UPmP: A Component-Based Configurable Software Platform for Ubiquitous Personalized Multimedia Services

Zhiwen Yu[1,2], Xingshe Zhou[2], Changde Li[2], Shoji Kajita[1], and Kenji Mase[1]

[1] Information Technology Center, Nagoya University, Japan
zhiwen@itc.nagoya-u.ac.jp, kajita@nagoya-u.jp, mase@nagoya-u.jp
[2] School of Computer Science, Northwestern Polytechnical University, P.R. China
zhouxs@nwpu.edu.cn, changde_lee@126.com

Abstract. As multimedia contents are becoming widely used in ubiquitous computing environments among many application fields, e.g. educational content management, entertainment, and live surveillance, the demand of personalized access to these contents has increased dramatically. Delivering ubiquitous personalized multimedia services (UPMSs) is a challenging task, which relies on many different functions. In this work, we propose a three-layer software platform, called UPmP to support efficient development and deployment of UPMSs. It fulfills the core functions for UPMS including service management, multimedia recommendation, adaptation, and delivery. We adopt component-oriented approach in building the platform. Therefore the configurability of the platform is inherently achieved. A representation model is introduced to hierarchically organize components and describe meta-level information about components. We also present a visual configuration tool together with a XML-based language for the purpose of platform configuration. The experimental results show the UPmP is flexible to be configured under different settings, and the overheads are acceptable.

1 Introduction

With rapid development of multimedia and communication technologies, it becomes possible to offer multimedia content to people whenever and wherever they are through different devices, such as personal computer, personal digital assistants (PDAs) and mobile phones. Multimedia content is widely used in ubiquitous computing environments among many application fields, such as digital course management, entertainment, and live surveillance. The number of multimedia content to access can be quite overwhelming. To quickly and effectively provide content from large amounts of media information, in the right form, to the right person, the multimedia content need to be personalized based on the user's preferences and his current contextual information, such as time of day, user location, and device conditions. These services are so-called ubiquitous personalized multimedia services (UPMSs).

Delivering UPMSs is a challenging task. It relies on many different functions, such as service management, multimedia adaptation, multimedia recommendation,

J. Ma et al. (Eds.): UIC 2006, LNCS 4159, pp. 1069 – 1079, 2006.

multimedia delivery, etc. Software infrastructures are needed to enable such functions to be achieved easily and systematically so that the service providers and application developers just need to concentrate on the application itself. In this paper, we present a software platform, namely UPmP (**U**biquitous **P**ersonalized **m**ultimedia **P**latform) to support efficient development and deployment of UPMSs. We adopt component-oriented approach in building the platform. Therefore the configurability of the platform is achieved inherently. Component-based software design has been widely utilized in many fields to implement complex functions. A software component is a unit of composition that can be deployed independently and is subject to composition by a third party [1]. Three major component models are presented and used successfully today: COM, CORBA, and Javabeans.

There are several benefits from using the UPmP software platform. First, it integrates third-party software to accomplish software reusability and complex function consummation. Second, the platform is configurable and allow service provider to select different functions based on the service needs. Third, the atomic components within the platform can be taken from pre-existing applications. It facilitates service development so as to reduce the cost of development as well as the time to market.

2 UPmP Architecture

The UPmP architecture consists of three layers: multimedia resources, service function components, and service instances (UPMSi), as shown in Fig. 1.

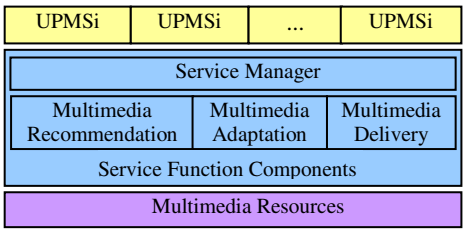

Fig. 1. UPmP architecture

2.1 Multimedia Resources

Multimedia resources are composed of multimedia content and corresponding description metadata. For the sake of interoperability with third-party services and applications, we adopt MPEG-7 description schema to represent multimedia metadata. The MPEG-7 *Creation DS* and *Classification DS* are used to describe information about the media item, such as the title, keyword, director, actor, genre, and language. This information is used to match user preferences. The *Variation DS* is used to specify variations of media content as well as their relationships. The *Variation DS* plays an important role in our content recommendation by allowing the selection among the different variations of the media content in order to select the most appropriate one in adapting to the specific capabilities of the terminal devices and network conditions.

