Dynamically Updatable UPnP Service for Device Role Management

Tadanobu Tsunoda, Akira Fujii and Nobutsugu Fujino
Fujitsu Laboratories Ltd.
4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211-8588, Japan
Email: {ta.tsunoda, akira-f, fujino}@jp.fujitsu.com

***Abstract*—UPnP is a networking protocol that allows devices embracing personal computers, smartphones and tablet computers to discover other devices and control their services. Using portable smart devices, the UPnP services needed by these devices often vary. According to the locations or the situations of users, the required roles of the devices that support users' activities change and the services needed for achieving the roles also vary. However, it is difficult to dynamically add or update UPnP services and change the roles of the devices. We propose a new dynamic update method of UPnP services in this paper. 'Service-updater service', one of UPnP services, is utilized in this method, which can update other UPnP services of its own device. Users do not have to install many services manually in advance. In addition, applying our method, it is also possible for smart devices to change their roles; according to users' situations, the devices can be turned into UPnP controllers, which control UPnP services of other devices. We have developed a prototype to which this method is applied and we have confirmed through the experiment that this technique effectively supports users' activities.**

***Keywords*-UPnP; service; device; updatable;**

I. Introduction

Universal Plug and Play (UPnP) [1] is a communication protocol for controlling various devices connected to a network, where users do not have to do special setups in advance. As well as bonjour[2] installed on iOS devices and service discovery technology adopted to Jelly Bean (Android 4.1)[3], UPnP is one of the most important protocols that can create an IP network without any manual configuration. UPnP makes it possible to control various UPnP devices using UPnP controllers, to which existing network technology such as Simple Service Discovery Protocol (SSDP)[4], SOAP and General Event Notification Architecture (GENA)[5] are applied. Hardware, such as a router and a WEB camera, is a typical device that uses UPnP. Not only such hardware, but also software that operates on a personal computer is able to serve as an UPnP device. For example, we can also regard personal computers, smartphones and tablet computers as media players or file sharing servers that use the UPnP protocol.

In recent years, smart devices such as smartphones and tablet computers become popular instead of personal desktop computers. These devices have specifications equivalent to personal computers that appeared several years ago. These devices have potential to carry out various jobs for users' activities. Therefore it is possible to apply smart devices to UPnP devices as well as personal computers. Generally, the setting positions of conventional UPnP devices such as media players or file servers are fixed. However, smart devices can be moved to various locations. And it is often the case that we want to change the role of these devices according to users' states. For example, when we are at home, we would like to watch TV and use the device as a TV remote control. In another case, when we are in a meeting room in an office, we use the device as a document viewer for meeting materials. Thus smart devices have to change the roles in order to support users' activities according to their locations and states. However, it is necessary to install every service required for the roles beforehand. Roles change according to users' activities and there is no telling which service is needed for the smart devices beforehand. Therefore it is desirable to enable devices to update UPnP services dynamically. OSGi[6] is a framework that allows dynamic update of the software component. However, using only UPnP protocol, a scheme that can update UPnP services more easily is desired.

In this paper, we propose a new method that enables UPnP devices to add, update and remove UPnP services by their own UPnP services. The special UPnP service called 'service-updater service' is defined by this technique. Installing this service in the devices, various UPnP services can be updated. In addition, applying our method, the roles of the devices can be changed according to users' activities. Only the specifications of UPnP is utilized in this method; the service-updater service behaves as one of UPnP services. Therefore this method does not need a special module except for the UPnP protocol. Moreover it is possible to notify the change of state of having updated the services to surrounding controllers by GENA. If appropriate new services for the notified status are known, it is possible for the controllers to prefetch proper service packages before it issues the service update command to the service-updater service.

We have applied this method and developed a prototype of the system that changes the roles of UPnP devices. According to the user status and the kind of terminals that exist in the surroundings, this system changes the role of each device and supports users' works effectively.

