Journal of Universal Computer Science, vol. 16, no. 10 (2010), 1291-1310
submitted: 29/7/08, accepted: 9/3/09, appeared: 28/5/10 © *J.UCS*

A Service-Oriented Platform for Ubiquitous Personalized Multimedia Provisioning

Zhiwen Yu
(School of Computer Science, Northwestern Polytechnical University, Xi'an, P. R. China
zhiwenyu@nwpu.edu.cn)

Changde Li
(School of Computer Science, Northwestern Polytechnical University, Xi'an, P. R. China
lichangde@gmail.com)

Xingshe Zhou
(School of Computer Science, Northwestern Polytechnical University, Xi'an, P. R. China
zhouxs@nwpu.edu.cn)

Haipeng Wang
(School of Computer Science, Northwestern Polytechnical University, Xi'an, P. R. China
haipeng.wang@gmail.com)

Abstract: As multimedia contents are becoming widely used in ubiquitous computing environments among many application fields, e.g., education, entertainment, and live surveillance, the demand of personalized access to these contents has increased dramatically. The provisioning of ubiquitous personalized multimedia services (UPMSs) is a challenging task, which involves a lot of heterogeneous entities ranging from objects, devices to software. In this work, we propose a three-layer software platform, called UPmP to support efficient development and deployment of UPMSs. It fulfills the core functionalities for ubiquitous personalized multimedia provisioning including service management, multimedia recommendation, adaptation, and delivery. We adopt service-oriented approach in building the platform. The enabling technologies such as component representation, service lifecycle management, platform configuration, and service composition are described in detail. The experimental results show that the UPmP is flexible to be configured under different settings.

Keywords: ubiquitous multimedia, personalization, platform, service-oriented, service management, service composition, platform configuration
Category: H.3.1

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 has been widely used in ubiquitous computing environments among many application fields, e.g., education, entertainment, and live surveillance. The choices of multimedia content offered to the user can be quite overwhelming. To quickly and effectively provide the right content

from large amounts of media information, in the right form, to the right person, we need to customize multimedia content based on the user's preferences and his current contextual information, such as time-of-day, user location, and device capabilities [Yu et al, 06a]. Such service is so-called ubiquitous personalized multimedia services (UPMSs).

The provisioning of UPMSs is a challenging task. First, UPMSs are highly dynamic. They are likely to be adapted at run time according to the current changing context. Second, delivering UPMSs involves a lot of heterogeneous entities ranging from objects, devices to software. The entities interact and service one another to complete complex tasks. The interoperability is a big problem because these entities may originate from different sources and therefore use different ways to present their capabilities and connectivity requirements. The two issues can be addressed by using service-oriented architecture.

Service-oriented architecture (SOA) is a software architectural concept that defines the use of services to support the requirements of entities (McGovern et al, 03). In a SOA framework, entities in the environment are represented in form of services and made available to other entities in a standardized way. As the functions of every entity are described using common convention, entities can thus understand each other and collaborate to achieve a certain goal.

In this paper, we propose a software platform, namely UPmP (**U**biquitous **P**ersonalized **m**ultimedia **P**latform) that enables UPMSs to be achieved easily and systematically. It takes the service-oriented system architecture and contains a number of collaborating components. 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 allows 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.

The rest of this paper is organized as follows. Section 2 introduces the three-layer architecture of the UPmP. Section 3 presents a hierarchical model to organize service components and a description language, namely SCDL based on XML to describe them. Section 4 gives the ubiquitous personalized multimedia service lifecycle management model. Section 5 describes an XML-based platform configuration language (XPCL). Section 6 illustrates the service composition under different conditions. The implementation and experiment results are presented in Section 7, followed by Section 8 with related work. Finally, we conclude in Section 9.

2 UPmP Architecture

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

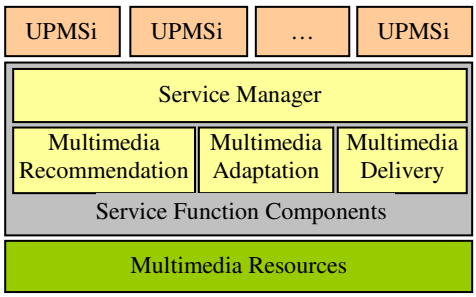


Figure 1: UPmP architecture

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 (MPEG7, 03) 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.

