A Generic DRM Framework for J2ME Applications

Nuno Santos, Pedro Pereira, Luís Moura e Silva

WIT-Software
Rua Pedro Nunes, IPN, 3030-Coimbra, Portugal
Email: luis@wit-software.com

Abstract

Recently, a new generation of mobile phones with support for Java has been taking widespread acceptance by the market, creating a business potential for downloadable Java Games and enterprise applications. However, it is relatively easy to forward Java programs between two Java phones. This opens the door for illegal peer-to-peer forwarding, with the consequent loss of revenues for content providers and operators. Therefore, DRM solutions are essential in order to protect copyrighted Java applications.

In this paper we present a generic DRM framework that supports different solutions for protecting the copyright of Java applications. This framework is mainly targeted to Mobile Operators, it is totally transparent to content providers and does not require any special support at the user's handsetss. It also allows the development of new custom built DRM solutions, providing a flexible platform for Java oriented DRM techniques.

Keywords: Java J2ME; DRM; copyright-protection; code instrumentation.

1. Introduction

The first generation of Java [1] enabled phones were very limited in terms of functionality. They were only able of downloading MIDlet[1] applications from the network and of executing them. They offered no simple way to copy a Java application to another terminal or PC. These limitations were a natural Digital Rights Management (DRM) [2] solution, since the only way to obtain a Java application was by downloading it from the network. However, the most recent mobile phones are much more feature-rich. These terminals often have a user visible file-system and are able to connect to another terminal or PC using USB, Bluetooth or infrared. This makes it easy for users to copy Java applications to other terminals. In this scenario, DRM is essential to prevent users from illegally forwarding copyrighted Java MIDlet applications.

[1] *MIDlets are small applications written in the Java programming language that run in all mobile phones that support the Mobile Information Device Profile.*

The current terminals are mostly based on the J2ME CLDC[2] 1.0 [3] and MIDP[3] 1.0 [4] specifications, which do not have any kind of DRM support. Hence, those terminals are an easy target for copyright violations, which can result in a significant loss of revenues for the application developers and the mobile operators.

This document presents a DRM framework that was developed by WIT-Software and has been integrated with a commercial Java Download Platform. The framework allows application providers to add DRM protection to MIDlets applications in their binary form, requiring no access to their source code. It works by doing some code-instrumentation of the MIDlet JAR file, adding copyright-protection code that will be executed on the terminal when the user starts that application contained in the JAR file.

The framework is sufficiently generic, allowing DRM solutions to be developed independently and deployed on a case basis. It can be used as a standalone DRM tool or as an Application Programming Interface (API). The former option is especially interesting for application providers, who can use the API to integrate the DRM framework with Over-The-Air (OTA) [5] provisioning systems. In this way, the instrumentation of the MIDlet is delayed until download time, allowing the system to choose the most appropriate DRM solution for the mobile phone that is requesting the MIDlet application.

The rest of this paper is organized as follows: Section 2 describes the related work on DRM solutions and code instrumentation. Section 3 discusses the suitability of J2ME MIDP 1.0 and 2.0 profiles for implementing DRM solutions. Section 4 presents the structure of the DRM framework. Section 5 describes three DRM solutions implemented for the framework. Finally, Section 6 concludes this paper.

2. Related Work

The Open Mobile Aliance (OMA) has defined a specification of DRM systems for mobile devices [6]. This specification addresses the protection of any type of media that can be delivered to mobile phones. This includes music, video, and applications, among others. It defines three methods of distributing content and right objects[4]:

---

[2] *Connected Limited Device Configuration.*
[3] *Mobile Information Device Profile.*
[4] *A rights object specifies the way the content can be used, like how many times it can be used, if copying is allowed or not, etc.*

- **Forward Lock:** The content is delivered unencrypted to the device, without any rights objects. It is up to the device to enforce a default set of rights and ensure that the content cannot be forwarded.
- **Combined Delivery:** The content is delivered unencrypted, together with a rights object. The mobile phone enforces the usage permissions specified in the rights object.
- **Separate Delivery:** The content is delivered encrypted. A rights object is delivered separately using WAP push. Since the content is encrypted, it can be forwarded freely. The receiving users will have to obtain a license before using it.

Several members of the industry, such as Ericsson, Siemens, Nokia [7] and DRMSecure [8], have already committed to the OMA DRM specification and are implementing parts of it in their products[5]. The main limitation of the DRM OMA specification is that it requires special support from the mobile device. Therefore, this specification does not solve the problem addressed by the framework presented in this paper, which is to protect Java applications delivered to the existing portfolio of Java-enabled phones.

