Modeling Service based on XML Schema for MPEG Multimedia Middleware

Hendry and Munchurl Kim*

Abstract

Recently MPEG (Moving Picture Experts Group) has started a new standardization activity called MPEG Multimedia Middleware (M3W). The M3W supports multimedia applications independent of hardware and software such as operating systems so that the portability and interoperability of multimedia applications are greatly improved. In doing so, the M3W adopts the concept of a service-based middleware. The M3W standardizes a set of multimedia Application Programming Interfaces (APIs) and their implementations can be provided by third parties in the form of services. In this paper, we introduce the usage of XML metadata in describing the standard multimedia APIs and third party's services and also we explain the role of the XML Schema within the M3W in providing transparent access from application to the M3W services.

1. Introduction

The increasing appetite of end-user for the multimedia content has pushed the software vendors to be able to develop multimedia applications in a short time. To make easier and faster the development of multimedia applications, a new approach is being taken in a standardized way by which a multimedia application needs only to assemble some components or services, and links them together to form a specific application. Some codes or modules that are common for multimedia applications can be implemented separately in the form of services so that any application can be developed in a higher level.

The service-based approach offers several advantages for the application developers. Firstly, it makes the applications more modular, hence increasing their reusabilities. An application that uses many reusable modules can be developed faster since they are not necessarily to be implemented again. Secondly, the service-based approach enables the multimedia applications to be enhanced with respect to their portability and interoperability over different platforms since any specific processing module to a certain platform can be implemented as a service. To this end, the multimedia applications

* 한국정보통신대학교(ICU) 공학부 부교수

become less dependent of specific platforms.

MPEG, through MPEG Multimedia Middleware (M3W), provides a service-based middleware that is dedicated for multimedia applications[2]. For the M3W, MPEG is currently standardizing a set of multimedia APIs while the implementations of those standardized APIs are left for the third party's implementations for services. For the rest of the paper, the M3W service will be written as *Service* (with capital S). To improve the concept of modularity and reusability, a *Service* may also have dependency on the other *Services* so that the *Service* may have a hierarchical structure in the M3W. Multimedia applications need to know only the syntaxes and semantics of the standardized set of APIs of the M3W while how they are implemented is outside the scope of the M3W standard.

In order to provide a transparent access to the *Services* of the M3W, the M3W manages the relationship between the standardized APIs and the *Services* that provide the implementations and also the relationships among the *Services*. Those relationships are described by using XML Schema by which the syntaxes and semantics of their data types are standardized. In this paper, we explain the role of the XML Schema in the M3W for providing the transparent access from the applications to the *Services*. When a request from an application is made to use certain functionality (through APIs), the M3W seeks for the availability of the *Services*, initiates them, and then returns the handler to the requester. All these tasks are possible with the provision of XML description in the M3W.

This paper is organized as follow: In Section 2, we explain the M3W architecture. Readers can get more detailed information about the M3W and their interaction in Section 3. We explain the concept of the hierarchical structure of *Service* in Section 4, and describe the role of XML Schema in the M3W for providing the transparent access from applications to *Services* in the middleware in Section 5. We show a use-case example taken from a real multimedia APIs specification with the instances of XML description and their usage. Finally, we present the conclusion in Section 6.

2. M3W Architecture

The M3W lies between the computing platform and the applications/other middleware (which MPEG considers as an application for M3W as well) to bridge the access gap between them. As shown in Fig. 1, the communication between the M3W and the upper layer (applications) is provided through the defined API sets which are supposed to be standardized in the MPEG. While the link from the application to M3W is standardized, the link from M3W to the platform is not standardized. It is left as an implementation issue of the *Services* that realize the M3W APIs [4].

As shown in Fig. 2, *Service* has a direct arrow to access the platform. *Service* can be im-

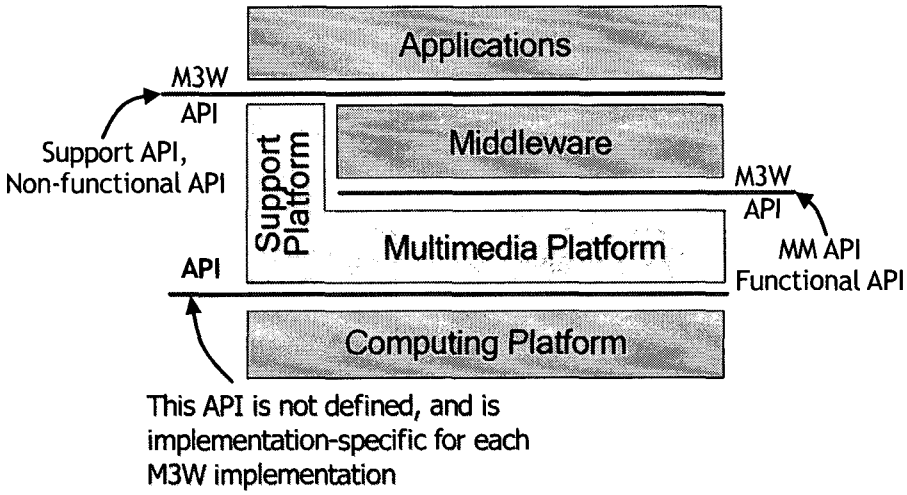


Fig. 1. M3W position in the computing layer [4]

plemented in a customized way by accessing a special API or resource provided by the platform where it is running. By this approach, the application does not depend upon the computing platform since the middleware (M3W) has taken over the dependency to the platform. When an application is moved or ported to a new platform, minimum or no further modification effort may be required since it can just simply access the same (standard) available functionality but implemented by a different *Service* (for the new platform).