Multimedia content and metadata are stored in two distinct databases. A file repository is used for storing media content. The content repository stores media of various kinds of media modalities, file formats, bit-rates and resolutions. An XML database Xindice (Xindice, 03) is used for storing media metadata (MPEG-7 XML files) because of its efficient querying.

The multimedia resources layer can integrate varied multimedia repositories 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 (Thorne and Sim, 05). The O.K.I Repository Specification has been used to successfully integrate several applications with multiple content repositories.

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 content is just downloaded.

2.3 Service Instances

The service instances are concrete ubiquitous personalized multimedia services requested by a wide range of devices in ubiquitous environments. A service usually proceeds as follows:

(1) A user logs on the system, and sends a request for personalized multimedia. The request contains user preferences, terminal capabilities, and network static characteristics.
(2) Multimedia objects are ranked according to the user preferences and the recommendation list is sent to the user.
(3) The user selects one item from the list.
(4) The system determines the appropriate presentation form of the content according to terminal capabilities and network static characteristics. If the desired variation is not existed, it will be generated.
(5) The multimedia object in right form is streamed or downloaded.

3 Component Representation Model

We adopt component-based approach to represent software entities in our system. 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 (Szyperski, 98). Three major component models are presented and used successfully today: COM (COM, 03), CORBA (CORBA, 00), and Javabeans (JAVA, 03).

Service 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 and 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.

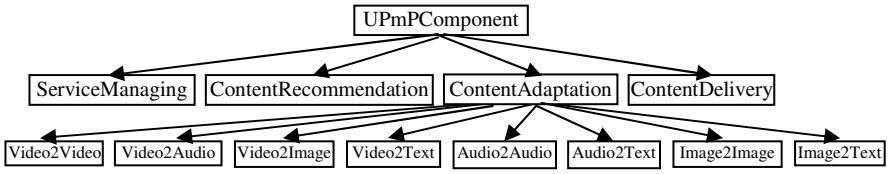


Figure 2: Component hierarchy

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 the 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 Figure 2. For space consideration, we here merely present the detailed structure of ContentAdaptation component.

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. Figure 3 shows the DTD (Document Type Definition) structure of SCDL. 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 components. The components with the same parent class are regarded as brother components. For instance, Video2Image and Video2Text are brother components.

```
<!-- Top level element -->
<!ELEMENT SComponent (SComponentDescription)>
<!ELEMENT SComponentDescription (GeneralInformation, PublicInterface, CompositionLogic)>

<!-- GeneralInformation declaration -->
<!ELEMENT GeneralInformation (Name, Type, Annotation, ParentClass)>
<!ELEMENT Name(#PCDATA)>
<!ELEMENT Type(#PCDATA)>
<!ELEMENT Annotation(#PCDATA)>
<!ELEMENT ParentClass(#PCDATA)>

<!-- PublicInterface declaration -->
<!ELEMENT PublicInterface (Input, Output, Executors)>
<!ELEMENT Input (Format)*>
<!ELEMENT Format (#PCDATA)>
<!ELEMENT Output (Format)*>
<!ELEMENT Format (#PCDATA)>
<!ELEMENT Executors (Executor)*>
<!ELEMENT Executor (#PCDATA)>

<!-- CompositionLogic declaration -->
<!ELEMENT CompositionLogic (Following, FollowedWith, CombinedWithBrotherComp)>
<!ELEMENT Following (Type)*>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT FollowedWith (Type)*>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT CombinedWithBrotherComp(#PCDATA)>
```

Figure 3: The DTD structure of SCDL

Ready When a multimedia object is assigned to a service, it jumps to the *Ready* state, and is ready for scheduling. If a service is selected to run, it jumps to the *Running* state. If it is ended by the user or the service provider, the service will enter the state of *Completed*.

Running Service in the *Running* state means media streaming or file downloading. In the process of running, (i) if the service finishes successfully or is broken by the user or service provider, it jumps to the *Completed* state; (ii) if network available bandwidth decreases and makes the multimedia delivering impossible, the service enters the *Blocked* state.

Figure 4 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 or 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 can be followed by the components of ContentDelivery in composing a service. However, it cannot be combined with its brother components.

```
<?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>
```

Figure 4: Component metadata (example)

4 Service Lifecycle Management Model

We deploy the FSM (Finite State Machines) to represent and manage service state of our ubiquitous personalized multimedia services. A ubiquitous personalized multimedia service (UPMS) is modeled as a deterministic FSM.