There are some other companies with similar commercial offerings. The SDC Java DRM [9] is a technology for delivering content to mobile devices. The content is packaged inside a container together with the code necessary to access it. This container is protected using obfuscation and encryption techniques. In the device, the code is interpreted by a Java Virtual Machine on the device, enforcing the DRM rights. The available documentation was not very complete or clear, but it seems the system requires the presence of private keys in the mobile phone side. There was no description about key distribution and the portability of this scheme.

Other proposed scheme is MacroSafe [10], a product from Macrovision for content delivery. The solution is similar to the Separate Delivery mode of the OMA-DRM specification. Encryption is used to protect the content, which is delivered with a rights object. The client needs to retrieve the encryption key to be able use the content. This solution requires the presence of the MacroSafe's Client software on the client's device. There is no mention to whether the client will run on a J2ME device, but this seems unlikely since the specification of the client software seems to impose some device requirements that are not currently achieved by the J2ME devices.

[5] Nokia has recently launched a mobile phone supporting the OMA DRM standard: Nokia 6220.

3. J2ME MIDP 1.0 and DRM

Some hardware support for encryption and unique identification numbers is extremely important for DRM mechanisms. The presence of these resources is the basis for implementing strong DRM measures [2]. Unfortunately, the CLCD 1.0 and MIDP 1.0 specifications do not provide any adequate support. They define a very limited execution environment. In particular, some important features are missing, namely:

- There is only a very limited access to the file-system. This is done by means of a Persistent Record Store (PRS), which is a device-managed container. This makes it easy for any user to access the data that has been written by the MIDlet application on the mobile phone;
- There is no way of obtaining the terminal IMEI or any other type of device identification;
- Only a subset of the Java API is supported. There is no support for JNI (Java Native Interface), reflection or cryptography;
- Most devices running MIDP 1.0 are very limited in resources (CPU and memory). This is not a limitation of the specification, but it prevents the use of memory or CPU intensive DRM mechanisms. In particular, it makes strong cryptography almost unpractical.

These limitations severely constrain the type of DRM solutions that can be implemented. Traditional implementations like encryption, digital signatures, secure hardware and unique identification of the device are hard or even impossible to apply in such a restricted environment. Nevertheless, it is still important to have some kind of protection. Even if a DRM solution is not very hard to break, it will be useful if it prevents a significant number of DRM violations. Therefore, it is important to use the available support of the J2ME environment in the best possible way to protect the copyrights of J2ME downloaded applications.

There are some features of the MIDP 1.0 specification that can be useful for the implementation of a DRM solution:

- It is possible to read/write to the PRS, thus making it possible to keep a license together with the MIDlet JAR;
- It is possible to extract values from the JAD and the JAR manifest;
- It is also possible to obtain the current local time of the mobile phone.

Outside the specifications, some vendors provide proprietary extensions to MIDP 1.0. For instance, with the Siemens phones it is possible to obtain the IMEI identification. But exploiting vendor-specific extensions will most likely result in different and incompatible DRM

implementations, each one suited only for a certain type of terminal. This approach has some management difficulties, like ensuring that the right DRM solution is used on the user's device. There are several ways to solve this problem:

- MIDlet applications contain the implementation of all possible DRM systems. This would increase considerably the size of the application;
- Create different versions of the same MIDlet application for each type of terminal. When a client requests a MIDlet application, the most appropriate version will be downloaded. Even so, this may create some difficulties in managing a full package of MIDlet versions;
- Create only one version of the MIDlet application. When a client requests the MIDlet, the mobile phone is identified and the MIDlet is instrumented, by adding the corresponding DRM system. The drawback of this solution is the extra time it would take to instrument the code on-the-fly.

The framework described in this paper implements the last solution. It keeps the DRM implementations separated from the MIDlet applications. When there is a J2ME application download, the framework is used to instrument the MIDlet application with a specific DRM solution. This is done by including the DRM implementation classes in the MIDlet application and by performing some code transformations. The DRM code is executed the first time the Java application is started on the mobile handset. The generated MIDlet application can be sent directly to the terminal. Once there, the DRM code performs the necessary verifications and it only executes the MIDlet if there is a valid license.

