

An Implementation Experience of Domain Management in Marlin

Sye Loong Keoh
Dept. of Information & System Security
Philips Research Europe
High Tech Campus 34, 5656 AE, Eindhoven
The Netherlands
sye.loong.keoh@philips.com

Koen Vrieling
Dept. of Information & System Security
Philips Research Europe
High Tech Campus 34, 5656 AE, Eindhoven
The Netherlands
koen.vrieling@philips.com

ABSTRACT

Digital Rights Management (DRM) is used to protect copyrighted content from unauthorized use. However, this has taken away the jurisdiction of the consumers over their purchased content as they can no longer freely access the content at any place using any device; given that the license authorizing the consumption of content is typically bound to a particular device. Domain Management provides the flexibility to the consumers to manage their purchased content as they are given the rights to govern their own domain membership. Consumers can dynamically add and remove devices from the domain subject to a domain policy that complies with the policy set by content owners or service providers. We share our implementation experience of domain management using a DRM technology called *Marlin*. We perceive this work as the first local domain management implementation in a real life practical DRM System.

Categories and Subject Descriptors

D.2.0 [Software Engineering]: General—*protection mechanisms, standards*

General Terms

Security, Standardization

Keywords

Domain Management, Digital Rights Management

1. INTRODUCTION

The arrival of digital era has enabled the ease of information dissemination. With just a click away on the web, the Internet surfers are able to obtain information such as breaking news around the world, friend's current geographical location [3], multimedia content, and access to scientific publications, etc. Rights management is used to specify

and enforce the usage rules for consuming such digital information. For example, location information is explicitly shared with entities we *trust*, rendering BBC news/videos on *iPlayer* is constrained by *geographical* region, i.e., only those living in the UK can play the videos, music tracks and movies in digital form must be *purchased* before they can be played and rendered on devices. Thus, rights management is not only about constraining how digital information is obtained, but also constraining the dissemination of information further such as copying, transferring and sharing with others.

Digital Rights Management (DRM) is introduced to enable content owners to protect their content from unauthorized use and it enables the persistent governance of content by consumers and other participants in the content distribution value chain [6]. The content is typically encrypted and access to the content is governed by a policy (also known as a *license*) set by the content owner who specifies the rules and the required credentials under which the decryption key can be obtained. The integrity of the policy/license must be preserved and the policy/license must be evaluated/executed in a safe and trusted environment. In this case, the consumers do not have the jurisdiction over the specification and enforcement of policies or licenses. Thus, they lose the control over their purchased content as they can no longer freely access the content at any place using any device.

In this paper, we describe an approach to provide consumers with the flexibility of managing access to their content using the concept of *domain management* in Marlin DRM system [13]. This is built on our previous work on domain [12, 10]. By devolving the domain management locally to a home network, this enables the consumers to manage their own domain membership. Devices can be dynamically added to and remove from the domain provided that the domain membership policy is not violated, and that the content owner has sufficient trust on the local domain manager to enforce the policies agreed *a priori*. We share our implementation experience of local domain management in Marlin and discuss its associated complexities as we perceive this work as the first local domain management implementation in a real life practical DRM System.

This paper is structured as follows: Section 2 outlines related work in the current literature and various standardization efforts. We briefly introduce Marlin technology as the foundation for a flexible DRM system [11] in Section 3. Section 4 presents the concept of domain management in the context of Marlin, while Section 5 describes the im-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DRM'09, November 9, 2009, Chicago, Illinois, USA.

Copyright 2009 ACM 978-1-60558-779-0/09/11 ...\$10.00.

plementation details. In Section 6, we further discuss the implementation issues and the market adoption of Marlin DRM System. Finally we conclude the paper with future work in Section 7.

2. RELATED WORK

Open Mobile Alliance DRM (OMA-DRM) 2.0 [15] aims at protecting audio and visual content sent to mobile phones. The media object is encrypted using a Content Encryption Key (CEK), while there is a Rights Issuer (RI) that is responsible for defining the usage rights using the Rights Expression Language (REI): Open Digital Rights Language (ODRL) [9] for the media. The usage rights are then packaged together with the CEK and bound to the content. The bindings can be associated with either a device or an Authorized Domain (AD). An AD is a named set of devices, to which content may be bound. Access to the content is then granted only to the device that can prove to be a member of the AD (using certain cryptographic techniques). In OMA-DRM, the management of AD is centralized, i.e., the RI keeps a record of all the devices in the user's domain. This poses a limitation in that the user must be able to contact the RI in order to add new devices into his AD by means of a direct online connection or via another OMA device that is connected to the Internet. Furthermore, media objects from different ADs of distinct Service Providers cannot be shared even though the ADs are associated with the same user.

Vasanta *et al.* extended the OMA-DRM domain management by devolving the management of domains to a Domain Manager known as *Heinmdall* [16] that acts as a broker between devices in an Authorized Domain (AD) and any content providers from which the content for the domain can be sourced. *Heinmdall* can aggregate all the ADs from different service providers by being a member of all the ADs. This allows *Heinmdall* to have access to all domain keys and then securely convey them to the user's devices. Consequently, the user's devices would be able to access content bound to all the ADs even though they are issued by different Rights Issuers. However, this adds an extra middleware layer between the service providers and the consumer devices instead of a fully integrated solution that fits well with the OMA-DRM infrastructure.

