Personal Home Server: A Software Infrastructure for Supporting Spontaneous and Personalized Interaction in Home Computing Environments

Tatsuo Nakajima† and Ichiro Satoh‡
†Department of Computer Science
Waseda University
3-4-1 Okubo Shinjuku Tokyo 169-8555, JAPAN
‡National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, JAPAN
tatsuo@dcl.info.waseda.ac.jp, ichiro@nii.ac.jp

ABSTRACT
In this paper, we propose a personal home server that makes it possible to coordinate a variety of home appliances. Since each person has a different personal home server, it allows him/her to personalize how to use the appliances according to his preferences. Also, a personal home server can discover and configure appliances at any places, such as railroad stations, cars, and streets as well as houses in a seamless way. We show the design and implementation of a personal home server, and present its current status.

1. INTRODUCTION
Ubiquitous computing and mobile computing are key technologies in future embedding computing. The concept of ubiquitous computing enables everyday objects, including appliances and environments to be smart by embedding small computing devices[2]. Such devices are disappearing inside various kinds of appliances, and invading every aspect of our daily lives. Therefore, the spontaneous collaboration of computing devices embedded in our environments is necessary to offer various and advanced services. However, the number of these devices will be increased dramatically, and it is difficult to adopt current middleware infrastructures because their spontaneous interaction is not personalized to reduce the number of appliances that a user interests to use currently. Also, a ubiquitous computing environment may not be suitable for offering personalized services because devices embedded in our environments may not be able to identify users without complex globally distributed authentication services.

On the other hand, the concept of mobile computing means that computing devices, for example, note-PCs, PDAs, or wearable computers, are carried by their users, since these computers are very small and powerful. Thus, the execution of an application on the devices can be personalized according to a user’s preferences in an easy way. Also, since the user’s preferences can be maintained in his portable device, it is easy to change the preferences over time by respective users, and the preferences can be used at any places since a user can carry his device anytime anywhere. Therefore, the mobile computing approach provides both personalization and protecting privacy simultaneously. However, currently, mobile devices can access a fixed set of services on the Internet. We need a new software infrastructure for spontaneously accessing various services that will be embedded in our daily environments.

Since the above two approaches have been often treated to be posed as polar technologies, they have been explored independently. This paper proposes a new framework, called a personal home server, to alleviate the disadvantages of one approach by exploiting the advantages of another approach. A personal home server offers the following characteristics.

- Each personal home server enables a user to control home appliances embedded in our environments according to a user’s preferences and situations.
- The personal home server can be used to configure a variety of appliances at any places, and the configuration is customized by using the same personalization policies.

A variety of computing devices for enriching the quality of our daily lives are in fact already present in almost every room of a modern building or house and in many of the public facilities of cities. One goal of our system is to hide complexities in coordination among various devices from a user.

In our approach, since a personal home server chooses the configuration of appliances near a user, it is easy to configure them at any places where s/he is. For example, our system enables a user to control a variety of appliances in an airport, in a station, and in an automotive space according to his/her preferences. Also, our approach can offer a variety of personalized services by using the same personalization policies to access appliances anytime anywhere[6]. Therefore, the system allows us to interact with environments in a seamless and personalized way.

2. DESIGN ISSUES
This section describes four design issues when designing a personal home server to satisfy the requirements described in the previous section. We believe that in the future, traditional problems such as real-time, fault tolerance, and battery management will still be important, but future embedded systems should take into account various new issues such
as spontaneous interaction, personalization, privacy protection, security, and interoperability. We hope that the discussions presented in the paper will be helpful for reconsidering traditional issues in embedded systems by taking into account the new requirements.

2.1 Spontaneous Interaction

Currently, embedded systems like cellular phones and digital televisions are becoming more and more complex. The functionalities in the systems are duplicated, and a user always confuses to use the systems. We believe that reducing complexities in our daily lives is an important issue for future embedded systems. One of the approaches to solve the problem is to decompose complex systems into simple and specialized appliances to reduce complexities caused by the genericity[7], and a service is offered by composing the appliances in a spontaneous way. Future middleware infrastructures for ubiquitous computing should support spontaneous interaction to make it possible to aggregate appliances without setting up the appliances and environments.

Spontaneous interaction is also attractive to construct ubiquitous computing environments in an incremental way. Building smart environments by embedding computers and sensors in our environments in not practical since the approach is expensive to change our environments. Adding smart artifacts like information appliances and sensor augmented daily objects in an incremental way is an expected approach, and coordinating them spontaneously is very important to realize ubiquitous computing environments in a practical way.

