Journal of Software Engineering and Applications, **2012, 5, 883-893**
http://dx.doi.org/10.4236/jsea.2012.531103 Published Online November 2012 (http://www.SciRP.org/journal/jsea)

An Ubiquitous Service Mobility Mechanism in the Cross Domain Service Framework

Jung-Sik Sung, Jongmoo Sohn, Yoon-Sik Yoo, Jaedoo Huh

Green Computing Department, ETRI, Daejoen, Korea.
Email: jssung@etri.re.kr

Received October 8th, 2012; revised November 6th, 2012; accepted November 16th, 2012

ABSTRACT

Service mobility has become a new issue in the area of service convergence with the advent of versatile mobile devices. Hence, we propose an open service framework with support for service mobility that executes contents and applications in a dynamic environment. For the framework, the concept and role of a device and its application for a service are redefined into the new concepts of device, execution engine, and service. Mechanisms for device profiling, user preference learning, and profile-based service recommendation are designed to enable device-capability-aware service recommendation functionality. Furthermore, a seamless service syndication mechanism is added for continuity and synchronization of service upon change of terminal, network status, or personal preference. A prototype system integrates functionalities with proprietary protocol and a content transcoder to support multiple device platforms. The prototype shows the capability of the framework in service mobility support and its advancement into an open international application platform.

Keywords: Open Service Framework; Service Mobility; Service Synchronization; Profiling; Service Recommendation

1. Introduction

Beyond the convergence of simple networks, convergence services that accept various contents, application programs, service platforms, mobile devices, PCs, TVs, and so on are beginning to earn interest. Current convergence services focus more on network convergence, where heterogeneous devices and services are provided by a single network. On the other hand, device convergence provides devices that support heterogeneous services over several networks such that the services are provided using a device's capability of changing networks seamlessly. Recently, the convergence of services with content streaming over multiple network domains and devices using the unique characteristics of ubiquitous devices, such as 3-screen play [1,2], are entering the spotlight. The design of a convergence service should address mobility, heterogeneity, and user-centric issues [3]. There are two types of mobility: terminal and service mobility. Terminal mobility, as the classical part of mobility management, places devices at the endpoints of communication, and focuses on connection continuity over a change of access points [4]. Therefore, in order to provide consecutive communication, location and handoff of a mobile terminal is managed at four different layers of the International Organization for Standardization's (ISO) Open Systems Interconnection (OSI) model, namely, at the link, network, transport, and session layers. It is well known that the representative research on network layer terminal mobility is mobile IP [5]. Service mobility, on the other hand, is considered as maintaining a connection even when terminals or networks are changed due to user movement or personal preference [4]. Focusing on service mobility under user movement and heterogeneous devices, a major problem with this service convergence is to build a platform that is applicable to services supported by heterogeneous service platforms and devices with their own platforms. A platform for convergence services plays the role of an infrastructure to execute content and application programs smoothly without obstacles, and provide interoperability between devices and services. We propose an open service platform to provide the facilities stated above.

The open service framework presented in this paper is a platform that supports convergence services on devices, such as Windows PCs, Linux PCs, mobile and smart phones, and the service platforms of each device. The framework recommends the best available service for the user and device, and then constructs and executes a service execution engine for the selected service on the connected device. As the user changes the device due to movement or personal preference, the service is provided continuously with transformed content suitable for the

new device.

Starting with Section 2, where various service mobility architectures are revised, we go on to present our open service framework in Section 3, where the features of the OSCD service framework to support convergence service including various service platforms and service mobility are described in detail. The prototype implementation used to verify the feasibility of the proposed service framework is described in Section 4. And finally, we summarize and conclude the paper in Section 5.

2. Related Works

There have been several studies on service mobility with user migration. The authors in Ref. [6-8] have addressed the issue of integration of heterogeneous networks. The authors in Ref. [6] and Ref. [7] have proposed the Mobile People Architecture (MPA), which aims to put a person, rather than the devices the person uses, at the endpoints of a communication session. MPA enables users to contact each other from anywhere, using a variety of communication media such as email, telephones, and ICQ messages. MPA uses a globally unique Personal Online ID to identify a user, and utilizes the personal proxy located at the user's home network for location tracking. In this architecture, the personal proxy achieves the function of personalized and continuous service. The ICEBERG project [8] aimed to integrate cellular telephony networks with the Internet. Just like MPA, ICEBERG had the view that people will continue to use multiple devices and networks for communication. As the ICEBERG network is an overlay network of iPOPs on top of the Internet, it needs a modification of existing network components. MPA and ICEBERG emphasize providing contactability and reachability rather than a user-centric personal operating environment.

Mobile agent based frameworks [9,10] were proposed to provide personal mobility in accessing Internet services. NetChaser [9] supports three Internet services, namely, Web, e-mail, and FTP using four assistants: user, HTTP, mail, and FTP assistants. Assistants operate at a proxy server close to the user. NetChaser traces the user's network location and records the history of service, so as to offer users the ability of maintaining service in spite of the change of terminal identity. NetChaser supports a personalization scheme only. In order to provide true personal mobility that requires the integration of contactability and personalization, Ref. [10] suggested an Integrated Personal Mobility Architecture (IPMoA). IPMoA is a mobile agent based personal mobility framework that utilizes mobile agents. There are three agents: personal application assistant (PAA), personal file assistant (PFA), and personal communication assistant (PCA). These three assistants migrate to the visiting network's agent execution environment. These mobile agent based frameworks are organized similarly to how home agents and foreign agents are organized in Mobile IP, since they also use two subnets to locate and trace the user in this model. Therefore, the session establishment may take longer than a direct connection. Furthermore, the application time may also be longer than executing the applications on a local machine.

