On Dynamic Service Integration in VNA Architecture

Jin Nakazawa¹, Yoshito Tobe³, and Hideyuki Tokuda¹¹, Members

SUMMARY In forthcoming home network environment, computation capability will be embedded invisibly in home appliances, sensors, walls, ceilings, and floors. People will conduct various tasks using multiple devices simultaneously without consciousness of using computers. In this paper, first, we propose an application model named Virtual Network Appliance (VNA) model which simplify and expand device utilization. In the model, each device has VNA runtime system and function objects, called VNA components, running on it. A user task is defined in an application called VNA which is a logical appliance consisting of abstract function requirements and a message graph among them. Second, we propose Virtual Plug&Play mechanism which is a dynamic service integration mechanism in VNA model implementation. When a user conducts a task, he/she makes a VNA runtime system on a userside terminal load a VNA definition appropriate for the task. Virtual Plug&Play dynamically discovers required VNA components and establishes the message graph as defined. Since XML documents are used to describe a VNA, users can share and customize it easily. We call the device integration done by Virtual Plug&Play top-down integration, which existing middleware do not aim at. Finally, we show that Virtual Plug&Play affords practical performance for top-down integration by performance evaluation.

key words: information appliances, home networks, middleware, components, and VNA

1. Introduction

Home network is a heterogeneous and dynamic environment. It will consist of various devices including information appliances, sensors, and computers. It will be used by various people: children, elders, computer-aided people, homemakers, and handicapped people. In such an environment, they conduct various tasks using multiple device functions simultaneously. For example, when we play the video, we use a VCR, a TV, and lights. In addition, people are mobile even at home; we might want to use the nearest TV wherever we are. In such cases, the working set of device functions changes frequently. In consequence, we recognize that the home network is one of derivatives of the pervasive computing environment where new application models are needed[1].

This paper proposes a novel service integration architecture, named Virtual Network Appliance architecture (VNA architecture). In the architecture, each device is supposed to have manipulation objects, corresponding to each device function, running on VNA runtime system in the device. A user’s task and its working set of device function objects are represented by a logical appliance, called Virtual Network Appliance (VNA). A VNA is an autonomous system where device function objects interact each other using a messaging mechanism our architecture provides. Using an XML-based markup language (VNA Markup Language: VM-L), a VNA is defined as a set of function templates and a message graph among them.

The core mechanism of VNA architecture is Virtual Plug&Play. Virtual Plug&Play mechanism (1) loads a VNA definition, (2) dynamically constructs mappings between a function template in a VNA and a device function object, and (3) establishes a message graph among them.

Several university and industry groups are investigating toward integrating information appliances. HAVI[2], UPnP[3], and ECHONET[4] are the examples which provide primitive interfaces for programming device manipulation objects. Ninja[5] [6], MASH[7][8], TSpaces[9][10], and IHOMES[11][12] are examples aiming at composing integrated services by distributed device manipulation objects on user-side terminals. The basic idea is to use network objects[13] of applianceside manipulation objects together on a terminal, such as PDAs, to control independent appliances remotely. However, since a home network includes sensing devices and data transmission devices as well as control devices, “universal remote controller” mechanisms do not provide users with flexible utilization of the home network.

The advantage of VNA architecture is that users can deal with multiple information appliances as a VNA, in addition to remotely controlling them. Thus our architecture can be recognized as a “universal device interaction” architecture. Our mechanism enables users not only to define VNAs by themselves, but also to use pre-defined VNAs from third-parties. We call the former bottom-up integration, and the latter top-down integration. The top-down integration is our challenge which existing middlewares do not aim at. Since our architecture does not force users to install any kind of server programs, users can achieve flexible utilization of information appliances with little management cost.
Furthermore, adopting VML to describe logical appliances enables easy customization of them.

The rest of this paper is structured as follows. In section 2, we discuss problems of information appliances integration. Section 3 describes a system model of our architecture and details prototype implementation. Then in section 4 we propose some applications which highlight advantages of our architecture. Section 5 provides results of experiments done using the prototype implementation and evaluates our architecture. In section 6 and 7, we discuss related work and future work, respectively. Finally, section 8 concludes this paper.

2. Service Integration in Home Network

A home network consists of various appliances provided by various vendors for various people connected by various network architectures; simply put, it is a heterogeneous environment. In contrast, existing software and hardware technologies have been developed for computers and their skillful users; they are for homogeneous environment. For example in my apartment house where I live alone, there are over 10 appliances. Considering a house where a family lives, there might be 20 to 30 appliances. Therefore a question is, how should we design, implement, and provide a service integration middleware for such a heterogeneous environment?

2.1 System Requirement

Considering that people at home are not always computer-aided nor skillful, a service integration middleware for home networks should provide the “utilization costs free” mechanism. Such utilization costs can be categorized into three: installation, management, and operation costs. During series of use cases, a service integration middleware should include the following features.