Fig. 2 shows the architecture of the M3W in terms of the M3W entities. One of the M3W entities is the *Logical Component* which indicates a group of multimedia APIs that perform a certain multimedia functions. The *Logical Component* provides a list of APIs definitions while the implementation of those APIs is provided by the *Service(s)*. In other words, the

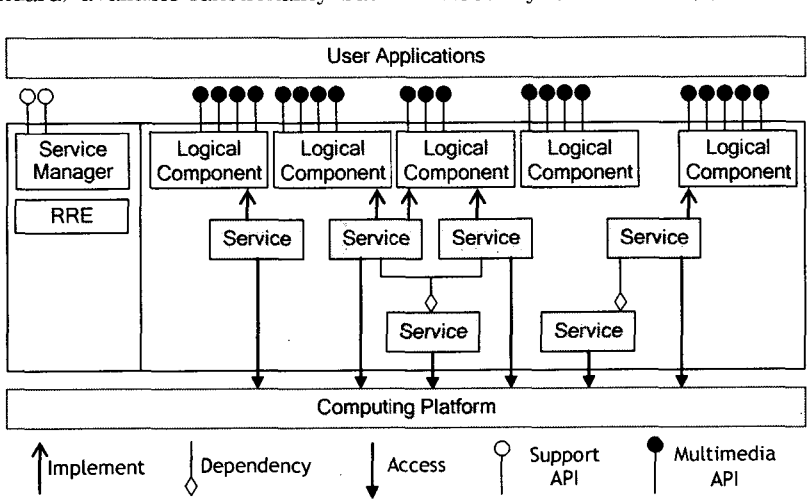


Fig. 2. M3W entities [1]

Logical Component and the *Service* can be analogized as an abstract class and an instance, respectively. The class definition contains a set of APIs that can be used to access its functionality while their implementations are provided by the instance of that class [1].

The M3W has several support *Services* that are dedicated for the management of the middleware operations. Regarding the management of the available *Services* in the middleware, there are two major support *Services*: *Runtime Environment* (RRE) and *Service Manager*. The RRE is responsible to manage the storing, retrieving, instantiation, and initialization of the *Services*. The *Service Manager*, on the other hand, is responsible to link *Service* instances, resolving their association with the *Logical Component*, and returning the handler of the *Service* instance to the requester (application). In short, the *Service Manager* receives a request for a function (whose APIs are exposed by the *Logical Component*), takes a decision on which the *Services* are needed, request the RRE to instantiate and initialize them, and then *Service Manager* links them before handling the handler to the requester. More details will be described in the later sections.

3. Hierarchical Structure of Service

Service is designed to be flexible, modular, and efficient in its implementation. This can be achieved by letting an implementation of a *Service* use the functionality that is provided by other *Services*. Hence, a *Service* may depend on other *Service*(*s*) and have a hierarchical dependency in their structures [1].

Fig. 3 illustrates the implementation of a *Service* that efficiently uses the functionality from other *Services*. A video decoder *Service*, for example, may consist of several independent functions such as a variable length coding (VLC) function, discrete cosine transform (DCT) function, quantization function, motion estimation/compensation function, etc. The video decoder *Service* can be implemented as mod-

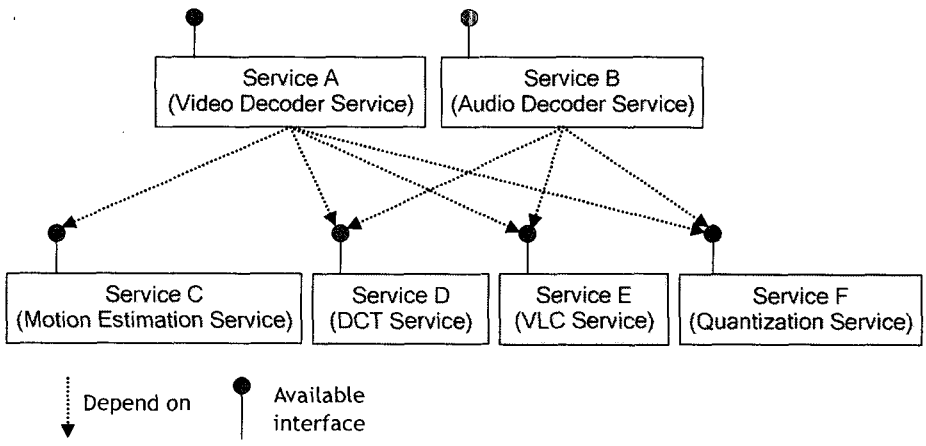


Fig. 3. Service composition example (1)

ular as possible by using other implementations that offer the required functionalities. Hence, in its implementation, the implementer needs to code the execution flow of those *Services*.