Definition 1 (UPMS State Machine), A UPMS State Machine is a 5-tuple UPMS=(S, I, f, s_0, F):

- S is a finite set of service states;
- I is a finite set of elements, which is defined in Definition 2, each element is an input to the state machine;
- f: $(S \times I) \rightarrow S$ is the transition function;
- $s_0 \in$ S is the initial state;
- F $\subset$ S is a finite set of final states.

Definition 2 (Input of UPMS State Machine, I), I is a finite set of elements as inputs to the state machine. Each element is a 2-tuple (R, E). *R* represents the reason why the event *E* takes place. In each element, *R* can be NULL, while *E* cannot be NULL.

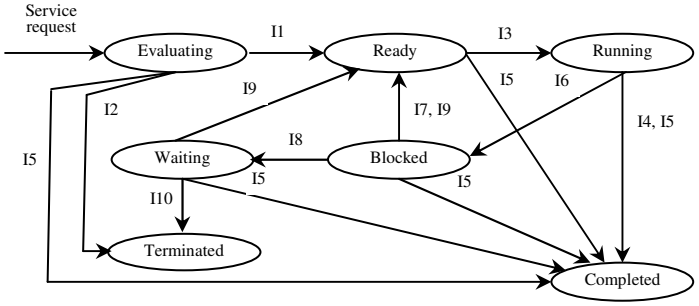


Figure 5: UPMS State Machine Model

Figure 5 shows the state machine model and complete state transition of UPMS. A service request from terminal user brings a new UPMS into birth. In its lifecycle, a UPMS could be in several states, represented by ellipses in the figure. The arrows between states represent state transitions, and corresponding character strings (I1, …, I10) stand for conditions of the transitions.

The element values of UPMS State Machine are presented in Table 1.

The states in UPMS State Machine are described in detail as follows.

Evaluating When a client sends a request to the server for multimedia service, the service is in the *Evaluating* state. In this state, the server selects or generates a multimedia object according to user preferences, terminal capabilities, and network characteristics: (i) if there exist media content satisfying the three requirements, the service enters the state of *Ready* directly; (ii) if there is a content meeting the user's preference, but not suitable for the device and network, the system first performs adaptation for the content, and then make the service be *Ready*; (iii) if there is no content that can satisfy the user's preference, the service jumps into the *Terminated* state; (iv) if the user or the service provider stops the service, the service will be *Completed*.

Element	Value
S	{Evaluating, Ready, Running, Blocked, Waiting, Completed, Terminated}
I	{I1, I2, I3, I4, I5, I6, I7, I8, I9, I10} I1=(Satisfy user preferences, device capabilities, and network static characteristics, Join()) I2=(Cannot satisfy user preferences, device capabilities, and network static characteristics, Discard()) I3=(NULL, Schedule()) I4=(Service successfully finishes, Finish()) I5=(User or service provider stops the service, Finish()) I6=(Cannot transfer for network bandwidth changes, Block()) I7=(Can transfer after transcoding, WakeUp()) I8=(Cannot transfer after transcoding, LayAside()) I9=(Network bandwidth is satisfied for delivery, WakeUp()) I10=(Time over, Discard())

f	f(Evaluating, I1)=Ready;
	f(Evaluating, I2)=Terminated;
	f(Evaluating, I5)=Completed;
	f(Ready, I3)=Running;
	f(Ready, I5)= Completed;
	f(Running, I4)=Completed;
	f(Running, I5)=Completed;
	f(Running, I6)=Blocked;
	f(Blocked, I7)=Ready;
	f(Blocked, I9)=Ready;
	f(Blocked, I8)=Waiting;
	f(Blocked, I5)=Completed;
	f(Waiting, I9)=Ready;
	f(Waiting, I5)=Completed;
	f(Waiting, I10)=Terminated

s_0	Evaluating
F	{Completed, Terminated}

Table 1: UPMS state machine element values

Blocked When a service is in the *Blocked* state, it continuously monitors the network bandwidth (Fang et al, 2005), and makes transcoding for the multimedia object; (i) if the adjusted object can be delivered in the current network condition, then the service is woken up, and put into the *Ready* state; (ii) before the transcoding finishes, the bandwidth changes to be sufficient for delivering the original version, the service enters the *Ready* state immediately; (iii) whatever trancoding is made, the object cannot be delivered in the current network condition, then the service jumps to the *Waiting* state; (iv) if the user or the service provider stops the service, the service will be *Completed*.

