The Coral DRM Interoperability Framework

Ton Kalker
Hewlett-Packard
1501 Page Mill Road
Palo Alto, California
Email: ton.kalker@hp.com

Knox Carey
Intertrust Technologies
955 Stewart Drive
Sunnyvale, CA 94085
Email: knox@intertrust.com

Jack Lacy
Intertrust Technologies
955 Stewart Drive
Sunnyvale, CA 94085
Email: lacy@intertrust.com

***Abstract*—The use of Digital Rights Management (DRM) technologies for the enforcement of digital media usage models is currently subject of a heated debate. Consumer organization [1] and national governments [2] complain that DRM technology interferes with basic personal rights, such as the right to make copies for personal use or the right to use content on any platform of choice. This issue has lately gained increased attention by a trend in some European countries to force DRM vendors and online media stores to open up their respective DRM technologies, i.e. make them *interoperable*. In this paper we present a DRM interoperability framework that allows multiple DRM systems to seamlessly work together while at the same time requiring minimal modification to existing DRMs. We believe that our solution, which is supported by many players in the world of digital media, is capable of addressing many of the end-user criticisms of DRM. Moreover it sufficiently flexible to allow new business models to be developed.**

I. Introduction

Lack of interoperability among DRM systems acts as a serious impediment to the development of a robust digital content distribution market, impacting all members of the value chain:

- *Consumers* are put off by content and services that do not work with all of their devices
- *Device Makers* must choose to integrate a single DRM technology, thus limiting their reach, or multiple DRMs, increasing their cost
- *Distributors* must choose DRM systems supported by popular devices, limiting their ability to address a broader set of consumers with different devices
- *Content Providers* see a smaller addressable market due to the fragmented nature of the downstream value chain.

Typical attempts at resolutions to the problems described above have involved standardization around a particular DRM technology and vertical integration of the Distribution and Device business functions. This trend creates a market in which consumers must invest in devices, services, and content unified by their use of a common DRM technology. As a result, consumer choice is limited, the flexibility of Distributors and Device Makers is constrained, and consumers must continue to be aware of DRM when acquiring and using devices.

The solution to the problems described above is DRM interoperability. If DRM systems could be made to work together, consumers could acquire content with the expectation that it will continue to work regardless of the services or content they initially acquire. Device makers could choose the DRM system that best meets their requirements, service providers could sell to a wider variety of devices, and ultimately content providers would benefit from a market with fewer impediments. The topic of DRM interoperability is relatively new and has until recently received little attention in the standardization and academic community. Notable exceptions include the work of the Digital Media Platform [3], the Coral Consortium [4] and MPEG-21 [5]. Some initial academic research can be found in [6] [7] [8] [9] [10] [11].

There are three basic approaches to DRM interoperability: (a) adopt a single standard DRM (b) provide interoperability via bilateral agreements and (c) create a scalable, multilateral, standardized interoperability framework. The standardization approach has worked in some cases, but different, non-interoperable standards have emerged in various environments. The bilateral approach is not scalable and potentially leads to great consumer confusion. Coral adopts the latter approach, with the *explicit goal* of minimizing impact on the existing value chain. The Coral approach is designed to make it easy for existing systems to integrate the Coral interoperability framework without major redesign of technology or business models.

In this paper we describe the basic architectural approach adopted by the Coral specifications, focusing in detail on the application of the core technologies to emerging usage models such as authorized domains. We describe the design of a particular Coral ecosystem, which adds a set of policies to the core technology in order to enable some of these new usage models to be deployed across devices and service providers. Finally, we discuss issues around the deployment of the technology in a practical setting.

II. Coral

The Coral Interoperability Framework (Coral IF) has a layered structure. At the bottom of the Coral IF stack is the Coral Architecture [4], which serves as a toolkit for building interoperable ecosystems. This Section describes the structure of the Coral Architecture, while the next Section introduces the specific ecosystem Ecosystem-A built on top of the Coral Architecture.

A. The Coral Architecture

The Coral Architecture is based on the notion of a *Rights Token* (RT). An RT is a DRM-independent data structure (P, C, U) that asserts that principal P is allowed to access content resource C under the usage model specified by U. For example, P may refer to a device, to a group of devices, a user or a group of users. A principal consists of a namespace specifier and a unique identifier within that namespace. Typically all the identifiers within a namespace are managed by a single entity (for example a store front or a service provider) that is able to guarantee uniqueness.

