A Top-down Integration Scheme for Networked Appliances using VNA Markup Language

Jin Nakazawa¹  Tadashi Okoshi¹  Yoshito Tobe¹  Hideyuki Tokuda¹,²

¹Graduate School of Media and Governance, Keio University  ²Faculty of Environmental Information, Keio University

This paper presents a top-down service integration scheme for heterogeneous information appliances on heterogeneous networks for various users, called Virtual Network Appliance (VNA) architecture. Instead of merely connecting appliances seen in existing approaches, VNA architecture accommodates creating a logical appliance called a VNA. VNA Markup Language is our solution to define VNAs independent of specific home networks. Our new mechanism provides users with top-down integration as well as bottom-up integration. Users can download or even purchase pre-defined VNAs from third parties and dynamically configure them in the top-down integration, while users can create a logical appliance by assembling functions by themselves. 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.

1 Introduction

Information appliances are heterogeneous devices: their network interfaces, network protocols, CPU architectures, and operating systems might be different. Thus we need a unified middleware for integrating services provided by each information appliance. But being different from middlewares for computer networks, we cannot count on much configuration to users in home networks. Since they are not always computer-aided, it is desirable that a home network and a middleware on it are dynamically reconfigurable without any configuration by hand when an information appliance is plugged in the home network.

There have been an abundance of activities to standardize an interface for interoperability between information appliances. HAVi [1] is intended to support A/V devices with IEEE 1394 interfaces. Jini [2] is a system which integrates Java-based services based on a mobile code technique. UPnP [3] combines VNAs with user applications using HTTP, thereby enabling a control through Web browsers. However, these standards are merely intended to connect appliances and do not provide users with dynamically reconfigurable schemes.

This paper proposes a novel service integration infrastructure, named Virtual Network Appliance architecture (VNA architecture)[4], which eliminates users load to utilize the home networks by top-down integration scheme. Aiming at integrating heterogeneous appliances on the home networks, we focus on composing logical appliances, named VNAs, by assembling functional components regardless of physical boundaries of the appliances. A VNA is defined as a set of function templates using an XML-based markup language (VNA Markup Language: VML). To make VNAs portable among home networks, we designed, implemented, and evaluated a new mechanism called Virtual Plug&Play.

The rest of this paper is structured as follows. In section 2, we discuss design issues of information appliances integration. We address the point of view on bottom-up and top-down integration, and centralized vs. distributed configuration. Section 3 describes an overview of our model and its implementation, then in section 4, we propose an application. In section 5, we show results of experiments done on the prototype implementation. In section 6, we discuss related works. In section 7, we summarize our ongoing works, and section 8 concludes this paper.

2 Design Issues

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 environments. Consequently a middleware aiming to integrate information appliances need to provide new mechanisms for supporting heterogeneity. This section describes design issues on such mechanisms.

2.1 Top-down vs. Bottom-up

Since defining logical appliances in function-based mechanism is complicated and undesirable task, portability of them is essential. Logical appliances assembled by Jini is portable with assist of Jini Lookup Service. They are however implemented in Java language and thus force users to install them in some computers. To eliminate users’ burden originated by the installation, a logical appliance should be defined as a set of abstract function requirements, called function templates in this paper, using a markup language. The integration middleware assembles an instance of the logical appliance from those function requirements. It can be considered that there are two ways of integration in function-based mechanisms: bottom-up and top-down.

Bottom-up Users create logical appliances by assembling functions on their own.

Top-down Users acquire pre-defined logical appliances and inject them into a middleware.

2.2 Distributed vs. Centralized

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. These
information can be managed in two ways: distributed and centralized.

In the centralized management, all the information is stored in a server program. 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, 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.

2.3 Implementation Cost and Utilization Cost

An implementation cost absorbed by a vendor raises consumers’ payments for an appliance. But our assumption is that technological breakthrough will make the whole hardware more affordable. For example, a set of CPU, RAM, and storage needed for a Java VM to run will be much cheaper than it is now. Thus, we have been ignoring the vendors’ implementation costs, and taking a user-oriented design principles as follows: top-down integration and distributed management.

Based on aforementioned design issues, our top-down integration scheme should provide the following "costs free" mechanism to reduce users' utilization costs.

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

- **No need for management cost**
  - No consecutive operation to manage a home network.

- **No need for operation cost**
  - No operation to achieve dynamic adaptation of logical appliances.
  - Less knowledge needed for using logical appliances in a middleware.