Waiting When a service is in the *Waiting* state, (i) the network bandwidth increases and is sufficient for the object to be delivered, then the service is woken up, and be put into the *Ready* state, i.e., waiting for another scheduling; (ii) the preset waiting time is over, then the service enters the *Terminated* state; (iii) if the user or the administrator ends the service, then the service enters the *Completed* state.

Completed The service successfully finishes or is stopped by the user or service provider. It releases all of its resources (such as service identifier, priority, etc.).

Terminated The function is similar to the *Completed* state. The difference is that the *Terminated* state means a service abends due to resource dissatisfaction (e.g., multimedia, terminal capabilities, and network).

According to the definition of state and its transition, there are three service queues in the system, ready queue, blocked queue, and waiting queue. Selecting services in the blocked or waiting queues into the ready queue is simply complying the FCFS (First Come First Serve) principle. To select a service in the ready queue for running is a little complex. Our scheduling mode is Non-Preemptive based on dynamic priority. It is invoked at two occasions: (1) a service enters the queue of ready services, or (2) a specific running service enters the state of *Completed* or *Blocked*. We define that the larger the priority is, the earlier the service will run. The determination of priority conforms to two principles: FCFS and BWFS (Blocked or Waiting First Serve).

5 Platform Configuration

The UPmP platform offers a set of optional functionalities, which can be switched off at the platform initialization 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 supports 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.

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 user 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 Figure 6.