In our approach, a personal home server detects the appliances near a user, and generates a user interface automatically according to information retrieved from them. The approach does not require to set up environments where a user visits. Thus, the user can operate a variety of appliances anytime, anywhere.

2.2 Personalized Services and Privacy

Our future daily lives will be more and more complex. The personalization of services is a key to reducing the complexities. The personalization requires to use personal information, and the information should be hidden from other persons to protect a user’s privacy.

In our approach, a user can carry his/her own personal home server. The approach can maintain personal information within the user’s personal home server, that is always bound to each user. Thus, the server does not leak the personal information about its user to other parties. The approach can introduce a mechanism for dynamically generating HTML-based user interfaces for controlling appliances according to the personal information.

The personal information can be used to personalize services at any places since the information is always carried by a user. This means that we can use the same personalization policies whenever we like to access appliances. In the future, our social relationship will become complex, and it is better not to share each user’s privacy information because the personal relationship will be dynamically changed according to each social situation. A personal home server can solve the problem, and complex infrastructures are not necessary to protect privacy.

2.3 Simplicity and Interoperability

In the future, everyday objects may give various information through embedded sensors. Also, the objects should be integrated by middleware infrastructures with home appliances. Therefore, simple protocols are desirable for future devices embedded in our daily environments.

Our approach adopts the UPnP(Universal Plug and Play protocol) to control home appliances. We believe that UPnP is one of the most promising candidate as a common protocol that a variety of middleware infrastructures support. Therefore, it is easy to operate commercial home appliances from our personal home server. Also, the SOAP(Simple Object Access Protocol) protocol adopted in UPnP is simple to be implemented in small devices.

The approach allows various middleware infrastructures like Jini and HAVi(Home Audio/Video Inteoperability) to communicate with each other through SOAP-based proxies in an easy way as described in [4]. Also, the approach can integrate Web services with our home computing service in an easy way because of using the same protocol. We believe that integrating Web services and home computing services is useful to offer more pleasurable and useful services. Also, our approach offers a light-weighted security mechanism to operate appliances. In our approach, each personal home server can know the existence of appliances that have the permission to operate.

2.4 Naming Scheme

We use URL as an intermediate representation for not only specifying but also controlling appliances, because URL is one of the simplest representations to identify networked devices from a variety of existing user-side devices, such as smart cellular phones, PDAs, and PCs.

Today, it is popular that home appliances can be controlled from a Web browser by transmitting HTML forms. However, the POST request is used to transmit a command in usual cases, and the approach is not easy to transmit the command from popular presentation documents such as Flash, MS Power Point or SMIL(Synchronized Multimedia Integration Language) documents that will be widely used as user interface for future appliances. Also, it is difficult to automatically generate an HTML document containing commands for detected appliances near a user because each command requires a different HTML form to transmit a POST request. The fact limits the design possibility of our user interface. Our naming scheme allows a user interface to transmit commands by using a GET request that is used in Flash, MS Power Point and SMIL. Also, our approach allows us to specify our preferences in URLs as attributes for the appliances.

We believe that our approach allows interaction designers to design various home computing applications by using their favorite presentation design tools. Thus, the cost to develop an appliance will not become expensive.

3. DESIGN AND IMPLEMENTATION

Figure 1 presents an overview showing how a personal home server works. A personal home server is implemented in a personal and portable device like a cellular phone, a wrist watch, or a jacket. Thus, the server can be carried by a user anytime anywhere.

3.1 An Overview of Personal Home Servers and Appliances
In ubiquitous computing environments, it is not easy to specify which functionality is useful in advance, because the required functionality should be different according to a user's situation. In our approach, both a personal home server and a home appliances consist of several components and each component should be dynamically loaded or unloaded according to the current situation.

3.1.1 Software Architecture of Personal Home Server

Figure 2 shows the structure of a personal home server. In the current implementation, we have constructed 12 components. SOAP, GENA, SSDP, UPnP, HTTP Server, HTTP Clients are components for implements the UPnP protocol, that is a low level protocol invoked by the components described below and implementing high level functionalities offered by a personal home server. Also, the protocol is used to offer services implemented in a personal home server. For example, a user can notify his/her friends by sending a command to an alarm service on the friends' personal home servers.

Database is a component to store information about appliances near a user. PreferenceInference is a component to support personalization in our system. In-Forwarder is a component to receive a command encoded in a URL and converts the command to SOAP. Out-Forwarder is a component to transmit a SOAP command to a target appliance. Jini/SOAP is used to send a command to Jini devices by converting a SOAP command to a RMI invocation. The component is used to show that our system can control a variety of appliances supporting different home computing middleware infrastructures[4]. UI is a component that generates an HTML-based user interface document from information specified in service specification documents received from respective appliances.

