Oct. 2004.
- [68] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based Access Control Models. *Computer*, 29(2):38–47, feb 1996.
- [69] Andreas U. Schmidt, Omid Tafreschi, and Ruben Wolf. Interoperability challenges for DRM systems. In *IFIP/GI Workshop on Virtual Goods*, Ilmenau, Germany, 2004.
- [70] Bruce Schneier. The fallacy of trusted client software (cryptorhythms column). *Information Security Magazine*, Aug. 2000.
- [71] S. Sun, L. Lannom, and B. Boesch. The handle system overview. Technical report, Corporation for National Research Initiatives, Nov. 2003.
- [72] Eliot Van Buskirk. A poison pen from the RIAA. *WIRED*, Feb. 28 2007. www.wired.com/politics/onlinerights/news/2007/02/72834.

References

- [73] Xin Wang. MPEG-21 rights expression language: Enabling interoperable digital rights management. *IEEE Multimedia*, 11(4):84–87, October/December 2004.
- [74] World Wide Web Consortium. *Web Services Policy 1.5 Framework*, Sep. 2007.
- [75] Jianwen Xiang, Dines Bjorner, and Kokichi Futatsugi. Formal digital license language with OTS/CafeOBJ method. In *Proceedings of the sixth ACS/IEEE International Conference on Computer Systems and Applications*, Doha, Qatar, Apr. 2008.
- [76] eXtensible Rights Markup Language (XrML) 2.0 Specification, November 2001.
- [77] XrML 2.0 technical overview, version 1.0, March 2002.
- [78] Zhiyong Zhang, Qingqi Pei, Jianfeng Ma, Lin Yang, and Kefeng Fan. Cooperative and non-cooperative game-theoretic analyses of adoptions of security policies for DRM. In *Proceedings of the 6th Annual IEEE Consumer Communications and Networking Conference*, pages 1–5, Las Vegas, NV, Jan. 2009.