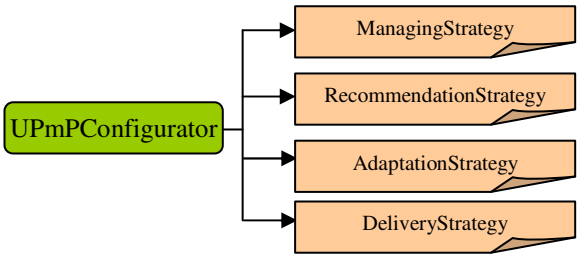


Figure 6: UPmP configurator

For purpose of configuration at initialization 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. Besides, several XML query languages, e.g. XQuery (Boag, 05) and XPath (Clark and DeRose, 99) can be used to access and lookup the XML-based configuration file. 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>, which specify whether the services enter *Blocked* or *Waiting* state respectively. 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>
```

Figure 7: Platform configuration (example)

Figure 7 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.

6 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 mainly 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.

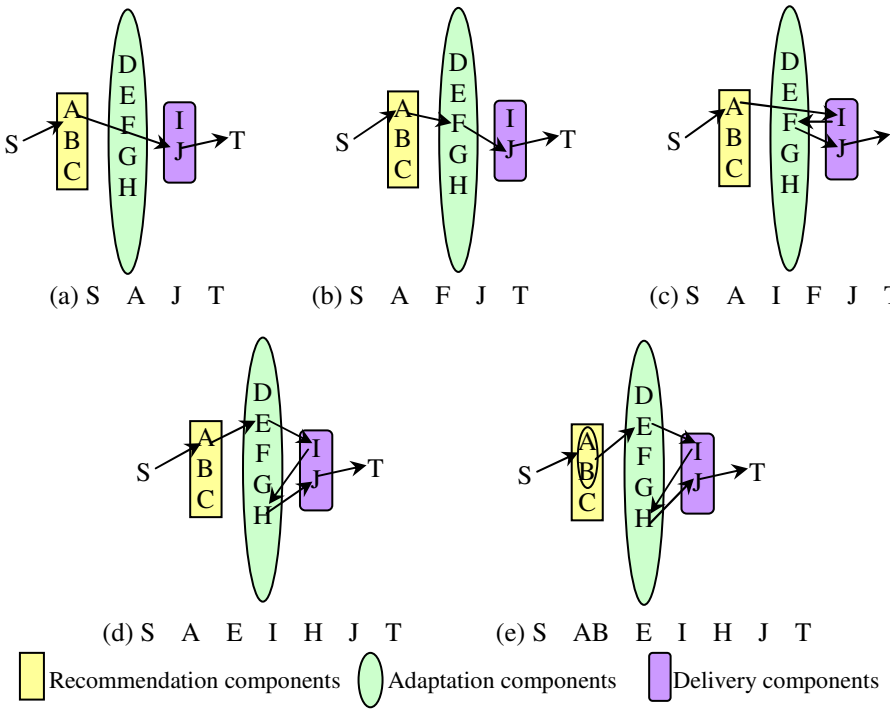


Figure 8: Service composition examples

Figure 8 illustrates several 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 (e.g, Rule-based Recommendation, Content-based Recommendation, and Collaborative Recommendation). The oval contains adaptation components D, E, F, G, and H (e.g, Video2Image, Video2Audio, Video2Text, Image2Text, and Audio2Text). The delivery components I and J (e.g., Video/Audio Streaming and Image/Text Downloading) are represented in the rounded rectangle. In Figure 8a, since the appropriate variation already exits, the service goes directly to deliver it. In Figure 8b, the service firstly performs video-to-text transcoding, and then downloads the text. In Figure 8c, 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 Figure 8d, 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 and last sends the text. In Figure 8e, component A and B are combined for the recommendation.

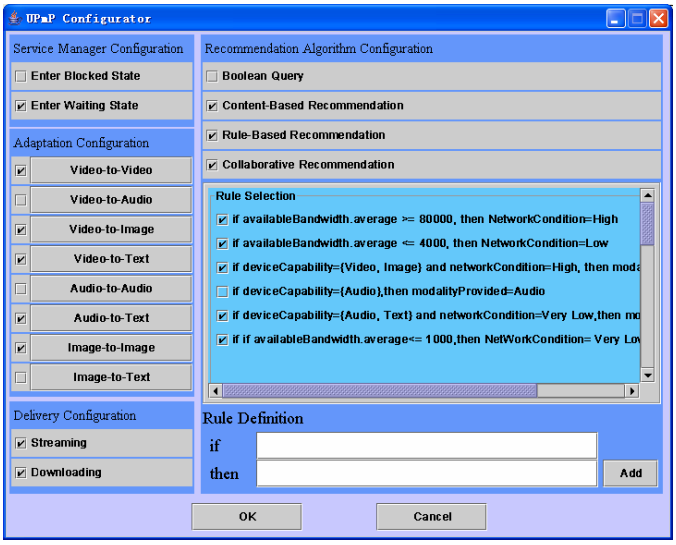


Figure 9: Main interface of UpmPConfigurator

7 Implementation and Experiment

7.1 Prototype Implementation

We have implemented a prototype of UPmP. The visual platform configuration tool UPmPConfigurator was implemented in Java Swing. The components were all implemented in Java and built as .jar files. Some media adaptation components were integrated from third party. The service components were deployed in a network of computers, which improved the performance of the system in terms of throughput and scalability. To handle component dependency during platform configuration and

service composition, we utilized the Dependency Injection (Fowler, 04) pattern provided by the Spring framework (Spring, 04). Figure 9 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 (Yu et al, 06b, Yu et al, 08). The developers can directly choose some existing rules or define specific rules of their own.

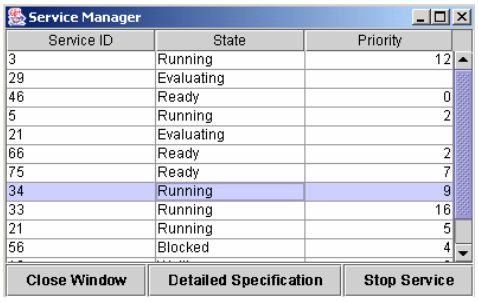


Figure 10: Service manager interface

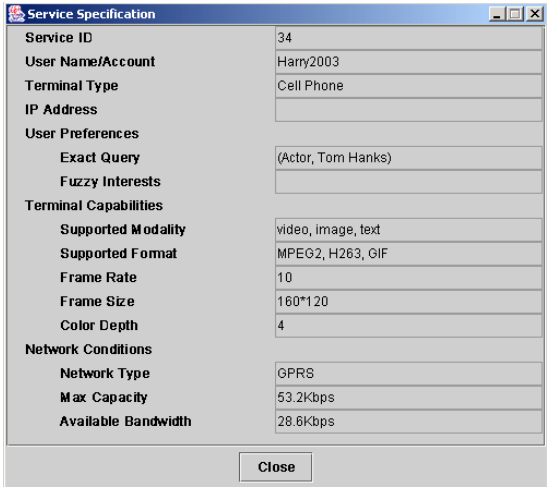


Figure 11: Service detailed specification

Figure 10 shows the main interface of service manager. It provides basic information of the services including service ID, state, and priority. When the service provider wants to break off a specific service, he can select it and click the "Stop Service" button to stop it. Once a service is selected and the button of "Detailed

Specification" is clicked, details of the service will be showed (see Figure 11). In this dialogue, service information, such as user name/account, terminal type, IP address, user preferences, terminal capabilities, as well as network conditions, are displayed in detail.

7.2 Experimental Results

We mainly evaluated 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 was conducted on a PC with 1.6 GHz Pentium 4 CPU, and 1 GB RAM running Windows XP. There were five different setup configurations involved in this experiment. The configuration details are presented in Table 2. 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 selected all the transcoders of it.