3.1.2 Structure of Home Appliances

Our system supports UPnP-based appliances. UPnP is a simple protocol and can be implemented in small devices. It adopts SOAP to transmit commands to appliances. Our personal home server can control commercial UPnP-based home appliances by replacing SCPD(Service Control Protocol Declaration) service specification documents to our RDF(Resource Description Framework) based service specification documents. Since our middleware infrastructure has been implemented based on a component framework, we can reuse most of components for developing personal home servers to develop home appliances. In our current implementation, we have built an X10-based light appliance and a television appliance. We have also implemented a UPnP based MP3 player on a standard PC.

3.1.3 How Does a Personal Home Server Work?

The personal home server automatically collects service specification documents containing information about home appliances near a user by using the SSDP component, and creates a database storing information about these appliances in the Database component. Then, the UI component creates a presentation document automatically. A display appliance near the user also detects the personal home server, and retrieves the presentation document containing the automatically generated user interface. The display appliance shows the presentation document. The document contains URLs defined by our URL-based naming scheme and embedding the attributes of appliances near a user and their commands, that are retrieved from the collected service specification documents received from the appliances. The presentation document is customized according to a user's preference by using the PreferenceInference component. When a user touches the display, a URL containing the attributes of an appliance and its command is transmitted to his personal home server via an HTTP message. The In-Forward component identifies a target appliance by accessing the Database component, and translates a command field contained in the URL to a SOAP command by sending a query to the Database component. Finally, the SOAP command is forwarded to the target appliance by the Out-Forward component.

3.2 URL-based Naming Management

Our framework allows a user to access one or more ap-
appliances through a personal home server. However, it is difficult for the user to identify one of the most appropriate appliances among them. Therefore, we introduce a URL-based naming convention for specifying and controlling appliances. In our approach, by embedding the attributes of appliances and commands in URLs, an HTML-based presentation document can be used to control home appliances. The convention is defined within the standard URL but the path elements of the URL form can contain some additional information according to the following ABNF-like syntax:

```
path = request *( "&" request)
request = search | command
search = "%3F" pair ;%3F = ?
command = "!" pair ; "!" name
pair = name = "" value
name = <HEX escaped string>
value = <HEX escaped string>
```

The `?` attribute specifies a query expression and is a field-value pair, where a field describes the property of the query (such as location, or owner), and a value is a string or an integer. The `!` attribute specifies a command, where a field describes the property of the command, and a value is a string or an integer. The URL definition is very flexible because we can specify various attributes to identify a target home appliance. We can also use attributes that represent context information such as location. A personal home server can select an appliance in a context-aware way.

http://A.B.C.D/?function=light&type=ceiling&power=on

The above URL is included in an HTML document when a light appliance is detected by a personal home server, and the document is shown as a user interface by a display appliance near a user to control the light appliance. The command specified in the URL is transmitted to a personal home server whose IP address is A.B.C.D. In our approach, since a presentation document is generated by a personal home server, its IP address can be included in the presentation document automatically. The `?function=light?type=ceiling` element specifies an electric light controlled from a user. In this case, we like to control a ceiling light. The `!power=ON` element signals to the server and nominates a specific function, called `power` provided with the `ON` value by the appliance. The URL is a request to turn on a ceiling light. The example shows that our scheme can identify an appliance that a user likes to control by specifying their attributes.

### 3.3 Service Management

In our system, the Database component in a personal home server knows respective appliances via the SSDP (Simple Service Discovery Protocol) protocol. Each home appliance transmits advertisement messages periodically. Also, the component asks respective appliances to transmit advertisement messages by sending a SEARCH message. After receiving the advertisement messages, the component transmits an HTTP GET request to get a service specification document to each appliance. The personal home server transmits the SEARCH message periodically, and the service specification document of an appliance is removed if there is no advertisement message within a fixed duration.

The Database component in a personal home server contains all service specification documents detected currently. Our service specification document is represented as an RDF document. It contains a link to a WSDL (Web Service Definition Language) document identifying commands that can be accepted. If an appliance contains several functionalities, its specification document may contain several links to WSDL documents. Also, attributes of the document are used to identify a target appliance. For example, if a URL contains `? function=light&type=ceiling`, the `function` tag and `type` tag in the document are examined, and the appliance whose service specification document’s tags are matched to the attributes in the URL is chosen as a target appliance.