This paper is organized as follows. We introduce the related works in section II. The detail of the method of dynamic service updating is explained in section III. The

978-1-4673-5077-8/13/$31.00 ©2013 IEEE

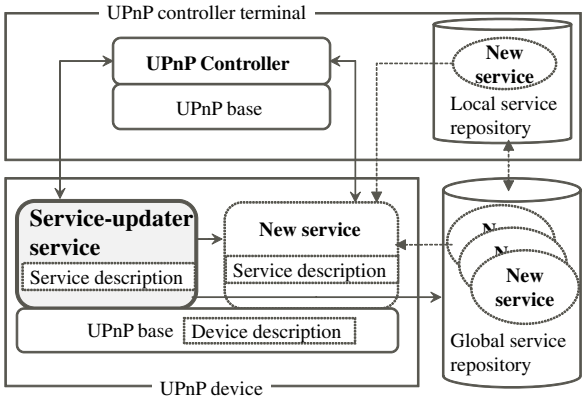

Figure 1. The structure of the main components of our proposed system.

```
<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
 <specVersion> <major>1</major> <minor>0</minor> </specVersion>
 <actionList>
  <action> <name> X_UpdateService </name>
   <argumentList>
    <argument> <name> ServiceName </name>...</argument>
    <argument> <name> ServiceUri </name> ...</argument>
   </argumentList> </action>
  <action> <name> X_DeleteService </name>
   <argumentList>
    <argument> <name> ServiceName </name> ... </argument>
   </argumentList> </action> </actionList>
 <serviceStateTable>
  <stateVariable sendEvents="yes"> <name> X_UpdateTime</name>...</stateVariable>
  <stateVariable sendEvents="yes"> <name> X_UserState </name>...</stateVariable>
  <stateVariable sendEvents="yes"> <name> X_InstalledServices</name>...</stateVariable>
 ...</serviceStateTable></scpd>
```

Figure 2. The service description of the service-updater service.

method of device role management using the service update scheme is stated in section IV. We explain the evaluation and the implementation of the proposed method in section V and VI respectively. The conclusion of this paper is stated in section VII.

II. Related Works

OSGi[6] is one of the frameworks that make it possible to update the software modules on the devices dynamically. It is possible to execute the module update and control the services remotely. However, the core components of OSGi are not sufficient because their functions for the device discovery are poor for distributed networks. The Distributed OSGi document RFC 119[7] describes a way to solve the problem. Distributed OSGi needs a central server for registering the information on the devices. Referring the server, controllers can discover proper services. However, it is troublesome for every controller because it must access the central server to check the information on the devices every time before it controls them. The status of the device must be notified through the server although the status information is automatically notified to every controller by GENA technique of UPnP.

Instead of OSGi itself, it is also possible to put other protocols, such as UPnP, as bundles of OSGi. The bundle service is able to operate as a function of discovering and eventing. A technique using the bundle that utilizes SLP[8], one of the protocols of service discovery, has been proposed as a conventional application system of OSGi[9]. Another study has proposed a method that manages distributed services with an original service discovery bundle[10]. However, in addition to OSGi, these systems need to install the special bundles for discovering and eventing. If it is only controlling a single device, using OSGi is troublesome and feature creep because we have to take much time and effort for the advance preparation.

Other methods that uses JINI or TR-069 have been also proposed[11][12]. However, these conventional techniques do not have all of the functions of discovering, updating and controlling the devices with one particular protocol. Therefore it is necessary to combine two or more methods.

III. Dynamically Updatable UPnP Service

In this section, we give the details of the dynamic service updating method we have proposed. Fig.1 illustrates the structure of the main components of our proposed system. This is designed on the idea that the control is realized only based on the specifications of UPnP. The special UPnP service called 'service-updater service' is defined for adding, updating and deleting UPnP services and this service is installed as a basic module in the UPnP device. An UPnP controller can discover and control the service-updater service utilizing UPnP technology; by SOAP commands dispatched from the controller, the service-updater service achieves updating or deleting UPnP services of its own device. In addition, this method effectively uses SSDP[4] and GENA[5], which are also included in the specifications of UPnP; it is possible to advertise the existence of the service-updater service and notify update event easily. Since the existing UPnP specifications are utilized in this method, the controllers that control the service-updater service can be easily implemented. On the other hand, it is difficult for such protocols as bonjour and the network technology adopted to Android 4.1 to create service updating system like ours because these protocols do not have all of the functions of discovering, controlling and eventing.