One important thing in service mobility is to keep the service session even when terminals or networks are changed due to user migration or personal preference. Maintaining the service session means providing people with seamless and continuous service when they change networks or terminals as they migrate. Also, it provides service synchronization by managing personal-service-specific information in a heterogeneous service platform. However, previous works have not addressed this session mobility issue

3. One-Service-Cross-Domain Service Framework

We propose a one-service-cross-domain (OSCD) service framework as an open service framework to support convergence services. It is a technology to generalize a ubiquitous computing environment by providing an environment that eases execution and combination of domain-subordinated services and/or contents by organically integrating independent service domains. Therefore, the OSCD service framework is an optimized integrated service framework that provides continuous services that are not constrained by physical user environments such as multimedia transmission, integrated information management and preservation, educational broadcasting, and games. **Figure 1** shows the structure of the OSCD service framework.

The OSCD service framework supports the registration of service and execution engines using profiling models, and performs automatic detection of a suitable service execution engine. The framework also supports a search function for services that are suitable for the user and the targeted terminal device. Moreover, it supports real-time content adaptation for targeted terminals, semantic translation including a communication protocol translation, and seamless service continuity so that a user can continue using a service across different terminals.

3.1. Profile Modeling

In order to provide the best service depending on a particular user context, support for flexible management and a decision process based on an accurate and clear description of contextual information is required. In the proposed system, a particular context is divided into three conceptual domains: terminal device, service, and

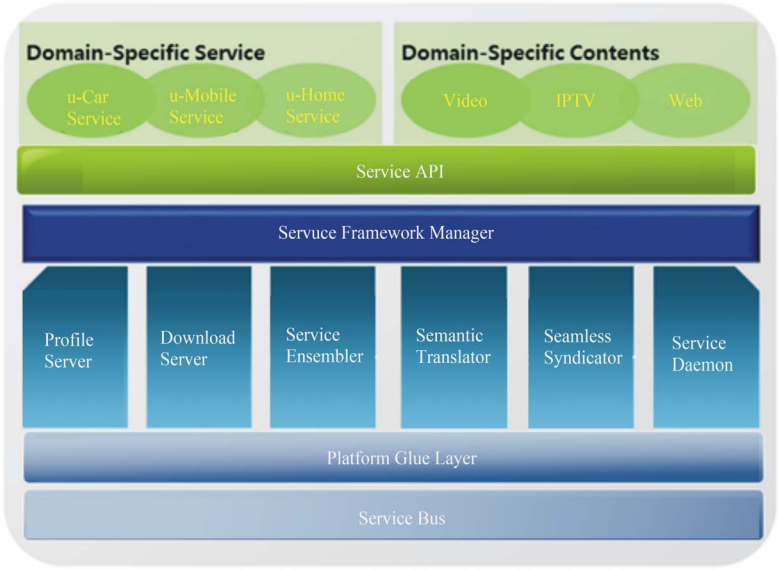

Figure 1. The proposed service framework.

execution engine, which provides a running environment for the service. Each domain is described by the name of the profile.

- *Device Profile*. The device profile describes the currently available resources and status of the terminal device available for running execution engines. The resource and status are represented by sets of terminal device attributes including specifications of hardware/software/network characteristics and data gathered by sensors in the device.
- *Service Profile*. Each service in the proposed system has a service profile to describe its attributes. A service profile implicitly declares the minimum requirement of the execution engine for proper delivery of service.
- (*Execution*) *Engine Profile*. The execution engine, software that runs on the terminal device to deliver a service to the end user, has device resource requirements and service attributes the engine can provide. The engine profile implicitly declares the minimum resource requirements of the device and the functional specifications of the engine.

Each profile is described by a proprietary description scheme similar to UAProf [11], a concrete implementation version of CC/PP [12] based on RDF [13]. Each profile, according to the representation scheme of CC/PP, contains one or more components, where each component consists of sets of attribute-value pairs where attribute/component names, data value types, and their resolution types are defined and constrained by the vocabulary and schema. Attributes in the implemented vocabulary are categorized into five components by their logical relativity: hardware, software, network characteristics of terminal device, multimedia, and network capability of service. Attribute value types are defined in the schema, and include Boolean, number, literal, literal-bag (set of literal arguments without sequence), literal-seq (set of literal arguments with sequence), dimension, and range.

The proposed system needs contextual information to be delivered from a device to a server upon the server's request only when a user is logged into the designated server using the user's terminal device. Since the delivery is always initiated by the server, self notification of changed context as in SLP [14], Jini [15], and uPnP [16] are not required. Moreover, since user preference information and service/engine profiles are always located at the server side, it leaves the device profile as the only contextual information that needs to be delivered.