In most DRMs, the right to use a resource in a particular way is coded in an explicit license with DRM-specific syntax and semantics, which vary widely across DRMs. In other DRMs, content licenses are implicit and hard coded in DRM clients and/or content servers.

In a typical electronic content distribution transaction, there is no distinction between the acquisition of rights and the fulfillment of those rights via a native DRM license. Coral splits these transactions into two phases: acquisition of content rights encoded in a DRM-independent Rights Token, and fulfillment of those rights using a native DRM technology. The latter phase may be repeated multiple times to obtain rights in different DRM systems.

The following example shows how content rights are acquired and fulfilled in typical Coral deployment.

- **Acquisition**: Alex visits his favorite online content store and purchases an item C. As a result, a Rights Token (P, C, U) is created, where the Principal P designates a specific set of devices registered by Alex, and usage model U designates the right to render C for one month.
- **Instantiation**: Alex selects a device δ and requests an instantiation of the Rights Token. The interoperability framework performs the following steps:
  - **DRM verification**: The IF verifies that the device δ uses a DRM that supports usage model U. Specifically, the DRM must implement a secure clock so that access to C can expire at the end of the month.
  - **Principal resolution**: The IF verifies that device δ is a Member of the set of devices P.
  - **Content resolution**: The IF locates a service S (or device) that has content C available in a format that is compatible with device δ.
  - **Rights Token resolution**: The IF casts the principal P, the content C and the usage model U into namespaces meaningful for the selected DRM and service S.
  - **License creation**: The IF requests that S create a native DRM license corresponding to the resolved Rights Token. Service S fulfills this request using a DRM specific protocol. It may also assist in the acquisition of the associated content.

As a result of this process, a DRM-independent Rights Token is converted into a DRM-specific license. It is important to note that all of the steps described above are typically transparent to the end-user — the user should only be aware of the fact that purchase of content leads to executable content on his selected rendering device few minutes later. Moreover, the indirection through S allows the device and its supported DRM to be completely unaware of the Coral IF.

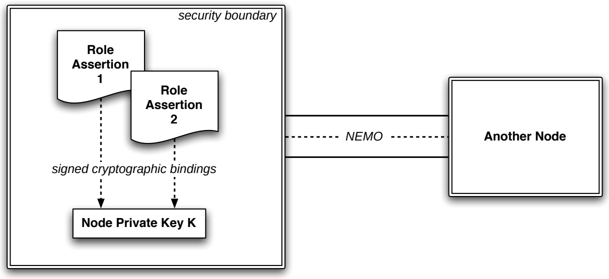

Fig. 1. Coral: Nodes, Roles and NEMO

B. Coral Components

The Coral Architecture is created from three basic building blocks: (1) secure computational units, (2) a secure messaging protocol and (3) secure assertions authorizing specific behaviors. The Coral secure computational units are referred to as *nodes* and they are defined by a security boundary that shields all internal computations and data from the external environment. Each node is associated with a certified private/public key pair. The private key is internal to the node (and never revealed), whereas the public key facilitates public identification for the associated node. Nodes communicate with each other via the NEMO [12] secure messaging framework. The NEMO framework provides for point-to-point message confidentiality and integrity between two communicating nodes using standard web service technologies.

The final component in Coral is the notion of a *role*. A role is essentially a *signed assertion* that a specified node is certified to exhibit certain behavior associated with that role. Coral roles define a baseline set of behaviors that are refined by further specifications to create a fully interoperable environment. The Coral Architecture is best considered as a toolkit for building such environments, referred to as Coral *ecosystems*. In this Section we will give an overview of the major roles in Coral. The next section provides an overview of these roles as they are applied to an actual ecosystem.

C. Coral Roles

There are three main classes of roles: (1) roles that handle Rights Tokens, (2) roles that resolve principals and (3) roles that handle content resources.

