Pervasive Application Rights Management Architecture

Ivana Dusparic

A thesis submitted to the University of Dublin, Trinity College

in fulfillment of the requirements for the degree of

Master of Science

July 2005

Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise stated, it is entirely my own work.

Ivana Dusparic

Dated: 6th July 2005

Permission to Lend and/or Copy

Acknowledgements

First and foremost, I would like to thank my supervisor, Jim Dowling, for his guidance, support, and enormous patience throughout this project. Also, special thanks to Dominik Dahlem for the cooperation on PARMA and for always being available to discuss views and ideas, and to Daniel Molinard for all the help with PARMA implementation.

Big thanks goes to Elizabeth Daly and Elizabeth Gray for helping me sort out my English language problems, and putting up with my mood swings during the thesis write-up. Thanks to everyone in the Distributed Systems Group for being such a great crowd that I actually did not miss home all that much during this project.

Thanks to my family for their great help and understanding when I decided to undertake this thesis.

And finally, thanks to people who didn't want to be named, but without whose support this thesis would have not been possible, they know who they are!

Ivana Dusparic
University of Dublin, Trinity College
July 2005

Contents

List of Figures

List of Tables

Table of Acronyms

- AOP - Aspect Oriented Programming
- DD - Data Dictionary
- DRM - Digital Rights Management
- EDS - Enterprise DRM Service
- GPRS - General Packet Radio Service
- GPS - Global Positioning System
- HTTP - Hypertext Protocol
- IMEI - International Mobile Equipment Identity
- J2ME - Java 2 Mobile Edition
- J2SE - Java 2 Standard Edition
- LAN - Local Area Network
- LMS - License Management System
- MIDP - Mobile Information Device Profile
- MSISDN - Mobile Station Integrated Services Digital Network Number
- ODRL - Open Digital Rights Language
- OMA - Open Mobile Alliance
- OS - Operating System
- PARMA - Pervasive Application Rights Management Architecture
- REL - Rights Expression Language
- SOAP - Simple Object Access Protocol

- UI - User Interface
- URI - Uniform Resource Identifier
- VDS - Vendor DRM Server
- WAP - Wireless Application Protocol
- WSDL - Web Services Description Language
- XML - eXtended Markup Language
- XrML - eXtensible rights Markup Language
- XSLM - X-Open Software License use and Management System
- XSLT - eXtensible Stylesheet Language Transformation

Summary

This dissertation describes an application rights management architecture that combines license management with digital rights management to provide an integrated platform for the specification, generation, delivery and management of application usage rights for pervasive computing environments. A new rights expression language is developed, extended from the existing language, ODRL, which allows the expression of mobile application usage rights and supports fine-grained usage models. An audit-based usage rights model is introduced to support application usage without requiring immediate network availability. Rights are designed to be enforced on the device itself, without the need for communication with a server for every validation, realising the goal of reduced wide-area communications by mobile devices. The addition of the rights enforcement source code to the original application is facilitated using aspect-oriented programming and is performed at the license management server rather than by the developer. Back-end license management services, hosted by the application vendor, and/or enterprise customer, are based on web services so they can deliver applications and licenses to any client device that supports SOAP and/or HTTP(S) protocols and provide an integrated platform for management of licenses on both desktop and mobile devices.

The thesis was motivated by the lack of suitable usage right management systems for the emerging pervasive environments. Pervasive computing devices include handheld computers, personal digital assistants, mobile phones, pagers, and others, and are generally characterised by mobility, occasional network connectivity, constrained resources, and, in the case of mobile phones, unique hardware identification by an IMEI number. Communication via wide-area network is available in the mobile environment but is still costly when compared to fixed networks. However, mobile devices can communicate using a variety of free short-range protocols, such as Bluetooth, Infrared, and 802.11.

Software applications running on pervasive computing devices are generally licensed using software License Management Systems (LMS), distributed software architectures that manage software licenses throughout their life-cycle, or Digital Rights Management (DRM) systems, systems that specify and enforce usage rights on copyrighted information. Both these types of systems are originally developed for desktop devices in the fixed network environment.

Existing LMS assume constant network connectivity present in LANs and require network connections for license validation and enforcement. This assumption can not be made in pervasive environments because network connectivity is only occasional and often costly. Existing LMS also require the addition of calls to a licensing API in the licensed application's source code. This increases the application development time, as it is performed by the application developer, and it requires knowledge of licensing APIs.

DRM systems that are used to manage usage rights on applications do not require a network connection or knowledge of proprietary APIs, however they are primarily designed for management of usage rights on content. As DRM systems treat applications and content in the same manner, they fail to support fine-grained usage models for applications. License validation and enforcement for an application is performed only once, on application loading, but license enforcement needs to be performed throughout the application execution to support flexible usage models such as feature-based, audit-based, and metered usage models.

A pervasive license management system should overcome the limitations of the existing DRM and desktop LMS when deployed in pervasive computing environments. It should minimize the amount of wide area network traffic to reduce overhead costs introduced by the licensing system. It also requires usage rights models that do not make the assumption of an immediate network connection, to account for the occasional connectivity of mobile devices. A variety of application usage models should be supported so users can choose the best value model. Also, this system should provide a central point for managing all of the licensed software to support enterprise customers, whether it runs on mobile or desktop devices, and integrate easily within other enterprise services, such as payment or asset management. The architecture described in the thesis is designed to meet all of these requirements.

This thesis is a part of the PARMA project that provides a full usage rights architecture. A rights expression language, usage rights servers and a means of integration of usage rights in applications are provided in this thesis, while a client usage right enforcement architecture is provided by the remaining parts of the project.

Chapter 1

Introduction

1.1 Introduction

This chapter introduces the problem area that this thesis is exploring, by introducing the characteristics of license management systems, digital rights management systems, and pervasive computing environments. It motivates the need for a pervasive application rights management architecture, outlines the scope of the thesis and provides a short overview of the architecture design. Finally, this chapter provides a thesis road-map.

1.2 Problem Area

1.2.1 Characteristics of License Management Systems

License management systems are distributed software applications that manage software licenses through their full life-cycle, including the license generation, distribution, enforcement and revocation. Existing systems generally consist of a central licensing server and distributed client libraries linked to licensed applications [Mac03b]. Existing license management systems were designed for a LAN environment that assumes constant network connectivity, and they require a network connection for license enforcement. If the network is not accessible at the time of application start up, application execution will not be allowed, in order to prevent the license misuse [Mac03b]. Relying on a network connection for the license enforcement is not feasible in the mobile environment as occurrences of network disconnections are more frequent [GKN$^+$96]. Also, frequent network communication with the licensing server at application start up, and possibly during the execution and at application shut down [XSL99], adds unnecessary costly network traffic to mobile applications. Another disadvantage of existing desktop licensing models is that they require the addition of calls to the licensing API in the original application source code. These calls need to be scattered around the application's source code, since, depending on the usage model, API calls for enforcement may need to be performed throughout the application. For example, in a feature-based model, licensing API calls may be required at the beginning and at the end of the execution of every feature. The addition of this source code imposes extra work on application developers, as they need to familiarize themselves with the licensing API and make modifications to the application source code [Mac], therefore increasing software development time. Also, the insertion of licensing code in the application source code negatively affects application modularization, making the application less understandable, and more difficult to maintain and update [Tsa03].

1.2.2 Characteristics of DRM systems

Digital Rights Management (DRM) systems are systems designed to prevent the illegal use of copyrighted information by specifying and enforcing usage rights on that information [Ian01, HYP02]. Usage rights are generally specified using a Rights Expression Languages (REL). All of today's RELs are defined using XML schemas and data dictionaries [Gut03]. DRM systems were traditionally concerned with content usage rights enforcement, but are now also specifying and enforcing rights on mobile applications. An example of a DRM system specifying usage rights on both content and mobile applications is OMA DRM [MA02]. DRM systems enforce usage rights on the device executing the application without the need for a network connection, and as such are suitable for rights enforcement on both desktop and mobile environments. However, a lack of means to express and enforce fine-grained usage rights models makes existing RELs unsuitable for basing the pervasive usage rights management architecture on. The set of usage rights models on content, that DRM systems are primarily concerned with, is smaller than a possible set of usage rights models on an application. Usage rights are validated only when a user attempts to view the content. If the user does not own the usage rights on that content, an enforcement architecture denies access to the content. DRM applies the same enforcement on applications. If the user does not own the right to execute the application, application execution is terminated [MA02, MA04]. This all-or-nothing means of license enforcement does not support fine-grained usage models such as a feature-based model. In the feature-based model, users can have usage rights on some features in the application, while they do not have usage rights for other features. A typical example of this model is a trial license where users are allowed to create documents using the application, but are not allowed to save them. In such feature-based models, usage rights have to be validated and enforced before every major feature in the application is executed to determine if users have rights on the execution of that feature. As such a model is available for existing desktop applications, integrated pervasive usage rights management applications should support it as well, both in desktop and mobile environments. Another usage model that existing DRM systems do not implement is the utility-pricing or audit-based usage rights model. This is a form of a post-pay model where actual application usage is logged and users are charged for the application usage afterwards, at regular intervals, usually monthly [Mac04e]. Market research shows that this is one of the preferred models for application usage billing by customers [KGS04] so digital rights and license management systems should offer it as a possible model. Additionally, as the task of DRM systems is only to enforce usage rights on a device executing the application, they do not deal with server side application and usage rights storage, retrieval, distribution and billing. Therefore, on their own, they are not suitable for pervasive usage rights management.

1.2.3 Characteristics of Pervasive Computing Environment

Pervasive computing is defined as a computing environment where numerous computing devices are embedded in the environment and seamlessly interact with people and each other [Swi00]. A pervasive computing environment covers servers, desktops, and a variety of other devices of different sizes and capabilities, including laptops, personal digital assistants (PDAs), mobile phones, set top boxes, game consoles, as well as other small computing devices embedded into everyday appliances such as microwaves, refrigerators, and vending machines [Gri02]. The majority of these devices are mobile, and devices are occasionally connected to each other or to wide-area networking infrastructures (e.g. the Internet), via wired or wireless channels [KFJ01]. The cost of wide-area networking in the wireless environment is still high when compared to the cost of access in a fixed network environment. Wireless connections fees are typically based on the amount of bytes transferred over a communications link, so software on mobile devices needs to minimize the network traffic overhead when communicating with other devices over paid networking channels, such as GPRS [gpr] . However, in the pervasive environment, devices can also communicate with each other via several free short-range wireless protocols, with the choice of protocol depending on the number and distance of the devices involved in the communication (InfraRed [DA01], Bluetooth [SIG03], 802.11 [IEE99]). This opens new channels for super-distribution and sharing of the data and software applications that do not impose a cost on the user.

Mobile pervasive devices are also generally small in size, which results in limited processing, memory and storage capabilities. The average PDA or a smartphone today has 32Mb of storage and a 144-206MHz processor [Wor04, Pac](which is at least 10 times less than the average PC today), limiting the nature of software applications that can be installed on these devices. Mobile phones, as a subset of pervasive computing devices, have additional characteristics based on specific hardware and the wireless telecommunication networks they are connected to. Mobile phones are often uniquely identifiable by an IP address, when connected to wide area networks, their phone number (MSISDN), or by a hardware ID (IMEI [Tel04]).

Due to the nature of mobile devices and wireless networks, typical mobile software applications differ from typical desktop applications. Enterprise mobile applications are often just clients for remote access of central data servers, such as AirClic's Mobile Delivery Solutions [Air04], or applications using Global Positioning System [GPS95] that keep track of and communicate with mobile workforce in the field, such as FleetOnline [Com04]. Mobile applications purchased by individuals are most often used for entertainment, including games, followed by personal productivity and utility applications [Han03]. Mobile applications generally have a shorter life-span than desktop applications, as they are often aimed at a single device make or model, and consumers upgrade mobile devices more often than desktop devices with mobile operators often offering discounted devices upgrades every 12 months [O204]. For this reason, consumers are less willing to pay large sums of money for mobile applications. The average price of a perpetual license [1] for a mobile application for PDAs or Java-enabled phones in 2004 is 13 Euro, which is relatively small when compared to the price of perpetual license for desktop applications [Han03]. Some enterprise applications in the mobile environment are paid for either by usage, such as FleetOnline [Com04] or by monthly subscriptions, such as Symmetry Pro[Inf04], spreading the payments over time rather than requiring large up-front payments.

This demonstrates that in the mobile environment, customers prefer more flexible pay-per-use and subscription software licensing models and are only willing to buy perpetual licenses at a low price [gam03]. However, very few vendors offer flexible licensing models and are either trying to expand licensing management systems from desktop and fixed networks environment into mobile environments (e.g. FLEXnet [Mac04c]), or to apply digital rights management methods of license enforcement for content to mobile applications (e.g. OMA DRM [MA02]). Neither of these approaches can fully satisfy the requirements of pervasive environments, due the differences in device capabilities and network availability, nor can they satisfy the preferences of mobile applications consumers who require flexible usage models [KGS04].

1.3 Motivation for a Pervasive Application Rights Management Architecture (PARMA)

License management systems and digital rights management systems need to be adapted to the mobility and resource-constrained characteristics of a pervasive environment. At the same time, they should also meet the requirements for traditional fixed network environments in order to provide a complete and integrated platform for license management from a single point within the enterprise.

The requirements of an integrated license management system for pervasive environments are as follows:

- The need of existing licensing systems for a network connection at the time of license enforcement must be eliminated.
- The network traffic must be minimized to reduce the cost associated with the license delivery and enforcement.
- The size of the client side enforcement software must be minimized because of the memory constraints of mobile devices.

[1]Perpetual license allows unlimited usage of the application for an indefinite time, unless the license agreement is terminated or breached.

- The amount of additional work by the developer required to license the application must be minimized.
- The application rights management architecture must support a variety of usage rights models, including fine-grained ones, to meet the needs of both end-user customers and enterprise customers in both mobile and desktop environments.

Since no existing license management system meets all of the above criteria, the PARMA architecture was designed to fulfil these requirements and adapt the application rights management process to pervasive environments, by combining features of LMS and DRM. PARMA adopted and extended DRM means of expressing usage rights, by extending the rights expression language ODRL. In section 2.4.2 arguments are presented in favour of extension of this particular language over other existing RELs.

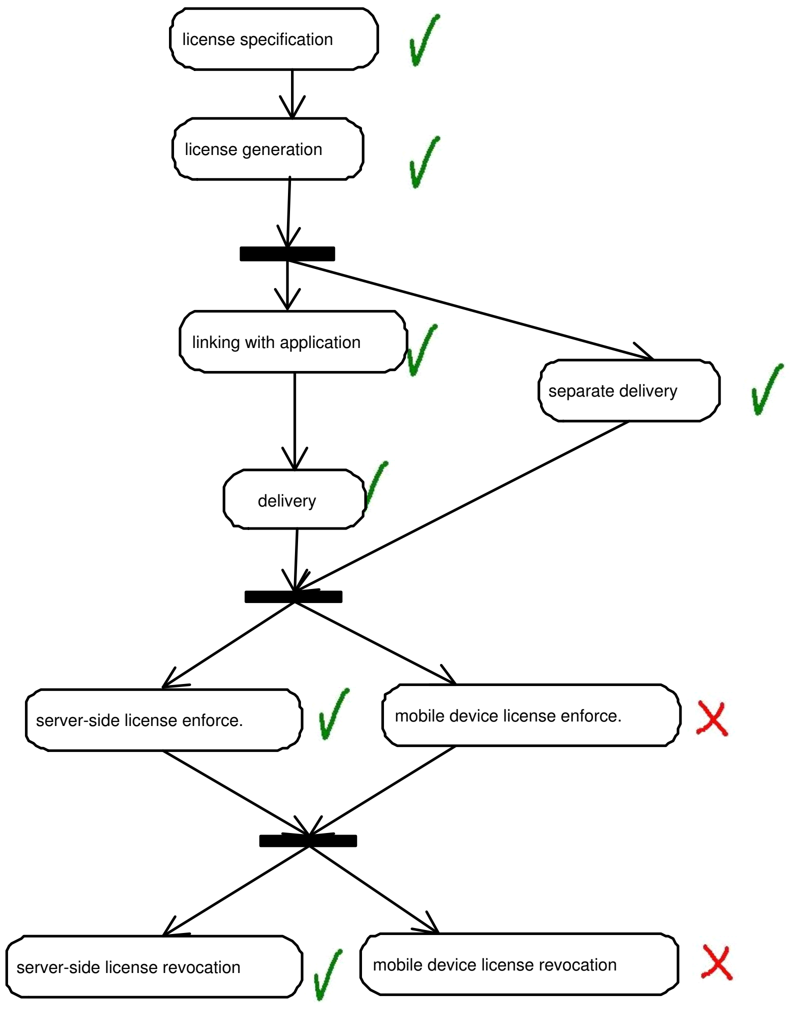
PARMA was also designed to support and encourage super-distribution of applications, i.e. copying the applications from one device to another. PARMA design encourages super-distribution of the application with attached trial usage rights files, so users can preview the content or test the application before buying a full usage rights file. This approach has advantages for both vendors and consumers. The lack of limitations on forwarding the applications, particularly if applications are forwarded with a default set of demo usage rights, encourages users to share the applications with their peers. In this way, a larger number of end users can try out the application, increasing the application popularity and sales, which benefits the application vendor [Buh03]. Also, if given a choice, consumers prefer to download the applications off their peers via free distribution channels, such as Bluetooth and InfraRed, rather than via costly wide-area network communication channels.

1.4 Scope

The complete application usage rights management process manages usage rights through their full life-cycle, from specification to expiration or revocation. This thesis does not cover all of the usage rights management steps, and is a part of a larger project that will provide the remaining license states.

The thesis provides tools that support usage rights specification, usage rights generation from the given specifications, usage rights validation against the defined rights expression language grammar, integration of the usage rights in the target application, usage rights delivery, and server side usage rights enforcement and revocation. A usage rights enforcement and revocation architecture on the client device is not covered in this dissertation (see figure 1.1).

The architecture developed supports both individual developers and large vendors of enterprise applications. Likewise, clients can be individual end-user customers, or large enterprises managing the licenses for all of their employees. As architecture is developed as a web service, clients can be both desktop and mobile devices.



license specification
license generation
linking with application
separate delivery
delivery
server-side license enforce.
mobile device license enforce.
server-side license revocation
mobile device license revocation

Figure 1.1: Scope of the thesis - license life-cycle

The thesis defines a rights expression languages for the expression of fine-grained usage models such as feature-based model and audit-based model. The integration of usage rights management source code in applications is performed using AOP, leaving the original application's source code unchanged.

1.5 Design Overview

The design of the pervasive application rights architecture described in this dissertation was guided by the requirements of pervasive environments and user requirements for usage rights management systems for both desktop and mobile applications. The main design goals were to support a wide-range of usage rights models, eliminate the need for proprietary API calls in licensed applications, and provide a platform that will integrate usage rights management on mobile and desktop devices. Additionally, this architecture was designed to minimize the client side footprint, minimize the client-server traffic, eliminate the need for an on-demand network connection, be compatible with existing standards, support fine-grained usage models, and take advantage of the new device identification means available in the mobile environment.

The architecture consists of two servers, a Vendor DRM Server (VDS) and an Enterprise DRM Server (EDS). The VDS is designed to manage application usage rights on behalf of one or multiple application vendors. All developers, whether large vendors or small application providers, can register with the VDS and submit their applications and supported usage rights models to the VDS. End-users wishing to download applications and usage rights communicate directly with the VDS, as do enterprise administrators who wish to obtain usage rights for enterprise applications. The EDS server is hosted by a large enterprise that uses the licensed applications, and it manages applications and licenses on behalf of the enterprise. The set of features on the EDS is similar to the set of features on the VDS; the only difference is that the EDS deals only with the employees within the licensed enterprise, while the VDS deals with the end-user customers and enterprise customers who wish to use the vendor's applications. Both servers are exposed as WSDL interfaces and accessed over SOAP.

The thesis defines a new REL, extended from Open Digital Rights Language (ODRL), that can define fine-grained usage models for applications, PARMA REL. The PARMA REL can express three novel usage rights models: a *feature-based model*, that is used to limit usage rights on the specific methods within the application, a *subscription model*, that entitles clients on full usage rights during the certain period they are subscribed for, and a *post-paid audit-based model*, that logs the actual application usage, and bills the clients based on the usage data later. The PARMA REL can also define traditional usage rights models such as node-locked, user-locked, concurrent, and time-limited. By offering a variety of usage rights models, most of them requiring payments at set intervals rather than once-off and up-front, PARMA REL is offering flexibility in the usage rights definition required by both vendors and customers [KGS04].

Usage rights using the PARMA REL are expressed in XML documents. XSLT transformation is used to convert XML documents into Java aspect-oriented (AO) code [Kis02]. Aspect-oriented programming (AOP) enables the weaving of REL-specific code into the original application, without modifying the application's source code. Methods in the original application, where usage rights enforcement code should be inserted, are specified when defining the usage rights, and converted into *pointcuts* in aspect-oriented code [Kis02]. At compile time, code specified in a Java aspect file is woven in the original application before or after a specified pointcut. For the feature-based usage model these pointcuts represent methods before which usage rights should be validated, and if execution of the given method is not allowed, the enforcement architecture should prevent the method from executing. For the audit-based usage model, pointcuts can represent the methods whose execution should be logged as the billing information. In all other usage models, pointcuts are inserted at the application start up, and the enforcement architecture either allows the execution to proceed or shuts down the application, based on the permissions and constraints in the usage rights file. The advantage of using XSLT transformations and AOP techniques to insert the usage rights enforcement code is that the process of adding the enforcement in the target application is automated and does not require any action by the developer. Unlike other license management systems, PARMA does not require knowledge of the proprietary APIs, since enforcement code is generated automatically and inserted into the original application only at compile time, without modifying the original source code.

PARMA combines the techniques from DRM, specifically, rights expression languages, and techniques from license management systems, specifically, server side management of the usage rights, to meet the pervasive application rights management requirements outlined in the section 1.3. Since the DRM community refers to permissions and constraints as *usage rights* and the software licensing community as a *license*, these two terms will be used interchangeably when referring to the PARMA architecture.

1.6 Dissertation Structure

Chapter 2 contains the background and state-of-the-art in software licensing and digital rights management technologies. It describes existing licensing techniques and license models and analyses their suitability for different environments.

Chapter 3 provides details of the PARMA rights expression language. It presents features adopted from existing rights expression language and describes their suitability for the architecture presented in the thesis. It motivates the need for new features and explains the new PARMA REL, introduced to provide new flexible usage rights models.

Chapter 4 presents the design of the pervasive application rights management architecture. It describes how the usage rights are created, how are they mapped to aspect-oriented code and how are they woven into the licensed application. It also describes the design and requirements of two DRM servers, the VDS and the EDS.

The implementation is discussed in chapter 5. This chapter describes interfaces for registration with the VDS and the EDS servers, application discovery, download and licensing, as well as tools for generation of usage rights and integration with the licensed application source code.

Chapter 6 provides a comparison of PARMA features developed in this thesis and other existing DRM and software license management systems. It discusses how PARMA addresses the limitations of existing systems and evaluates a new approach to feature-based licensing using AOP.

Final chapter 7 summarizes and concludes the dissertation.

Chapter 2

Background

2.1 Introduction

Software licensing techniques have evolved over time, following the developments in software engineering, hardware capabilities and networking. Through this development, the main purpose of licensing has remained the same: the protection of an application from intentional or unintentional misuse. Software licensing techniques also aim to provide convenient and flexible means of application acquisition, payment and application usage terms for honest software users. Specific systems in different environments deal with the licensing process in different ways, and each environment has its own specific manner of license specification, generation, distribution, validation, enforcement and revocation. Software licensing techniques depend on the type of software application and the target users, the application cost, the hardware and networking environment that the application runs on, and on the players involved in the licensing and distribution process. To meet the different requirements of these actors and environments, numerous licensing models have been implemented. The following sections describe how licensing techniques have changed since the beginning of software development and what has influenced these changes. Particular attention is given to the models implemented today and several standards in software licensing, license management and digital rights management are described and evaluated.

It is important to note that software license management has not attracted much academic interest, and there is a general lack of academic research resources on this topic. Information gathered on license management systems that is presented in the following sections was obtained from vendors' white papers, product datasheets, and user and developer manuals for specific license management products.

2.2 Development of Software Licensing

2.2.1 Standalone Computers

Before the introduction of networks, in the late 1960s and 1970s, all computers were stand-alone. The standard way to copy data and applications from one computer to another, or to deliver new applications to a computer, was through a physical storage medium storing the data or applications. License models on standalone computers have to be fully enforced only on a device itself.

Shrink-Wrapped License Model

The first model used to express software licenses was a so-called *shrink-wrapped* model. In this model, software application is stored on a storage medium (floppy disk, CD) that is wrapped in paper, a plastic cover or a box, and delivered to a user as a physical package. The packaging contains printed terms of usage and states that the user must automatically accept these terms and is legally obliged to comply with them if the packaging of the software is opened. For the applications with longer and more detailed licensing terms, the full text of the license is stored inside the box, and terms are considered to be accepted when a user first runs the software application. These licenses evolved into *electronic shrink-wrapped license*, that are accepted by users providing a yes/no answer to a question asking them to accept the licensing terms that are displayed once the software application is started [Wal96].

Since stand-alone computers are not connected via LANs or to the Internet, there is no possibility to remotely manage software licenses, to enforce compliance, to check if users conform to the license or to revoke the misused license. The shrink-wrapped license model fully relies on existing copyright law to ensure compliance and to prosecute users found in breach of the terms of use they accepted. This type of license grants the user unlimited usage and never expires, as it can not be managed after the medium has been sold. This often results in a higher application purchase price. It is important to note that, even though users have unlimited license for a lifetime usage of the application and own the medium with the application installation files, they do not own the application itself, only the right to use it. They do not own the right to copy or distribute the application, or to resell or give the application to other users. However, in the shrink-wrapped model, there is no method to physically prevent users from copying the application, distributing it to other users, and installing it on several devices. Even though this is often illegal according to the terms of use the user initially accepted, a large number of users either intentionally or unintentionally are breaking licensing terms. This resulted, and still results, in high piracy rates in the software industry and financial losses for software vendors. Estimates show that the largest piracy rates are related to the shrink-wrapped licensed software [Sof02]. Shrink-wrapped licensed software today is often combined with techniques that prevent users from copying (or backing-up) the content of the original CD to prevent piracy.

Today, shrink-wrapped licenses are mostly replaced by volume licensing, subscription and metered models for enterprise customers [KGS04]. Consumer software aimed at individual users, such as computer games, still sells mostly as shrink-wrapped packages [Orl04].

Serial Number License Model

Some software applications, during the installation process, require a *serial number* to be entered. The serial number is usually a combination of numbers and letters, and, if entered correctly, the installation is allowed to proceed. Initially this serial number was printed in the installation details that came with the application. The application verified that a valid serial number was entered, but since there was no connection to other computers or a central database of issued licenses, it was not possible to enforce that the application was only executed on a particular device or to revoke a license on an illegally installed application. This allowed illegal software copying and distribution. Serial number licensing is still used today, but due to the wide spread of the Internet connectivity and centralised database management of the issued licenses, this model has a means of enforcing the association of one serial number with a specific number of devices (see section 2.2.3).

Time-Limited License Model

A specified *time period* may also be used to restrict software licenses. It can be used to define the *demo* or *trial* version of an application in which a user is allowed to evaluate the application without payment for a certain period, usually 30 days. Another situation where a time period is used is to define a licensed usage period on purchased applications, for example one or two years. A time-limited model can be used on stand-alone computers because the only validation it needs is against the system clock. After a specified time period has passed, the time-limited license is revoked. Early implementations of time-limited licenses were easily abused by setting the system clock backwards to the date that was within the licensed time-period, after the license expiration. Today, time-limited licenses are able to detect tampering with the system clock and to revoke the license when tampering is detected. An example of a licensing application that enables distribution of software with time-limited models that detect clock-tampering is a PE Enforcer, developed by Cathonian Ventures Ltd [Enf04]. Time-limited licenses are still in use as they can efficiently limit a usage period on an application.

Feature-Based License Models

Simple versions of *feature-based licensing* have also been introduced for applications on standalone computers. A single software application can be released in a number of different versions containing different levels of features. These sets of features are aimed at users with different levels of application proficiency and needs. An example of a product distributed in this way is Altova's XMLSpy, that is released in enterprise, professional and home edition [Alt04]. These versions differ in price and therefore ensure that advanced users can pay to use all of the features they want, while allowing less advanced users to avoid paying extra money for the features they do not need or do not know how to use.

The other option for feature-based licensing is that an application is a distributed in a single package, but during the installation process, users are prompted to choose which version of the applications they wish to install. These versions are usually a minimal installation or beginner version, containing a minimum set of basic features; standard installation; and advanced/expert installation, containing the extra features that only expert users require.

These simple feature-based models are still widely used today, often combined with an additional licensing model that provides the enforcement of usage terms and prevent copying and multiple installations.

Site-Licensing Model

The *site-licensing model* for the licensing of applications was introduced to make the licensing of software for large enterprises and educational institutions more affordable. This model licenses all of the computers within a certain company or organization (on a site), or persons associated with it, to install and use a given application, or a set of applications from the same vendor, in an unlimited manner [UCD04].

Hardware-Based Licensing: Dongle

License compliance on standalone computers can also be ensured by using a small piece of hardware, a so-called *dongle*. A dongle contains the license key that the application needs to verify every time it is executed. A dongle needs to be attached to the designated computer port in order for the application to run. Some existing software licensing systems, such as FLEXnet, still support a dongle as a means of ensuring the license compliance [And98].

.NET Licensing

.NET licensing was introduced in 2002 with as a part of Microsoft's .NET Framework [msn04] for software development and is implemented on a single device without LAN, WAN, or Internet communication. Its basic implementation provided with the framework verifies the validity of the license by checking the registry value at the application runtime, but developers can extend it to support more sophisticated licensing models by extending the license manager. Enabling applications for .NET licensing also requires the insertion of the API calls to the license manager in the application source code [Scr04], giving a disadvantage of additional developer work and possible code tangling and scattering, that many of the existing license management systems presented in section 2.3 also have.

2.2.2 Local Area Networks and Enterprise Licences

Since the introduction of Local Area Networks (LANs), most enterprises and educational institutions connect desktops, servers, printers and other devices using a LAN. LANs enabled new ways of application and license delivery as nodes could obtain software and licenses from each other over the network. Also, software applications in LANs can reside on a single machine, and can be remotely connected to from other devices on the network.

New ways of enforcement have been introduced as well, as nodes within a LAN could all be contacted from a central server managing license information. Networks also enabled new ways of device identification by their IP address, host name, or by a unique MAC address of the network adapter. Additionally, users on a network typically have unique network login names or usernames that can be used as a means of identification.

These characteristics of a networked infrastructure resulted in a differentiation between the methods in which individual users and enterprises or groups are licensed.

Concurrent License Model

Before computers were connected by LANs, every computer within a company had to be licensed individually, even if only a few of the computers used the given application at any given time. This method of licensing resulted in enterprises paying for more licenses than they were actually using. To avoid this in the LAN environment, the *concurrent model*, also called *floating-license model*, was introduced. An enterprise can purchase a certain number of licenses for a network and the application can be installed on more computers than there are available licenses, as long as the number of simultaneous application users on the network does not exceed the number of purchased licenses. All licenses are stored on a server, and, prior to application usage, a client requests the license from a server. A client is granted a license as long as the number of issued licenses is not larger than the number of purchased licenses [Mac04d]. In this way, licenses are said to *float* on the network, since any computer within the network can use them. Often, the concurrent model requires the application to confirm license usage throughout the execution, and to return the license to the server upon application termination, for a license to be made available for other users. Therefore, this model requires TCP/IP connection for the duration of application execution. If the connection is lost and license usage is not confirmed with the server, the license times out, and application is terminated. Requirements for constant TCP/IP connection also create scalability problems on the licensing server, as the number of simultaneous connections the server needs to handle from devices using the licensed application can be very high in larger enterprises.

Node-Locked Model

Within a network, licenses can also be locked to a certain node, and only that node (or nodes) can use the application. This model is known as the *node-locked model*. A license contains information about the identification of the node, for example the node's static IP address, host-name, or MAC address of a network card. This licensing model contains an enforcement mechanism that ensures that an application can be executed only on the node that provides the identification specified in the license [Met04]. Execution of the node-locked licensed software does not require a network connection or the presence of a licensing server [Mac04d], however this model emerged only with the availability of LANs since computers in LANs have unique identification characteristics that licenses can be locked to.

Named-User Model

Within a LAN, a user generally has access to more than one computer within the network, but the user uses the same network login information on each computer. This enables the implementation of the *named-user* or *user-locked model*, which allows a given user, identified by the unique network username, to execute an application on any node within the network[Mac04d]. In the named-user model, an application can be installed on several machines and named users can run the application on any of them, or an application can be installed on a single server machine and only named users can remotely use the application. The latter method is often used in the licensing of enterprise server applications and databases accessed by multiple users. For example, Oracle provides a licensing model that limits the number of named users accessing the database[Dar03].

Per-CPU Licensing Model

Enterprise servers and databases used by multiple employees can also be licensed **per-CPU**, depending on how many CPUs are installed on the server running the application. This model depends on the processing power of the application host computer, and does not depend on the number of users accessing the application. For example, Oracle also supports this licensing model, and charges for database licensing based on the number of CPUs server running the application has installed [Dar03] .

Value-Based Models

Another license model introduced for enterprises is the *value-based model*. It does not directly benefit from LAN connections but is applicable for enterprises that wish to license several computers. A full software application is licensed to an enterprise for unlimited usage, but the actual price that the enterprise pays for the licenses differs based on the value enterprise can obtain from using the application, either by cost saving, or by generating profits. Value-based models can be based on the total revenue of the company, total number of employees in the company, or any other factor that value derived from the application can depend on [Wel02].

License Management Systems

The increasing complexity of enterprise licensing options has initiated the development of a new type of application in the software industry, known as license management systems (LMS). Up to this point, licensing code was part of the software application itself and was developed by the application vendor. Due to increased complexity of the licensing code, the licensing was separated into an application module by itself, developed separately from the original application, often by a different vendor, and later incorporated in the original application. Existing license management systems are discussed in more detail in section 2.3.

2.2.3 The Internet

As Internet connectivity became widely available, some vendors have started to require that all computers that have installed a given application must establish a connection with the vendor's licensing server to confirm the validity of their licenses. If a license is not valid, servers have capabilities to prevent product installation or to at least identify illegal users. This ability has strengthened the enforcement of licenses for installed software, and has introduced a new model for delivering application and software license by a download from a vendor's or application distributor's website.

License Activation

Serial key licensing has been strengthened by the *online license activation*, enabled by the wide availability of the Internet. The license key enables users to run an application a certain number of times during its *preliminary installation*. For a full final installation that allows unlimited usage, however, users need to obtain the activation key by phone or over the Internet. An example of products licensed this way are the Novell application suites [Nov04]. Activation can also be done automatically by connecting to the vendor's system via the Internet, as is the case with Manifold's products [Man01]. During activation, the vendor's software can create a non-unique hardware identification of a user's device based on hardware configuration at the time of activation [Mic04a]. If the same serial number is later reused to install software on other device, differences in the hardware configuration of the original device and the new device will be detected and installation will not be allowed.

Pay-per-Use License Models

Within LANs, licensing servers are able to log and measure the duration of actual application usage. With the availability of an Internet connection, this information can be transmitted to the vendor's licensing server that can bill customers only for the actual time that the application was used for. This licensing model is called the *metered* or *pay-per-use license model* [Mac04d], and it can be pre-paid or post-paid, depending on whether payment is settled before or after the actual application usage has taken place. In the pre-paid model, the customer pays in advance for a certain number of executions or time period of application usage. Usage information within the network is recorded and licenses are granted as long as the overall usage does not exceed the purchased number of units. In the post-paid model, information on the usage on all computers within the network is gathered at the server and sent to the vendor for the billing in set time intervals [Mac04d]. Pay-per-use models are preferred by enterprises that occasionally use an application, while applications which are used often and by many users within the enterprise are typically licensed by one of the flat-fee licenses, such as site licenses.

Subscription Model

Another recently introduced licensing model is *subscription*. Instead of paying large sums of money up-front for a perpetual license, customers pay monthly or yearly subscriptions for an application. A subscription license covers application support, updates and online tutorials during the subscription period [ITU04]. International Data Corporation (IDC) research shows that this model is suitable for both enterprise customers and software vendors, and predicts that software licensing will be shifting more towards subscription-based models in the future [KGS04].

2.2.4 Mobile Networks and Pervasive Devices

The further improvement of computing hardware and software led to the development of portable computers and the development of mobile phones into computing platforms. The number of mobile computing devices has been rapidly increasing over the past decade, and, in 2003, approximately 1.5 billion mobile phone devices were in use [For04], as well as 80 million laptops and tablets [Tec03], and 30 million PDAs [MB03]. These numbers are set to increase as the availability of wireless networks and the capabilities of wireless devices grow.

Mobile devices can be connected via networks, whether fixed or wireless. Mobile devices are occasionally-connected devices, i.e. in mobile networks, the on-demand connectivity present in desktop environments, cannot be assumed due to losses of network availability which are more likely to happen when devices are moving [GKN$^+$96]. Also, in the desktop environment, the frequency and amount of communication with the licensing server is not minimized by the system designers as the cost of the network traffic is insignificant. Conversely, the cost of data exchange over WAP or GPRS in the mobile environment is still reasonably high, so the frequency and amount of client-server communication required by a licensing system should be minimized [MH03]. However, devices in the mobile environment can use more economical short-range protocols to exchange data, e.g. Infrared and Bluetooth. This variety of available communication channels enables new mechanisms for software and license delivery, as well as the separate delivery of licenses, i.e. a license can be distributed via a different channel than the channel the application was distributed over.

Connectivity of devices via networks enables some of the mobile devices to be identified by their IP, host-name, or a MAC address. In the mobile environment, devices may have a unique identification even when they are disconnected from the network. A special case of mobile devices are mobile phones, that can additionally be identified by the IMEI number or a phone number. This unique identification can be used in developing new mechanisms to enforce licensing policies [Nok04a].