In other cases, supposed that an audio decoder *Service* also resides in the same middleware. With the hierarchical concept of the *Service* implementation, it may not be necessary to have two DCT, VLC, and quantization *Services*. The same *Services* that are used for the video decoder *Service* can be used to serve the audio decoder *Service* as well. So, the middleware can be very efficient in the sense that there is no duplication of the offered functions.

The binding between the *Service* and its dependencies is mediated by the *Service Manager*. During its instantiation and initialization, the *Service* can request to the *Service* Manager to provide its dependent *service(s)*. The *Service Manager* gives the best possible *Service* by using the knowledge from the available metadata (XML description of the *Services*) [1].

4. XML Description Roles in M3W

The XML description is used to describe the *Logical Component* and *Service*. These *Logical Component* and *Service*'s descriptions are important for the *Service Manager* in performing its operation since it has to keep information about the entities that it manages. The *Logical Component* description tells the Service Manager about the list of the *Logical Components* (with list of its multimedia APIs in it) that the middleware can offer to the applications. The *Service* description tells the *Service Manager* about which interfaces of the *Logical Components* are implemented and provided, and also about which interfaces are required (implemented by other *Service(s)*) in order to be able to execute the *Service*.

4.1. Logical Component Description

Fig. 4 shows a fragment of the complete structure of the *Logical Component* model. As shown in Fig. 4, the *Logical Component* hier-

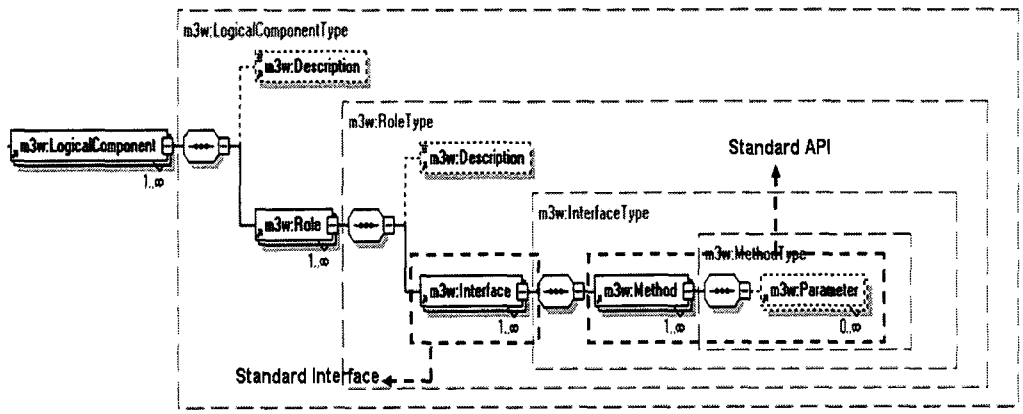


Fig. 4. Structure of Logical Component description (Fragmented from the complete structure)

archically contains roles, interfaces, and APIs (methods). The *Logical Component* description is an input for the *Service Manager* as configuration by the middleware implementer or middleware administrator. The complete syntaxes and semantics are provided in [1].

4.2. Service Description

Fig. 5 shows the structure of a *Service* model specified by XML Schema. As indicated in Fig. 5, several *Services* are described together in a *Component*. A *Component* is used as a container. It may be imagined like an DLL in which several OCX objects are carried together. A *Component* is also a unit for trading. A software vendor that produces *Services* can package their *Services* (which are related each other) in a single *Component* and deliver/sell it.

The *Service* description shown in Fig. 5 is related to the *Logical Component* description by the InterfaceID element. Hence, we can say that the interfaces that are listed in the *Logical Component* description are implemented by the *Services* which are related through the InterfaceID.

The *Service* may have a hierarchical structure because it may have dependency on other interfaces which are implemented by other *Services*. This dependency is formed by the existence of RequiredInterfaces element as the child element of the *Service* as shown in Fig. 5. The complete syntaxes and semantics can be found in [1].