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

Table 2: Configuration details

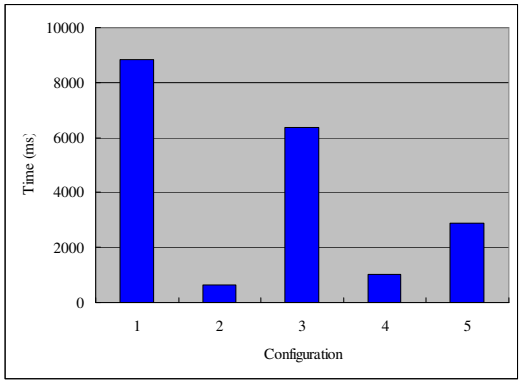
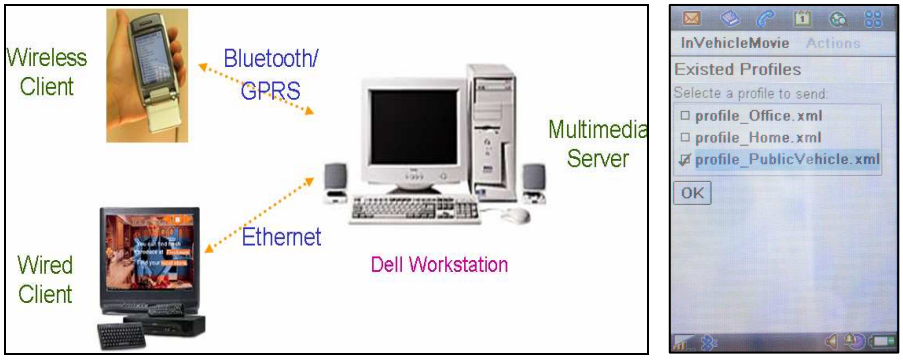


Figure 12: Overhead of UPmP configuration

The experimental results are shown in Figure 12, in which the time for each configuration is an average value of 10 runs. The completed configuration (loading and linking all components) 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. The time spent on the loading and liking service managing and recommendation components is merely 0.6 s. Also the configuration of content delivery components takes only about 1 second. The real running system configuration, Configuration 5 costs 2.9 s, which is acceptable. In general, the overhead of platform configuration 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.



(a) (b)

Figure 13: Application: (a) deployment; (b) client user interface

7.3 Application

Based on the UPmP platform, we developed a ubiquitous movie recommender. It can recommend interesting movies to users according to their preferences, devices, networks, and locations (e.g., office, home, and public vehicle) with the infrastructure support of the UPmP. Figure 13a illustrates the application deployment. The UPmP platform is deployed on a Dell workstation serving as a multimedia server. The application runs at both server and client sides. There are two types of clients, wireless mobile phone and wired PC. The mobile phone connects to the server through Bluetooth and GPRS, while the PC uses Ethernet. Figure 13b shows a user interface on the mobile phone. Through it users can select different environmental profiles. Then the system performs recommendation based on the user profile and current context, such as device capability, network connection and, activity (accompanying parents at home or being alone in public vehicle). We mainly evaluate the application from user's perspective by analyzing user feedback, e.g., whether the users were satisfied with the recommended content, whether they were satisfied with the presentation form, and whether they were satisfied with the time latency. The results showed that the recommendation precision values were around 0.7 (i.e., most

of the content recommended were interesting for the users), the users were generally satisfied with the movies and their presentations, and they found the time latency acceptable although there was some delay in content provisioning. The application also verifies the effectiveness of our infrastructure.

8 Related Work

A number of service-oriented systems have been proposed for multimedia services. ISIS (Mlivoncic et al, 05) is a service-oriented grid infrastructure for multimedia management in digital libraries. All functionalities are encapsulated by services, such as storage services and feature extraction services that are used to compose the ISIS applications. The CASSANDRA Framework (Lange et al, 07) is a modular, real-time and distributed media processing system that supports cooperation of connected consumer electronics devices through a service oriented approach. The USON framework (Takemoto et al, 02) is a service provision framework for a ubiquitous computing environment based on P2P technology. The system automatically chooses and uses suitable devices among the ubiquitously surrounding terminals. Although the above mentioned systems achieve flexible content provision by using service-orientated approach, they did not support multimedia personalization in ubiquitous computing environments. Kohncke and Balke (Kohncke 06) proposed to adapt multimedia content to the terminal capabilities of the user by integrating MPEG-21 metadata with a complex preference framework. They use Web services as basic modules for building multimedia applications. The European Daidalos project (Yang et al, 05) aims at building a service-oriented architecture to personalize services in pervasive environments, such as tailoring content, services and interfaces. The basic idea of these two works is similar to ours. But our system provides more capabilities such as service lifecycle management, service composition and platform configuration.

