A Pervasive Application Rights Management Architecture (PARMA) based on ODRL

Dominik Dahlem, Ivana Dusparic and Jim Dowling

Distributed Systems Group
Department of Computer Science
Trinity College Dublin
Dominik.Dahlem@cs.tcd.ie, Ivana.Dusparic@cs.tcd.ie, Jim.Dowling@cs.tcd.ie

Abstract. Software license management is currently expanding from its traditional desktop environment into the mobile application space, but software vendors are still applying old licensing models to a platform where application rights will be specified, managed and distributed in new and different ways. This paper presents an open-source pervasive application rights management architecture (PARMA) for fixed network and mobile applications that supports the specification of application rights in a rights expression language (REL) based on ODRL. Our rights specification model uses aspect-oriented programming to generate modularized rights enforcement behaviour, which reduces development time for rights models such as feature-based usage rights and nagware. PARMA manages vendor and customer application rights over multiple platforms using a web services architecture and a container model on the client-side. The container model also supports the integration of services such as payment and encourages the super distribution of the rights object with associated default (evaluation) rights.

Keywords: Application Rights Management, Pervasive Computing, ODRL, Aspect-Oriented Programming.

1 Introduction

Software licensing has been a relatively successful example of rights management based on enforcement. The movement of license management from its traditional desktop environment into the mobile application space has so far not resulted in the appearance of new licensing architectures that utilise mobile-device features such as unique device identification, super distribution of applications over multiple communication channels (e.g. Bluetooth and GPRS) and the separate delivery of application usage rights [01]. Such a licensing platform must also deal with mobile-specific problems such as disconnected operation and resource constrained environments. The DRM community has been more successful at identifying some of these features and problems than the software licensing community, but due to its focus more on rights management of content than applications existing DRM specifications and architectures have left many of the aforementioned issues unresolved. Additionally, there is a requirement for applications vendors who distribute both desktop and mobile applications for an integrated (pervasive) licensing or application rights management architecture that will manage their applications over many platforms.

One of the major limitations of existing licensing systems is how licensing behaviour (application usage rights) is specified and integrated into applications. Existing distributed software licensing platforms [02] and standards [03] support the specification of licensing behaviour in applications by providing language- and platform-specific bindings to their licensing architectures. These solutions are typically aimed at vertical markets, require programmer knowledge of proprietary application programming interfaces (API) and do not provide the integration and flexibility required for more ubiquitous software deployment. This all leads to significant problems in using existing licensing technologies in the mobile application domain.

This paper investigates how the use of a declarative programming language (or a rights expression language (REL)) for the specification of a user's application usage rights (or license terms) can greatly improve the application development process. We introduce a REL that extends ODRL [04] and supports the specification of fine-grained application rights, such as specifying access rights at the

object and method level in object-oriented applications. The REL is integrated into applications using aspect-oriented (AO) technology [05]. AO technology supports for the separation of rights management concerns from application concerns. Calls to a licensing API that are scattered [05] around application source code can be encapsulated into single, manageable aspects producing better modularization, and hence improved maintenance, of rights management for applications. The REL also supports the use of services external to application. For example, application usage rights can be associated with payment services allowing users with an evaluation copy of an application to acquire better usage rights for it.

The rights enforcement architecture provides a container model with plug-in services that mediates application interaction with the services. The container is designed for resource constrained devices and supports a base set of services, including application rights object management, security and the separate delivery of rights objects to applications. This paper also describe how the application rights management architecture handles the aforementioned pervasive computing and mobility problems. In particular, we introduce a novel audited pay-per-use model designed specifically for mobile phone application software.

Section 2 of this paper introduces and motivates the open-source PARMA model and Section 3 describes our rights expression language. Section 4 details the mapping of our REL to aspects and the weaving of aspects into applications. Section 5 describes the rights enforcement architecture and we conclude with the status of our work and future work.

2 Background and Motivation

An application rights management architecture for pervasive computing environments must encompass a broad range of platforms and stakeholders. Application vendors need to be able to specify and organize rights, application distributors need to be able to sell rights to consumers and client devices must be able to run the applications while enforcing the rights and allowing the modification of those rights. Application rights can be defined using traditional software licenses or with special purpose *rights expression languages* (REL) such as ODRL or XrML [06]. However, for fine-grained specification of application rights, i.e. at the application feature level, there is a requirement for rights to be associated with application functionality, such as at the method invocation level for objects. Traditional DRM is concerned with specifying whether or not a given user has a right to view the passive content on the given device at the given time. Application DRM differs because users can be allowed to execute one part of the application but not the other. For example, a user can create documents, but not save them with a demo version of the application. Or users are allowed to execute all parts of the application but billed differently for using different features, as opposed to being billed per application execution/viewing as is the case with passive content DRM. Therefore, DRM concerned with active content has to be more extensive than DRM over passive content.

Also, users will often receive evaluation versions of software or versions of software that degrade over time [07] that they can later upgrade to a full version. For this reason, it is necessary that application rights management architectures allow rights to be upgradeable and that support is provided for the integration of external services such as payment. In our opinion, an application rights specification language has to provide support for the association of changes in rights with use of external services, such as payment.

The PARMA model is being developed according to open-source principles and provides support for the specification of application rights and the mapping to a rights enforcement architecture that also allows the integration of external services using a container model. The PARMA architecture also addresses mobility-specific issues such as disconnected operation, resource constrained devices, super distribution of applications (e.g. over Bluetooth), and the separate and combined delivery of rights. A new audit-based rights model is specifically introduced for mobile devices with occasional connectivity, to account for possibility of loss of network coverage and for the costs of GPRS connection/traffic. Instead of checking application rights and initiating payment at the time of application execution, information about application usage is safely stored on the device. Once device is back on the network, it can transfer the data to the DRM server over GPRS, or via free short range protocols like Bluetooth or 802.11, and users can be billed for the application usage according to their log. A backend architecture is implemented as a web service [08] and allows vendors to manage application rights over a pervasive computing environment.

2.1 Motivation

Typical and popular applications for mobile devices differ from desktop applications due to issues such as user-interface disparities, ease of use and cost of network connectivity [09]. Retail applications for mobile devices, including games, typically have a shorter life-span than their desktop or console equivalents and customers appear less willing to make large up-front investments in expensive software licenses [10]. However, some vendors have had success using novel application licensing solutions such as FleetOnline [11], who distribute their base software for tracking the location of vehicle drivers in the U.K. free of charge and charge per location request or message using sms billing.

It is our belief that the current model of restricting the distribution of mobile application software is delaying the adoption of such software, as users appear unwilling to pay either the high costs of application downloading over GPRS networks or the full-cost license for the application software. For example, existing DRM architectures, such as Nokia's [12], encourage the use of forward lock to prevent the unauthorized further distribution of applications. However, the low cost of distribution of digital media and applications over Bluetooth contrasts sharply with the high cost of GPRS. Given a choice, users would prefer to acquire applications over a communication channel that is free instead of paying for distribution as well as the application or content.

In contrast to existing DRM systems, however, our model encourages the removal of restrictions on distributing application software. Application usage rights are determined during application usage by software. The architecture adapts licenses to the user's usage rights and integrates payment and other container, including a payment service. In particular for the retail and gaming software markets, this model will encourage users to distribute applications that come with default usage rights. More rights can be acquired after application installation by the user without the need to download a new copy of the software.

Application vendors have, in recent years, moved into the mobile application domain and are now releasing products for multiple platforms. With this comes a requirement to centrally manage licenses, payment and application rights. PARMA also provides a unified (or pervasive) application rights management architecture for both fixed network and mobile applications. The integration of application rights management architectures over multiple platforms and devices is achieved using a web services backend architecture. Web services are platform and language independent and enable the easy integration of the application rights management architecture with other backend systems, such as asset management and billing systems.