• No need for installation cost
  – No initial operation such as installing a server program into a computer,
  – No monetary cost to purchase additional hardware, such as a server computer to utilize a target appliance.

• No need for management cost
  – No consecutive operations to manage a home network.
  – No knowledge about configuring a middleware.

• No need for operation cost
  – No operation to achieve dynamic adaptation of logical appliances.
  – No knowledge needed for using logical appliances in a middleware.

Each of section 2.2, 2.3, and 2.4 provides a discussion on design principles to achieve the abovementioned features.

2.2 Appliance-based and Function-based Integration

An information appliance has multiple functions. For instance, a VCR has four functions: power switch function, videotape I/O function, tuner function, and timer function. A TV also has four functions: power switch function, tuner function, audio player function, and video display function. Thus a middleware for these appliances can be designed in two ways: appliance-based and function-based integration. In appliance-based mechanisms like UPnP, each appliance has a manipulating object which is used as a remote-controller on user-side terminals. On the other hand, function-based integration mechanisms have the manipulation object for each function.

Adopting function-based approach will enable users to define a set of arbitrarily selected functions as a logical appliance. For example, a logical appliance “play a videotape” might be set to 1) turn a TV on; 2) turn a VCR on; 3) start playing the videotape, and 4) forward the A/V output from the VCR to the nearest TV. In the appliance-based mechanisms, a user needs to remotely command each step above.

2.3 Centralized and Distributed Service Management

A service integration middleware should manage and retrieve the following information: profile of appliances and their functions, location of appliances and users, and state of appliances, functions, and users. This information can be managed in two ways: distributed and centralized.

In the centralized management, all the information is stored in a server program, such as LDAP, SLP[14], and SDS[15]. Therefore users are forced to set up the server program and manage a server machine. Considering both installation and management costs of users, the centralized architecture is not appropriate for home networks.

The distributed management does not depend on any server programs. Instead, a directory mechanism in each appliance stores information of all the appliances in a network. Thus, the distributed “server-less” architecture can reduce installation and management cost of users. However, we should consider that it raises implementation cost since it requires more sophisticated mechanisms and hence richer hardware in each appliance: more amount of memory to store profiles and more powerful CPU to process the profiles.

2.4 Bottom-up and Top-down Service Integration

Defining logical appliances in function-based mechanis-
m is an undesirable task for users at home. Users are not always computer-aided and skillful enough to select functions they need and assemble them into logical appliances. We call such an integration manner done by users themselves bottom-up integration. Bottom-up integration is complicated and hence supposed to be an optional task.

Logical appliances should be portable. Once the logical appliance “play a videotape” is defined, sharing the definition among users eliminates the duplicate definition by individual users. Logical appliances might be saved in tiny storage, such as smart cards and floppy disks, or WWW server, and delivered. A user who acquired such a predefined logical appliance makes an integration middleware to load the definition and assemble an instance of it. We call this middleware-assisted integration manner top-down integration.

In top-down integration, a logical appliance should be dynamically reconfigured when availability of required functions has changed; for example when 1) an information appliance, which has the constituent function of the logical appliance, is moved from a room to another, 2) an information appliance, which has a function providing better QoS than the existing constituent function, is plugged in, and 3) the user of the logical appliance has moved to other room, requiring the function alliance to follow the user.

3. VNA Architecture

In this section, we first describe VNA model and overview system configuration of its prototype implementation. Second, we detail a directory mechanism and then Virtual Plug&Play mechanism along with VNA Markup Language.

3.1 VNA Model

VNA model integrates information appliances through two phases: distribution and integration. Figure 1 depicts an overview of VNA Model.

3.1.1 Distribution

In distribution phase, information appliances are decomposed into software components called VNA components, which correspond to functions of information appliances. A VNA component is named by component profile. Table 1 summarizes component profile records.

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the function</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor</td>
<td>Name of the appliance’s vendor</td>
</tr>
<tr>
<td>identifier</td>
<td>VNA identifier of the function</td>
</tr>
<tr>
<td>description</td>
<td>Free text description about the function</td>
</tr>
<tr>
<td>url</td>
<td>URL for related information</td>
</tr>
<tr>
<td>location</td>
<td>Location of the appliance</td>
</tr>
<tr>
<td>port count</td>
<td>Number of ports the component has</td>
</tr>
<tr>
<td>ports</td>
<td>List of port names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the port</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>VNA identifier of the port</td>
</tr>
<tr>
<td>description</td>
<td>Free text description about the port</td>
</tr>
<tr>
<td>type</td>
<td>input or output</td>
</tr>
<tr>
<td>data type</td>
<td>MIME type of data sent/received</td>
</tr>
<tr>
<td>protocol</td>
<td>Protocol name used in the port</td>
</tr>
<tr>
<td>destination</td>
<td>Destination list if type is output</td>
</tr>
<tr>
<td>source</td>
<td>Source list if type is input</td>
</tr>
</tbody>
</table>