1) Handling Rights Tokens: The most important role in this class is the *Rights Mediator* role, which mediates the transfer of Rights Tokens between different systems. It does so by querying repositories (referred to as *Rights Registries*) for existing Rights Tokens and possibly transforming these Rights Tokens as allowed by the rules of the ecosystem. The resulting Rights Tokens are presented to systems called Rights Instantiators for conversion to native DRM licenses. Note that

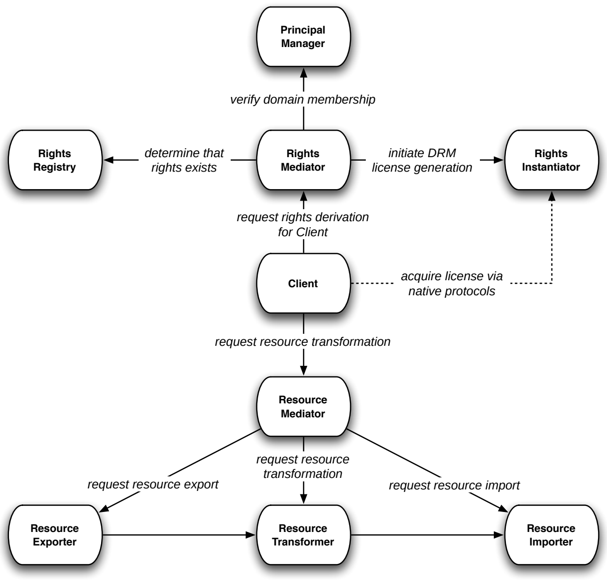

Fig. 2. A Summarized Overview of Roles in Coral

the Right Mediator does not directly query DRM systems for information about issued licenses: Coral does not assume that DRMs are willing and/or able to provide such information to an interoperability framework. This approach allows existing DRM systems to easily participate in Coral-based ecosystems.

2) Handling Principals: The most important role in this class is the *Principal Manager*, which manages relationships between principals within and across namespaces. Two principals are either related by a directed relation of type `<ISA>` or an equivalence relation of type `<IS>`. A relation P_1`<ISA>`P_2 means that principal P_1 may be replaced by the principal P_2 in all circumstances, but not vice versa. A relation P_1`<IS>`P_2 indicates that P_1 and P_2 may be used interchangeably and is in effect equivalent to two `<ISA>` relations. A Principal Manager is responsible for creating principals and relations and querying existing principals and relations. A Principal Managers may also interact with systems that create native DRM representations of the relations it manages. Coral does not assume that DRMs are willing and/or able to be queried for principals and their relations.

It is not necessary in Coral that all principals and relations within a given ecosystem be stored in single central database. The Coral IF enables a distributed approach in which the resolution of names and relations between them are handled by multiple, cooperating systems. This approach is similar to that advocated by the Liberty Alliance[1] [13].

3) Handling Content: The Coral Architecture is mainly concerned with properly handling the rights for content: its central data structure, the Rights Token, is a DRM-independent proof of rights that can be transformed into native DRM licenses. The acquisition of the actual encrypted content, however, is a separate process that is outside of the scope of Coral. However, there are relevant use cases in which the Coral Architecture needs to offer support for content handling, for example, when content acquisition from a networked server is impossible or impractical. In such cases, acquisition of content may involve extracting content from one DRM-protected environment and importing it into another. Security considerations require that content in transit between the protected environments be protected by a Secure Authenticated Channel (SAC) that is trusted by all parties. The Coral Architecture defines a profile of TLS [14] as a default SAC for content transfer, but other SACs are also allowed. The SAC between source and sink is managed by the *Resource Mediator* role. The Resource Mediator may request that a *Resource Exporter* export the content to a *Resource Importer* or to a *Resource Transformer* for format translation. Resource Transformers apply format transformations as needed by the receiving party (e.g. rate or resolution reduction, file format conversion and others). Resource Importers, Exporters and Transformers may be implemented directly by DRM systems, but this is not required. Regardless, export of content from one DRM into another DRM must always be approved by the exporter.

4) Miscellaneous Roles: The Coral Architecture also contains a number of supporting roles. In particular, there are roles for handling authentication and certificates, roles for validating role assertions, roles for updating ecosystem policy variables and roles that help in assessing various kinds of capabilities (device, DRM and others). These roles will not be discussed in this paper. Full details on the Coral specifications can be obtained at [4].

