Proceedings of the 5th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, Madrid, Spain, February 15-17, 2006 (pp156-162)

A Distributed DRM Platform Based on a Web-Oriented Watermarking Protocol

FRANCO FRATTOLILLO, SALVATORE D'ONOFRIO
Research Centre on Software Technology
Department of Engineering, University of Sannio
Corso Garibaldi, n. 107, Benevento
ITALY

Abstract: This paper presents a distributed digital rights management platform (DRMp) based on a web-oriented watermarking protocol. The platform enables web service providers (SPs) to dynamically supply copyright protection services on behalf of web content providers (CPs) in a security context. Thus, CPs exploiting the platform can take advantage of copyright protection services supplied by SPs acting as trusted third parties (TTPs) without having to directly implement them. On the other hand, SPs can follow the proposed design approach to implement protection procedures that can be easily exploited by CPs without imposing a tight coupling among the involved web entities.

Key–Words: Digital rights management, watermarking protocol, service framework

1 Introduction

Digital watermarking [3] has gained popularity as a main technology to set up reliable DRMps, which are software platforms purposely developed to implement the copyright protection of digital contents distributed on the Internet [1]. However, some documented and relevant problems, such as the "customer's right problem" [11] and the "unbinding problem" [10], can arise if DRMps are directly managed by CPs that autonomously implement the watermarking procedures or insert watermarks that are not properly bound to the distributed contents or to the transactions by which the contents are purchased. In fact, such DRMps end up implementing protection processes that do not correctly take into account the buyers' rights, since the watermark is autonomously inserted by CPs, i.e. the copyright owners, without any control.

A possible solution to the above problems consists in developing DRMps that can exploit TTPs able to implement watermarking and dispute resolution protocols by which CPs can both obtain an adequate protection of their contents distributed on the Internet and determine the identity of guilty buyers with undeniable evidence [10]. However, this means that DRMps have to be characterized by distributed software architectures that enable distinct web entities to dynamically cooperate in a secure context without imposing a tight coupling among them.

This paper presents a distributed DRMp based on a web-oriented watermarking protocol able to solve the problems described above, such as the "customer's right problem" and the "unbinding problem" [5]. The platform enables SPs to dynamically supply copyright protection services based on watermarking technologies on behalf of CPs in a security context. Thus, CPs that exploit the proposed platform can take advantage of copyright protection systems supplied by SPs acting as TTPs without having to directly implement them. On the other hand, SPs can follow the proposed design approach, based on web services [2] and other specific programming techniques [6, 15], to implement copyright protection procedures, such as watermarking procedures, which can be easily integrated into the proposed DRMp and exploited by CPs [4].

The outline of the paper is as follows. Section 2 describes the watermarking protocol assumed as service model by the DRMp. Section 3 presents the architecture of the DRMp adopting the proposed protocol. Section 4 describes the main implementation details of the platform. In Section 5 a brief conclusion is available.

2 The Watermarking Protocol

The proposed DRMp implements a copyright protection process that exploits a watermarking protocol based on four web entities: the buyer (B), the content

provider or seller (CP), the service provider (SP), and the trusted watermark certification authority (WCA). The protocol, which is widely discussed in [5],

- solves both the "customer's right problem" and the "unbinding problem";
- keeps B anonymous during the purchase web transactions;
- enables users who are not provided with digital certificates issued by Certification Authorities to purchase protected digital contents;
- does not require B to be able to autonomously perform security actions that are not automatically implemented by common web browsers;
- does not require a double watermark insertion, thus both avoiding to discredit the embedded protection and a possible degradation of the final quality of the distributed contents.

In particular, the protocol assumes that B is identified by means of a code associated to his/her credit card, and forces WCAs to be directly involved in the commercial transactions that take place whenever B wants to buy a digital content distributed by CP [5].

2.1 The Protection Protocol

Figure 1 shows the scheme of the interactions taking place during a web transaction by which B purchases a copy of the digital content X distributed by CP. In particular, the notation N.n denotes the message dispatch from the entity N, at the step n of the protocol. Messages are exchanged through SSL connections characterized by different authentication schemes negotiated at the connection start-up. Furthermore, the notation $E_{entity}(data)$ specifies a ciphered token whose $data$ are encrypted with the $entity$'s secret key, whereas the notation $Eph_{entity}(data)$ specifies a ciphered token whose $data$ are encrypted by exploiting a cryptosystem that is "privacy homomorphic" with respect to the watermark insertion [10].