An example of a service specification document for a light appliance is specified as follows.

```xml
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:core='http://www.dcl.info.waseda.ac.jp/homecomp/
  xmlns:soap='http://www.w3.info.waseda.ac.jp/homecomp/
    schemas/soap/1.08#'
  xmlns:soap='http://www.w3.info.waseda.ac.jp/homecomp/
    schemas/soap/1.08#'
  xmlns:core='http://www.dcl.info.waseda.ac.jp/homecomp/
    schemas/core/1.08#'>
  <rdf:Description rdf:about="urn:homecomp:x10light1-855503ae">
    <core:friendlyName>Light Controller Device</core:friendlyName>
    <core:modelName>Light Controller Device</core:modelName>
    <core:manufacturer>X10</core:manufacturer>
    <core:location>Living Room</core:location>
    <core:modelURL>http://www.x10.org</core:modelURL>
    <core:modelDescription>This device allows us to control the brightness of a light appliance via the X10 protocol.</core:modelDescription>
    <core:function>light</core:function>
    <core:name>light</core:name>
    <core:controlURL>/control/light1</core:controlURL>
    <core:WSDL>light.wsdl</core:WSDL>
    <core:operation>Get</core:operation>
    <core:type>Light</core:type>
    <core:opName>Get</core:opName>
    <core:operationURL>/control/light1</core:operationURL>
  </rdf:Description>
</rdf:RDF>
```

The Database component offers two functions to access information in the database. The first function is used to transmit a command. When a personal home server receives a URL from a display appliance, it extracts the attributes of function for a target appliance like `?func=television` in the URL. The service database returns an IP address of the appliance whose service specification document’s `func` tag is `television`. Thus, the command encoded in the URL is converted to a SOAP request, and the SOAP request is transmitted to the target appliance specified by the IP address. The second function is used for generating an HTML-based presentation document automatically. The function returns an array containing information of respective appliances. The information of the appliances includes the names, command names, and the attributes of the appliances contained in service specification documents. This information is converted to URLs, and they are embedded in an automatically generated HTML document.

### 3.4 Security Support

It is not practical to control any appliances from each user’s personal home server. Thus, we offer a security mechanism to protect appliances. However, the mechanism should be simple to be implemented in a small device. Our security mechanism extends the above appliance discovery protocol, and the personal home server can know the existence of appliances that have the permission to operate.

In our scheme, we assume that each appliance offers RF tags that contain an encryption key. The SSDP messages transmitted from an appliance are encrypted by the key. A user who likes to use the appliance needs to insert one of the
tags into his/her personal home server. The personal home server contains an RF tag reader, and the reader retrieves the key from the inserted tag. Finally, the key is registered in the database of the server.

When a personal home server receives an SSDP message, the message is decrypted by the key registered in the personal home server. If the decryption is successful, the personal home server transmits an HTTP request to get a service specification document. Currently, we have implemented our scheme by using the RF code Inc.’s spider system.

When a user likes to control multiple appliances, s/he needs to have tags for respective appliances. We assume that future tags will be very small, and they can be inserted in a personal home server in a convenient way. In our approach, the key is discarded when a corresponding tag is removed after its expiration time is passed.

Also, a user can give a key temporarily to her friend to access home appliances in her home. In this case, she closes her personal home server to her friend’s personal home server. The keys in her personal home server are stored in her friend’s personal home server. If she likes to give the permission of a particular appliance, she closes her tag of the appliance to her friend’s personal home server. The keys will be discarded after the expiration time is passed. Also, we like to extend our approach in the future to discard the keys when her friend goes out from her home.

In public space, tags containing keys for operating appliances may be embedded in the appliances. Also, the keys can be embedded in various objects such as tables and walls. When a user closes to the appliances, tables or walls, his/her personal home server stores the keys automatically to decrypt SSDP messages received from the appliances.

Our system may have a problem when multiple persons try to access the same appliance. For example, when two persons like to change the TV channel of the same television, the system changes the channel in an order to receive commands. Currently, we do not know whether we need a mechanism to solve the conflict among commands. We believe that solving the conflict is a social issue, and we need to examine the behavior of actual users at home to solve the problem. Currently, as one of the solutions to solve the problem, we are implementing a lock command to access appliances in an exclusive way. When a lock command is transmitted to an appliance, it returns an encryption key. The advertisement messages from the appliance is encrypted by the key while the appliance is locked, and SOAP requests from a user’s personal home server to the appliance should be encrypted by the key. Therefore, other persons’ personal home servers cannot control the appliance because they do not know the key. Also, the user interfaces in their personal home servers do not show the names and commands for the locked appliance. After an unlock command is received by the appliance, it stops to encrypt the advertisement messages.