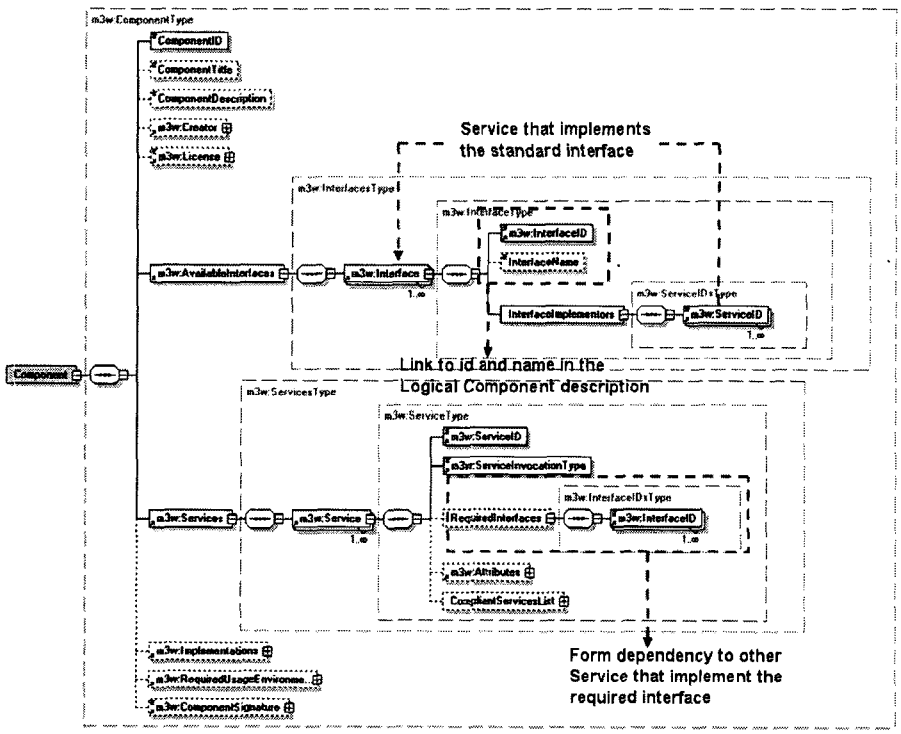


Fig. 5. Structure of *Service* description

Unlike the *Logical Component* description which is input to the *Service Manager* as a manual configuration, the *Service Manager* can actively and automatically collect the *Service* description at the installation of the *Service* into the middleware.

4.3. Transparent Service Access

The *Service Manager*, the *Logical Component* description, and the *Service* description are the key factors for providing a transparent access from an application to the *Service*. The access is transparent in the sense that the application does not have to know anything about the *Service*. The application just needs to request a multimedia function by simply asking the *Service Manager* to provide the implementation of the function, so that *Service Manager* handles the rest.

Fig. 6 shows a visualization of the steps to get a *Service*. The application first sends a request to get a *Service* for the *Logical Component* whose function it needs. The *Service Manager* upon receiving the request will infer within its knowledge (*Logical Component* description and *Service* description) to get the identification of the appropriate *Service* that implements the requested *Logical Component*. In the case that there are several *Services* that implement the requested *Logical Component*, the *Service Manager* will decide the best *Service* to be instantiated. The policy of deciding the best *Service* depends on the implementation of the *Service Manager*. For example, the *Service Manager* can be implemented to give higher priority to the *Service*