B visits the CP's web site and, after having chosen X, negotiates with CP to set up a common agreement AGR, which states the rights and obligations of both parties as well as specifies the digital content of interest. During this negotiation phase B may have free access to the CP's web site or use a registered pseudonym, thus keeping his/her identity unexposed.

Then, B communicates his/her will of buying X to CP by sending the negotiated AGR (B.1). Upon receiving AGR, CP generates the token $E_{CP}(TID, XD, AGR, T_{CP})$ that includes: (1) the transaction identifier TID, which is a code used by

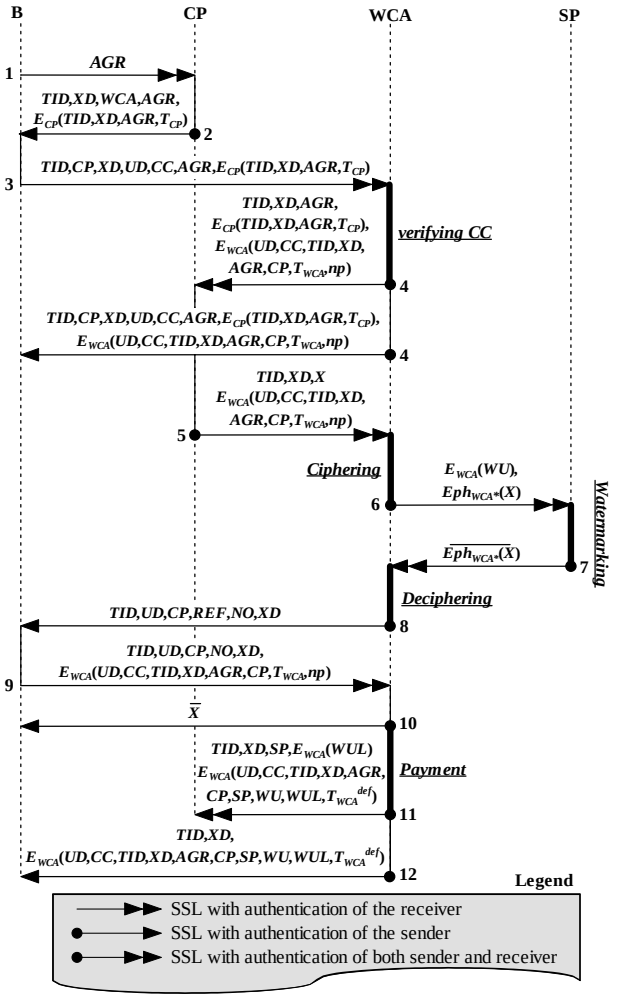

Figure 1: The protection protocol.

CP to identify the current transaction; (2) the description of X, denoted as XD; (3) the purchase order represented by the negotiated AGR; (4) the timestamp T_{CP}, which is generated by CP in order to make the token's freshness assessable. The token is sent to B (CP.2) together with plaintext information, such as the reference WCA to the WCA selected by CP.

B receives the message CP.2 and sends WCA the token $E_{CP}(TID, XD, AGR, T_{CP})$ (B.3), previously received from CP, together with further plaintext information, such as: (1) TID, XD and AGR, whose definitions are reported above; (2) CP, which is the reference to CP; (3) UD and CC, which are respectively the identity and the number associated to the B's credit card. In fact, data exchanged in the message B.3 allow WCA to check if B can pay X and to unambiguously identify B. Therefore, if data associated to the credit card are incorrect or the credit card turns out to be invalidated, the transaction is interrupted.

B.3 is considered by WCA a purchase order involving B and CP. Therefore, after verifying

the B's credit card, WCA generates the token $E_{WCA}(UD, CC, TID, XD, AGR, CP, T_{WCA}, np)$, which includes, among the others, T_{WCA} and np: the former is a timestamp that makes the token's freshness assessable, whereas the latter is a flag specifying that the B's credit card has not been charged yet. The token is sent to B as a "temporary" purchase certificate, whereas is sent to CP as a "temporary" sale certificate (WCA.4). In addition, WCA also returns some other information to B and CP in order to enable them to make a check on the current transaction. In fact, B and CP cannot access the information contained in the temporary, ciphered certificates, and this prevents them from maliciously modifying the certificates. However, they can verify the plaintext data exchanged in WCA.4, and so they can abort the transaction if data turn out to be invalid.