4. J2ME DRM Framework

4.1 Overview

The main objective of the DRM framework described in this paper is to extend the Over-The-Air (OTA) provisioning servers with support for locking MIDlet applications at download time. In particular, it has the following goals:

- **Generic:** It should be possible to support different DRM solutions. This is necessary due to the limitations of the J2ME specification, which makes it difficult to implement a generic solution. Therefore, it is not realistic to expect that one solution will be adequate for all terminals and for all situations. It is more likely that a number of different solutions will co-exist, each one tailored to a different terminal. Hence, a framework that is able to support different solutions may have a significant advantage.

- **Transparency to the content provider:** The content provider should only provide an unprotected MIDlet application, without having to worry about DRM systems.
- **Transparency to the user:** The average user should not be aware of the presence of DRM mechanisms.

4.2 Description

The DRM framework consists of a component that integrates with the OTA server. Figure 1 presents an overall overview of the architecture.

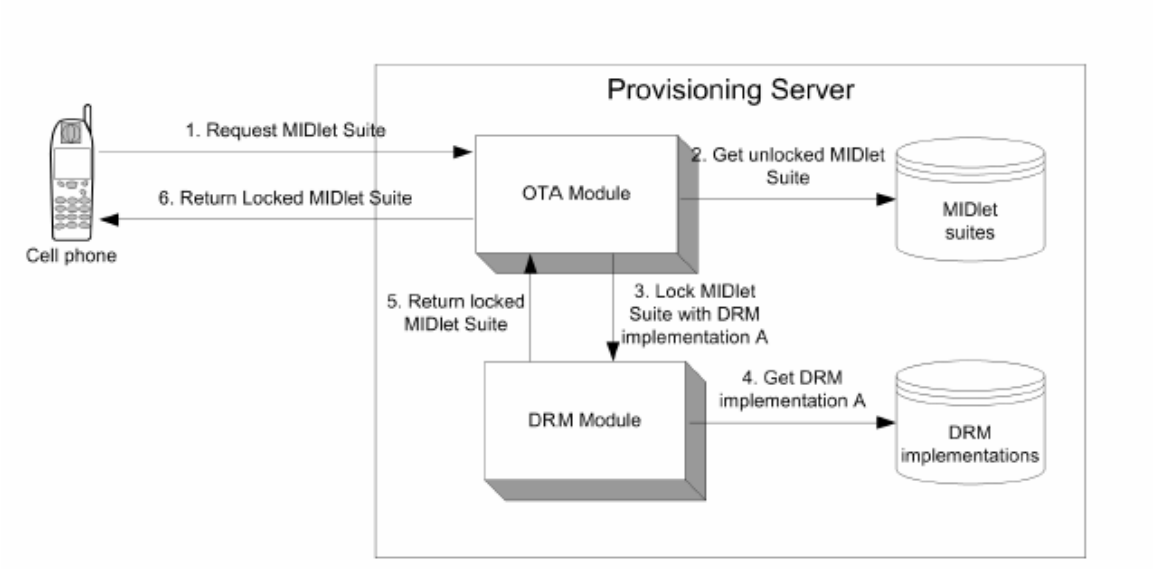

Figure 1: Architecture of the DRM framework

The DRM module maintains a database of DRM implementations. When a MIDlet application is requested by a client, the OTA server decides if it is necessary to protect it. If so, it chooses the DRM solution that is most adequate to the mobile handset, and uses the DRM framework to protect the MIDlet application with the corresponding DRM solution.

The DRM framework instruments the MIDlet JAR: it includes the DRM classes and changes the MIDlet's code so that when they are executed in the client's device, the DRM verification algorithm will be called before anything else. After being transformed, the MIDlet application is re-packaged, obfuscated and pre-verified. The result of this process is a MIDlet JAR ready to be sent to the mobile phone and protected by a DRM algorithm.

Structure of a DRM Solution

The DRM framework is general so that different DRM solutions can be developed separately and can be deployed into the framework in their binary form. For this to be possible, DRM solutions must follow certain rules of packaging, as described next:

- **Terminal side implementation:** A JAR file containing the classes that will be packaged together with the MIDlet application.
- **Server side instrumentation classes:** An optional server side implementation. These classes are only used at the server. Their purpose is to allow the DRM solution to perform custom instrumentation on the MIDlet and on the client side implementation. For instance, if the DRM solution needs to write a certain value to the MIDlet during code instrumentation, it is not safe to write it on the JAD, as it would be easily visible by the user. Instead, it is safer to insert the value inside the class files of the MIDlet application. This way, the value will be hidden from the average user.
- **Configuration file:** A standard Java properties file, specifying several properties, like the name of the main classes for the phone and server side implementations. It also describes the properties of the available solutions that can be accessed during instrumentation.