3 The PARMA Rights Expression Language

Many DRM systems use rights expression languages to define rights on the content usage. Rights expression languages are computer-readable, very often in XML format, and provide information on content, owners of the content, and rights of usage for content users. Traditionally DRM is concerned with copyrighted content (music, video, etc) but recently rights expression languages, such as Open Mobile Alliance's (OMA) REL [13], have been used to define rights on mobile software applications as well [12].

The PARMA architecture primarily manages application usage rights on mobile devices and is not concerned with usage rights on other types of mobile content (music, video, ringtones etc.). Existing DRM systems that support applications as a possible content type are limited and support only simplified models such as whether the user/device is permitted to run the application and whether the user/device is permitted to forward the application to another user/device [12]. PARMA's focus on application usage rights will provide more fine-grained specification of rights over how the application can be used, as well as integration with payment and other services.

PARMA supports an extensible set of rights models including traditional license for unlimited usage, named user license, time-limited, feature-based model, subscription-based, pay-per-use where payment can be in real-time or audited, node-locked, and concurrent usage rights models. We believe that feature-based and pay-per-use models will be most attractive to application vendors and customers alike. All of these rights models introduce requirements on a REL used to define application usage rights and permissions.

Open Mobile Alliance's (OMA) REL Version 1.0 [13] is designed specifically for mobile devices and its schema consists of a subset of ODRL elements extended with elements specific for mobile environments. The OMA REL is the basis for DRM systems by both Nokia and SonyEricsson.

PARMA REL, designed in part for mobile devices, reuses some of the permissions and constraints from the OMA REL (execute, datetime, count etc) but beside simple permissions and constraints PARMA REL requires information about parties involved in issuing rights, amount and means of payment for the application, as well as finer definition of constraints. OMA REL by itself proves too narrow for PARMA REL mobile software licensing requirements and needs to be extended.

ODRL [04], as a superset of OMA REL, introduces some new elements that define parties involved in defining the permissions on content and payment elements PARMA can reuse. ODRL elements, however, do not fully cover PARMA requirements and we required adaptations to ODRL to meet the PARMA REL requirements. ODRL can be adapted in two ways: either by modifying or by extending the ODRL schema. Modification of the schema would allow for finer and more strict schema level validation of elements but we decided to perform this validation separately and instead extend ODRL to stay compliant with other DRM architectures implementing it. PARMA REL, as an extension of ODRL, is backwards compatible with ODRL and therefore with OMA REL. In that way, PARMA REL is compatible with current DRM systems incorporated in mobile phones.

3.1 PARMA REL Schema

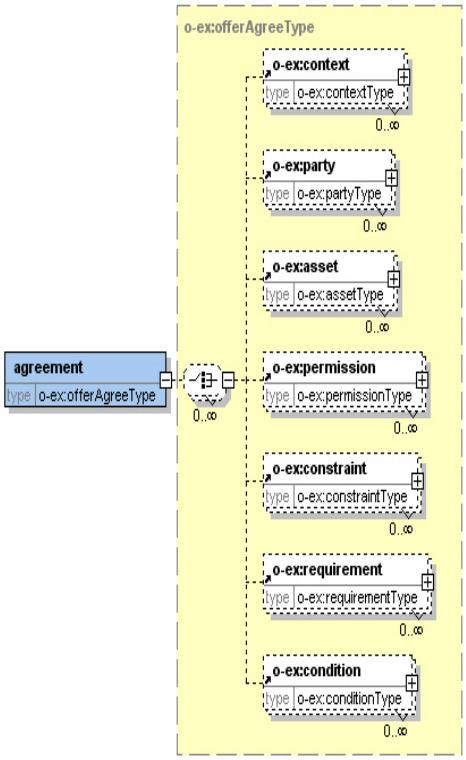

Figure 1 : ODRL Agreement Element

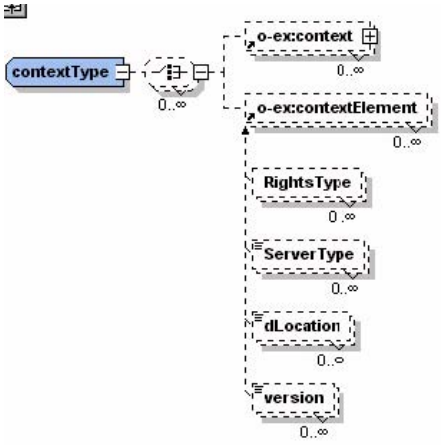

Figure 2 : PARMA Context Element (Extended ODRL)

The first element PARMA REL adds onto the ODRL schema is a rights type. In the PARMA schema 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, and Subscription. These are rights types currently supported by the PARMA architecture and every agreement issued by PARMA will fall in one of these categories. However, RightsType is open for extensions. Since a license model is directly related to offer and agreement types, the ideal case would be to make this element mandatory child element of context element that is child of agreement or offer type. However, ODRL uses the same definition of context element regardless of whether it is a child of agreement, offer, party, permission etc, therefore it cannot be defined as mandatory and this validation will have to be performed separately from the validation

against schema. A tool will be provided that allows application usage rights to be specified and allows validation of those rights against the schema and PARMA validation rules. After successfully validating the rights the tool will convert the rights to aspect code and weave it into the application.

```
<xsd:element name="RightsType" type="xsd:string" substitutionGroup="o-ex:contextElement"/ >
```

Rights objects in PARMA REL must be identified by their location therefore we need an element to store this information in the rights file. The ODRL definition of context elements includes the element dLocation that represents the digital location of the event/entity. PARMA REL uses this element to store a URI that refers to the rights object. In PARMA REL this element is mandatory and validation will again have to be performed against PARMA validation rules rather than against the schema. The context element also includes elements for the version name of the entity, used by PARMA REL to specify name and version of the application, although dLocation alone is often enough to uniquely identify the rights object.

Another group of elements introduced is the URI and type of the rights server.

```
<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>
```

The rights server is responsible for delivering application and usage rights, updates of usage rights, collecting audit data and billing for the application usage. Information on this server, following the ODRL structure, best fits within the party element because the server will represent a party that is either the owner or distributor of the application, or possibly an authorized enterprise that will distribute licenses to its employees. In the case that the party is the owner or distributor we could put this information in the rightsholder element, but in the situation where the party is an enterprise that would not be accurate. Therefore, PARMA REL extends the context element with the new element ServerType, and stores the information about the location of the server in the ODRL-defined element dLocation. Similar to the RightsType element, we would like to restrict the ServerType element to occur only within the context element that is a child of a party element, but again this validation is left to be done against PARMA validation rules before rights objects are converted to code. The ServerType element is mandatory (not by the schema but by PARMA rules validation) and can contain only two values- EDS, in the case of the server being Enterprise DRM Server, hosted by the company licensed to use the application, and VDS, in the case where the application is communicating with a Vendor DRM Server to obtain the rights. This information is important for configuration of the client side DRM engine that behaves differently depending on the type of server it is communicating with.

```
<o-ex:party>
    <o-ex:context>
        <o-dd:dLocation>URIofDRMServer</o-dd:dLocation>
        <parma:ServerType>EDS</parma:ServerType>
    </o-ex:context>
</o-ex:party>
```

Introducing these elements concludes the general part of all PARMA requirements – other elements PARMA included and/or distributed in already existing ODRL elements are directly related to rights information and usually specific to a license type declared in the new RightsType field.

3.2 PARMA Schema and Licensing Model Implementation

ODRL has defined an extensive list of permissions on content, however the only permission applicable to software usage is “execute”. All the constraints PARMA REL uses will be defined as constraints on the execute permissions.