Fig.2 illustrates the service description of the service-updater service. The actions that execute updating or deleting UPnP services according to the SOAP messages from the controllers are defined as follows: X_UpdateService action executes addition of a new UPnP service or update of a specified existing service of its own device. The argument 'ServiceName' indicates the name of the updated service. The argument 'ServiceUri' indicates the URI where the service repository in which new UPnP services are stored is located. X_DeleteService action executes deletion of the specified service. The argument 'ServiceName' indicates the

```
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
 <specVersion> <major>1</major><minor>0</minor></specVersion>
 <device>
  <deviceType>urn:flab.fujitsu.com:device:GenericTerminal:1</deviceType>
  <modelDescription>Tablet computer, Android OS</modelDescription> ...
  <serviceList>
    <service><serviceType>urn:flab.fujitsu.com:ServiceUpdaterService:1
      </serviceType> ...</service>
    <service><serviceType>urn:flab.fujitsu.com:DocumentViewerService:1
      </serviceType>...</service></serviceList>
 </device></root>
```

Figure 3. The device description of the UPnP device in which the service-updater service is installed.

name of the deleting service.

In addition, the service-updater service has some state variables: X_UpdateTime, X_UserState and X_InstalledServices. The service-updater service rewrites these values when a service is updated. These values are notified as state variables by GENA to every UPnP controller that has pronounced the subscription to the service. According to the notification, surrounding controllers can grasp the status of the device before they dispatch the service update command to the service-updater service. If appropriate new services for the status notified by GENA are known, it is also possible for the controllers to prefetch proper service packages.

Fig.3 illustrates the device description of the UPnP device in which the service-updater service is installed. This device description is dynamically rewritten by the service-updater service. The service-updater service and the updated service are declared in the serviceList element. By the discovery, the controller sends SOAP messages that issue the service update or the control of other services.

Fig. 4 illustrates the procedure of the operation of the proposed system. The concrete procedure of the service update operation is as follows:

1) Initial operation
   A controller (a service update controller) multicasts an M-Search message in a network and it tries to discover service-updater service among the surrounding devices. The device that has the service-updater service, which has received the M-Search message from the controller, transmits a response message to it and notifies existence of the service. The controller acquires the XML files of the device description and the service description from the device and grasps the control commands for it. According to the device description, the controller transmits the subscription message to the device so that the device status can be notified to the controller by GENA. Then the controller retrieves the initial state variables from the device.
2) Usual operation
   Whenever the states of the device change, the state variables are notified to every terminal that declares the subscription to the service. If appropriate new services for the notified state variables are known, the controller tries to prefetch the proper service package from the global service repository and store it into the local service repository.
3) Service update operation
   According to the device description and the service description, the controller transmits control commands as a SOAP message to the device. The service-updater service interprets the received SOAP message and requires the desired service package from the service repository according to the arguments, ServiceName and ServiceUri, specified in the message. If the prefetch operation is performed beforehand, the ServiceUri must be the location of the local service repository. The device acquires the service package including the service description and the program module of the new service from the repository. The service-updater service extracts the package and installs the new service on the UPnP base. It also updates the device description to advertise the existence of the new service. The service-updater service also notifies the update information of the new service to every controller by GENA. After the procedure, every UPnP controller can control the new service.

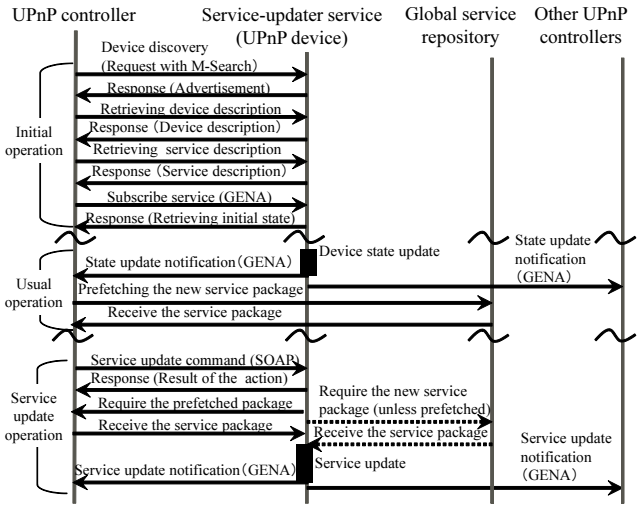

Figure 4. The procedure of the update operation of the proposed system.

In addition, the update of an UPnP controller can be also realized by the same mechanism as the service update. Generally, UPnP controllers are made of software components. Therefore we can regard the controller as a kind of UPnP services. That is, it is also possible to update a 'controller service' dynamically later. By using this mechanism, according to users' situations, it is also possible to change the role of the device and exchange the functions between the service and the controller dynamically.

Table I
THE COMBINATION OF DEVICES AND UPNP SERVICES.