After verifying WCA.4, CP sends WCA the watermarking request CP.5, which includes X. After receiving X, WCA encrypts it by exploiting a cryptosystem that is "privacy homomorphic" with respect to the subsequent watermark insertion [10]. WCA also generates the fingerprinting code WU, which will have to be embedded in X to identify B. To this end, in order to always associate the same code to the same buyer, WCA exploits two specific functions, Φ and Ψ: the former generates a binary code μ identifying the buyer on the base of CC and UD, whereas the latter generates a bit string τ depending on XD and T_{WCA}. Thus, μ is always the same for a given credit card, whereas τ varies under different digital contents and timestamps. WU is the concatenation of μ and τ. In addition, WCA exploits a further function ϵ to generate an extended version of WU, denoted as WUL, whose ciphered form will be used by CP to identify B. Then, WCA selects an SP and sends it the watermarking request (WCA.6), which includes $E_{WCA}(WU)$ and $Eph_{WCA^*}(X)$, where $*$ denotes that the content X has been ciphered with a one-time secret key.

After receiving WCA.6, SP can directly watermark $Eph_{WCA^*}(X)$, since the encryption function applied by WCA is "privacy homomorphic" with respect to the watermark insertion operation. Once $Eph_{WCA^*}(X)$ has been watermarked, SP sends WCA the message SP.7, which contains the new watermarked content $\overline{Eph_{WCA^*}(X)}$ obtained by inserting the watermark in the encryption domain.

WCA decrypts $\overline{Eph_{WCA^*}(X)}$ and generates $\bar{X}$, the final version of the watermarked copy of X. In fact, the privacy homomorphic cryptosystem used by WCA results in the following equalities:

$$\overline{Eph_{WCA^*}(X)} = Eph_{WCA^*}(\bar{X})$$

$$\bar{X} = Dph_{WCA^*}(\overline{Eph_{WCA^*}(X)})$$

where the operator Dph denotes the decryption function corresponding to the encryption function Eph.

Once generated $\bar{X}$, WCA notifies its availability to B. In particular, message WCA.8 also specifies a "nonce" NO and the reference REF to the download server, which can be also distinct from WCA and from which B may download the watermarked content $\bar{X}$. Then, if all data exchanged in B.9 are valid, B can contact the server REF and download $\bar{X}$ (WCA.10). Thus, after the correct download of $\bar{X}$, WCA can charge the B's credit card and generate the token $E_{WCA}(UD, CC, TID, XD, AGR, CP, SP, WU, WUL, T^{def}_{WCA})$, which represents the definitive version of the purchase and sale certificates to be sent to B and CP respectively (WCA.11 and WCA.12). In addition, WCA also sends CP $E_{WCA}(WUL)$, which will be used by CP to refer to the corresponding sale certificate, SP and TID in its databases.

2.2 The Identification and Arbitration Protocol

This protocol, shown in Figure 2, can be conducted whenever a pirated copy X' of a protected digital content X owned or distributed by CP is found in the market. To this end, CP can ask WCA for starting the protocol by sending it X' and the reference to the SP exploited to protect the original content X (CP.1).

WCA sends SP the ciphered content $Eph_{WCA}(X')$ (WCA.2), and this prevents SP from getting access to the final versions of the protected contents distributed by CP. Then, SP extracts the embedded watermark from $Eph_{WCA}(X')$ and communicates the fingerprinting code $E_{WCA}(WU')$ to WCA (SP.3). WCA takes charge of generating the

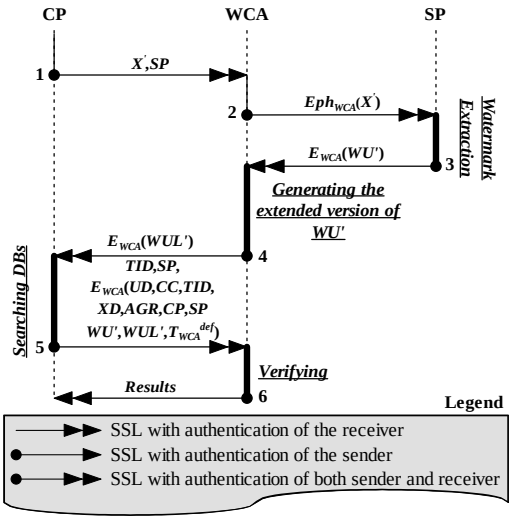

Figure 2: The identification and arbitration protocol.