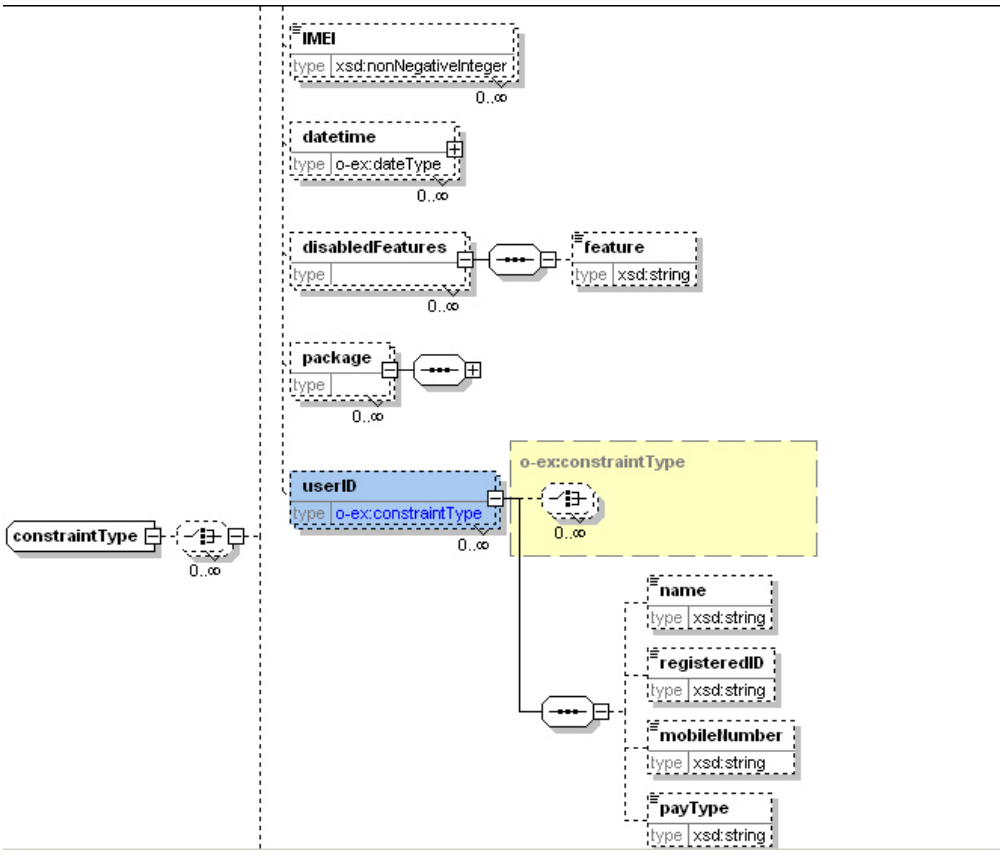

Figure 3 : PARMA Constraint Elements (Extended ODRL)

The only constraint that is mandatory (and its presence is ensured by validation of PARMA requirements) for all values of RightsType is the ODRL datetime constraint, that specifies start or end date, or both, for the period in which the user/device is permitted to execute the application. PARMA supports traditional usage rights models such as node-locked and named-user models by introducing restriction elements to uniquely identify the user. If a node is a user it is identified by the new restriction element IMEI number (International Mobile Equipment Identification number) and the element userID that optionally contains the user's name or his mobile phone number.

- **Feature-Based Model**

Another DRM model PARMA REL is implementing is a feature-based model, where certain features of the software application will be enabled or disabled depending on the agreement. For example, for a game application, a multiplayer over Bluetooth option might be disabled for a demo version of the application. PARMA REL allows the specification of such features using a new extension to the permission element called "disabledFeatures". It is a complex type that contains a sequence of zero or more "feature" elements that contain the name of the disabled feature.

```
<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>
```

However, PARMA also allows for much finer grained control over parts of the application. PARMA schema defines a "returnType packageName.className.methodName(argumentTypes)" structure as an extension of the requirement element. This structure contains the signature of the method within the application source code where the rights check should be performed. Permissions specify the signature of the method and the class/package it belongs to and the licensing code before calling this method

within the application checks the PARMA permissions to allow or deny the method call to be performed. In this way PARMA can, for example, enable game players with a demo license to play the game up to a 3rd level, but for higher levels prompt them to obtain an upgraded full license that supports all levels.

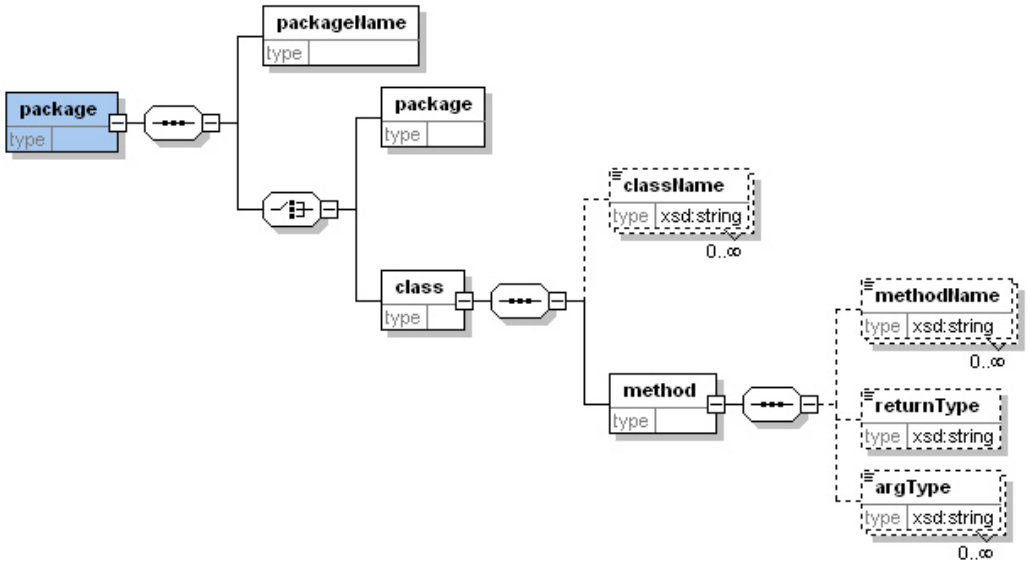

Figure 4 : PARMA Package Element

- **Pay Per Use models** (Audited and Real-Time)

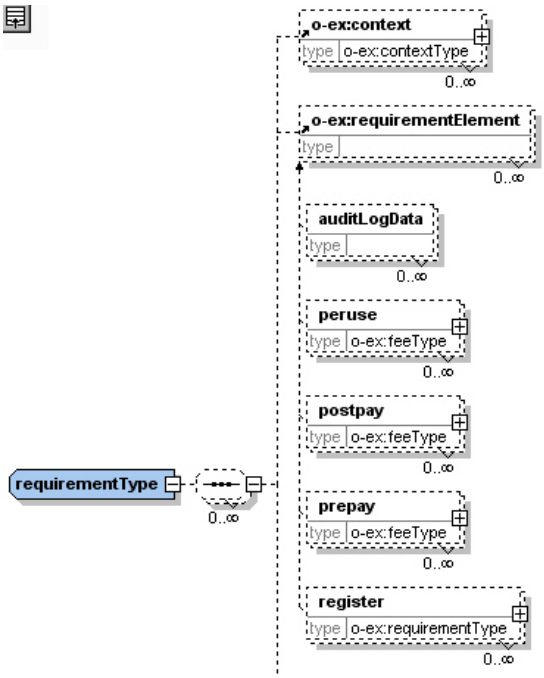

Figure 5 : PARMA Requirement Elements (Extended ODRL)