A VNA component can own its child components. An information appliance, thus, contains multiple VNA components in a tree structure. We call this tree structure, which initially exists in each information appliance independently, a VNA domain. Multiple VNA domains on a network are united into a single VNA domain in an ad hoc manner. A VNA domain, which unites every VNA domains in a network, thus represents the whole home network. Let A represent the union of information appliances accessible in a home network, and D a VNA domain, then,

\[
D = \{ VNA \text{ components running in } a \in A \} \tag{1}
\]

A VNA component can also have message ports \{p_1, \ldots, p_n\}. Each port is named by port profile whose records are summarized in Table 2.

3.1.2 Integration

In integration phase, one or more VNA components in a domain are combined to make a logical VNA domain. This logical domain is what we call Virtual Network Appliance (VNA), and defined with VNA Markup Language (VML). In a VML file, constituent VNA components are specified by function templates which includes one or more attribute-value pairs.
Suppose that there are a function template \( t \) and a VNA component \( c \). Then
\[
t \equiv c \text{ or } c \equiv t
\]
indicates that all the attribute-value pairs specified in \( t \) matches \( c \)'s profile. Let \( L \) be a logical VNA domain, and \( V \) a VNA, then,
\[
L = \{ \text{function templates} \} \\
V = \{ c \in D \mid \exists t \in L, \ c \equiv t \}
\]
In addition to function templates, a VML file includes definition on interaction among constituent VNA components as a directed graph \( G \):
\[
G = (P, M, \text{source, destination})
\]
where \( P \) and \( M \) represent the union of ports and message paths, respectively.

Our mechanism called Virtual Plug&Play can be expressed by function \( f \) in formula 5.
\[
f(L, G) = V
\]
Bottom-up integration in VNA architecture is a user's operation which defines \( L \) from given \( D \). Top-down integration is the operation done in Virtual Plug&Play which creates \( V \) from given \( L \).

3.2 Configuration

We implemented a prototype system of VNA model using Java language to prove effectiveness of Virtual Plug&Play mechanism. Figure 2 depicts an overview of the system configuration.

3.2.1 Software Configuration

The prototype system of VNA model was implemented in Java language, with approximately 2MBytes of bytecode. It provides an interface named Serdget which represents VNA component, and runtime which loads Serdgets in each information appliance.

A runtime consists of two parts: core and extensions. The core part provides both primitive functionalities for loading and unloading Serdgets, and interfaces for programmers such as Serdget, VNA, and Domain. They respectively implement VNA component, logical VNA domain, and VNA domain in VNA model. Serdget is a Java interface used by programmers to implement VNA components which directly manipulate the hardware functions. Domain is implemented as a subclass of javax.swing.tree.MutableTreeNodeModel and manages all the loaded Serdget instances in a VNA domain. VNA is a class used in the runtime to combine multiple Serdgets. Therefore the core is the minimum set of the VNA architecture for programmers and users.

Major functionalities of the VNA architecture are implemented in the extension part as Serdgets in consideration of adding, removing, and updating functionalities for the runtime in an information appliance already shipped. The prototype implementation includes the following extensions.

**Domain Directory** is a Plug&Play functionality for solid information appliances (detailed in Section 3.3.) It synchronizes tree information among every information appliances on which VNA runtime exists. Every runtime includes Domain Directory extension.

**Virtual Plug&Play** is a functionality for loading, unloading, and reconstructing VNAs (detailed in section 3.4.) Every runtime includes Virtual Plug&Play extension.

**Domain Browser** is a GUI functionality for end-users (detailed in section 3.5.) Domain Browser is used on information appliances including computers where display functionalities are available: for example, TV and PDA.

**Session Manager** is a messaging functionality among Serdgets. Domain Directories communicate each other through the Session Manager. Every runtime includes Session Manager extension.

**Smart Card Manager** is a functionality which loads VNA definition from smart cards. Smart Card Manager is used in an information appliance where a smart card reader is attached.

**L-VNA Module** is a location tracking functionality for users and appliances[16].

3.2.2 Hardware Configuration

We suppose a VNA runtime is statically stationed in
each information appliance, computer, and PDA including a cellular phone. The runtime of computers and PDAs needs to be installed by users, while information appliances are shipped with the runtime embedded. Legacy appliances without VNA runtime are supported by connecting with VNA-ready computers through available data connections such as RS232C, X.10, and IEEE1394.