III. A Domain Based Interoperability Framework

The Coral Consortium has built a domain-based Ecosystem (Ecosystem-A) that extends the Coral Architecture described above. This section will give a description of the main ideas involved in Ecosystem-A and indicate how it is built using the Coral architecture specification.

A. An Interoperable Domain Based Ecosystem

The main concept in Ecosystem-A is that of an *interoperable domain*: a set of devices among which content can flow seamlessly, independently of content format and DRM. Ecosystem-A defines policies for device participation in the domain as well as for content access and usage on those devices. Devices (or rather the relevant applications running on those devices) that are Members of a Domain are referred to as *Member Clients*. In Ecosystem-A, the purchase of content always binds content to a Domain D. That is, the proof of rights is encoded in a Rights Token of the form (D, C, U). This Rights Token maybe instantiated as a native DRM in any Member of the Domain D.

The node that manages a Ecosystem-A Domain is referred to as a *Domain Manager*. A Domain Manager is responsible for the creation of Domains as well as maintaining Membership relations between Domains and Members. The Domain Manager enforces the Ecosystem-A policies. For

[1]From the Liberty Alliance homepage: *The Liberty Alliance consortium is committed to developing an open standard for federated network identity that supports all current and emerging network devices.*

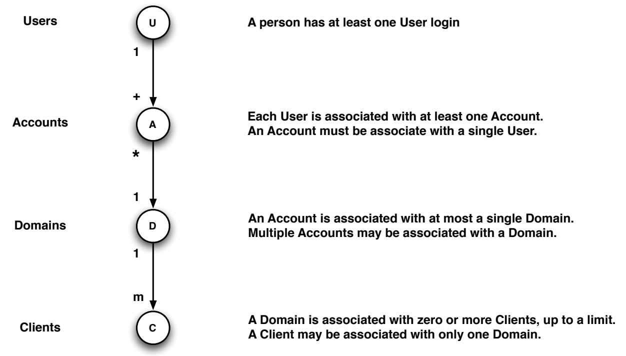
Users
U
A person has at least one User login
1
+
Accounts
A
Each User is associated with at least one Account.
An Account must be associate with a single User.
*
1
Domains
D
An Account is associated with at most a single Domain.
Multiple Accounts may be associated with a Domain.
1
m
Clients
C
A Domain is associated with zero or more Clients, up to a limit.
A Client may be associated with only one Domain.

Fig. 3. Principals and their Relations in Ecosystem-A

example, a device may be a Member of only one Domain at a time. The number of Members is also limited: if the limit is N and N devices are already Members of D, then the Domain Manager will not allow a new device to join, unless a current Member leaves. Since the ability to change the topology of a Domain determines the ability to access content, changing Domain topology requires authentication, such as a user identity combined with a password.

The process of attaching and maintaining Rights Tokens for a Domain is the responsibility of a *Rights Locker*. The choice made by Ecosystem-A is not to attach Rights Tokens to Domains directly, but rather via an intermediate principal, referred to as an *Account*. An Account is typically associated with a login name and password to which Rights Tokens are bound. A typical Account A might be of the form `user@monkeys4media.com` at the content store `www.monkeys4media.com`. The link with a Domain D is made by registering the Domain with the Account A, in a process that is very similar to registering clients to Domains. In particular, in order to register a Domain D to an Account A, a user will be required to log in to the Rights Locker managing the Account A and to explicitly request the registration of D with A. In Ecosystem-A, Rights Lockers enforce the policy that an Account can only be associated with a single Domain. However, a Domain can have multiple associated Accounts.