One of the features that distinguishes PARMA from other similar projects is the very flexible pay-per-use policy. It can accommodate real-time pay-per-use where users make payments immediately

before or after running the application for the certain amount of time or certain number of times. In the audited pay-per-use model information on the application usage is stored in a secure storage area of the mobile device and is occasionally transferred to the EDS and from there to VDS, or directly to VDS. Billing information is produced based on the usage data from this log. The PARMA schema defines the complex element auditLogData as an extension of requirementElement that has several boolean attributes that represent required parts of the log. All attributes are optional and their default values are true.

```
<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>
```

Other elements required for the implementation of PARMA payment policies are already defined in the ODRL schema. ODRL accounts for both prepaid and postpaid payment types and has corresponding elements to store this information. Another ODRL payment element PARMA is using is “peruse” where price per use (in PARMA’s case it is per execution) of the application is stored. Per use can be combined with both prepaid and postpaid payment types, and also both with real-time and audited approaches. Some payment types will require the user to be registered in advance with the rights server, so elements userID, register and dLocation of the registration server will be reused here as well.

```
<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-dd:prepay>
        <o-dd:payment>
            <o-dd:amount o-dd:currency="EUR">2.00</o-dd:amount>
        </o-dd:payment>
    </o-dd:prepay>
    </o-ex:requirement>
    <o-ex:requirement>
    <o-dd:register>
        <o-ex:context>
            <o-dd:dLocation>URIofApplication</o-dd:dLocation>
        </o-ex:context>
    </o-dd:register>
</o-ex:requirement>
```

4 Mapping REL into Application Rights Management Code

4.1 Introduction

In this section we explain the association of PARMA rights with the licensed software application (rights object). Rights have to be tightly-coupled with the application code in order to support fine-grained usage rights models. Before the particular method in the application is called, validation has to be performed that rights specifications allow for this call to be made. Generally, a rights object requires specific calls to a DRM system to be added to the code of the original application in order to perform validation, and these calls are usually scattered over several classes throughout the application. However, such modifications of the original source code to add rights enforcement are very inflexible and impose extra work on application developers.

As an example for the comparison we will describe rights enforcement using XSLM, a licensing standard proposed by the Open Group [03]. The first step of an XSLM-compliant application is to determine what level of application usage rights the server supports. Next, it has to establish the rights enforcing session and request the rights. After these calls are successfully completed, the user is allowed to run the application. During the 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. XSLM-compliant advanced DRM systems require rights objects to make at least eight API calls during the execution and in standard object-oriented applications. These calls cut across many different components of the application (on start up, during the execution and before application termination) and have to be added by the programmer with knowledge of licensing system APIs.

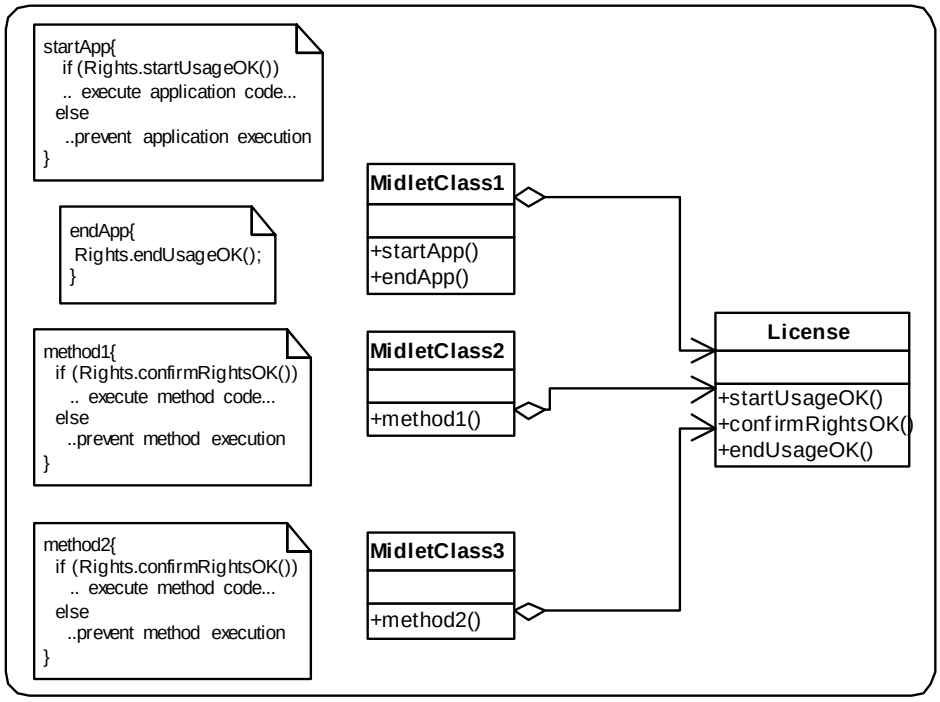

Figure 6 : Scattered Calls to Rights APIs

4.2 Generating code from Rights Files

PARMA uses aspect-oriented development techniques to add rights enforcing code to the application in a non-intrusive manner. Aspect code specifies join points in the original source code where rights enforcing APIs should be called. All the enforcement-related code is encapsulated in the aspect and it is woven into the original application at compile time using an Aspect Oriented Software Development (AOSD) tool, such as AspectJ [14] for Java applications. AOSD does not require any changes to the original source code and therefore it is very easy for developers to add rights enforcement to their application and maintain it in a single concern. In the example of an XSLM-compliant licensing system shown below, the application would still have to make at least eight calls to the DRM system. However, these calls do not cut across multiple, but rather are specified in a single aspect class and are injected into the application at compile time. Also, calls don't need to be added by the programmer with knowledge of APIs – a user-friendly tool is provided that allows the specification of rights without any programming knowledge. The tool generates the rights file in XML format, validates it against the PARMA schema, convert it to aspect-oriented code.

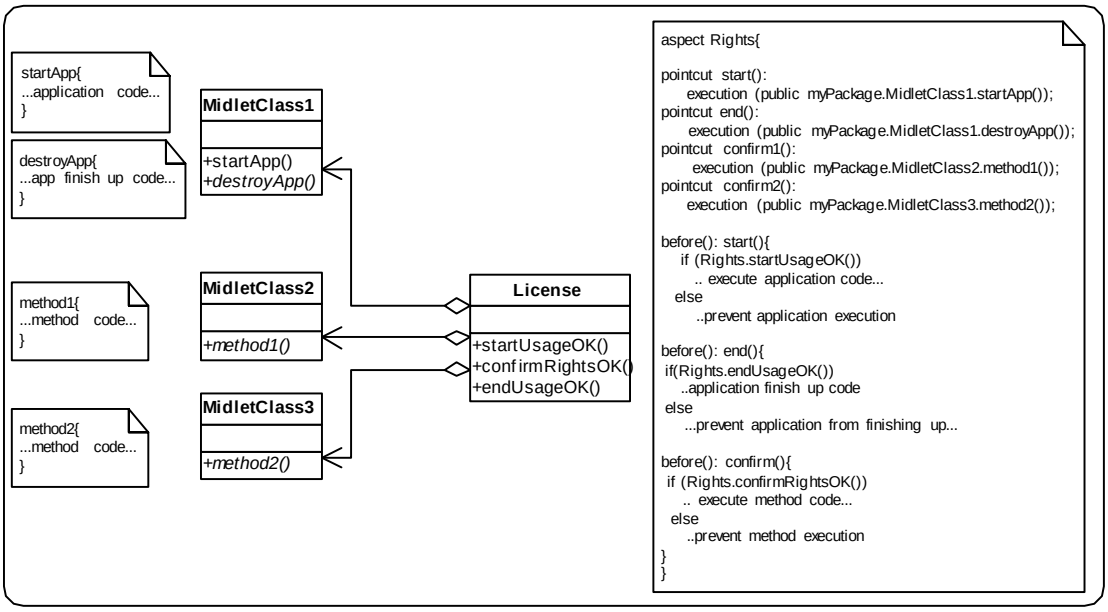

Figure 7 : PARMA AOP Approach

This approach introduces security risks associated with the rights enforcing code. A loosely-coupled approach of rights enforcement and the application decreases the effort of a potential hacker to rip off the licensing calls. We are investigating in techniques to render an application useless when it has been compromised with. PARMA also uses jar signing and verification techniques as well as code obfuscation to minimize the risk of a potential attack.