There has been much research work done specifically to provide systematic architectural support for ubiquitous multimedia delivery. Gamma (Lee et al, 03) 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 (Fu et al, 01) 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 (Ooi et al, 99) 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 (Wichadakul et al, 02) 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 (Davidyuk et al, 04) 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 service-oriented architecture.

Kalasapur et al (05) built adaptive multimedia services in heterogeneous wireless networks by exploiting a general middleware for pervasive computing. There is no sufficient support of content adaptation and recommendation with their system.

Recently, to deliver personalized multimedia to ubiquitous devices, some researchers (e.g., Onur and Alatan, 07; Rho et al, 05; Steiger et al, 03; Lemlouma and Layaida, 03; Belle et al, 02) have considered both user preference and device/network capability to generate appropriate presentation to terminals. However, none of them are proposed from the middleware perspective.

9 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 service-oriented software platform architecture; (2) introducing a component representation model that is helpful for component organization, indexing, and description; (3) presenting a service lifecycle management model; (4) designing a configuration tool and an XML-based platform configuration language; (5) illustrating service composition under different conditions.

Even though our UPmP platform supports ubiquitous personalized multimedia provisioning quite well and is flexible to be configured under different settings, there is still much work to do to develop different applications, e.g., healthcare, entertainment, and e-learning using the platform and improve the design. We also plan to extend the platform to support re-configuration for adapting to new changes at running time.

Acknowledgements

This work was partially supported by the High-Tech Program of China (863) (No. 2009AA011903), the National Natural Science Foundation of China (No. 60903125, 60803044), and the Program for New Century Excellent Talents in University (No. NCET-09-0079).

References

[Belle, 02] Belle, L.T., Lin, C.Y., and Smith, J.R.: Video Summarization and Personalization for Pervasive Mobile Devices. SPIE Electronic Imaging 2002 - Storage and Retrieval for Media Databases, San Jose, CA, 359-370, 2002

[Boag, 05] Boag, S., Chamberlin, D., Fernandez, M. F., Florescu, D., Robie, J., and Simeon, J.: XQuery 1.0: An XML Query Language, W3C Recommendation, 2005

[Clark, 99] Clark, J. and DeRose, S.: XML Path Language (XPath), W3C Recommendation, 1999

[COM, 03] COM, http://www.microsoft.com/com, 2003

[CORBA, 00] CORBA, http://www.omg.org/corba, 2000

[Davidyuk, 04] Davidyuk, O., et al.: Context-aware middleware for mobile multimedia applications. The 3rd International Conference on Mobile and Ubiquitous Multimedia, 213-220, 2004

[Fang, 05] Fang, Q., Jia, W., and Wu, J.: Available Bandwidth Detection with Improved Transport Control Algorithm for Heterogeneous Networks. The Third International Workshop on Mobile Distributed Computing (MDC), in conjunction with ICDCS'05, 656-659, 2005

[Fowler, 04] Fowler, M.: Inversion of Control Containers and the Dependency Injection pattern. http://www.martinfowler.com/articles/injection.html, 2004