The device profile is presented by a combination of reference profile Unified Resource Identifier (URI) and/or profile fragments with static/dynamic attributes. While URI and static attributes are required to be delivered only once at the initial phase, dynamic attributes should be delivered at any time upon a server's request in order to reflect a change of device status. Therefore, unlike HTTP or Wireless Session Protocol (WSP)-based profile exchange mechanisms [17,18] used in CC/PP and UAProf, a new delivery mechanism supporting both partial and on-demand profile delivery is proposed. The proposed delivery mechanism separates on-the-run profile delivery from the initial static/dynamic profile delivery. Hence, only the updated status is transferred to the server at the minimum network cost by sending a profile fragment containing updated attribute values only. **Figure 2** illustrates the device profile delivery scheme for the proposed system.

The fact that all profiles in the proposed system use a

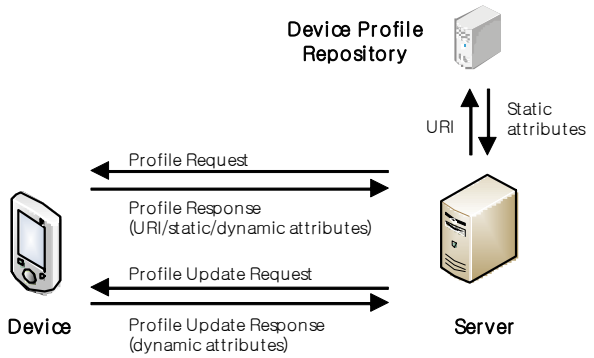

Figure 2. Profile delivery in the proposed system.

common description scheme based on a single schema, lead to the use of a single modeling mechanism, it enables the implementation of a simple comparator for simple reasoning and decision processes. Each profile is processed into an RDF-based model with functions for merging and updating profile fragments and dynamic attribute values, using Jena [19], a Java framework for semantic Web applications.

Based on the fact that all profiles are modeled into a common RDF based model, a simple profile comparator is implemented for simple reasoning and decision processes based on the modeled profiles without any additional semantic/ontology techniques such as OWL [20], SWRL [21], or Jess [22]. The implemented comparator simply decides whether two profiles comply with a given comparison rule. Comparison rules are described in the form of the schema used in the profile description and modeling, and specifies the attributes to compare, the comparison policies, conditions, and whether or not to store the difference of two attributes in case the condition/policy is not satisfied. The comparison condition specifies whether the rule is "mandatory" or "optional", and the policy specifies how two attribute values such as "match", "included" and "within range" are compared. The comparator outputs the final comparison results including the compliance with the comparison rule, quantified score, and the difference of attribute values in XML format. The result is used in the following reasoning or other decision procedures.

3.2. Dynamic Configuration of Execution Engine

An execution engine, defined as software that runs on a user device to mediate a particular service to the end user, is provided dynamically for the end user's device on the user's connection to the OSCD service framework. This execution engine conventionally is run directly by the user, assuming it is already installed on the user device. However, using the dynamic service execution environment provision function of the proposed framework, the user simply selects one of the recommended services, and the according execution engine is then downloaded automatically/dynamically forming an optimized service execution environment. The dynamic configuration of a service execution environment includes the following procedures: user device profiling, a search for user and device specific services, execution engine search procedure to find an engine that suites both the service selected by the user and the user device profile, transfer of selected execution engine to the user's device, and automatic installation/execution of downloaded engine.

Figure 3 shows the entire procedure. When a user connects and logs into the OSCD service framework, as shown in ① of **Figure 3**, the framework processes the user authentication, and the corresponding user information is inserted in a user DB, shown in A. Then, a device DB is created by obtaining profiling information from the user device executing the device profiling function, shown as B in **Figure 3**. For performance purposes, at the same time when the device profile is requested, the OSCD service framework simply reads the recently used service list, frequently used service list, and recommended service list from the DB using the user DB, and then sends the lists to the device, ② in **Figure 3**. In addition, the service ensembler in the OSCD service framework creates an execution engine list that enlists the service execution engines for the user's device through a comparison of execution engine profiles against the user device profile, as shown in C of **Figure 3**. When the user tries to select a service from the provided device-based service list, ③ in **Figure 3**, the framework transmits the service list corresponding to the device-based execution engine lists, D of **Figure 3**. When the user selects a service, ④ of **Figure 3**, the framework detects the suitable execution engine by looking at the session DB, device DB, and service DB, E of **Figure 3**, and then downloads the selected engine to the device. After the execution engine is downloaded to the device, the corresponding execution engine is installed or executed automatically depending on its type, as shown in ⑤ of **Figure 3**.

The detection of an execution engine optimized to a user-selected service and user device is performed in two steps, as shown in **Figure 4**, and by the service ensembler in the OSCD service framework. The service ensembler chooses an execution engine that most satisfies the requirements of the selected services and user device simultaneously. Detection of the most suitable execution engine is achieved by comparing functions offered by the execution engine and functions required by the service. The comparison is possible because of the fact that profiles are stated by the common sets of vocabulary and schema. In the first step of a profile comparison, the device profile is compared with each engine profile to check if the corresponding execution engine can be installed/executed in the device, as shown in ① of **Figure 4**.