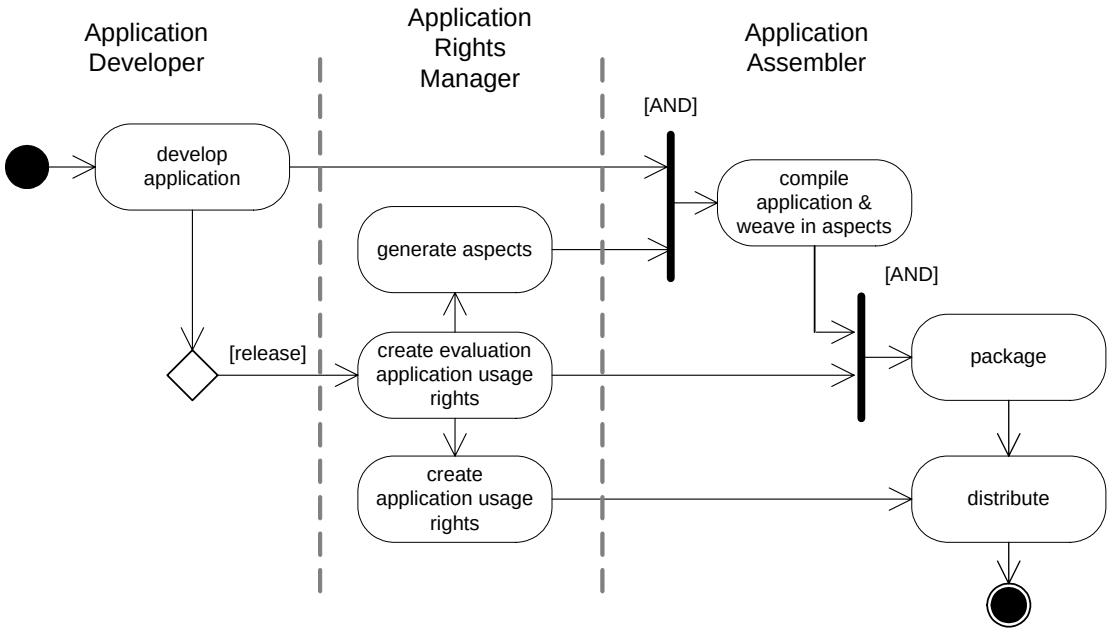

Figure 8 : PARMA Development Life-Cycle

The roles involved in the development life-cycle of an application and its associated usage rights are similar to those in the J2EE specification. On the one hand the developer is responsible for implementing the application without any rights management API calls. The application rights manager, on the other hand, is in charge of the declarative programming part. He maintains application usage rights and creates them for each application. Application rights can be categorized into two different rights models, the evaluation and any other ‘higher’ usage rights. The evaluation rights serve as a super set of the ‘higher’ usage rights and therefore are the basis of the aspect generation. It

presents all code-location where calls to the rights enforcement architecture should be inserted. All 'higher' application usage rights specialize rights based on the evaluation rights. Consequently, it is possible to distribute one rights object with a default evaluation rights agreement and upgrade at runtime of this application to another rights agreement without downloading a new version of this application with specific calls to the enforcement architecture. All 'higher' application usage rights are being made available on a rights locker server which can be co-located with the EDS or VDS.

The application assembler compiles the application with the generated aspects into a rights object, packages it with the evaluation usage rights agreement, and distributes it through a provisioning server.

4.3 Mapping of PARMA REL elements to aspect code

In traditional software licensing license validation is performed at different points in the application source code. Many of the supported rights models require the validation to be done only once, usually at the application start-up, while more fine-grained rights models specify validation to be performed in set time intervals, for real-time billing licenses, or at function or methods calls within the application. The latter type of model enables the use of feature-based rights and the release of upgradeable evaluation versions of software. Such two different licensing models define two different types of mapping from the PARMA rights file to join points in the aspect code.

For models that require the rights to be validated only at the beginning of the application, only one join point is specified in the aspect code, regardless of constraints in the rights definition. That point cut is a method `startApp()`, entry point to all MIDP applications. Rights validation is performed before this method is called and if validation fails application execution is not allowed to proceed.

```
pointcut checkLicense() :
  execution(public myPackage.MIDlet.startApp());

before() : checkLicense () {
  Context appContext = new Context();

  //make an API call to DRM rights enforcement engine
  boolean proceed = DRMEngine.validateLicense(appContext) ;
  if (proceed==true) {
    // do nothing, allow code in the startApp() to be executed and
    // start the application
  } else{
    // prevent the application from proceeding
  }
}
```

The aspect creates the application context object and stores all the available runtime information about the application in that object. This information includes join point information – the package, class, and method it belongs to, as well as types and values of the parameters. The aspect calls `DRMEngine.validateRights(appContext)` to validate the obtained runtime information against the rights. Based on the return value of this API call, the aspect either returns to the main application code and proceeds as normal, or exits the application with the error message informing users they are not authorized to execute the given application at the given time on the given device. Additionally, users can be provided with the option to purchase additional rights that will enable them to proceed with the execution, or instead of exiting the application, user can be allowed to run the demo version of the application.

The mapping of a PARMA rights definition to aspect code for feature-based rights model is more fine-grained and complex. The rights specifies the methods that require rights enforcement before their call. A sample right definition specifies that validation should be performed 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`. Pointcut can have values "before", "after", or "around" depending on whether rights check is needed before the execution, confirmation is required after the execution, or a replacement of a specified method is required.

```
<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>
```

The REL-to-Aspects tool maps these XML elements into aspect joint point as follows.

```
pointcut validateLicense (int i) :
  execution(* myPackage.myClassName.myMethod(int))
  && args(i);

before (int i): validateLicense (i) {

  Context appContext = new Context();

  //make an API call to DRM enforcement engine
  boolean proceed = DRMEngine.validateLicense(appContext);
  if (proceed==true) {
    // do nothing, allow code in myPackage.myClassName.myMethod
    // (int) to be executed
  } else{
    // prevent the myPackage.myClassName.myMethod (int) from
    // proceeding
  }
}
```

As a result, the rights enforcement architecture can evaluate e.g. the level of a game the user is currently in and denies access to the next level according to the usage rights.

5 Rights Enforcement Architecture

In previous sections we introduced how we accomplish rights management with a well-established standard, ODRL, and our aspect-oriented approach to inject DRM enabling code into any JAVA implemented application. Our emphasis on this approach is its ease-of-use. In order to enable an application with DRM only an ODRL-specific rights file is needed.

The following sections deal with the design of our rights enforcement architecture, its components, and which aspects we consider to be important in a successful DRM architecture. We aim to implement a framework for resource constraint and occasional connected devices. PARMA targets the J2ME environment, especially the MIDP 2.0 profile. In contrast to desktop computers mobile devices are not permanently connected to the internet and hence the design of such a framework requires a flexible model to decouple any DRM infrastructure from network connections. Consequently, flexible rights models and DRM systems are necessary to encourage a scenario where a user is not able to connect to a network to comply or upgrade the DRM agreement. With ODRL and our extensions to it we defined the building block to express such a flexible agreement. This section gives an abstract view on the framework and how we partition the components being involved.