Similar to OMA-DRM 2.0, Digital Video Broadcast — Content Protection Copy Management (DVB-CPCM) [8] uses the concept of Authorized Domain (AD) that groups devices and replaces the direct binding between content and devices. Thus, content can be bound to an AD instead of an individual device and this enables the content to be rendered on domain devices. A device can only be a member of one AD at any time and the content is constrained to the AD by which it is acquired and will not play on a device belonging to a different AD. However, to allow for more flexibility, a device can be re-assigned to another AD for the purpose of consumption of content assigned to that domain during which time it cannot access content which was bound to its original AD. There is no limit to the number of times a device may move between ADs as long as the Content-to-Authorized Domain binding is maintained. Content could also be moved “outside” the AD if allowed by the rights owner. However, the domain concept has not been realized with a concrete implementation.

Apple FairPlay is a special instance of the device-based AD concept where a user account with an online service is

required. Content can be bound to an AD in which a limited number of PCs and unlimited number of tethered devices such as iPods can be connected. The backend systems of the online service tracks the binding of content to the user account. Adding new devices and removing existing devices from the ADs would require the respective registration and deregistration step with the service provider. This DRM model is only restricted to Apple devices and it does not interoperate with other devices in the market.

Given that there are so many DRM technologies adopted in the market, and they typically do not interoperate with each other. The non-interoperability between different DRM ecosystems is caused by the differences in the data structures for encoding content, and that they have proprietary authorization and enforcement infrastructure. Coral Consortium [1] recognizes the need to enable different DRM technologies to interoperate with each other and hence proposing an additional layer on top of existing proprietary DRM ecosystems and providing some services to them. Concisely, the Coral approach is to standardize authorization outside of and independent of any DRM system, and provide mechanisms to translate authorization into enforcement relying on native DRM technology [7].

Imad proposed to manage domain membership based on location services [5], thus ensuring that devices, when joining a consumer domain, are located in physical proximity to the domain registered addresses. This restricts domain membership to devices in predefined geographical locations, helping to ensure that a single consumer owns and manages each domain. The author believes that this can curb illegal admission of remote devices into the domain to prevent illegal sharing of content. As a result, this increases the trustworthiness of the domain manager as only the eligible devices can be admitted into the domain.

Other DRM and copy protection systems such as Digital Transmission Content Protection (DTCP) [2] does not support the notion of *domain management* but focusing on the protection of illegal duplication of content during digital transfer. Windows Media Digital Rights Management (WM DRM) is a platform for the protection and secure delivery of content for playback on computers, portable devices and network devices. It is also a device-based DRM system in the sense that licenses are non-transferable and are bound to the rendering device, unless specified otherwise in the license.

3. MARLIN TECHNOLOGY

Marlin [13] is an initiative started by Intertrust, Philips, Samsung, Panasonic and Sony to develop open standards for interoperable DRM technology. It specifies technologies for building copy protection and DRM into consumer devices and services. The Marlin architecture is based on a base technology framework called *Octopus* which is detailed below.

3.1 Octopus

Octopus defines a set of objects, used to represent different elements and entities that comprise a DRM system, the relationships between these objects and ways of using them in computation. Entities such as devices, groups, user accounts, and subscriptions in the Marlin ecosystem are represented as *Octopus Nodes*. Four types of nodes have been defined, namely *Personality node* representing a de-

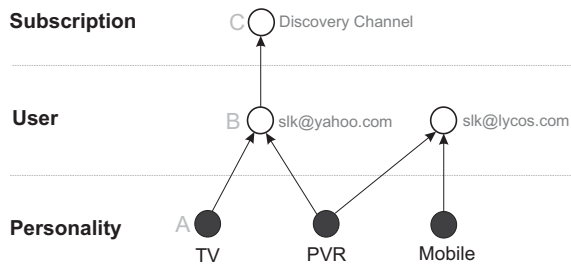


Figure 1: A directed graph formed by Octopus nodes and links

vice, *Domain node* and *User node* representing a group of devices and an user account respectively, as well as *Subscription node* representing a subscription to a service. Figure 1 shows an example of the representation of Octopus nodes in the business context, e.g., Television (TV), Personal Video Recorder (PVR), Mobile Phone (Mobile) are represented by *Personality nodes*, an account with Yahoo and Lycos are represented using *User nodes*, and a subscription to the Discovery Channel is represented as a *Subscription node*. Each node has some cryptographic properties in that it may contain *Sharing Keys* that may be shared with other nodes and *Confidentiality Keys* that must remain secret to the node. Both types of keys can be asymmetric or symmetric.

As shown in Figure 1, a relationship between two nodes is represented by an *Octopus Link* where the vertices are the Node objects. For a given set of nodes and links, we say that *Node C* is reachable from *Node A* if there exists a directed path between *Node A* and *Node C*. In the example shown in Figure 2, there is a path from *Node A* to *Node C* via *Node B*. Cryptographically, a link from *Node A* to *Node B*, i.e., *Link A-B* can be perceived as a placeholder for *Node B*'s private sharing keys. This link is essentially the encryption of *Node B*'s private sharing keys using *Node A*'s public sharing key, this enables *Node A* to obtain the private sharing keys of *Node B* if such a link exists. Similarly, *Link B-C* encapsulates the private sharing keys of *Node C*. Possessing this link enables *Node B* to obtain the private sharing keys of *Node C*. All links are signed and they must be verified before using them to determine a node's reachability.