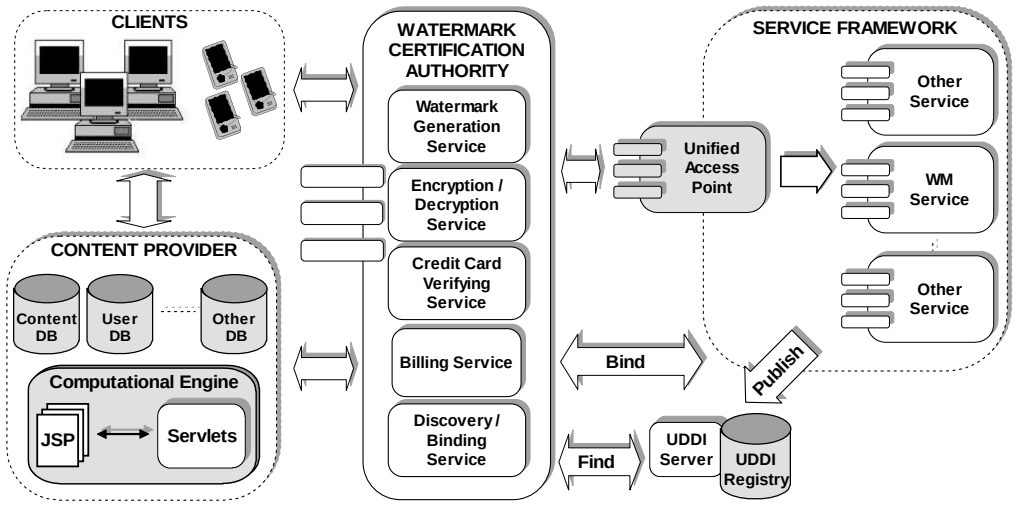

Figure 3: The architecture of the proposed DRMp.

extended version of WU', denoted as WUL'. Then, WCA sends CP $E_{WCA}(WUL')$ (WCA.4).

CP accesses its databases and uses $E_{WCA}(WUL')$ to search them for a match. When a match is found, CP retrieves the sale certificate $E_{WCA}(UD, CC, TID, XD, AGR, CP, SP, WU', WUL', T^{def}_{WCA})$ as well as further information associated to $E_{WCA}(WUL')$, and requires the buyer identification by sending these data to WCA (CP.5).

WCA decrypts the sale certificate and verifies all data received from CP. If all data turn out to be correct, the identity of the buyer is revealed, and WCA can adjudicate him/her to be guilty, thus closing the case. Otherwise, the protocol ends without exposing any identity.

3 The DRM Platform

The proposed platform, whose architecture is shown in Figure 3, consists of three main parts. The first part groups the web servers of a CP, and represents the "front-end" tier of the platform seen by user clients. The second part is represented by the WCA, and is the "middle" tier of the platform. The third part represents the "back-end" tier of the platform, and is basically composed of the web services [2] implemented by SPs and made available to the platform by means of a purposely designed "service framework" [4].

The platform has been developed according both to the WS-Security (Web Services Security) specifications, which define a set of SOAP header extensions for end-to-end SOAP messaging security, and to a "federated model" for the identity management of the web services' operators [14].

The front-end tier of the platform is represented by the CP's web applications, whose logic is assumed to be adapted to the distributed architecture of the platform. As a consequence, a CP wanting to exploit the proposed DRMp has to modify its applications in order to integrate them in the new interaction scheme involving the WCA.