5.1 Pervasive DRM Architecture

A flexible license model, such as an audit-based pay-per-use agreement, requires interaction with a DRM server to upload audit information and initiate payment. Additionally, the user should have the choice to migrate to a different rights model. As a result, interoperability with the content provider is crucial and ideally should be standardized. The PARMA architecture consists of Vendor DRM Server (VDS) hosted by the application vendor (or authorized distributor), Enterprise DRM Server (EDS) hosted by an enterprise that purchases sets of rights for its users, and mobile clients running the rights object and connecting to EDS or VDS. The EDS component can be left out in the case of retail rights distribution where clients would communicate directly to VDS to download applications and obtain usage rights.

On the server side, VDS exposes WSDL interfaces for enterprise customers (EDS) to register for the application usage, download the application, request/return usage agreements, select/change/cancel agreement options, send usage data for audit and make payments. The basis for a payment architecture incorporated in PARMA is Web Services Framework for Mobile Payment Services developed by David McKitterick [17]. EDS exposes similar WSDL interfaces to communicate with mobile clients, allow them to register for and download the given application, obtain permissions on its usage and send back auditing data. This architecture uses web services for communication between clients and EDS/VDS and is an upgrade of a licensing architecture developed at Trinity College Dublin by Niall Clarke where clients were communicating with servers using SOAP [18].

The development of the server side architecture is a great deal influenced by numerous U.S. patents on license management systems and in order to avoid infringing these patents several initial design decisions had to be altered. The VDS server will also implement JSR-124 J2EE Client Provisioning specification [30] to enable clients to detect suitable applications, download them together with usage rights and initiate payment.

Communication between clients and EDS somewhat differs from communication between EDS and VDS due to limited resources of mobile clients. The PARMA DRM file is created on EDS, aspect code is generated based on the rights specified in this file and packaged together with the original application. Both aspects and original PARMA DRM file are delivered to the client. The payment framework supports payment for applications and rights via SMS, mobile operators and credit card but is extensible to new payment methods. In the case of enterprise licensing, payment is not done between clients and EDS/VDS, but usage data is collected on EDS from all clients and then payment is settled between EDS and VDS.

For a successful DRM architecture security considerations should be taken into account at initial design decisions. However, cryptographic security measures are not the focal part of this paper. We assume the client participates in a trusted environment, her juridical privacy is respected, and communication channels are sufficiently secure. Further we assume that mobile code offers some degrees of security in terms of protecting the rights of the copyrights holder with watermarking, and provide some level of tamper-resistance with obfuscation techniques [04, 05]. We assume secure storage is achieved with encrypting usage data, keeping the DRM agreement and secret keys on a SIM card to prevent compromising with sensitive data. A detailed description of managing the persistent state of DRM systems is given in [19].

One of the goals of this project was to design generic components to be able to adapt new features easily. The following section on the container architecture sheds some light on the infrastructure of our framework. Then services are introduced and finally application provisioning, related and future work are discussed.

5.2 Container Architecture

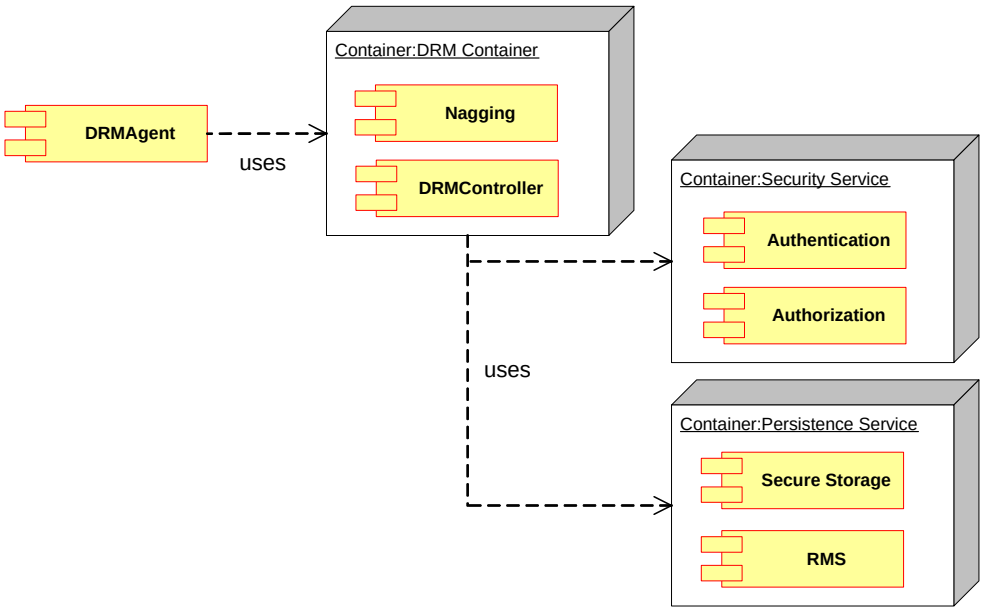

Figure 9 : DRM Container Architecture

The DRM framework provides a set of mechanisms whereby applications can be configured to support flexible rights management and billing options as well as a secure execution environment. The DRM framework is built around a light-weight interface-based Inversion of Control (type I) container architecture. The DRM support functionality is provided in isolated components (plug-ins). A component has to be registered with the container and then exposes its interface to other components which are interested in it. Components are not fixed to an implementation class, but rather to an interface. The advantage of this approach is the easy exchange of the implementation class without breaking dependent APIs. The fully qualified interface name of a component acts as a primary key. The container makes sure that only one class implementing this interface is instantiated. This can be seen as a pseudo singleton without implementing the singleton contract in a component explicitly. Consequently, components either need to be immutable or their methods synchronized. Instantiating a container is a two-phase process. First, all registered components are instantiated and in the second phase their dependencies are resolved. The implementation class can be either retrieved through a properties file which maps a plug-in to the implementation or it can be specified at registration time. All components implement a service provider interface (SPI) which declares callback methods for life-cycle, and dependency management.

The container is the central repository of instantiated components. Therefore memory management can be handled in one single place, rather than in multiple spaces. As well this approach offers a bit of security, because the container is the one and only instance which is responsible for creating components with a specialized factory class. This class and the repository can be exchanged, if special memory handling code is required. Common patterns in a memory constraint environment are object pooling and fixed allocation which both can be easily adapted in a new implementation [31].

The container can be arranged in a hierarchical structure whereby each child container is responsible for a specific task. In our architecture child-containers implement services which are only visible to its parent.

5.3 Services

The Services are controlled by a `DRMController` which is a component itself. The `DRMController` implements the delegation logic to each service and configures them when the container starts up. The following sequence diagram illustrates the interaction of the `DRMController` with the services provided by the child containers.