3 Virtual Network Appliance Architecture

VNA architecture consists of VNA runtime and VNA components. Runtime is supposed to run in each information appliance. A VNA component is an object which represents each function in an appliance. In current prototype system, implemented in Java language, VNA component is implemented by Sereget interface.

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 the information appliances. A VNA component is named by a component profile. Table 1 summarizes component profile records.

A VNA component can also have its child components. An information appliance, thus, has 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,

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

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

\[
t \equiv c
\]

indicates that all the attribute-value pairs specified in t matches c's profile. Let T be a logical VNA domain, and V a VNA, then,

\[
L = \{\text{function templates}\}
\]

\[
V = \{c | c \in D, t \equiv c, t \in L\}
\]

Our new mechanism called Virtual Plug&Play can be expressed by function \( f \) in formula 4.

\[
f(L) = V
\]

Bottom-up integration in VNA architecture is a user's operation which define \( L \) from given \( D \). Top-down integration is the operation done in Virtual Plug&Play which creates \( V \) from given \( L \).
3.2 Domain Directory

A Domain Directory 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 have Domain Directory and does not rely on any server program.

Domain Directory is implemented in one of VNA components, called VNADomainSerget. When a VNADomainSerget is initialized by a runtime, it advertises itself using ADVERTISE packet. VNADomainSergets which received the ADVERTISE packet reply with ACKNOWLEDGE packet. By these two packets, every VNADomainSerget can recognize each other. They subsequently exchange names and profiles of child nodes using CHILDREN packets and synchronize their tree configurations. After the first advertisement, the VNADomainSerget re-advertises itself every advertisement interval time.\(^1\) If no advertisement is received from a certain VNADomainSerget for a dead interval time,\(^2\) the VNADomainSerget and its child nodes are detached from the tree. Domain Directory, in consequence, realizes dynamic unification and disunion of VNA domains.

3.3 VNA Markup Language

VNA Markup Language (VML) is an XML-based language used to markup a logical VNA domain, a VNA definition and the union \(L\) in section 3.1. Users and vendors can hand-edit a VML file, 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 is 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.

```
<xml version="1.0">  
  <!DOCTYPE composite SYSTEM "composite.dtd">  
  <composite>  
    <name>The name of this VNA</name>  
    <comment>A description of this VNA</comment>  
    <seealso>A URL related to this VNA</seealso>  
  </head>  
  <body source="VNA.xml" name="">  
    <!-- FUNCTION TEMPLATES ARE DEFINED HERE -->  
    <!-- MESSAGING PATHS ARE DEFINED HERE -->  
  </body>  
</composite>
```

The description above is a skeleton of a VNA definition. A VNA has a set of name, a description, and a URL and it is defined in a head tag. A head tag is followed by a 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 match one existing Serget in a home network. MetamorphicSerget is a VNA component which searches an adequate VNA component in the domain directory based on the given attribute-value pairs in the param tag.

   ```
   <component source="DefaultMetamorphicSerget.xml"  
               name="name of Serget">  
     <params>  
       <default>  
         <param name="id" value="A VNA Identifier"/>  
       </default>  
     </params>  
   </component>
   ```

2. Specify a LookupSerget and hints to match every existing Sergets in a home network. LookupSerget searches adequate VNA components in the domain directory based on the given attribute-value pairs in the param tag. And it adds all the found VNA components to the VNA created.

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

   ```
   <component source="Button.xml" name="button1"  
              position="Center">  
     <message from="SOURCE_PORT_ID"  
              to="DESTINATION_PORT_ID"/>  
   </component>
   ```

VNA components have message ports. By connecting them each other, VNA components can cooperate as an improvised system. In VML, a messaging path between two VNA components is defined as a message path using message tag as follows.

3.4 Virtual Plug&Play

CompositeSerget class implements a logical VNA domain. Virtual Plug&Play creates VNA instances, the union \(V\) in section 3.1, by mapping function templates in the definition to existing VNA components. Both LookupSerget and MetamorphicSerget implement the mapping function, the function \(f\) in section 3.1. They cooperate with the Domain Directory through Java event delivery mechanism. Suppose that an information appliance is plugged off, and thus VNA domain of the appliance is detached. Then, the Sergets receive a TreeModelEvent from the Domain Directory. Since the event suggests that the tree configuration has been changed, MetamorphicSergets and LookupSergets iterate invoking lookup method. The event is also delivered whenever a profile of a VNA component changed. Meta-

\(^1\)The default value of advertisement interval is 10 seconds.
\(^2\)The default value of dead interval is twice of advertisement interval.
morphicSerdgets and LookupSerdgets are therefore dynamically adaptive to configuration of their VNA domain.