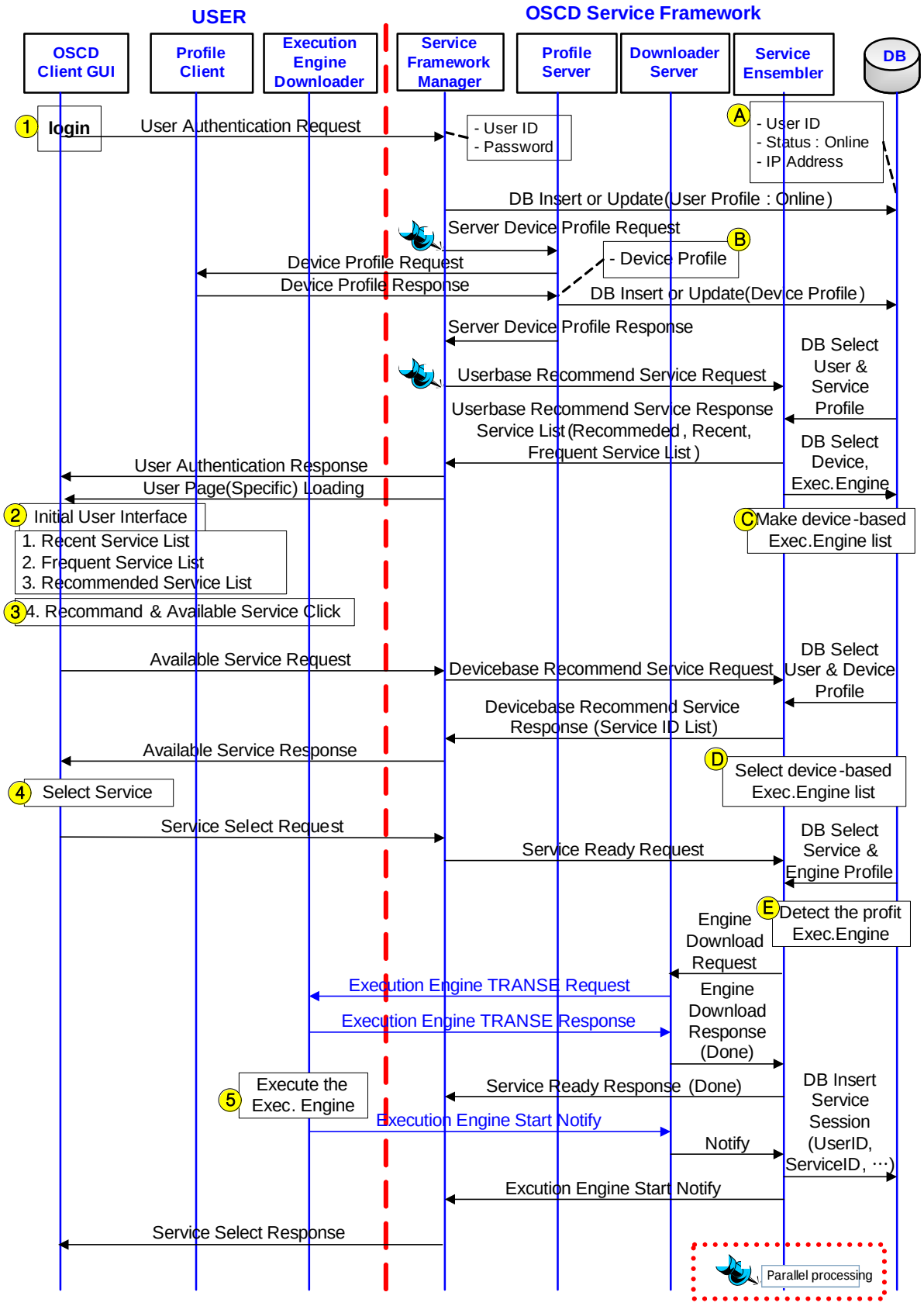

Figure 3. The procedure for dynamic configuration of an execution engine.

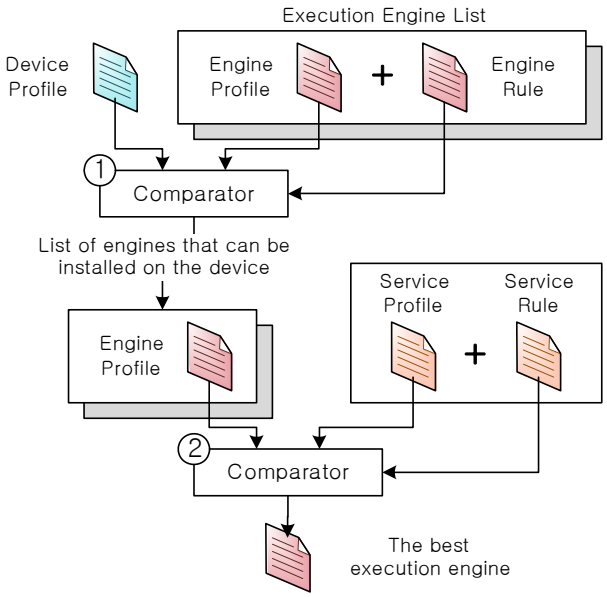

Figure 4. The execution engine detection procedure.

The device profile used in the comparison is the profile of the current user's device and refers to the contents already registered in the DB at a log-in procedure. In the second step, each profile of the installable/executable engine is compared with the chosen service profile to check whether the relationship between final services and execution engine is satisfied, ② of **Figure 4**. The final results of the decision procedures are scores for each compared engine using a proprietary quantization mechanism. The most suitable execution engine among the condition-satisfying engines is chosen by comparing the resulting scores.