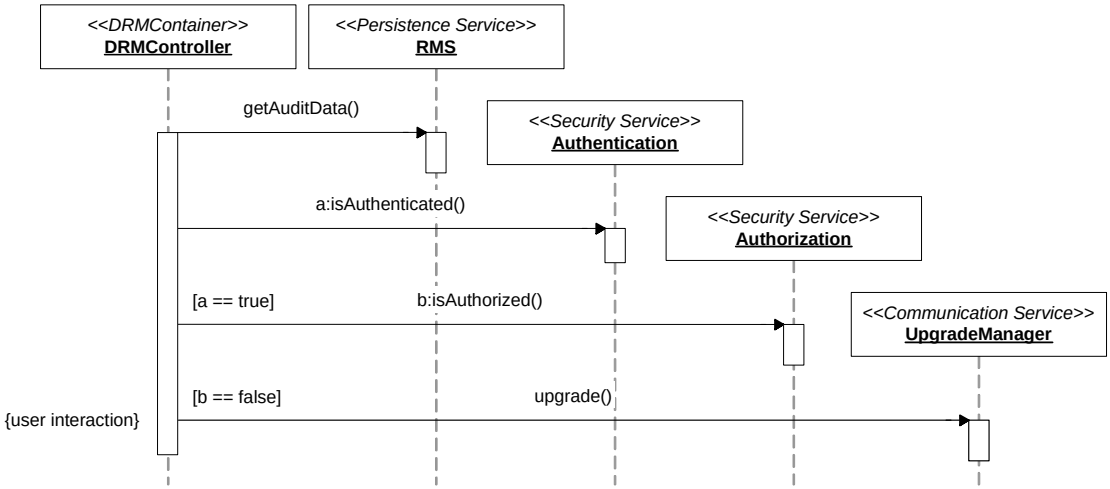

Figure 10 : DRMController Interaction

After the `DRMContainer` is initialized and a call to the `DRMAgent` is made, the controller first accesses the Persistence Service's Record Management System (RMS) to retrieve the runtime audit data and updates them accordingly. These include usage data, subscription date, etc. This data is initially recorded to a persistent store when the application first starts up. Usage data is appended with a Message Authentication Code (MAC), encrypted and persisted into the RMS, whereas the usage rights according to the DRM agreement are kept on a secure tamper-resistant smart card. In addition the Persistence Service provides a one-way-counter for the usage statistics which is triggered at start up time of the application.

The user is then authenticated based on certificates delivered with the application. If authentication returns successfully, the authorization service is consulted to check against the DRM agreement. The authorization service implements a simplified role-based policy language. The roles participating in PARMA are shown in Figure 11:

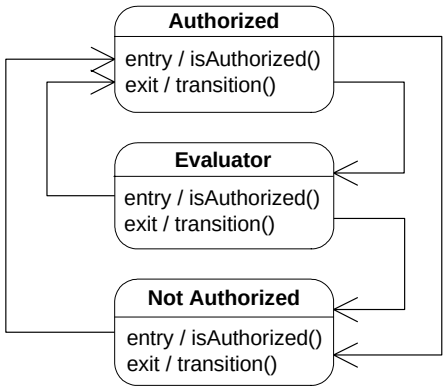

Figure 11 : DRM Roles

An authorized role represents a user with valid rights, whereas the evaluator represents a user who only is allowed to evaluate/preview an application. This approach helps in achieving a flexible rights model. Users should be able to overdraft rights agreements and comply at a later stage. The overdraft is expressed in ODRL. If the overdraft exceeds the tolerance level the user falls back into an evaluation mode. If the evaluation period is exceeded the user has to upgrade to a new rights agreement. This

process may involve payment of some sort. However, these roles are not fixed to our DRM system. If the ODRL agreement does not express such a flexible scenario, roles will be established according to the agreement.

If authorization fails the user has the option to upgrade the agreement via the communication service. The communication service abstracts the transport layer and applies security transparently, such as client side authentication, content encryption. On top of the transport and security layer are application-specific components, like the UpgradeManager.

5.4 Provisioning

OMA DRM 1.0 specified three methods for digital rights management: Forward-lock, separate and combined delivery of usage rights and the rights object.

A combined delivery of the rights object and the rights addresses the preview or evaluation scenario. We assume that evaluating applications based on restricted rights is a major aspect for customer and enterprise adoption of software. Separate delivery is a two stage process, where the rights object is delivered in an encrypted form to the client and the rights including the decryption key and maybe a certificate are delivered via WAP Push or alternatively via the Push registry which is defined in MIDP 2.0. However, the application management system (AMS) in Symbian OS does not yet support the installation of encrypted applications.

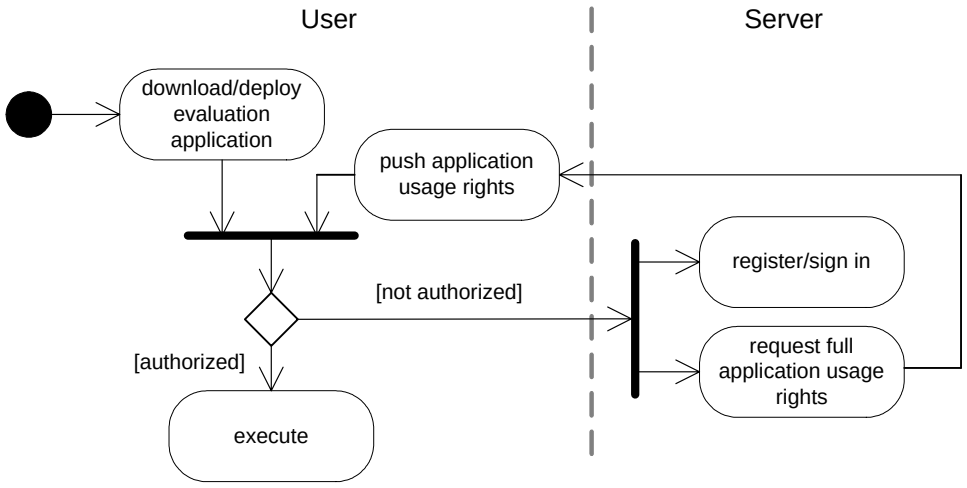

Figure 12 : Provisioning Cycle

PARMA combines both methods, separate and combined delivery. The user first downloads the application in combination with the evaluation usage rights from a content server, evaluates it and may upgrade later to a full version. Upgrading requires the negotiation of an agreement which is then delivered via the Push registry to the client. The MIDlet responsible for the Push request initializes the record store and stores the rights and the certificate on the smart card.

6 Related Work

In [22] a framework (FILIGRANE) is proposed which tackles relevant security issues in the mobile code commerce. The actors in a mobile code commerce scenario are identified which participate in the functional model of FILIGRANE. Although, this project does not address a DRM standard they implement practical security techniques to secure mobile code commerce including contract handling, provisioning, software protection, and usage control. A very interesting part of this paper is the analysis of the cost of piracy to break the protections and the measures they have taken to maximize the cost of piracy. Those measures involve encryption, obfuscation and watermarking of code and the use of smart cards to store sensitive data (e.g. the rules (rights) and the private key). However, encryption on a byte-code level is very easy to breach and so get access to the original byte-code. JAVA uses classloaders to

load classes and deliver them to the JVM via one well-defined final method `ClassLoader.defineClass(String, byte, int, int, ProtectionDomain)`. All classes must be delivered to the JVM via this aforementioned method. Although this method is final any class can be dumped in clear byte-code. To achieve this, the ClassLoader implementation has to be modified and repackaged (see rt.jar in the JAVA distribution) or specified in the -Xbootclasspath option [23].

This attack is more difficult to perform on J2ME devices. In contrast to J2SE classloading mechanisms can not be overriden or extended in application code. Further, it is more difficult to compromise with a J2ME installation on a mobile device, because the classloading capabilities are implemented in a native component, called Application Management System (AMS).

Nokia and SonyEricsson are in the process of implementing OMA REL based DRM systems on their mobile devices [1]. The version supported so far is OMA DRM 1.0 [13] that specifies three ways to protect copyrighted content – forward lock, combined delivery and separate delivery. Several Nokia phones so far support forward delivery while SonyEricsson's Z1010 is the first device that implements the full OMA DRM 1.0 specification supporting all three delivery methods. Both Nokia and SonyEricsson plan to support the full specification on all of their phones in forthcoming releases. Although efficient for content, this type of DRM system is not flexible enough for software applications as it does not support feature-based licensing. Also, the complete implementation of OMA DRM is done on the device, without DRM servers and back-end implementations that support client provisioning, payment and flexible license models.

ContentGuard[06] released their patented DRM system based on XrML eXtensible Rights Markup Language. XrML is not primarily aimed at wireless devices and currently does not have implementation on mobile phones.