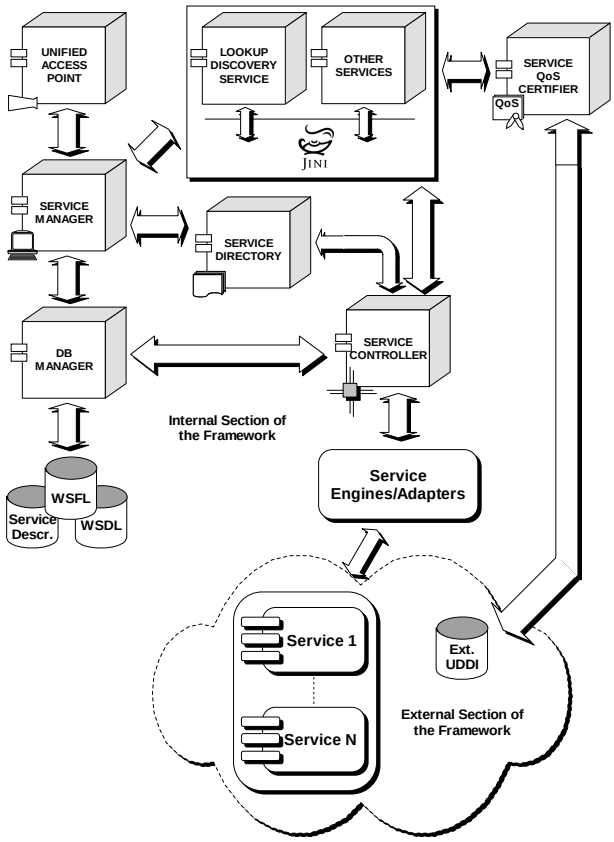

Figure 4: The service framework.

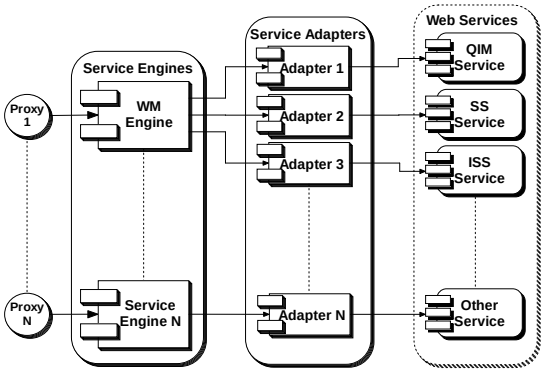

Figure 5: The invocation scheme implemented within the service framework.

WCA acts as a TTP and so it provides the DRMp with the security services needed to execute the watermarking and dispute resolution protocols. Therefore, its software architecture has a static internal characterization with respect to such assumptions. On the contrary, the back-end tier of the platform has been purposely designed as a service framework in order to make the architecture modularly extensible by web services dynamically loaded and supplied by distinct and external SPs. In particular, the service framework does not directly expose the web services that it groups, but it hides them behind a *unified access point*, which acts as a unique interface toward the external web entities for all the web services internally loaded. Such an interface, designed itself as a web service, takes charge of receiving the service requests specified according to what published in the UDDI registries and of dispatching them to the web services internally loaded according to the strategies implemented by the framework.

The choice of implementing the service framework as a "wrapper" for a set of web services hidden from external users and exposed through a *unified access point* enables the services that have to be supplied by the framework to be abstractly expressed. In fact, the wrapper allows the service framework to publish in the UDDI registries simplified and standard versions for the interfaces of the web services that are then dynamically loaded in the framework, and this makes such interfaces independent of the possible implementations actually developed by SPs.

4 The Implementation of the DRM Platform

This section focuses on the main implementation details concerning with the back-end tier of the proposed DRMp, which has been designed as a service framework (see Figure 4). In particular, to support flexibility and to make the service integration easy and dynamical, the service framework has been developed in Java. It consists of two main sections: the former comprises the internal services of the framework, whereas the latter comprises the web services supplied by SPs and dynamically loaded in the framework.

The internal services have been implemented in several software components whose interfaces have been defined according to a "component framework" approach [15]. In particular, each component of the framework implements a set of behavior rules that define the execution environment and the skeleton of the framework, i.e. the set of minimal services needed to enable SPs to dynamically integrate their web services in the DRMp. Furthermore, to facilitate the integration among the framework components and the external web services, four specific design patterns [6] have been used: "inversion of control", "separation of concerns", "proxy" and "adapter".

The first pattern enables the framework to control the execution of components and to coordinate all the events produced by transactions. The second pattern allows the internal services of the framework to be implemented as collections of well-defined, reusable, easily understandable and independent components [8]. The third pattern is exploited to create "surrogate" objects for the external web services grouped in the framework in order to control the access to them. Finally, the fourth pattern is used to convert the specific interfaces of the external web services into simplified interfaces that can be invoked by the internal components of the framework. In fact, the proxy and adapter patterns make it possible to dynamically load the external web services into the framework without imposing constraints on their public interfaces.

Figure 6 shows the scheme of the interactions, labelled by numbers, taking place when a service request is issued to the framework. In particular, the request is received by the *unified access point*, which is a web service that acts as a service dispatcher toward all the external web services dynamically loaded into the framework and made available to CPs. Therefore, when the *unified access point* receives a service request from outside (1), it contacts the *service manager* (2), which has the main task of managing the request.