An *Octopus Control* can be attached to a link object to enforce constraints on the link, e.g., expiry date. The control composes of executable byte code routines that check the constraints and usage rules. It is a signed Octopus object by the content owner, thus ensuring its integrity and trustworthiness. The control is executed in a trusted virtual machine called *Plankton* where the control routine encodes instructions for a stack-based execution engine. The instructions can perform basic computations, read and write in a memory buffer, perform tests and program flow control. Evaluation of usage rules and constraints may require storing of a value in a secure location and that it can be read at a later time. A persistent store where the control routines can read and write data into is provided in Octopus, it is called *SeaShell*. Each data in the SeaShell has metadata fields such as expiration date and ownership information which allows Octopus to manage the lifetime of the data as well as an access policy: By default, control routines can only access data that has the same owner as the author of the Control object from which the byte code was loaded.

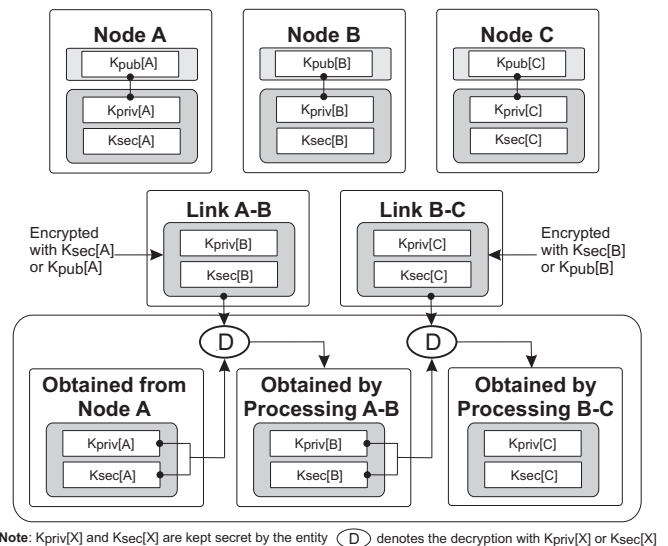


Figure 2: Nodes and Links in Octopus

A *Content* object is encrypted with a *Content Key*, while a license specifies and enforces the rules for obtaining the content key. The license includes four types of Octopus objects, namely *Content Key*, *Protector*, *Controller* and *Control* objects. The relationships between them is illustrated in Figure 3 and the functionality of each object is described below:

- *Content Key* object — contains an encryption key to decrypt the content. Access to the encryption key is only granted to recipients that are authorized to decrypt the content based on the usage rules encoded in the control object. The Content Key object also specifies the cryptosystem(s) that was used to encrypt the content.
- *Protector* object — provides a one-to-one mapping of content with the content key object. This enables the license object to find out the content key that was used to encrypt the content.
- *Controller* object — contains the information that allows Octopus to find out which control governs the use of one or more content key objects.
- *Control* object — represents the usage rules for the content item, it contains the information that allows Octopus to decide whether to permit certain actions on the content when requested by the application interacting with the content.

3.2 Marlin in Broadband Network Service

In an Electronic Sell Through (EST) business model, consumers can purchase multimedia content via the Internet and play/render them on their digital media players. Each device is pre-configured during the manufacturing time with a *Personality node* and the respective credentials such as a digital certificate for interacting with Marlin services. Prior to completing an online purchase, the consumer is required to create a user account with the content provider. In the Marlin ecosystem, a *User node* is issued to the consumer

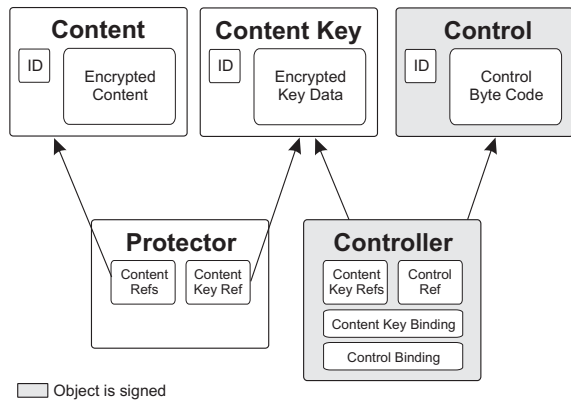


Figure 3: The components of a Marlin license

upon the creation of such an account. Subsequently, the content provider creates an Octopus Link between the device’s *Personality node* to the *User node*. This is achieved by requiring the DRM Client hosted on the device to execute a *Node/Link Acquisition* protocol in which the *Personality node* of the device is sent to the content provider to create such a binding. Octopus nodes and links themselves are essentially self-secured, however the protocol is designed to also ensure that they originates from an authentic source. The delivery of nodes and links is hence secured using web services security [14].

When the purchase has been completed, the DRM Client hosted on the device will start the *License Acquisition* process. The content provider encrypts the purchased content using a content key and then issues a license *bound* to the *User node* of the consumer. This means that the content key is encrypted with the *Public Sharing Key* of the *User node*. Therefore, only the entity which has access to the *Private Sharing Key* of the *User node* can decrypt the content key. Since the consumer has already opened an account with the content provider and obtained a link between its *Personality node* and its *User node*, it will be granted access to the encrypted content. As part of the license, a control object specifying the usage rules for consuming the content is also created. This control typically enforces the link validity and node reachability. In addition to *license binding*, Marlin also supports *license targeting*. When a license is *targeted* to an Octopus node, this requires the control object to check whether or not the targeted Octopus node is “*reachable*” from the device’s *Personality node*. For example, if the license is targeted to the *Subscription node* and when the license is evaluated, the control must ensure that there is a valid path from the *Personality node* to the *Subscription node*, e.g., *Personality node* → *User node* → *Subscription node*. If the license is *targeted* to the *Personality node*, this implies that the purchased content can only be consumed by the device itself, thus Marlin also supports device-based DRM.