Another distinction of the mobile phone environment from desktop environments and other mobile environments is the type and number of parties involved in the software licensing process, and the degree to which devices in these environments are open for custom software installations. In the desktop environment, the software licensing policy is set between the vendor and customer. In the mobile environment, besides vendors and customers, wireless operators and mobile device manufacturers are also involved in the licensing process [BAH03].

Wireless device manufacturers pre-install the operating system (OS) and user interface (UI) on the device and they cannot be changed or customized by the users. The choice of OS and UI further influences the type of custom application software that users can install on the device, since mobile applications are developed for a particular OS/UI. Additionally, OS/UI vendors, which are often device manufacturers themselves, can deny custom applications the access to specific features of the hardware or OS, such as important security features [All01] or access to the information about hardware characteristics that can be used to identify devices, such as IMEI, phone number, or device model. This further limits the type and capabilities of the applications that mobile users can install on their devices.

Wireless operators control the wide area networks that mobile devices can connect to, and therefore control one of the important distribution channels for software applications and licenses. Wireless operators set the cost of the wide area network traffic and in that way directly influence the cost of the application and license delivery to the device over wide area networks. Many mobile application distributors depend on wireless operators to obtain payment for their applications, via mechanisms these operators have in place for the billing of other communication services.

While the desktop environment is open, development and delivery of the software as well as payment mechanisms are not restricted, and vendors can choose the mechanisms themselves, the mobile network environment is much less open, and to an extent controlled by device manufacturers, OS/UI vendors and mobile operators.

Another difference between mobile and desktop environments is consumers' attitudes towards the content and applications they are using on the devices. Customers are accustomed to obtaining the content on desktops for free but are used to paying relatively large sums of money for perpetual licenses for desktop software. For example, a typical PC game today costs about 30 Euro [Ama04]. However, on mobile devices, users are willing to pay small amounts for the content, but are not willing to pay larger sums for the applications, as a typical mobile game costs 3-12 Euro [Nok04a]. This attitude is reflected in sales of games for the Nokia phone and gaming platform N-Gage. Two types of games are available for N-Gage, downloadable Java games, and full price retail games on multimedia memory cards. Sales figures show that N-Gage is the most popular platform for downloadable Java games, which cost approximately 9 Euro each, however, sales of the full price games, costing over 35 Euro, are low [gam03]. The difference in price users are willing to pay for mobile games, when compared to desktop games, is partly due to the smaller set of available features due to device constraints, e.g. smaller screens, lower quality of graphics. The lower price of mobile software is also influenced by users refusing to pay large amounts of money for mobile applications, as they have a shorter life-span than desktop applications. Mobile devices are storage-constrained, and in order to install a new application, users often have to delete the older applications. Also, mobile applications are often developed for a single device model, and, once a user upgrades a device, the application cannot be used on the new device.

The new conditions, outlined above, that distinguish mobile environments from desktop environments, bring about additional changes to software licensing mechanisms. Similarly to the licensing in desktop and LAN environments, licensing in the mobile environment depends largely on the type of application, whether it is an operating system, middleware or application software, as well as if it is aimed at individual or enterprise customers.

2.2.4.1 End-User Licensing

Mobile phone networks have additional characteristics that distinguish them from other mobile networks. For example, devices can be identified by their IMEI and phone numbers. Also, in mobile phone networks, device manufacturers and mobile operators play a significant role in the licensing process. With this in mind, the licensing on mobile phone devices is discussed separately from the licensing for other existing mobile devices, such as PDAs.

Licensing of Software for Mobile Phones

Most of the existing consumer mobile software aimed at individuals is licensed by locking an application to a single device, either by hardware identification such as IMEI, network identification, such as a MSISDN, or by the prevention of forwarding of applications to other devices, as implemented by OMA DRM 1.0 [MA02].

Typically, software licenses are either paid for using a credit card, downloaded to a PC and transferred to a mobile device using IR or Bluetooth, or paid for by a premium SMS and delivered directly to a device over WAP. If an application is downloaded to a PC, its installation is restricted to a single mobile device [Nok04a]. Prior to the application download, the user is requested to enter either the phone's IMEI or phone number, and a license serial key is generated based on that number [DR04]. When the application is installed, a serial key is entered and it is verified that the IMEI/phone number that the serial key is based on and the actual IMEI/phone number of the mobile device match. If they do not match, the installation is aborted. This successfully prevents the misuse of applications, as long as the licensed user does not wish to change the device or a phone number. Once the user changes either the device or a phone number, the application can not be reinstalled on the new device, and cannot be executed on the same device with new phone number. This is a significant drawback as it represents inconvenience for licensed application users.

Several software vendors and wireless operators are introducing the *pay-per-play* model, wherein users have to pay a small fee every time they play a game. These games are played online, can be either browser-based or developed in J2ME, and are often multi-player games [nin04]. Similarly, browser games and online J2ME games can be offered via the *subscription model*, whereby users pay in advance for a certain period of unlimited application usage, usually monthly [Dig04]. However, currently, pay-per-use and subscription models are offered only for games that are played online and users cannot start the application without a network connection. IDC and DataMonitor research shows that subscription and pay-per-play models are currently the most popular licensing models for both vendors and customers and that the number of wireless online gamers will steeply raise over the next few years [Gre03].

The Open Mobile Alliance specifies additional flexible usage models for mobile applications that allow the installation of software on several devices using the same license and pay-per-use models. These models will most likely be implemented fully on the next generation of mobile phones [MA04]. More details on Open Mobile Alliance specification are provided in section 2.4.2.2 .

Licensing of Software for Other Mobile Devices

The licensing of software for mobile devices, other than mobile phones, for individual end-users, has many similarities to the licensing of end-user applications on desktops or within LANs. The means of device identification are the same and licensing mechanisms do not make the assumption of constant network connectivity. Software licenses can be perpetual, time-limited, based on a serial number or activation key, or locked to a unique MAC address of the device's network card.

Licensing of enterprise software, however, greatly differs in the mobile environment, since the assumption of constant network connectivity made by enterprise licensing systems in the desktop environment, cannot be made in the mobile environment.

2.2.4.2 Enterprise Licensing

The licensing of mobile enterprise applications requires a combination of mobile licensing techniques for individual mobile customers and license management systems from the desktop environment, as neither of these is on their own fully adapted to the licensing of enterprise applications in mobile environment.

Some license management systems for desktops include an enforcement architecture for mobile devices, such as FLEXnet [Mac03b], and requests from mobile clients are processed in the same manner as requests from desktop clients. This model is not suitable due to the previously described differences in nature of mobile and desktop networks, as it requires a client to have network access to the server in order to start an application. Details of the FLEXnet architecture and licensing techniques are provided in section 2.3.2.

Since mobile devices are constrained in memory and processing power, applications running on them cannot have all of the features and functionalities available to desktop applications. Often, enterprise applications on mobile devices are only mobile clients used to access or synchronize data with a company's central server. Since application functionality in these cases is dependent on the network connection, it is possible to integrate the license verification with the establishment of the network connection. However, for other applications that do not require a network connection, licensing should not depend on network availability.

Licenses for enterprise mobile software are most often sold as *volume licenses*. A minimum number of licenses must be purchased in order to avail of a volume price. For example, Intellisync's Handheld Editions for Enterprise requires a minimum of five licenses to be purchased [Int04]. This is similar to typical desktop volume licensing.

Often, prior to the full license purchase, users of enterprise applications are offered a free *trial license*. For example, Intellisync Handheld Edition for Enterprise is provided for a free 30 days trial [Int04]. A full license must be purchased to enable the application usage after the trial period has expired.

Floating network licenses are more difficult to implement in the mobile environment and there is no system fully implementing this model at the moment. XTNDConnect [Sys04] provides a very simple version of the floating license. If an application has not been used on a licensed device for 14 days, the license used by that device is returned to the server and made available for other devices.

Subscription-based licensing is also a common model in mobile enterprise software licensing. Companies pay monthly, either a flat fee or a certain amount per employee, for access to the server providing the service, e.g. Infowave's Symmetry Pro service provides enterprise access to emails and calendars from fixed networks that is paid per employee, on a monthly basis [Sym04].

Pay-per-use and feature-based models, as well as more advanced floating license models are not yet fully implemented in the mobile enterprise environment, so the development of license management systems on mobile devices is expected to move in this direction in the future.

Apart from mobile phones, laptops and PDAs, pervasive devices include other network addressable devices such as pagers, set top boxes, vending machines, microwaves and others. These devices are becoming more powerful, and as they become programmable platforms, software for these devices will become more complex and require its own ways of licensing. Future licensing architecture should account for the possibilities of software licensing on these devices as well.

2.2.5 Analysis of Licensing Models and Future Trends in Licensing

By analysing the changes software licensing has gone through from the standalone computers to mobile networks, a shift from packaged software, paid for once-off and up-front, toward the notion of software as a service, paid for based on actual usage and in regular intervals, is noticeable. This change is also reflected in the way licenses are enforced. At first, licensing relied on the honesty of the user and existing legislation to prevent misuse, while today hardware and software mechanisms attempt to enforce the license compliance.

Licensing and payment models have also changed through different computing environments. Even though traditional flat-fee licenses are still in use, customers prefer flexible models that measure usage and should have an option of switching to models that give them the most value at a given point in time. Also, as the complexity of the licensing models increases, customers can be overwhelmed by this complexity and, as a consequence, do not take full advantage of their entitlement or breach the license terms unknowingly. For this reason, customers should have tools that provide reports on their actual license usage compared to the usage they are licensed for [KGS04].

IDC report on software licensing from 2004 predicts that licensing in both the desktop and mobile environments is expected to introduce more flexible models in the future, simplify the understanding of the licensing policies and introduce the pay-per-use or metered pricing and licensing models. Also, report claims it is likely that increased revenue will be generated from the maintenance and support of applications while initial purchase price of the applications will fall [KGS04].

2.3 Software License Management Systems

2.3.1 Introduction

As already mentioned in section 2.2.2, the increase in complexity of license management code caused the software licensing code to separate from the application code and form into an application subsystem. Often, licensing policies are not implemented by application developers, and third party licensing software is used to provide licensing models for applications in the post-testing phase. These systems generally include the addition of the code to the licensed application, installation of the licensing clients on devices running the application, and the licensing server that keeps track of licenses distributed on the network and license usage. The license specification platform, distributed client applications and the licensing server are together called a license management systems (LMS). The LMS manages a license throughout its life-cycle, including license specification, generation, delivery, validation, enforcement and revocation. Key requirements for a LMS is that the addition of licensing code does not impose unnecessary work on the developer, that execution of the licensing code does not present inconvenience for the end-user, that a variety of usage models are provided, and that it is easily integrates with existing software within an enterprise, such as asset management, billing and payment applications. The following sections describe details of the most popular license management system used in commercial products today, FLEXnet [Mac04c], and a proposed standard for integrated software licensing architecture, XSLM [OG04], which enables all software within an enterprise to be licensed and managed from a single point.

2.3.2 FLEXnet

FLEXnet is a software license management platform developed by Macrovision that evolved from the license management application FlexLM. It is the most popular software license management application in desktop environment today. This application/platform is in use by over 800 enterprises, and over 3000 software applications are enabled to support FLEXnet licensing [Mac03a].

2.3.2.1 FLEXnet Architecture

The FLEXnet application consists of four main parts: the FLEXlm client library, the license manager daemon, the vendor daemon, and a license file, as represented in figure 2.1.

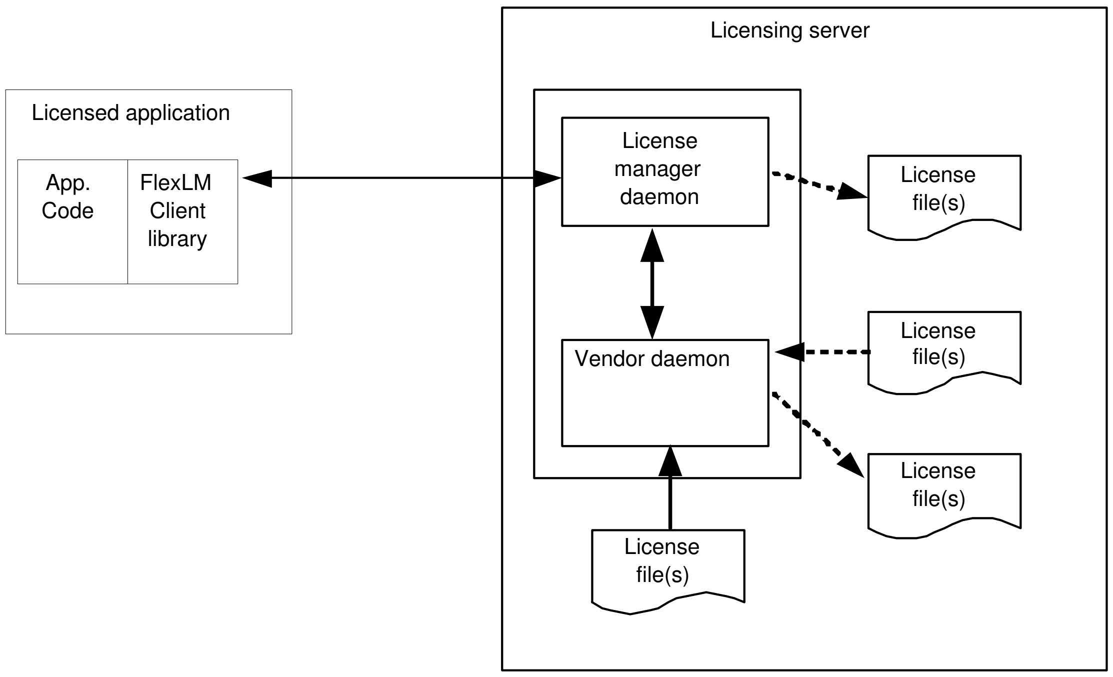


Figure 2.1: FLEXnet architecture [Mac03b]

FLEXnet-enabled applications are linked with the FLEXlm client library that provides the communication with the licensing server. This client communicates directly with the license manager daemon. The license manager daemon routes license requests to the appropriate vendor daemons. The vendor daemon is a process created for each vendor having a FLEXlm-licensed product on the network, and it keeps track of how many licenses are checked out and of the identity of current license users. If the vendor daemon determines that there is a license available for a requesting client, license use will be granted. Additional components of FLEXnet include the debug log file, the report file and the end-user administration options file [Mac03b].

FLEXnet supports over 1000 variations of licensing models including pay-per-use models, subscription, locked license models, floating model, time-limited, and value-based models[Mac04d].

2.3.2.2 FLEXnet and PARMA

PARMA is primarily designed for use in the mobile environment, where devices are only occasionally-connected and applications should be able to run even if the connection to the network is temporarily unavailable. Some FLEXnet licensing models, such as the concurrent model, require that the client application contacts the server to determine if there is an available license. If network connectivity at a given time is not available, a license will not be granted. PARMA needs to enable clients to use the application while disconnected from the network, and that is a major difference between PARMA requirements and the implementation of FLEXnet. Additionally, FLEXnet does not currently have client side implementations capable of running on any of the operating systems for mobile devices.

2.3.3 XSLM

XSLM stands for X-Open Software License use and Management System, and it is a proposal for a licensing standard defined by The Open Group [Ope04]. It defines a centralized license use and management system, and was introduced to solve interoperability problems among different license management systems, license file formats and operating systems. Licensing terms and conditions are expressed in an external license file stored on a server and separated from the program logic, allowing vendors to change licensing terms without changing the application. A licensed program executing on a client communicates with the licensing server in order to obtain a license[XSL99]. Although it solves the problems of interoperability, this standard introduces certain difficulties in implementation in pervasive environments as it requires the addition of licensing code to the application and it requires network availability throughout application execution. The additional code makes calls to the server to obtain the license, to confirm that the license is still in use during the execution, and to return the license after usage[XSL99].

2.3.3.1 XSLM System Architecture

The XSLM architecture is designed to integrate licensing for all of the software applications within an enterprise. Application vendors have to provide an implementation of specified licensing and management interfaces, and add calls to these interfaces to their application's source code. The client application makes a call to an application broker handling all of the licensing requests for the device. Based on the identification of the application that is making the call, the application broker routes the call to the application-specific agent that provides the above interfaces and implements license delivery, enforcement and revocation, as shown in figure 2.2.

2.3.3.2 XSLM Licensing Process

The first step of an XSLM-compliant application is to determine what level of application usage rights the server supports. Two levels are provided, basic and advanced. Next, it has to begin a rights enforcement session and request the rights. After these calls are successfully completed, the user is allowed to run the application. During execution, the application occasionally makes API calls to record application data, log the application usage, and confirm that the rights agreement is still in use. Before the application is closed, calls have to be made to release the rights in order to end the rights enforcing session. The XSLM-compliant advanced licensing system require rights objects to make at least eight API calls during the execution, as follows:

1. `xslm_query_api_level( )` - mandatory, must be called at least once, when the licensing server is accessed for the first time

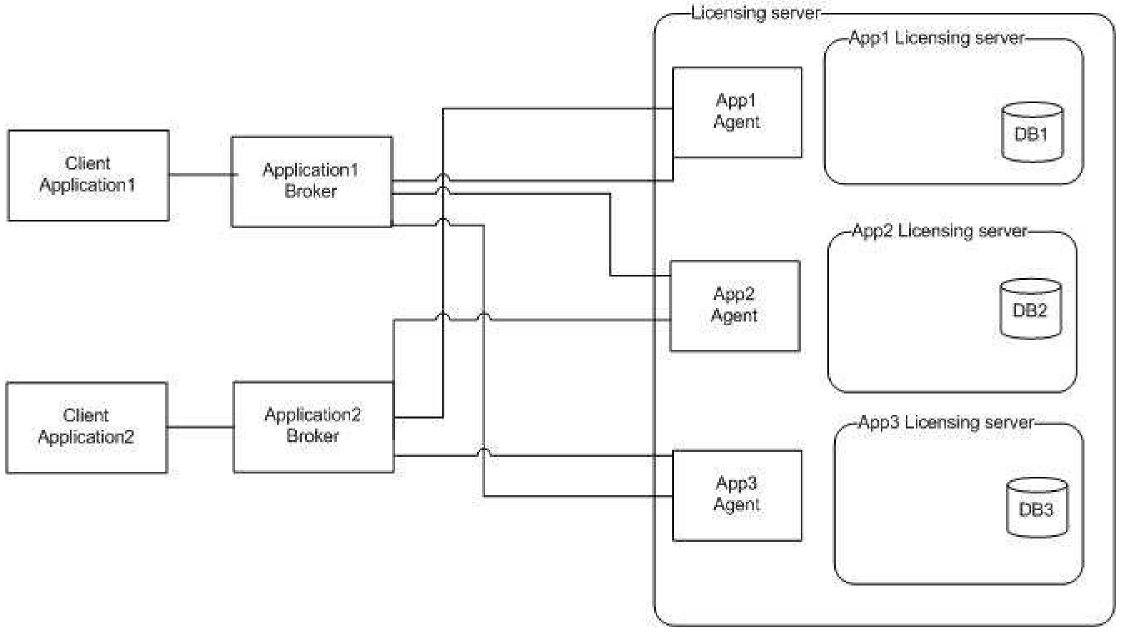


Figure 2.2: XSLM architecture

2. `xslm_adv_begin_session( ) -` mandatory, must be called in order to begin the license validation/enforcement session

3. `xslm_adv_request_license( ) -` mandatory, must be called in order to request the license from the server

4. `xslm_adv_confirm( ) -` mandatory, must be called during application execution to confirm the license is still in use and prevent timeout

5. `xslm_adv_log( ) -` optional, it logs the application specific messages

6. `xslm_adv_record( )-` optional, it records the application specific data

7. `xslm_adv_release_license( )-` mandatory, must be called to release a license for use by other users

8. `xslm_adv_end_session( ) -` mandatory, must be called to end the licensing session

The above calls cut across many different components of the application source code (on start up, during the execution and before application termination) and inserting these calls into application source code requires knowledge of the licensing system APIs provided by the licensing server [XSL99]. The licensing models that the XSLM-compliant architecture supports depend on the specific vendor implementations and are not defined by the XSLM standard.

2.3.3.3 XSLM and PARMA

XSLM was considered as a basis for the PARMA project but it was discarded due to the high average number of licensing calls a client must make to the server during application execution. In the desktop environment, due to the constant connectivity, numerous calls to the server are generally reliable and do not incur a cost overhead. However, in the mobile environment, bandwidth is costly and a network connection is not always available, which makes the high number of calls prohibitive. No commercial license management system has yet been developed to fully meet XSLM standard although it was first proposed in 1999 [XSL99]. This indicates low adoption of this standard even though it is the only existing software license management standard. PARMA's objective is to be based on widely adopted standards for compatibility purposes.

2.3.4 Comparison of Reviewed License Management Systems

As presented in the previous sections, there is a single dominant license management system present in the market, FLEXnet, and only one proposed standard for implementing inter-operable license management architectures, XSLM. Neither of them is suitable for rights management on mobile devices, as they require a network connection for the validation of the usage rights. Also, as table 2.1 shows, both of these license management architectures require additional programming to enable applications for distribution and licensing using the particular architecture. Additional source code needs to be added to the original application, requiring programming knowledge and knowledge of the API specifications of the given architecture.

	progr. knowledge	knowledge of API	changes to src-code	# of models requiring developer action
FLEXnet	yes	yes	yes	all
XSLM	yes	yes	yes	all

Table 2.1: PARMA vs other LMS

2.4 Digital Rights Management and Rights Expression Languages

2.4.1 Introduction

While software license management systems often enforce license compliance by requiring validation by a central licensing server, digital rights management implements the enforcement architecture on the device itself and does not require a licensing server. Licensing terms are expressed in a machine-readable file written in one of the rights expression languages. A client-side enforcement architecture reads the file and is configured according to its content to ensure compliance and to revoke the license if illegal use is detected. DRM systems generally provide the tool that generates DRM files compliant to the particular expression language used, but are not concerned with delivery of license and content to the device or license updates. Traditionally DRM was concerned only with content but today mobile applications are treated in a similar manner to content [MA02]. Consequently, DRM techniques and the state of the art in DRM technologies influenced the design and development of the PARMA architecture. The following few sections explain DRM and rights expression language basics and analyse existing rights expression languages in research and industry today.

2.4.1.1 DRM

DRM stands for Digital Rights Management and by definition represents management of the creation, manipulation, distribution, and consumption of digital or physical information. It is a technology that protects information by preventing its illegal copying, by imposing fees and by processing payments for the information [HYP02]. It is important to make a distinction between *digital rights management* and *management of digital rights*. DRM manages rights on non-digital content as well, however, the rights are expressed and interpreted in a digital way[Ian01]. The first generation of DRM focused on preventing the illegal copying and distribution of content by implementing security and encryption features, but most of the technologies failed due to the ease of exchange and copying of the digital information (e.g. DVD copy protection, peer-to-peer music sharing programs). The most recent generation of DRM systems have broader capabilities and are able to express various usage models. Forwarding and copying of the content is also enabled, but access to the copyrighted content is controlled and allowed only to the users and devices owning the valid permissions [Ian01].

DRM systems need to provide at least two architectures: functional and information architecture.

Functional architecture deals with asset creation, management and usage. Asset creation ensures validity of the rights on content creation from existing content, on rights assignment to the newly created content, and on processing of the content through review and approval work-flow. The asset management is concerned with obtaining content from the creators, storing it and enabling access to it together with its metadata. It enables assignment of rights to the parties who purchased content and processing of the payments for that content. Asset usage model deals with enforcement of the usage rights agreements. It enables usage of content only in the agreement with the rights user owns and it monitors the asset usage if the rights specific user owns require so [Ian01].

Information architecture is generally implemented by using a specific Rights Expression Language (REL). RELs are discussed in more details in the following section.

Traditionally, DRM and software license management were separate areas as DRM was primarily concerned with content. However, as new DRM systems emerged they also proved capable of expressing various usage models on the software applications. Some DRM systems are currently used to manage access to software applications, primarily on mobile devices, e.g. OMA DRM [MA02].

2.4.1.2 RELs

Rights Expression Languages (RELs) are the basis of existing DRM systems and provide a means of expressing access and usage rights on content and applications. RELs need to be machine-readable and easily exchangeable between systems and platforms which has resulted in all of the today's RELs being expressed in XML. A REL contains a valid syntax and some semantics. The syntax represents the grammar rules that a REL has to follow and is expressed in a XML schema. The semantics represent the meaning of valid expressions in the language and are defined by data dictionaries [Gut03]. In general, all RELs have a similar structure; they have the notion of usage rights, assets (content or application) and parties involved in defining rights on asset usage. Currently, there are several competing RELs in the DRM industry and the major ones will be covered in the next sections.

2.4.2 Existing RELs and DRM Systems

2.4.2.1 XrML/MPEG-21

XrML, an eXtensible Rights Markup Language, is a XML-based rights expression language defined by ContentGuard [Con04a] and based on Digital Property Rights Language (DPRL) invented at Xerox PARC [Xer04]. Microsoft's DRM system [Wat01] implements this specification and it is a basis for MPEG-21 [MDSG02]. Xerox patented XrML as a "System for Controlling the Distribution and Use of Digital Work Having Attached Usage Rights Where the Usage Rights are Defined by a Usage Rights Grammar" [MDP01], and it includes the patent for using the RELs to express usage rights attached to content. So far ContentGuard/Xerox has let the community use XrML royalty-free, but the DRM industry is reluctant to adopt this standard due to the concern that in the future royalties will be requested to use XrML [Wat01] (as ContentGuard has been recently purchased by Microsoft, Time Warner and Thomson [For05]). Development of XrML has been frozen at the 2.0 version and MPEG-21/5 is considered to be a 2.1 version of this language [xrm04].

The Moving Picture Experts Group (MPEG) REL (MPEG-21/5) standard is based on the XrML proposal and is accepted by the International Standards Organization (ISO) as a standard [DS04]. Even though not many implementations of MPEG REL exist yet, it is in the process of being standardized and it has strong support from industry, from vendors such as ContentGuard and Microsoft. The conclusion is that MPEG REL is likely to be one of the dominating rights expression languages in the future.

MPEG REL Structure

The MPEG REL consists of four main elements: principal, right, resource and condition. *Principal* is an entity to whom usage rights are granted; *right* is the activity or action the principal is allowed to exercise against some resource; *resource* is a digital object which the principal can be granted usage rights on; *condition* is one or more conditions that must be met before the principal is allowed to exercise rights on the resource. These elements combine to express usage rights files or licenses [PPD04]. MPEG REL schema can define usage rights on various types of content but on applications as well, as rights such as play, read, write, and execute are defined. Using the defined conditions, the following usage rights models can be specified: node-locked, pay-per-use, time-limited, metered, user-locked, count, and subscription. There are no elements that support the expression of audit-based or feature-based usage rights models [Con04b, MPE04].

Tool Support

ContentGuard provides RightsExpress [Con04b], an online tool for creating, modifying and validating MPEG REL files. Files generated can only be used for non-commercial purposes. The tool is designed to clarify MPEG REL syntax and semantics to help its wider acceptance and use, and not to be used in commercial products.

MPEG REL and PARMA

MPEG REL is primarily aimed at content rather than software applications and it does not support fine-grained feature-based usage rights models. Also, it is aimed at desktop rather than mobile environments. These are the core reasons that the PARMA project decided against basing its architecture on MPEG REL. A full analysis of MPEG-REL capabilities and comparison with PARMA is detailed in the evaluation chapter 6, after full PARMA REL has been introduced.

2.4.2.2 OMA DRM

Open Mobile Alliance [OMA04] Digital Rights Management standards are currently the basis for most DRM systems used on mobile phone devices [Let04]. All of the major players in the mobile industry are part of this alliance and include members from all parts of the mobile value chain, including mobile operators (Vodafone, Orange, T-Mobile), IT Companies (Symbian, Oracle, Sybase), wireless vendors (Nokia, SonyEricsson, Siemens, Motorola), and content providers (The Walt Disney Company, AOL) [Nok02]. Also, payment providers such as Visa and Mastercard are members of this alliance [Nok02]. Such a strong membership base indicates that standards introduced by this organisation will continue to dominate the mobile DRM industry. The currently implemented standard is OMA DRM 1.0, but specifications for OMA DRM 2.0 are published as a release candidate and is planned to be implemented in the year 2005.

Usage rights are expressed using OMA DRM REL.

OMA DRM REL

OMA DRM REL is a subset of ODRL, a rights expression language that is discussed in section 2.4.2.3. OMA DRM REL adopts the specification of assets from ODRL, as well as the specification of permissions and constraints on the usage of those assets. Usage rights models that can be expressed using the features of OMA DRM REL version 1.0 are time-limited and counted. OMA DRM REL version 2.0 adds elements to express additional usage rights models, namely node-locked, metered, subscription-based and user-locked. Usage rights can be expressed both on various content and applications, as permissions include play, execute, display and print. OMA DRM REL does not support specification of the parties involved in the licensing process as ODRL does.

OMA DRM 1.0

Version 1.0 describes three ways in which rules and rights on media (content or applications) can be defined and enforced. It separates the concepts of media and usage rights. Ownership of the media does not necessarily mean ownership of the usage rights for that media, and rights can be purchased and delivered separately.

These three ways to delivery media, and state and enforce usage rights are as follows [MA02]:

1. Forward-Lock

This method prevents the forwarding of the media from the device. The only way content can get to a device is through a legitimate purchase and download, therefore this method prevents any unauthorized usage of the content. The method used to implement forward-lock on the mobile devices is by hard-coding the device to enable or disable forwarding of the content of certain mime-types. There is no need for a rights object to be delivered to the device as enforcement is fully performed by the device's hardware[Let04]. The problem with this approach is that it imposes unwanted difficulties for the rightful owner of the usage rights for content. For example, users who purchased an mp3 file and downloaded it to a mobile device would like to listen to that file on a PC or an mp3-player as well, but forward-lock prevents them from transferring the file to other devices [MA02].

2. Combined Delivery

In a combined delivery two objects are delivered to the device at the same time, the content object and the rights object. The rights object contains permissions and restrictions on content usage. Content, similarly to forward-lock, cannot be forwarded.[MA02].

3. Separate Delivery

Separate delivery is similar to combined delivery but has additional security features. Two objects are delivered, the content and the rights objects, but they can be delivered using the separate channels. The content object is encrypted and the rights objects contains a key to decrypt it. The content object can be forwarded but content cannot be decrypted and used without purchasing a new rights object containing the key [Nok04c]. This model enables super-distribution of the content, and the download of the application over free distribution channels such as Bluetooth.

Tool Support

Nokia provides a Content Publishing Toolkit [Nok04b] for the generation of OMA DRM 1.0 compatible files. It supports forward lock, combined delivery and separate delivery models, and packages content and rights object in format suitable for download to a mobile device. There are no limitations on use of the tool for commercial and non-commercial purposes.

OMA DRM 2.0

Version 2.0 of OMA Digital Rights Management has an emphasis on security and more flexible usage models. The REL was extended to support new constraints to enable metered usage model and named-user usage model [MA04]. Rights objects are individually encrypted using the device's public key. The device and rights issuer mutually authenticate before the transactions, multi-cast and unicast streaming are secured, and new usage and business models are introduced, such as metered time, subscriptions, and gifting. Users are able to share the content between the several mobile devices belonging to them or within closed groups, e.g. family or office. Also, users are able to transfer content to other devices, such as a PC, supporting DRM [Buh04]. However, OMA DRM 2.0 specification is not yet implemented on any of the mobile devices.

OMA DRM and PARMA

OMA DRM supports J2ME applications but it only supports usage restrictions listed above, i.e. count, time period, and interval. The PARMA model requires finer control over the application, including execution of particular methods or features and execution only by an identified user. Therefore, PARMA is not designed to use OMA DRM standard, but PARMA schema is based on a superset of OMA DRM standard, ODRL, introduced in the next section.

2.4.2.3 ODRL

ODRL is the rights expression language that the PARMA REL is based on and therefore will be explained in more detail than other existing rights expression languages.

Introduction to ODRL

ODRL was designed by Renato Iannella in IPR Systems in 2000 [Ian02]. Currently, version 1.1 is in use and version 2.0 is planned for release in the year 2005. ODRL stands for Open Digital Rights Language and defines a machine-readable expression of usage rights on digital and physical content. ODRL does not provide mechanisms for enforcement of the usage rights, only means of their expression [Ian02].

ODRL expression language

ODRL defines an XML schema that rights files need to validate against as well as a data dictionary defining specific elements within the usage rights file. The ODRL expression language components are shown in figure 2.3. Three foundation elements are *assets*, *rights*, and *parties*. Asset refers to a uniquely identified digital or physical content. Rights define usage rights on the object and include *Permissions*, *Constraints* and *Conditions*. Permissions define the actions a user is allowed to perform on the asset (e.g. play and print), constraints define limitations to those actions (e.g. date/time limitations and counters), and conditions define conditions that need to be met in order for permissions to become valid (e.g. payment and registration). Parties include parties involved in the content distribution and usage, i.e. end users, also called asset consumers, and rights holders, who are parties holding the ownership of the content [Ian02].

The above elements can be combined to express *Offers* and *Agreements*. Offers represent the proposals from rights holders for specific Rights over their Assets. Agreements express that Parties entered into contracts or deals based on specific Offers.

Most entities in the model can support a specific *Context* element. A Context is relative to the entity and describes further information about that entity or the relationship between entities. ODRL can also express conditions under which given Rights can be revoked [Ian02].

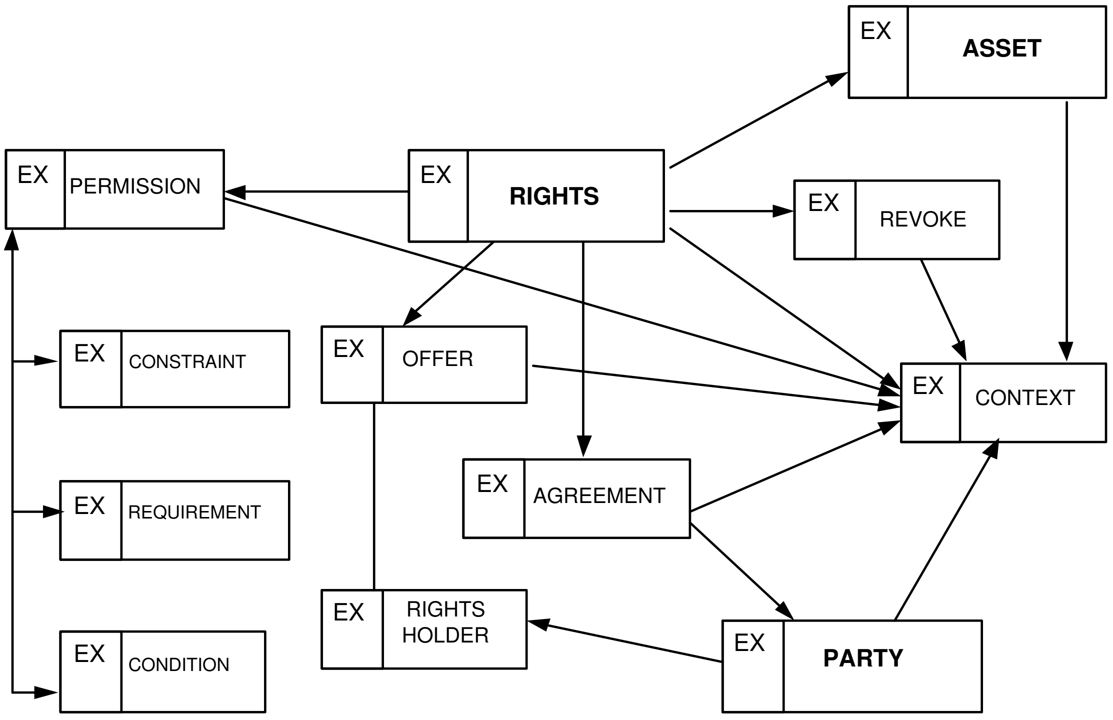


Figure 2.3: ODRL components [Ian02]

ODRL data dictionary

The ODRL data dictionary (DD) defines permissions, conditions, constraints, requirements and context elements that can be used within the ODRL rights file. ODRL is primarily aimed at content delivered to desktops, so a lot of the elements defined in ODRL are only applicable to a desktop environment. However, there is nothing in ODRL that prevents it from being applied to a mobile environment and to the software applications. The permissions ODRL file can specify over a certain content include display, print, play, execute, install, uninstall, and save. These permissions are very similar to XrML defined permissions. Requirements that can be defined in order for permissions to be valid include pre-pay, post-pay, per-use, and register. Presence of pre-pay, post-pay and per-use elements enables implementation of flexible payment policies. Permissions can be constrained by user or hardware characteristics, environment, purpose or duration of usage. This variety of constraints enables support for various license types and usage rights models. Usage rights models for applications that ODRL supports are concurrent, pay-per-use, time-limited, metered, user-locked, subscription-based, and counted.

ODRL context elements identify the entities, such as parties or assets, and provide additional information about them, such as id, location, name, and version. Full ODRL schema and data dictionary containing lists of all of the permissions, requirements, conditions, constraints and context elements, are provided in appendix D.

ODRL does not define any of these to be mandatory, but choice of elements used depends on the implementation and enforcement architecture [ODR02].

ODRL example license

ODRL files are expressing usage rights in XML validated against the ODRL schema. The example below shows the content of a sample rights usage file for an application in a desktop environment.

```
<?xml version="1.0" encoding="UTF-8"?>
<o-ex:rights xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<o-ex:agreement>
  <o-ex:asset>
    <o-ex:context>
      <o-dd:dLocation>www.some-app.com</o-dd:dLocation>
      <o-dd:name>Application1</o-dd:name>
    </o-ex:context>
  </o-ex:asset>
  <o-ex:party>
    <o-ex:context>
      <o-dd:dLocation>www.some-provider.com</o-dd:dLocation>
    </o-ex:context>
  </o-ex:party>
  <o-ex:permission>
    <o-dd:execute>
      <o-ex:constraint>
        <o-dd:datetime>
            <o-dd:start>2004-09-01</o-dd:start>
            <o-dd:end>2006-09-01</o-dd:end>
        </o-dd:datetime>
      </o-ex:constraint>
    </o-dd:execute>
  </o-ex:permission>
  <o-ex:requirement>
     <o-dd:peruse>
      <o-dd:payment>
        <o-dd:amount o-dd:currency="EUR">2.00</o-dd:amount>
      </o-dd:payment>
     </o-dd:peruse>
   </o-ex:requirement>
 </o-ex:agreement>
</o-ex:rights>
```

This ODRL file defines usage rights on the asset called *Application1* located at URI *www.some-app.com*. The provider of this application has the URI *www.some-provider.com*. The only permission allowed by this file is *execute*. ODRL specifications say that only listed permissions are considered allowed. Execution of this application is constrained by the date of usage. Application can be executed only between September 1st 2004 and September 1st 2006. The requirement that has to be fulfilled in order for execution to be permitted is that 2.00 EUR is paid for each execution of this application.