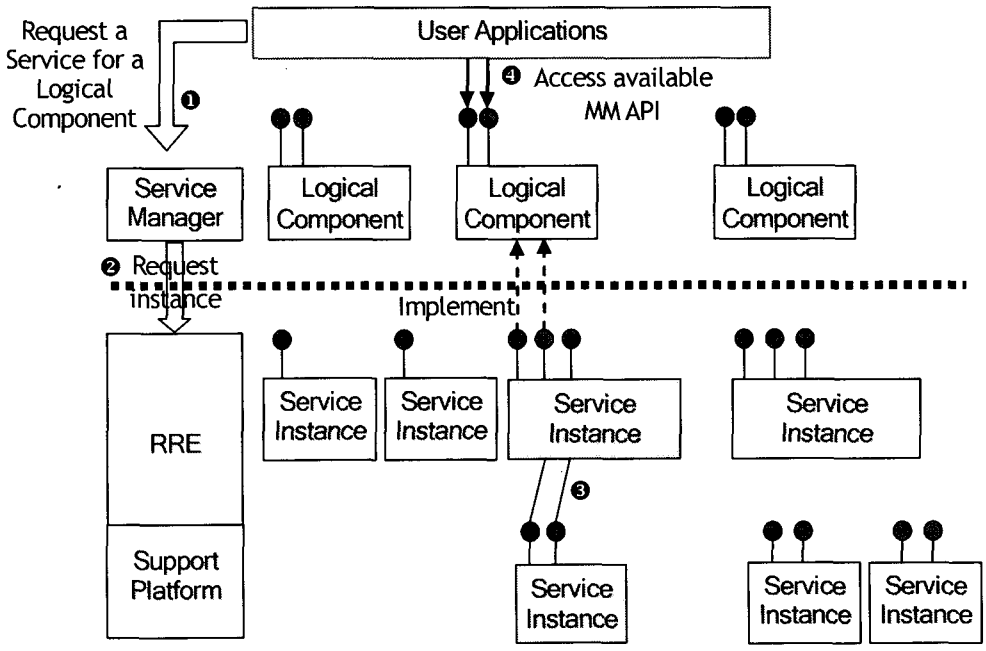


Fig. 6. Steps for requesting a Service

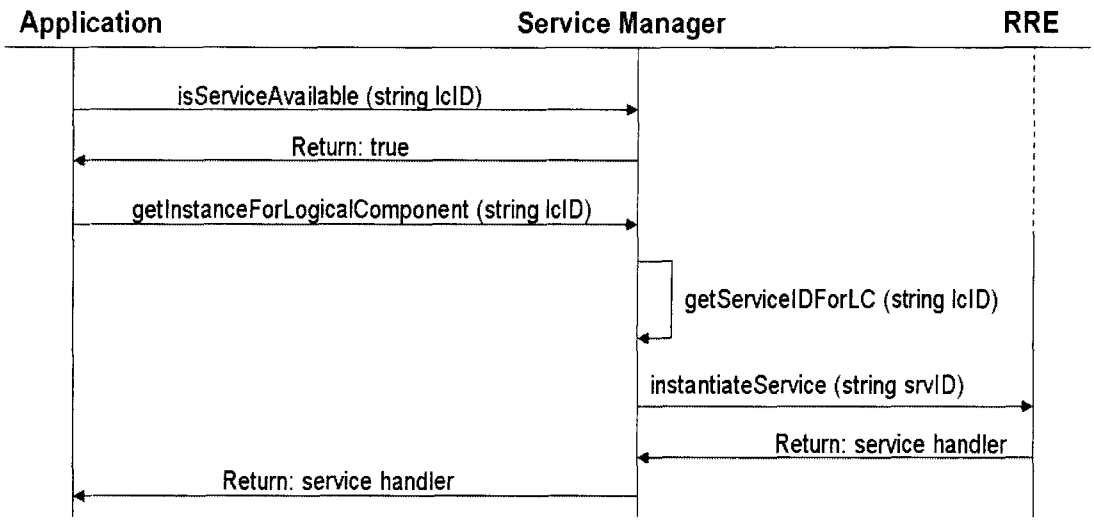


Fig. 7. Sequence diagram of requesting a Service.

that has less dependency or, in other extreme case, the *Service Manager* can give a higher priority to the *Service* that is provided by a certain vendor. After having the identification of the *Service* with its dependences, the *Service Manager* will ask *RRE* to instantiate the *Services*. Then, the *Service Manager* will link the *Services*. Finally, the *Service Manager* returns the handler of the *Service* to the application.

Fig. 7 shows a sequence diagram of the process described above. It shows the process in a more technical manner by showing which APIs of the support *Services* are invoked.

5. Use-Case: XML Description Instance Examples and Their Usage

In this section, we show a use-case description example of the *Logical Component* from UHAPI specification (that will be part of M3W Part 2: Multimedia API) [3][15]) about an audio controller and we also show the description of the *Services* that provide the implementation of the interfaces that are declared in the *Logical Component*. Through this simple example, we show how the *Service Manager* can take benefit from the descriptions.

Table 1 shows an example of the *Logical Component* description. In this example, the middleware just offers two *Logical Components* which are related to the audio controller function: automatic audio level controller and audio dynamic range controller.

Suppose there is Vendor A, a software developer who provides an implementation of the *Logical Component* about an audio dynamic range controller and its implementation is aligned to the M3W specification as shown in its *Service* description in Table 2. Vendor A implements the audio dynamic range controller