4. DOMAIN MANAGEMENT

Domain is used as a placeholder for a set of related devices and this allows the devices to be managed in a more efficient manner. This eases the device management in that policies (whether access control policies, management policies, or privacy policies) can be defined for the domain, hence

a single policy can be specified and enforced by a group of devices of similar functionality instead of having a policy for each individual device. In network management, computing devices are divided into different domains according to their organizational structure, while in a content management system, devices owned by the consumer that have been associated with his user account are typically grouped together as a *User Domain*. A *Domain Manager* or *Registration Service* is needed to manage the membership of the domain and a membership admission policy must be defined to govern admittance to the domain. Note that, the *domain manager* can be deployed in the Internet or locally in a home network. In Marlin, the following membership admission policies are supported:

- *Satisfying the domain policy* — a domain policy specifying the cardinality constraints to restrict the number of members that can be admitted into the domain must be satisfied, e.g., a domain can only have a maximum of five devices at a time.
- *User account registration* — for Internet based domain manager, a consumer can register for a user account online in order to obtain his domain membership. Once the user account has been created, devices belonging to the user can be associated with the user account subject to the satisfaction of the domain policy. In case of Marlin, the devices are linked to the user account by means of Octopus nodes and links.
- *Proximity check* — for a locally deployed domain manager, it can check the proximity of the requesting device to ensure that the device is in close proximity to the domain manager before granting admission to the domain. This is to prevent the consumer from trying to admit remote devices into the domain. For example, the Digital Video Broadcast (DVB) requires the broadcast content to be only accessible to local devices at home. Remote access over the Internet is restricted depending on the control bit of the broadcast signal. Therefore, proximity check can be used to distinguish between local and remote devices.

4.1 Marlin Shared Domain (MSD)

In the current Marlin infrastructure, the *User node* representing a user account is considered as a domain where multiple devices belonging to the user can be added into this domain. This is termed as *User Domain*. As mentioned previously, the *Personality node* is used to represent a device and if a link between the *Personality node* and the *User node* exists, this means that the device is a member of the *User Domain*. As shown in Figure 4, when a consumer has two user accounts with two different service providers, he essentially creates two user domains and all his devices must be linked to both user domains in order to enable his devices to consume content bound to both domains. This has the limitations in that in order to gain access to the content licensed to the user domain, every device must have the capability to go online to the service provider and request to join the user domain. Furthermore, when the service provider goes out of business, the consumer cannot add new devices into the domain anymore.

We introduce the concept of *Shared Domain* in the Marlin ecosystem as an intermediary between the service providers

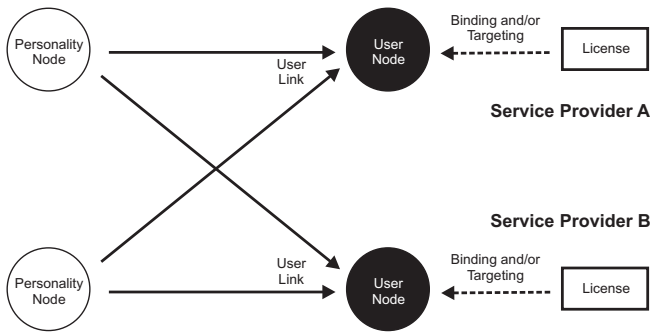


Figure 4: Existing node-link topology in Marlin

and the consumers, taking the advantage of Octopus’s extensibility feature. As illustrated in Figure 5, multiple *user domains* from different service providers can be associated with a *shared domain*, and all devices belonging to the user only need to be linked to the *shared domain* once. This extension preserves the security properties of Octopus in that the node-link relationship remains the same and hence the content key can be obtained by possessing a directed path from the *Personality node* to the *User node* (via the *Domain node* as the intermediary). However, it is assumed in our implementation that the *shared domain* is managed locally by a *domain manager* that the service provider trusts. The service provider trusts that the domain manager would enforce the domain policy truthfully, i.e., it enforces the most restrictive cardinality constraint among the policies defined by the service providers.

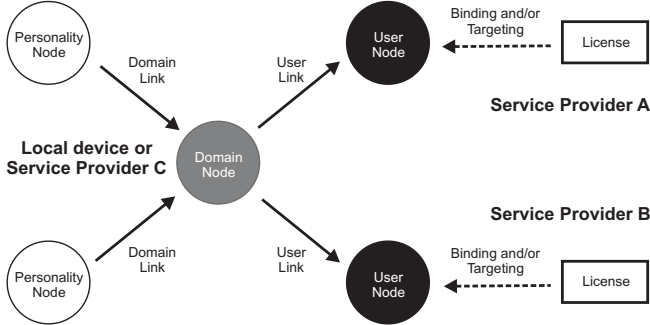


Figure 5: Marlin Shared Domain node-link topology

This greatly improves the user’s experience in the sense that consumers can now manage their own domain membership without relying on the availability of an online connection to communicate with the service providers. New use cases can be supported, a shared domain can be created for a household to enable members of the household to share their content with each other by linking their respective *user domains* to the household *shared domain*. Furthermore, by linking their devices to the *shared domain*, the content can be consumed by any device that are members of the household. As the *domain manager* is located in the home network, devices can be replaced by executing *Marlin de-registration* protocol to invalidate the domain link. Alternatively, an expiring domain link can be issued to the member device when joining the *shared domain*, thus requiring all the member devices to renew their respective domain

link from the local *domain manager* periodically. A device can be easily de-registered by not renewing its domain link.