The *service manager* accesses the *database manager* (3), which determines the characteristics, i.e. the "profile" (4), that a web service provided by the framework should have in order to match the requirements derived from the external request issued. To this end, the *database manager* manages a repository (see Figure 7) which specifies some main parameters able to characterize the web services' implementations. Such parameters can specify, for exam-

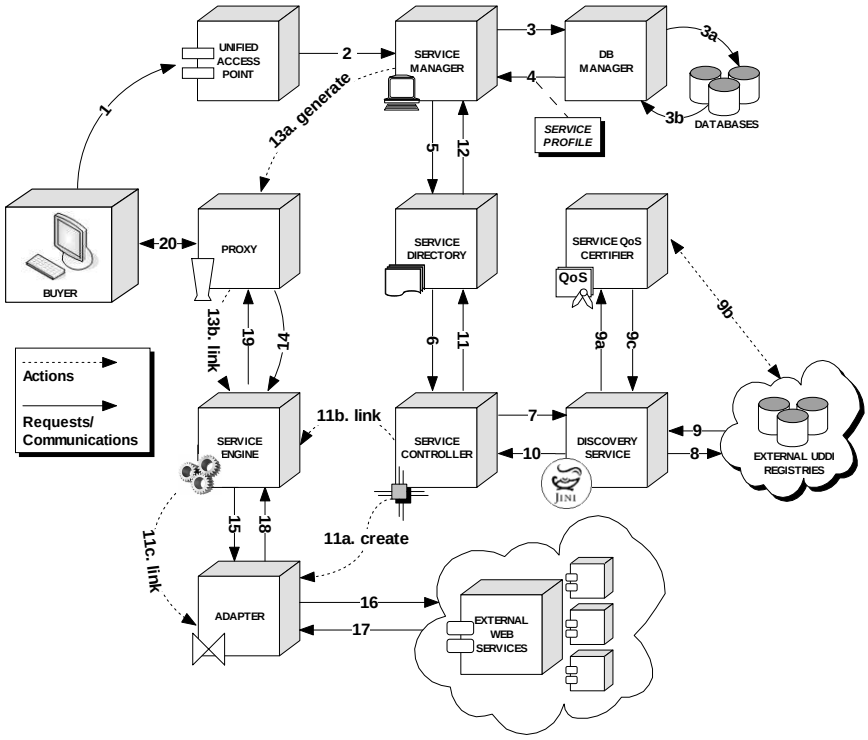

Figure 6: The interaction scheme implemented within the service framework.

ple, the protection level achievable by a given service implementation as well as the implementation behavior with respect to web transactions. Therefore, the available repository makes it possible to determine a particular "service profile" depending on both protection and QoS (quality of service) requirements specified in the service requests originally issued to the framework.

Then, the *service manager* searches the *service directory* (5) for a web service having the profile thus determined. If the service results in being already loaded and available within the framework, the *service directory* returns the information about the service (12) received from the *service controller* (6,11) to the *service manager*, which thus can create a specific *proxy* object (13a,13b) that has the task of routing the service request coming from outside (20) to the corresponding *service engine* according to the invocation scheme shown in Figure 5 (14,19). Such an engine is an intermediate software layer that translates the service invocations performed on the *proxy* objects to invocations on *adapter* objects (15,18). In fact, a *service engine* receives service requests coming from the *proxy* objects and specified according to the simplified web interfaces published by UDDI registries, and takes charge of invoking the corresponding and selected web services according to their particular interfaces (15,16,17,18). To this end, to enable a dynamical loading of external web services into the framework without imposing constraints on the web services' interfaces, the *service engine*s exploit the *adapter* objects, which are dynamically generated and loaded whenever an external web service is discovered and loaded into the framework (11a,11b,11c) [7, 9].

If the required service is not available within the framework, the *service directory* has to start the *discovery service* supplied by specific framework components implemented by exploiting the JINI framework [13]. The *discovery service* is activated via the *service controller* (6,7). It can search UDDI registries to discover the required service (8) and can return the result to the *service directory* (9,10,11), which communicates it to the *service manager* (12). To this end, it is worth noting that the *discovery service* is designed to exploit an extended UDDI model by which it is possible to find for a web service with the requested protection and QoS characteristics [12]. Therefore, the *discovery service* can search UDDI registries to discover a web service with the characteristics imposed by the service profile previously determined. Once the web service is found, the *service QoS certifier* can verify the service characteristics claimed by its SP (9a,9b,9c): if the characteristics map with the requirements specified by the service profile, the web service is loaded into the framework and the profile is stored in the internal repository by the *database manager*.