ODRL Evaluation

ODRL elements support various usage models. Additional elements can be introduced by extending the ODRL schema. All the extensions should be backward compatible with the original ODRL. As mentioned earlier in this chapter, Open Mobile Alliance defined the OMA DRM 1.0 version as a subset of ODRL. All the OMA DRM 1.0 files are compatible with ODRL implementations as they contain only elements that belong to a subset of ODRL. ODRL also supports super-distribution model which enables reselling of the content, and it supports XML digital signatures for authenticity and security of the ODRL files.

ODRL has been submitted to W3C (World Wide Web Consortium) but W3C pointed out a few changes ODRL should implement in order to be considered for inclusion in the list of W3C Activities [WW02]. The main objections include lack of the *not* and a wildcard (*) operators. ODRL states that no permission should be assumed unless it is explicitly stated as a granted permission. Therefore, in the case that the user should have all the possible permissions, it is not practical to list all of the possible permissions. This is where a wildcard should be used to express all of the permissions. Similarly, if the user is entitled to all but one permission, instead of listing all the allowed permissions, it is more convenient to list only one permission the user is not entitled to, and use the *not* operator to express that the permission is not allowed [Wen04].

Tool Support

There is no publicly available tool that supports specification and generation of ODRL files.

ODRL and PARMA

ODRL proved to be a solid basis for meeting the PARMA requirements. Support for flexible usage and payment models is provided as well as support for XML security features. ODRL is a parent set of OMA DRM that is currently the most widely used REL in the mobile environment, but has additional features required by PARMA that OMA DRM does not include. This is why ODRL was chosen to be the base REL for PARMA implementation. However, ODRL does not have all the elements PARMA requires, specifically elements used in feature-based flexible application licensing model and in licensing of applications for J2ME mobile platform. Therefore, ODRL needed to be extended to support these additional PARMA requirements. Details of elements PARMA REL adopted from ODRL and details of new extensions are provided in chapter 3.

2.4.2.4 Comparison of reviewed DRM systems and RELs

When existing RELs were considered for the expression of usage rights in PARMA, several of their characteristics were considered, as summarised in the table 2.2. No language fully satisfied the PARMA requirements; XrML was patented, XrML and ODRL lack elements suitable for mobile devices and OMA DRM is not extensible without affecting the compatibility with its parent REL ODRL. Also, extensions to OMA DRM would have to be extensive and overlap with elements that ODRL already incorporates, making ODRL a more suitable choice for building PARMA REL on.

	open spec.	patented	content	applications	extensible	tool support	desktop	mobile
XrML/MPEG-21	x	x	x	x	x	x	x	
OMA DRM	x		x	x		x		x
ODRL	x		x	x	x		x	

Table 2.2: Comparison of reviewed RELs

The capabilities of existing RELs to express various usage rights models PARMA needs to support were considered as well. As table 2.3 shows, ODRL supports the widest range of the models, but still lacks elements for expressing node-locked, audit-based and feature-based models.

	Node	Concur	PrePay	PostPay	Audit	Time	Meter	User	Feature	Subscr	Count
MPEG-21	x		x	x		x	x	x		x	x
OMA 1.0						x					x
OMA 2.0	x					x	x	x		x	x
ODRL		x	x	x		x	x	x		x	x

Table 2.3: Comparison of application usage models that reviewed RELs support [MDSG02, MA02, Ian02]

2.5 Other Related Work

The research community is working on further improvement of the license management and digital rights management systems. Several of them are related to the area of pervasive application rights management as they're aimed at mobile content or applications, or they make use of the REL used in PARMA.

Guth, Neumann and Strembeck designed an enforcement architecture based on digital contracts expressed using ODRL. An interpreter is provided that transforms ODRL file into generalized Contract Schema, that could also be interpreted from other RELs to provide interoperability. The access control component is configured by this contract schema and grants or denies rights based on it [GNS03].

HP's Perry and Williamson describe a licensing solution for digital content aimed at Pocket PC's. Their model encourages and rewards super-distribution, as opposed to disabling forwarding as many of existing systems do. Temporary licenses obtained by super-distribution are cancelled on the first following connection to the Internet when the license bank determines that a user owns only a temporary license and prompts the user to purchase a full one. However, if the user prevents this connection, the temporary license would be time unlimited. Licenses are expressed using XML tags, but do not conform to any standardized RELs [HP04].

Santos, Pereire and Silva developed a prototype of DRM framework for mobile J2ME applications. There is no enforcement architecture on the device itself, but prior to the application download additional code containing licensing algorithm is added to MIDlets (J2ME applications) . Additional code is executed at the application start up, and execution is aborted if it is determined that license is not valid. However, actual license validation and enforcement code is not covered in the prototype and should be provided by a provider of a DRM solution [SPS03].

2.6 Summary

This chapter presented an overview of the history of software licensing and discussed the licensing models used in different computing environments, i.e. stand-alone computers, LANs, and mobile environments. The development of software licensing in these environments is characterised by strengthening of the enforcement mechanisms and increases in flexibility of the application usage terms and license payment models. Initial licensing models relied on legal implications of the license misuse to enforce the compliance, resulting in high piracy rates. Today, application vendors have software and hardware mechanisms in place that attempt to enforce the compliance. Some of the first software licenses entitled users for unlimited lifetime usage of the application, while recently more flexible licenses have been introduced, such as time-limited, node-locked, and user-locked. License management systems were also discussed, in particular FLEXnet LMS and proposed license management standard XSLM. Although existing LMSs are suitable for desktop environments, they are not suitable for deployment in pervasive environments, as they rely on on-demand network availability. LMSs generally require the addition of the licensing code in the original application source-code, adding extra responsibilities for application developer. DRM systems and RELs were also discussed, but DRM is concerned primarily with content, although recently it has been used to express usage rights on applications. Existing RELs generally lack elements that allow the expression of flexible application usage rights models. For this reason, existing RELs are not considered adequate to express PARMA usage rights files.

Chapter 3

PARMA REL

3.1 Introduction

The PARMA REL is developed as a part of this thesis to express a variety of flexible usage rights models on mobile and desktop software applications. The PARMA REL extends ODRL REL adding the elements required for the identification of mobile devices and the identification of users, as well as elements that improve flexibility of application usage models. The PARMA REL includes the full ODRL 1.1 schema, however, the PARMA architecture is not designed to support all of the ODRL elements. ODRL elements specifically defined for handling content cannot be used in a software environment and are not supported by PARMA. The following sections describe ODRL elements supported in the PARMA REL and custom PARMA REL elements.

3.2 Usage Rights Models

A usage rights model is a set of permissions and constraints defining a legal usage of a certain content or an application. A usage rights model in a DRM system is equivalent to the licensing model in a LMS. Throughout this dissertation, a usage rights model is referred to as a *usage model*, or a *model*.

PARMA is designed to support various software usage rights models, including the traditional ones adopted from standalone PC environments or fixed network environments as well as more flexible models, such as feature-based and audit-based models. The PARMA REL does not limit the set of constraints and conditions that can be expressed in a single usage file, and enables the combination of usage right models to enable a more fine-grained definition of usage rights. For example, a single usage model can be limited by time, named user, and have limited set of features.

PARMA supports the following models:

- unlimited usage model
- named user usage model

- time-limited usage model
- feature-based usage model
- subscription-based usage model
- pay-per-use usage model (real-time or audit-based)
- node-locked usage model
- counted usage model
- metered usage model
- concurrent usage model

These models and the mechanisms they use for enforcement in the existing LMS and DRM systems are described in the previous chapter. Usage rights models for desktop environments are implemented in a similar way to the implementation of these models in the previously described architectures. In the unlimited usage model, no constraints on the execute permission are specified; in the named user model, the usage rights file contains the identification of a single user owning the usage rights; the time-limited model allows unlimited usage up to a certain date; the subscription model allows unlimited usage of application(s) provided by a given provider or service during the subscription period, which may be renewed monthly or yearly; the node-locked model enables the unlimited usage of software on a uniquely-identified device; and in the concurrent usage model, unlimited usage is allowed as long as the maximum number of simultaneous users or devices is not exceeded. Counted and metered models are implemented in a similar manner to their implementation in the DRM systems, specifically ODRL.

Two models provided by the PARMA REL that are not available in the existing usage rights architectures that are suitable for mobile devices are audit-based pay-per-use usage rights model and the feature-based usage rights model.

PARMA implements two types of pay-per-use usage rights models: real-time and audit-based. In the real-time pay-per-use model, the user pays for application usage, either directly before or after application usage, depending on the implementation of the payment system which is out of the scope of this dissertation. PARMA REL adopted the ODRL elements that support both pre-paid and post-paid pay-per-use, and payment can be based either on the duration of the application usage or the number of times the application or a certain feature is executed.

The audit-based pay-per-use model enables users to use an application for a certain period, accumulate the usage information, and pay for past usage at the end of the defined period. PARMA REL defines an element that specifies the information about application usage that should be logged. Logs are stored in a secure storage area, and at the end of the period transferred to the server where the usage information is used to determine payment amounts. It is essential that this information represents the actual application usage, and that abuse or tampering of the data is prevented. If the usage data is tampered with, or not transferred to the server at the end of given period, the usage rights are not extended and the application can not be executed until correct usage data is sent for billing.

The feature-based model is a usage rights model that allows certain features within the application to be blocked from execution, or for billing to be based on the number or duration of execution of certain features. It differs from other usage models because usage rights need to be checked throughout the application execution, not only at application start up. In implementations of feature-based models in existing LMS, additional source code containing licensing API calls needs to be added before and/or after source code implementing a given feature in the original application. These API calls require a network connection to validate the usage rights with the licensing server, making the implementation unsuitable for mobile environments as an on-demand network connection cannot be guaranteed. Also, the insertion of these calls should ideally be removed in desktop environments as well, in order to minimize the modifications to the original applications. PARMA REL provides the feature-based model implementation suitable for both desktop and mobile environments, aiming to improve its implementation in the desktop environment and extended it to mobile environment. In PARMA REL, the features are defined in a usage rights file using the PARMA REL pointcut elements and mapped into Java aspects that are compiled into the original application to validate usage rights when these features are executed. This validation should require no network connection as the details of the licensed features are stored on the device, and no modifications are performed on the original application as licensing is implemented using AOP.

The existing PARMA architecture implements support for the above listed models, however the PARMA REL can also be extended to support additional models as necessary.

3.3 ODRL Elements Adoption and Extensions

The PARMA REL extends the full ODRL schema so all of the ODRL elements will validate against the PARMA schema. However, since PARMA deals with digital rights on applications, the PARMA implementation does not support ODRL elements dealing specifically with content. The full list of elements adopted from ODRL is provided in this section. Appendix D contains full ODRL schema and data dictionary.

3.3.1 Adoption of Existing ODRL Elements

PARMA adopts five basic ODRL elements that every usage rights definition contains: rights, offer, agreement, asset, and party [Ian02].

- *rights* - root ODRL/PARMA element. It defines usage rights on the application and is expressed using permissions, conditions and constraints.

- *offer* - proposal from one party (rights holder) to another party (user) for certain usage rights on an asset.
- *agreement* - contract/deal between parties about usage rights on an asset.
- *asset* - in ODRL it represents any digital or physical content. Since PARMA deals only with software applications, in PARMA architecture an asset represents a software application.
- *party* - entity involved in usage rights issuing. In the PARMA architecture it can be either the end-user, enterprise customer, software vendor or software distributor.

Each of these five elements can have child elements. PARMA adopts child elements from four groups: permissions (children of the agreement element), requirements (children of the agreement element), constraints (children of the permissions element) and context (can be a child of all of the above elements).

- *permission* - ODRL defines numerous permissions on content and applications. PARMA is concerned only with the DRM of software usage rights so *execute* is the only permission that makes sense in the PARMA environment and it is adopted from ODRL.
  - *execute* - permission that allows the execution of the software application that usage rights are defined for. ODRL rules state that no permission on an asset is granted unless that is explicitly stated. Since the only permission PARMA usage rights files can express is *execute*, this permission has been made mandatory in PARMA usage rights files. If this permission is not present, usage rights file does not make sense because it defines no permissions over an asset.
- *requirement* - set of conditions that have to be met before stated permission is granted to a user. Requirements include elements for specification of the payment amounts.
  - *accept* - requirement that refers to a user accepting terms of use defined by the rights holder.
  - *register* - requirement that a user registers with a certain rights holder's service before exercising the permissions.
  - *prepay* - requirement that a user pays for usage rights prior to the execution of the application.
  - *postpay* - requirement that a user pays for usage rights after the execution of the application.
  - *peruse* - requirement that a user pays a certain amount per execution of the application, in order to obtain the rights on execution. The user can be billed for the execution before or after the execution.

Prepay, *postpay* and *peruse* requirements are expressed as ODRL's feeType element type, and they can either express a payment amount or a percentage of total payment. Also, feeType expresses the currency for the payment.

- *constraint* - represents certain restrictions on the execution of the software application.
  - *count* - states the number of times the application can be executed.
  - *datetime* - represents a range expressed in date and time. It can be expressed with min and max values, or with the fixed date. It defines that a user is entitled to executions only on a certain date, or between certain dates. The range can be open-ended as well, meaning that a user cannot execute an application before a certain date, but can do so anytime after; or that user cannot execute an application after a certain date, but can do so any time before it.
  - *accumulated* - represents the maximum amount of metered usage time a user is entitled to. It contains both values and units for metrics conforming to the ISO8601 standard for date and time formats [W3C97]. For example, it can express that a user is allowed to execute the application for up to a maximum of certain number of hours.
- *context* - contains additional information about any of the PARMA entities. It can be a child of the agreement, offer, asset or party element.
  - *uid* - contains unique identification code for agreement, application or party.
  - *dLocation* - represents the digital location of the agreement, application or party. It must be expressed as an URI. When present within the asset element, it uniquely identifies the software application, and as such is a mandatory PARMA element.
  - *name* - name of the agreement, application or party.
  - *version* - version of the entity. In PARMA, this is specifically used to distinguish between different versions of software applications.
  - *date* - date and time related to an agreement, application or party. It can, for example, represent the date on which agreement was agreed or the date of application release.

3.3.2 Custom Extensions

The PARMA REL contains additional elements that needed to be added in order to support more flexible usage rights models. The additional elements PARMA adds to ODRL are: RightsType, ServerType, AuditLogData, IMEI, UserID, DisabledFeatures, and Pointcut. These elements are added as extensions to context, constraint and requirement elements. New context elements are introduced to specify additional information about the usage rights model and usage rights server. Constraint elements are extended to support node-locked, user-locked and feature-based usage rights models. An additional requirement element is introduced to enable support for the audit-based pay-per-use usage rights model. Following subsections describe the custom PARMA REL elements and their usage. Appendix A.1 contains the full PARMA REL schema with definitions of all of the custom extensions.

3.3.2.1 Context Elements

PARMA REL introduces two new context elements to extend the ODRL schema, RightsType and ServerType.

- *RightsType*

The RightsType element is defined as a contextType element, and it can contain the following values: NodeLocked, Concurrent, PayPerUse-Audit, PayPerUse-RT, Unlimited, TimeLimited, FeatureLimited, NamedUser, Counted, Metered, and Subscription.

These are the rights types currently supported by the PARMA architecture, and every agreement issued by PARMA will belong to one of these categories. However, RightsType is open for extensions and the list of values it can contain is not fixed in the schema. Since a license model is directly related to offer and agreement types, the ideal case would be to make this element a mandatory child element of the context element that is a child of agreement or offer type. However, ODRL uses the same definition of the context element regardless of its parent element, and therefore it cannot be defined as mandatory. This validation must be performed separately from the validation against the schema.

- *ServerType*

PARMA-enabled applications can obtain their usage rights and process payments through two types of servers, the EDS and the VDS. Information about the server is stored in the usage rights file, so that the client enforcement architecture knows the location and type of the licensing server that issues and updates the usage rights. Alternatively, depending on the client side implementation, this information can be specified in the configuration of the client architecture. Context, as defined by ODRL, has an element dLocation which PARMA uses to specify the URI of the server. Context, however, does not have any elements that can be used to define whether this server is of the EDS or the VDS type. Therefore PARMA extended the ODRL schema with the ServerType element. This element is a child of the context element and can contain only two possible values, EDS or VDS.

Because the server belongs to a rights holder party, PARMA needs to restrict the ServerType element to occur only within the context element that is a child of a party element. This validation, however, cannot be done in schema because ODRL defines context as a child of any entity. Therefore, this validation cannot be performed against PARMA REL schema, but needs to be performed separately from validation against the schema.

3.3.2.2 Constraint Elements

PARMA REL introduces several new constraint elements to support the range of the flexible usage rights models. The new elements are IMEI, introduced to support the node-locked model, UserID for user-locked model support, and DisabledFeatures and Pointcut, for feature-based model implementation.

- *IMEI*

IMEI stands for International Mobile Equipment Identifier and is a unique identifier of a mobile phone device. It can be used to lock the application usage to a single device identified by the IMEI, and in PARMA this restriction is used to implement the node-locked usage rights model.

- *UserID*

The UserID constraint is introduced to support the user-locked usage rights model. Users can be uniquely identified by a user ID obtained when registering with the PARMA system, by their mobile number, or by their name. UserID element also contains the payment type a user selected at the registration, in order to group the users by the payment type, and restrict usage rights only to a group of users using a certain payment type. All of the elements that are a part of the UserID element can be used to restrict the permission in the user-locked usage rights model. UserID is an extension of constraint ODRL type and consists of four child elements: name, registeredID, mobileNumber and payType.

- *disabledFeatures*

The DisabledFeatures element is an extension of the constraint element and consists of zero or more features listed as strings.

It enables vendors to specify a list of application features to be disabled during the application execution. For example, for a game application, a multi-player over Bluetooth option might be disabled for a demo version of the application. This possibility, however, is not implemented in the existing version of the PARMA architecture. In order for it to be implemented, the system needs to provide a mapping between the feature and the sequence of methods or events that constitute that feature. The mapping can easily be stored within the client architecture, but specifying the mapping needs to be done by the developer/vendor, and requires additional steps in application testing and submission.

- *pointcut*

Pointcut refers to the aspect-oriented programming term representing the location in an application's source code where an execution should be intercepted to execute some additional code. A pointcut in PARMA REL is an extension of the constraint type and represents a full method signature. It also states whether additional code should be executed before, after or around the specified method. Additional code should validate and enforce the usage rights and either prevent the method from executing or log the information about the executed method.

The pointcut element consists of the adviceType and package elements. AdviceType specifies the location in relation to the method where additional code should be executed, and it is limited to three values: before, after and around. Package describes the method as defined in the target object-oriented programming language. It consists of sub-packages, class name, method name, method return type, and list of zero or more types of method parameters. For a definition of the pointcut element refer to a full PARMA schema in appendix A.1.

The following example of the PARMA REL code expresses that the execution should be intercepted before the method *myMethod* that takes *int* as its only parameter, has *myReturnType* as a return type, and belongs to the class *myClassName* within *myPackage*:

```
<parma:pointcut>
 <parma:adviceType>before</parma:adviceType>
  <parma:package>
   <parma:packageName>myPackage</parma:packageName>
   <parma:class>
    <parma:className>myClassName</parma:className>
    <parma:method>
     <parma:methodName>myMethod</parma:methodName>
     <parma:returnType>myReturnType</parma:returnType>
     <parma:argType>int</parma:argType>
    </parma:method>
   </parma:class>
  </parma:package>
</parma:pointcut>
```

Pointcuts enable the implementation of a flexible feature-based model. Some common strategies for implementation of the feature-based model are to terminate the application when the execution of a particular method specified in pointcut is attempted, to log the information about execution of particular methods, or to prevent a method or a feature from being executed, while allowing the remaining features in the application to be used.

Based on the presence of the pointcut element in the usage rights file, application usage rights are either checked only at the application start up, or before/after every method listed in the usage rights file. If no pointcut is specified, usage rights are checked at the entry point to the program, i.e. in a J2ME program before method `startApp()`. If at that point it is determined that a given user/device does not have usage rights for a given application, the execution of the application is terminated.

3.3.2.3 Requirement Elements

- *auditLogData*

This element is introduced to support the audit-based pay-per-use model.

The AuditLogData element contains three attributes: dateTime, duration, and accumulated, each of which represents the type of information relating to the application execution that should be logged . All of the attributes are defined as boolean data type, and can either be set to true or false, noting that a given attribute should be or should not be logged. The default value for attributes is 1, which is equivalent to true, meaning that information should be logged.

Logged usage data is stored in a secure storage area, external to the usage rights file, and is used for determining payment amounts for audit-based usage rights model.

3.4 Additional Validations

The PARMA usage rights file needs to be validated against the PARMA schema, but there are also some additional requirements which the usage rights file must satisfy in order to be fully compliant with the PARMA architecture. This is an additional step in the usage right creation, but is integrated into rights file generation tools, so it does not impose any extra work on developers. However, if PARMA files are specified using third-party tools or by hand, it is important to check for these validations.

The additional validations required are:

- The RightsType element is mandatory, and can occur only as a child of a context element that is a child of an agreement or offer element. If this element was defined as mandatory in the schema, it would mean that it must appear everywhere where the context element appears. However, that is not desired as it is mandatory only in the agreement or offer context, not in the context referring to a party or asset. Therefore this validation needs to be performed outside of the schema. In the PARMA implementation it is performed in the usage rights generation tools.
- The dLocation element is mandatory, as a child of the context element that is a child of asset element. As it is the unique identification of the application usage rights, it must be present. However, it could have not been made mandatory in the schema since, similarly to RightsType, it is part of the context that can occur within elements other than asset.
- The ServerType element is mandatory in the party context, and that is the only place where it can occur. It cannot occur in the context element within any other element.
- The IMEI element is mandatory if the RightsType element contains the value 'NodeLocked'. The IMEI element uniquely identifies the node, and if this element is not present, the usage rights file would not define the node to which the usage rights are locked to.
- The Count element is mandatory if the RightsType element contains the value 'Counted'.
- The accumulated element is mandatory if the RightsType element contains the value 'Metered'.
- The auditLogData element is mandatory if the RightsType element contains the value 'PayPerUse-Audit'. The audited usage rights model can be implemented only if usage information is logged, so it is mandatory to specify what type of information about execution should be logged.
- The userID element is a mandatory constraint if RightsType contains the value 'NamedUser'. A user who is allowed to execute the application needs to be uniquely identified by values in the userID in order for the named user model to be implemented.

- The register element is a mandatory requirement if the RightsType contains the value 'NamedUser'. A user in a usage rights file is uniquely identified by user id, which is obtained when the user registers with a given server.

The additional validations could have been avoided by specifying certain elements within the schema differently, and the schema validation alone would be enough to confirm that files are compliant with PARMA. However, if the schema is to be defined in this way, the original ODRL schema structure would need to be changed, and the new REL would no longer be compliant with ODRL and OMA DRM. A decision was made to keep the compliance so that PARMA can process a usage rights file created in other DRM systems and vice versa. Therefore, the need for additional validations is a consequence of retaining the PARMA interoperability with OMA DRM and ODRL.

When designing PARMA REL, the number of mandatory elements was kept at the minimum, to minimize the size of the usage rights files. Usage rights file are stored on the client device, so limiting the size of these files is important to support the mobile devices with constrained memory and storage capabilities.

3.5 ODRL Compatibility and PARMA REL Extensibility

PARMA REL extends ODRL, which itself is a parent REL of OMA DRM REL. Therefore, PARMA usage rights files are backward compatible with both of these standards. A PARMA-compliant system will process ODRL and OMA DRM usage rights files. Elements that belong to the subset of ODRL that PARMA adopted are processed just as PARMA elements are, i.e. permissions would be granted with given restrictions. Additional permissions that are not recognized by PARMA are ignored, even though they might be otherwise valid ODRL elements. Depending on the client side implementation, PARMA can either choose to ignore the unknown requirements and constraints, or have permissions automatically invalidated due to unknown requirements/constraints. ODRL behaves similarly when encountering a PARMA file; recognized permissions are granted, the rest of the permissions are ignored. Recognized constraints, conditions and requirements are enforced, but the presence of the unrecognized requirements results in the permission being denied. This compatibility is important because PARMA-enabled or ODRL-enabled applications are not tied only to the devices implementing one of these RELs, but they can also be copied from one device to another and permissions would still be valid. Being able to transfer applications legally purchased from one device to another is one of the crucial requirements users consider when purchasing an application and selecting a vendor/provider.

3.6 Summary

PARMA REL is a REL developed to express PARMA usage rights. It is extended from the existing REL, ODRL, that is on its own suitable for content and desktop environments, but lacks features specific to mobile environments and fine-grained application usage rights management. PARMA REL supports ten usage rights models: unlimited, node-locked, concurrent, time-limited, feature-based, subscription, real-time pay-per-use, audit-based pay-per-use, counted, and metered usage rights models. Apart from the validation against PARMA REL schema, PARMA usage rights files need to satisfy an additional set of requirements in order to be valid, primarily to ensure the presence of mandatory PARMA elements. PARMA REL is backward compatible with ODRL and OMA DRM, and additional validations were introduced to achieve support for various usage rights models while retaining this compatibility.

Chapter 4

Architecture and Design

4.1 Introduction

PARMA is a pervasive usage rights management architecture designed for both consumer and enterprise applications running on both desktop and mobile devices. It is designed for two types of software vendors: individual developers or small companies providing simple consumer applications, and medium/large vendors providing enterprise applications. Consequently, two types of customers can use PARMA: end-users who download mobile applications for entertainment, personal productivity, and travel assistance, and enterprise customers who distribute the application to their employees as a tool for their everyday business activities.

4.2 Client Requirements

The two different categories of application vendors have different requirements for the distribution and usage rights management of their applications, generally dependent on the size of the vendor and application.

Small companies or individual developers that develop a smaller number of simpler mobile applications distribute them through existing mobile application portals, e.g. Handango [Han04], network operators, e.g. Vodafone [Vod04] or device manufacturers, e.g. Nokia [Nok04a], as they do not have the facilities to market and distribute the application themselves. The usage rights management architecture chosen by a distributor needs to be simple to use in order to make it cost-effective for smaller vendors to distribute their application through this channel.

Medium and large vendors prefer to have more control over the application and usage rights distribution, and they have the facilities to distribute the application and to host the usage rights server within their company.

Different types of mobile software users also have different expectations from a usage rights management architecture. End-user customers want the process of obtaining the application to be as convenient and as cost-effective as possible. Once they have paid for and downloaded an application together with its usage rights, they do not want the usage rights management system to in any way interfere with the normal application usage [Cla02]. The rights management system should not slow down the start up time of the application due to the usage rights validation, require any additional user interaction such as authentication, or prevent forwarding of the application to other device(s) belonging to the user. Users want to be able to run their applications during times of network unavailability [Cla02], for example in remote areas without network coverage. Also, users prefer flexible subscription or pay-per-use models [KGS04]. The legal means of obtaining the application and usage rights should be affordable and convenient enough to discourage users from obtaining illegal application copies.

Enterprise customers should be able to manage the distribution of software and the usage rights within their company themselves. They want the usage rights to be expressed in a simple and understandable way, that will make it clear whether they are compliant with the granted usage rights or not. Enterprises also want to be able to produce reports on past application usage, and on remaining usage rights [Cla02]. The usage rights management system for enterprise customers should be easily integrated with other enterprise services, such as payment systems and asset management. A variety of usage rights models should be offered so that different employees within the company can get the application with different usage rights models based on their application usage habits and needs. Payments for the applications and usage rights should not be large up-front payments, but small amounts spread over time, either by implementing a subscription or a pay-per-use model [KGS04]. In common with individual customers, enterprise customers do not want usage rights management system to interfere with normal application execution, as any unreasonable usage rights denial or slow down of the application execution will have a negative impact on employee productivity.

4.2.1 Design Goals

PARMA needs to meet all of the above requirements of vendors and customers as well as requirements imposed by the nature of a pervasive computing environment outlined in the previous chapters. PARMA aims to improve the areas where existing license management and usage rights management systems fail in pervasive computer environments. From this, major design goals and design guidelines were compiled.

Main design goals:

1. To support a variety of both simple and fine-grained usage models, in particular feature-based and audit-based models.

2. To remove the need for the addition of proprietary API calls to the usage rights architecture in the original application source code by the developer.

3. To provide an integrated platform for usage rights management for both mobile and desktop devices

within enterprise environment.

Additional design guidelines:

- minimize the size of the client side of the usage rights management architecture.
- minimize the WAN communication traffic required by the usage rights management systems, in order to minimize cost.
- do not assume network availability for normal application execution
- be compatible with existing standards in rights management where possible
- make the usage rights models simple enough to understand and use but complex enough to support fine-grained usage rights models
- take advantage of the new identification means available in the mobile environment, e.g. IMEI, phone number

Scope

This dissertation does not aim to implement the full licensing life-cycle, as presented in the introduction chapter in figure 1.1. Client side enforcement and revocation of the usage rights are not implemented, only the licensing stages where usage rights servers are involved. These stages are:

- usage rights specification
- usage rights file generation
- integration of usage rights and licensed application
- usage rights file delivery
- server-side usage rights enforcement
- server-side usage rights revocation

4.3 Architecture Overview

The complete PARMA architecture, illustrated in figure 4.1, supports two different types of DRM servers to support two different types of potential clients.

A vendor DRM Server (VDS) is hosted by the application vendor, and is managing usage rights for the application(s) that the vendor has developed. The VDS can also be hosted by the software distributor who is distributing applications and managing usage rights on behalf of many software application providers. The VDS can distribute applications to enterprise customers or directly to end-user clients wishing to use any of the applications hosted. The VDS enables application developers to avoid distributing their applications through prohibitively expensive closed distribution channels controlled by device manufacturers and network operators. In PARMA, application and license download can be done without the involvement of device manufacturers or network operators, and licensing is settled only between the vendor or distributor and a customer.

An Enterprise DRM Server (EDS) is hosted by the enterprise client who downloads applications and usage rights from the VDS server and distributes them to its employees. The EDS manages application and usage rights distribution within the enterprise. The vendor hosting the VDS is not concerned with how the applications and usage rights are distributed within the enterprise. By introducing the EDS, the enterprise is able to download the application and a bulk of usage rights for the application from the VDS, and host them on the EDS. The EDS further distributes the individual usage rights and application to the enterprise employees, without the involvement of the VDS.

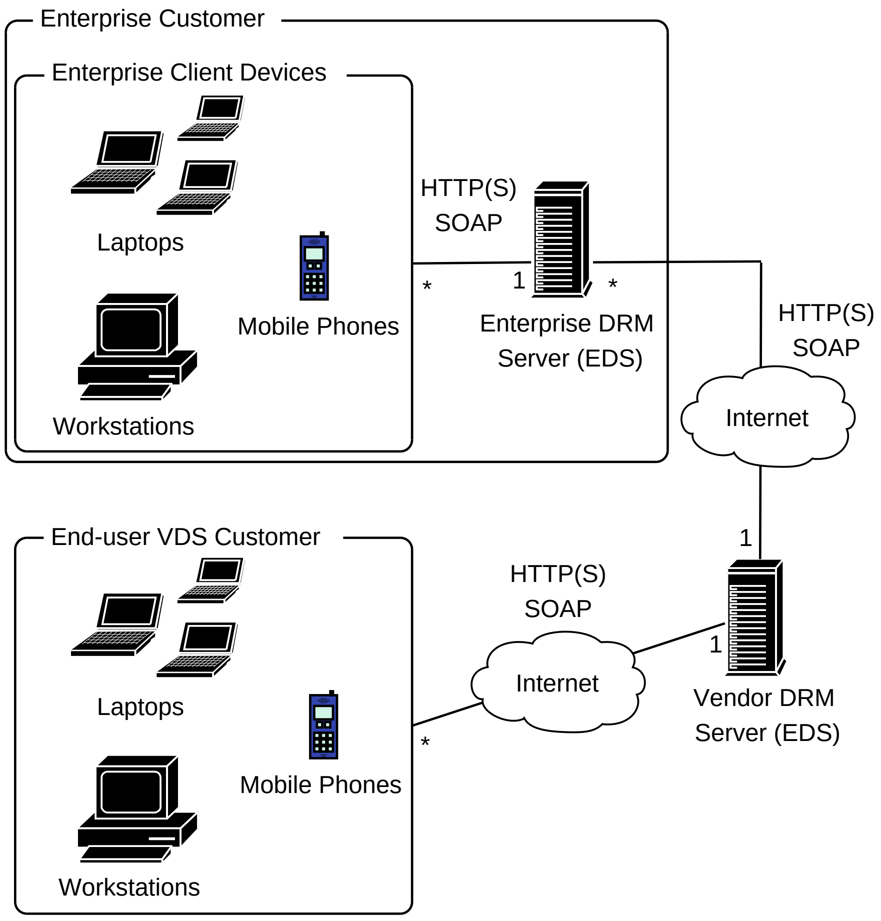


Figure 4.1: PARMA architecture

The EDS and the VDS are implemented as web services, and usage rights generation tools are implemented as servlets. Clients can invoke these services via HTTP(S) [W3C99a] and SOAP [W3C03]. Any device supporting either of these protocols can download applications and usage rights from the servers. Examples of devices that support these protocols and can be PARMA clients are mobile phones, PDAs, laptops, and desktops.

The language and platform independence of web services enables PARMA to meet one of the main design goals, that is to create an integrated platform for management of all software applications within enterprise environment regardless of the platform applications are running on or the language applications are developed in.

Since web services act as middleware, using web services enables the integration of various applications regardless of the language or platform the application is implemented on and regardless of where it is located. Web services enable the integration of PARMA with other existing enterprise services, such as payment or asset management.

4.4 Use Cases

There are four main PARMA usage scenarios:

- An application developer or a vendor wants to register and upload an application for distribution.
- An enterprise customer wants to obtain application and usage rights for an application on behalf of its employees.
- An employee wants to obtain application and usage rights from the enterprise server.
- An individual customer wants to obtain application and usage rights from an application vendor/distributor.

4.4.1 The Application Developer

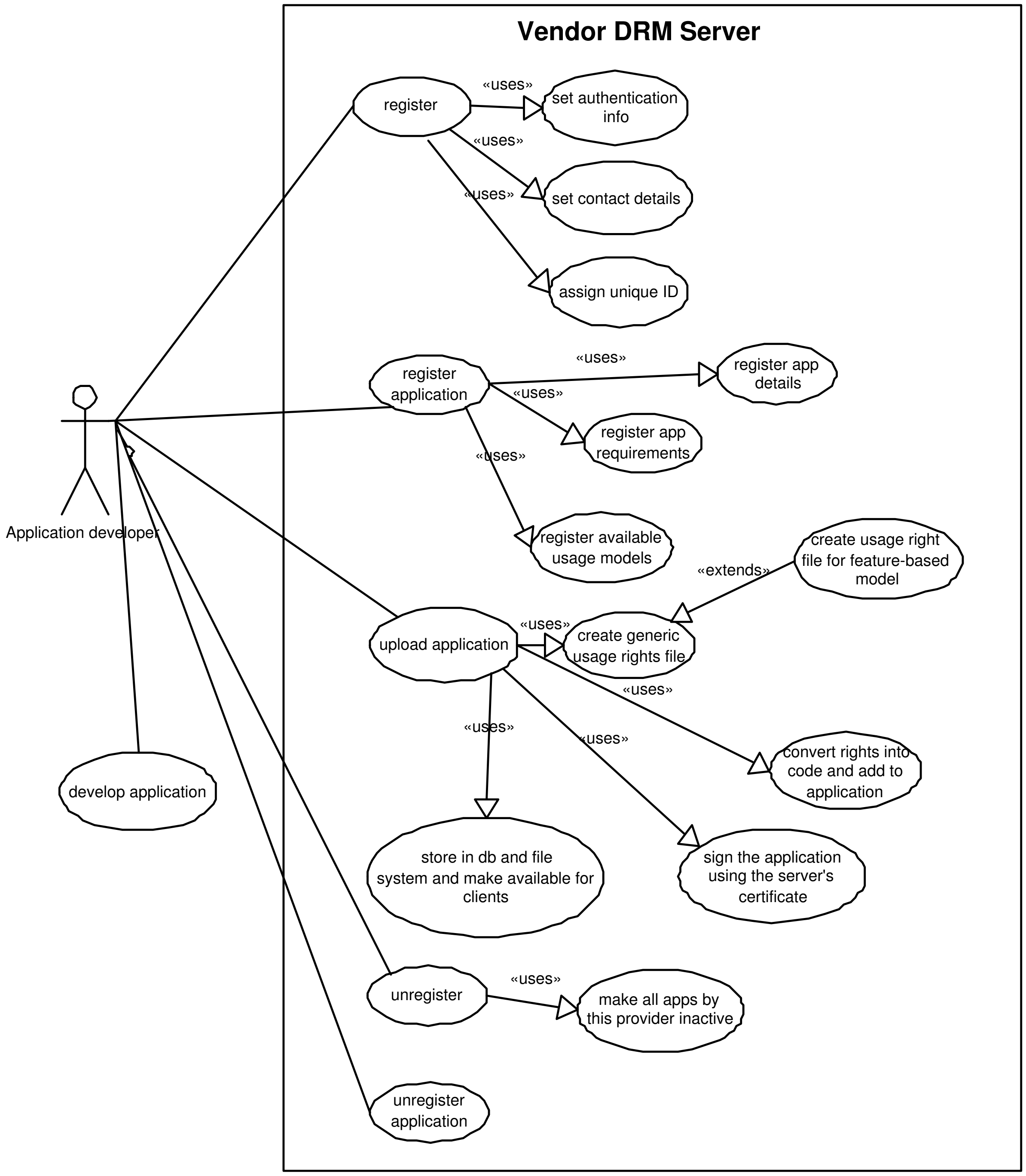

Vendor DRM Server
register
«uses»
set authentication info
«uses»
set contact details
«uses»
assign unique ID
register application
«uses»
register app details
«uses»
register app requirements
«uses»
register available usage models
create usage right file for feature-based model
«extends»
upload application
«uses»
create generic usage rights file
«uses»
convert rights into code and add to application
«uses»
«uses»
store in db and file system and make available for clients
sign the application using the server's certificate
unregister
«uses»
make all apps by this provider inactive
unregister application
Application developer
develop application

Figure 4.2: Application developer. Interaction with the VDS.