[Fu, 01] 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

[JAVA, 03] JAVA, http://java.sun.com, 2003

[Kalasapur, 05] Kalasapur, S., Kumar, M., and Shirazi, B.: Personalized Service Composition for Ubiquitous Multimedia Delivery. The Sixth IEEE International Symposium on World of Wireless Mobile and Multimedia Networks (WoWMoM'05), 258-263, 2005

[Kohncke, 06] Kohncke, B. and Balke, W.T.: Personalized Digital Item Adaptation in Service-Oriented Environments. The First International Workshop on Semantic Media Adaptation and Personalization (SMAP'06), 2006

[Lange, 07] Lange, F. et al: CASSANDRA Framework: A Service Oriented Distributed Multimedia Content Analysis Engine. Eight International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS'07), 2007

[Lee, 03] Lee, Y.W., Chandranmenon, G., and Miller, S.C.: Gamma: A Content-Adaptation Server for Wireless Multimedia Applications. Lucent Technologies white paper, 2003

[Lemlouma, 03] Lemlouma, T., and Layaida, N.: Encoding Multimedia Presentations for User Preferences and Limited Environments. IEEE ICME, 165-168, 2003.

[McGovern, 03] McGovern, J., Tyagi, S., Stevens, M., Mathew, S.: Service Oriented Architecture, Chapter 2, Java Web Services Architecture, The Morgan Kaufmann Series in Data Management Systems, July 2003.

[Mlivoncic, 05] Mlivoncic, M. et al.: A Service-Oriented Grid Infrastructure for Multimedia Management and Search. In: Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures: 6th Thematic Workshop of the EU Network of Excellence, 167-187, 2005

[MPEG7, 03] MPEG7, http://www.chiariglione.org/MPEG/standards/mpeg-7/mpeg-7.htm, 2003

[Onur, 07] Onur, O.D. and Alatan, A.A.: Video Adaptation Based on Content Characteristics and Hardware Capabilities. The Second International Workshop on Semantic Media Adaptation and Personalization (SMAP'07), 15-20, 2007

[Ooi, 99] Ooi, W.T., et al.: Dali: A Multimedia Software Library. Proceedings of 1999 SPIE Multimedia Computing and Networking, 264-275, 1999

[Rho, 05] Rho, S.M., Cho, J.W., and Hwang, E.J.: Adaptive Multimedia Content Delivery in Ubiquitous Environments. WISE 2005 Workshops, 43-52, 2005

[Spring, 04] Spring Framework, http://www.springframework.org/, 2004

[Steiger, 03] Steiger, O., Ebrahimi, T., and Sanjuan, D.M.: MPEG-based Personalized Content Delivery. IEEE Intl Conf. on Image Processing, 45-48, 2003

[Szyperski, 98] Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley, Reading, Mass, 1998

[Takemoto, 02] Takemoto, M. et al.: The Ubiquitous Service-Oriented Network (USON)—An Approach for a Ubiquitous World Based on P2P Technology. In Proc. of International Conference on Peer-to-Peer Computing, 2002

[Thorne, 05] Thorne, S., and Sim, S.: Integrating Applications with Repositories Using the O.K.I Repository OSID. JA-SIG Conference, 2005

[Wichadakul, 02] Wichadakul, D., Gu, X.H., and Nahrstedt, K.: A Programming Framework for Quality-Aware Ubiquitous Multimedia Applications. ACM Multimedia 2002, 631-640, 2002

[Xindice, 03] Xindice, http://xml.apache.org/xindice/, 2003

[Yang, 05] Yang, Y. et al: A Service-Oriented Personalization Mechanism in Pervasive Environments. In Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05), 2005

[Yu, 06a] Yu, Z.W., Zhou, X.S., Zhang, D.Q., Chin, C.Y., Wang, X.H., and Men, J.: Supporting Context-Aware Media Recommendations for Smart Phones, IEEE Pervasive Computing, Vol. 5, No. 3, pp. 68-75, July-September, 2006

[Yu, 06b] Yu, Z.W., Zhang, D.Q., Zhou, X.S., et al.: An OSGi-Based Infrastructure for Context-Aware Multimedia Services, IEEE Communications Magazine, Vol. 44, No. 10, pp. 136-142, 2006

[Yu, 08] Yu, Z.W., Nakamura, Y., Zhang, D.Q., et al.: Content Provisioning for Ubiquitous Learning, IEEE Pervasive Computing, Vol. 7, No. 4, pp. 62-70, October-December 2008