Table 1. Example of Audio Controller Logical Component description instance

```
<?xml version="1.0" encoding="UTF-8"?>
<M3WLC xmlns="http://mpeg-m3w/MM_APIs"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://mpeg-m3w/MM_APIs D:\MPEG\Schemas\M3WLogicalComponent.xsd"
version="1.0">
   <LogicalComponents>
      <ParamClassTypes>
         <ParamClassType name="uhErrorCode_t">
            <Description>This type represents the error code that can be thrown by the API of
this Logical Component
               </Description>
            </ParamClassType>
            <ParamClassType name="uhAavl_ValStat_t">
               <Description>This type represents the validity status of the related
property/properties
         </Description>
            </ParamClassType>
            <ParamClassType name="uhAavl_Ntf_t">
               <Description>
                  Values of this type represent notification functions.
            That is, there is a one-to-one correspondence between these values and the functions
            in the notification interface uhIAavlNtf
               </Description>
            </ParamClassType>
            <ParamClassType name="uhAavl_NtfSet_t">
               <Description>
                  Values of this type represent sets of notification functions which are
                  typically constructed by logical OR-ing of values of type uhAavl_Ntf_t
               </Description>
            </ParamClassType>
            <ParamClassType name="uhAavl_AllNtfs_t">
               <Description>Defines the set of all values of type uhAavl_Ntf_t</Description>
            </ParamClassType>
            <ParamClassType name="uhAdrc_Mode_t">
               <Description>Values of this type represent the DRC modes settable by the
user</Description>
            </ParamClassType>
            <ParamClassType name="uhAdrc_ModeSet_t">
               <Description>Values of this type represent notification functions</Description>
            </ParamClassType>
            <ParamClassType name="uhAdrc_NtfSet_t">
               <Description>
            Values of this type represent sets of notification functions that are
                  typically constructed, by logical OR-ing of values uhAdrc_Ntf_t
               </Description>
```

```
</ParamClassType>
            <ParamClassType name="uhAdrc_AllNtfs">
               <Description>Defines the set of all values of type uhAdrc_Ntf_t</Description>
            </ParamClassType>
         </ParamClassTypes>
         <LogicalComponent id="urn:MPEG-M3W/LC/AVL" name="AutomaticVolumeLeveling">
            <Description>
         Provides interface suites for the control over the automatic volume leveling functionality
            </Description>
            <Role id="urn:MPEG-M3W/ROLE/AAVL"
name="AudioAutomaticVolumeLevelerRole">
               <Interface id="urn:MPEG-M3W/INF/uhlAavl" name="uhlAavl">
                  <Method name="Subscribe"output="void">
                     <Parameter name="pINotify"type="*uhIAavlNtf"/>
                     <Parameter name="cookie"type="UInt32"/>
                     <Parameter name="notifs"type="uhAavl_NtfSet_t"/>
                  </Method>
                  <Method name="Unsubscribe" output="void">
                     <Parameter name="pINotify"type="*uhIAavlNtf"/>
                     <Parameter name="cookie"type="Uint32"/>
                     <Parameter name="notifs"type="uhAavl_NtfSet_t"/>
                  </Method>
                  <Method name="Enable"output="void">
                     <Parameter name="enable"type="Bool"/>
                  </Method>
                  <Method name="GetEnabled" output="Bool"/>
                  <Method name="SetLevel"output="void">
                     <Parameter name="level"type="Uint32"/>
                  </Method>
                  <Method name="GetLevel"output="Uint32"/>
               </Interface>
               <Interface id="urn:MPEG-M3W/INF/uhlAavlNtf" name="uhlAavlNtf">
                  <Method name="OnSubscriptionChanged" output="void">
                     <Parameter name="cookie"type="Uint32"/>
                     <Parameter name="notifs"type="uhAavl_NtfSet_t"/>
                  </Method>
                  <Method name="OnActEnabledChanged" output="void">
                     <Parameter name="cookie"type="Uint32"/>
                     <Parameter name="status"type="uhAavl_ValStat_t"/>
                     <Parameter name="enable"type="Bool"/>
                  </Method>
               </Interface>
            </Role>
         </LogicalComponent>
         <LogicalComponent id="urn:MPEG-M3W/LC/ADRC"
name="AudioDynamicRangeController">
```

```
            <Description> Provides interface suites for the audio dynamic range control functionality
</Description>
            <Role id="urn:MPEG-M3W/ROLE/AdynRngCon"
name="AudioDynRgnControllerRole">
              <Interface id="urn:MPEG-M3W/INF/uhlAdrc" name="uhlAdrc">
                <Method name="Subscribe"output="void">
                  <Parameter name="pINotify"type="*uhIAdrcNtf"/>
                  <Parameter name="cookie"type="Uint32"/>
                  <Parameter name="notifs"type="uhAdrc_NtfSet_t"/>
                </Method>
                <Method name="Unsubscribe" output="void">
                  <Parameter name="pINotify"type="*uhIAdrcNtf"/>
                  <Parameter name="cookie"type="Uint32"/>
                  <Parameter name="notifs"type="uhAdrc_NtfSet_t"/>
                </Method>
                <Method name="GetSuppModes" output="uhAdrc_ModeSet_t"/>
                <Method name="SetMode"output="void">
                  <Parameter name="mode" type="uhAdrc_Mode_t"/>
                </Method>
                <Method name="GetMode"output="uhAdrc_Mode_t"/>
               <Method name="GetActualMode" output="uhAdrc_Mode_t"/>
            </Interface>
            <Interface id="urn:MPEG-M3W/INF/uhlAdrcNtf" name="uhlAdrcNtf">
              <Method name="OnSubscriptionChanged" output="void">
                <Parameter name="cookie"type="Uint32"/>
                <Parameter name="notifs"type="uhAdrc_NtfSet_t"/>
              </Method>
              <Method name="OnActualModeChanged" output="void">
                <Parameter name="cookie"type="Uint32"/>
                <Parameter name="actualMode" type="uhAdrc_Mode_t"/>
              </Method>
            </Interface>
          </Role>
        </LogicalComponent>
    </LogicalComponents>
</M3WLC>
```

simply in a single *Service*.

Based on the information in Table 1, an audio player application for example can request an audio dynamic range controller functionality. Fig. 8 shows a sequence diagram with the completed set of input parameters when the application requests.

The application requests the audio dynamic range controller to the *Service Manager* by giving the urn:MPEG-M3W/LC/ADRC. Upon receiving the request, the *Service Manager* checks the *Logical Component* description for