The multimedia resources layer can integrate varied multimedia repository from a wide range of content providers by leveraging the O.K.I (Open Knowledge Initiative) Repository OSID (Open Service Interface Definition), which gains access to content in a manner that hides the technical detail by which that content is provided [2]. The O.K.I Repository Specification has been used to successfully integrate several applications with multiple content repositories, such as Sakai [3], which aims at building a collaboration and learning environment for higher education.

2.2 Service Function Components

The service function components are deployed as two sub-layers. The top layer is Service Manager. The bottom layer contains three components: Multimedia Recommendation, Multimedia Adaptation, and Multimedia Delivery. The Service Manager is responsible for lifecycle management of services. It interacts with services directly, and invokes functionalities supported by the function components in the bottom layer. Multimedia Recommendation is to select the right content in the right form for a service request. It takes user preference, terminal capability, and network condition into account. Multimedia Adaptation adjusts multimedia content to different requirements from service manager. It mainly involves two kinds of processes: content summarization and content transcoding, e.g. video-to-image conversion. Multimedia adaptation can be statically done at authoring time prior to delivery or dynamically done on-the-fly if needed. Multimedia Delivery is responsible for streaming or downloading media content to various terminals through different networks. If the modality recommended is continuous video or audio, the media deliverer streams the content to terminals. On the other hand, if the modality is static image or text, the media deliverer just downloads the content.

2.3 Service Instances

The service instances are concrete ubiquitous personalized multimedia services requested by a wide range of devices in ubiquitous computing environment. There are two typical scenarios for ubiquitous personalized multimedia services. One is providing a recommendation list with top L items. The other one is directly presenting the item with the highest score according to user preferences.

3 Component Representation

Components are functional units forming the UPmP platform. They can be composed to provide ubiquitous personalized multimedia services. A component comprises two parts: a metadata description and a processing entity. Component description presents detailed information of the component including component name, category, programming language, interface, hardware requirement, and software requirement (e.g. libraries, depended components). The component description is mainly used in platform configuration as well as service composition. Component entity is a software program (code) to accomplish a particular function.

For the sake of efficient organization, we model the components as a hierarchy. We give 3-layer definition to UPmP component hierarchy classification. The root element UPmPComponent is abstraction for all components. It is mainly divided into four categories, i.e. the second layer includes four items, which are ServiceManaging, ContentRecommendation, ContentAdaption, and ContentDelivery. The leaf components are different implementation algorithms or mechanisms of the abstract function in the upper layer. The UPmP component hierarchy structure is shown in Fig. 2. For space consideration, we here merely present the detailed structure of ContentAdaptation component.

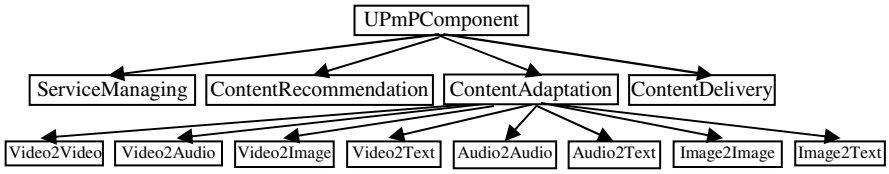

Fig. 2. Component hierarchy

For efficient discovery and reuse, information about the component should be described and advertised. Then the system can lookup and match desired components for a particular service according to the component metadata. We propose a service component description language (SCDL) based on XML to describe component. SCDL defines three kinds of information of a component: general information, public interface, and composition logic. The general information indicates a component's name, category, parent class, and also gives a brief annotation for the component. The category structure is the same as component hierarchy presented above. Category and annotation are very useful for component search and match. The public interface describes input and output formats, and also the real running entities of the component. The composition logic is provided to support service composition. It indicates the order of a component when it is composed with other categories of components using the Following and FollowedWith elements. It also indicates whether a component can be combined with its brother component. The components with the same parent class are regarded as brother components. For instance, Video2Image and Video2Text are brother components.