3.3. User & Device Available Service Recommendation

The proposed framework can recommend currently available services based on user preference and device characteristics. The following sections describe a learning algorithm according to user preference gathered by examining the service usage history and a recommendation algorithm for services that can be run on the used device. **Table 1** lists four DB structures used for learning about user preference.

When a service is explicitly selected, keywords and categories of the service are stored in KH-TBL and CH-TBL. When specific keywords and categories are searched, they are stored in KS-TBL and CS-TBL. Each of the four DB tables has frequency and recency values, where each entry is ordered by the weighted sum of these values. The user preference learning algorithm for each table is summarized in **Table 2**.

The recommended service list for a specific user is created by searching and re-ordering services that are categorized by keywords and categories with the highest scores from the learning algorithm. Since enlisted services may not have an appropriate execution engine registered for all types of devices, the framework creates an execution engine list for engines that are executable on the device, when an arbitrary device is connected. Indeed, the execution engine list by device is a service execution engine list file by device that records the score value from the device and engine profiles each time a new device is registered.

Table 1. Data structures for user preference learning.

Table	Description
KH-TBL	User selected keyword historic table
CH-TBL	User selected service category historic table
KS-TBL	User searched keyword table
CS-TBL	User searched category table

Table 2. User preference learning algorithm.

Initialize *KH-TBL*, *CH-TBL*, *KS-TBL*, *CS-TBL* are zero
Repeat the following steps:
Observe the user action
Calculate frequency & recency of KH-TBL, CH-TBL, KS-TBL, CS-TBL according to user action
Plus frequency, recency each TBL respectively
Multiply each above value by each weighted value
Lined up in a row from the highest value

Recommendations of user and device available services are determined by these two lists. Services that exist in both the recommendation list by the user and service execution engine list by device are added to the final recommendation service list. For the implemented system, a candidate service with the highest score has the highest ranking in the list.

3.4. Seamless Service Syndicator

3.4.1. Architecture of Seamless Service Syndicator

A seamless service syndicator in the OSCD service framework provides the service synchronization function with mainly a syndication process for service synchronization between heterogeneous terminals and service servers.

Figure 5 illustrates the message flow of a seamless service syndication for a service synchronization. The seamless service syndicator consists of two parts. The first part is related to Syndication Index Save (SIS), and

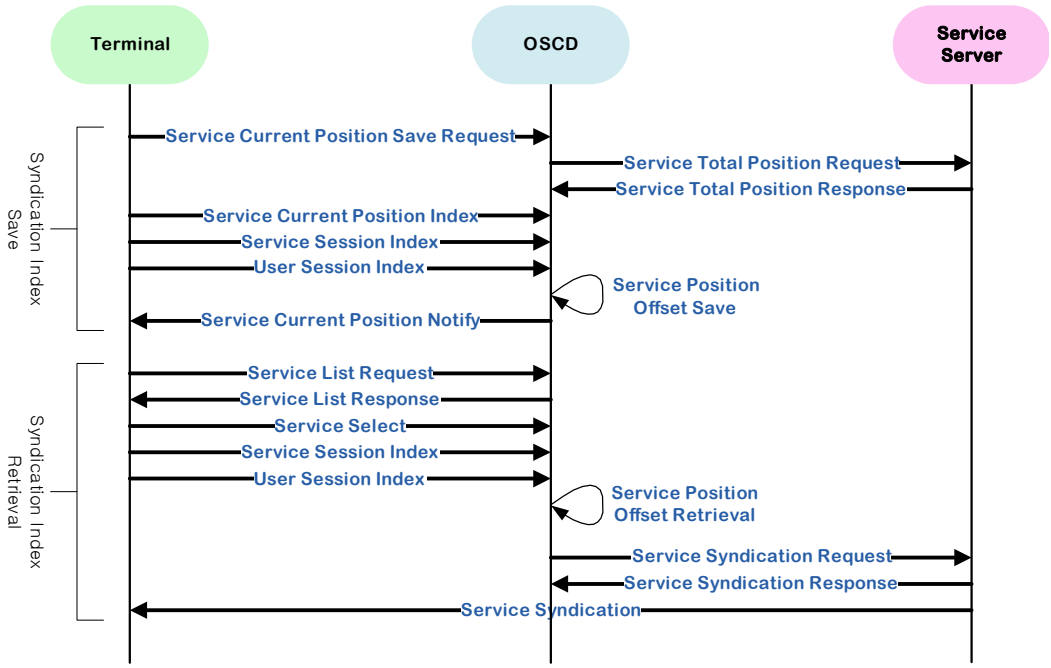

Figure 5. Message flow of seamless service syndicator.

the second part is related to Syndication Index Retrieval (SIR) for service synchronization.

First, the SIS procedure for service synchronization is as follows.

1) A user on a terminal invokes a command of service pause, service stop, or user logout from the Service Server through the OSCD service framework.

2) Current position offset of the service being used is stored in a seamless DB upon service synchronization command.

3) The OSCD service framework requests the service's complete position to provide to the user and obtains the position offset from the service server.

4) The terminal sends the service session information and user session information to the OSCD service framework.