A typical use case scenario for Ecosystem-A is as follows

1) Alice purchases a number of Ecosystem-A devices.
2) Alice visits `www.monkeys4media.com` and sets up an Account A associated with the login name `alice@monkeys4media.com`.
3) Alice indicates that she wants Ecosystem-A content.
4) The Content Store offers to be Alice's Domain service provider. Alice can choose to accept this as the default, or provide a location for an alternative service.
5) Alice agrees with the default case. A Domain D is created that represents Alice's devices. The actual identifier D need not be known by Alice; she only needs to know her login name and password.
6) The Rights Locker at the Content Store associates Alice's Domain D with her Account A.
7) Alice registers her Ecosystem-A-enabled devices with the Domain Manager. This operation is infrequent, required only for Domain joining and re-joining.
8) Alice purchases a one month subscription for content C. The subscription usage model is designated by U.
9) The Content Store inserts a Rights Token of the form $T = (A, C, U)$ into the Rights Locker for Account A.
10) Alice requests content for device δ in her Domain. After verifying that all relevant principal relations are still valid, a Rights Mediator at the Content Store extends the Rights Token $T = (A, C, U)$ to a Rights Token $T' = (A \rightarrow D \rightarrow \delta, C, U)$. The first component of the new Rights Token reflects the hierarchical relationships between Account, Domain and Member.
11) The Rights Mediator invokes the appropriate license servers to create licenses for content C.
12) Feedback from the License Server causes Alice's device δ to request a DRM-specific license and content.

In the scenario above, most actions by Alice are infrequent (setting up Accounts, registering devices). Her daily experience is (i) that she purchases content for her set of devices D and (ii) the content is available on all of the devices regardless of their supported DRMs.

B. Risk Analysis and Usability for Domain Models

In this section we consider security and usability aspects for interoperable domain models.

1) Security of the Domain Model: The Domain model benefits consumers by making the policies for accessing content on member devices completely consistent. However, there are illegitimate uses of the Domain model that must be limited by policies. For example, a user might register a device, fill it up with content and then sell it. When the value of the content is larger than the value of the device, this attack might be economically attractive. Limiting the number of devices in a Domain helps to create disincentive against this attack.

Limiting the number of devices in a Domain, however, means that users may potentially fill up all of the available Domain slots[2]. In order to cope with this, Ecosystem-A offers users the option to remove existing Members from and join new Members to a Domain.

Ideally, when a device leaves a Domain, a synchronized transaction occurs between the Domain Manager and the departing device: when device δ leaves, the Domain Manager deletes the relation between δ and D and asks device δ to disable all Domain content. However, there may be circumstances in which the departing device is no longer available for synchronization. In such cases, a Domain Manager can:

1) Refuse to let the device leave, and be prepared for customer calls when customers have filled up their Domain slots and can no longer add new device
2) Unilaterally delete δ from the Domain D and accept the fact that the device δ still has access to Domain content, as it was never requested to disable Domain content.

[2]In a well-defined ecosystem, the limit is sufficiently large so that average users will never notice it.

In Ecosystem-A, option 2 was chosen, along with additional policies that minimize the side effects. In particular, Domain Membership expires if a Member device does not connect to its Domain Manager for renewal within a certain period of time. Also, whenever δ tries to join a Domain other than D, the new Domain Manager will force δ to disable all content for D. The expiration period should be long enough to fall outside of typical consumer experience.

2) Usability of Domains in a Static Environment: Since Domain memberships for infrequently-used devices expire under the policies of Ecosystem-A, users must authenticate to the Domain Manager to renew Domain memberships for these devices. Reauthenticating to the Domain Manager is not difficult for devices outside of the home, but it is awkward for the static devices in a user's home. Ecosystem-A thus allows automatic renewal of Domain Membership for devices that are close to a *Domain Anchor*, which serves as a spatial center for the Domain[3]. Domain Members that are determined to be proximate[4] to the Domain Anchor are given a special authentication token that can be used to prove locality of the member device and enable renewal of Domain Membership.

C. Building Ecosystem-A on Top of Coral

In this section we consider the construction of Ecosystem-A on top of the Coral Architecture, considering in particular how the main components of Ecosystem-A refine basic Coral Architecture roles.

For example, a Domain Manager is an instance of the Coral Architecture Principal Manager specialized to manage Domain and Member principals and their relations. The `<ISA>` relation of the Coral Architecture represents the membership relation. Domain Managers are specified in more detail than a Coral Architecture Principal Manager:

- A Principal Manager does not impose any conditions on the cardinality of its relations; a Domain Manager does. In particular, it requires that Members belong to only one Domain and that the number of Members for a given Domain be less or equal to some number N.
- *A* Principal Manager exposes a basic interface to update ecosystem variables (referred to as *Policy Variables*), but no specifics are defined. In Ecosystem-A there is a much more refined *Policy Update* interface. In particular, it allows for the update of the Domain Limit N and the values of various time-out periods[5].