4.2 MSD Conceptual Model

Figure 6 shows the relationship between various components of the Marlin *Shared Domain*. Three managing entities are defined, namely *Registration Service*, *Domain Manager*, and *License Service*. Both the *Registration Service* and *License Service* are deployed at the Service Provider’s, while the *Domain Manager* is deployed locally in the home network. The *Registration Service* manages the user accounts and approves the shared domains to be associated with the user account. It must check the domain policy of the shared domain whether it complies with the service provider’s policy before issuing a *User link* between the *Domain node* and the *User node*. On the other hand, the *License Service* is responsible for issuing Marlin license to the consumers. It determines the license binding and targeting of the encrypted content, a license can be *bound* to the *User node* or *Domain node*, in which the content key is encrypted with the *User node*’s public sharing key and *Domain node*’s public sharing key respectively. The control in the license determines the usage controls and policies for releasing the content key. Lastly, the local *Domain Manager* provides similar functionalities to the *Registration Service* in that it manages the membership of the shared domain, enforces membership admission policy, (i.e., checking domain policy and determine device’s proximity) and issues *Domain nodes* and *Domain links* to member devices.

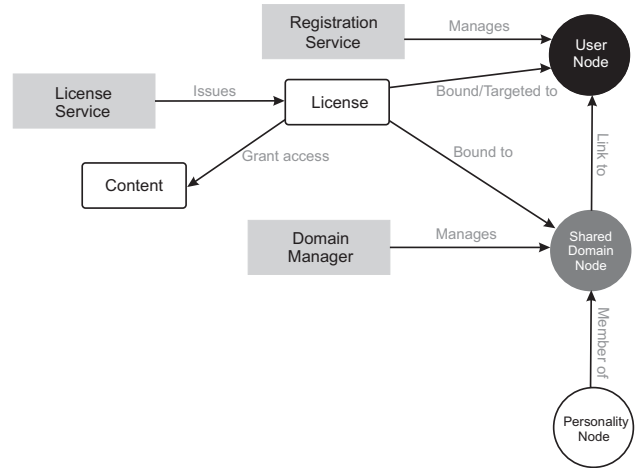


Figure 6: Marlin Shared Domain Conceptual Model

The following describes various license management schemes to cater for different types of requirements:

4.2.1 User Bound and Targeted

This is the basic license management scheme in that the license is *bound* and *targeted* to the *User Node*. Hence, the content is accessible to devices that are a member of the *shared domain* that has been associated with the *user domain*. As shown in Figure 7, this license scheme enables the consumers to share content obtained from multiple service providers with all devices that are members of the *shared domain*. *Alice* and *Bob* can share their content and render on the common *TV* and *PVR* in the living room in which both devices are members of the *Smith’s Family Shared Domain*.

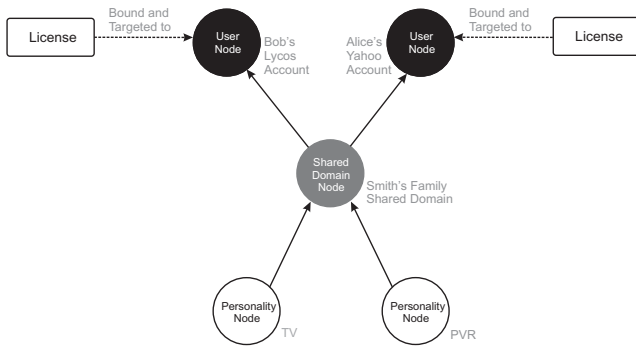


Figure 7: Usage Model of binding and targeting the license to the User node

4.2.2 Domain Bound and User Targeted

This is termed as releasable management scheme, where the *License Service* can selectively release access of content to selective shared domains. In this case, the license is *targeted* to the *User node* but it is *bound* to the *Domain node* the content is released to. There could be multiple shared domains being associated with a *User Domain* as illustrated in Figure 8, the service provider can restrict the content access to a limited number of shared domains, for example the service provider only allows one shared domain to be associated with a user account. As shown in Figure 8, *Alice* has a user account with *Yahoo*, and since she stays with her parents, she links it to her family's shared domain, i.e., *Smith Family Shared Domain*. However, after she finishes her high school, she moves out of her parent's house and goes to college to further study. In her college hostel, she sets up a *Party Central shared domain* with her classmates and links her *Yahoo* account to *Party Central shared domain* too. This does not relinquish her *Yahoo* account with her parent's shared domain. When she purchases a pre-release movie from the Internet provided by *Yahoo*, the content provider can restrict the content access to only one shared domain instead of all shared domains that have been associated with her *Yahoo* account. As *Alice* is away from her parent's home, she prefers to share the content with her classmates, thus a license that is *targeted* to *Alice's Yahoo User node* but *bound* to the *Party Central shared domain* is issued. This would forbid the members of *Smith Family Shared Domain* to access the pre-release movie, while only enabling *Alice's* classmates who are members of the *Party central shared domain* to consume the movie that *Alice* has purchased.