5) The OSCD service framework calculates the service's complete position to be offered to the user and the current position offset. Calculation of the service position offset of the service being used is performed using a user session key (USK) and service session.

The next process is SIR, as the user moves to a heterogeneous terminal in order to resume a paused service. The procedure for SIR is as follows.

1) The user in a terminal requests a service list from the OSCD service framework.

2) The OSCD service framework confirms the USK and delivers the service list to the terminal.

3) The user selects the paused service to resume from the given service list.

4) A service resume command invokes a delivery com- mand. The delivery command sends service session key (SSK) and USK of the user selected service to the OSCD service framework.

5) The OSCD service framework retrieves a user's service position offset from the seamless DB.

6) The OSCD service framework requests a service resume from the Service Server and computes the retrieved service position offset at the point where the service resumes.

7) Synchronized service is provided to the user from the point of service pause, service stop, or user's logout.

Finally, following the described SIS and SIR procedures, the service resumes from the service synchronization point, and the user migrates to a heterogeneous terminal.

A simple architecture of the proposed seamless service syndication, as shown in **Figure 6**, may be constructed with two end terminals, a service server, and an OSCD server where the proposed framework resides. The implications of the architecture in **Figure 6** are as follows.

- Step 1: The user accesses the OSCD server for a service invocation. Execution engine on terminal 1 connects to the OSCD server for a user session initiation through a gateway between terminal 1 and the OSCD server.
- Step 2: The OSCD server sends control messages to the service server for delivery of service to terminal 1 after confirmation of a user session and receiving a user session initiation and service invocation command.
- Step 3: The user is provided with selected service

from the service server.

- Step 4: After the user migrates to terminal 2, the service position index and user session move to terminal 2. The service position offset and user session are synchronized at terminal 2 for service synchronization.
- Step 5: The OSCD server sends control messages to the service server for resuming service to terminal 2 after confirmation of the user session and service position.
- Step 6: Resumed service is delivered to the user on terminal 2.

3.4.2. Seamless Service Syndicator Algorithm

As summarized in **Table 3**, a seamless service syndication algorithm for service synchronization requires support of the OSCD service framework to keep track of service mobility by recording service pause or stop behaviors for a heterogeneous terminal, where T_k^j is the *k-th* terminal of the *i-th* user in the *j-th* session. If a user wants to request service position preservation on a terminal before user and service movement to a heterogeneous terminal, the complete service position taken from the service server is stored in the OSCD service framework. Also, the terminal informs the OSCD service framework of the service current position index (CPI). Then, the OSCD service framework maps SSK and USK to CPI. The mapped service position offset is stored in a seamless DB in the OSCD service framework. The saved service position offset information is notified to the terminal for user confirmation.

When the user wants to resume a paused service from a migrated heterogeneous terminal, the user again requests a service list from the OSCD service framework. If the user selects a paused service, the service position offset is retrieved from the seamless DB. The service obtained from the service server resumes from the retrieved service position offset.

4. Prototype Implementation

A prototype is implemented to verify the feasibility of the proposed service framework for service mobility.

The prototype provides a Web and proprietary GUI-based method for access of the framework so that devices with conventional Web browsers connect to the framework through a Web site, while devices without Web browsers use a proprietary GUI to access the framework. **Figure 7** shows the two connection methods to the framework. Both user interfaces provide basic user log-in/out and service search, view, and selection functions, while Web-based UI provides more pages for comprehensive management of the framework including service/engine registration and management.

Figure 8 illustrates a VoD service scenario with service synchronization where the contents in the VoD server are streamed to user terminals through the service framework. Several users are assumed to move to/from different zones with their mobile devices. Since devices have different performances and capabilities, protocol and content transcoding are also performed by the translators based on the profile and predefined service characteristics in the service framework.

Figure 9 shows the actual demonstration environment for the scenario in **Figure 8**. The service framework is implemented on Linux 2.6.9 using Java and GNU C++ developing languages. The database is implemented with MySQL 14.12. Also, Gtk+ 2.0 for GUI and Apache HTTP Server are used as the components for development. Three terminals use openSUSE11, Windows CE 5.0, and Windows Mobile6 classic operating systems. Embedded Visual C++ 4.0 is used as a developing language, and FFplay and TCPMP are utilized as media players for content rendering. Hence, the devices for the service are categorized into Linux-based PC, Windows CE-based PMP, and Windows Mobile-based PDA for simplicity. The scenario is as follows: User-1 is watching VoD using a PC in his home through the framework with a log-in to the OSCD server. The content is transferred in HD-level MPEG-4 at 30 fps because the PC has the capability to support the content. After a while, User-1 stops (or pauses) the content, logs out of the OSCD server, and moves to a hotspot zone with a mobile device, *i.e.*, a PMP or PDA. The server saves the user and content information. To resume the service on the mobile device, User-1 logs in again to the server he was

Table 3. Seamless service syndication algorithm.