4 Application

As a testbed for our architecture, we have built a room called Smart Space Laboratory (SSLab)[5]. SSSLab has heterogeneous information appliances embedded: 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 also has many kinds of sensors such as weather sensors, motion processors[6], 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 have built a VNA called Virtual VCR (VVCR) using a E-code spider location sensor, 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. The VCR is defined by VML as follows.

When a user loads a VVR by inserting a smartcard which includes the definition, a VVCR is assembled by a location sensor, a TV nearest to the user, and the VCR in the livingroom. 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 of the vcr is connected to the display. The user gets audio/visual outputs from the vcr simply injecting the videotape and the smartcard.

When the user moved to another room, the location sensor detects it and triggers dynamic reconstruction of the VVCR. The Virtual Plug&Play iterates searching the display nearest to the user. Here, the user achieves location-adaptive utilization of information appliances without any operation costs.

5 Evaluations

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 Seridget 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 2.

### Table 2: Appliances used for evaluation

<table>
<thead>
<tr>
<th>Appliance A</th>
<th>Appliance B</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>TYPE</td>
</tr>
<tr>
<td>Yokogawa DUONUS[7]</td>
<td>IBM Thinkpad 600X</td>
</tr>
<tr>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td>AMD AM636DX-153</td>
<td>Intel PentiumMM 650MHz</td>
</tr>
<tr>
<td>RAM</td>
<td>RAM</td>
</tr>
<tr>
<td>32MB</td>
<td>192MB</td>
</tr>
<tr>
<td>NIC</td>
<td>NIC</td>
</tr>
<tr>
<td>10Base-T Ethernet</td>
<td>10Base-T Ethernet</td>
</tr>
</tbody>
</table>

5.1 Appliance Booting

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

A runtime

**Step 1** loads a VCML file where it’s 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 Seridget.

**Step 5** and sequentially invoke init() and start() methods of the Seridget.

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 Seridget, let us exclude step 5 for this evaluation. We, thus, measured elapsed time for both steps 1 to 3 and 4 on following two configurations on the target appliance B.

**Configuration 1** We varied the size of the entry class...
Figure 2: Time cost for loading Serdgets(1)

Figure 3: Time cost for loading Serdgets(2)

Figure 4: Time costs for mapping operation

from 1.1KBytes to 3.3KBytes keeping the size of the Java archive file approximately 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 874Bytes.

Elapsed times needed for steps 1 to 4 on each configuration are summarized in Figure 2 and Figure 3, 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 spends approximately 80 mill seconds if their bytecode sizes are 3.3K-Bytes. In consequence, we consider the current prototype provides practical performance for appliance booting.

5.2 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 scalability of this operation, we measured time costs for the mapping operation changing the number of function templates in the VNA. The result is shown in Figure 5.2

By linearization, it is figured out that mapping operation takes approximately 4 mill seconds for each function template. Our assumption is that commonplace V-NAs include less than 30 function templates, which take approximately 120 mill 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

Many projects and products propose distributed component platforms exploiting mobile code techniques[8]. 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[9][10]. Their Dynamic Redirector Stub provides a single interface of a service for their clients, and masks load-balancing and failover 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.

The MASH project also at UCB proposes a document-based framework for manipulating the components that comprise distributed Internet applications[11][12][13]. In their framework, XML documents are used to describe both server-side functionality and the mapping between a client’s applications and the servers they access. Although they are taking a similar approach with us which adopts XML for service integration, the XML document lacks in defining an interaction mechanism which VML provides.

The Service Location Protocol[14] and the Service Directory Service[15] are generic resource discovery and service registration mechanisms. They exploit client/server architecture, which brings users an additional task. As aforesaid, a single centralized server architecture raises users’ management and operation cost. Our middleware, thus, has symmetric data structure for managing Serdgets in the Domain Directory.

Munson et al.[16] propose a device control architecture using TSpaces[17] 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 middle-
ware as a proxy for legacy appliances. As an implementation language of universal information appliances, the MoDAL[18] is being developed. It is an XML-based user interface markup language built on top of the TSpaces.

7 Future Work

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

For the further development, VML is to be extended to include user interface definition. One approach we are considering is exploiting XSL[19] and the Cascading Style Sheets[20][21] which support auditory rendering.

When a user conducts top-down integration using predefined 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.

8 Conclusions

In this paper, we proposed a top-down integration scheme for Virtual Network Appliance architecture using VNA Markup Language. VNA is assembled with arbitrary functional components running in each information appliance. With VNA, thus, users can integrate information appliances, regardless of their physical boundaries. 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 sensible and 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