Application developers need to register their details with the PARMA architecture in order to distribute their applications using the VDS. The registration process includes the registration of authentication information and contact details, and it results in the assignment of a unique identifier generated to identify all PARMA clients, both developers and customers. Identifiers are unique within a single PARMA system. If a client registers with several EDS or VDS servers, each registration results in an assignment of a different identifier, unique within that system.

After registering their details, developers and vendors can register applications for distribution through PARMA. Full registration of the application includes registering the application details, system requirements and the usage rights models that vendors wish to offer for this application. After registration, the developer uploads the application and the application is enabled for distribution through PARMA. This process includes the generation of generic usage rights, the generation of usage rights enforcement code, the compilation of an application and digital signing of the application.

A full list of activities application developers can perform at the VDS is illustrated in figure 4.2.

4.4.2 The Enterprise Client

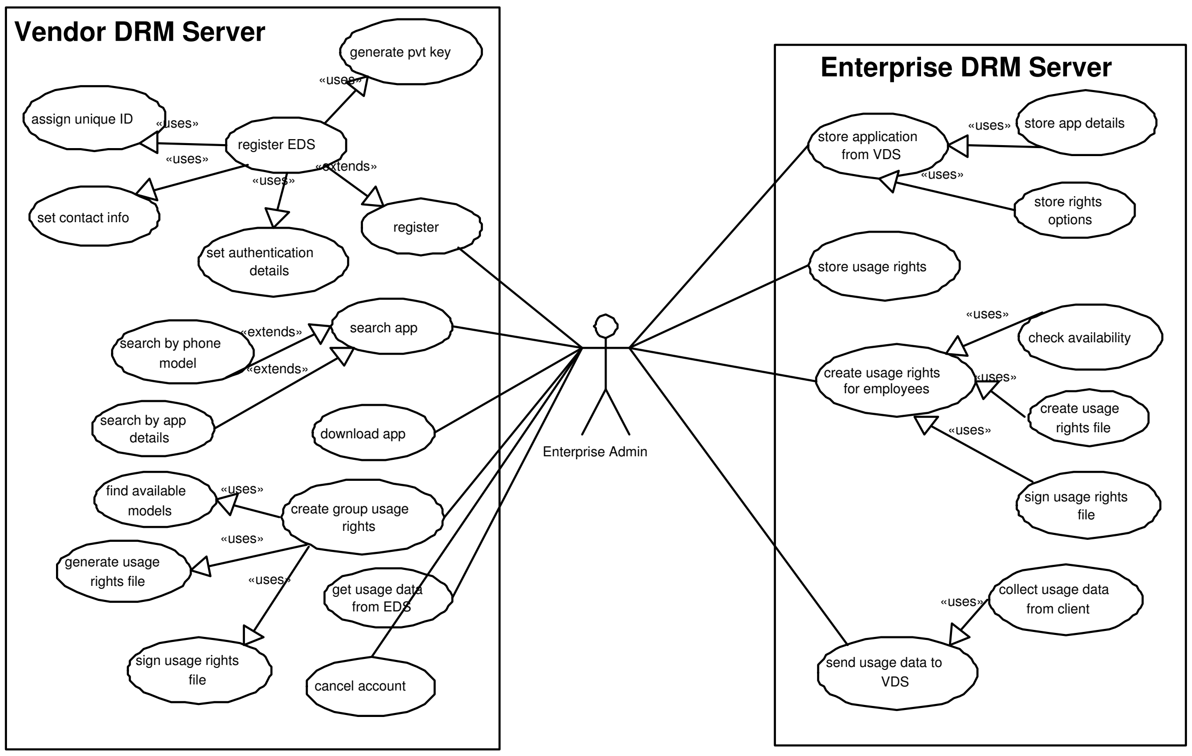


Figure 4.3: Enterprise client. Interaction between the VDS and the EDS.

Enterprise clients are companies who wish to purchase application and usage rights from the vendor, and distribute them to enterprise employees.

Persons assigned to handle application usage rights management for an enterprise customer interact with both the vendor's DRM server and the enterprise DRM server hosted by the enterprise customer, as shown in figure 4.3. They register the enterprise as a VDS client and are able to browse and search through the available applications. Once they have found a suitable application, they are able to download it, select the usage rights options and download the group usage rights file for their company. The application and usage rights downloaded from the VDS are stored on the EDS, and used to generate the usage rights for individual employees. There is also a facility for an administrator to initiate the sending of the usage data gathered on the EDS to the VDS for billing.

4.4.3 The Enterprise Employee

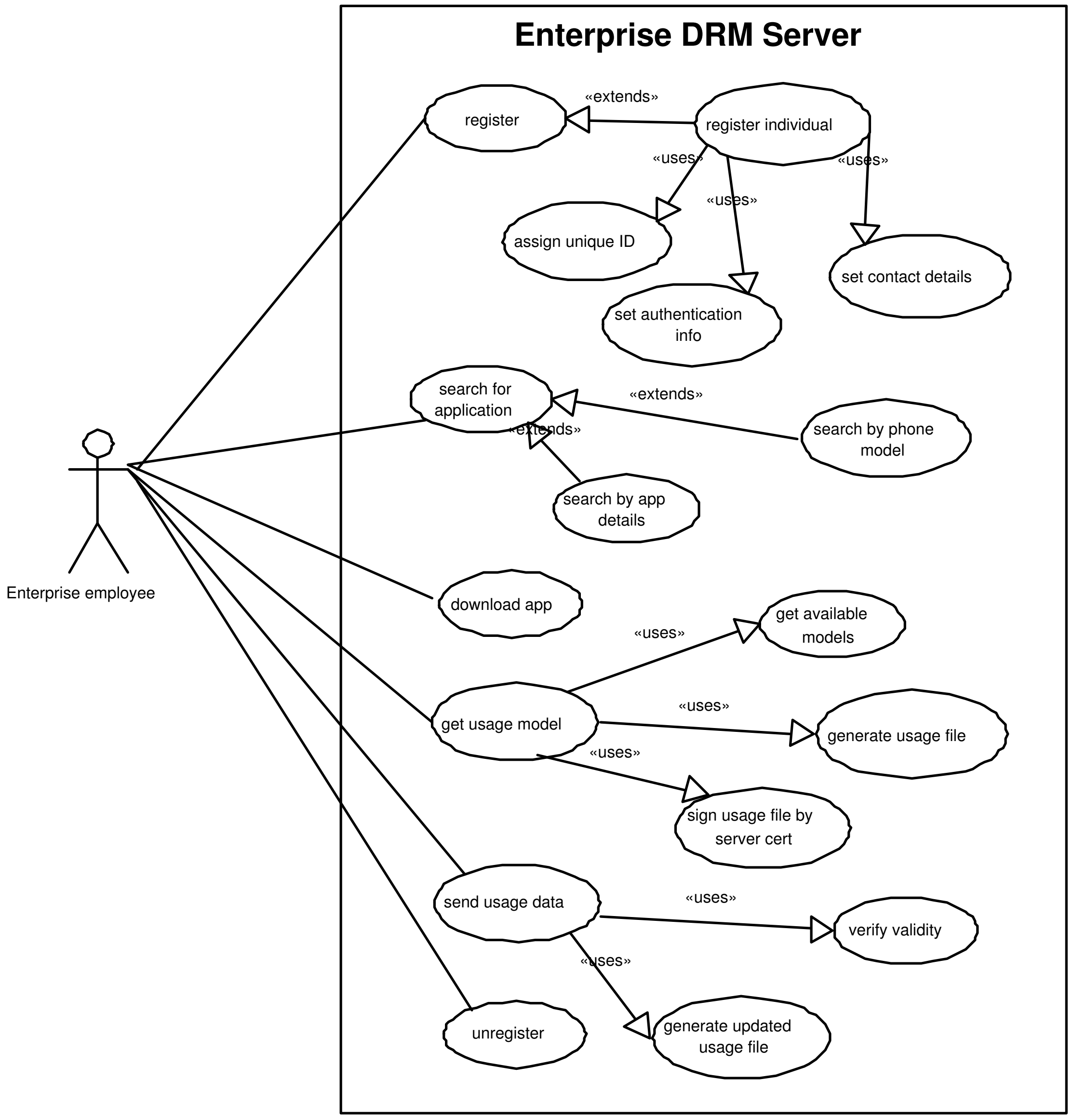


Figure 4.4: Interaction of an enterprise employee with the EDS.

The employees of an enterprise who interact with a local EDS can register with the DRM server, browse through the applications, download the applications and usage rights, and send usage data to the EDS. Usage data is collected at the EDS and forwarded to the VDS for billing. The diagram of actions performed by an enterprise employee is shown in figure 4.4.

4.4.4 The End-User

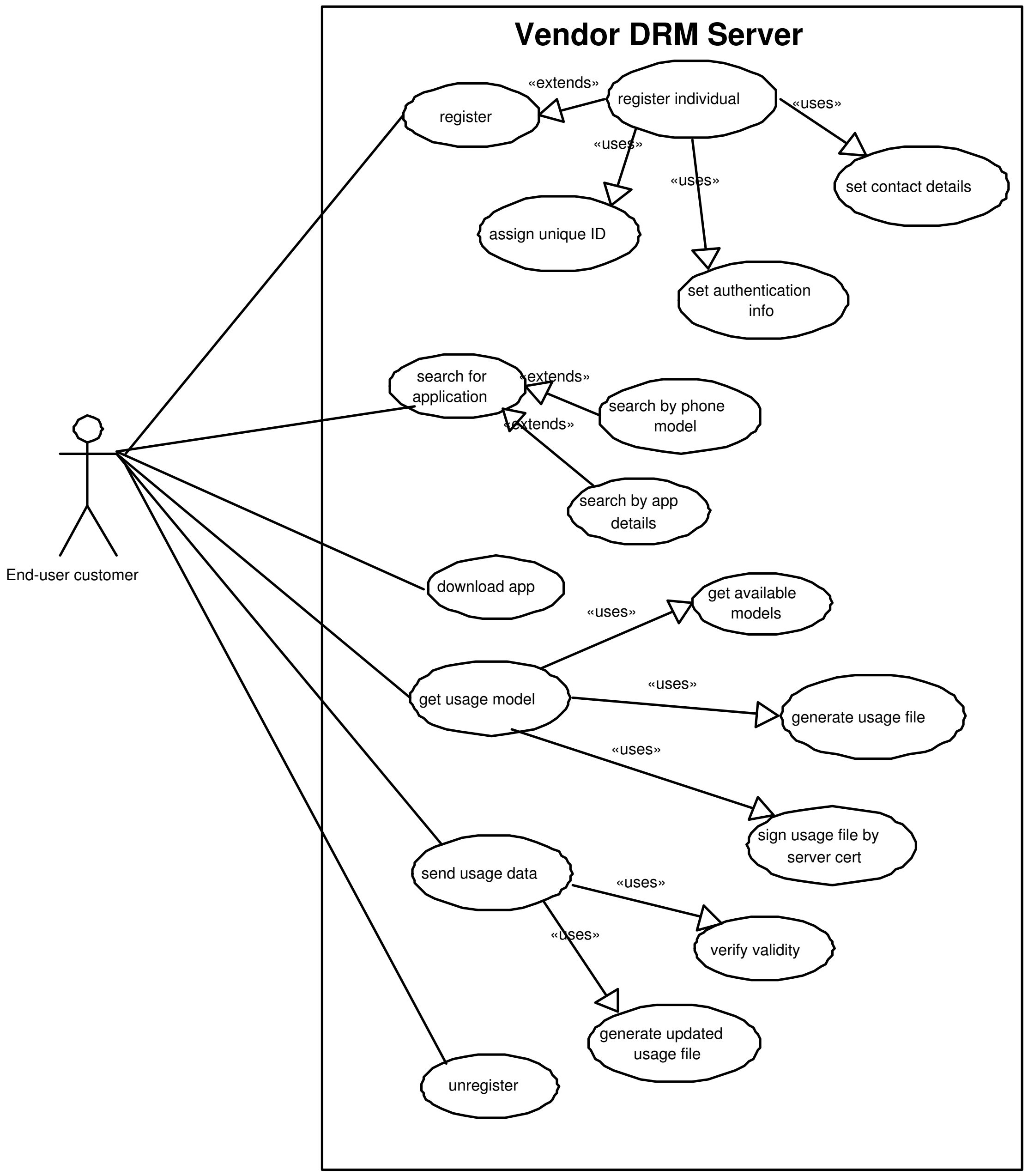


Figure 4.5: End-User Customer - VDS interaction

The end-users, who are customers of the VDS, can use the VDS in a similar manner to how employees interact with the EDS. An end-user customer can register, browse and download the applications, download usage rights and send usage data for billing. The diagram of actions performed by an end-user customer is shown in figure 4.5.

4.4.5 Design Implications

Upon examination of the above use cases, it is clear that a lot of functions overlap between the services provided by the EDS to the employees, and services provided by the VDS to the end-users. Both client types use the same interfaces for common functions, i.e. the registration interface and application management interfaces.

The specification and generation of usage rights is done separately for end-users, enterprise customers and enterprise employees, as the differences between the different client types in this case are more significant. Enterprise customers are issued group usage rights files that have a different set of mandatory elements than usage rights files issued to end-users and enterprise employees. End-user usage rights files and enterprise usage rights files are signed using the VDS private key, while employees' usage rights files are signed by the EDS private key. Also, the VDS has no restrictions on the quantity and constraints it can set in the issued rights files, while the quantities and constraints issued by the EDS depend on the group usage rights file that the EDS has purchased off the VDS.

4.5 Application Usage Rights Generation

Once an application is uploaded to a PARMA server, it must be updated to include the PARMA rights enforcement code. One of the main design goals was to make the insertion of rights enforcement as convenient for the developer as possible, as well as to eliminate the need for manual code insertions and knowledge of proprietary APIs that some of the existing license management systems require [OG04]. For simple usage rights models such as node-locked or user-locked, rights enforcement code needs to be executed only at the application start up to allow only licensed users and nodes to execute the application. In fine-grained usage models, such as feature-based and audit-based, usage rights enforcement becomes a cross-cutting concern. Calls to the usage rights management system need to be inserted throughout the application, to verify usage rights on specific models, or to confirm that the application is still in use throughout the execution. These models require calls to the rights enforcement API to be scattered around the application, modifying numerous classes.

The above requirements are met by the capabilities of aspect-oriented programming (AOP) [Kis02], so a design decision was made to use AOP to insert rights enforcement code into the original application. Using AOP, all of the rights enforcement code can be specified in a single aspect, separate from the original application classes, and woven with the original source code at compile time. Separation of rights enforcement source code from the application source code makes maintenance of the application easier and simplifies upgrades and changes in the usage rights models. Potentially, AOP can be used to add licensing functionality to applications using any of the existing license management systems, such as FLEXnet. However, that would still require application developers to write the rights enforcement aspect code themselves, imposing additional work on vendors, and limiting them to the licensing functionality provided by those systems. For this reason, PARMA automates the AOP code generation.

The application developer only has to specify the methods in the original application before or after which enforcement code should be inserted. These methods are expressed as pointcut elements from the PARMA schema described in chapter 3 and are part of the constraints on application execution expressed in the usage rights file. The usage rights file containing pointcuts is converted into aspect code using XSL Transformations (XSLT) [W3C99b], that is woven into the original application, as shown in figure 4.6 [1].

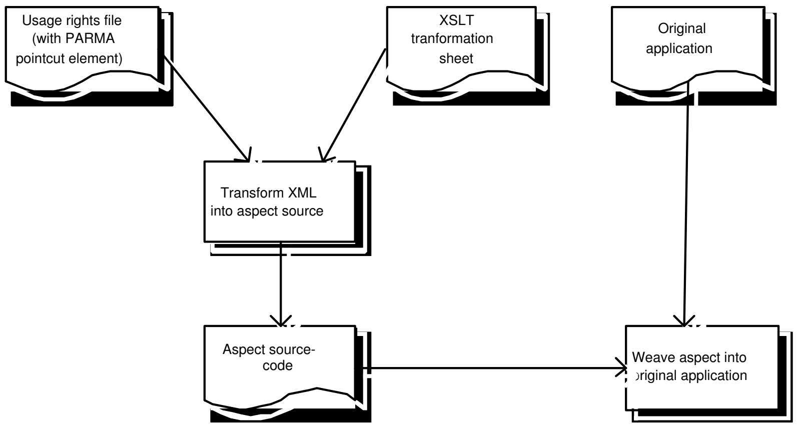

Usage rights file (with PARMA pointcut element)
XSLT tranformation sheet
Original application
Transform XML into aspect source
Aspect source-code
Weave aspect into original application

Figure 4.6: Aspect generation and weaving

The work done in this thesis generates only a skeleton for pointcuts that intercept the original application, therefore not providing a full implementation. The body of the actual aspect method depends on the implementation of the client side enforcement architecture, that is provided by the other parts of the PARMA project. Some of the possible implementations could log the information about the method execution for the later billing in audit-based usage rights model, prevent methods from execution in the feature-based usage rights model, or allow the execution to proceed but display nagging messages to the user, in implementations of the demo usage rights models. Depending on the client side implementation, the licensing behavior could be provided via software libraries that vendors can reuse, or vendors could add their own aspect body for custom implementations.

The generation of the pointcuts in the current version of the PARMA architecture is implemented only for J2ME applications. PARMA can be extended with implementation for other Java versions and for other programming languages that have AOP implementations.

4.6 Usage Rights Models

One of the major PARMA design goals is to provide a variety of usage models to suit both enterprise and end-user customers. PARMA is designed to support unlimited, named-user, time-limited, feature-

[1] The transformation sheets PARMA uses were developed as a part of a final year project in Trinity College Dublin [McM04], according to the PARMA specifications and under guidance of PARMA project members.

based model, subscription-based, pay-per-use (real-time or audit-based), node-locked, counted, metered, and concurrent usage rights models. The PARMA REL, described in chapter 3, was designed to express details of the usage rights in these models. All but the audit-based pay-per-use model and the concurrent model can be enforced on a device by an enforcement architecture that interprets the PARMA usage rights file. The audit-based pay-per-use model and the concurrent usage model require interaction with the server for rights enforcement. In an audit-based model, in order to extend the time-limited usage rights, users must send the logs of the previous application usage to the server for billing. In a concurrent model, users can obtain the usage rights for the application only if the number of users who have concurrent usage rights at the moment does not exceed the maximum number of licensed users. The rights management interface, that both the VDS and the EDS expose to clients, contains methods for server side validation and enforcement of these two models, described in more details in chapter 5.

4.7 Security Issues

The PARMA architecture manages application usage rights, and, although payment for the usage rights is out of the scope of this dissertation, PARMA needs to verify and ensure the non-repudiation of the information that the payment for usage rights is based on. PARMA servers and clients need to authenticate themselves to one another, and all communication between clients and servers should remain confidential.

4.7.1 Integrity and Non-Repudiation of the Usage Data

The audit-based usage model generates usage data that is periodically sent from the customer (both end-users and enterprise customers) to the VDS for billing. PARMA needs to provide the integrity of the usage data, i.e. it needs to ensure that the usage data has not been modified during the data generation, while it is stored on the device or during its transfer to the VDS. The data that the VDS uses to generate the payment information must be identical to the original usage data generated by the client side enforcement architecture, to avoid overcharging or undercharging the client for application usage. Once the usage data is received by the VDS, PARMA needs to ensure that the data received is generated by and comes from the client that is billed for the application usage based on this data, and not from the third party with possible malicious intentions, i.e. PARMA needs to authenticate the data sender. Non-repudiation is also provided by PARMA, so that users who generated usage data cannot claim generated data does not belong to them in order to avoid payments for the application usage.

Integrity of the usage data, authentication of the usage data signer, and non-repudiation of the usage data are ensured using the public-key cryptography, specifically digital signatures. The EDS and the VDS servers are assigned private keys and during installation and the EDS registration process they obtain each other's public keys. Usage data is digitally signed by the EDS's private key when it is generated by enterprise employees or received by the EDS from the employees using the application.

The VDS verifies the digital signature using the EDS's public key, and if the usage data was tampered with, the digital signature verification fails. As only the EDS has an access to the EDS private key, digital signatures confirm the identity of the EDS. Data integrity and signer identification together provide the non-repudiation of the usage data, so that, in the case of any disputes, it can be proven that the usage data is generated by the EDS and its content is exactly the same as it is received by the VDS.

4.7.2 Integrity of the Usage Rights Files, Non-Repudiation and Replay Attacks

Usage rights models other than the audit-based model do not generate the usage data, but are billed for based on the terms expressed in the usage rights file. Therefore, PARMA needs to provide the integrity of the usage rights files as well, i.e. to ensure that usage rights files are not modified, in order to avoid clients being granted more usage rights than they have purchased, or prevented from availing of the full quantities of usage rights they own. Also, PARMA needs to authenticate the issuer of the usage rights files, in order to ensure that the usage rights that the client architecture uses for granting the access to the application come from the EDS or the VDS, and not from unauthorized parties. Non-repudiation of the usage rights files also needs to be provided so that third parties can verify that usage rights files are issued by the EDS or the VDS and that their content was not modified after issuing.

Usage rights files are secured in a similar manner to the usage data, i.e. using digital signatures to provide integrity, authentication, and non-repudiation. Usage rights files are signed by the VDS's private key or the EDS's private key (that was generated for the EDS at the registration), prior to their issuing to the enterprise and end-user clients, respectively.

Usage rights files are also vulnerable to so-called replay attacks. Usage rights files can be copied, and as no content of the file has been changed, the digital signature would still be valid. However, if a single usage rights file entitled a user for, for example, 10 executions of the application, multiple copies of that file will allow a number of executions that is a multiple of 10. In order to prevent this, each usage rights file is assigned a unique id, that is stored within the usage rights file. The client architecture has to record the id of all of the usage rights files and disallow duplicates. The id of the issued usage rights file cannot be changed, as that would invalidate the digital signature created on the content of the usage rights file.

4.7.3 Authentication, Authorisation, and Confidentiality

PARMA also must ensure that clients and servers authenticate each other, that only authorised clients are granted access to the servers, and that the communication channel is protected so no third party can view the content of the messages exchanged.

Authentication of the EDS and the VDS is done using public-key cryptography, as the VDS and the EDS servers have private keys and each other's public keys for verification. Authentication and authorisation of the clients is implemented using usernames and passwords or PINs, as end-users then do not have to generate their own private keys to authenticate themselves.

Confidentiality of the data exchange and protection of the communication channels is ensured by the use of Secure Sockets Layer in all communications between the clients and the EDS or the VDS, and between the EDS and the VDS .

4.8 PARMA REL Tool Support

PARMA provides tools for the specification and generation of usage rights files as well as their validation against the PARMA schema and against the additional set of validations listed in chapter 3. These tools are a part of the VDS and the EDS servers. The tool can create group usage files containing several different usage rights models that are created for enterprise customers, and end-user usage rights file containing all of the permissions and constraints needed for enforcement of usage rights on the client device. The tool support is important for the adoption of PARMA REL because it simplifies the rights file generation and validation.

4.9 Design Summary

This chapter described PARMA's main design goals, PARMA use cases, requirements analysis and an overview of the architecture design. PARMA is a pervasive application rights management architecture that provides usage rights specification, integration of usage rights with a target application, application and usage rights file delivery, and server side usage rights enforcement. PARMA's main goals are to provide pervasive usage rights management for both desktop and mobile devices, available to both end-users and enterprise customers, to support a variety of flexible usage rights models, and to remove the need for the insertion of calls to the proprietary APIs in the target application source code. Four types of clients are supported: individual and enterprise customers, and small and large application vendors. PARMA consists of two DRM servers, an enterprise DRM server (EDS) that manages usage rights within a single enterprise, and a vendor DRM server (VDS) that manages usage rights for end-users and enterprise customers. The EDS and the VDS servers expose registration interfaces that allow clients to register and manage their authentication information, and allow application vendors to register and upload their applications. Usage rights are embedded into the target application using AOP code that is automatically generated from usage rights files. Audit-based and feature-based models require server side enforcement, provided by PARMA, in order to renew their usage rights. PARMA provides a tool for the specification and generation of PARMA REL usage rights files. The tool also performs additional validations of the usage rights files required by PARMA. Security in PARMA is provided using username/password authentication and asymmetric key cryptography.

Chapter 5

Implementation

5.1 Introduction

The implementation of PARMA is based on open-source technologies as PARMA is intended to be released as an open-source application. It is based on Java, XML, and web services technologies. Web services technologies (WSDL, SOAP, Java Servlets) are the most suitable implementation choice for a pervasive rights management architecture as implemented methods on the server can be accessed from a client developed in any programming language on any client device as long as it supports the SOAP/HTTP(S) communication protocols. The remaining technologies were chosen for their open-source nature, wide adoption and capabilities that meet the requirements of PARMA implementation.

5.2 Choice of Technologies

PARMA was developed using the open source developer tool Eclipse [Fou04b] (v. 2.1.3 and 3.0) and is based on open source tools and products. Both the vendor and enterprise DRM servers are implemented as Java servlets [Mic04b] (v. 2.3.2) on Apache Jakarta Tomcat [AJP04] (v.5.0.19) and Apache Axis [AWSP04] (v 1.2) implementation of SOAP 1.2 [W3C03]. The WSDL [W3C01] interfaces are defined in version 1.1 and converted into Java J2SE [Mic04d] code using the WSDL2Java tool that is a part of Apache Axis. Security was implemented using Java Cryptography Extension (as provided with J2SE v1.4.2), Apache XML Security implementation of XML encryption and XML Digital Signatures [ASF04] (v.1.1) and Bouncy Castle Crypto API [LotBC04] (v. 1.24). Usage rights enforcement code is expressed and woven into the original application using AspectJ [Fou04a] (v. 1.2). The databases were implemented on the VDS and the EDS using MySQL server 4.0 [mys04].

5.3 Components

5.3.1 Introduction

The VDS and the EDS have an identical internal component structure, illustrated in figure 5.1.

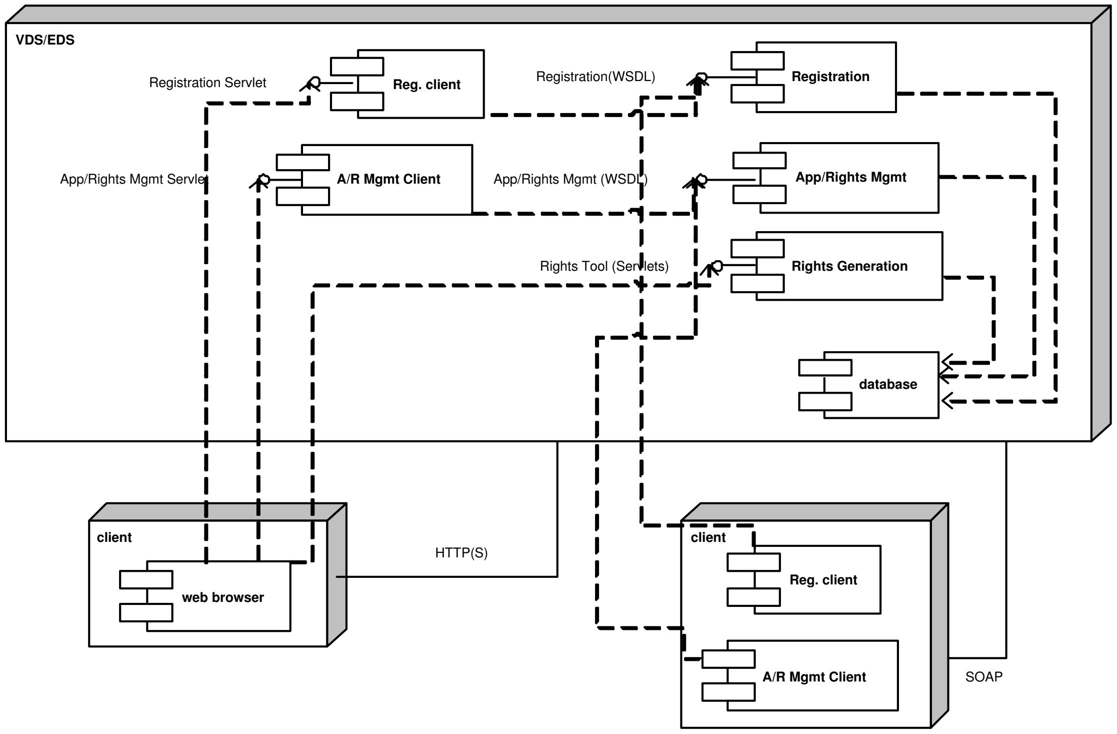


Figure 5.1: VDS/EDS component diagram

Although the internal structure is the same, there are minor differences in the implementation of these components. These differences are due to different restrictions that the vendor and the enterprise customer have on the specification and distribution of usage rights. For example, vendors have no restrictions on the quantity of the usage rights they can issue to clients, while enterprise customers can issue only quantities equal to or smaller than the quantity they have purchased from the VDS.

Both servers have a database and a set of database access objects used by internal components for communication with the database. Servers contain registration and management components exposed to clients as two WSDL interfaces. The client side enforcement architecture can invoke implementations of WSDL interfaces in two ways. It can access the web services directly, via SOAP, given that the client side enforcement architecture implements the web service clients for these services. An alternative way to access web services is using a web browser that will invoke provided servlets, that act as clients to provided web services.

The tool for the specification and generation of usage rights files is implemented as a servlet, running on both the VDS and the EDS, and can be accessed by any device over HTTP(S) protocol. The upload of applications by application providers is also implemented as a servlet.

The following subsections describe details of the implementation of each of these components.

5.3.2 Client, Application and Usage Rights Database

Databases on the VDS and the EDS have identical database schemas represented in the entity-relationship diagram in figure 5.2. The information stored includes client details, application details, details of available usage rights models per application, details of issued application usage rights per application and client, logs of application usage for audit-based billing per application and client, logs of execution of all the methods on the server, and information on available architecture software updates.

The table *Client* (see figure 5.2) stores the information about all of the registered clients, both customers and application providers. The *Authentication* table stores the usernames and passwords for the clients, as well as information required for retrieval of the forgotten passwords and PINs. Client and Authentication tables have a 1:1 mapping, based on the client id, that is a primary key for both tables. The application providers can register applications for distribution through PARMA, and information about those applications is stored in the table *Service*. The Service table has a service id as a primary key, and a provider id as a foreign key that relates it to the client id in the Client table, identifying the service provider. One provider can register many services, i.e. Client and Service tables have a 1:many relationship. The *Requirements* table stores the system requirements of the registered applications, and has a service id as a foreign non-unique key relating it to the Service table. One application can have many requirements, so Service and Requirements are related in 1:many relationship. The Service is also linked to the *RightsOptions* table, that stores the information of usage rights models that a given service supports. RightsOptions has a service id as a foreign non-unique key. Service and RightsOptions are related in a 1:many relationship, i.e, one application can support many usage rights. RightsOptions also have a rights model id as a foreign key, that relates it to the table *Rights*, that stores all of the usage rights models available in PARMA. RightsOptions is related to Rights in a 1:many relationship, i.e. one record in RightsOptions can be related to only one rights model from the Rights table, but one rights model can be related to many records in RightsOptions. The database also stores the information on all of the usage rights files issued to the clients, in the *IssuedRights* table. IssuedRights has rights model id, user id, and application id as foreign keys, relating it to Rights, Client and Service tables, respectively. The single rights file, represented as a single record in IssuedRights, can be issued to only one client, but a single client can be issued many usage rights files, i.e. IssuedRights and Clients tables have a many:1 relationship. Also, a single rights file can define rights only for one application, but rights for one application can be defined in many rights files, i.e. IssuedRights and Service tables are in a many:1 relationship. However, the single rights file can define many usage rights models, and a single usage rights model can be used in many rights files, so IssuedRights and Rights tables have a many:many relationship.

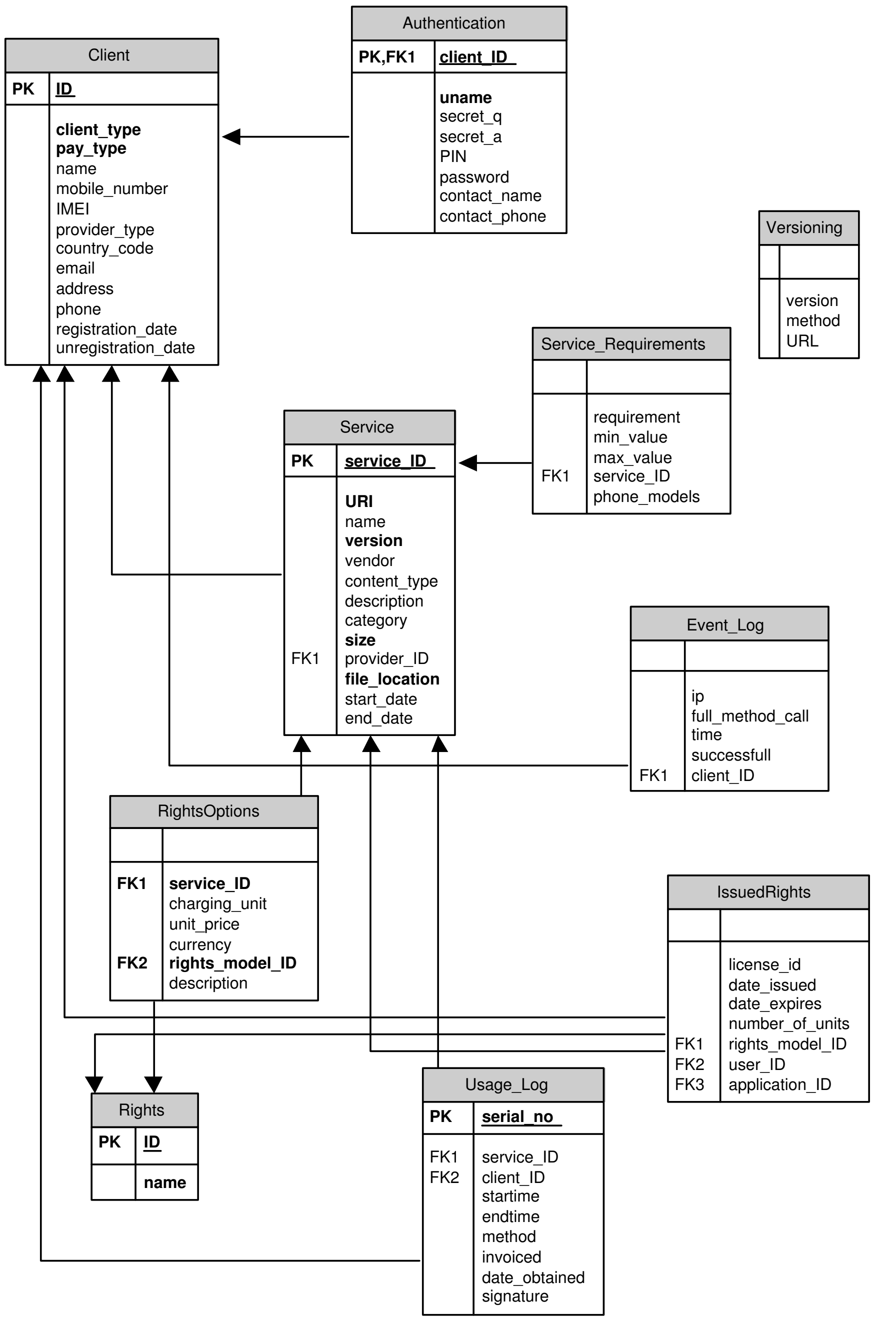


Figure 5.2: VDS/EDS database design

The table *UsageLog* stores the information about application usage gathered through audit-based usage rights models. Records in UsageLog are linked to a given client, in a many:1 relationship, i.e. one client can have many usage records but a single usage records belongs to only one client. Similarly, UsageLog is related to the Service in a many:1 relationship, i.e. service can have many usage logs generated using the application given service defines, but a single usage records contains usage information on only one service. The table *Versioning* is not related to any other tables, and it stores the information about any available PARMA interface updates that clients need to download.

As the EDS and the VDS support different types of clients, the EDS database stores the details of enterprise employees and applications the enterprise has purchased usage rights for, while the VDS database stores details on end-users and enterprise customers, as well as all applications available for distribution.

5.3.3 WSDL Interfaces, Data Types and Implementation

WSDL interfaces exposed by the EDS and the VDS are specified in two main components: EDSRegistration and EDSRightsManagement (see appendix B.1, B.2). Both WSDL files use data types from the schema defining PARMA REL, as well as data types from a separate VDS_EDS_DataTypes schema (see appendix B.3) containing data types needed by methods in EDSRegistration and EDSRightsManagement.

5.3.3.1 Client Registration Interface and Security

Client Registration

The registration interface provides methods for the registration of four different client types and registration of a new service by a service provider. PARMA clients can be application users (end-users or enterprises) and application providers (individual developers and vendors). The VDS provides registration of both enterprise and individual customers and application providers, while the EDS provides registration only for the individual employees within the enterprise.

The VDS_EDS_DataTypes schema defines an abstract client_id type that all of the other client types are extended from, adding the identification elements that depend on the client type. However, a minimum set of id elements is common to all clients. Those are id, client type, payment type, and basic authentication information.

At registration time, all clients are required to choose a unique username and a password or a PIN to be used for further logins to the system. Enterprise customers are required to choose a password, since it is assumed that enterprise administrators will be accessing the system from desktops, or from the EDS. End-user customers are required to choose a PIN number, since they have more familiarity with using PINs than passwords on mobile phones. For end-user customers, all of the other registration details are optional, so clients can use PARMA without providing their name, IMEI or a phone number, to ensure maximum privacy. The optional set of details includes name, address and an email address. Enterprise customers need to identify themselves by their name and the country their business is registered in, and optionally provide a full address, phone number and an email address. The sequence of registration steps for PARMA users is shown in figure 5.3.