As a testbed for our architecture, we have built a room called Smart Space Laboratory (SSLab). SSLab has about 30 heterogeneous embedded information appliances: a projector, lights, a fan, an air conditioner, a Sony Play Station 2, and several kinds of A/V appliances including an IEEE1394-ready VCR. It is also equipped with many kinds of sensors such as weather sensors, Toshiba Motion Processor, illuminometers, and location sensors. Some of these appliances and sensors are connected to PCs, and some are directly connected to heterogeneous networks: IEEE802.11b cell, IEEE1394, and FastEthernet.

We suppose that a VNA can be saved in a smart card and delivered by a third-party. For users who acquired the smart card, we assembled a computer in the SSLab which has a smart card slot. The VNA runtime in the computer includes a smart card manager extension, and activates Virtual Plug&Play based on the VNA definition in a smart card inserted to the slot. Thus users can utilize a VNA only with one action.

Currently, we have implemented Serdgets for the following hardware: Pioneer plasma display, IEEE-1394 VCR, a fan, room lights, Yokogawa thermometer, E-code Spider RFID sensor, and a weather sensor.

3.3 Domain Directory

A Domain Directory is a directory service mechanism which manages profiles of all the VNA components in a network, the union D in section 3.1. To eliminate user's task originated by management, every VNA runtime is configured to hold Domain Directory and does not rely on any server program. Figure 3 depicts a configuration overview of Domain Directories in a home network.

3.3.1 Registration

Domain Directory mechanism is implemented in one of the extensions named VNADomainSerdget. When an information appliance has plugged in and a VNADomainSerdget is initialized by a runtime, it advertises itself using ADVERTISE packet on one-hop IP multicast channel. Each VNADomainSerdget which has received the ADVERTISE packet replies with an ACKNOWLEDGE packet using a UDP unicast. By these two packets, every VNADomainSerdget can recognize each other.

They subsequently exchange names and profiles of child nodes using CHILDREN packets and synchronize their tree configurations with sorting nodes in the instantiation order. After the first advertisement, the VNADomainSerdget re-advertises itself every advertisement interval time.\(^1\) If no advertisement is received from a certain VNADomainSerdget for a dead interval time\(^1\), the VNADomainSerdget and its child nodes are detached from the tree. Domain Directories, in consequence, register each other by flooding VNA component profiles, and provide high performance retrieval of component profiles without communicating with external directory servers.

Although registration is a time-consuming process, it occurs only when network configuration has changed; e.g. an information appliance is plugged in or off, an information appliance is moved, and power failure occurred. Since these configuration changes rarely occur, we consider that the advantage on high performance retrieval exceeds the disadvantage on the time-consuming registration.

3.3.2 Directory Interfaces

VNADomainSerdget provides the following two methods for searching Serdgets with attribute-value pairs used for matching with Serdget's profiles.

```java
public Identifier getSergetMatched(Serializable hints)
public Identifier[] getUnmatched(Serializable hints)
```

In both methods, the hints parameter contains attribute and value pairs as hints in java.util.Hashable entries. Both methods trace the Serdgets which the VNADomainSerdget manages, and determines candi-

\(^1\)The default value of advertisement interval is 10 seconds.

\(^1\)The default value of dead interval is twice of advertisement interval.
dates to return. The former returns the Identifier instance corresponding to a Serdget which has found firstly. The latter returns Identifiers of all the Serdgets which has matched the hints. VNA Domain Serdget provides derivative methods which exploit regular expression. Both of the methods in the current prototype implement linear search, which provides a sufficient performance for a network in SSLab exist.

VNA Domain Serdget also provides event notification mechanism and Java interfaces to detect network configuration changes. One of the interfaces is TreeModelListener which is used for detecting information appliances plugged in and off. The other interface is TableModelListener whose method is invoked when content of a VNA component profile changed. Virtual Plug&Play mechanism implements both of these interfaces to dynamically reconstruct VNAs. Both of these interfaces are defined as follows.

```java
package javax.swing.tree.event;
public interface TreeModelListener
extends java.util.EventListener {
    // Indicates a Serdget is created */
    public void treeNodesInserted(TreeModelEvent);
    // Indicates a Serdget is removed */
    public void treeNodesRemoved(TreeModelEvent);
}

public interface TableModelListener
extends java.util.EventListener {
    // Indicates a profile value changed */
    public void tableValueChanged(TableModelEvent);
}
```

3.4 Virtual Plug&Play

A logical VNA domain, a VNA definition and the union L in section 3.1, is defined using VNA Markup Language (VML). Appendix A shows the DTD of VML. Virtual Plug&Play is a mechanism which constructs VNA instance, the union V and a message graph G in section 3.1, based on the VML file and dynamically reconstructs it triggered by TreeModelEvent or TableModelEvent.

3.4.1 Loading and Reconstructing VNA

VNA class implements a logical VNA domain, the union L in section 3.1. It constructs a VNA, the union V in section 3.1, by mapping function templates to existing Serdgets using classes named MetamorphicSerдget and LookupSerдget. Both of the classes implement the mapping function, the function f in section 3.1. Figure 3.4.1 depicts the state diagram of these classes.