Activity	User state (location)	Combination of devices	Kind of device	Role	UPnP service
Presentation	Meeting room	Only tablet computer	Tablet computer	Document viewer and slideshow controller	Document viewer service and slide controller service
		Smartphone and tablet computer	Smartphone	Slideshow controller	Slide controller service
			Tablet computer	Document viewer	Document viewer service
		Tablet computer and projector	Tablet computer	Slideshow controller	Slide controller service
			Projector	Document viewer	Document viewer service

IV. DEVICE ROLE MANAGEMENT

The dynamic service update method we proposed in section III makes it possible to transform the roles of smart devices according to users' situations. Since smartphones and the tablet computers carry GPS receivers and various sensors, it is possible for a system to estimate the locations or other statuses of users. It is also possible to change the device role dynamically according to the existence of other devices in the surroundings. It is decided by the device discovery of SSDP and the interpretation of the device description in which the type of the device is stated. The vendor of an UPnP device can describe characteristic information into a device description. For example, it is possible to describe the information about the type of the device into the modelDescription element in the description XML as shown in Fig.3. The vendor can define 'smartphone' into modelDescription of the smartphone or 'tablet' into modelDescription of the tablet computer.

Defining the relation between the situation and the UPnP service, it is possible to update the role of the device according to users' situations. The controller has such a table as illustrated in Table I; according to the types of devices in the meeting room, the service that should be dynamically added to each device is decided. This is how the controller can determine the service the controller should add to each device based on the information described by the device description and the state variables notified by GENA technique. In addition, the controller can grasp which service should be prefetched from the global service repository.

Fig. 5 illustrates the image that the service of each device is changed according to the types of devices in a meeting room. The controller that controls the service-updater service is installed in a server. In the meeting room A, since a smartphone and a tablet computer exist, the slide controller service is added to the smartphone and the slide viewer service is added to the tablet computer. On the other hand, in the meeting room B, since an UPnP projector exists, the slide controller service is added to a tablet computer and the slide

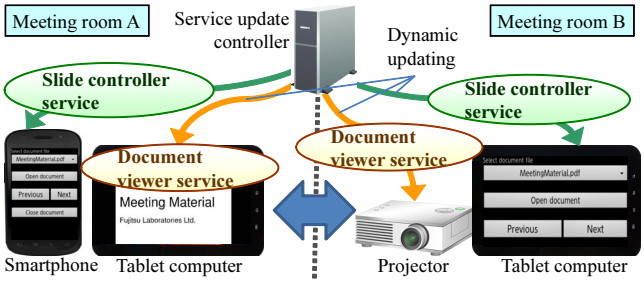

Figure 5. The roles and the services of each device are changed according to the types of devices in the meeting room.

viewer service is added to the projector. In this way, smart devices are always carried by users and the roles needed by users should change frequently. Our method changes UPnP service dynamically, updates the roles of the devices according to users' situations and effectively supports users' activities.

V. EVALUATION OF OUR METHOD

In order to confirm the performance of our proposed method, we evaluated the developed system. As conventional technology for comparing with our technique, we chose distributed OSGi (DOSGi)[7]. We used DOSGi implementation by Apache cfx[13]. In use of DOSGi, the central server for registering the information on the devices is needed. The DOSGi implemented by Apache cfx uses a ZooKeeper server[14]. Using this server, it is possible to discover distributed services and acquire the device state from remote terminals. The list of the equipment used for the evaluation is shown in Table II. The residential gateway in the list carries not only the wireless LAN function of IEEE802.11b/g/n, but also the function of IEEE802.16e-2005 (WiMAX). In this experiment, we arranged the service packages onto a WEB server as a global service repository on the Internet. Therefore it is possible to communicate not only between the smart devices but also between the smart devices and the WEB server through the Internet. In this experiment, we changed the size of the packages of the new services that are dynamically added. And we measured the service activation time, which it took from the dispatch of the service update command by the controller to the completion for the preparation for the use of the updated service.

The result of this experiment is shown in Fig.6. This chart shows comparison between DOSGi, our technique without the prefetch (DU-UPnP) and our technique using the prefetch method (DU-UPnP with prefetch). It turns out that the service activation time of our technique without the prefetch and DOSGi are very near. Since both the technique of our method and OSGi use the dynamic class loader of Java platform for the modular dynamic addition, the internal processing time is almost the same. In order to reboot an internal module for the dynamic addition of a service as

Table II
THE LIST OF THE EQUIPMENT USED FOR THE EVALUATION.