### 3.5 Personalization Management

The PreferenceInference component allows services to be personalized by using stored preference rules that encode its user’s preferences. Let us assume that a URL received by the component contains “?function=TV”, and the statement is passed to a module for managing personalization. The module expands the statement by using rules specified in the presence service.

We assume that a personal home server contains the following rules:

- \( \text{Function}(x, \text{TV}) \) -> \( \text{Location}(x, \text{living-room}), \text{Support}(x, \text{BS}). \)
- \( \text{Function}(x, \text{light}) \) -> \( \text{Type}(x, \text{ceiling}), \text{Location}(x, y), \text{Location}(\text{me}, y), \text{Room}(y). \)

The PreferenceInference component customizes a presentation document according to a user’s preference. For example, let us consider the above rules. When the component detects several types of light appliances, the second rule filters light alliances whose type is not a ceiling light, and the presentation document contains only ceiling lights that reside in a room where a user is. The component omits information about other types of light appliances. Also, the presentation document contains information about televisions whose location is a living room and that support the BS digital\(^1\) according to the first rule. In this case, ?function=TV specified in the URL is expanded to ?function=TV&?support=BS&?locations=living-room by using the first rule, and the URL is included in an automatically generated presentation document. Thus, a television that supports the BS digital and resides in a living room is selected according to a user’s preferences. The example shows how a personal home server can generate a customized presentation document according to a user’s preferences.

### 3.6 User Interface Generation

The section presents an example showing how a personal home server generates an HTML document for controlling home appliances. In our current prototype system, the UI component generates HTML documents containing URLs to control appliances detected by the personal home server. The URLs are filters according to rules stored in the PreferenceInference component.

Our personal home server does not assume to have a display appliance. It uses a display near a user, and transmits an advertisement message to a display appliance. Then, the display appliance near him transmits a GET request to retrieve a presentation document from the personal home server. Finally, the document is rendered on a display by a Web browser in the display appliance. The approach enables us to create new attractive services. For example, an approach to deal with lights in the example allows us to trigger commands automatically when a presentation document is rendered.

### 3.7 Current Status

Our system has been written in Java, and we have adopted the OSGi(Open Service Gateway initiative) Service Gateway Specification as a component framework. In the current prototype, we have adopted Oscar which is a open source implementation of the OSGi service gateway specification. The system is current running on Linux, and we are using the Blackdown’s Java virtual machine. Our personal home server is currently implemented on HP’s iPAQ, and it can communicate with home appliances via IEEE802.11.

In this section, we describe how our prototype system works by using an example. In the example, we consider that a television and two lights are in a room. We assume that one is a ceiling light and another is a floor light. When a

\(^1\)One of Japanese digital satellite broadcasting methods
user enters a room, his personal home server in his pocket detects appliances by receiving SSDP advertisement messages containing IP addresses from the television and the lights. The personal home server accesses Web servers of both appliances, and retrieves their service specification documents. Then, the documents are stored in the Database component in a personal home server.

On the other hand, a display appliance near a user detects an advertisement message from the personal home server, and retrieves an automatically generated HTML document. The document contains URLs generated from information in service specification documents for controlling the television and a ceiling light according to our URL-based naming scheme. The user specifies that he likes to use only ceiling lights in a rule stored in the PreferenceInference component. Thus, the component removes information about a floor light in the presentation document.

The document is shown by a browser of the display appliance as shown in Figure 3. In this example, a user can navigate the browser by using a touch panel. When the user clicks a URL, http://102.10.2.2/?func=light&type=ceiling!power=on/ in the document, a GET command containing the URL is transmitted to the personal home server. The server searches a light appliance that matches specified attributes to the information stored in the Database component by comparing the func and type field in the URL and the func and type tag in service specification documents, and finds that its IP address is 102.10.2.10 specified in the light’s service specification document. Also, a command encoded in the URL is converted to a SOAP request, power("ON"). Finally, the SOAP request is transmitted to a target light appliance whose IP address is 102.10.2.10, then the appliance turns on its power. In our current prototype, light appliances are controlled via an X10 device, and an analog television is controlled through an intelligent remote control device.

4. CONCLUSION

In this paper, we have described the design and implementation of a personal home server. The personal home server configures home appliances near a user in a spontaneous way. Since each user has his own personal home server, the configuration is determined according to his preference. In traditional approach based on a home gateway, the configuration is determined according to a place where a user is, but our approach determines the configuration according to a person. We believe that our approach is more natural and simple since the configuration policy can be defined for each person. In the near future, each person has several personal devices anytime anywhere, and it is realistic to configure appliances by using a personal home server.

5. REFERENCES