The terminal side implementation must provide a class extending an abstract class `wit.j2me.drm.DRMVerifier` (provided by the DRM framework). This class defines a `verify()` method, which should implement all the DRM verifications, starting the MIDlet if it succeeds and aborting the execution otherwise.

A similar requirement applies to the server side implementation, which must provide an implementation of the `wit.j2me.drm.CustomInstrumenter` interface. It defines an `instrument()` method, which should perform the custom transformations to be done at the server side.

Locking Process

This section explains how a MIDlet application is instrumented with a DRM solution. A MIDlet application has one or several MIDlets. The DRM algorithm must be executed before any other code of the MIDlet, so it is necessary to change the MIDlet to redirect execution to the DRM algorithm as soon as possible. When the user starts a J2ME application on its mobile phone, the application manager will show all the existing MIDlets, allowing the user to choose the one that will be executed. After that, the application manager will create an instance of the MIDlet's main

class and execute the `startApp()` method. In this process the following methods of the MIDlet are invoked, in this order: static constructor, instance constructor, `startApp()`. In order to execute the code corresponding to the DRM solution, it is necessary to intercept execution in the static constructor, which is the first method that is executed. Next, there is a description of what happens during instrumentation:

1. All classes contained in the terminal side implementation are inserted in the MIDlet application. This makes the DRM code available in the mobile phone. It is still necessary to change the original structure of MIDlet Jar, so that the DRM code is executed.
2. Each MIDlet is instrumented. The original static constructor is renamed and a new one is created. This new constructor calls `verify()` on the `DRMVerifier` implementation provided by the DRM solution (which was inserted in the MIDlet application in the previous step). This way, when any MIDlet is created, it will start the DRM algorithm.
3. All other public methods of the MIDlet, which can be called by the application manager, are renamed and replaced by stubs. These stubs will only call the corresponding methods of the original MIDlet if the DRM algorithm finished successfully. This is necessary because some DRM implementations might take a long time to execute. In this situation it is desirable to show a message to the user. This is not possible if the thread that is used by the application manager software to create the MIDlet (the dispatch thread) is blocked in the verification process. So, a new thread should be created and the dispatch thread released. In this situation, the dispatch thread will believe that the MIDlet is ready to execute and call the instance constructor and the `startApp()` method. If these methods were not replaced, the MIDlet would be started while the DRM algorithm is running.
4. (Optional) If a server side implementation is provided, it is used to perform custom instrumentation. The `instrument()` method of the `CustomInstrumenter` implementation is invoked, passing as parameters the MIDlet application and the client side implementation.
5. The MIDlet is re-packaged and obfuscated. Obfuscation makes it harder to understand the locking mechanism and to reverse engineer it. It also makes it smaller. This is important to minimize the extra footprint created by the DRM classes.
6. Finally, the MIDlet is pre-verified. This is necessary because the transformations that were performed removed the `StackMap` attribute that was on the original MIDlet classes.

At the end of this process, the MIDlet application is ready to be sent to the terminal. The instrumentation is performed using the `Javassist` library [11]. There are other options for

instrumenting Java code, the most significant one being the ByteCode Engineering Library (`BCEL`) [12]. An initial prototype of the DRM framework made use of BCEL. But it soon proved to be too complex for the task at hand. Therefore we decided to use `Javassist`, which focuses mainly on high-level manipulations. These higher-level abstractions proved to be more appropriate to the task, allowing us to implement the solution in less time and in a much simpler way.

Disadvantages

The DRM framework described before has a few disadvantages.

- The locking process takes a few seconds. Depending on the server and on the DRM solution, it may take something from one to five seconds. This may create a noticeable delay for the client who is waiting for the Java application.
- The MIDlet application size generally increases, as more classes are added to it. This mainly depends on the number and the size of the classes to will be added. It also depends on whether the original MIDlet was obfuscated or not. If it was not, it is even possible that the size decreases, as the size reduction obtained by obfuscating the MIDlet classes might compensate for the size of the new classes.
- Finally, there is an overhead in starting the MIDlet on the mobile phone, caused by the execution of the DRM algorithm. This depends on the complexity of the DRM solution.

5. Some Sample DRM Solutions

To test the framework described in the previous chapter, several DRM solutions have been implemented. They have different trade-offs. The first solution that will be presented does not require the mobile phone to connect to the server in order to obtain a license, but is relatively easy to bypass. The other solutions are harder to break, but require extra connections to obtain and verify the application license. The final solution works only on mobile phones supporting the Wireless Messaging API (WMA) [13]. These are three examples of DRM techniques for Java applications that can be deployed with the generic framework described in this paper.