4.2.3 Domain Bound Only

It is also possible for the *License Service* to impose further cardinality constraints such that only one device can consume the content at any time, even though there are more than one device in the shared domain. In this case, the license is *bound* to the *Domain node*, and it is not *targeted* to any *Octopus nodes*. However, this requires the *License Service* to send a trusted agent to write a secure state variable into the device's seashell database when acquiring the license. Thus, in order to consume the content, the control object in the license must verify the state of the variable in the device's seashell database and only the device with the valid state variable is granted access to the content key

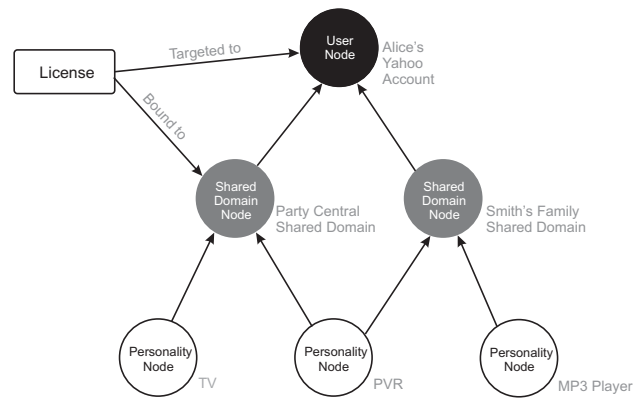


Figure 8: Usage Model of binding the license to the Domain node, but targeting the license to the User node

by the control object in the license. Other clients when attempting to consume the content will be denied access as their devices do not contain the state variable even though they are members of the shared domain.

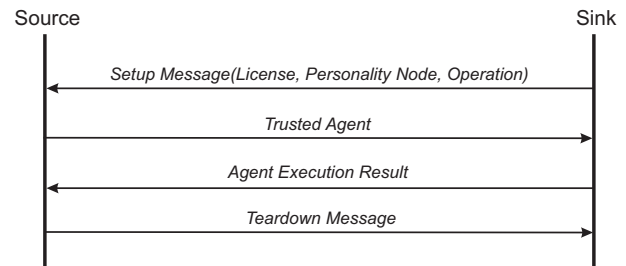


Figure 9: License Transfer Protocol

This type of license can be *transferred* to another device in the *shared domain*, e.g., transferring the rights from the TV at home to the portable video player (PVR) so that the consumer can play the content while she is traveling. The *License Transfer Protocol* (LTP) is triggered where a trusted control agent (embedded in the license) in the original device (source) is invoked in order to move the state variable to the device requesting for license transfer (sink). As shown in Figure 9, the sink device initiates the LTP by sending a setup message to the source device indicating the transfer operation *move*. The setup message also consists of the license to be transferred, the data structure containing the public Octopus *Personality Node* of the sink device. Upon receipt of the message, the source device verifies the signature of the request and invokes the *Control.Actions.Transfer.Perform()* action on the control object of the license. This would trigger an obligation on the source device to send a *trusted agent* to the sink device. The agent is executed on the sink device, it must ensure that the sink device is a member of the shared domain, subsequently a secure state variable is created on the sink device's seashell database. The result of the agent execution is then sent back to the source device. This serves as a confirmation that the secure variable has been created successfully in the sink device, and the source device must delete the corresponding seashell variable. Lastly, a teardown message is sent from the source

device to the sink device, confirming the successful transfer of the license. In the case that license transfer is only permitted between devices that are in close proximity with each other, the source device must perform proximity check on the sink device in addition to the transfer of the secure state variable.

5. IMPLEMENTATION

5.1 Jetski

There is a Java reference implementation of Marlin DRM technology called *Jetski* which contains the Marlin Server and Marlin client. We have extended *Jetski* to support *Marlin Shared Domain*. Figure 10 shows the architectural components of the *Marlin Shared Domain* implementation. The Marlin Server is typically operated by a service provider and it consists of the *Registration Service* and *License Service*. The *Registration Service* serves the roles of issuing nodes and links as well as invalidating links. This service has been extended to support linking *Domain nodes* to *User nodes*. It is also the service provider’s responsibility to add additional metadata called “*ContextTag*” to the Octopus objects it manages, i.e., *User nodes*, *User links*, and licenses. This provides an efficient way of grouping the related nodes and links together, so that the nodes and links bundle can be distributed among the members of the shared domain in the local home network.

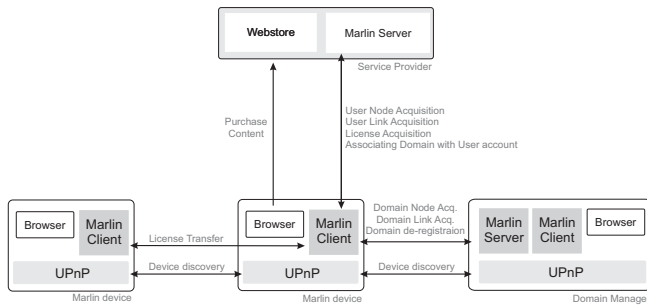


Figure 10: Marlin Shared Domain architectural overview

The existing implementation of *Registration Service* is modified so that it can also be deployed on a local device in a home network as a *Domain Manager*. As the issuance of *Domain nodes* and *Domain links* are the same as the issuance of *User nodes* and *User links*, existing protocols can be used to facilitate local domain management. However, new node attributes are added to the *Domain node* in that it contains a *Domain ID* as its identifier and a reference to the *Domain Manager*, where a *Domain UPnP UUID* is used to uniquely identify the *Domain Manager* in UPnP context. Figure 11 shows an example of node and link attributes of the *Marlin Shared Domain* node and link topology.