Fig. 3 shows a component metadata example. The component's name is *SComponentExample* and its category is *Video2Image*. The annotation and parent class are also given. From PublicInterface part, it can be seen that the input of the component is *MPEG* file, and the output is *JPEG* and *BMP* files. Two executors, *mpeg2jpeg.jar* and *mpeg2bmp.jar* are specified as the real running entities of the component. The composition logic indicates that the component can follow the components of *ContentRecommendation* and *ContentDelivery*, and be followed by the components of *ContentDelivery* in composing a service. However, it cannot be combined with its brother component.

```
<?xml version="1.0" encoding="UTF-8"?>
<SComponent xmlns="http://www.dcel.nwpu.edu.cn/SComponent_Schema">
<SComponentDescription>
    <GeneralInformation>
      <Name>SComponentExample</Name>
      <Category>Video2Image</Category>
      <Annotation>Transforming a video content into images.</Annotation>
      <ParentClass>ContentAdaptation</ParentClass>
    </GeneralInformation>
    <PublicInterface>
      <Input>
            <Format>MPEG</Format>
      </Input>
      <Output>
            <Format>JPEG</Format>
            <Format>BMP</Format>
      </Output>
      <Executors>
            <Executor>mpeg2jpeg.jar</Executor>
            <Executor>mpeg2bmp.jar</Executor>
      </Executors>
    </PublicInterface>
    <CompositionLogic>
      <Following>
        <Category>ContentRecommendation</Category>
        <Category>ContentDelivery</Category>
      </Following>
      <FollowedWith>
        <Category>ContentDelivery</Category>
      </FollowedWith>
      <CombinedWithBrotherComp>NO</CombinedWithBrotherComp>
   </CompositionLogic>
</SComponentDescription>
</SComponent>
```

Fig. 3. Component metadata (example)

4 Platform Configuration

The UPmP platform offers a set of optional functionalities, which can be switched off at the platform initializtion time. This flexibility is useful because not all the systems need to exploit the completed capabilities offered by the platform. In the simplest cases, the platform should support the development of lighter systems, which reduce the overhead during the interaction with the user. In particular, the developer may choose the multimedia recommendation techniques best suiting the requirements of the application domain. For instance, collaborative filtering efficiently support recommendation of multimedia, but it only works if ratings of the items are available. In contrast, content-based filtering is more suitable to the cases where meta-level information about the items is available.

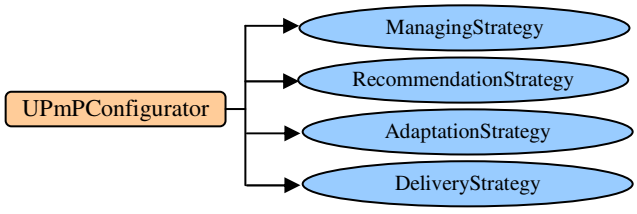

Fig. 4. UPmP configurator

To provide this flexibility, we have made the platform configurable so that the developers can select the functionalities offered by the platform. A GUI-based tool, called UPmPConfigurator, is proposed to customize platform functionalities according to needs or characteristics of different systems. The UPmPConfigurator mainly includes four parts: service managing strategy, content recommendation strategy, adaptation strategy, and delivery strategy as shown in Fig. 4.