1: if $T_k^j(i)$ *requests Service Position Save, i.e.* $I = j = k$ then
2: if *OSCD* ← *Service Total Position* then
3: *Tkj(i) informs Current Posistion Index*
4: if *OSCD maps* {*SSK* + *USK* + *CPI*} then
5: if *OSCD* ← *Service Position Offset* then
6: $T_k^j(i)$ ← *Current Posistion Offset Notify*
7: if $T_k^j(i)$ *requests Service List, i.e.* $I = j \neq k$ then
8: if $I = j$ then
9: *OSCD* ← *Service Position Offset Retrieve*
10: else
11: $T_k^j(i)$ *selects Service*
12: if *OSCD requests Service Synchronization* then
13: if *SS responses Service Synchronization* then
14: $T_k^j(i)$ ← *Service Synchronization*

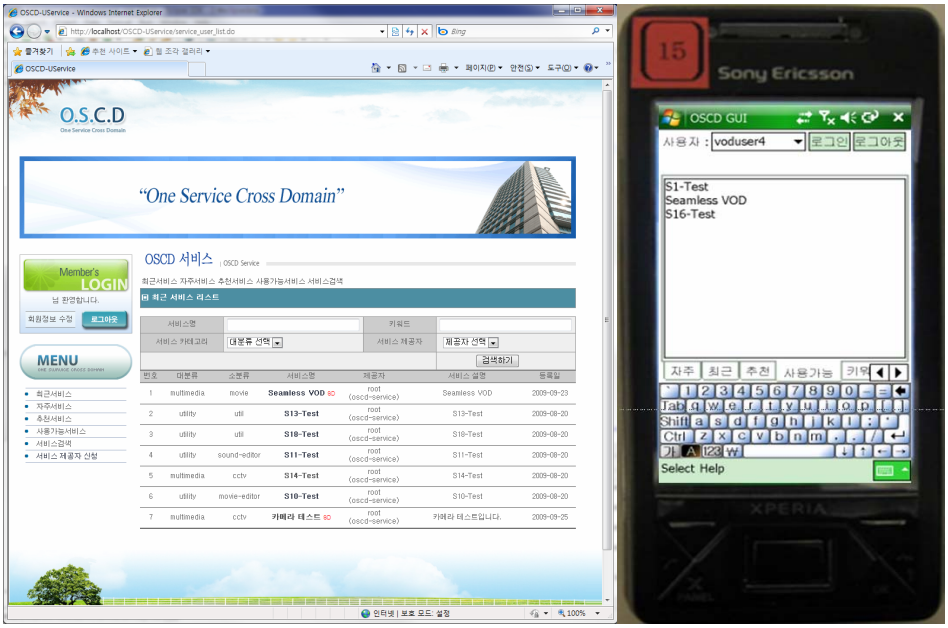

Figure 7. Two types of OSCD initial scr.

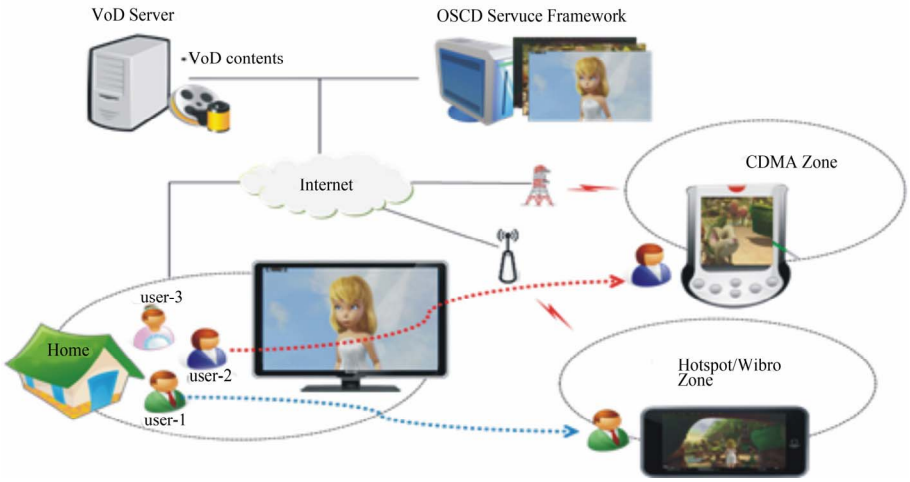

Figure 8. A scenario for supporting service synchronization.

using. At this time, the service recognizes that translations are needed to cope with the capability difference of the devices. For example, the protocol is translated from TCP to RTSP if the player on the device cannot play datastreamed using TCP. Also, for a PDA, the resolution is pulled from HD-level to VGA (640 × 480) at 15fps, and for PDA, the content is transcoded into QVGA (320 × 240) at 24fps to fit the capabilities of the device. User-1 can then view the transformed content from the very point where he paused on the PC, because the server saved the information of the user and the service when he logged out.

5. Conclusion

The paper proposed an open service framework that supports convergence services including heterogeneous service platforms and devices with their own independent platforms. It plays the role of an infrastructure to execute content and application programs with a dynamic configuration using mechanisms such as user preference learning, service and execution engine profiling, and real-time device profiling. It also supports service mobility to provide continuity and service synchronization when terminals or networks are changed due to user movement or a change of personal preference. We implemented a prototype service framework to verify continuity and synchronization of service. We showed not only service mobility of one user's migration but also of multiple users' sharing one single session among them. Interest in App stores is growing with the rapid expansion

Figure 9. Demonstration environment for supporting service synchronization.

of smart phones [23]. The current App stores are organized into application programs based on the same platform or device such as Apple's iPhone or Google's Android phone. Since the current App stores have a closed ecosystem, at least by the operators, developers have to develop applications for each platform and users have to choose from several independent App stores with their devices. To solve this problem, mobile operators have agreed to develop an open international application platform and have organized a Wholesale Applications Community (WAC) with the establishment of a common standard to be completed by the end of 2010. However, no resolution of this work is available at present. The OSCD service framework can perform the activity of an open App store before an open international application platform is completed. If developers enroll application programs and the device characteristics of the application programs into the framework, then users will be able to access the OSCD service framework and can automatically detect application programs suitable for their device and perform a selfinstallation after dynamic downloading. Since the major purpose of an open international application platform is to provide an open application program interface (API) to developers, the OSCD service framework may well evolve into an open international application platform if some sort of open API concept is adopted.