7 Status of PARMA and Future Work

The development of our framework is in its early stages. Areas which require more attention are privacy, JavaCard access, a policy language in XML, and the interoperability with the server entities.

Privacy is a big issue in DRM systems and contributes significantly to the adoption of such a system, if done properly. The legal and technical privacy rights need to be explored in future versions of our framework. According to Brands' book [25] protecting one's privacy involves restricting the amount of data being submitted electronically to an absolute minimum because once submitted this information is not under control of the user anymore. It can be abused for e.g. marketing purposes. Additionally, different transactions of a user should not be linkable, unless the cost outweighs the benefits. In [24] security and privacy issues are assessed and applied to Digital Rights Management Systems. We are going to explore this area further and implement a workable solution with practical privacy measures.

The ability to store sensitive data in a tamper-resistant manner is crucial to every DRM system. In the very near future the Security and Trust Services API for J2ME (JSR-177) [32] will be available on J2ME mobile phones. This specification spans security services which rely on the interaction with a "Security Element" to provide secure storage, secure execution, and custom security features to allow e.g. payment. An implementation which facilitates the access to a smart card to store and retrieve sensitive data can be further abstracted with JSR-177. In combination with JSR-177, Security Assertion Markup Language (SAML) [26], XML digital signatures [27], XML encryption [28], Web Services secure XML protocol family (WS-Security [29]) we can provide a more generic and sophisticated communication layer to the content servers. An interesting task would be to define a Web Services-based Digital Rights Messaging protocol to exchange audit data and renew a DRM agreement.

Moreover, the generic design of our framework encourages the use of a dedicated XML-based rules description in order to configure the policy language used in our framework. The rules need to be transformed via XSL/T from the policies (rights) in the DRM agreement. It would then be possible to describe a DRM agreement in a different language, such as XrML without changing the implementation.

A big task is the interoperability with the content server. The entities involved in a complete DRM architecture need to be implemented either in a single server, which is not ideal, or as separate independent entities. Further, interoperability of the client with the back-end servers can be increased in a standard way with implementing the J2EE Client Provisioning specification [16].

8 Conclusions

We have introduced our extensions to the ODRL schema to represent a flexible software usage rights scenario. ODRL was extended to add more details about payment and user identification, to support constraints specific to mobile environment, and to differentiate between rights models for mobile software applications. Our experience with ODRL in general is that it is difficult to validate its schema because it is designed in an abstract way and allows many interpretations. Therefore, the semantics of application usage rights introduced in PARMA requires additional validation to make sure that all parties, the assets and rights for the assets conform to the PARMA model.

With our framework we accomplished a prototype for pervasive software licensing which enables and encourages the use of a flexible DRM agreement, such as feature-based policies. PARMA's aspect-oriented approach to usage rights code insertions also greatly reduces application developers' involvement in usage rights specifications for these flexible models and separates usage rights concerns from application and management concerns. Our framework is designed for extensibility and provides therefore a solid basis for our future work.

Bibliography

[01] Johannes Rastas, "OMA DRM - Technical overview", Nokia Media & Music Digital Rights Management (DRM) Workshop, November 4th, 2003.

[02] Macrovision, "Overcoming the Software Licensing Complexity Crisis", FlexNet White Paper, 17 Dec. 2003. <http://www.macrovision.com/pdfs/PlatformWP.pdf.>.

[03] XSLM Resource Center. Dec. 16 2003. <http://www.xslm.org>.

[04] R. Iannella. Open Digital Rights Language (ODRL) Version 1.1, Aug. 8 2002, <http://odrl.net>.

[05] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. "Aspect-oriented programming". In ECOOP'97, LNCS 1241, pages 220--242, 1997.

[06] ContentGuard Inc. eXtensible rights Markup Language (XrML), Version 2.0. November 2001, <http://www.xrml.org/>.

[07] Barry Fox, "Subversive code could kill off software piracy ", New Scientist, 10 October 03, <http://www.newscientist.com/news/news.jsp?id=ns99994248>.

[08] W3C (2003) Web Services Description Language (WSDL). Dec 17 2003. http://www.w3.org/TR/wsdl.

[09] Bellamy, R., Brezin, J., Kellogg, W.A., Richards, J. and Swart, C. "Designing an E-Grocery Application for a Palm Computer: Usability and Interface Issues". IEEE Personal Communications, 8, 4, 60-64, August 2000.

[10] Antony Adshead, "Cheaper field staff tracking services", Computer Weekly, May 13, 2003.

[11] gamesindustry.biz, "Nokia N-Gage tops mobile game download chart", Dec 18 2003, <http://www.theregister.co.uk/content/68/34241.html>.

[12] Matt Volpi, "Nokia DRM Tools", Nokia Media & Music Digital Rights Management (DRM) Workshop, November 4th, 2003.

[13] Open Mobile Alliance, "OMA Digital Rights Management version 1.0", 19 Nov. 2002, <http://www.openmobilealliance.org/tech/docs/index.htm#DRM>.

[14] IMEI codes, Dec. 17 2003, <http://www.accesscomms.com.au/imei.htm>

[15] Gregor Kiczales et Al, "An Overview of AspectJ", In ECOOP, pages 327-- 353, 2001.

[16] JCP, J2EE Client Provisioning Specification, JSR-124, <http://www.jcp.org/aboutJava/communityprocess/first/jsr124/>.

[17] David McKitterick, A Web Services Framework for mobile payment services, M.Sc. thesis at the Department of Computer Science, Trinity College Dublin, 2003.

[18] Niall Clarke, Distributed Software Licensing Framework based on SOAP, B.A. thesis at the Department of Computer Science, Trinity College Dublin, 2003.

[19] William Shapiro and Radek Vringalek, How to manage persistent state in {DRM} systems, Digital Rights Management Workshop, 2001.

[20] A. Monden and H. Iida and K. Matsumoto and Katsuro Inoue and Koji Torii, A practical method for watermarking JAVA programs, compsac2000, 24th Computer Software and Applications Conference, 2000.

[21] developers.sun.com, FAQ <http://developers.sun.com/techtopics/mobility/midp/questions/obfuscate/>.

[22] G. Hachez, L. Den Hollander, M. Jalali, J.-J. Quisquater, and C. Vasserot, Towards a Practical Secure Framework for Mobile Code Commerce, In Pieprzyk et al. POS00, pages 164--178. 7, 2000.
[23] Vladimir Roubtsov, Cracking JAVA byte-code encryption, <http://www.javaworld.com/javaqa/2003-05/01-qa-0509-jcrypt_p.html>, 2003.
[24] Joan Feigenbaum, Michael Freedman, Tomas Sander, Adam Shostack, Digital Rights Management Workshop, p. 76-105, 2001.
[25] Stefan A. Brands, Rethinking Public Key Infrastructures and Digital Certificates, MIT Press, Cambridge Massachusetts, 2000.
[26] OASIS Security Services TC, SAML, 17 Dec. 2003 <http://www.oasis-open.org/committees/security/>
[27] W3C, XML-Signature Syntax and Processing, 17 Dec. 2003 <http://www.w3.org/TR/xmldsig-core/>.
[28] W3C, XML-Encryption Syntax and Processing, <http://www.w3.org/TR/xmlenc-core/>.
[29] OASIS Security Services TC, WS-Security, <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss>.
[30] JCP, J2EE Client Provisioning Specification, <http://www.jcp.org/en/jsr/detail?id=124>.
[31] James Noble and Charles Weir, Small Memory Software – Patterns for systems with limited resources, Addison-Wesley, 2001.
[32] JCP, Security and Trust Services API for J2ME,< http://www.jcp.org/en/jsr/detail?id=177>.