For purpose of configuration at initializtion time, an XML-based platform configuration language (XPCL) is designed. The Extensible Markup Language (XML) is an ideal configuration language, because it is the universal format for structured documents and data on the Web and also extensible. However, XML itself does not tell platform administrator how to specify platform configuration parameters for his/her services. We define a set of suitable tags to specify platform running policies based on XML syntax. The XPCL is accordingly divided into four parts. The <UPmPConfiguration> tag is the root tag. It contains the entire four parts parameter or policy configuration. The <ServiceManagingStrategy> tag contains two tags: <EnterBlockedState> and <EnterWaitingState>. They specify whether the services enter *Blocked* or *Waiting* state respectively. The services are modeled with a lifecycle management model based on FSM (Finite State Machines), which is presented in our early work [4]. The <RecommendationStrategy> contains at least one <RecommendationAlgorithm> tag, which specifies a particular algorithm, e.g. content-based recommendation. The <AdaptationStrategy> contains at least one <Transcoding> tag, which is also a container tag. The <Transcoding> tag has one required attribute, "type", which identifies the type/class of the transcoding. It contains at least one <Transcoder> tag. The <DeliveringStrategy> contains at least one <DeliveringMode> tag.

```
<?xml version="1.0" encoding="UTF-8"?>
<UPmP xmlns="http://www.dcel.nwpu.edu.cn/UPmP_Schema">
<UPmPConfiguration>
  <ServiceManagingStrategy>
     <EnterBlockedState>YES</EnterBlockedState>
     <EnterWaitingState>NO</EnterWaitingState>
  </ServiceManagingStrategy>
  <RecommendationStrategy>
    <RecommendationAlgorithm>Content-based Recommendation</RecommendationAlgorithm>
      <RecommendationAlgorithm>Rule-based Recommendation</RecommendationAlgorithm>
  </RecommendationStrategy>
  <AdaptationStrategy>
    <Transcoding type="Video2Image">
      <Transcoder>MPEG2JPEG</Transcoder>
    </Transcoding>
    <Transcoding type="Video2Text">
      <Transcoder>MPEG2TXT</Transcoder>
    </Transcoding>
    <Transcoding type="Audio2Text">
      <Transcoder>WAV2TXT</Transcoder>
    </Transcoding>
  </AdaptationStrategy>
  <DeliveryStrategy>
    <DeliveringMode>Streaming</DeliveringMode>
    <DeliveringMode>Downloading</DeliveringMode>
  </DeliveryStrategy>
</UPmPConfiguration>
</UPmP>
```

Fig. 5. Platform configuration (example)

Fig. 5 gives an example of UPmP platform configuration. The services are set to enter the Blocked state but not the Waiting state. Two recommendation algorithms, content-based recommendation and rule-based recommendation are included. For multimedia adaptation, this configuration sets three transcoding mechanisms: Video2Image, Video2Text, and Audio2Text. The corresponding transcoders are MPEG2JPEG, MPEG2TXT, and WAV2TXT. The delivery strategy includes audio/video streaming and image/text downloading.

An XPCL-based description file is generated after a user completes the configuration through the visual UPmPConfigurator tool. The configurator interprets the configuration file, loads specified components, and builds a running platform.

5 Service Composition

A multimedia service is built as the composition of UPmP supported functional components. In our system, service composition is the selection of suited service components in order to deliver the service to a terminal in appropriate form. It consists of two steps. The first step is abstract function selection, e.g. multimedia adaptation. The second one is concrete handler selection, e.g. Video-to-Image transcoder.

Fig. 6 illustrates some composition paths for multimedia services under different conditions. S denotes the start point of services, while T stands for the terminal point. The rectangle contains components for recommendation purposes including components A, B and C. The oval contains adaptation components D, E, F, G, and H. The delivery components I and J are represented in the rounded rectangle. In Fig. 6(a), since the appropriate variation already exits, the service goes directly to deliver it. In Fig. 6(b), the service firstly performs video-to-text transcoding, and then

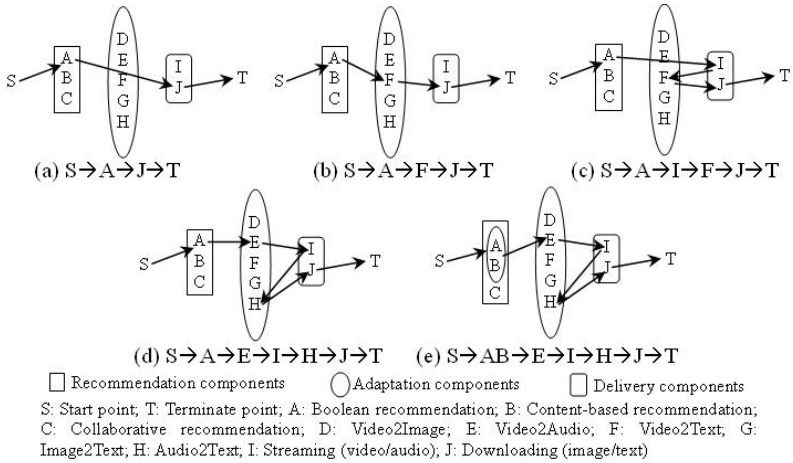

Fig. 6. Service composition examples

downloads the text. In Fig. 6(c), the service first performs video streaming under high bandwidth, then for the decrease of bandwidth, it performs video-to-text transcoding, and then delivers the text. In Fig. 6(d), the service first performs video-to-audio transcoding, then audio streaming, and then for bandwidth decrease dramatically it has to perform audio-to-text transcoding, last sends the text. In Fig. 6(e), component A and B are combined for the recommendation.