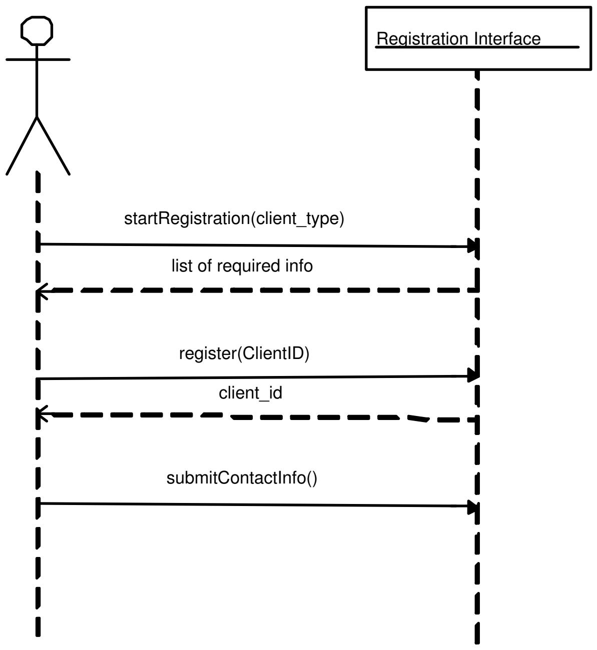


Figure 5.3: Client registration

Distribution and Generation of Private and Public Keys at Registration

At the end of basic registration with the VDS (prior to optional registration of the additional contact details), clients are provided with a link to download the Java keystore. The VDS's public key is imported in the keystore so clients can use it to verify that the applications and usage files they download come from the VDS. End-user clients get the Java MEkeystore used with J2ME, while enterprise customers get the J2SE version of keystore for desktops. During registration, a private key is generated for every enterprise customer and imported in the EDS keystore. This private key is used to sign the application usage data stored on the EDS that will later be sent to the VDS for billing. As employees within the enterprise will have their usage rights files generated by the EDS, the EDS also needs to sign these usage files with its private key to ensure that employees receive usage files only from the EDS. When employees register with the EDS, they are provided with the keystore containing both the VDS's public key (as applications are signed using the VDS's private key), and the EDS's public key (as usage rights file are signed using the EDS's private key).

Client and server keystores are also used for authentication when invoking the web services and servlets over HTTPS. End-users do not have private keys so they authenticate themselves to the servers using usernames and passwords or PINs. The EDS and the VDS servers have their own private keys, and end-users and enterprise employees have public keys, corresponding to servers' private keys, imported in their keystores. These keys are used to authenticate the EDS and the VDS to enterprise employees and end-users, respectively.

Application Registration

Applications can be registered only by registered service providers, after they have logged into the system. New applications can be registered by providing only the application name and a unique URI, to provide the location of the application being uploaded. However, it is recommended to register application details and requirements, so clients can determine if an application is suitable for their device and intended purpose. Application details include the name of the vendor, the application version, the type of application, a short description, and the size of the application. The set of application requirements is not fixed, and vendors can enter the name of the requirement together with the minimum and/or maximum values that the client device needs to support in order to execute the application. Application requirements can also be expressed as a list of device models that the application is supported on. The sequence of application registration steps is shown in figure 5.4.

The registration interface also provides methods for application providers to enter details of usage rights models that their application supports. When an application provider registers an application, the full list of models that PARMA supports is offered and application providers are required to choose from the list, customizing the terms of the model by adding the billing unit, price per unit, currency, and description. A single usage rights model can be entered more than once, providing different billing units. For example, the subscription model can be charged monthly or yearly.

Methods for changing password/PIN, for obtaining the forgotten password/PIN and for terminating registration with the system are also provided in the Registration interface. For a full list of operations provided by the EDSRegistration interface refer to the appendix B.1.

5.3.3.2 Rights Management Interface

The rights management interface contains methods for searching through the available applications, obtaining application details, device requirements and available usage rights models. Applications can be browsed by any of the application details, such as name, category, vendor, and size, or by a mobile device model. The rights management interface also includes methods that aid enforcement of the audit-based pay-per-use model and the concurrent usage rights model.

Audit-based usage rights model implementation

Two methods are provided to facilitate the audit-based model, getAuditData, and sendAuditData. The

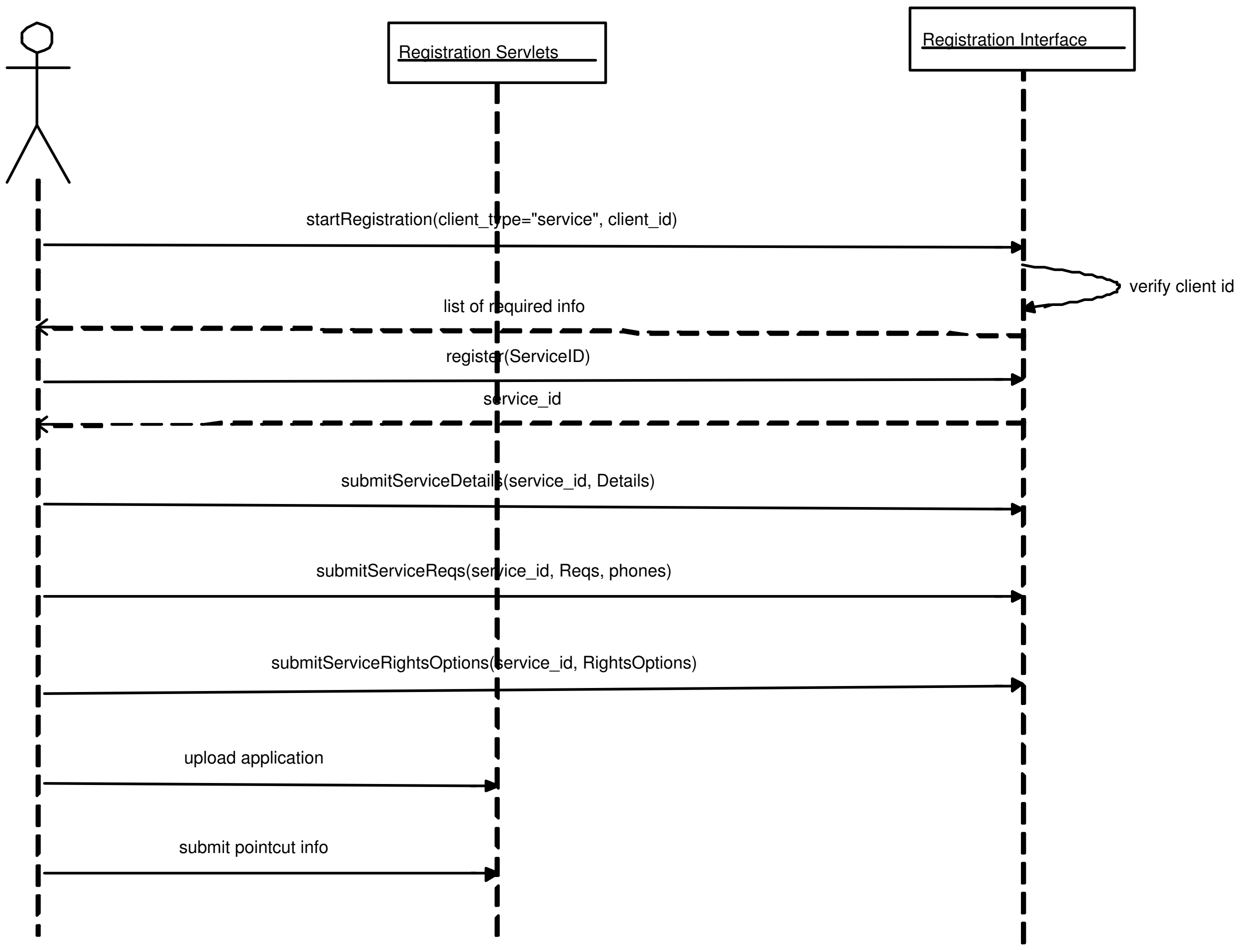

Registration Servlets
Registration Interface
startRegistration(client_type="service", client_id)
verify client id
list of required info
register(ServiceID)
service_id
submitServiceDetails(service_id, Details)
submitServiceReqs(service_id, Reqs, phones)
submitServiceRightsOptions(service_id, RightsOptions)
upload application
submit pointcut info

Figure 5.4: Application registration

method getAuditData is exposed by the EDS and it reads the application usage data stored on the EDS, while the method sendAuditData is exposed by both the EDS and the VDS. On the EDS, sendAuditData receives the data from the client devices, and on the VDS it receives the data from the EDS. It also verifies the usage data sent by the EDS by checking the digital signatures and stores valid data in the VDS database for later billing. If the usage data is verified, a new usage rights file for the client is generated, containing the usage rights for the next billing period, where the billing period is specified by the application vendor. If usage data is not sent to the VDS by the end of the specified billing period, usage rights on the application expire. The means of collecting data on the device and the form in which data is transferred (encrypted or unencrypted, digitally signed) is out of the scope of this dissertation and should be handled by the client enforcement architecture. The client side enforcement architecture will be provided by a separate part of the PARMA project and will be integrated with the server side provided by this dissertation [DDD04].

The methods getAuditData and setAuditData are designed so the transfer of the usage data can be initiated by either the EDS or the VDS, depending on the vendor's and customer's preferences and agreement.

Once data is transferred to the EDS, it is stored in the database, with each individual record being signed by the EDS's private key. Signing this data with the private key is important so that, once data is transferred to the VDS, it can be guaranteed that usage data comes from the EDS with the given client id matching its private key. The EDS is billed for application usage based on this data. When usage data is stored on the EDS, every individual record is assigned a unique serial number, and message digests of these records, in blocks of fifty, are calculated and stored in the database. Message digests are used to verify that the content of the records has not been modified while stored on the EDS. Updates to the table where this data is stored are not granted so the digests cannot be tampered with, and the original usage data cannot be modified on the EDS. If either message the digest verification or digital signature verification fail, the user is not granted a new usage rights file with rights for the new billing period.

The usage data life-cycle is illustrated in figure 5.5. Existing implementation of PARMA implements the usage data states on the EDS and the VDS, while client side states are to be provided by the client side enforcement architecture.

Concurrent usage rights model implementation

The concurrent usage rights model in PARMA is implemented differently than in existing desktop license management systems (see section 2.3.2). In concurrent usage models for desktop architectures, a client must connect to the server to obtain the temporary usage rights, and, once the application exits, usage rights must be returned to the server immediately so they can be made available for other users. To implement this model, the assumption is made that a network connection is available both at the application start up and termination. However, that assumption can not be made in the mobile environment,

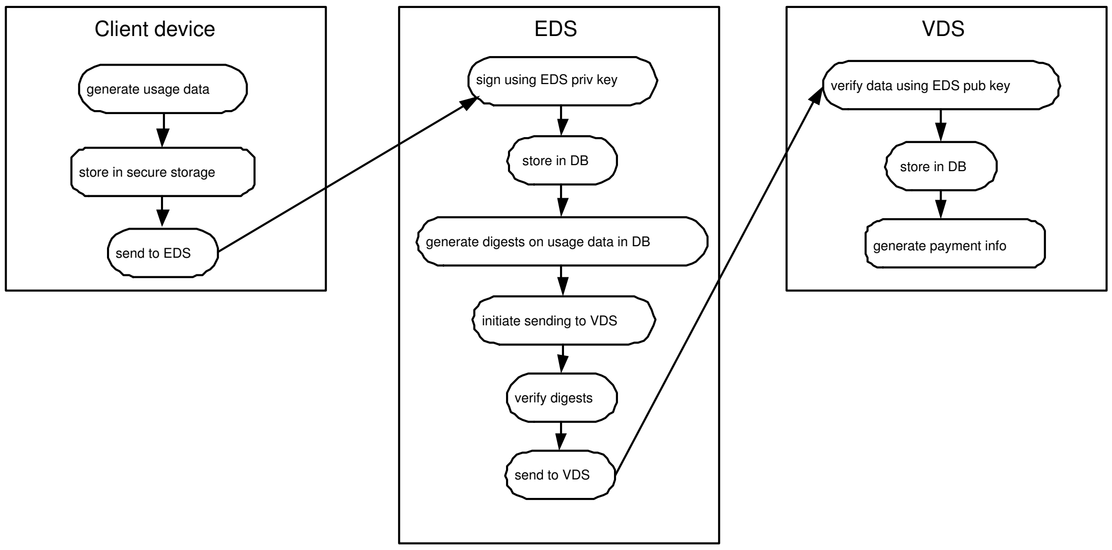


Figure 5.5: Usage data life-cycle and security

as occasionally-connected devices do not establish frequent network connections as desktop devices do. Therefore, a concurrent model is implemented in PARMA as a time-limited usage model with short time-outs. The VDS can issue unlimited number of concurrent licenses to individual clients, and the EDS can issue only a number that is equal to or less than the number of concurrent usage rights the EDS has purchased from the VDS. Clients request usage rights through rights file generation servlets, and specify the time period for which they plan to use the application, expressed in hours or days. The maximum period that the usage rights can be obtained for is set to seven days. Such a small limit on the maximum period ensures that no device locks up the usage rights for too long and that other devices that need the usage rights are able to obtain them from the server. If clients need to use the application for longer than seven days, the time-limited usage rights model might be more suitable than the concurrent usage rights model. If the request was submitted to the VDS, usage rights are granted as the VDS has no restrictions on the number of concurrent usage rights it can issue. If the request was issued to the EDS, the servlet checks the group usage rights file to determine the maximum number of concurrent devices that the enterprise is granted the rights for, and checks the database to determine how many concurrent rights have already been granted to devices when the request is received. If it is determined that there are sufficient rights available, the servlet creates a new usage rights file for the client, and the number of issued concurrent rights at that moment is incremented. Once the time period specified in the concurrent usage right file expires, the number of the issued concurrent rights in the database is decremented.

A complete WSDL interface that defines all of the operations exposed by the EDSRightsManagement is contained in appendix B.2.

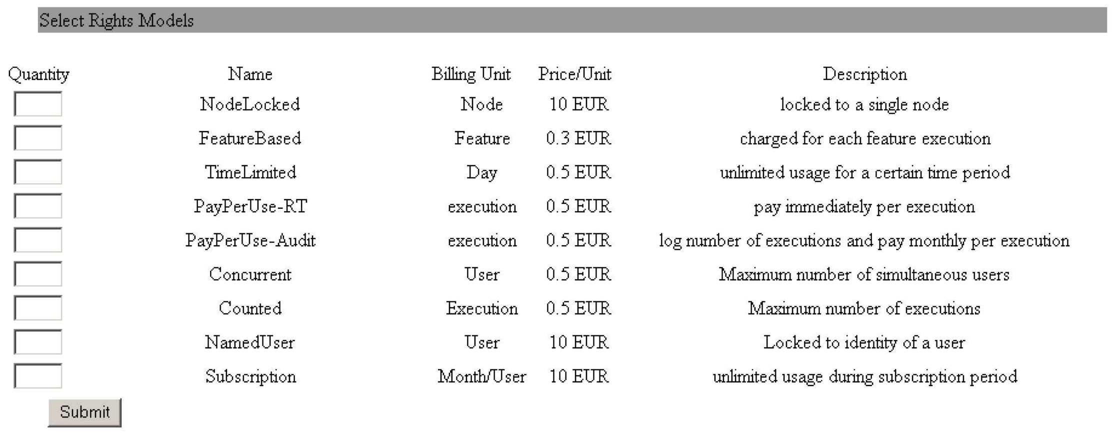
5.3.4 Usage Rights File Specification and Generation

PARMA usage rights files, even though they all conform to the PARMA REL schema, are generated in three different ways, depending on the server they are generated on, and on the type of a client they are generated for. Usage rights files can be generated on the VDS for the EDS, on the VDS for end-user clients, and on the EDS for enterprise employees. The specification and generation process is implemented using Java servlets.

5.3.4.1 The VDS to the EDS Usage Rights File

Usage rights for enterprise applications are granted to an enterprise for all of their employees. Employees do not communicate directly with the VDS, nor does the VDS have a knowledge of how the usage rights are distributed within the enterprise. The usage rights generation module of EDS ensures that the total quantity of usage rights granted to the enterprise is not exceeded when the individual employee usage rights are issued.

A list of available usage rights models for the chosen application is offered, listing the sample billing units, sample prices and a short description of each model, as illustrated in figure 5.6.


Select Rights Models

Quantity	Name	Billing Unit	Price/Unit	Description
	NodeLocked	Node	10 EUR	locked to a single node
	FeatureBased	Feature	0.3 EUR	charged for each feature execution
	TimeLimited	Day	0.5 EUR	unlimited usage for a certain time period
	PayPerUse-RT	execution	0.5 EUR	pay immediately per execution
	PayPerUse-Audit	execution	0.5 EUR	log number of executions and pay monthly per execution
	Concurrent	User	0.5 EUR	Maximum number of simultaneous users
	Counted	Execution	0.5 EUR	Maximum number of executions
	NamedUser	User	10 EUR	Locked to identity of a user
	Subscription	Month/User	10 EUR	unlimited usage during subscription period

Figure 5.6: EDS usage rights specification

The person responsible for selecting the rights usage options on behalf of the enterprise must fill in the desired quantities of a specific model. Usage rights granted to the EDS are not constrained in any way apart from the constraint on the quantity of individual usage rights that can be distributed from the EDS, pointcuts for the feature-based model, start date on the subscription model and an expiration date on the pay-per-use audit-based model. Each RightsType element represents the usage rights model granted to the enterprise, and its attribute quantity represents the maximum number of units that can be distributed to employees within the enterprise. Units are dependent on the usage rights model and on the vendor's preferences specified at the time of application upload.

```
<o-ex:context>
 <parma:RightsType parma:quantity="5">
   <parma:name>NodeLocked</parma:name>
 </parma:RightsType>
 <parma:RightsType parma:quantity="10">
   <parma:name>FeatureBased</parma:name>
 </parma:RightsType>
 <parma:RightsType parma:quantity="10">
   <parma:name>PayPerUse-Audit</parma:name>
 </parma:RightsType>
 <parma:RightsType parma:quantity="10">
   <parma:name>Concurrent</parma:name>
 </parma:RightsType>
 <parma:RightsType parma:quantity="200">
   <parma:name>Counted</parma:name>
 </parma:RightsType>
  <parma:RightsType parma:quantity="5">
   <parma:name>Subscription</parma:name>
 </parma:RightsType>
</o-ex:context>
```

The above example of the EDS group usage rights file grants the node-locked usage model for up to 5 employees, the feature-based for up to 10, the audit-based pay-per-use model for 10 employees, the concurrent usage rights for 10 concurrent users, 200 executions within a counted usage rights model that can be distributed over any number of employees, and 5 subscription usage models.

The node-locked, concurrent and counted models do not require any constraints at this point; constraints are added when the EDS distributes usage rights to the employees. However, feature-based, audit-based, and subscription models require constraints that are specified by the VDS and passed on to employees through individual usage rights created by the EDS. The restrictions required by these models are as follows:

- The pointcuts for the feature-based model are specified by the application developer at the time of the application upload, and are saved in a test usage rights file. When the EDS requests a feature-based model, these pointcuts are copied from the generic usage rights file into the EDS usage rights file.

- The audit-based pay-per-use model requires a time constraint that denotes the date that the usage rights will expire if the audit log containing usage data for the past period is not sent to the VDS for billing. This date can be a week, a month, or a year away from the date of the usage rights file generation, depending on the vendor's preference for the given application that is specified in the database.

- The subscription model requires only a start date as a constraint, which is set to a date of usage rights file generation. It is the responsibility of the enforcement architecture on the device to verify that usage rights for this model did not expire.

Usage rights files issued by the VDS to the EDS differ in format from end-user and enterprise employee usage rights. End-user/employee usage rights do not contain the quantity attribute, as they are only ever issued to a single client. Also, end-user/employee usage rights contain a broader set of restrictions referring to the individual clients, such as client's user id, IMEI number, or the number of allowed executions, while the EDS usage rights file contains only the above listed constraints that refer to all of the employees in the enterprise.

5.3.4.2 The EDS to the Employee Usage Rights File

Depending on the preference of the enterprise customer, usage rights files for employees within the enterprise can be specified by the employees themselves, or by the EDS administrators. All of the usage rights that are granted are recorded in the database to enable tracking of the issued and remaining rights. Prior to the specification of the individual usage rights files, an overview of available usage models and quantities is given, calculated from the original EDS group usage rights file and the information on issued rights that is stored in the database.

When specifying an individual usage rights file, the constraints dependent on the usage model need to be provided. For the feature-based, subscription and audit-based models, constraints present in the EDS usage rights file are copied to individual usage rights file, since they cannot be modified by the EDS. The location of the EDS server, that the client enforcement architecture communicates with, is provided as a URI, along with the application identification information. The information provided is checked against the additional PARMA requirements specified in section 3.4, and when they are met, a usage rights file is generated and offered for download.

5.3.4.3 The VDS to the End-User Usage Rights File

Usage rights for end-user clients are created by the clients themselves via servlets provided by the VDS. For the feature-based model, constraints are copied from the generic usage rights file created after the application upload (see section 5.3.5). Start date for the subscription model is set to the date of usage rights specification, and the audit-based expiration date is set based on the billing settings in the database specified by the application vendor. All of the other constraints, if required by the chosen usage rights model, are provided by the customer. The presence of these requirements is checked according to additional PARMA validations specified in section 3.4.

5.3.4.4 Security of Usage Rights Files

After usage rights are generated, they are signed using the private keys of the issuing server to ensure that no unauthorized modifications are done to the usage rights contained in them.

Usage rights files generated for end-users and enterprises are signed by the VDS's private key. The enforcement architecture on the client interprets the usage rights files only if it can verify the validity of the signature using the corresponding public key stored in the client's keystore.

Usage rights files issued to the enterprise employees are signed by the EDS's private key, and the enforcement architecture on the client interprets them only if this signature is valid when verified with the EDS's public key stored in the employee's keystore. As employee usage rights are a subset of the usage rights issued to the EDS, the signature of the usage rights file containing the EDS usage rights, generated using the VDS's private key, needs to be verified. If the EDS usage rights file was tampered with, the EDS is prevented from issuing usage rights to its employees.

Digital signatures are generated using Apache XML Security [ASF04] and private-public key pairs generated by Java Keytool [Mic04e], and are embedded in the usage rights files.

5.3.5 Application Upload

The VDS and the EDS store details on all of the applications distributed through PARMA in a local database together with a link to the location of the application installation files. In PARMA, the application is stored on the VDS and the EDS servers themselves. A developer who wishes to distribute the application through PARMA needs to register with the VDS and register a new application. At the end of the registration process, the developer is required to upload the application distribution. After the application is uploaded, the developer is prompted to enter the full signatures of methods before or after which the enforcement architecture should validate the usage rights.

As details of inserting the validation code into the original application is platform dependent, the current version of PARMA implementation supports only the upload of J2ME applications. If the uploaded application does not support the feature-based usage model, the only method that will trigger usage rights validation is the startApp() method, which is the first method that gets executed when the J2ME application is called. The signature of this method is always the same, its return type is *void* and it takes no parameters, but the developer must enter the name of the package and class that the method belongs to in the particular application. From this information, a temporary test usage rights file is generated containing the definition of a pointcut expressed in PARMA REL.

```
<o-ex:permission>
 <o-dd:execute>
  <o-ex:constraint>
   <parma:pointcut>
    <parma:adviceType>before</parma:adviceType>
    <parma:package>
     <parma:packageName>package0.package1</parma:packageName>
     <parma:class>
      <parma:className>class0</parma:className>
      <parma:method>
       <parma:methodName>method0</parma:methodName>
       <parma:returnType>void</parma:returnType>
       <parma:argType>String</parma:argType>
       <parma:argType>int</parma:argType>
      </parma:method>
     </parma:class>
    </parma:package>
   </parma:pointcut>
  </o-ex:constraint>
 </o-dd:execute>
</o-ex:permission>
```

Using a XSLT transformations sheet, *parma:pointcut* elements are converted into pointcuts in the newly generated AspectJ Java file. AspectJ technology is an implementation of AOSD for Java (see appendix C) and that enables the integration of additional source code with the original application, without modifications to the original application's source code. The thesis uses this capability of AspectJ to insert the calls to the licensing code into applications.

```
pointcut pcut_method0 (String variable_1,  int variable_2) :
 call (void package0.package1.class0.method0(String,  int))  && args( variable_1,  variable_2);

before(String variable_1,  int variable_2) : pcut_method0( variable_1,  variable_2) {
//client side enforcement code goes here
}
```

The AspectJ aspect file contains pointcuts and client side enforcement code, that can prevent the execution of the method, prevent the execution of the application, or log the method execution for the later billing. This functionality is provided by the client-side enforcement architecture.

The AspectJ file is compiled, woven into the original application, and together with the original application packaged into a jar file. AspectJ technology enables aspects to be woven into either the application source code, the compiled classes, or even a packaged jar, depending on what the developer chooses to upload. Typically, it will be performed on a jar file. The "jarred" application, that is distributed to clients, is signed by the VDS's private key, so that clients can verify that the application they have downloaded in fact does come from the EDS or the VDS. After the jar is signed, its details are entered into the application database and the application is made available for download. The full life-cycle of the application and usage rights from application development to execution on a client device is detailed in figure 5.7.

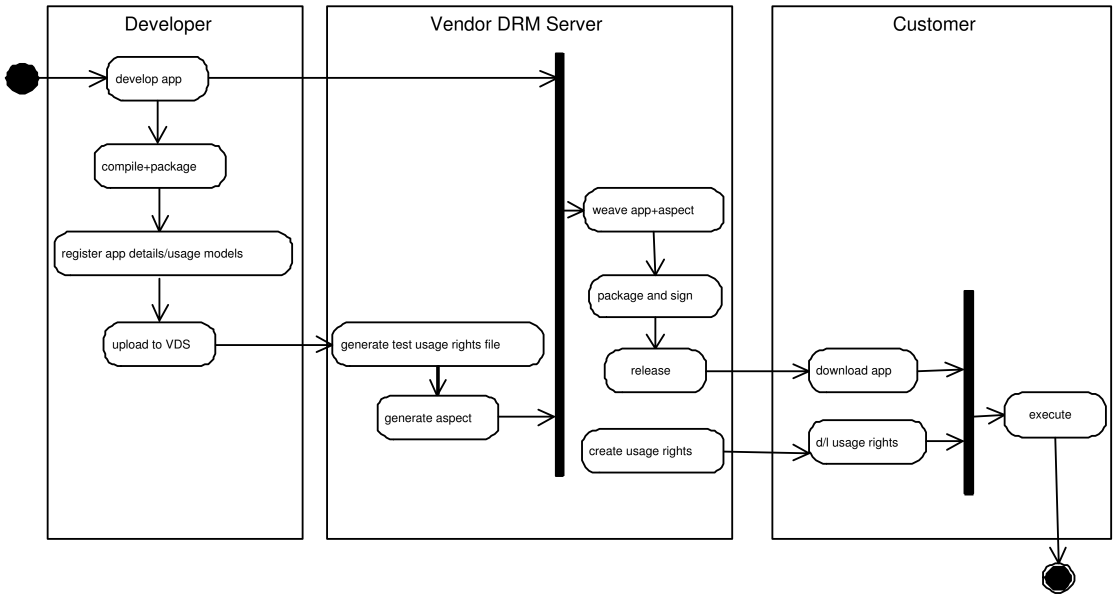


Figure 5.7: PARMA application life cycle

It is important to note that the entire process that follows the application upload and pointcut specification is automated. The developer does not need to have any knowledge of the enforcement architecture code, aspect-oriented programming, proprietary APIs, or needs to perform any other additional work on the application once it is uploaded.

5.4 Implementation Summary

This chapter introduced all of the PARMA components and detailed out how each of them is implemented. Registration and usage rights management functionality are implemented as web services and exposed to clients as WSDL interfaces. The implementation language is Java. The tool for the generation of usage rights is implemented as a Java servlet, and it can create two different types of usage rights files, group usage rights files for the EDS, and individual usage rights files for end-users and enterprise employees. For the enterprise customer, a tool generates group usage rights, that the EDS distributes further to enterprise employees. For end-users and enterprise employees, a tool on both servers generates individual usage rights files. Integration of the usage rights into the target application is done in two ways, depending on the usage models offered. If the application does not support feature-based or audit-based models that are paid for by feature usage, then a Java aspect is generated that contains a single pointcut, at the application's startApp method, inserting the license validation code only at the application start up. If the feature-based and audit-based models are supported, the developer needs to specify signatures of the methods comprising the restricted features. A generic usage rights file is generated that contains those methods specified as pointcuts. The usage rights file is converted to Java aspect source code using XSLT, and woven into original application. Both servers also contain a database, that stores the information about clients, applications, and issued usage rights.

Chapter 6

Evaluation

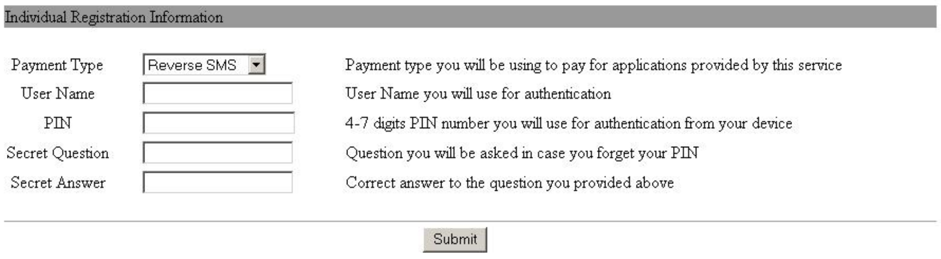
This chapter evaluates the PARMA architecture using a capability comparison based approach. PARMA is compared to other rights expression languages and license management systems using real world examples. It shows the step by step process of registration of the application provider, upload of the application, and the specification of the usage rights models the application can be associated with. The end-user interaction with PARMA is evaluated, covering registration, applications browsing and download, as well as the usage rights specification and download. By analysing these processes, this chapter evaluates to what extent the design goals set out in chapter 4 have been met and how.

6.1 Application and Usage Rights Management Process using PARMA

As stated in chapter 5, PARMA clients can call the server side implementation of registration and usage rights management interfaces directly, if the client side implements SOAP clients for these methods. If client side implementation does not create SOAP calls itself, clients can access the PARMA servlets that make calls to the PARMA server side. The following two subsections describe the PARMA usage from the customer perspective and from the application provider perspective, who are accessing the PARMA interfaces using servlets.

6.1.1 Customer Perspective

Users wishing to download applications and usage rights using PARMA, whether they are end-user customers or enterprise customers, must first register with the VDS. Enterprise employees register with the EDS. A HTML page for PARMA registration enables users to select what type of client they wish to register as. This page also offers a link to the registration of a new service, for registered application providers. Once users have selected a client type, a form that requires registration details is displayed, that is customized for the different client types. Figure 6.1 illustrates a sample form for the registration of end-user customers with the VDS.


Individual Registration Information

Payment Type	Reverse SMS	Payment type you will be using to pay for applications provided by this service
User Name		User Name you will use for authentication
PIN		4-7 digits PIN number you will use for authentication from your device
Secret Question		Question you will be asked in case you forget your PIN
Secret Answer		Correct answer to the question you provided above

Figure 6.1: PARMA customer registration

A confirmation web page after the registration provides users with their assigned ID and a link for the download of the security keystores. A link offering the optional registration of contact details is also offered, together with a link for browsing the available applications. Applications can be searched for based on various criteria, shown in figure 6.2.

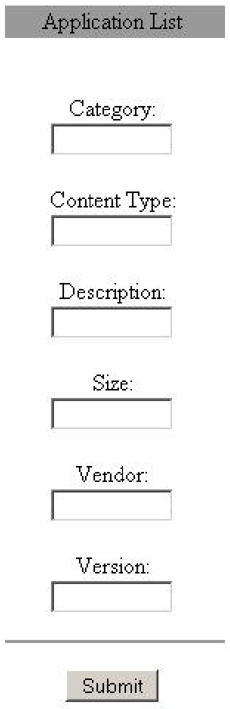


Figure 6.2: Application search

A list of applications matching the criteria is displayed after the submission of this form. If no criteria was entered, all of the available applications are displayed. Every application listed contains links to the application details, usage rights models available for the application, and a link for application download (see figure 6.3).

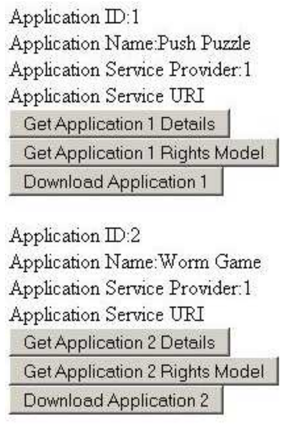
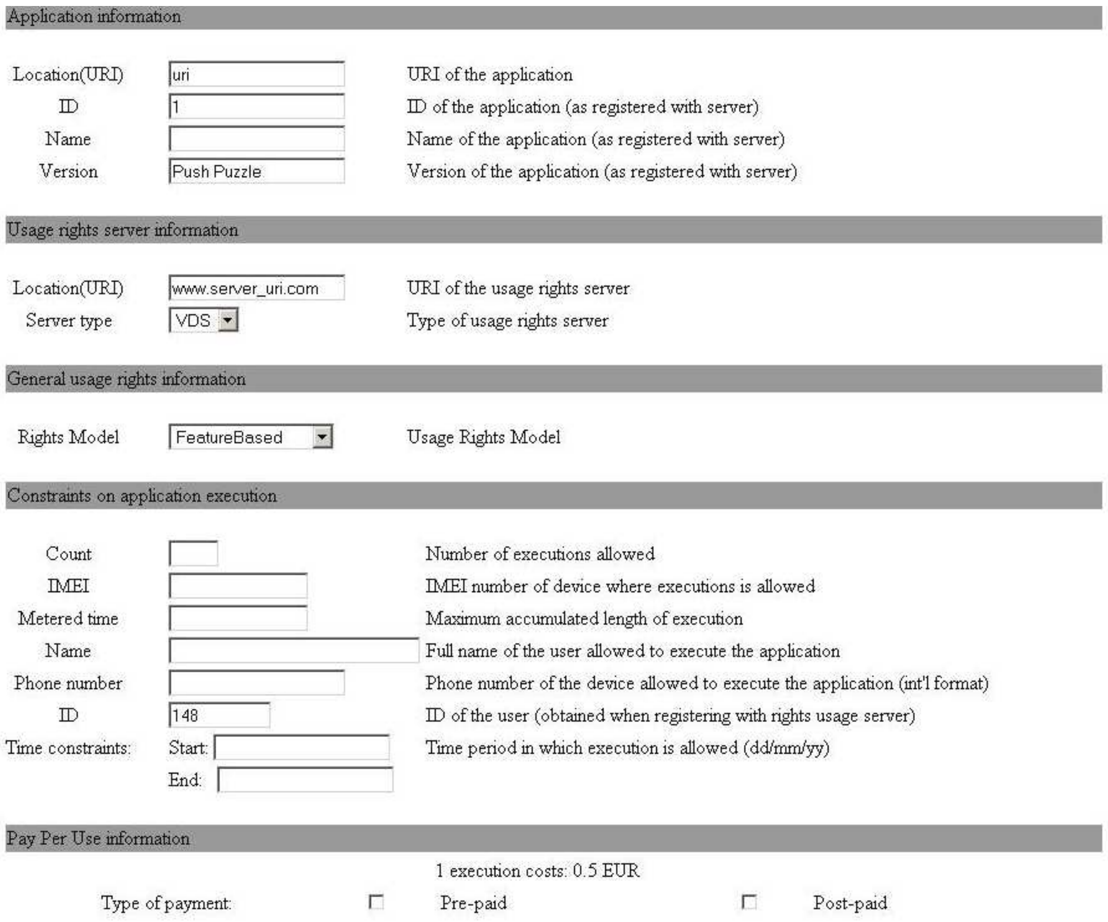

Application ID:1
Application Name:Push Puzzle
Application Service Provider:1
Application Service URI
Get Application 1 Details
Get Application 1 Rights Model
Download Application 1
Application ID:2
Application Name:Worm Game
Application Service Provider:1
Application Service URI
Get Application 2 Details
Get Application 2 Rights Model
Download Application 2

Figure 6.3: List of available applications

Forms for application search, browsing and download are formated so they can be seen on devices with limited display capabilities. This enables users to search for applications both directly from their mobile device or from their desktop devices.

After the application download, users must obtain the usage rights files in order to acquire the rights on the application execution. A form for requesting the usage rights file is illustrated in figure 6.4. Default information about the application is already entered in the form, based on the application ID, while the user needs to select the usage rights model and the required constraints.



Application information

Location(URI)	uri	URI of the application
ID	1	ID of the application (as registered with server)
Name		Name of the application (as registered with server)
Version	Push Puzzle	Version of the application (as registered with server)

Usage rights server information

Location(URI)	www.server_uri.com	URI of the usage rights server
Server type	VDS	Type of usage rights server

General usage rights information

Rights Model	FeatureBased	Usage Rights Model

Constraints on application execution

Count		Number of executions allowed
IMEI		IMEI number of device where executions is allowed
Metered time		Maximum accumulated length of execution
Name		Full name of the user allowed to execute the application
Phone number		Phone number of the device allowed to execute the application (int'l format)
ID	148	ID of the user (obtained when registering with rights usage server)
Time constraints:	Start:	Time period in which execution is allowed (dd/mm/yy)
	End:	

Pay Per Use information

1 execution costs: 0.5 EUR

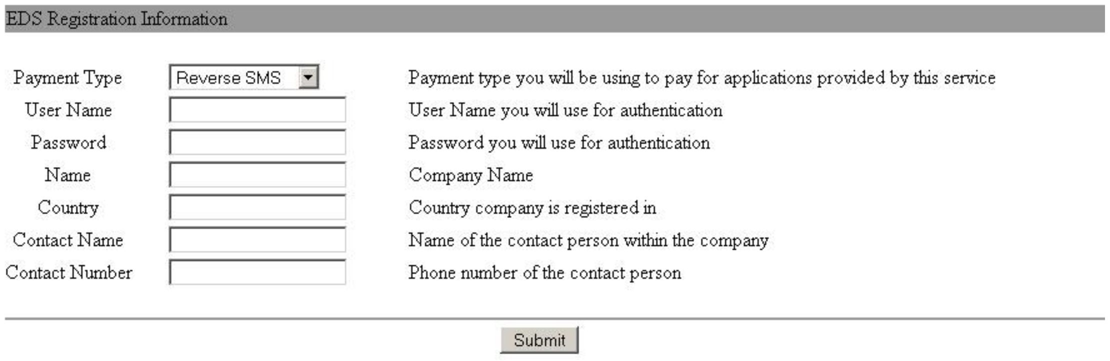
Type of payment: ☐ Pre-paid ☐ Post-paid

Figure 6.4: Usage rights specification

After submitting the above form, a usage rights file, expressed in the PARMA REL, is created and the link for its download is displayed. A usage rights file contains the information about the licensed application, information about the licensing server, selected usage rights model, as well as any relevant permissions, requirements and constraints. For samples of usage rights files that can be created using this form see appendix A.2. The application and the usage rights files can be downloaded directly to a target device, or downloaded to a PC and transferred to a target device over free short range networks.