LookupSerдget is an abstract class, and configured to look up Serdgets by using getAllSerдgetsMatched method of the Domain Directory, mentioned in section 3.3.2. It gives attribute-value pairs specified in param tag in a VML file as hints parameter of the method.

And it adds all the found VNA components to the VNA created. MetamorphicSerдget is also an abstract class which is configured to search a Serдget using getSerдgetMatched method of the Domain Directory. And it metamorphoses to the Serдget. LookupSerдget and MetamorphicSerдget iterates searching if no Serдget was found.

MetamorphicSerдgets and LookupSerдgets cooperate with the Domain Manager through the event notification interfaces described in the section 3.3.2. When an information appliance is plugged in or off, then the Serдgets receive a TreeModelEvent. In addition, when the profile of a Serдget changed, the Serдgets receive a TableModelEvent. In both cases, the Serдgets iterate search operation and reconstruct a VNA.

If MetamorphicSerдgets or LookupSerдgets have found appropriate Serдgets, they establish message paths. A message path is configured by adding new values in the destination or source attributes of the port profile shown in Table 2.

3.4.2 VNA Markup Language

Function templates and a message graph are defined in a VML file. Users and vendors can edit a VML file manually, or construct it using GUI provided by VNA architecture. Since VML is a markup language, not a programming language, a VNA definition is highly portable among networks. Thus we suppose that a VNA definition can be saved in smart cards or small memory devices by third-parties, and delivered to users. In consequence, the users can achieve top-down integration without installation costs.
The description above is a skeleton of a VNA definition. A VNA has a set of a name, a description, and a URL which is defined in a head tag. A head tag is followed by body tag where function templates and messaging paths are defined.

A function template can be described using VML by one of the following three manners.

1. Specify a MetamorphicSerget and hints to map one existing Serget in a home network.

```xml
<component source="DefaultMetamorphicSerget.xml" name="name of Serget"/>
```

2. Specify a LookupSerget and hints to map every existing Sergets in a home network.

```xml
<component source="DefaultLookupSerget.xml" name="name of Serget">
  <param>
    <default/>
    <param name="name" value="printer"/>
    <param name="location" value="around here"/>
  </param>
</component>
```

3. Specify other VNA components in component tag of VML as follows, resulting in a new VNA component instance to be created by the runtime.

```xml
<component source="ButtonClick.xml" name="button1" position="Center"/>
```

A message graph is defined as a set of message paths between two Sergets using a message tag as follows.

```xml
<message from="SOURCE_PORT_ID" to="DESTINATION_PORT_ID"/>
```

In the definition, from and to attributes represent the last two elements in Formula 4.

### 3.5 User Interface

In addition to the abovementioned mechanisms in the runtime, the current prototype system provides users with a simple GUI which is used on user-side terminals for browsing Sergets and their profiles in a home network. Figure 3.5 is a screenshot of the GUI named Domain Browser. Domain Browser window consists of two parts: tree view and profile view. Tree view is a front-end of Domain Directory and shows a list of Sergets in a tree structure, where leftmost nodes represent information appliances in the network. When a tree node is single-clicked, profile of its corresponding Serget is shown in the profile view.

When a tree node is double-clicked, the corresponding Serget's GUI is shown to be used as a remote controller. The GUI is implemented in each Serget as a migratory object and Domain Browser acquires its instance using Object Transport extension in VNA runtime. Suppose a GUI for a TV power Serget which simply includes “power on” and “power off” buttons. When one of the buttons is clicked, the corresponding method of the Serget is invoked remotely. Since Domain Browser includes tree nodes for all the Sergets managed in Domain Directory, it can be used as a universal remote controller.
4. Applications

This section describes some applications. Each of them highlights an advantage of one action integration, VNA portability, and dynamic reconfiguration of our architecture.

4.1 Wearable Entrance

Using an E-code spider location sensor, a light, a Sony EVI-D30 camera and our transcoding mechanism called TranService, we have built Wearable Entrance VNA (WEV). Figure 4 shows the configuration of WEV.

A WEV detects a person coming in front of the entrance using the location sensor, and generates an event. The event is delivered to the camera and the entrance light. On event reception, the entrance light turns itself on and the camera changes its angle to shoot a picture of the person. The picture is sent to the cellular phone gateway. Then the gateway changes the size and color depth of the picture, and sends it to the cellular phone. In consequence, a user of WEV can be aware of visitors on the cellular phone wherever he or she is.

Appendix B shows WEV definition in VML. Each of constituent VNA component is specified using DefaultMetamorphicSerdget, and they are connected through message ports. If Virtual Plug&Play has failed, for example there is no Serdget whose type is camera, the DefaultMetamorphicSerdget makes a transition to sleep phase. Domain Browser also requests the user to install a camera into the network.