5.2 Local Management

Discovery of devices in a local area network is based on UPnP [4]. New resource attributes are added to the Content Directory Service (CDS) to enable Marlin UPnP devices to discover not only the content, but also the corresponding license and the required nodes and links bundle (as the *User nodes and links* in association with a user account only need

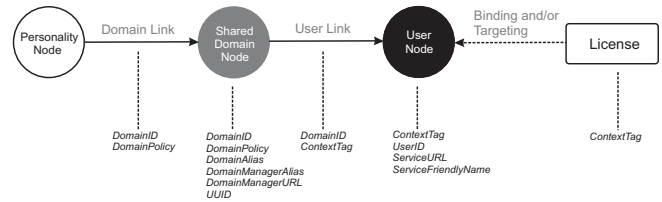


Figure 11: Marlin Shared Domain node-link topology

to be obtained once from the Service Provider, they can be shared with other devices in the shared domain). Similarly, when uploading a content to the CDS, the client invokes the *CDS.CreateObject()* method which returns URLs that enable the client to upload the content, license, as well as node and links bundle to the CDS using HTTP POST.

Several non-standard UPnP methods have also been introduced to facilitate local management of *Marlin Shared Domain* as follows:

- *GetMarlinNodeInfo()* — enables the Digital Media Controller (DMC) to check the compliancy of the Digital Media Renderer (DMR) with Marlin before downloading the content onto the DMR for rendering. This is necessary to provide a good indication to the user that the rendering device is eligible to consume the content. When this method is invoked by the DMC on the DMR, the Marlin <UId> of the DMR is returned and this allows the DMC to check whether or not the Marlin license is targeted/bound to the DMR.
- *GetMarlinActionToken()* — enables a UPnP device in the local network to obtain an “*Action Token*” from the local *Domain Manager* in order to join the shared domain. The device must first discover the *Domain Manager* via UPnP before it requests to join the domain. An *Action Token* contain a set of processing instructions the device must perform in order to obtain the node, links, or license for a chosen content.
- *PutMarlinActionToken()* — enables the *Domain Manager* to pro-actively send an action token to the newly discovered device to join a shared domain.
- *GetMarlinNodeLinks()* — enables UPnP devices to obtain the related nodes and links bundle that carries the same “*ContextTag*” from the *Domain Manager*, Digital Media Server (DMS), and other devices in the local area network. When this method is invoked, all nodes and links that has the “*ContextTag*” attribute of which the value is equal to the value of the contextTag attribute in the request are returned. As a result, the device which triggers this method can look at the attributes of the *Domain nodes* returned by this method call and determine the shared domain to which it has to join in order to consume a content if it has not already done so. Furthermore, devices that have just been added to the shared domain can obtain the *User nodes and User Links* from other devices in the shared domain in order to consume content that have been purchased in the past without needing to contact the service providers.

- *PutMarlinNodeLinks()* — enables the *Domain Manager* to pro-actively push the nodes and links bundle to devices that have just joined the shared domain, thus ensuring that they have the required nodes and links to play content stored in the DMS. Additionally, when the nodes and links have been updated, the *Domain Manager* may use this function to update the nodes and links in the devices.

5.3 Testing in FishNet

The Marlin Conformance Test Suite called FishNet is a test environment that ensures that the implementation of Marlin Architecture complies with the specifications. Test cases have been developed to test Marlin Clients and Servers to ensure that the correct nodes, links and licenses are issued by the Server and Domain Manager. We also ensure that the protocols are executed correctly. Furthermore, different types of licenses are evaluated and tested based on the new Marlin Shared Domain node-link topology in order to ensure that existing licenses are still valid and they are not affected.

6. DISCUSSION

Digital Rights Management (DRM) requires tremendous supports from the consumer electronics (CE) device manufacturers, content providers, telecommunication companies, etc to work together in order to make it happen. Standardization activities such as Coral, Marlin, DVB, OMA, etc play an important role to ensure that the appropriate DRM technologies are adopted in the content distribution value chain. By standardizing the DRM technologies, this enables various devices to interoperate with each other, thus a critical mass of supported devices can be created. Content providers which adopt the standardized technologies can also benefit in that their content can be protected from illegal copying and piracy, while only accessible to trusted devices that comply with the standards. However, a challenge remains as there are a wide spectrum of DRM technologies that do not interoperate with each other, getting them to work seamlessly with each other would pose great difficulties. This is further exacerbated by the unwillingness of various industry players to participate in standardization activities, but would rather keep to their own business models and use their own content protection technologies. We believe that this is an ongoing effort to unite all parties in the value chain to work together in order to create a healthy ecosystem and instil an awareness regarding the correct use of copyrighted content.