6.1.2 Application Provider Perspective

On the initial PARMA registration web page, application providers select one of two options, individual application provider registration or software vendor registration. The registration form is customized based on the application provider type that is selected. Figure 6.5 illustrates the details required for the registration of a software vendor.

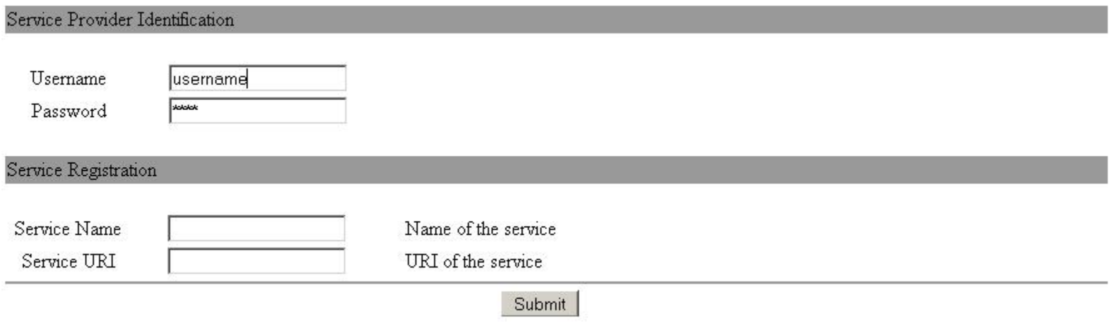


EDS Registration Information

Payment Type	Reverse SMS	Payment type you will be using to pay for applications provided by this service
User Name		User Name you will use for authentication
Password		Password you will use for authentication
Name		Company Name
Country		Country company is registered in
Contact Name		Name of the contact person within the company
Contact Number		Phone number of the contact person

Figure 6.5: Service provider registration

After basic registration, application providers can enter optional contact details or register a new service. The basic service registration requires only the service name and URI (see figure 6.6).



Service Provider Identification

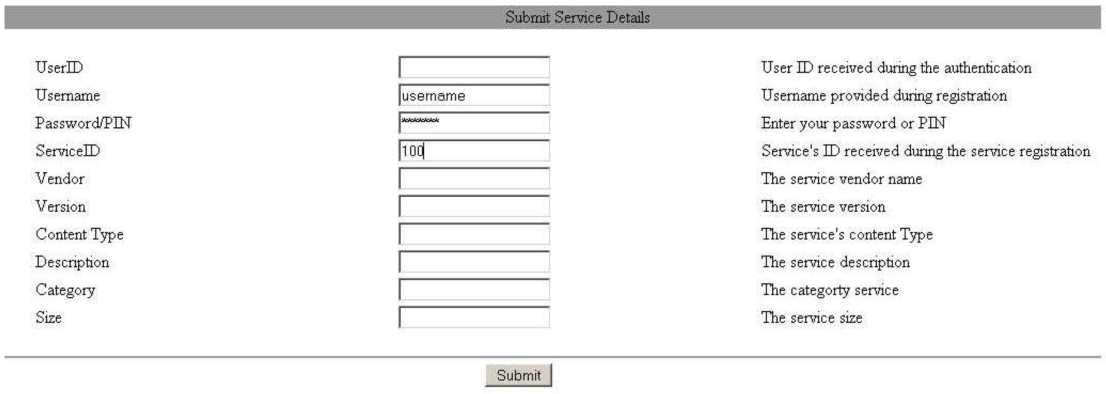
Username	username
Password	****

Service Registration

Service Name		Name of the service
Service URI		URI of the service

Figure 6.6: Service registration

Although the submission of application details that follows the application registration is optional, specifying these details helps users to locate the required application quicker when searching applications by certain criteria. Application details that the application provider can enter are illustrated in figure 6.7.



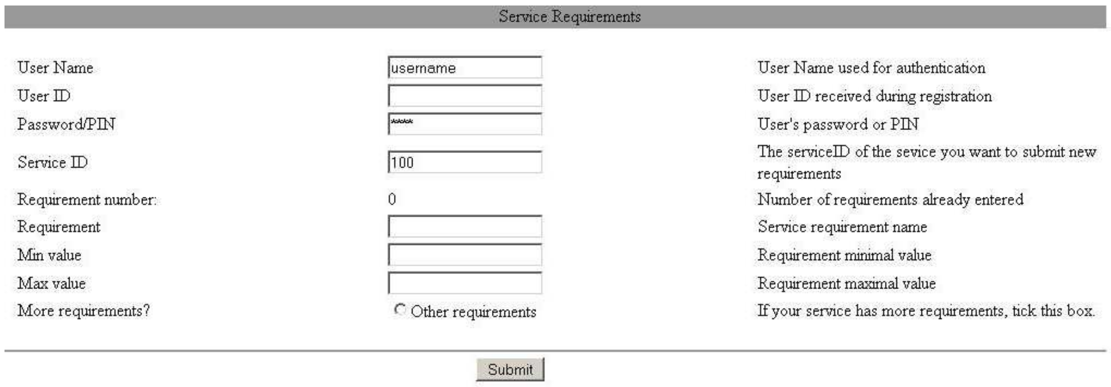
Submit Service Details

UserID		User ID received during the authentication
Username	username	Username provided during registration
Password/PIN	******	Enter your password or PIN
ServiceID	100	Service's ID received during the service registration
Vendor		The service vendor name
Version		The service version
Content Type		The service's content Type
Description		The service description
Category		The categorty service
Size		The service size

Submit

Figure 6.7: Service details registration

Providers can also submit device requirements for devices running the application, such as the device's screen size, required storage space, or memory (figure 6.8) or can simply list the devices and model numbers that the application is supported on (figure 6.9).

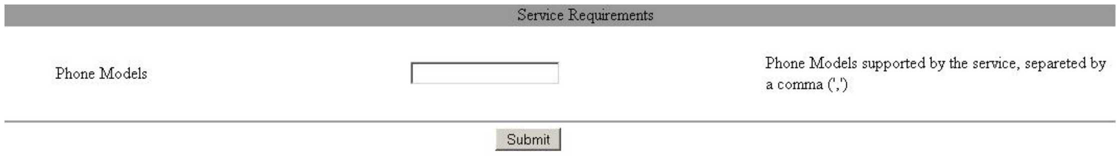


Service Requirements

User Name	username	User Name used for authentication
User ID		User ID received during registration
Password/PIN	****	User's password or PIN
Service ID	100	The serviceID of the sevice you want to submit new requirements
Requirement number:	0	Number of requirements already entered
Requirement		Service requirement name
Min value		Requirement minimal value
Max value		Requirement maximal value
More requirements?	○ Other requirements	If your service has more requirements, tick this box.

Submit

Figure 6.8: Device requirements



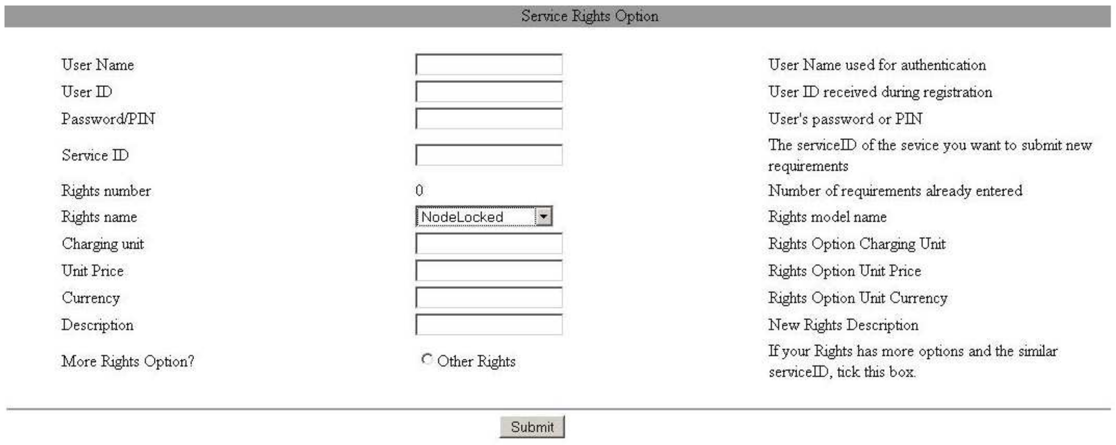
Service Requirements

Phone Models		Phone Models supported by the service, separeted by a comma (',')

Submit

Figure 6.9: List of supported devices

Application providers need to specify the usage rights models they wish to offer their application with. They need to select the models from the list of models offered by the architecture, and specify billing units and the price per unit (see figure 6.10).

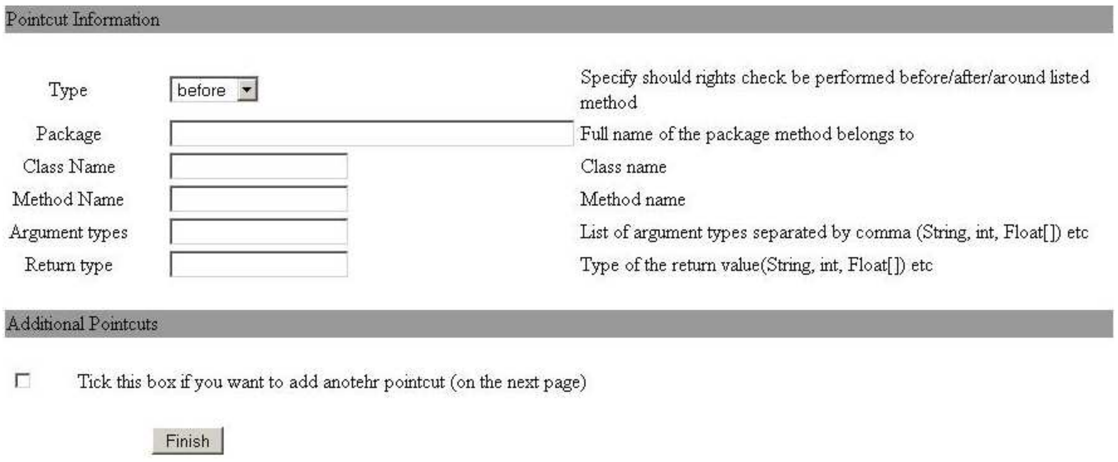


Service Rights Option

User Name		User Name used for authentication
User ID		User ID received during registration
Password/PIN		User's password or PIN
Service ID		The serviceID of the sevice you want to submit new requirements
Rights number	0	Number of requirements already entered
Rights name	NodeLocked	Rights model name
Charging unit		Rights Option Charging Unit
Unit Price		Rights Option Unit Price
Currency		Rights Option Unit Currency
Description		New Rights Description
More Rights Option?	○ Other Rights	If your Rights has more options and the similar serviceID, tick this box.

Figure 6.10: Specification of available usage rights models

After specifying application details, requirements and offered usage rights models, application providers can upload the application to the VDS. If providers do not wish to offer the feature-based model for this application, the application registration process ends after the application upload. If the feature-based model is offered, vendors need to specify the signatures of methods that usage rights are managed separately for. Each individual method is specified in the form that is illustrated in the figure 6.11.



Pointcut Information

Type	before	Specify should rights check be performed before/after/around listed method
Package		Full name of the package method belongs to
Class Name		Class name
Method Name		Method name
Argument types		List of argument types separated by comma (String, int, Float[]) etc
Return type		Type of the return value(String, int, Float[]) etc

Additional Pointcuts

☐ Tick this box if you want to add anotehr pointcut (on the next page)

Figure 6.11: Pointcuts specification for feature-based model

Submitted pointcut definitions are transformed into Java aspect code that is woven into the uploaded application, and stored on the VDS.

6.2 Evaluation of PARMA Design and Implementation

The analysis of the capabilities and shortcomings of the state-of-the-art systems in application rights management and software license management, presented in chapter 2, provides a set of requirements that PARMA should meet in order to design a usage rights management architecture for a pervasive computing environment. These requirements were set out in chapter 4.2.1 as the main design goals and are as follows:

1. To support a variety of both simple and fine-grained usage models, in particular feature-based and audit-based, that tailored to the mobile environment.
2. To remove the need for the insertion of proprietary API calls to the usage rights architecture in the original application by the developer.
3. To provide an integrated pervasive platform for usage rights management for both mobile and desktop devices within the enterprise.

The following subsections evaluate how PARMA met each of these requirements.

6.2.1 Flexible Usage Rights Models

The PARMA project defined a new rights expression language, PARMA REL, extended from ODRL to support a variety of usage rights models for both desktop and mobile applications. Other existing RELs, reviewed in chapter 2, are developed either primarily for desktop applications or primarily for content, and as such they do not have the elements necessary to support PARMA requirements. PARMA aimed to support all of the traditional usage rights models applied to desktops, such as node-locked, but also introduced new usage models for mobile applications, such as feature-based and audit-based models. PARMA REL is also designed to be open, extensible and compatible with current standards.

Table 6.1 shows a comparison of PARMA with other RELs in regard to extensibility, platform support, content/application support, tool support and openness. XrML/MPEG-21 is not suitable for the expression of usage rights as XrML is primarily aimed at desktop content. XrML REL is also patented with the U.S. Patent Office. OMA DRM lacks extensibility and flexibility since it was primarily aimed at content and simple MIDP applications and ODRL alone is not suitable since it is aimed at content on desktops. PARMA combines the features from OMA DRM and ODRL with new features suitable for mobile applications to represent a REL satisfying the above stated requirements.

Tool support for the specification and generation of usage rights files is one of the important factors in the adoption of a REL. PARMA, unlike the ODRL schema that PARMA is based on, provides the

	open spec.	patented	content	applications	extensible	tool support	desktop	mobile
XrML/MPEG-21	x	x	x	x	x	x	x	
OMA DRM	x		x	x		x		x
ODRL	x		x	x	x		x	
PARMA	x			x	x	x	x	x

Table 6.1: PARMA REL compared with other RELs

servlet tool for generation of PARMA compatible files (see section 5.3.4).

An important feature of PARMA shown in the above table is extensibility. New rights usage models can be added by defining a new XML schema that extends the PARMA schema. The new schema would include all of the usage rights models provided by ODRL and PARMA together with constraints for newly defined usage models. PARMA also provides a tool for specifying and generating valid PARMA usage rights files, enabling developers and customers to specify usage rights without any knowledge of the syntax and grammar of the underlying REL. PARMA does not support the specification of usage rights on content; it is designed for the expression of usage rights on applications only. Table 6.1 compares PARMA with existing RELs, and shows that PARMA is the only REL suitable for both desktop and mobile devices. Table 6.2 compares existing RELs and PARMA in terms of supported usage rights models.

	Node	Concur	PrePay	PostPay	Audit	Time	Meter	User	Feature	Subscr	Count
MPEG-21	x		x	x		x	x	x		x	x
OMA 1.0						x					x
OMA 2.0	x					x	x	x		x	x
ODRL		x	x	x		x	x	x		x	x
PARMA	x	x	x	x	x	x	x	x	x	x	x

Table 6.2: PARMA vs other RELs application usage models support [MDSG02, MA02, Ian02]

It should be noted that existing implementations of DRM systems based on these languages do not necessarily implement all of the models in the above table. Support for a certain model in the above table is based upon the existence of the elements that can be used to express the permissions and constraints needed to define a given usage rights model. For example, the user-locked model can be expressed if the REL supports a way of identifying the user. On the other hand, some implementations based on the existing RELs support additional usage rights models that are not expressed in the REL. For example, the OMA DRM 1.0 implementation provides the node-locked model even though the REL version 1.0 has no means of specifying the node on which the application is being executed. The node-locked model is implemented by preventing the application from forwarding to any other node and does not require the presence of the usage rights file.

As table 6.2 shows, PARMA supports the widest range of usage rights models for software applications, and the PARMA architecture implements all of the models listed. While other RELs are more suitable for content on either mobile or desktop devices, PARMA is more suitable for the expression of usage rights on applications, primarily on mobile devices, but also desktops.

One particular advantage of PARMA is that it supports two models that have not yet been implemented on mobile devices, the feature-based model and the audit-based model.

The feature-based model supports the specification and management of usage rights for particular features within an application. Existing RELs are concerned with usage rights for the whole application and do not have a finer control over the execution of particular methods within the application. PARMA REL introduces a new element called *pointcut* that defines particular methods within the application for which usage rights can be enforced.

The audit-based model enables the implementation of a pay-per-use model without the need for a network connection from the device to a central usage rights server. Existing usage rights management systems that support pay-per-use models require the device to maintain and open a TCP/IP connection to the server for the duration of application execution. PARMA REL introduces elements that specify the nature of information that should be logged during application execution, saved in a secure storage area, and sent to the server periodically for pay-per-use billing.

For example, the license management system FLEXnet supports all of the usage rights models PARMA supports but these models are supported only on desktop devices and laptops. FLEXnet does not use a REL to express usage rights; the usage rights are expressed in license files in a proprietary format and often need to be verified on the server side of the architecture. The advantage of PARMA, when compared to FLEXnet, is that all of the defined usage rights models are supported on mobile devices as well. PARMA expresses usage rights using a REL that is based on ODRL and is compatible with ODRL and OMA DRM usage rights architectures.

6.2.2 Using AOP to Weave the Usage Rights Enforcement Code into Applications

The PARMA project uses a novel means of embedding the usage rights enforcement code in the original application. Usage rights are specified in a XML file and, using XSLT transformation sheets, transformed into aspect-oriented code. Aspect-oriented code encapsulates the usage rights management code and is maintained separately from the application code. Existing licensing systems require the insertion of calls to the licensing API within the application code. Using AOP, rights enforcement code is not inserted in the application itself, but it is specified in a separate class called an aspect, that is woven with the original application at compile time. Currently, no other existing rights application management architecture or DRM system uses AOP.

The following sections evaluate AOP and automated code generation against techniques used in other existing licensing management systems using example application that is a part of Sun's Wireless Developer's Toolkit [Mic04c]. A sample MIDP game provided with this toolkit, Push Puzzle, is used to demonstrate the changes that are required to the application in order to enable it for distribution using existing licensing systems and then PARMA.

The evaluation of AOP is performed in regard to the PARMA goals, rather than in regard to usual set of metrics used for the evaluation of aspect oriented programming. As there are no metrics suites that are commonly accepted for the evaluation of AOP, AOP is typically evaluated using the metrics suites for object-oriented systems [TCB04]. The evaluation of AOP using these suites measures code understandability, maintainability, reusability, and testability. However, a major goal of PARMA was to eliminate the need for modification of the original application source code while enabling it for usage rights management, and not to necessarily improve these characteristics of the application source code. PARMA achieves the goal that usage rights management source code should be completely separate from the application source code and that the addition of usage rights code should be possible without having any programming knowledge. The AOP approach makes this separation of the usage rights code and elimination of the programming knowledge possible.

Evaluation of AOP for Usage Rights Management

Using AOP gives PARMA several advantages over existing license management system:

- the original source code remains unchanged
- the licensing code is maintained in a single module rather than scattered around the application
- the amount of rights enforcement code that must be written and integrated with an application is minimized

If no changes are carried out to the original application source code, enabling and disabling the application for distribution through a particular license management system is simply a matter of recompiling the application with or without the additional source file containing all of the licensing code. In order to enable the application for distribution through FLEXnet or XSLM-based systems, the developer needs to add calls to the licensing API in the application source code. The addition of licensing code to the original application creates separate versions of source code for each licensing system through which the application is distributed. Existing license management systems generally support a few different APIs with different levels of licensing complexity. FLEXnet, for example, has six different APIs: Trivial API, Simple API, FLEXible API, Java API, LSAPI (Licensing API) and SLWG (Software License Working Group) [Mac04b]. If developers want to enable an application for licensing using each of these APIs, they need to develop six separate versions of the original application source code, each of them containing its own particular calls to the particular API. Similarly, XSLM-based license management systems have two levels of APIs, basic and advanced. The source code of the application that uses the basic API differs from the source code of the application using advanced API, so developers need to create two new separate versions of application source code. Having multiple copies of the application source code increases the development and implementation time required to keep all of the copies up-to-date. In some programming languages, such as C and C++, calls to different versions of APIs can be defined in a single source code using conditional compilation directives. However, this approach still requires changes to the application's source code in order to integrate license enforcement. In PARMA, only one version of the original source code is required. Any updates or changes are applied to a single version of source code, and are then recompiled with different aspect files to support different license management systems or different levels of API within a single license management system.

Apart from simplifying the maintenance of the original application, the separation of rights enforcement code and application code also simplifies the maintenance of the rights enforcement code. Storing enforcement code in separate classes from the application code enables switching between usage rights models and updates of the calls to the licensing API to be performed without the modifications to the application. In order to be enabled for distribution with a different usage rights model, the original application just needs to be recompiled with a different Java aspect file implementing new usage rights model.

Evaluation of Automated Generation of the Usage Rights Management Code

PARMA eliminates the need to modify the target application's source code to enable the license enforcement by using AOP. However, specifying the AOP source code still requires knowledge of the licensing APIs as well as knowledge of aspect oriented programming syntax. To completely avoid any additional programming by the developer in regard to enabling the application for licensing, the generation of aspect-oriented licensing code and its weaving in the original application is performed using support tools.

For traditional licensing models, such as node-locked and user-locked, developers do not need to perform any additional actions apart from specifying that they wish to offer the application for distribution using those licensing models. The only PARMA usage rights model that requires additional developer interaction is the feature-based model, and the pay-per-use audit-based model in circumstances where developers wish the application to be billed based on the usage of a particular feature. However, not even the feature-based model requires any additional code insertions by the developer; the developer only needs to specify the signature of the method(s) implementing that feature. The information on methods is saved as a pointcut element in the PARMA usage rights file and transformed into an aspect-oriented pointcut in the aspect class containing the license enforcement code, and then woven into the original application.

	progr. knowledge	knowledge of API	changes to src-code	# of models requiring developer action
FLEXnet	yes	yes	yes	all
XSLM	yes	yes	yes	all
PARMA	no	no	no	1

Table 6.3: PARMA vs other LMS

As shown in table 6.3, PARMA is the only usage rights management system that does not require knowledge of the proprietary API or programming. Only one usage model requires vendor intervention, while in all of the other license management systems, available usage rights models require the developer to perform changes on the original source code.

The above advantages of PARMA over FLEXnet and XSLM-based licensing systems are presented using two examples implementing a straight-forward node-locked rights usage model and a fine-grained feature-based model using these license management systems.

Node-locked Model

The node-locked model is one of the simpler usage rights models to implement. At application start up, license enforcement code needs to check that the identification of the node running the application is the same as the identification of the licensed node specified in the usage rights file. No further validation or enforcement is required during application execution or at application shut down.

In XSLM, since it is only a proposed standard, details of the implementation of particular licensing models are not specified. However, it is specified that at the beginning of every application execution, the application needs to make a call to the licensing server to request the license. At the end of application execution, regardless of the licensing model implemented, it has to return the checked-out license to the server. The code below shows the PushPuzzle file, with a minimum number of calls inserted that enable the application to be licensed using the XSLM-based system. These calls need to be added to the first method that runs when the application starts, startApp, and to the last method that runs before the application exits, destroyApp.

```
public void startApp()
{
 if (xslm_basic_request_license( )==STATUS_OK)
  display.setCurrent(canvas);
 else
  //display error and exit
}
```

```
 public void destroyApp(boolean unconditional)
 {
  display.setCurrent((Displayable)null);
  canvas.destroy();
  if (score != null)
   score.close();
  xslm_basic_release_license( )
 }
```

The number of calls can increase if the implementation requires confirmation that a license is still in use throughout execution, or if the implementation of the license model requires use of the advanced API.

In the FLEXnet implementation, usage rights on separate features are specified in a license file stored on a device (where the feature can also be a full application). Prior to the execution of every specified feature, or an application, the license file needs to be checked to verify that the user/device has permissions to execute the application. At the end of execution, the license is checked back in with the system. In MIDP applications, the license needs to be checked out for execution at application start up, in the startApp method, and returned back to the licensing system at the end of execution, in the destroyApp method. Therefore, to enable the application for a node-locked model in FLEXnet, the application source code of the two methods must be modified.

```
public void startApp()
{
 if (lp_checkout(, , feature, version, , license file loc, ))
  display.setCurrent(canvas);
 else
  //display error and exit
}
```

```
public void destroyApp(boolean unconditional)
{
 display.setCurrent((Displayable)null);
 canvas.destroy();
 if (score != null)
  score.close();
 lp_checkin();
}
```

In PARMA, to enable the application for distribution using the node-locked model, the developer does not need to perform any additional activities or add any code to the original application (see section 6.1.2). A license generation tool automatically creates the code, that will be, during the execution, executed before the startApp method. However, this code is not inserted in the original application, but is held separately in the aspect file, containing the code as shown below.

```
pointcut pcut_startApp() : call (void example.pushpuzzle.startApp());
before() : pcut_startApp()
{
 //implementation specific usage rights enforcement code goes here
}
```

Table 6.4 compares the amount of code an application developer needs to add to the original application to enable support for the node-locked usage model using FLEXnet, XSLM and PARMA. It also shows the number of places where the original application source code needs to be modified to add this licensing code. It is clear that PARMA requires no code additions by the developer and does not modify the original source code, while other licensing systems do it in a minimum of two to three places.

	# of lines added by developer	# of places orig source was modified
FLEXnet	min 2+1	1+1
XSLM-based	min 2+1	2+1
PARMA	0	0

Table 6.4: Comparison of PARMA and other LMS for node-locked model

Apart from having to make calls to the usage rights API, the usage rights package needs to be imported in every class that will call the usage rights API. Imports of these packages add one extra line per original class (+1 in the table 6.4).

Feature-based Model

The advantage of automated code generation and encapsulating usage rights in Aspects becomes even more obvious when enabling applications for licensing using fine-grained usage rights models, such as a feature-based model. In this model, the usage rights file needs to be checked prior to the execution of every specified feature. The following example shows a trial version of the PushPuzzle game that has the sounds disabled, the undo option disabled, and the option of saving a high score disabled.

The sample application consists of four classes. The PushPuzzleCanvas class contains the method for playing sounds that needs to be disabled. The method for saving the highest scores, that needs to be disabled, is part of the Score class. The board class contains the method for undoing the move that also must be disabled for a demo version in this example.

XSLM does not specify details of particular model implementations, it only specifies methods for checking out and checking in the license with the licensing server. Therefore, every specified feature must check out a separate license. If a user is licensed on the execution of that feature, execution is permitted and the license needs to be checked back in with the server at the end of execution. The API call `xslm_basic_request_license()` must be called at the beginning of the method for saving the high score, at the beginning of the method for undoing the move, and at the beginning of the method for playing sounds. At the end of each of these methods `xslm_basic_release_license()` must be called.

In FLEXnet, the license file specifies permissions either on a full application, or on separate features. For the feature-based model, a license file contains permissions per feature, and a license must be checked out prior to the execution of every separately licensed feature. This approach requires licensing code to be added at the beginning of the execution of every feature. Therefore, the API call `lp_checkout`() must be called at the beginning of the method for saving the high score, at the beginning of the method for undoing the move, and at the beginning of the method for playing sounds. At the end of each of these methods, `lp_checkin()` must be called, to return the checked-out license to the server.

Figure 6.12 shows the points in the original source code of the Push Puzzle game where the calls to the licensing API of FLEXnet system must be inserted to enable the above described feature-based model. Licensing code is scattered around three classes, lowering modularization of the code. Code is scattered in the same manner using the above described XSLM license management system.

In the PARMA feature-based model, unlike the node-locked model, a minor intervention of the de-

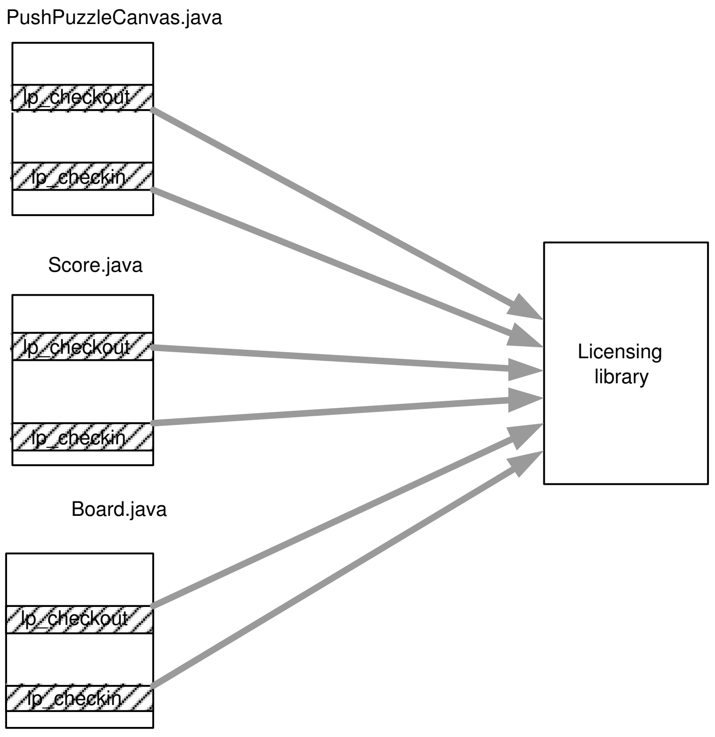


Figure 6.12: Feature-based licensing with FLEXnet and XSLM

veloper, at the time of the application submission, is required. The developer must provide the method signatures of the methods comprising the separately licensed features, for the usage rights generation tool to know where to insert the automatically generated code. From these signatures, a usage rights file is created. A usage rights file that the generation of code for this example is based on, is provided in appendix A.2.3. This usage rights file is converted into a separate Java aspect class, containing the code shown below.

```
pointcut pcut_setLevelScore (int variable_1, int variable_2) :
call (boolean example.pushpuzzle.Score.setLevelScore(int, int))
&& args( variable_1, variable_2);
before(int variable_1, int variable_2) : pcut_setLevelScore( variable_1, variable_2)
{
//implementation-specific usage rights enforcement code
}
pointcut pcut_undoMove () :
call (int example.pushpuzzle.Board.undoMove()) && args();
before() : pcut_undoMove()
{
//implementation-specific usage rights enforcement code
}
pointcut pcut_play (byte[] variable_3) :
call (void example.pushpuzzle.PushPuzzleCanvas.play(byte[]))
&& args( variable_3);
before(byte[] variable_3) : pcut_play( variable_3)
{
//implementation-specific usage rights enforcement code
}
```

The aspect class defines three pointcuts that refer to the three methods where execution should be intercepted to perform license validation. Even though license enforcement code is specified in a separate file and no reference to it exists in the original methods, using AOP, this code will be executed before three restricted methods are called.

Figure 6.13 shows how license enforcement code is separated in a single file using AOP, without modifying the original source code of the Push Puzzle game. Modularity of the code is improved; the application and licensing code are loosely-coupled enabling both to be changed independently of each other. The aspects are now only tightly coupled to a pointcut in source code. The code in the aspect can be modified independently of the target classes.

PARMA, FLEXnet and XSLM are compared in table 6.5 with regard to the amount of code required to be added by the developer and the number of locations within the source code where the application needs to be modified in order to enable the feature-based model that disables three features. PARMA

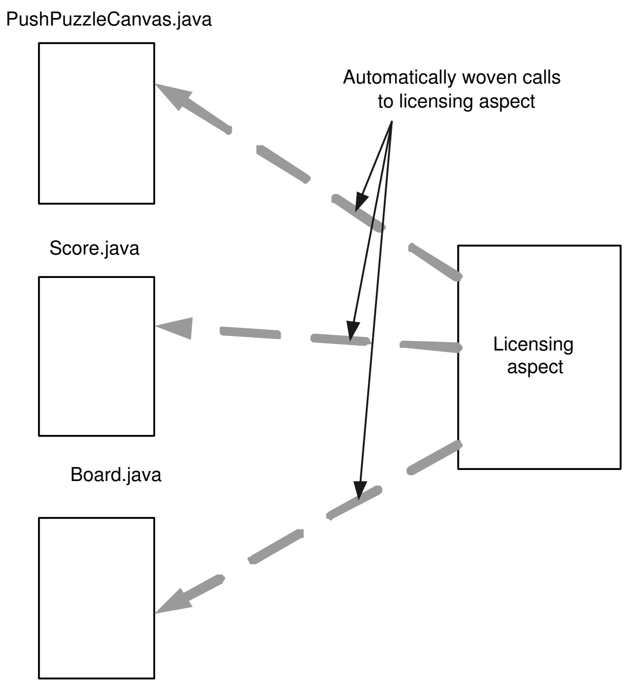


Figure 6.13: Improved application modularity using PARMA usage rights models

requires no code additions by the developer or changes to the original source code, while other license management systems require modifications to at least three different locations in the source code and the addition of at least 6 lines of licensing code.

	# of lines of code added by developer	# of places orig source was modified
FLEXnet	min 6+3	3+3
XSLM-based	min 6+3	3+3
PARMA	0	0

Table 6.5: Comparison of PARMA and other LMS for feature-based model

As in the implementation of the node-locked model, the feature-based model usage rights package must be imported in every class that will call the usage rights API, increasing the number of lines required to be added by the developer by 3, one for each class in the implementation (table 6.5).

Analysing the code examples from the node-locked and feature-based models, we see that different usage rights models in FLEXnet and XSLM require the addition of code to different methods and classes within the application. In PARMA, all of the changes relating to the change of the usage rights model are done in a single file, the aspect source file. Therefore, if an application is to be enabled for distribution with

x number of different usage rights models using FLEXnet and XSLM, ***x*** different versions of application source code need to exist. This number can be reduced using design patterns, such as the abstract factory pattern, however, using PARMA, only one version of the source code need exist. The original source code is compiled with different aspect files to generate different versions of the application or different usage rights models, rather than maintaining different versions of the original source code.

6.2.2.1 Limitations of PARMA AOP Approach

Automatic generation of rights enforcement code in the current version of PARMA has been implemented only for MIDP applications. The AOP implementation for Java used in PARMA, AspectJ, is primarily aimed at the development of J2SE code, so J2ME (MIDP) code generation was additionally challenging due to the smaller subset of java classes it supports. J2SE has a broader set of capabilities than J2ME, so extending the implementation to support J2SE is simple and straight forward once the J2ME version is implemented. One of the disadvantages of MIDP over J2SE, is that it lacks the java.lang.reflection API that AspectJ aspects uses to obtain the runtime information on the application code. Currently, rights enforcement code cannot obtain the runtime values of variables which can represent a limitation on certain usage rights models. As mobile devices become more powerful, J2ME versions may be extended to support a larger subset of J2SE, including the reflection API, and overcome this problem.
PARMA does not fully utilize all of the capabilities of aspect-oriented programming as its primary research focus was not AOP. PARMA uses only one type of AOP join-point, the method call, out of several available ones (e.g. method call, method execution, constructor join point, field access). Also, PARMA uses only three types of advice: before, around, and after, and does not differentiate between the *after returning* and *after throwing* types of after advice [Kis02]. Specific implementations of the client side enforcement architecture can extend PARMA to support any of the other capabilities of AOP.

6.2.3 Pervasive Rights Management

The use of Web Services technologies to implement the PARMA architecture enables the integration of rights management for applications regardless of the programming language the application is developed in, or platform it is running on. PARMA registration and application management interfaces are described using WSDL. WSDL describes the format of remote calls that clients can make to the service. The underlying implementation of the exposed methods is implemented in Java, but it can be implemented in any language and can be modified without changes to the WSDL interfaces or web services clients [CW04].

Clients invoke web services over the SOAP protocol. PARMA clients must either support an implementation of SOAP or the HTTP(S) protocol in order to access PARMA interfaces.

A rights specification and generation tool is provided as a Java servlet, accessed over the HTTP(S) protocol, so rights can be requested, specified and generated by any client with a HTML browser and a HTTP(S) connection. If the target device does not have HTML/HTTP(S) support, usage files can be created on a different device and delivered separately from the application over any available communication channel. Usage rights files are expressed in XML, so they can be interpreted by any device with XML support.

Such a design enables PARMA registration and application management interfaces to be called from any device supporting SOAP over HTTP(S), and usage rights files can be interpreted on any device capable of processing XML files. Desktops, smartphones, PDAs, and mobile phones with MIDP 2.0 support, that are a primary target of the PARMA architecture, all meet the above requirements.

Platform independence and minimal client requirements are a distinct advantage of PARMA when compared to the currently most widely used license management system FLEXnet. FLEXnet is supported on 26 different platforms, but there is a separate version for each platform, making upgrades and maintenance more difficult for both the customer and the license management system vendor. FLEXnet client side is also platform-dependent and there is no edition for Symbian, Palm OS, or a Windows CE. Therefore, FLEXnet does not support smartphones and PDAs as clients [Mac04a].

The proposed license management standard XSLM was designed to solve the problems of interoperability between different licensing systems on different platforms, but as explained in chapter 2, a full implementation of this standard does not yet exist.