In this example, when a user injects a WEV definition to a VNA runtime, WEV is assembled with ubiquitous inexpensive devices by Virtual Plug&Play mechanism. Users do not need specific knowledge about the integration nor each integrated devices. Thus users can exploit sophisticated functionality such as WEV with little installation cost.

4.2 Intensive Air Conditioner

Suppose an Intensive Air Conditioner (IAC) VNA which consists of an air conditioner, a fan, and a thermometer, whose definition is described in Appendix C. VNA components, running in both the air conditioner and the fan, have ports through which their power can be controlled. The thermometer is connected to a PC and VNA component for the thermometer is running on VNA runtime in the PC. VNA component for the thermometer holds temperature threshold which is defined in a VNA definition.

When a user virtually plugs in IAC, message paths are established between the thermometer and both the air conditioner and the fan. When the thermometer VNA component detects that temperature has got higher than the threshold, it sends messages to on ports of both the air conditioner and the fan. They start cooling and temperature drops to less than the threshold. The thermometer VNA component detects it and sends messages to off ports of them. By series of this cooperation, IAC keeps temperature around the threshold.

Although IAC is a very simple VNA, portability of VNA provides an advantage for the user. The user can carry IAC saved in a smart card, and virtually plug it in where the user is. The IAC is instantiated, including mappings and message paths, by Virtual Plug&Play mechanism. Therefore, the user can reflect one’s preference, temperature in this case, wherever the user is.

4.3 Virtual VCR

We have built a VNA called Virtual VCR (VVCR) using a plasma display, a TV, and a VCR. VNA components running in both the plasma display and the TV have ports to control the power and the input source. The VCR is connected to IEEE1394 bus and a VNA component in it has ports to play, pause, stop, forward, and rewind the videotape. Figure 4.3 shows the configuration of VVCR and its definition using VML is shown in Appendix D.
When a user loads a VVC R by inserting a smart card which includes the definition, a VVC R is assembled by a TV nearest to the user, and the VCR in the living room. This assembly is done by the Virtual Plug & Play mechanism in the runtime where the definition is loaded. To search VNA components, Virtual Plug & Play passes hints specified in the param tags to the Domain Directory. For example, [location=livingroom, type=vcr] is passed to the Domain Directory in case of the VCR in the definition. Then the output audio/video stream of the VCR is forwarded to the display. The user gets audio/video outputs from the VCR simply injecting the videotape and the smart card.

When the user moves to another room, L-VNA Module extension detects it and makes Domain Directory to notify a TreeModelEvent for DefaultMetamorphicSerdget named tv. The VVC R intergrades to configuratoin B in the Figure 4.3. The DefaultMetamorphicSerdget subsequently makes transition to search phase, and VVC R is reconstructed with another display device nearest to the user. At the same time, the destination attribute of port of the VCR is replaced by the newly discovered display device’s input. Here, the user achieves location-adaptive continuous utilization of information appliances without any operation costs.

5. Evaluation

Our middleware, VNA architecture, runs on heterogeneous information appliances. Since there are information appliances whose resources are limited, runtime should be lightweight. We evaluated time costs needed to load a Serdget and analyzed resource consciousness. We also analyzed scalability by evaluating time costs for constructing the domain directory. Experiments were done using two target appliances, which are summarized in Table 3.

Although the evaluation herein does not include variability in network loads, it provides the foundation of evaluating computational cost. Quantifying it is important for a system where dynamic loading and configuration occur.

5.1 Appliance Booting

When an appliance is plugged in, a runtime in it starts running a Serdget by the following 5 steps.

A runtime

Step 1 loads a VCML file where its Java archive file name and entry class name are defined,

Step 2 obtains a file handle for the Java archive file,

Step 3 loads bytecode for the entry class,

Step 4 creates an instance of the Serdget,

Step 5 and sequentially invokes init() and start() methods of the Serdget.

Steps 1 to 3 are file system intensive operations, while step 4 is memory intensive. Since time cost for the step 5 depends on each Serdget, let us exclude step 5 for this evaluation. We, thus, measured elapsed time for both steps 1 to 3 and 4 on the following two configurations on the target appliance B.

Configuration 1 We varied the size of the entry class from 1.1KBytes to 3.3KBytes keeping the size of the Java archive file approximately to 0.2MBytes.

Configuration 2 We varied the size of Java archive file from 1.1KBytes to 3.3KBytes keeping the size of the entry class to 874Bytes.
Elapsed times needed for steps 1 to 4 on each configuration are summarized in Figure 9 and 10, respectively. On both configurations, the time cost for step 4 was constantly 2.2msec. In consequence, loading Serdgets is more sensitive to the bytecode size of the entry class.