6 Implementation and Experiment

We have implemented a prototype of UPmP. The visual platform configuration tool UPmPConfigurator is implemented in Java Swing. The components are all implemented in Java and built as .jar files. Some media adaptation components are integrated from third party. To handle component dependency during platform configuration and service composition, we can utilize the Dependency Injection [5] pattern provided by the Spring framework [6]. Fig. 7 shows the main interface of UPmPConfigurator. It allows the developer to choose different strategies for service management, content adaptation, content delivery, and content recommendation. When a content adaptation button, e.g. Video-to-Image, is clicked, a list of transcoders will be presented for user's choice. For ubiquitous multimedia recommendation, rule-based technique is always combined with the other recommendation algorithms. It is used to infer the appropriate presentation form of a selected content from network condition and device capability [7]. The developers can directly choose some existing rules or define specific rules of their own.

We mainly evaluate our system by measuring the overhead of UPmP's configuration in terms of time. It includes the time to parse and interpret the configuration XML file, load specified components, and link them together. The experiment is conducted on a PC with 1.6 GHz Pentium 4 CPU, and 1 GB RAM running Windows XP. There are five different setup configurations involved in this experiment. The configuration details are presented in Table 1. Configuration 1 is a completed setup with all the four categories of components selected. Configuration 2, 3, and 4 test the overheads of different types of components. Configuration 5 is the configuration for a real running system. When selecting a category of content adaptation component, e.g. Video-to-Image, we simply select all the transcoders of it.

Table 1. Configuration details

Configuration	Selected components
1	all the four categories of components
2	all the service managing and recommendation components
3	all the content adaptation components
4	all the content delivery components
5	Enter Blocked State, Video-to-Image, Video-to-Text, Audio-to-Text, Streaming, Downloading, Content-Based Recommendation, and Rule-Based Recommendation

The experimental results are shown in Fig. 8, in which the time for each configuration is an average value of 10 runs. The completed configuration takes about 8.5 seconds. We can also observe that the main configuration overhead comes from linking content adaptation components, which takes nearly 75% of the total configuration time. Fortunately, this overhead is merely generated during the platform setup. It does not affect the performance of service delivery at running time. Through this experiment, we could conclude that the UPmP is flexible to be configured under different settings, and the overheads are acceptable.