5.1 Connectionless Approach

The first solution is transparent to the user and requires no support from the network other than the instrumentation phase during the download of the MIDlet application. Unfortunately, it is somewhat weak. This solution makes the following assumptions:

- After the MIDlet being downloaded to a mobile phone, the user will execute it for the first time within a brief period of time.
- During that timeframe, the user will not try to copy the MIDlet application to other devices.
- The clock of the user's mobile phone is set to the current time and the user will not try to change it.

The timeframe mentioned in the first and second assumptions is the validity period of the MIDlet application. This should be a period of a few hours. When the MIDlet is being instrumented, a timestamp is placed inside the DRM classes. In the first execution at the mobile phone, the MIDlet checks the time on the phone and verifies if it is still inside the validity period. If so, it makes an entry in its PRS indicating that the verification was successful, and executes the application. Otherwise, it registers an unsuccessful verification in its PRS and aborts the application. The next time it executes, it will check its PRS and execute only if it finds the record for successful verification.

Advantages

The main advantages are simplicity and transparency. Honest users, those that do not try to make illegal copies, will not notice the existence of a DRM solution. The size overhead of this solution is small (2k to 3k) and its execution is fast. Another advantage is that no extra communication is required with the Mobile Operator. All the verifications occur within the mobile phone. There is also no need for any server side infrastructure other than the DRM framework.

Disadvantages

The timing assumptions made are very strong, therefore making this DRM solution easy to break. For instance, if there is a significant drift between the clock in the server where the MiDlet is instrumented and the user's mobile phone, the MIDlet application can fail to run for the first time. Also, this solution cannot prevent the user from copying the JAR and installing it on another phone before the validity has expired. Another way to break the protection is by changing the time of the phone clock. Finally, if the user has successfully obtained a license for the MIDlet

application, he can overcome the DRM protection by copying the JAR together with the PRS to another phone.

5.2 Server Licensing Approach

In this approach, when the user requests the download of a J2ME application, the server generates a unique license and a key. The license and a hash of the key are stored in the MIDlet application, which is to be sent to the client's device. The server keeps a copy of the license and the key. When the MIDlet application is first executed the following will happen:

1. The MIDlet establishes an HTTP connection with the server and sends the license.
2. The server checks if the received license is valid and answers accordingly. If the license is valid, it sends also the key.
3. If the phone receives the key, it compares the hash of key with the hash stored in the MIDlet. If they are equal, the MIDlet executes.

The purpose of the key is to prevent a malicious user from sending a forged HTTP packet to validate the license. If the server simply sends the same `Ok` message to all MIDlets, it would be easy to forge this message. By using a key, each confirmation is unique. Furthermore, the user has no way of guessing which packet the MIDlet is expecting from its content, since only the hash of the key is stored[6].

This solution requires the presence of a license server, with the following interfaces:

- **With the Provisioning Server:** A standard API interface, allowing the provisioning server to create a license and the corresponding key for a download request.
- **With the client's terminal:** An interface to be used by the client's terminal to validate the license. Internally, the server must be able to generate licenses and keys. It should keep track of them until the client connects to authenticate itself.

Advantages

This solution is stronger than the connectionless approach. An untampered MIDlet can only execute after receiving the corresponding key that is maintained in the server. Therefore, it must validate its license before executing. This solution can also be easily adapted to support super-distribution of MIDlet applications. For this to be achieved, the server must allow the same

[6] One property of the hash algorithms is that it is not feasible to guess the original message from the hash.

license to be validated more than once and must be able to bill the cost of the J2ME application to the owner of the mobile phone that is requesting the license.

Disadvantages

The main disadvantage is that the *honest* user will notice the DRM mechanism. The MIDlet will have a significant delay in the first execution while it establishes an HTTP connection. Also, this connection will probably have to be paid by the user. Finally, the user must have network coverage to install the application. At the server side, it is also necessary to maintain a license server with a full history of users and downloaded applications.

5.3 SMS-based Approach

This approach is similar to the Server Licensing Approach described in Section 5.2. The main difference is that SMS messages are used to validate the MIDlet. This also requires a slightly different license server, which instead of having a Web interface must have an SMS interface. Apart from that, the core functions are similar.

Advantages

Using SMS messages has some advantages over HTTP. First, the license can be sent in a free SMS message, which is more desirable for the end-user. Second, it is easier to support super-distribution, by using MT (Mobile Terminated) messages to charge for the use of the J2ME application.