We suppose that usual appliances have no more than 10 functions. Therefore in such appliances, 10 VNA components are loaded on booting. From the experiments, loading 10 VNA components consumes approximately 80 mili seconds if their bytecode sizes are 3.3KBytes. In consequence, we consider the current prototype provides practical performance for appliance booting.

5.2 Constructing Domain Directory

When an information appliance is plugged in, VNA runtime on each information appliance reconstructs Domain Directory. Since the reconstruction process includes flooding profiles of all the VNA components on each runtime, its performance determines scalability of our system. We suppose that there are approximately 30 information appliances at a typical home, including individual lights. We also suppose that typical information appliances have 10 to 20 Serdgets including extensions we provide. In such a home network, we consider all the reconstruction process should be finished in 1 minute.

In this experiments, we will figure out how many information appliances the current prototype can manage. We measured elapsed time for a reconstruction process when appliance B joined a appliance A's VNA domain. Configurations on appliance A and B is shown in Table 3. We configured, for simplicity, both VNA runtimes to have the same number of Serdgets.

Figure 11 summarizes results for 6, 16, 26, 36, 46, and 56 Serdgets cases. The result shows that reconstruction process between two VNA runtimes spends approximately 1 second if they have 16 Serdgets for each. Assuming there are n information appliances in a home network, there are \( \binom{n}{2} \) such combinations of two VNA runtimes in the network. On this situation, time cost needed for all the VNA runtimes to finish reconstructing Domain Directory is shown in Figure 12. Figure 12 shows that the current prototype can manage no more than 11 information appliances to finish the reconstruction process in 1 minute. To provide users with smooth utilization of our system, we need implementation enhancement.

5.3 VNA Booting

When a user conducts top-down integration using a pre-defined VNA, Virtual Plug&Play maps its function templates to existing Serdgets. To figure out the practical number of function templates in a VNA, we measured time costs for the mapping operation changing the number of function templates in the VNA. The result is shown in Figure 13.

By linearization, it is found that mapping operation takes approximately 4 mili seconds for each function template. Our assumption is that commonplace VNA's include less than 30 function templates, which take approximately 120 mili seconds for mapping operation. The result here shows that our distributed
management mechanism implemented in the Domain Directory provides users with high search performance and high scalability for the top-down integration.

6. Related Work

Jini[18], ECHONET, HAVi and UPnP include Plug & Play mechanisms for integrating information appliances. Although Jini provides users with ability to construct logical appliances in Java programs, users need to install the programs and a directory server called Jini Lookup Server. Jini is, thus, not enough for them to achieve both bottom-up and top-down integration without high operation cost. HAVi integrates A/V appliances connected by IEEE1394 network. And UPnP includes an integration mechanism on top of HTTP. Although they standardize interfaces for device manipulation objects, they lack mechanisms for constructing compositional services.

Several projects and products, described in the following paragraphs, propose service composition framework in addition to the primitive interfaces for objects.

The Ninja project at UCB provides a facility called Base, which is a Java-based clustered environment of homogeneous nodes for building and executing infrastructure services[6][3]. Their Dynamic Redirector Stub provides a single interface of a service for their clients, and masks load-balancing and fail over mechanism in the cluster. Although they achieve remote-control capability for distributed services from the sophisticated three-layered middleware, it forces users a high management cost if adopted at home and thus is rather developer-oriented. Our middleware is a user-oriented mechanism with support for top-down integration and server-less distributed components.

Munson et al.[19] propose a device control architecture using TSpaces[9] aiming at integrating information appliances as "universal information appliances." They classified network integration solution into the IP thin-waist and the middleware thin-waist ones. Based on these two solutions, they provide a gateway middleware as a proxy for legacy appliances. As an implementation language of universal information appliances, the MoDAL[10] is being developed. It is an XML-based user interface markup language built on top of the TSpaces. The MASH project at UCB proposes a document-based framework for manipulating the components that comprise distributed Internet applications[20][7][8]. In their framework, XML documents are also used to describe both server-side functionality and the mapping between a client's applications and the servers it accesses. Although they are taking a similar approach which adopts XML for service integration, the XML document lacks in defining an interaction mechanism which VML provides.

The intelligent home project (IIHOME) at the U-MASS multi-agent systems lab is an exploration in the application of multi-agent systems technology to manage home networks and information appliances[11][12]. They recognize water, hot water, electricity, etc. as shared resources. They have implemented a simulated home environment, where distributed home-control agents control appliances and negotiate over the shared resources. Although their abstraction including shared resources is significant, the abstraction is only limited to a control aspect of appliances. In contrast, our approach covers a design aspect of logical appliances in a flexible fashion with composition and decomposition of functions.

7. Ongoing Work

Our ongoing work involves in optimizing current prototype implementation and investigating various approaches.