Another important characteristic of the PARMA architecture, apart from platform independence, is its support for usage rights validation and enforcement without the need for an immediate network connection to the licensing server. Devices in pervasive environments do not always necessarily have an Internet connection so normal application execution and license enforcement should not make the assumption of an available network connection. The majority of PARMA usage models do not require an Internet connection for the usage rights management process. If necessary, both the application and usage rights can be transferred to one device from another device over short range protocols, so an expensive Internet connection is not required at any stage of the usage rights management process. All of the usage rights models except the audit-based and the concurrent model do not need a network connection for rights enforcement. The audit-based and concurrent model need an occasional connection to the rights management server in order to renew usage rights. This connection can be established at any time suitable for the client, so long as it is within the maximum period specified by the application vendor. The maximum period that an application can run without a network connection in these two models is configurable by the vendor but the typical length is a month for the audit-based usage model, and a week for the concurrent model.

Support for the disconnected application execution on the mobile devices gives PARMA an advantage when compared to other usage rights management systems. FLEXnet does not require a network connection for any model apart from the concurrent model. However, the only devices supported by FLEXnet are desktops and laptops, so even though rights enforcement does not require a network connection, there is no support for rights enforcement on mobile devices [Mac04a].

The XSLM standard requires that a licensing session is established at the beginning of application usage and therefore it requires that a network connection is available for application execution [XSL99].

Some licensing models for consumer mobile applications, such as the models supported by OMA DRM, do not require an Internet connection for rights enforcement. However, flexible usage models such as pay-per-play are supported only when devices are connected to the Internet. PARMA supports an audited pay-per-play model (audit-based model) that gathers usage data on the device and transfers it to the server in time intervals set by vendor.

6.3 Evaluation Summary

This chapter showed PARMA usage from the perspective of an end-user and an application provider. The steps for user registration, application browsing and download, as well application registration and download are simple and straight forward. The chapter discussed how and to what extent the goals of PARMA were met. A variety of usage rights models are supported, including models specifically tailored to occasionally connected devices, such as audit-based, and to the fine-grained control of application execution, such as feature-based. Usage rights enforcement is integrated into the application using usage rights files converted into AOP code and woven into the original application, removing the need for the addition of the enforcement source code in the original application by the application developer. PARMA is suitable for usage by a variety of devices, as the only criteria that devices need to satisfy in order to use PARMA is to support HTTP(S) and SOAP protocols. These three characteristics make PARMA suitable for application management in pervasive environments.

Chapter 7

Conclusion

This chapter concludes the thesis by summarizing the main findings of the research presented in the thesis and gives directions for future work in the area of pervasive application usage rights management.

7.1 Conclusion

This dissertation investigated the development of software licensing and digital rights management on applications developed for stand-alone computers, and computers networked via LANs, wide-area networks, and mobile networks. Developments in computing require changes in the way software licenses are delivered to target devices, validated, enforced, and revoked. Early software licenses, known as shrink-wrapped licenses, had no technical means of license enforcement or revocation; they relied on copyright law to ensure license compliance. Later, serial numbers and activation keys were introduced to validate the licenses. Also, license expiration based on a specified time-period was introduced. With the development of LANs, new licensing models have emerged. Unique hardware identification based on the MAC address enabled locking of a license to a single node, and unique usernames on the network enabled locking of a license to a single user. Concurrent licensing models were also implemented, by allowing a specified maximum number of instances of the application to run on the network at the same time, regardless of the identity of the device or the user running them. LANs also enabled the development of license management systems. License management systems are distributed applications that manage the specification, generation, delivery, validation, enforcement and revocation of software licenses, by embedding licensing code into the target application. The wider use of mobile devices, such as PDAs and mobile phones, have resulted in further changes to the way software is licensed, particularly on mobile phones. New means of identification that characterise mobile phones, e.g. MSISDN and IMEI, have enabled locking of the software license to a particular device. Also, DRM systems, that have traditionally targeted digital content, have been applied to the applications as well.

The dissertation was particularly concerned with license management in a pervasive environment, investigating the requirements for license management systems and digital rights management systems implemented in this environment. Existing LMS and DRM systems are suitable for desktop and LAN environments they were original developed for, but have several pitfalls when applied to the pervasive environment. DRM systems have a primary focus on content rather than applications, and as such they lack support for flexible fine-grained usage models for applications. Existing LMS are primarily designed for the desktop environment and make an assumption of constant network connectivity. Often, they do not support application execution and license validation if a connection to the licensing server is not available. In a pervasive environment, devices are occasionally connected to each other and licensing servers, so the assumption of devices having constant connectivity cannot be applied. Also, LMSs require developers to modify the source code of the application by inserting the calls to the licensing API, and consequently increasing the time required for application development. A particular lack of suitable usage rights management systems exists in the enterprise environment, where all applications purchased by the enterprise, for both mobile and desktop devices, should be managed from a single point.

The thesis introduced PARMA, a pervasive rights management architecture, that takes a novel approaches to the several stages of usage rights management, to improve usage rights management in pervasive environments.

The PARMA architecture consists of two digital rights management servers, the VDS and the EDS. The VDS manages usage rights for the individual end-users and for the enterprise customers. The EDS manages usage rights for employees within a single company, where that company is an enterprise customer of the VDS. Application vendors can host the VDS themselves, or upload their application to a different VDS for distribution.

Web services are used for the implementation of PARMA to enable the integration of usage rights management for all applications regardless of the platform they are running on and language they are developed in. This approach is particularly suitable for the enterprise environment because it enables integration with existing back-end systems. The loosely-coupled design approach of PARMA also enables changes and upgrades to the server side of the usage rights management system without affecting existing clients, so long as the method signatures of methods exposed by the servers are not modified. Methods are described using WSDL interfaces, and the underlying implementation of methods can be modified without requiring modifications on PARMA clients.

Usage rights models offered by PARMA are specified using PARMA REL. PARMA REL is extended from an existing standardised REL, ODRL, to introduce two novel usage rights models that have not yet been implemented on mobile devices, the audit-based model and the feature-based model. The audit-based model implements a post-paid pay-per-use model by logging the information on application usage, storing it on the device, and periodically transmitting it to the licensing server for billing. The feature-based model enables the specification of licensing terms for particular methods within the application, rather than the full application. Usage rights management systems that support these fine-grained usage rights models offer more flexibility for customers when purchasing the application and application usage rights. The list of usage rights models is not fixed, as PARMA REL allows the addition of new usage rights models if needed. These usage rights models in PARMA are designed so that their enforcement does not require network connection, as they do in LMSs developed for desktop environments. Server side implementation of these models is provided by this thesis and requires a connection with a client only at the time of an application and a usage rights file download. The only time when communication is required after that point is in the case of an audit-based usage model, and the frequency of that communication is defined by the administrator, usually once a week or once a month.

Enforcement of these usage rights models in PARMA does not depend on the availability of a network connection in mobile environments, as it does in the LMSs developed for the desktop environments. Network outages are more common in the pervasive environment and wide-area network traffic is costly, so PARMA minimizes the amount of wide-area network traffic and allows uninterrupted application execution even if the network is not available.

Application developers who wish to distribute their applications through PARMA do not add licensing to the application themselves. Developers only need to specify the usage rights models for the application. License enforcement code is automatically generated from the usage rights files and woven into the original application using AOP. PARMA showed that a reduction in the developers involvement in application licensing is achieved by automating the generation of the code for usage rights validation and enforcement. PARMA also showed that using AOP to specify usage rights management code in a single file and leaving the original application source code intact, greatly improves the modularity of application code. Consequently, it simplifies the maintenance of both application source code and the license enforcement code. Automatic generation of usage rights enforcement code using XSLT removes the need for developers' interaction in the licensing process, and eliminates the need for developers to learn proprietary APIs in order to deliver their application through license management systems.

7.2 Future Work

This thesis has made several contributions towards an integrated licensing platform by supporting a variety of usage models and the automatic generation of license enforcement code. However, the implementation of PARMA can be further enhanced to support programming languages and platforms not yet supported and to use the full capabilities of aspect-oriented programming.

The current version of PARMA only supports automatic generation of MIDP 2.0 source code that can run on mobile phones. Additional XSLT transformation sheets can be added to support the transformation of usage rights file into programming languages other than Java, and to support usage rights management for applications developed for other platforms. As license enforcement code is woven into the original application using aspect-oriented programming, PARMA can be extended to support applications developed in any language that has an implementation of AOP, e.g. AspectC.

Further work can be done on minimizing the footprint of the SOAP clients interacting with the licensing server. Smaller SOAP clients can run directly on resource constrained devices, rather than communicating with SOAP clients via servlets, as it is the case in the current PARMA implementation.

PARMA can also be integrated with client side enforcement architectures and payment services to provide an integrated usage rights management and payment solution for individual and enterprise customers. Once the client and server sides of the architecture are integrated, the process of discovering suitable applications for the devices can be automated, rather than achieved manually as is the case in existing PARMA implementation.

Appendix A

PARMA REL

A.1 PARMA Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:schema.parma.tcd.ie"
elementFormDefault="qualified" attributeFormDefault="qualified" version="1.0"
xmlns:o-ex="http://odrl.net/1.1/ODRL-EX" xmlns:parma="urn:schema.parma.tcd.ie"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns1="http://odrl.net/1.1/ODRL-DD">
<xsd:import namespace="http://odrl.net/1.1/ODRL-EX"
schemaLocation="http://odrl.net/1.1/ODRL-EX-11.xsd"/>
<xsd:import namespace="http://odrl.net/1.1/ODRL-DD"
schemaLocation="http://odrl.net/1.1/ODRL-DD-11.xsd"/>
<xsd:annotation>
<xsd:documentation xml:lang="en">
parma Digital Rights Management Schema for Trinity College.
Copyright 2003 Trinity College Dublin, Computer Science Department,
Distributed Systems Group. All rights reserved.
</xsd:documentation>
</xsd:annotation>
<xsd:element name="RightsType" type="xsd:string" substitutionGroup="o-ex:contextElement"/>
<xsd:element name="ServerType" substitutionGroup="o-ex:contextElement">
 <xsd:simpleType>
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="EDS"/>
   <xsd:enumeration value="VDS"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<xsd:element name="IMEI" type="xsd:nonNegativeInteger" substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="userID" substitutionGroup="o-ex:constraintElement">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="o-ex:constraintType">
```

```
    <xsd:sequence>
     <xsd:element name="name" type="xsd:string" minOccurs="0"/>
     <xsd:element name="registeredID" type="xsd:string" minOccurs="0"/>
     <xsd:element name="mobileNumber" type="xsd:string" minOccurs="0"/>
     <xsd:element name="payType" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
<xsd:element name="disabledFeatures" substitutionGroup="o-ex:constraintElement">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="feature" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name="auditLogData" substitutionGroup="o-ex:requirementElement">
 <xsd:complexType>
  <xsd:attribute name="dateTime" type="xsd:boolean" use="optional" default="1"/>
  <xsd:attribute name="duration" type="xsd:boolean" use="optional" default="1"/>
  <xsd:attribute name="accumulated" type="xsd:boolean" use="optional" default="1"/>
 </xsd:complexType>
</xsd:element>
<xsd:element name="pointcut" substitutionGroup="o-ex:constraintElement">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="adviceType" maxOccurs="unbounded">
    <xsd:simpleType>
     <xsd:restriction base="xsd:string">
      <xsd:enumeration value="before"/>
      <xsd:enumeration value="after"/>
      <xsd:enumeration value="around"/>
     </xsd:restriction>
    </xsd:simpleType>
   </xsd:element>
   <xsd:element name="package">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="packageName"/>
       <xsd:choice>
        <xsd:element name="package"/>
        <xsd:element name="class">
         <xsd:complexType>
          <xsd:sequence>
           <xsd:element name="className" type="xsd:string" minOccurs="0"
            maxOccurs="unbounded"/>
           <xsd:element name="method">
            <xsd:complexType>
             <xsd:sequence>
              <xsd:element name="methodName" type="xsd:string" minOccurs="0"
               maxOccurs="unbounded"/>
              <xsd:element name="returnType" type="xsd:string" minOccurs="0"/>
              <xsd:element name="argType" type="xsd:string" minOccurs="0"
               maxOccurs="unbounded"/>
             </xsd:sequence>
            </xsd:complexType>
           </xsd:element>
          </xsd:sequence>
         </xsd:complexType>
        </xsd:element>
       </xsd:choice>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

A.2 Sample Usage Rights Files

A.2.1 Node-Locked Model

```
<?xml version="1.0" encoding="UTF-8"?>
<o-ex:rights xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
xmlns:parma="urn:schema.parma.tcd.ie"
xmlns:ns1="http://odrl.net/1.1/ODRL-DD"
xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:schema.parma.tcd.ie F:\c_direct\parma-10.xsd">
<o-ex:agreement>
 <o-ex:context>
  <parma:RightsType>NodeLocked</parma:RightsType>
 </o-ex:context>
 <o-ex:asset>
  <o-ex:context>
   <o-dd:dLocation>URIofApplication</o-dd:dLocation>
  </o-ex:context>
 </o-ex:asset>
 <o-ex:party>
  <o-ex:context>
   <o-dd:dLocation>URIofDRMServer</o-dd:dLocation>
```

```
    <parma:ServerType>EDS</parma:ServerType>
   </o-ex:context>
  </o-ex:party>
  <o-ex:permission>
   <o-dd:execute>
    <o-ex:constraint>
     <parma:IMEI>99999999999999</parma:IMEI>
    </o-ex:constraint>
   </o-dd:execute>
  </o-ex:permission>
</o-ex:agreement>
</o-ex:rights>
```

A.2.2 Audit-Based Model

```
<?xml version="1.0" encoding="UTF-8"?>
<o-ex:rights xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
xmlns:parma="urn:schema.parma.tcd.ie"
xmlns:ns1="http://odrl.net/1.1/ODRL-DD"
xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:schema.parma.tcd.ie F:\c_direct\parma-10.xsd">
<o-ex:agreement>
  <o-ex:context>
   <parma:RightsType>PayPerUse-Audit</parma:RightsType>
  </o-ex:context>
  <o-ex:asset>
   <o-ex:context>
    <o-dd:dLocation>URIofApplication</o-dd:dLocation>
   </o-ex:context>
  </o-ex:asset>
  <o-ex:party>
   <o-ex:context>
    <o-dd:dLocation>URIofDRMServer</o-dd:dLocation>
    <parma:ServerType>EDS</parma:ServerType>
   </o-ex:context>
  </o-ex:party>
  <o-ex:permission>
   <o-dd:execute/>
  </o-ex:permission>
  <o-ex:requirement>
   <parma:auditLogData parma:dateTime="true" parma:duration="true" parma:accumulated="false"/>
   </o-ex:requirement>
  </o-ex:agreement>
</o-ex:rights>
```

A.2.3 Feature-Based Model

```
<?xml version="1.0" encoding="UTF-8"?>
<o-ex:rights xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
xmlns:parma="urn:schema.parma.tcd.ie"
xmlns:ns1="http://odrl.net/1.1/ODRL-DD"
xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:schema.parma.tcd.ie F:\c_direct\parma-10.xsd">
 <o-ex:agreement>
  <o-ex:context>
   <parma:RightsType>PayPerUse-Audit</parma:RightsType>
  </o-ex:context>
  <o-ex:asset>
   <o-ex:context>
    <o-dd:dLocation>URIofApplication</o-dd:dLocation>
   </o-ex:context>
  </o-ex:asset>
  <o-ex:party>
   <o-ex:context>
    <o-dd:dLocation>URIofDRMServer</o-dd:dLocation>
    <parma:ServerType>EDS</parma:ServerType>
   </o-ex:context>
  </o-ex:party>
 <o-ex:permission>
  <o-dd:execute>
 <o-ex:constraint>
 <parma:pointcut>
  <parma:adviceType>before</parma:adviceType>
   <parma:package>
    <parma:packageName>example.pushpuzzle</parma:packageName>
    <parma:class>
     <parma:className>Board</parma:className>
     <parma:method>
      <parma:methodName>undoMove</parma:methodName>
      <parma:returnType>int</parma:returnType>
     </parma:method>
    </parma:class>
   </parma:package>
  </parma:pointcut>
 <parma:pointcut>
  <parma:adviceType>before</parma:adviceType>
   <parma:package>
    <parma:packageName>example.pushpuzzle</parma:packageName>
    <parma:class>
     <parma:className>Score</parma:className>
     <parma:method>
```

```
      <parma:methodName>setLevelScore</parma:methodName>
      <parma:returnType>boolean</parma:returnType>
      <parma:argType>int</parma:argType>
      <parma:argType> int</parma:argType>
     </parma:method>
    </parma:class>
   </parma:package>
 </parma:pointcut>
 <parma:pointcut>
  <parma:adviceType>before</parma:adviceType>
   <parma:package>
    <parma:packageName>example.pushpuzzle</parma:packageName>
    <parma:class>
     <parma:className>PushPuzzleCanvas</parma:className>
      <parma:method>
       <parma:methodName>play</parma:methodName>
       <parma:returnType>void</parma:returnType>
       <parma:argType>byte[]</parma:argType>
      </parma:method>
     </parma:class>
    </parma:package>
   </parma:pointcut>
  </o-ex:constraint>
 </o-dd:execute>
</o-ex:permission>
</o-ex:agreement>
</o-ex:rights>
```

A.2.4 Metered Model

```
<?xml version="1.0" encoding="UTF-8"?>
<o-ex:rights xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
xmlns:parma="urn:schema.parma.tcd.ie"
xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:schema.parma.tcd.ie F:\c_direct\parma-10.xsd">
<o-ex:agreement>
 <o-ex:context>
  <parma:RightsType>Metered</parma:RightsType>
 </o-ex:context>
 <o-ex:asset>
  <o-ex:context>
   <o-dd:dLocation>URIofApplication</o-dd:dLocation>
  </o-ex:context>
 </o-ex:asset>
 <o-ex:party>
```

```
  <o-ex:context>
   <o-dd:dLocation>URIofDRMServer</o-dd:dLocation>
   <parma:ServerType>EDS</parma:ServerType>
  </o-ex:context>
 </o-ex:party>
 <o-ex:permission>
  <o-dd:execute>
   <o-ex:constraint>
    <o-dd:accumulated>P30H</o-dd:accumulated>
   </o-ex:constraint>
  </o-dd:execute>
 </o-ex:permission>
</o-ex:agreement>
</o-ex:rights>
```

Appendix B

WSDL Interfaces

B.1 Registration Interface

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:registration.vds_eds.parma.tcd.ie"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="urn:registration.vds_eds.parma.tcd.ie"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
xmlns:vds="urn:common.parma.tcd.ie"
xmlns:parma="urn:registration.vds_eds.parma.tcd.ie">
<wsdl:types>
 <xsd:schema targetNamespace="urn:registration.vds_eds.parma.tcd.ie"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:soapenc="xsd:string"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:vds="urn:common.parma.tcd.ie"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
 <xsd:import location="parma-10.xsd" namespace="urn:schema.parma.tcd.ie"/>
 <xsd:import namespace="urn:common.parma.tcd.ie" schemaLocation="VDS_EDS_DataTypes.xsd"/>
 </xsd:schema>
</wsdl:types>
<wsdl:message name="loginResponse">
 <wsdl:part name="loginReturn" type="vds:LoginReturn"/>
</wsdl:message>
<wsdl:message name="loginRequest">
 <wsdl:part name="id" type="xsd:long"/>
 <wsdl:part name="uname" type="xsd:string"/>
 <wsdl:part name="pass" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="changePasswordResponse">
 <wsdl:part name="changePasswordReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="changePasswordRequest">
 <wsdl:part name="id" type="xsd:long"/>
 <wsdl:part name="uname" type="xsd:string"/>
 <wsdl:part name="old_pass" type="xsd:string"/>
 <wsdl:part name="new_pass" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="queryForgottenAuthResponse">
 <wsdl:part name="queryForgottenAuthReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="queryForgottenAuthRequest">
 <wsdl:part name="id" type="xsd:long"/>
 <wsdl:part name="uname" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="submitForgottenAuthResponse">
 <wsdl:part name="submitForgottenAuthReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="submitForgottenAuthRequest">
<!--in case of individual its secret answer, in case of company its a contact person-->
 <wsdl:part name="in0" type="xsd:string"/>
 <wsdl:part name="id" type="xsd:long"/>
 <wsdl:part name="uname" type="xsd:string"/>
 </wsdl:message>
<wsdl:message name="queryRightsListResponse">
 <wsdl:part name="queryRightsListResponse" type="vds:RightsList"/>
</wsdl:message>
<wsdl:message name="queryRightsListRequest">
</wsdl:message>
<wsdl:message name="submitContactInfoResponse">
 <wsdl:part name="submitContactInfoReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="submitContactInfoRequest">
 <wsdl:part name="contact_info" type="vds:ContactInformation"/>
 <!-- id from ClientId-->
 <wsdl:part name="id" type="xsd:string"/>
 <wsdl:part name="current_version" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="registerResponse">
 <wsdl:part name="registerReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="registerRequest">
 <wsdl:part name="client_id" type="vds:ClientId"/>
 <wsdl:part name="current_version" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="unregisterResponse">
 <wsdl:part name="unregisterReturn" type="xsd:boolean"/>
</wsdl:message>
<wsdl:message name="unregisterRequest">
 <wsdl:part name="id" type="xsd:long"/>
 <wsdl:part name="pass" type="xsd:string"/>
 <wsdl:part name="service_id" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="submitServiceReqsResponse">
 <wsdl:part name="submitServiceReqsReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="submitServiceReqsRequest">
 <wsdl:part name="service_id" type="xsd:long"/>
 <wsdl:part name="requirements" type="vds:ServiceReqs"/>
 <wsdl:part name="current_version" type="xsd:string"/>
 <wsdl:part name="phone_models" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="submitServiceDetailsResponse">
 <wsdl:part name="submitServiceDetailsReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="submitServiceDetailsRequest">
 <wsdl:part name="service_id" type="xsd:long"/>
 <wsdl:part name="details" type="vds:ServiceDetails"/>
 <wsdl:part name="current_version" type="xsd:string"/>
 <wsdl:part name="client_id" type="vds:ClientId"/>
</wsdl:message>
<wsdl:message name="submitServiceRightsOptionsResponse">
 <wsdl:part name="submitServiceRightsOptionsReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="submitServiceRightsOptionsRequest">
 <wsdl:part name="service_id" type="xsd:long"/>
 <wsdl:part name="rights_options" type="vds:RightsOptions"/>
 <wsdl:part name="client_id" type="vds:ClientId"/>
</wsdl:message>
<wsdl:portType name="EDSRegistration">
 <wsdl:operation name="login" parameterOrder="id uname pass">
  <wsdl:input message="impl:loginRequest" name="loginRequest"/>
  <wsdl:output message="impl:loginResponse" name="loginResponse"/>
 </wsdl:operation>
 <wsdl:operation name="changePassword" parameterOrder="id uname old_pass new_pass">
  <wsdl:input message="impl:changePasswordRequest" name="changePasswordRequest"/>
  <wsdl:output message="impl:changePasswordResponse" name="changePasswordResponse"/>
 </wsdl:operation>
 <wsdl:operation name="queryForgottenAuth" parameterOrder="id uname">
  <wsdl:input message="impl:queryForgottenAuthRequest" name="queryForgottenAuthRequest"/>
  <wsdl:output message="impl:queryForgottenAuthResponse" name="queryForgottenAuthResponse"/>
 </wsdl:operation>
 <wsdl:operation name="submitForgottenAuth" parameterOrder="in0 id uname">
  <wsdl:input message="impl:submitForgottenAuthRequest" name="submitForgottenAuthRequest"/>
  <wsdl:output message="impl:submitForgottenAuthResponse" name="submitForgottenAuthResponse"/>
 </wsdl:operation>
 <wsdl:operation name="queryRightsList" parameterOrder="right_name">
  <wsdl:input message="impl:queryRightsListRequest" name="queryRightsListRequest"/>
  <wsdl:output message="impl:queryRightsListResponse" name="queryRightsListResponse"/>
 </wsdl:operation>
 <wsdl:operation name="submitContactInfo" parameterOrder="contact_info id current_version">
  <wsdl:input message="impl:submitContactInfoRequest" name="submitContactInfoRequest"/>
  <wsdl:output message="impl:submitContactInfoResponse" name="submitContactInfoResponse"/>
 </wsdl:operation>
 <wsdl:operation name="register" parameterOrder="client_id current_version">
  <wsdl:input message="impl:registerRequest" name="registerRequest"/>
  <wsdl:output message="impl:registerResponse" name="registerResponse"/>
 </wsdl:operation>
 <wsdl:operation name="unregister" parameterOrder="id pass service_id">
  <wsdl:input message="impl:unregisterRequest" name="unregisterRequest"/>
  <wsdl:output message="impl:unregisterResponse" name="unregisterResponse"/>
 </wsdl:operation>
 <wsdl:operation name="submitServiceReqs"
  parameterOrder="service_id requirements current_version phone_models">
  <wsdl:input message="impl:submitServiceReqsRequest" name="submitServiceReqsRequest"/>
  <wsdl:output message="impl:submitServiceReqsResponse" name="submitServiceReqsResponse"/>
 </wsdl:operation>
 <wsdl:operation name="submitServiceDetails"
  parameterOrder="service_id details current_version client_id">
  <wsdl:input message="impl:submitServiceDetailsRequest" name="submitServiceDetailsRequest"/>
  <wsdl:output message="impl:submitServiceDetailsResponse" name="submitServiceDetailsResponse"/>
 </wsdl:operation>
 <wsdl:operation name="submitServiceRightsOptions"
  parameterOrder="service_id rights_options client_id">
  <wsdl:input message="impl:submitServiceRightsOptionsRequest"
   name="submitServiceRightsOptionsRequest"/>
  <wsdl:output message="impl:submitServiceRightsOptionsResponse"
   name="submitServiceRightsOptionsResponse"/>
 </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="EDSRegistration" type="impl:EDSRegistration">
 <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="queryForgottenAuth">
   <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="queryForgottenAuthRequest">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="queryForgottenAuthResponse">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
     </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="submitForgottenAuth">
  <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="submitForgottenAuthRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="urn:server.parma.tcd.ie" use="encoded"/>
   </wsdl:input>
   <wsdl:output name="submitForgottenAuthResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="urn:server.parma.tcd.ie" use="encoded"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="changePassword">
   <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="changePasswordRequest">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="changePasswordResponse">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="queryRightsList">
    <wsdlsoap:operation soapAction=""/>
     <wsdl:input name="queryRightsListRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       namespace="urn:server.parma.tcd.ie" use="encoded"/>
     </wsdl:input>
     <wsdl:output name="queryRightsListResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       namespace="urn:server.parma.tcd.ie" use="encoded"/>
     </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="submitContactInfo">
     <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="submitContactInfoRequest">
       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:server.parma.tcd.ie" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="submitContactInfoResponse">
       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="register">
   <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="registerRequest">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="registerResponse">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="unregister">
   <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="unregisterRequest">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="unregisterResponse">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="submitServiceReqs">
   <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="submitServiceReqsRequest">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="submitServiceReqsResponse">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="submitServiceDetails">
   <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="submitServiceDetailsRequest">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="submitServiceDetailsResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="urn:server.parma.tcd.ie" use="encoded"/>
    </wsdl:output>
   </wsdl:operation>
```

```
     <wsdl:operation name="submitServiceRightsOptions">
      <wsdlsoap:operation soapAction=""/>
       <wsdl:input name="submitServiceRightsOptionsRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="urn:server.parma.tcd.ie" use="encoded"/>
       </wsdl:input>
       <wsdl:output name="submitServiceRightsOptionsResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="urn:server.parma.tcd.ie" use="encoded"/>
       </wsdl:output>
     </wsdl:operation>
    <wsdl:operation name="login">
     <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="loginRequest">
       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:server.parma.tcd.ie" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="loginResponse">
       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:server.parma.tcd.ie" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="EDSRegistrationService">
 <wsdl:port binding="impl:EDSRegistration" name="EDSRegistration">
  <wsdlsoap:address location="http://cartman2.dsg.cs.tcd.ie:8080/axis/services/EDSRegistration"/>  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

B.2 Rights Management Interface

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:apachesoap="http://xml.apache.org/xml-soap"
 xmlns:impl="rightsmgmt.vds_eds.parma.tcd.ie"
 xmlns:intf="rightsmgmt.vds_eds.parma.tcd.ie"
 xmlns:com="rightsmgmt.vds_eds.parma.tcd.ie"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
 xmlns:vds="urn:common.parma.tcd.ie"
 xmlns:ns="http://odrl.net/1.1/ODRL-DD"
 xmlns:ns1="http://www.w3.org/2001/04/xmlenc#"
 xmlns:ns2="http://www.w3.org/2000/09/xmldsig#"
 xmlns:ns3="urn:schema.parma.tcd.ie"
 targetNamespace="rightsmgmt.vds_eds.parma.tcd.ie">
 <wsdl:types>
  <xsd:schema targetNamespace="rightsmgmt.vds_eds.parma.tcd.ie"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:com="rightsmgmt.vds_eds.parma.tcd.ie"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:vds="urn:common.parma.tcd.ie"
   xmlns:o-ex="http://odrl.net/1.1/ODRL-EX/">
  <xsd:import namespace="urn:common.parma.tcd.ie" schemaLocation="VDS_EDS_DataTypes.xsd"/>
 </xsd:schema>
</wsdl:types>
<wsdl:message name="requestAppDetailsResponse">
 <wsdl:part name="requestAppDetailsReturn" type="vds:ServiceDetails"/>
</wsdl:message>
<wsdl:message name="requestAppDetailsRequest">
 <wsdl:part name="service_id" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="requestAppReqsResponse">
 <wsdl:part name="requestAppReqsReturn" type="vds:ServiceReqs"/>
</wsdl:message>
<wsdl:message name="requestAppReqsRequest">
 <wsdl:part name="service_id" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="sendAuditDataResponse">
 <wsdl:part name="confirmation" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="sendAuditDataRequest">
 <wsdl:part name="id" type="xsd:long"/>
 <wsdl:part name="audit_data" type="vds:AuditData"/>
 <wsdl:part name="service_id" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="getAuditDataResponse">
 <wsdl:part name="audit_data" type="vds:AuditData"/>
</wsdl:message>
<wsdl:message name="getAuditDataRequest">
 <wsdl:part name="service_id" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="getIssuedRightsResponse">
 <wsdl:part name="audit_data" type="vds:IssuedRightsList"/>
</wsdl:message>
<wsdl:message name="getIssuedRightsRequest">
 <wsdl:part name="service_id" type="xsd:long"/>
```

```
</wsdl:message>
<wsdl:message name="requestAppListResponse">
 <wsdl:part name="requestAppListReturn" type="vds:AppList"/>
</wsdl:message>
<wsdl:message name="requestAppListRequest">
 <wsdl:part name="service_details" type="vds:ServiceDetails"/>
</wsdl:message>
<wsdl:message name="requestRightsModelsResponse">
 <wsdl:part name="requestRightsModelsReturn" type="vds:RightsModels"/>
</wsdl:message>
<wsdl:message name="requestRightsModelsRequest">
 <wsdl:part name="service_id" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="requestAppList4DeviceResponse">
 <wsdl:part name="requestAppList4DeviceReturn" type="vds:AppList"/>
</wsdl:message>
<wsdl:message name="requestAppList4DeviceRequest">
 <wsdl:part name="device_list" type="vds:DeviceList"/>
</wsdl:message>
<wsdl:portType name="EDSRightsManagement">
 <wsdl:operation name="requestAppList" parameterOrder="service_details">
  <wsdl:input name="requestAppListRequest" message="com:requestAppListRequest"/>
  <wsdl:output name="requestAppListResponse" message="com:requestAppListResponse"/>
 </wsdl:operation>
 <wsdl:operation name="sendAuditData" parameterOrder="id audit_data service_id">
  <wsdl:input name="sendAuditDataRequest" message="com:sendAuditDataRequest"/>
  <wsdl:output name="sendAuditDataResponse" message="com:sendAuditDataResponse"/>
 </wsdl:operation>
 <wsdl:operation name="getIssuedRights" parameterOrder="service_id">
  <wsdl:input name="getIssuedRightsRequest" message="com:getIssuedRightsRequest"/>
  <wsdl:output name="getIssuedRightsResponse" message="com:getIssuedRightsResponse"/>
 </wsdl:operation>
 <wsdl:operation name="getAuditData" parameterOrder="service_id">
  <wsdl:input name="getAuditDataRequest" message="com:getAuditDataRequest"/>
  <wsdl:output name="getAuditDataResponse" message="com:getAuditDataResponse"/>
 </wsdl:operation>
 <wsdl:operation name="requestAppDetails" parameterOrder="service_id">
  <wsdl:input name="requestAppDetailsRequest" message="com:requestAppDetailsRequest"/>
  <wsdl:output name="requestAppDetailsResponse" message="com:requestAppDetailsResponse"/>
 </wsdl:operation>
 <wsdl:operation name="requestAppReqs" parameterOrder="service_id">
  <wsdl:input name="requestAppReqsRequest" message="com:requestAppReqsRequest"/>
  <wsdl:output name="requestAppReqsResponse" message="com:requestAppReqsResponse"/>
 </wsdl:operation>
 <wsdl:operation name="requestAppList4Device" parameterOrder="device_list">
  <wsdl:input name="requestAppList4DeviceRequest" message="com:requestAppList4DeviceRequest"/>
  <wsdl:output name="requestAppList4DeviceResponse" message="com:requestAppList4DeviceResponse"/>
 </wsdl:operation>
 <wsdl:operation name="requestRightsModels" parameterOrder="service_id">
  <wsdl:input name="requestRightsModelsRequest" message="com:requestRightsModelsRequest"/>
  <wsdl:output name="requestRightsModelsResponse" message="com:requestRightsModelsResponse"/>
 </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="EDSRightsManagement" type="impl:EDSRightsManagement">
 <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="requestAppList">
  <wsdlsoap:operation/>
  <wsdl:input>
   <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
  </wsdl:input>
  <wsdl:output>
   <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="sendAuditData">
 <wsdlsoap:operation/>
  <wsdl:input>
   <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
  </wsdl:input>
  <wsdl:output>
  <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="getIssuedRights">
  <wsdlsoap:operation/>
   <wsdl:input>
    <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
   </wsdl:input>
   <wsdl:output>
    <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
   </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="getAuditData">
  <wsdlsoap:operation/>
   <wsdl:input>
    <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
   </wsdl:input>
 <wsdl:output>
 <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
 </wsdl:output>
</wsdl:operation>
<wsdl:operation name="requestAppReqs">
 <wsdlsoap:operation/>
  <wsdl:input>
   <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
  </wsdl:input>
  <wsdl:output>
  <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="requestAppDetails">
 <wsdlsoap:operation/>
  <wsdl:input>
   <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
  </wsdl:input>
  <wsdl:output>
   <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="requestAppList4Device">
 <wsdlsoap:operation/>
  <wsdl:input>
   <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
  </wsdl:input>
  <wsdl:output>
   <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="requestRightsModels">
 <wsdlsoap:operation/>
  <wsdl:input>
   <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
  </wsdl:input>
  <wsdl:output>
  <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   namespace="rightsmgmt.vds_eds.parma.tcd.ie"/>
   </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="EDSRightsManagementService">
 <wsdl:port binding="impl:EDSRightsManagement" name="EDSRightsManagement">
  <wsdlsoap:address
   location="http://cartman2.dsg.cs.tcd.ie:8080/axis/services/EDSRightsManagement"/>
 </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

B.3 Common Data Types Schema (VDS_EDS_DataTypes.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:common.parma.tcd.ie"
elementFormDefault="qualified"
attributeFormDefault="qualified" version="1.0"
xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
xmlns:vds="urn:common.parma.tcd.ie"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://odrl.net/1.1/ODRL-DD"
xmlns:ns2="urn:schema.parma.tcd.ie">
<xsd:import namespace="http://odrl.net/1.1/ODRL-EX" schemaLocation="http://odrl.net/1.1/ODRL-EX-11.xsd"/>
<xsd:import namespace="http://odrl.net/1.1/ODRL-DD" schemaLocation="http://odrl.net/1.1/ODRL-DD-11.xsd"/>
<xsd:import namespace="urn:schema.parma.tcd.ie" schemaLocation="parma-10.xsd"/>
<xsd:annotation>
 <xsd:documentation xml:lang="en">
 parma Digital Rights Management Schema for Trinity College.
 Copyright 2003 Trinity College Dublin,
 Computer Science Department, Distributed Systems Group. All rights reserved.
 </xsd:documentation>
</xsd:annotation>
<xsd:complexType name="AppList">
 <xsd:sequence>
  <xsd:element name="application" type="vds:ServiceId" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AuditData">
 <xsd:sequence>
  <xsd:element name="singleLog" type="vds:LogInfo" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="IssuedRights">
 <xsd:sequence>
```

```
<xsd:element name="user_id" type="xsd:long"/>
<xsd:element name="app_id" type="xsd:long"/>
<xsd:element name="rights_model_id" type="xsd:long"/>
<xsd:element name="datetime_issued" type="xsd:string"/>
<xsd:element name="datetime_expires" type="xsd:string"/>
<xsd:element name="quantity" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LoginReturn">
 <xsd:sequence>
  <xsd:element name="sucessfull" type="xsd:boolean"/>
  <xsd:element name="id" type="xsd:long"/>
  <xsd:element name="uname" type="xsd:string"/>
  <xsd:element name="password" type="xsd:string"/>
  <xsd:element name="client_type" type="vds:ClientType"/>
  <xsd:element name="provider_type" type="vds:ProviderType"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="IssuedRightsList">
 <xsd:sequence>
  <xsd:element name="issued_rights" type="vds:IssuedRights" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="EDSRightsModel">
 <xsd:sequence>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="quantity" type="xsd:long"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="EDSRightsModelList">
 <xsd:sequence>
  <xsd:element name="models" type="vds:EDSRightsModel" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LogInfo">
 <xsd:sequence>
  <xsd:element name="starttime" type="xsd:string"/>
  <xsd:element name="endTime" type="xsd:string"/>
  <xsd:element name="event" type="xsd:string"/>
  <xsd:element name="sequence_no" type="xsd:long"/>
  <xsd:element name="signature" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DeviceList">
 <xsd:sequence>
  <xsd:element name="device_make_model" type="xsd:string" maxOccurs="unbounded"/>
 </xsd:sequence>
```