6. Acknowledgements

This work was supported by the IT R&D Standardization program of MKE/KATS, [2011-PM10-02, Development of Smart Utility based Green WPAN Standardization]

REFERENCES

[1] Z. Chen, C. Lin and X. Wei, "Enabling On-Demand Internet Video Streaming Services to Multi-Terminal Users in Large Scale," *IEEE Transactions on Consumer Electronics*, Vol. 55, No. 4, 2009, pp. 1988-1996. doi:10.1109/TCE.2009.5373760

[2] Y.-F. Chen, *et al.*, "Project GeoTV—A Three-Screen Service: Navigate on Smart Phone, Browse on PC, Watch on HDTV," *6th IEEE Consumer Communications and Networking Conference*, Las Vegas, 10-13 January 2009, pp. 1-2. doi:10.1109/CCNC.2009.4785025

[3] E. Lavinal, N. Simoni, M. Song, *et al.*, "A Next-Generation Service Overlay Architecture," *Annals of Telecommunications*, Vol. 64, No. 3-4, 2009, pp. 175-185. doi:10.1007/s12243-008-0082-x

[4] H. Si, Y. Wang, J. Yuan, *et al.*, "A Framework and Prototype for Service Mobility," 2009 *World Congress on Computer Science and Information Engineering*, Los Angeles, 31 March-2 April 2009, pp. 315-319. doi:10.1109/CSIE.2009.667

[5] C. E. Perkins, "Mobile Networking through Mobile IP," *IEEE Internet Computing*, Vol. 2, 1998, pp. 58-69. doi:10.1109/4236.656077

[6] P. Maniatis, *et al.*, "The Mobile People Architecture," *Mobile Computing and Communications Review*, Vol. 1, No. 2, 1999, pp. 36-42. doi:10.1145/329124.329153

[7] M. Roussopoulos, *et al.*, "Person-Level Routing in the Mobile People Architecture," *Proceedings of the 2nd USENIX Symposium on Internet Technologies & Systems*, Boulder, 11-14 October 1999, pp. 36-42.

[8] H. J. Wang, *et al.*, "ICEBERG: An Internet-Core Network Architecture for Integrated Communication," *IEEE Personal Communications*, Vol. 7, No. 4, 2000, pp. 10-19. doi:10.1109/98.863991

[9] A. D. Stefano and C. Santoro, "Net Chaser: Agent Support for Personal Mobility," *IEEE Internet Computing*, Vol. 4, No. 2, 2000, pp. 74-79. doi:10.1109/4236.832949

[10] B. Thai, *et al.*, "Integrated Personal Mobility Architecture: A Complete Personal Mobility Solution," *ACM Mobile Networks and Applications*, Vol. 8, No. 1, 2003, pp. 27-36.

[11] UAProf, "User Agent Profile," Open Mobile Alliance, Approved Version 2.0, 6 February 2006.

[12] CC/PP, "Composite Capabilities/Preference Profiles: Requirements and Architecture," W3C Working Draft, World Wide Web Consoritum, 21 July 2000.

[13] RDF, "Resource Description Framework (RDF): Concepts and Abstract Syntax," W3C Recommendation, World Wide Web Consoritum, 10 February 2004.

[14] SLP, "Service Location Protocol, Version 2," Request for Comments (RFC) 2608, Internet Engineering Task Force (IETF), 1999.

[15] Jini, "Jini Community Resources: Jini Specification v1. 0.1." http://java.net/projects/jini/

[16] uPnP, "Universal Plug and Play Device Architecture Version 1.0," UPnP Forum, 2008. http://www.upnp.org/

[17] CC/PPex, "CC/PP Exchange Protocol Based on HTTP Extension Framework," W3C Note, World Wide Web Consoritum, 24 June 1999.

[18] W-HTTP, "Wireless Profiled HTTP," Wireless Applica-

tion Protocol Forum, Ltd., Version 29, 2001. http://www.wapforum.org/

[19] Jena, "Jena—A Semantic Web Framework for Java." http://jena.sourceforge.net/index.html

[20] OWL, "OWL Web Ontology Language Overview," W3C Recommendation, World Wide Web Consoritum, 10 February 2004.

[21] SWRL, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," W3C Member Submission, World Wide Web Consoritum, 21 May 2004.

[22] Jess, "Jess: the Rule Engine for the Java™ Platform." http://www.jessrules.com/

[23] D. Chamblerlain, "The Apps Store Is Born: Smartphones Enable New Marketing and Advertising Opportunities Worldwide," In-Stat, 2009. http://www.instat.com.