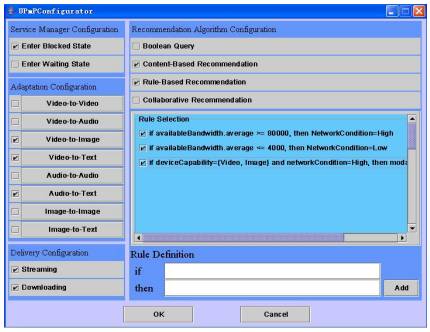

Fig. 7. Main interface of UPmPConfigurator

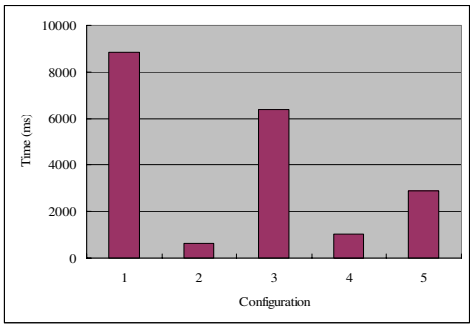

Fig. 8. Overhead of UPmP configuration

7 Related Work

There has been much research work done specifically to provide systematic architectural support for ubiquitous multimedia delivery. Gamma [8] is a content-adaptation server for wireless multimedia applications, which supports the automatic and transparent transcoding for individual users according to their pre-configured user profiles. CANS [9] is an application-level infrastructure for injecting application-specific components into the network. It supports automatic deployment of transcoding components for ubiquitous and network-aware access to Internet services. Dali [10] is a set of reusable libraries, which can be used for building processing-intensive multimedia software. Gamma, CANS, and Dali are mainly towards content adaptation without the support of service management and multimedia recommendation. QCompiler [11] is a programming framework to support building ubiquitous multimedia applications, which are mobile and deployable in different ubiquitous environments, and provide acceptable application-specific Quality-of-Service (QoS) guarantees. However, QCompiler does not involve server platform configuration and personalization functionalities. CAPNET [12] is a context-aware middleware for mobile multimedia applications. It fulfils broad functionalities including service discovery, event management, context storage, media content retrieval and adaptation to various mobile devices. But CAPNET is not built based on component and is not configurable in setup.

Recently, to deliver personalized multimedia to ubiquitous devices, some researchers have considered both user preference and device/network capability to

generate appropriate presentation to terminals e.g. [13] and [14]. However, none of them are proposed from the middleware perspective.

8 Conclusion

We described the architecture and key features of the UPmP, a general-purpose platform to support the deployment of ubiquitous personalized multimedia services. The major contributions of this paper include: (1) proposing a three-layer architecture for the general-purpose software platform; (2) introducing a component representation model that is helpful for component organization, indexing, and description; (3) presenting a configuration tool and an XML-based platform configuration language; (4) illustrating service composition under different conditions.

Future work is planned on the extension of the platform to support re-configuration at running time. We also envision deploying function components in a network of computers to improve performance in terms of throughput and scalability. It will call for the incorporation of load sharing and task scheduling mechanisms.

Acknowledgment

This work is partially supported by the Ministry of Education, Culture, Sports, Science and Technology of Japan under the "Development of Fundamental Software Technologies for Digital Archives" project, and the Doctorate Foundation of Northwestern Polytechnical University of China.

References

1. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley, Reading, Mass (1998)
2. Thorne, S., and Sim, S.: Integrating Applications with Repositories Using the O.K.I Repository OSID. JA-SIG Conference. (2005)
3. Sakai, http://sakaiproject.org/
4. Yu, Z.W., Zhou, X.S., Zhang, D.Q., Lugmayr, A., and Yu, Z.Y.: A Ubiquitous Personalized Multimedia Service Model Based on FSM, Proc. Of the 6th IEEE Intl. Conf. on Information and Technology: Coding and Computing, USA, 801-802 (2005)
5. Fowler, M.: Inversion of Control Containers and the Dependency Injection pattern. http://www.martinfowler.com/articles/injection.html. (2004)
6. Spring Framework, http://www.springframework.org/
7. Yu, Z.W., Zhang, D.Q., Zhou, X.S., Chin, C.Y., Wang, X.H., and Men, J.: Supporting Context-Aware Media Recommendation for Smart Phones, IEEE Pervasive Computing Magazine, Vol. 5, No. 3, July-September (2006)
8. Lee, Y.W., Chandranmenon, G., and Miller, S.C.: Gamma: A Content-Adaptation Server for Wireless Multimedia Applications. Lucent Technologies white paper. (2003)
9. Fu, X., Shi, W., Akkerman, A., and Karamcheti, V.: CANS: Composable, Adaptive Network Services Infrastructure. USENIX Symposium on Internet Technologies and Systems (USITS), March 2001, 135-146 (2001)

10. Ooi, W.T., et al.: Dali: A Multimedia Software Library. Proceedings of 1999 SPIE Multimedia Computing and Networking, 264-275 (1999)
11. Wichadakul, D., Gu, X.H., and Nahrstedt, K.: A Programming Framework for Quality-Aware Ubiquitous Multimedia Applications. ACM Multimedia 2002, 631-640. (2002)
12. Davidyuk, O., et al.: Context-aware middleware for mobile multimedia applications. The 3rd International Conference on Mobile and Ubiquitous Multimedia, 213-220 (2004)
13. Steiger, O., Ebrahimi, T., and Sanjuan, D.M.: MPEG-based Personalized Content Delivery. IEEE Intl Conf. on Image Processing, 45-48 (2003)
14. Lemlouma, T., and Layaida, N.: Encoding Multimedia Presentations for User Preferences and Limited Environments. IEEE ICME, 165-168 (2003)