Web server (Global service repository)	Fujitsu FMV ESPRIMO D5270 (OS: Windows XP, CPU: Celeron E1400 2.0GHz, Memory: 2.0GBytes)
Central server for OSGi	Fujitsu FMV ESPRIMO D5270
Smartphone	Fujitsu T-02D (Android OS 4.0.4)
Residential gateway	Aterm WM3600R
Tablet computer used in section VI	Fujitsu F-01D (Android OS 3.2)
Projector connected device used in section VI	Fujitsu FMV R8280 (OS: Windows XP, CPU: Celeron 723 1.2GHz)

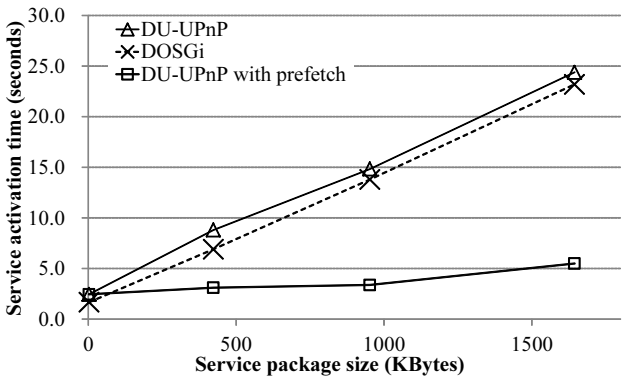

Figure 6. The comparison of the service activation time, which it took from the dispatch of the service update command by the controller to the completion for the preparation for the use of the updated service.

Figure 7. When there is only a tablet computer in the meeting room, the document viewer service and the slide controller service are installed on the device at the same time.

Figure 8. When there are a tablet computer and a smartphone in the meeting room, the document viewer service is installed into the tablet computer and the slide controller service is installed into the smartphone.

for the implementation of our system, it takes some more seconds than DOSGi, which does not perform rebooting at all. However, the time for the download and the deployment of the archived package is dominant. Moreover our technique using prefetch can carry out the fastest completion of preparation of the updated service. By this method, the controller acquires the service packages that are needed for the device beforehand. And exchanging the packages is locally performed between the controller and the device and completes at higher speed with the communication links of wireless LAN than ones of WiMAX, which are not always stable. By our method, the prefetch technique can be applied because the state of every device is automatically notified using GENA based on the specifications of UPnP. On the other hand, since DOSGi depends on the central server, the state of the device cannot be grasped unless the controller queries the information to the server. Our method achieved the completion of updating and preparing for the service within only five seconds when the device needed 1.0 MBytes of package. It was confirmed by this experiment that our method is practical enough and especially it is effective for the dynamic update to apply prefetch technique.

VI. IMPLEMENTATION OF THE PROTOTYPE SYSTEM

Extending the evaluated system explained in section V, we developed a new prototype as an instance of the system as we described in section IV, which proves the ability of changing the roles of the devices dynamically according to the condition whether a smartphone, a tablet computer or a projector exists in a meeting room. The equipment for the system is also shown in Table II. We installed the service update controller into the server in the meeting room. If a device enters the room, the controller of the server issues the update message to the service-updater service on the device according to the types of devices that exist in the room.

Fig.7, Fig.8 and Fig.9 are the pictures taken in the meeting room in the experiment of the prototype system. The operation of the devices also follows the Table I. When there is only a tablet computer in the meeting room, the document viewer service and the slide controller service are installed on the device at the same time as shown in Fig.7. The upper part of the screen becomes the viewer of the document, which the document viewer service provides, and the controller buttons are arranged on the lower part, which the slide controller service provides. We can check the contents of a meeting material with only one tablet computer because the screen has both the viewer and the controller.

On the other hand, when there are a tablet computer and a smartphone in the meeting room, the document viewer service is installed into the tablet computer and the slide controller service is installed into the smartphone as shown in Fig.8. In this case, the tablet computer becomes just