A Rights Locker is an example of Coral node incorporating more than one Coral Architecture role. Since it manages relations between Account and Domain principals, it depends on the Principal Manager role. A Rights Locker is also a Rights Registry that stores and manages Rights Tokens. The Rights Locker extends the basic roles of the Coral Architecture by defining policies such as a limit on the frequency of Account-Domain changes and timeout values for cached Rights Tokens.

[3]Domain Anchors might be limited in number. For example in Ecosystem-A, only one Anchor per Domain is allowed.

[4]The notion of proximity in Ecosystem-A is based on devices being in a common private IP subnet and on the number of hops between devices.

[5]Ecosystems may choose to enable or disable dynamic policy updates. For example, the one-domain-per-device restriction is static in Ecosystem-A and is not dynamically updated.

IV. Conclusions

In this paper we have presented the Coral Architecture as well as Ecosystem-A an interoperable ecosystem built on top of the Coral Architecture. We have described the Coral Architecture as a toolkit consisting of roles, nodes and secure messaging. We have sketched the construction of an intuitive domain based ecosystem on top of this toolkit. In particular we have shown how the components of Ecosystem-A can be constructed by refining and grouping the primitive roles of the Coral Architecture. Within the Coral Consortium other ecosystems than Ecosystem-A have been constructed on top of the Coral Architecture as well. From the lessons learned, the Coral Consortium is currently in the process of re-structuring some of the architectural components in order to allow even more flexible ecosystem design. The Coral goals remains to provide tools and methods to enable different DRMs to flourish, protecting the intellectual property of content owners to the maximum extent, at the same time hiding the presence of DRM from the end-user.

Acknowledgment

The authors are indebted to all the members of the Coral Consortium. Many of the ideas presented in this paper can be traced back to intense but fruitful discussions in Coral meetings and at Coral conference calls.

References

[1] INDICARE (The INformed DIalogue about Consumer Acceptability of DRM Solutions in Europe. [Online]. Available: http://www.indicare.org

[2] B. Rosenblath, "The french legislation on DRM interoperability: right problem, wrong solution," Mar. 2006. [Online]. Available: http://www.indicare.org/tiki-read_article.php?articleId=190

[3] DMP, "Digital media project." [Online]. Available: http://www.dmpf.org

[4] Coral. [Online]. Available: http://www.coral-interop.org

[5] MPEG-21, "MPEG-21 multimedia framework." [Online]. Available: http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm

[6] P. A. Jamkhedkar and G. L. Heileman, "Drm as a layered system," New York, NY, USA, pp. 11–21, 2004.

[7] R. Safavi-Naini, N. P. Sheppard, and T. Uehara, "Import/export in digital rights management," in *DRM '04: Proceedings of the 4th ACM workshop on Digital rights management*. New York, NY, USA: ACM Press, 2004, pp. 99–110.

[8] X. Wang, "MPEG-21 Rights Expression Language: enabling interoperable Digital Mights Management," *IEEE Transactions on Multimedia*, vol. 11, no. 4, pp. 84 – 87, Oct. 2004.

[9] A. U. Schmidt, O. Tafreschi, and R. Wolf, "Interoperability challenges for DRM systems," in *Proceedings of IFIP/GI Workshop on Virtual Goods*, 2004. [Online]. Available: http://virtualgoods.tu-ilmenau.de/2004/

[10] N. Rump, "Can Digital Rights Management be standardized?" *IEEE Signal Processing Magazine*, vol. 21, no. 2, pp. 63 – 70, Mar. 2004.

[11] R. Koenen, J. Lacy, M. MacKay, and S. Mitchell, "The long march to interoperable digital rights management," in *Proceedings of the IEEE*, vol. 92, no. 6, June 2004.

[12] W. Bradley and D. Maher, "The NEMO P2P service orchestration framework," in *Proceedings of the 37th Hawaii International Conference on System Sciences*, Jan. 2004.

[13] Liberty Alliance. [Online]. Available: http://www.projectliberty.org

[14] RFC3546 (TLS). [Online]. Available: http://www.ietf.org/rfc/rfc3546.txt