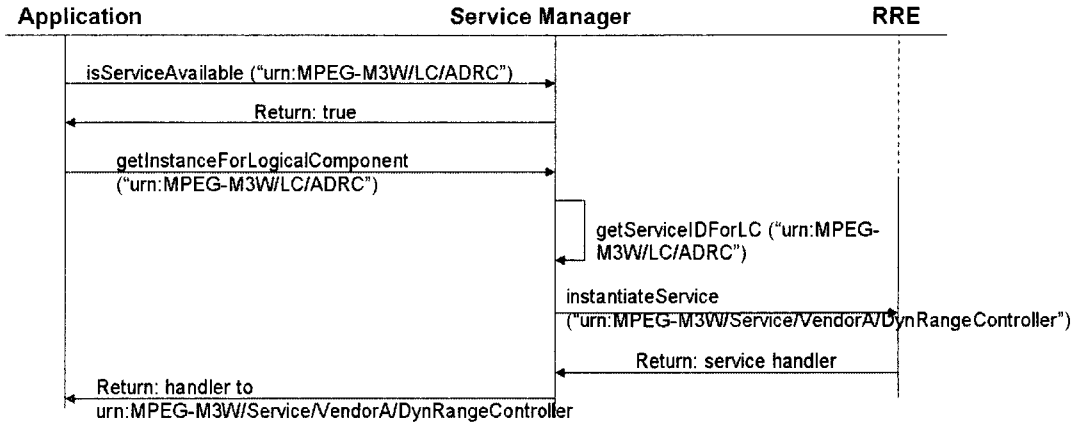


Fig. 8. Requesting Service that implements urn:MPEG-M3W/LC/ADRC

what interfaces are required for the requested functionality. It finds out that urn:MPEG-M3W/LC/ADRC requires urn:MPEG-M3W/INF/uhlAdrc and urn:MPEG-M3W/INF/uhlAdrcNtf. So it seeks in the *Service* description whether there is at least one *Service* that implements the required interfaces and finds that the *Service* whose identification is given by urn:MPEG-M3W/Service/ VendorA/DynRangeController. So the *Service Manager* asks the *RRE* to instantiate the *Service* and then returns the handler to the application.

Suppose later that the middleware gets an implementation of the audio controller from other vendors. Vendor B also provides an implementation of the *Logical Components* in Table 1. However, unlike Vendor A that implements an audio dynamic range controller in a single *Service*, Vendor B implements it by having dependency on the other interfaces. Table 3 shows the Vendor B's *Service* description.

After insertion of Vendor B's *Service* to the middleware, when processing the same request such as the one shown in Fig. 8, the *Service Manager* has to decide which *Service* should be instantiated since there are two *Services* that implement the requested *Logical Component*. This decision process is inserted after getServiceIDForLC and before instantiateService calling to the *RRE*.

5. Conclusion

The M3W is a service-based middleware that is dedicated for multimedia applications. Through the M3W, MPEG will offer a set of standardized APIs while their implementations are provided by the *Service(s)*. Since the *Services* are developed by third parties, there is a necessity to have a mechanism to bind the *Logical Component* and the *Service*. This binding will provide information about which *Services* implement a certain *Logical Component*.

In this paper, we describe the usage of XML in providing the binding between the *Logical Component* and the *Services*. Both the *Logical*

Table 3. Service description implemented by Vendor B

```
<?xml version="1.0" encoding="UTF-8"?>
<Component xmlns="http://mccb.icu.ac.kr/m3w"
xmlns:UED="urn:mpeg:mpeg21:2003:01-DIA-NS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://mccb.icu.ac.kr/m3w D:\MPEG\Schemas\M3WService.xsd">
  <ComponentID>urn:MPEG-M3W/Component/VendorB/AudioController</ComponentID>
  <ComponentTitle>VendorB--AudioController</ComponentTitle>
  <ComponentDescription>Provide implementation for audio controller</ComponentDescription>
  <Creator>
    <CreatorName>Vendor B Inc.</CreatorName>
    <CreatorWebsite>http://www.vendorb.com</CreatorWebsite>
  </Creator>
  <AvailableInterfaces>
    <Interface>
      <InterfaceID>urn:MPEG-M3W/INF/uhlAavl</InterfaceID>
      <InterfaceName>uhlAavl</InterfaceName>
        <InterfaceImplementors>
          <ServiceID>urn:MPEG-M3W/Service/VendorB/AudioLeveler</ServiceID>
        </InterfaceImplementors>
      </Interface>
     <Interface>
        <InterfaceID>urn:MPEG-M3W/INF/uhlAavlNtf</InterfaceID>
        <InterfaceName>uhlAavlNtf</InterfaceName>
        <InterfaceImplementors>
          <ServiceID>urn:MPEG-M3W/Service/VendorB/AudioLeveler</ServiceID>
        </InterfaceImplementors>
      </Interface>
      <Interface>
        <InterfaceID>urn:MPEG-M3W/INF/uhlAdrc</InterfaceID>
        <InterfaceName>uhlAdrc</InterfaceName>
        <InterfaceImplementors>
          <ServiceID>urn:MPEG-M3W/Service/VendorB/DynRangeController</ServiceID>
        </InterfaceImplementors>
      </Interface>
      <Interface>
        <InterfaceID>urn:MPEG-M3W/INF/uhlAdrcNtf</InterfaceID>
        <InterfaceName>uhlAdrcNtf</InterfaceName>
        <InterfaceImplementors>
          <ServiceID>urn:MPEG-M3W/Service/VendorB/DynRangeController</ServiceID>
        </InterfaceImplementors>
      </Interface>
    </AvailableInterfaces>
    <Services>
      <Service>
        <ServiceID>urn:MPEG-M3W/Service/VendorB/AudioLeveler</ServiceID>
```

```
            <ServiceInvocationType>in-process</ServiceInvocationType>
        </Service>
        <Service>
            <ServiceID>urn:MPEG-M3W/Service/VendorB/DynRangeController</ServiceID>
            <ServiceInvocationType>in-process</ServiceInvocationType>
            <RequiredInterfaces>
            <InterfaceID>urn:MPEG-M3W/INF/uhlAavl</InterfaceID>
            <InterfaceID>urn:MPEG-M3W/INF/uhlAavlNtf</InterfaceID>
        </RequiredInterfaces>
    </Service>
  </Services>
</Component>
```