The next step of this works is enhancing messaging functionality. Since Domain Directory and Session Manager extensions in the current prototype can deal only with a single broadcast domain, a VNA can not overlay more than one home networks. For example, a "Video Phone VNA" which consists of two sets of a microphone and a TV, one at the source and another at the destination home network. To realize such a VNA, we are investigating enabling the Domain Directory extension to unify multiple distributed home networks in an ad-hoc manner.

For the further development, VML is to be extended to include user interface definition. For example, assume that the "Video Phone VNA" needs the name of destination home network indicated, such as grandfather's home. Rather than providing only graphical user interface mechanism as what Domain Browser includes, we'd prefer to support oral and gesture input/output as well. As we mentioned in Section 6, there are several researches aiming at user interface description using XML-based languages such as MoDAL[10], XUL[21], UML[22], and ISL[8]. Although they all provide sophisticated description power, they support only graph-
ical user interface. One approach we are investigating is exploiting XSL [23] and the Cascading Style Sheets [24, 25] which support auditory rendering.

When a user conducts top-down integration using pre-defined VNA, the user's preference should be involved in the mapping operation. We are aiming to implement three policies for the operation: environmental mapping, QoS mapping, and time mapping. For the three policies to work correctly, various sensors should be integrated to provide environmental information such as humidity, temperature, and location in a common way. Although we have an implementation of location aware service management mechanism, other sensors embedded in SSLab should be used flexibly.

We have not standardized typing and naming policy for Serdgets and their ports. Function templates can be described with specification on vendor, description, or name attributes in the component profile using regular expression. However, a typing standard facilitates to guarantee that a third-party VNA works correctly. The current prototype system does not provide enough capability for specific description of function templates. We are investigating ways of such a specific description of both function templates and message graphs. One strict manner is that we have a concrete standard over all of the information appliances and sensors. Since a strict standard, however, lacks flexible adaptation for a new information appliance out of the standard, we have been implementing a thesaurus instead of it.

8. Conclusion

In this paper, we proposed a top-down integration scheme called Virtual Plug&Play for Virtual Network Appliance architecture using VNA Markup Language. VNA is assembled with arbitrary functional components running in each information appliances. With VNA, thus, users can integrate information appliances, regardless of their physical boundaries. Since VNA architecture includes an interaction mechanism among device function objects, users are able not only to control appliances remotely, but also to deal them as autonomous systems.

We classified these integration approaches into bottom-up and top-down integration. An advantage of our middleware over other researches aiming at service integration is that a definition of integration is portable since it is described using VNA Markup Language. Vendors will be able to ship VNA definitions saved in smart cards or tiny memory cards, and users obtain and utilize the pre-defined VNAs on their own home networks. This is the way of integration which we call top-down integration. Our server-less distributed architecture, implemented in the Domain Directory and Virtual Plug&Play, made it possible with little management and operation costs, while other researches require much higher.

Finally we evaluated the current prototype implementation. The results have demonstrated that prototype implementation of Virtual Plug&Play affords practical performance for top-down integration.

Acknowledgements

The authors would like to thank the members of both RT-HDI and HOME projects for their advice.

References

Appendix A: VML DTD

<!ATTLIST composite (head, body)>  
<!ATTLIST head (name, comment,seealso, information*)>  
<!ATTLIST name (#PCDATA)>  
<!ATTLIST comment (#PCDATA)>  
<!ATTLIST seealso (#PCDATA)>  
<!ATTLIST information (#PCDATA)>  
<!ATTLIST body (params|component|message*)>  
<!ATTLIST body name CDATA #REQUIRED>  
<!ATTLIST component (params|component)>  
<!ATTLIST component source CDATA #REQUIRED>  
<!ATTLIST component name CDATA #REQUIRED>  
<!ATTLIST component position CDATA #IMPLIED>  
<!ATTLIST params (default|if)>  
<!ATTLIST default (param)>  
<!ATTLIST if (if|param)>  
<!ATTLIST if condition CDATA #REQUIRED>  
<!ATTLIST param EMPTY>  
<!ATTLIST param name CDATA #REQUIRED>  
<!ATTLIST param value CDATA #REQUIRED>  
<!ATTLIST message EMPTY>  

Appendix B: The Definition of WEV

<html version="1.0">  
<!DOCTYPE composite SYSTEM "composite.dtd">  
</composite>  
<head>  
<!-- We do not provide definitions here -->  
</head>  
</body>  
</html>

Appendix C: The Definition of LAC

<html version="1.0">  
<!DOCTYPE composite SYSTEM "composite.dtd">  
</composite>  
<head>  
<!-- We do not provide definitions here -->  
</head>  
</body>  
</html>
Appendix D: The Definition of VVCR

<composite xmlns="http://www.w3.org/1999/xhtml">

<compart source="DefaultMetamorphicSerdget.xml" name="vcr" />
<default>
<param name="location" value="livingroom" />
<param name="type" value="vcr" />
</default>

</composite>