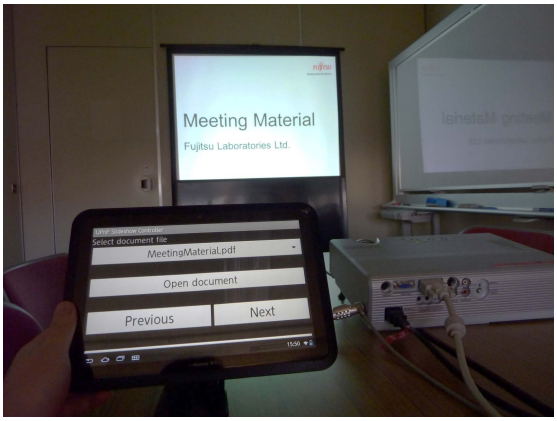

Figure 9. When there are a tablet computer and a projector in the meeting room, the document viewer service is installed into the projector and the slide controller service is installed into the tablet computer.

a viewer. The smartphone has the function of the remote control of the viewer. When there are a tablet computer and a projector in the meeting room, the document viewer service is installed into the projector and the slide controller service is installed into the tablet computer as shown in Fig.9. In this case, the tablet computer changes the role again. It becomes the UPnP controller although it has been the UPnP device in the case of Fig.8.

As described so far, we can change the roles of the devices according to the types of the surrounding devices. In all the combination of the devices, we have confirmed that every service has been updated within only several seconds, which is corresponds to the result of the evaluation stated in section V, and this system can support users' activities effectively.

VII. Conclusion

Smartphones and tablet computers are always carried by the users, and the roles for which these devices are needed should change according to users' locations and situations. Using these smart devices as UPnP devices, it is necessary to update the service of the device according to the role. We have proposed the method of updating UPnP services by service-updater service. Since this method uses only the specifications of UPnP, it does not need to install any extra software except for the UPnP module beforehand. In addition, it is easily possible to discover the service-updater service or to detect the updated service using the mechanism of SSDP and GENA. Furthermore, it is possible by acquiring the information on the type of the device from the device description to update the suitable service for cooperation with the surrounding devices. We have developed the prototype system that changes the roles of the smart devices dynamically by proposed mechanism when these devices exist in a meeting room. In this system, the service of each device is immediately updated according to the types of neighbor terminals. The dynamic update of UPnP services by the service-updater service and the dynamic change of the roles of devices are very effective and we have confirmed that users' activities are appropriately supportable by our system.

The limitations of this technique is that the devices that do not exist in the same subnetwork are uncontrollable. In addition, it is a security issue of this method that everyone can execute updating services of other devices in the same subnetwork. However, since our technique follows the specifications of UPnP, existing solutions considered for conventional UPnP systems can be also applied to the method. Consideration and implementation of these solutions into our system are future works of our study.

References

[1] UPnP Forum, "Upnp device architecture 1.0," *http://upnp.org /specs /arch /UPnP-arch-DeviceArchitecture-v1.0.pdf*, 2008.

[2] Apple Inc, "Bonjour overview," *https://developer.apple.com /library /mac /#documentation /Cocoa /Conceptual /NetServices /Introduction.html*.

[3] Google Inc, "Android 4.1 apis," *http://developer.android.com /intl /ja /about /versions /android-4.1.html*, 2012.

[4] Y. Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright, "Simple service discovery protocol/1.0," *IETF internet draft*, 1999.

[5] J. Cohen and S. Aggarwal, "General event notification architecture base," *IETF internet draft*, 1998.

[6] OSGi Alliance, "Osgi core release 5 specification," *http://www.osgi.org /Download /Release5*, 2012.

[7] OSGi Alliance , "Osgi service platform release 4 version 4.2 - early draft," *http://www.osgi.org /download /osgi-4.2-early-draft.pdf*, 2008.

[8] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service location protocol, version 2," *IETF RFC2608*, 1999.

[9] M.-X. Chen and T.-C. Tzeng, "Integrating service discovery technologies in osgi platform," *Comput. Stand. Interfaces*, pp. 271–279, 2011.

[10] E. Kim, K. Yun, and J. Choi, "Rsp: A remote osgi service sharing scheme," *Ubiquitous, Autonomic and Trusted Computing, 2009. UIC-ATC '09.*, pp. 318–323, 2009.

[11] P. Dobrev, D. Famolari, C. Kurzke, and B. A. Miller, "Device and service discovery in home networks with osgi," *Communications Magazine, IEEE*, vol. 40, no. 8, pp. 86–92, 2012.

[12] P.-Y. Chen, "A novel network module for medical devices," *Engineering in Medicine and Biology Society*, pp. 1553–1556, 2008.

[13] Apache CFX, "Distributed osgi," *http://cxf.apache.org /distributed-osgi.html*.

[14] "Apache zookeeper - home," *http://zookeeper.apache.org/*.