When this phase ends, the new external web ser-

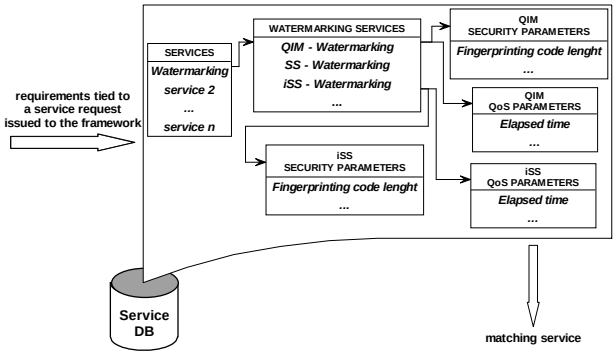

Figure 7: The service description database.

vice results in being loaded into the framework, and its *adapter* object is generated and linked to the corresponding *service engine* by the *service controller* (11a,11b,11c). Then, the *service manager* can generate the *proxy* object (13a,13b) that will receive the service requests issued to the framework and coming from outside (20).

5 Conclusions

This paper presents a distributed DRM platform based on a web-oriented watermarking protocol and implemented by using web services and specific programming technologies. The platform enables SPs to dynamically supply copyright protection services on behalf of CPs in a security context, whereas CPs exploiting the platform can take advantage of a copyright protection system without having to directly implement it. In fact, the proposed design approach, based on a service framework, makes the DRM platform modular and easily extensible, since the copyright protection services developed by SPs can be easily and dynamically exploited by CPs without requiring a tight coupling among the involved web entities.

References:

[1] M. Barni and F. Bartolini, Data Hiding for Fighting Piracy, *IEEE Signal Processing Magazine*, Vol. 21, No. 2, 2004, pp. 28–39.

[2] R. Brunner, F. Cohen *et al.*, *Java Web Services Unleashed*, SAMS Publishing, 2001.

[3] I. Cox, J. Bloom and M. Miller, *Digital Watermarking: Principles & Practice*, Morgan Kaufman, 2001.

[4] F. Frattolillo and S. D'Onofrio, An Effective and Dynamically Extensible DRM Web Platform, *Procs of the Int'l Conference on High Performance Computing and Communications*, in LNCS, Vol. 3726 , pp. 411–418, Sorrento, Italy, September 21–24, 2005.

[5] F. Frattolillo and S. D'Onofrio, A Web Oriented Watermarking Protocol, *Procs of the Int'l Conference on Signal Processing*, in *Enformatika*, Vol. 7, pp. 91–96, Prague, Czech Republic, August 26–28, 2005.

[6] E. Gamma, R. Helm *et al.*, *Design Patterns*, Addison Wesley, 1995.

[7] G. C. Gannod, H. Zhu and S. V. Mudiam, On-the-fly Wrapping of Web Services to Support Dynamic Integration, *Procs of the 10th Working Conference on Reverse Engineering*, Victoria, Canada, November 2003.

[8] R. Johnson and B. Foote, Designing Reusable Classes, *Journal of Object-Oriented Programming*, Vol. 1, No. 2, 1988, pp. 22-35.

[9] M. Lampe, E. Althammer and W. Pree, Generic Adaptation of Jini Services, *Procs of the Ubiquitous Computing Workshop*, Philadelphia, USA, October 2000.

[10] C. L. Lei, P. L. Yu *et al.*, An Efficient and Anonymous Buyer-Seller Watermarking Protocol, *IEEE Trans. on Image Processing*, Vol. 13, No. 12, 2004, pp. 1618–1626.

[11] N. Memon and P. W. Wong, A Buyer-Seller Watermarking Protocol, *IEEE Trans. on Image Processing*, Vol. 10, No. 4, 2001, pp. 643–649.

[12] S. Ran, A Model for Web Services Discovery With QoS, *ACM SIGecom Exchanges*, Vol. 4, No. 1, 2003, pp. 1–10.

[13] W. K. Richards, *Core JINI*, Prentice-Hall, 1999.

[14] S. Shin, *Secure Web Services*, http://www.javaworld.com, 2005.

[15] C. Szyperski, *Component Software. Beyond Object-Oriented Programming*, Addison Wesley, 1997.