Component and the *Service* are modeled with XML description so that the middleware by the *Service Manager* can manage them. Together, the *Service Manager,* the *Logical Component* description, and the *Service* description are the key elements that make it possible the transparent access from the application to the *Service*. We also present a simple example of how the XML Schema models the *Logical Component* and the *Service*. Through the example, we show how the descriptions are used and how the transparent access in M3W becomes possible with the usage of XML technology.

References

[1] MPEG Systems Group.WD 3.0 of ISO/IEC 23004-3 Component Model. ISO/IEC JTCI/ SC29/WG11/N7918. Bangkok, Thailand. Jan 2006.

[2] MPEG Systems Group. WD 3.0 of ISO/IEC 23004-1 Architecture. ISO/IEC JTCI/ SC29/ WG11/N7917. Bangkok, Thailand. Jan 2006.

[3] MPEG Systems Group. WD 3.0 of ISO/IEC 23004-2 Multimedia API. ISO/IEC JTCI/ SC29/WG11/N7599. Nice, France. October 2005.

[4] MPEG Systems Group. M3W Tutorial. ISO/IEC TC JTCI/SC29/WG11/N7607. Nice, France. Oct 2005.

[5] Hendry,. et. al. Modeling M3W Logical Service using XML Schema. ISO/IEC JTCI/SC29/WG11/M12590. Nice, France. Oct 2005.

[6] Hendry,. et. al. M3W Service Manager, Service, and Metadata. ISO/IEC TC JTCI/ SC29/WG11/M12196. Poznan, Poland. Jul 2005.

[7] Hendry,. et. al. Response for CfP on MPEG Multimedia Middleware (M3W). ISO/IEC TC JTCI/SC29/WG11/M11871.Busan, South Korea. Apr 2005.

[8] MPEG. Call for Proposals on Multimedia Middleware (M3W). ISO/IEC TC JTCI/ SC29/WG11/N6981. Hong Kong, China. Jan 2005.

[9] Gelissen, Jean H.A. White paper on M3W:

Middleware for MPEG applications. http://www.idt.mdh.se/rtmm/rtmm-philips-mpeg.pdf. 2005.

[10] MPEG Requirements/Systems/MDS Group. MPEG Multimedia Middleware: Context and Objective. ISO/IEC TC JTCI/SC29/WG11/N6335. Munchen, Germany. Mar 2004.

[11] MPEG Requirements/Systems Group. MPEG Multimedia Middleware Requirements v.2.0. ISO/IEC TC JTCI/SC29/WG11/N6835. Palma de Mallorca, Spain. Oct 2004.

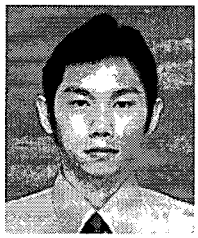
[12] Birbeck, Mark, et. al. Professional XML 2nd Edition. Wrox Press Ltd. United States. 2002.

[13] Wagner, Matthias,. et. al. An XML-based Multimedia Middleware for Mobile Online Auctions. International Conference on Enterprise Information System (ICEIS) 2001. Portugal. 2001.

[14] Open Services Gateway Initiative (OSGi). OSCAR: An OSGi framework implementation. http://oscar.objectweb.org/.

[15] Universal Home API (UHAPI). UHAPI Specification 1.1. http://www.uhapi.org/home

[16] XMLSpy™ Documentation. http://www.altova.com/download_doc.html.



Hendry

- 1997년 8월: University of Indonesia Faculty of Computer Science (학사)
- 2005년 2월: 한국정보통신대학교, 공학부 (석사)
- 2005년 2월~현재: 한국정보통신대학교 박사과정
- 관심분야 : 멀티미디어 DRM/IPMP, MPEG-21, MPEG-A/E

Munchurl Kim

- 1989년 2월: 경북대학교, 전자공학과 (공학사)
- 1992년 12월: University of Florida Electrical & Computer Engineering (석사)
- 1996년 8월: University of Florida Electrical & Computer Engineering (박사)
- 1997년 1월~2001년 2월: ETRI 방송미디어 연구부 팀장
- 2001년 2월~현재: 한국정보통신대학교 부교수
- 관심분야 : 비디오 부호화 및 처리, 대화형 멀티미디어 DMB/IPTV, MPEG-4/7/21, MPEG-A/E