A trust issue arises when we proposed to devolve the domain management functionality to a local device under the governance of consumers. The content providers are worried that the *Domain Manager* would not be trusted enough to enforce the domain policy efficiently, thus maliciously admitting significantly more devices than permitted into the shared domain. However, we argue that if we can trust the Marlin client on the device to not maliciously release the content key to ineligible devices and enforces the usage rules governed by the license, it is reasonable to manufacture a device that can serve as a trusted *Domain Manager* that enforces the domain policy securely. It is undoubtedly that a secure execution platform that is tamper-resistant must be used in order to ensure the integrity of the domain policy and prevent malicious tampering of the domain management. Additionally, a revocation mechanism has to be

put in place so that *Domain Manager* that has been detected to be compromised must be revoked. Consequently, the *Service Providers* would not issue new *User Links* to the shared domains administered by the compromised *Domain Manager*.

On the other hand, different service providers may have different domain policies that need to be enforced, and it is the responsibility of the *Domain Manager* to choose a policy that satisfies all the service providers. Thus, either all service providers agree on a common policy to be enforced which is rather infeasible, or the service providers trusts the *Domain Manager* to only enforce the most restrictive domain policy among the service providers. Consider that *Service Provider A* has a domain policy that specifies a cardinality constraint of at most four devices can consume its content, while *Service Provider B*'s domain policy enforces a constraint of at most six devices. If content from both service providers are made available to the shared domain managed by the *Domain Manager*, enforcing *Service Provider B*'s domain policy would violate *Service Provider A*'s domain policy. Thus, we advocate that the *Domain Manager* should enforce the most restrictive policy to allow for the sharing of content purchased from different service providers.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a domain management implementation in Marlin to relax the strict control of DRM on the content consumption by enabling consumers to manage their own domain membership. Consequently, the license is no longer bound to a single device, but a group of devices that can consume the content freely without the need to contact the service provider to obtain an authorization each time a device is added to the domain. We extended Marlin DRM technology by taking advantage of Octopus's extensibility feature and this preserves the security properties of Octopus.

The second contribution of this paper is to share our implementation experience of domain management in Marlin which we perceive as the first implementation of local domain management in a real world practical DRM system. Various technologies are integrated together, for example web services are used to secure the interaction between components of Marlin components, UPnP are used as discovery of devices and *Domain Manager* in a local network environment. Through this implementation, we have demonstrated that it is feasible to deploy DRM to cater for copyrighted content protection, while keeping in mind the needs of consumers to have the ability to manage their purchased content.

Future works include investigating the possibility of protecting digital broadcast using Marlin technology and how it can be implemented in combination with domain management. Further studies on the requirement of digital broadcast is needed in order to devise a suitable protection scheme.

8. ACKNOWLEDGMENTS

We gratefully acknowledge our colleagues, Marcel van Nieuwenhoven, Maaike Gerritsen, Kees Wouters, Henk van Gestel from Philips Applied Technologies (AppTech) and Gerard Lokhoff from Philips Intellectual Properties and Standards (IP&S) for valuable discussions and implementation efforts.

9. REFERENCES

- [1] Coral Consortium, <http://www.coral-interop.org>.
- [2] Digital Transmission Content Protection (DTCP), <http://www.dtcp.com/>.
- [3] Google Latitude, <http://www.google.com/latitude/>.
- [4] UPnP Device Architecture, <http://www.upnp.org/>.
- [5] I. Abbadi. Authorised domain management using location based services. In *Mobility '07: Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, pages 280–287, New York, NY, USA, 2007. ACM.
- [6] G. Boccon-Gibod, J. Boeuf, and J. Lacy. Octopus: An Application Independent DRM Toolkit. In *6th IEEE Conference on Consumer Communications and Networking CCNC*, pages 1–7. IEEE, 2009.
- [7] G. Doërr and T. Kalker. Design rules for interoperable domains: controlling content dilution and content sharing. In *DRM '08: Proceedings of the 8th ACM workshop on Digital rights management*, pages 39–50, New York, NY, USA, 2008. ACM.
- [8] DVB. Digital Video Broadcasting - Home, <http://www.dvb.org/>.
- [9] R. Iannella. Open Digital Rights Language (ORDL) Version 1.1, <http://www.w3.org/TR/ordl/>, 2002.
- [10] F. Kamperman, L. Szostek, and W. Baks. Marlin Common Domain: Authorized Domains in Marlin technology. In *4th IEEE Conference on Consumer Communications and Networking CCNC*, pages 935–939. IEEE, 2007.
- [11] R. H. Koenen, J. Lacy, M. MacKay, and S. Mitchell. The Long March to Interoperable Digital Rights Management. *Proceedings of the IEEE*, 92(6):883–897, 2004.
- [12] P. Koster, F. Kamperman, P. Lenoir, and K. Vrieling. Identity-based drm: Personal entertainment domain. *Transactions on Data Hiding and Multimedia Security*, 4300:104–122, 2006.
- [13] MDC. Marlin Architecture Overview, <http://www.marlin-community.com/>, 2006.
- [14] MDC. The Role of NEMO in Marlin, <http://www.marlin-community.com/>, 2006.
- [15] OMA. DRM Architecture Version 2.0.1, <http://www.openmobilealliance.org/>, 2008.
- [16] H. Vasanta, R. Safavi-Naini, N. P. Sheppard, and J. M. Surminen. Distributed management of oma drm domains. In J.-K. Lee, O. Yi, and M. Yung, editors, *7th International Workshop on Information Security Applications, WISA*, volume 4298 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2006.