```
</xsd:complexType>
 <xsd:complexType name="RightsModels">
  <xsd:sequence>
 <xsd:element name="rights_type" type="vds:RightsType" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="RightsType">
 <xsd:sequence>
  <xsd:element name="rights_model_name" type="xsd:string"/>
  <xsd:element name="billing_unit" type="vds:BillingUnit" maxOccurs="unbounded"/>
  <xsd:element name="description" type="xsd:string" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="RightsName">
 <xsd:sequence>
  <xsd:element name="rights_model_name" type="xsd:string"/>
  <xsd:element name="rights_model_ID" type="xsd:long"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="BillingUnit">
 <xsd:attribute name="unit_name" type="xsd:string" use="required"/>
 <xsd:attribute name="price_per_unit" type="xsd:double" use="required"/>
 <xsd:attribute name="currency" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:element name="context" type="vds:contextType"/>
<xsd:complexType name="contextType">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:element ref="vds:context" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:choice>
</xsd:complexType>
<xsd:attributeGroup name="IDGroup">
 <xsd:attribute name="id" type="xsd:ID"/>
 <xsd:attribute name="idref" type="xsd:IDREF"/>
</xsd:attributeGroup>
<xsd:complexType name="ServiceReqs">
 <xsd:sequence>
  <xsd:element name="requirement" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="minValue" type="xsd:string" use="required"/>
    <xsd:attribute name="maxValue" type="xsd:string"/>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ServiceDetails">
 <xsd:sequence>
```

```
  <xsd:element name="version" type="xsd:string"/>
  <xsd:element name="vendor" type="xsd:string"/>
  <xsd:element name="contentType" type="xsd:string"/>
  <xsd:element name="description" type="xsd:string"/>
  <xsd:element name="category" type="xsd:string"/>
  <xsd:element name="size" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="RightsOptions">
 <xsd:sequence>
  <xsd:element name="license" type="vds:RightsType" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="RightsList">
 <xsd:sequence>
  <xsd:element name="rightsNames" type="vds:RightsName" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AuthenticationInfo" abstract="true">
 <xsd:sequence>
  <xsd:element name="uname" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="IndividualAuth">
 <xsd:complexContent>
  <xsd:extension base="vds:AuthenticationInfo">
   <xsd:sequence>
    <xsd:element name="pin" type="xsd:string"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="FullIndividualAuth">
 <xsd:complexContent>
  <xsd:extension base="vds:IndividualAuth">
   <xsd:sequence>
    <xsd:element name="secret_q" type="xsd:string"/>
    <xsd:element name="secret_a" type="xsd:string"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="EDSAuth">
 <xsd:complexContent>
  <xsd:extension base="vds:AuthenticationInfo">
   <xsd:sequence>
    <xsd:element name="password" type="xsd:string"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="FullEDSAuth">
 <xsd:complexContent>
  <xsd:extension base="vds:EDSAuth">
   <xsd:sequence>
    <xsd:element name="contact_person_fullname" type="xsd:string"/>
    <xsd:element name="contact_person_phone_no" type="xsd:string"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ContactInformation">
 <xsd:sequence>
  <xsd:element name="email" type="xsd:string"/>
  <xsd:element name="address" type="xsd:string"/>
  <xsd:element name="phone" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ClientId" abstract="true">
 <xsd:sequence>
  <xsd:element name="id" type="xsd:long"/>
  <xsd:element name="client_type" type="vds:ClientType" minOccurs="0"/>
  <xsd:element name="payment_type" type="vds:PayType" minOccurs="0"/>
  <xsd:element name="authentication" type="vds:AuthenticationInfo" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="PIN">
 <xsd:restriction base="xsd:int">
  <xsd:pattern value="[0-9][0-9][0-9][0-9]"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="ClientType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="individual"/>
  <xsd:enumeration value="EDS"/>
  <xsd:enumeration value="service"/>
 <xsd:enumeration value="service_provider"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="PayType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="ReverseSMS"/>
  <xsd:enumeration value="OperatorBilling"/>
  <xsd:enumeration value="CreditCard"/>
  <xsd:enumeration value="Invoice"/>
 <xsd:enumeration value="DirectDebit"/>
</xsd:restriction>
</xsd:simpleType>
 <xsd:simpleType name="ProviderType">
  <xsd:restriction base="xsd:string">
  <xsd:enumeration value="individual"/>
  <xsd:enumeration value="ISV"/>
 </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="IndividualId">
 <xsd:complexContent>
  <xsd:extension base="vds:ClientId">
   <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="mobile_number" type="xsd:string"/>
    <xsd:element name="IMEI" type="xsd:long"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ServiceId">
 <xsd:complexContent>
  <xsd:extension base="vds:ClientId">
   <xsd:sequence>
    <xsd:element name="service_URI" type="xsd:string"/>
    <xsd:element name="service_provider_ID" type="xsd:long"/>
    <xsd:element name="name" type="xsd:string"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ServiceProviderId">
 <xsd:complexContent>
  <xsd:extension base="vds:ClientId">
   <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="ProviderType">
     <xsd:simpleType>
      <xsd:restriction base="xsd:string">
       <xsd:enumeration value="individual"/>
       <xsd:enumeration value="ISV"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
   </xsd:sequence>
  </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="EDSId">
 <xsd:complexContent>
  <xsd:extension base="vds:ClientId">
   <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="countryCode" type="xsd:string"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
</xsd:schema>
```

Appendix C

Aspect Oriented Programming

C.1 Terminology

- Join point - identifiable point in the execution of a program where crosscutting concerns are woven in, such as method call, field assignment etc.
- Pointcut - join point (or several join points) together with context at that point, such as object method was called on, method parameters etc.
- Advice - source code that is executed at a join point described by a pointcut; can be executed before, after or instead of the join point.
- AspectJ - Java implementation of aspect-oriented programming.
- Aspect - part of the program that cross-cuts a main program concern. Also, a base unit of AspectJ, similar to a class being a base unit of Java programming language.
- Weaving - combining Aspects and remaining source code into a complete application.

C.2 Overview

Aspect oriented programming (AOP) is a technique introduced for design and coding of cross-cutting concerns within a software application. Object-oriented programming (OOP) cannot modularize all of the concerns in complex applications and as such is prone to code tangling and code scattering. Code tangling occurs when multiple concerns are dealt with in a single application module/class, making the code difficult to understand and maintain. Code scattering occurs when a single concern is dealt with in several modules/classes, also making the code more difficult to maintain and increasing the risk of inconsistency. AOP deals with these cross-cutting concerns in a way that prevents code tangling and code scattering [Lad03, Kis02].

In AOP, a single concern is addressed in a separate aspect that provides its implementation in a form of an advice. The aspect also specifies the pointcuts at which an advice is to be executed. Integration with the remaining application source (or bytecode in some Java implementations) is done at the compile time, in a process of weaving. In that way, separate concerns are dealt with in separate aspects and are not scattered around the application [Kis02, Lad03] .

Appendix D

ODRL

D.1 ODRL Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://odrl.net/1.1/ODRL-EX"
xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
elementFormDefault="qualified" attributeFormDefault="qualified" version="1.1">
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd"/>
<xsd:import namespace="http://www.w3.org/2001/04/xmlenc#"
schemaLocation="http://www.w3.org/TR/xmlenc-core/xenc-schema.xsd"/>
<xsd:element name="rights" type="o-ex:rightsType"/>
<xsd:element name="offer" type="o-ex:offerAgreeType"/>
<xsd:element name="agreement" type="o-ex:offerAgreeType"/>
 <xsd:complexType name="offerAgreeType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="o-ex:context" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:party" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:asset" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:permission" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:constraint" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:requirement" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:condition" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
 </xsd:complexType>
 <xsd:complexType name="rightsType">
  <xsd:complexContent>
   <xsd:extension base="o-ex:offerAgreeType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
     <xsd:element ref="o-ex:revoke" minOccurs="0" maxOccurs="unbounded"/>
     <xsd:element ref="o-ex:offer" minOccurs="0" maxOccurs="unbounded"/>
     <xsd:element ref="o-ex:agreement" minOccurs="0" maxOccurs="unbounded"/>
     <xsd:element ref="ds:Signature" minOccurs="0"/>
    </xsd:choice>
    <xsd:attributeGroup ref="o-ex:IDGroup"/>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
 <xsd:element name="context" type="o-ex:contextType"/>
 <xsd:element name="contextElement" abstract="true"/>
 <xsd:complexType name="contextType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="o-ex:context" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:contextElement" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
  <xsd:attributeGroup ref="o-ex:IDGroup"/>
 </xsd:complexType>
 <xsd:complexType name="partyType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="o-ex:context" minOccurs="0"/>
   <xsd:element ref="o-ex:rightsholder" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:party" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:container" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:asset" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
  <xsd:attributeGroup ref="o-ex:IDGroup"/>
 </xsd:complexType>
 <xsd:element name="party" type="o-ex:partyType"/>
 <xsd:element name="rightsholder" type="o-ex:rightsHolderType"/>
 <xsd:element name="rightsHolderElement" abstract="true"/>
 <xsd:complexType name="rightsHolderType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="o-ex:context" minOccurs="0"/>
   <xsd:element ref="o-ex:rightsHolderElement" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:container" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
  <xsd:attributeGroup ref="o-ex:IDGroup"/>
 </xsd:complexType>
 <xsd:complexType name="assetType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="o-ex:context"/>
   <xsd:element ref="o-ex:inherit"/>
   <xsd:element name="digest">
 <xsd:complexType>
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="ds:DigestMethod"/>
   <xsd:element ref="ds:DigestValue"/>
  </xsd:choice>
  </xsd:complexType>
 </xsd:element>
 <xsd:element ref="ds:KeyInfo"/>
 </xsd:choice>
 <xsd:attributeGroup ref="o-ex:IDGroup"/>
  <xsd:attribute name="type">
   <xsd:simpleType>
    <xsd:restriction base="xsd:NMTOKEN">
     <xsd:enumeration value="work"/>
     <xsd:enumeration value="expression"/>
     <xsd:enumeration value="manifestation"/>
     <xsd:enumeration value="item"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
 </xsd:complexType>
 <xsd:element name="asset" type="o-ex:assetType"/>
 <xsd:complexType name="inheritType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="o-ex:context" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
  <xsd:attribute name="override" type="xsd:boolean" default="false"/>
  <xsd:attribute name="default" type="xsd:boolean" default="false"/>
 </xsd:complexType>
 <xsd:element name="inherit" type="o-ex:inheritType"/>
 <xsd:element name="permission" type="o-ex:permissionType"/>
 <xsd:element name="permissionElement" abstract="true"/>
 <xsd:complexType name="permissionType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="o-ex:context" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:permissionElement" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:container" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:constraint" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:sequence" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:requirement" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:condition" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:asset" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
  <xsd:attribute name="exclusive" type="xsd:boolean" use="optional"/>
  <xsd:attributeGroup ref="o-ex:IDGroup"/>
 </xsd:complexType>
 <xsd:element name="constraint" type="o-ex:constraintType"/>
 <xsd:element name="constraintElement" abstract="true"/>
 <xsd:complexType name="constraintType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="o-ex:constraint" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:constraintElement" minOccurs="0" maxOccurs="unbounded"/>
```

```
   <xsd:element ref="o-ex:container" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:sequence" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:context" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
  <xsd:attributeGroup ref="o-ex:IDGroup"/>
  <xsd:attribute ref="o-ex:type"/>
 </xsd:complexType>
 <xsd:attribute name="type" type="xsd:anyURI"/>
 <xsd:element name="requirement" type="o-ex:requirementType"/>
 <xsd:element name="requirementElement" abstract="true"/>
 <xsd:complexType name="requirementType">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="o-ex:context" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:requirementElement" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:container" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:sequence" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="o-ex:IDGroup"/>
 </xsd:complexType>
 <xsd:element name="condition" type="o-ex:conditionType"/>
 <xsd:element name="conditionElement" abstract="true"/>
 <xsd:complexType name="conditionType">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="o-ex:context" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:conditionElement" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:permission" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:constraint" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:container" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o-ex:sequence" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="o-ex:IDGroup"/>
 </xsd:complexType>
 <xsd:complexType name="revokeType">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
   <xsd:element ref="o-ex:context" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="o-ex:IDGroup"/>
 </xsd:complexType>
 <xsd:element name="revoke" type="o-ex:revokeType"/>
  <xsd:complexType name="sequenceType">
   <xsd:sequence>
    <xsd:element ref="o-ex:seq-item" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="order" default="total">
   <xsd:simpleType>
    <xsd:restriction base="xsd:NMTOKEN">
     <xsd:enumeration value="total"/>
    <xsd:enumeration value="partial"/>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:attribute>
</xsd:complexType>
<xsd:element name="sequence" type="o-ex:sequenceType"/>
<xsd:complexType name="containerType">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:element ref="o-ex:container" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:permission" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:permissionElement" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:constraintElement" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:conditionElement" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:requirementElement" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:rightsHolderElement" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:constraint" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:condition" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:sequence" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:requirement" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:party" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:choice>
<xsd:attribute name="type" default="and">
 <xsd:simpleType>
  <xsd:restriction base="xsd:NMTOKEN">
   <xsd:enumeration value="and"/>
   <xsd:enumeration value="in-or"/>
   <xsd:enumeration value="ex-or"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:attribute>
<xsd:attributeGroup ref="o-ex:IDGroup"/>
</xsd:complexType>
<xsd:element name="container" type="o-ex:containerType"/>
<xsd:complexType name="seqItemType">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:element ref="o-ex:container" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:permission" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:permissionElement" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:constraintElement" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:conditionElement" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:requirementElement" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:rightsHolderElement" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:constraint" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:condition" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:sequence" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="o-ex:requirement" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:choice>
 <xsd:attribute name="number" type="xsd:integer" use="required"/>
</xsd:complexType>
<xsd:element name="seq-item" type="o-ex:seqItemType"/>
<xsd:attributeGroup name="IDGroup">
<xsd:attribute name="id" type="xsd:ID"/>
<xsd:attribute name="idref" type="xsd:IDREF"/>
<xsd:anyAttribute namespace="##other"/>
</xsd:attributeGroup>
</xsd:schema>
```

D.2 ODRL Data Dictionary

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://odrl.net/1.1/ODRL-DD"
xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified" version="1.1">
<xsd:import namespace="http://odrl.net/1.1/ODRL-EX"
schemaLocation="http://odrl.net/1.1/ODRL-EX-11.xsd"/>
<!-- Declare all the Permission Elements -->
<xsd:element name="display" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="print" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="play" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="execute" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="sell" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="lend" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="give" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="lease" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="modify" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="excerpt" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="aggregate" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="annotate" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="move" type="o-ex:permissionType"
```

```
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="duplicate" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="delete" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="verify" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="backup" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="restore" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="install" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="uninstall" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<xsd:element name="save" type="o-ex:permissionType"
substitutionGroup="o-ex:permissionElement"/>
<!-- Declare the Payment Element (used in Requirements) -->
<xsd:element name="payment">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="amount">
    <xsd:complexType>
     <xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
       <xsd:attribute name="currency" type="xsd:NMTOKEN" use="required"/>
      </xsd:extension>
     </xsd:simpleContent>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="taxpercent" minOccurs="0">
    <xsd:complexType>
     <xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
       <xsd:attribute name="code" type="xsd:NMTOKEN" use="required"/>
      </xsd:extension>
     </xsd:simpleContent>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<!-- Define the dataTypes used for Requirements that use Payment element -->
<xsd:complexType name="feeType">
 <xsd:complexContent>
  <xsd:extension base="o-ex:requirementType">
   <xsd:sequence>
```

```
    <xsd:element ref="o-dd:payment"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<!-- Declare all the Requirements Elements -->
<xsd:element name="prepay" type="o-dd:feeType"
substitutionGroup="o-ex:requirementElement"/>
<xsd:element name="postpay" type="o-dd:feeType"
substitutionGroup="o-ex:requirementElement"/>
<xsd:element name="peruse" type="o-dd:feeType"
substitutionGroup="o-ex:requirementElement"/>
<xsd:element name="accept" type="o-ex:requirementType"
substitutionGroup="o-ex:requirementElement"/>
<xsd:element name="register" type="o-ex:requirementType"
substitutionGroup="o-ex:requirementElement"/>
<xsd:element name="attribution" type="o-ex:requirementType"
substitutionGroup="o-ex:requirementElement"/>
<xsd:element name="tracked" type="o-ex:requirementType"
substitutionGroup="o-ex:requirementElement"/>
<!-- Declare all the RightsHolder Elements -->
<xsd:element name="fixedamount" substitutionGroup="o-ex:rightsHolderElement">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="o-ex:rightsHolderType">
    <xsd:sequence>
     <xsd:element ref="o-dd:payment"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
<xsd:element name="percentage" substitutionGroup="o-ex:rightsHolderElement">
 <xsd:simpleType>
  <xsd:restriction base="xsd:decimal">
   <xsd:minInclusive value="0"/>
   <xsd:maxInclusive value="100"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>
<!-- Declare all the Context Elements -->
<xsd:simpleType name="uriAndOrString">
 <xsd:union memberTypes="xsd:anyURI xsd:string"/>
</xsd:simpleType>
<xsd:element name="uid" type="o-dd:uriAndOrString" substitutionGroup="o-ex:contextElement"/>
<xsd:element name="role" type="xsd:anyURI" substitutionGroup="o-ex:contextElement"/>
<xsd:element name="name" type="xsd:string" substitutionGroup="o-ex:contextElement"/>
```

```
<xsd:element name="remark" type="xsd:string" substitutionGroup="o-ex:contextElement"/>
<xsd:element name="event" type="xsd:string" substitutionGroup="o-ex:contextElement"/>
<xsd:element name="pLocation" type="xsd:string"
substitutionGroup="o-ex:contextElement"/>
<xsd:element name="dLocation" type="xsd:anyURI"
substitutionGroup="o-ex:contextElement"/>
<xsd:element name="reference" type="xsd:anyURI"
substitutionGroup="o-ex:contextElement"/>
<xsd:element name="version" type="xsd:string"
substitutionGroup="o-ex:contextElement"/>
<xsd:element name="transaction" type="xsd:string"
substitutionGroup="o-ex:contextElement"/>
<xsd:element name="service" type="xsd:anyURI"
substitutionGroup="o-ex:contextElement"/>
<xsd:element name="date" type="o-dd:dateType"
substitutionGroup="o-ex:contextElement"/>
<!-- Declare all the Constraint Elements -->
<xsd:element name="individual" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="group" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="cpu" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="network" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="screen" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="storage" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="memory" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="printer" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="software" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="hardware" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="spatial" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="quality" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="format" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="unit" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="watermark" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
```

```
<xsd:element name="purpose" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="industry" type="o-ex:constraintType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="count" type="xsd:positiveInteger"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="range" substitutionGroup="o-ex:constraintElement">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="o-ex:constraintType">
    <xsd:sequence>
     <xsd:element name="min" type="xsd:decimal" minOccurs="0"/>
     <xsd:element name="max" type="xsd:decimal" minOccurs="0"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
<xsd:element name="datetime" type="o-dd:dateType"
substitutionGroup="o-ex:constraintElement"/>
<xsd:simpleType name="dateAndOrTime">
 <xsd:union memberTypes="xsd:date xsd:dateTime"/>
</xsd:simpleType>
<xsd:complexType name="dateType">
 <xsd:complexContent>
  <xsd:extension base="o-ex:constraintType">
   <xsd:choice>
    <xsd:sequence>
     <xsd:element name="start" type="o-dd:dateAndOrTime" minOccurs="0"/>
     <xsd:element name="end" type="o-dd:dateAndOrTime" minOccurs="0"/>
    </xsd:sequence>
    <xsd:element name="fixed" type="o-dd:dateAndOrTime" minOccurs="0"/>
   </xsd:choice>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
<xsd:element name="accumulated" type="xsd:duration"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="interval" type="xsd:duration"
substitutionGroup="o-ex:constraintElement"/>
<xsd:element name="recontext" type="xsd:boolean"
substitutionGroup="o-ex:constraintElement"/>
<!-- Transfer Permission is defined as a ContainerType to enable complete
expression of rights in the Constraint -->
<xsd:element name="transferPerm" substitutionGroup="o-ex:container">
 <xsd:complexType>
  <xsd:complexContent>
```

```
   <xsd:extension base="o-ex:containerType">
    <xsd:attribute name="downstream" default="equal">
     <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
       <xsd:enumeration value="equal"/>
       <xsd:enumeration value="less"/>
       <xsd:enumeration value="notgreater"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:attribute>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Bibliography

[Air04] AirClic. Mobile delivery track and trace services, 2004. URL: http://www.airclic.com/solutions/solutions_mobile_delivery.html.

[AJP04] The Apache Jakarta Project. Apache Jakarta Tomcat, 2004. URL: http://jakarta.apache.org/tomcat/.

[All01] Jonathan Allin. *Wireless Java for Symbian Devices*. ISBN: 0471486841. Wiley, 2001.

[Alt04] Altova. XMLSPY 2004, August 2004. URL: https://shop.altova.com/.

[Ama04] Amazon.co.uk PC Games, September 2004. URL: http://www.amazon.co.uk/exec/obidos/tg/browse/-/300729/ref%3Dbr%5Findx%5Fvg%5F1%5F1/026-8452427-4866800.

[And98] Don Andersen. Licensing The Future With FLEXlm. *The BASIS Advantage*, 2(4), 1998.

[ASF04] The Apache Software Foundation, April 2004. URL: http://xml.apache.org/security/.

[AWSP04] The Apache Web Services Project. Apache Axis, 2004. URL: http://ws.apache.org/axis/.

[BAH03] Booz, Allen, and Hamilton. Future Mobile Entertainment Scenarios. Mobile Entertainment Forum. Whitepaper, March 2003.

[Buh03] Willms Buhse. OMA Secure Content Delivery for the Mobile World. In *Proceedings of the 2003 ACM workshop on Digital rights management*. ACM Press, October 2003.

[Buh04] Willms Buhse. OMA Download and DRM Group. In Renatoo Iannella and Susanne Guth, editors, *Proceedings of the First International Workshop on the Open Digital Rights Language ODRL*, April 2004.

[Cla02] Niall Clarke. Distributed Software Licensing Framework based on Web Services and SOAP. Final Year Project Trinity College Dublin, 2002.

[Com04] Teydo Company. FleetOnline, 2004. URL: http://www.fleetonline.net/.

[Con04a] ContentGuard, 2004. URL: http://www.contentguard.com/.

[Con04b] ContentGuard. RightsExpress Tool, April 2004. URL: `http://rightsexpress.contentguard.com/`.

[CW04] Sandeep Chatterjee and James Webber. *Developing Enterprise Web Services.* ISBN: 0131401602. Penguin Books (NZ) Ltd, November 2004.

[DA01] InfraRed Data Association. IrDA Data Specifications, 2001. URL: `http://www.irda.org/displaycommon.cfm?an=1&subarticlenbr=7`.

[Dar03] Barbara Darrow. New Oracle Database Aims to Fill SMB Niche. CRN News, October 2003. URL: `http://www.crn.com/sections/breakingnews/breakingnews.jhtml?articleId=45101&_requestid=74896`.

[DDD04] Dominik Dahlem, Ivana Dusparic, and Jim Dowling. A Pervasive Application Rights Management Architecture (PARMA) based on ODRL. In Renato Iannella and Susanne Guth, editors, *Proceedings of the First International Workshop on ODRL*, April 2004.

[Dig04] Fairfax Digital. The virtual girlfriend is here, August 2004. URL: `http://www.theage.com.au/articles/2004/08/24/1093246503576.html`.

[DR04] Digi-Red. Symbian OS 7650/3650 Products FAQ, 2004. URL: `http://www.worldup.com/support/faq7650.htm`.

[DS04] DRMWatch Staff. ISO approves MPEG REL. *DRM Watch*, April 2004. URL: `www.drmwatch.com/standards/article.php/3334611`.

[Enf04] *PE Enforcer*, 2004. URL: `http://www.skaro.net/enforcer/frames.html?index.html`.

[For04] Forbes.com. Nokia says number of mobile phone users to hit 2 billion by 2007, June 2004. URL: `http://www.forbes.com/technology/feeds/wireless/2004/06/14/wireless01087222502107-20040614-004600.html`.

[For05] Forbes.com. Eu Reviews Microsoft, Contentguard Deal, February 2005. URL: `http://www.forbes.com/associatedpress/feeds/ap/2005/02/11/ap1820824.html`.

[Fou04a] Eclipse Foundation. AspectJ Project, 2004. URL: `http://eclipse.org/aspectj/`.

[Fou04b] Eclipse Foundation. Eclipse IDE, 2004. URL: `http://www.eclipse.org/`.

[gam03] gamesindustry.biz. Nokia N-Gage tops mobile game download chart. *The Register*, November 2003. URL: `http://www.theregister.co.uk/2003/11/27/nokia_ngage_tops_mobile_game/`.

[GKN$^+$96] Robert Gray, David Kotz, Saurab Nog, Daniela Rus, and George Cybenko. Mobile agents for mobile computing. Technical report, Dartmouth College, 1996.

[GNS03] Susanne Guth, Gustaf Neumann, and Mark Strembeck. Experiences with the Enforcement of Access Rights Extracted from ODRLbased Digital Contracts. In *Proceedings of the 2003 ACM workshop on Digital rights management*. ACM Press, October 2003. URL: `wi.wu-wien.ac.at/people/Guth/p21-guth.pdf`.

[gpr] General Packet Radio Service GPRS Service Description, GSM 02.60 version 6.1.1 Release 1997.

[GPS95] Global Positioning System. Standard Positioning Service Signal Specification, June 1995.

[Gre03] Robyn Greenspan. Online, Wireless Gaming Gaining. *atnewyork.com*, January 2003. URL: `http://atnewyork.internet.com/news/article.php/1572231`.

[Gri02] Robert Grimm. *System Support for Pervasive Applications*. PhD thesis, University of Washington, 2002.

[Gut03] Susanne Guth. *Lecture Notes in Computer Science*, chapter Rights Expression Languages. Springer-Verlag Heidelberg, 2003.

[Han03] Handango Yardstick Second Quarter 2003. Report, August 2003. URL: `http://www.handango.com/YardStick.jsp?Ckey=YardStick2_2003`.

[Han04] Handango, September 2004. URL: `http://www.handango.com/`.

[HP04] HP Handheld Devices, 2004. URL: `http://welcome.hp.com/country/us/en/prodserv/handheld.html`.

[HYP02] Seong Oun Hwang, Ki Song Yoon, and Chang Soon Park. Design and Implementation of a Licensing architecture for Distribution of Copyright-Protected Digital Contents. *Telecommunications Review*, 12(5), October 2002.

[Ian01] Renato Iannella. Digital Rights Management (DRM) Architectures. *D-Lib Magazine*, 7(6), June 2001.

[Ian02] Renato Iannella. ODRL Specification 1.1, August 2002. URL: `http://odrl.net/1.1/ODRL-11.pdf`.

[IEE99] IEEE. IEEE Std. 802.11a Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999. URL: `http://standards.ieee.org/getieee802/`.

[Inf04] Infowave. Symmetry Pro Enterprise General Overview, August 2004. URL: `http://www.infowave.com/products/spro_enterprise.html`.

[Int04] Intellisync. Intellisync Handheld Edition for Enterprise, August 2004. URL: `http://www.intellisync.com/go/products/client-synchronization/intellisync-handheld-edition-for-enterprise/index.cfm`.

[ITU04] ITUtilityPipeline. Software: The Subscription Pricing Model, April 2004. URL: `http://www.itutilitypipeline.com/showArticle.jhtml?articleId=18902677&printableArticle=true`.

[KFJ01] Lalana Kagal, Tim Finin, and Anupam Joshi. Trust-based security in pervasive computing environments. *Communications*, December 2001.

[KGS04] Amy Mizoras Konary, Stephen Graham, and Laurie A. Seymour. The Future of Software Licensing: Software Licensing Under Siege. Technical report, IDC, March 2004.

[Kis02] Ivan Kiselev. *Aspect-Oriented Programming with AspectJ*. ISBN:0672324105. SAMS, July 2002.

[Lad03] Ramnivas Laddad. *AspectJ in Action*. Manning, 2003.

[Let04] John Lettice. Guilty until proven innocent - DRM the mobile phone way. *The Register*, July 2004. URL: `http://www.theregister.co.uk/2004/07/15/oma_drm_for_phones/`.

[LotBC04] The Legion of the Bouncy Castle. Bouncy Castle Crypto API, 2004. URL: `http://www.bouncycastle.org/`.

[MA02] Open Mobile Alliance. Digital Rights Management Version 1.0, September 2002.

[MA04] Open Mobile Alliance. DRM Rights Expression Language Version 2.0, April 2004.

[Mac] Macrovision. FLEXlm Programmers Guide.

[Mac03a] Macrovision. The Case for a Universal Licensing Platform. Whitepaper, 2003. URL: `www.macrovision.com`.

[Mac03b] Macrovision. *FLEXlm User Guide*, 9.2 edition, July 2003. URL: `http://www.macrovision.com/services/support/enduser/TOC.htm`.

[Mac04a] Macrovision. Download and Information Page for lmgrd and Utilities, 2004. URL: `http://www.macrovision.com/services/support/flexlm/lmgrd.shtml`.

[Mac04b] Macrovision. FLEXlm Details, September 2004. URL: `http://www.globetrotter.com/flexlm/lmdet.shtml`.

[Mac04c] Macrovision. FLEXnet, 2004. URL: `http://www.macrovision.com/products/flexnet/index.shtml`.

[Mac04d] Macrovision. Flexnet Publisher Licensing Module, 2004. URL: `www.macrovision.com/products/flexnet_publisher/licensing_module.shtml`.

[Mac04e] Macrovision. FLEXnet Publisher Utility Pricing Module, 2004. URL: `http://www.macrovision.com/products/flexnet_publisher/utility_pricing_module.shtml`.

[Man01] Manifold. Manifold Software Activation Keys and Serial Numbers, 2001. URL: http://www.manifold.net/activation/activation_help.html.

[MB03] Brad A. Myers and Michael Beigl. Handheld Computing. *Computer - Innovative Technology for Computing Professionals*, September 2003.

[McM04] John McMullen. An Evaluation of AOP and Declarative Programming Applied to Mobile Software Licensing. Final Year Project Trinity College Dublin, 2004.

[MDP01] Mairead Martin and Doug Pearson. Rights Metadata: XrML and ODRL for Digital Video. presentation, August 2001.

[MDSG02] ISO Multimedia Description Schemes Group. Text of ISO/IEC CD 21000-Part 6 - Rights Data Dictionary (RDD), July 2002. URL: http://www.chiariglione.org/mpeg/working_documents.htm.

[Met04] Metrowerks. Metrowerks Licensing Models Overview, 2004. URL: http://www.metrowerks.com/MW/Develop/Licensing_Models.htm.

[MH03] Osbourne McGraw Hill. Oracle 9i Mobile - The Mobile Environment. Oracle Whitepaper, May 2003.

[Mic04a] Microsoft. Description of the Office Activation Wizard, August 2004. URL: http://support.microsoft.com/default.aspx?kbid=293151.

[Mic04b] Sun Microsystems. J2EE Java Servlet Technology, 2004. URL: http://java.sun.com/products/servlet/.

[Mic04c] Sun Microsystems. J2ME Wireless Toolkit, September 2004. URL: http://java.sun.com/products/j2mewtoolkit/.

[Mic04d] Sun Microsystems. Java 2 Platform, Standard Edition (J2SE), 2004. URL: http://java.sun.com/j2se/.

[Mic04e] Sun Microsystems. keytool - Key and Certificate Management Tool, 2004. URL: http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html.

[MPE04] MPEG Working Documents, 2004. http://www.chiariglione.org/mpeg/working_documents.htm.

[msn04] Microsoft .NET, 2004. URL: http://www.microsoft.com/net/.

[mys04] MySQL database server, 2004. URL: http://www.mysql.com/products/mysql/.

[nin04] ninemsn. Australia's First Real-time Multiplayer Mobile Game Launched, August 2004. URL: http://games.ninemsn.com.au/article.aspx?id=14624.

[Nok02] Nokia. New Global Organization, the Open Mobile Alliance, Formed to Foster Worldwide Growth in the Mobile Services Market, June 2002. URL: `http://press.nokia.com/PR/200206/863116_5.html`.

[Nok04a] Nokia Software Market by Digital River, Inc., 2004. URL: `http://www.softwaremarket.nokia.com/`.

[Nok04b] Nokia. Content Publishing Toolkit, 2004. URL: `http://www.forum.nokia.com/main/0,,034-57,00.html`.

[Nok04c] Nokia. Nokia DRM and Download FAQ, July 2004. URL: `http://www.forum.nokia.com/main/`.

[Nov04] Novell. Novell Product Activation, 2004. URL: `http://www.novell.com/products/activation/`.

[O204] O2. Checking Upgrade Eligibility, 2004. URL: `http://web.o2.ie/personal/services/speak_easy/upgrade.jsp`.

[ODR02] Odrl Data Dictionary, August 2002. URL: `http://www.odrl.net/1.1/ODRL-DD-11-DOC/index.html`.

[OG04] The Open Group. The XSLM Resource Center, 2004. URL: `www.xslm.org`.

[OMA04] Open Mobile Alliance, 2004. URL: `www.openmobilealliance.org/`.

[Ope04] The Open Group, July 2004. URL: `http://www.opengroup.org/`.

[Orl04] Andrew Orlowski. Why shrink wrap software won't die. *The Register*, April 2004. URL: `http://www.theregister.co.uk/2004/04/28/avanquest_shrink_wrap/`.

[Pac] Hewlett Packard. Compaq iPAQ Pocket PC H3800 Series Specifications. URL: `http://h18000.www1.hp.com/products/quickspecs/10977_na/10977_na.HTML`.

[PPD04] Josep Polo, Jose Prados, and Jaime Delgado. Interoperability between ODRL and MPEG-21 REL. In Renato Iannella and Susanne Guth, editors, *Proceedings of the First International Workshop on ODRL*, April 2004.

[Scr04] Kenn Scribner. Applications Licensing using the .NET Framework, 2004. URL: `http://www.developer.com/net/net/article.php/3074001`.

[SIG03] Bluetooth SIG. Specification of the Bluetooth System. Master Table of Contents and Compliance Requirements, November 2003. URL: `www.bluetooth.com`.

[Sof02] Softletter. On Dealing With Software Pirates, October 2002. URL: `http://www.softletter.com/samples/v18n23sample.html`.

[SPS03] Nuno Santos, Pedro Pereira, and Luis Silva. A Generic DRM Framework for J2ME Applications. In Olli Pitkanen, editor, *Proceedings of the First International Mobile IPR Workshop: Rights Management of Information Products on the Mobile Internet*. Helsinki Institute for Information Technology, HIIT Publications, August 2003.

[Swi00] Colin Edward Swindells. Use That There! Pointing to Establish Device Identity. Master's thesis, Simon Fraser University, 2000.

[Sym04] Symmetry Pro Enterprise, August 2004. URL: `http://www.symmetrypro.com/Compaq/buy.asp`.

[Sys04] Extended Systems. XTNDConnect Server GE Frequently Asked Questions, August 2004. URL: `http://resolution.extendedsystems.com/esi/service+support/xtndconnect+server/faqs/default.htm`.

[TCB04] Shiu Lun Tsang, Siobhan Clarke, and Elisa Baniassad. Object Metrics fo Aspect Systems: Limiting Empirical Inference Based on Modularity. Technical report, Trinity College Dublin, 2004.

[Tec03] Credant Technologies. CREDANT Technologies Adds Support for Laptops, Tablets and Smartphones Revolutionizing Security for the Mobile Enterprise, February 2003. URL: `http://www.credant.com/news/viewnews.php?nid=27`.

[Tel04] Telestial. Glossary, 2004. URL: `http://www.telestial.com/glossary.htm`.

[Tsa03] Shiu Lun Tsang. An Evaluation of AOP for Java-based Real-time Systems Development. Master's thesis, Trinity College Dublin, 2003.

[UCD04] Information and Educational Technology Glossary, 2004. URL: `http://iet.ucdavis.edu/glossary/`.

[Vod04] Vodafone Live, September 2004. URL: `http://www.vodafone.ie/live/`.

[W3C97] W3C. Date and Time Formats, September 1997. URL: `http://www.w3.org/TR/NOTE-datetime`.

[W3C99a] W3C. Hypertext Transfer Protocol – HTTP/1.1, 1999. URL: `http://www.w3.org/Protocols/rfc2616/rfc2616.html`.

[W3C99b] W3C. Xsl Transformations (XSLT) Version 1.0, November 1999. URL: `http://www.w3.org/TR/xslt`.

[W3C01] W3C. Web Services Description Language (WSDL) 1.1, March 2001. URL: `http://www.w3.org/TR/wsdl`.

[W3C03] W3C. Soap Version 1.2 Specification, June 2003. URL: `http://www.w3.org/TR/soap/`.

[Wal96] Jonathan Wallace. Are Shrink-Wrapped Licenses Enforceable? *UniForum Association*, December 1996. URL: `http://www.uniforum.org/news/html/publications/ufm/dec96/legal.html`.

[Wat01] DRM Watch. eXtensible Rights Markup Language (XrML), Version 2.0, November 2001. URL: `http://www.giantstepsmts.com/DRM%20Watch/xrml20.htm`.

[Wel02] Mary Welch. Value Pricing: A Viable Software Pricing Metric? Technical report, Saugatuck Technology, July 2002. URL: `www.saugatuck.com`.

[Wen04] Rigo Wenning. DRM and the Web. In Renato Iannella and Susanne Guth, editors, *Proceedings of the First International Workshop on ODRL*, April 2004.

[Wor04] InfoSync World. P900 Technical Specification, 2004. URL: `http://www.infosyncworld.com/hardware/whandhelds/specs/27.html`.

[WW02] Rigo Wenning W3C. Team Comment on ODRL Submission, September 2002. URL: `http://www.w3.org/Submission/2002/06/Comment`.

[Xer04] Xerox. Palo Alto Research Center, 2004. URL: `http://www.parc.xerox.com/`.

[xrm04] XrML, 2004. URL: `http://www.xrml.org/about.asp`.

[XSL99] Systems Management: Software License Use Management (XSLM). Technical standard, The Open Group, March 1999.