Disadvantages

This solution has the same disadvantages as the Server Licensing Approach. The time it takes for the registration can also be noticeable, as SMS messages may take a while to reach its destination. Finally, the terminal must support the Messaging API for sending SMS, which is an optional package of MIDP 2.0. Currently, most terminals are still using MIDP 1.0 and even on MIDP 2.0 terminal, the availability of the Messaging API is not always certain, since it is an optional package.

5.4 Analysis of the DRM Solutions

The DRM solutions presented in this chapter are only a sample of what is possible to do using our DRM framework. The development of these three solutions allowed us to validate the advantages and disadvantages of the framework. The interface provided by the framework proved to be adequate: it abstracts the developer from the low-level details – like instrumentation, JAR re-packaging, manifest updating, pre-verification, and so on. It also proved to be easy to use. The interfaces and classes that had to be implemented are well identified and defined, making it easy for the developer. Also, the possibility of bundling external classes with the DRM solution proved to be very important, as it allowed the solutions to be developed using standard coding practices, like splitting the code across multiple classes, using inheritance, and previously developed packages, among others. All the solutions described in this chapter were tested with several off-the-shelf J2ME applications and they have proved to be effective in all those cases.

6. Conclusion

This paper presented a DRM framework to protect J2ME applications. This development was motivated by the limitations of the J2ME platform that do not provide any specific support for DRM techniques. The current specifications (MIDP 1.0 and MIDP 2.0) does not offer as well any support for the standard tools that are usually used to implement copyright protection schemes, like encryption, key management or unique identification of the device.

Our solution tries to overcome this limitation and the platform we have presented in this paper provides flexible support to deploy different DRM solutions. The framework works by instrumenting MIDlet applications with copy-protection code. This can be done to any Java application in its packaged format (JAR), without having access to the source code. This makes the DRM framework transparent to the Content Providers. The framework is also extensible, allowing different DRM solutions to be developed and deployed.

This paper also presented three particular DRM solutions, each one with different trade-offs. One is easy to break, but has very few requirements. The two others are stronger, but they require more resources from the mobile phone and the network. In the future, other DRM solutions will be implemented, exploiting the available resources on each type of mobile handset.

References

[1] J. Gosling, B. Joy, G. Steele, and G. Bracha, "The Java Language Specification". Boston, Massachusetts, EUA:Addison-Wesley, 2000.

[2] Q. L. Reihaneh, R. Safavi-Naini, and N. Sheppard, "Digital Rights Management for Content Distribution," in *Proc of Australasian Information Security Workshop*, (Adelaide, Australia), 2003.

[3] Sun Microsystems Inc., "Connected Limited Device Configuration (CLDC) Specification, V1.1 (JSR-139)." Available at: `http://jcp.org/aboutJava/communityprocess/final/jsr139`.

[4] Sun Microsystems Inc., "Mobile Information Device Profile (MIDP) Specification, V1.0 (JSR-37)." Available at: `http://jcp.org/aboutJava/communityprocess/final/jsr037`.

[5] Sun Microsystems Inc., "Over-The-Air User Initiated Provisioning Recommended Practice." Available at: `http://java.sun.com/j2me/docs/`.

[6] Open Mobile Alliance, "Digital Rights Management Version 1.0, Specification." Available at: `http://www.openmobilealliance.org/documents.asp`.

[7] Nokia, "Nokia's White Paper on DRM." Available at `https://secure.forum.nokia.com/main/1,,040,00.html?fsrParam=1%2D3&fileI%D=2894`.

[8] DRMSecure, "DMDMobile." Available at `http://www.dmdsecure.com/DMDmobile.htm`.

[9] Secure Digital Container (AG), "SDC Java DRM." Available at `http://www.digicont.ch/sdc_java_drm/core_technology/index.html`.

[10] MacroVision™, "How MacroSafe™Works." Available at `http://www.macrovision.com/solutions/video/drm/overview.php3`.

[11] S. Chiba. "Javassist – a reflection-based programming wizard for Java". In Proc. Of OOPSLA'98 Workshop on Reflective Programming in C++ and Java, October 98.

[12] M. Dahm, "Byte code engineering," in *Java-Informations-Tage*, (Düsseldorf, Germany), pp. 267–277, Sept. 1999.

[13] Sun Microsystems Inc., "Wireless Messaging API (WMA) Specification, V1.0 (JSR-120)." Available at: `http://jcp.org/aboutJava/communityprocess/final/jsr120/`.