<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Adopting Model-Driven Development for Integrated Services and Appliances in Home Network Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Igaki, Hiroshi; Nakamura, Masahide; Matsumoto, Ken-ichi; Aoyama, Mikio</td>
</tr>
<tr>
<td><strong>Citation</strong></td>
<td>APSEC’06: 2006 13th Asia Pacific Software Engineering Conference, 6-8 Dec. 2006, Kanpur, India</td>
</tr>
<tr>
<td><strong>Issue Date</strong></td>
<td>2006</td>
</tr>
<tr>
<td><strong>Resource Version</strong></td>
<td>author</td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>© 2006 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.</td>
</tr>
<tr>
<td><strong>DOI</strong></td>
<td>10.1109/APSEC.2006.17</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10061/12761">http://hdl.handle.net/10061/12761</a></td>
</tr>
</tbody>
</table>
Adopting Model-Driven Development for Integrated Services and Appliances in Home Network Systems

Hiroshi Igaki1, Masahide Nakamura2, Ken-ichi Matsumoto2, Mikio Aoyama1
1Department of Information and Telecommunication Engineering, Nanzan University, 27 Seirei, Seto 489-0863, Japan
igaki@nanzan-u.ac.jp, mikio.aoyama@nifty.com
2Graduate School of Information Science, Nara Institute of Science and Technology, Japan
{masa-n, matumoto}@is.naist.ac.jp

Abstract

The technology of a home network system (HNS) allows integration of several kinds of home appliances to provide a user with value-added integrated services. Development of the integrated HNS services require implementation of the appliance components (with APIs) and the services, according to each home-network environment. There are various implementation standards such as DLNA, ECHONET, OSGi and Jini for the HNS applications. Therefore, even if a developer can choose the optimal one, it’s very difficult to develop the HNS integration services by composing a new HNS implementation. In this paper, we propose a model-driven development of integrated HNS service applications. In our former manuscript, platform-independent design language for verifying HNS service scenarios was proposed. Our model-driven development method uses this design language as a meta-model of integrated HNS services. By model transformation to concrete implementation together with verification by SMV (Symbolic Model Checking), productivity and quality of the integrated HNS services are improved.

1 Introduction

A home network system (HNS) is an emerging domain of ubiquitous applications that intends to provide users with smart and convenient home services [8][13]. A HNS consists of multiple networked appliances. The appliances include general household appliances and sensors, such as TVs, DVDs, ventilators, air-conditioners, thermometers, which are connected to LANs at home. Each appliance usually exhibits a set of device control interfaces (i.e., APIs) to the network, by which the users or external software agents can control the appliances via the network. The communication among appliances are performed with an underlying HNS protocol.

An advantage of HNS lies in integration of features of multiple appliances via the network. For instance, integrating a thermometer, a ventilator and an air-conditioner would implement energy-saving air-conditioning service, typically called an HVAC service as illustrated in Figure 1. Similarly, orchestrating a TV, a DVD player, speakers, lights and curtains would implement Theater services, where a user can watch movies in a theater-like atmosphere through a single point of operations. Thus, integration of appliances realizes various value-added services [15][16]. We call such services integrated HNS services.

Multiple implementation platforms are currently being proposed to develop such HNS applications [10], such as DLNA [3], HAVi [7], ECHONET [4], Jini [11], OSGi [18], and X-10 [22]. Developers need to choose the optimal combination of HNS implementation standards for each HNS application. Such complexity on heterogeneous platform erodes quality and productivity of application development. Especially, in integrated HNS service, service quality, relia-
bility, and maintainability are critical subjects, because this kinds of applications are closely related to a user’s everyday life.

Software quality, such as portability, maintainability and reliability are usually guaranteed only in the specific platform, since HNS applications are usually developed based on a implementation platform. As a result, several problems, such as quality deterioration and increase of development cost, may be caused by the complexity of multiplatform environment.

Therefore, in the domain of HNS applications, the mechanism, which does not depend on a specific platform, of verifying the service quality and reliability, is important. Development framework which supports various implementation protocol and generation of actual implementation artifacts will greatly improve the quality and productivity of the created HNS application.

In this article, we propose MDD(Model-Driven Development) framework to support verifying platform-independent model of integrated services with SMV(Symbolic Model Verifier), and to generate actual implementation artifacts semi-automatically. Our MDD framework adopts an application architecture [9] which can integrate multiple HNS standards together. This architecture provides abstraction and encapsulation of appliance features in service component layer. The exhibited features can be accessed with common interface from external. The combination of the features is published as an integrated service.

In our MDD framework, we apply rule-based transformation to platform-independent model to appliance components and integrated services. In an appliance component, since a skeleton independent of device/middleware is directly generated, developers implement only device/middleware-specific parts. In an integrated service, since the whole of implementation artifacts are generated automatically. It becomes possible to develop high-quality integrated HNS services at low-cost by combining verification process and code generation process.

We actually implemented a HVAC service with our MDD framework. System model which defines appliances and home environment in HNS, is transformed to a program skeleton of each appliance component for Java Web services. Service model which presents concrete scenarios for integrated services, is transformed to an integrated service implementation for Perl script. We show a part of generated source codes in Section 4.

In Section 2, we state about the application framework which can use several HNS standards, and integrated HNS service models which does not depend on any platform and can be verified and model checked by SMV. In Section 3, our MDD framework and transformation rules from service and system models to actual implementations are denoted. In Section 4 and Section 5, we introduce actual development about HVAC service. Service model and system model of HVAC is inputted to our MDD framework, and outputted actual implementation artifacts.

2 Integrated HNS Service

As shown in Section 1, there exists various middleware implementation standards for HNS application. In [9], we proposed a new HNS application architecture for integrated service illustrated in Figure 2. In this architecture, the service component layer aggregates the features of the appliances as a set of services, and exports the services to the network with exported methods. Such self-contained and abstracted appliance features can be accessed from external in a device-independent manner without considering platform complexity.

Moreover, in [12], we established platform-independent models verifiable with SMV(Symbolic Model Verifier [14][20]) for integrated HNS services. HNS applications need to be verified carefully because they are related to our daily-life (For instance, wrong invocation of appliance feature whose power supply should be turned on beforehand). Such wrong integrated service in HNS may exert danger on a user. So it is important to verify integrated HNS service.

In the following Section 2.1, we state about some important definition to express integrated HNS service model. The full specification is found in [12]

2.1 Model of Integrated HNS Services

Our integrated HNS service model consists of a system model and a service model. Figure 3 is an example of the system model which denotes a part of constitution of the
HVAC service stated in Section 1. Like this, the system model defines HNS constitution of arranged appliances and home environment.

The TYPEDEF section declares types commonly used in the system. The proposed language supports three types: Boolean (i.e., \{true, false\}), integer, or enumeration. An integer type is specified by the range between upper and lower bounds, e.g., \{18..28\}. An enumeration type is defined by enumerating concrete elements, e.g., \{ON, OFF\}.

The ENVIRONMENT section defines an environment object. In our HNS model, the environment consists of only environment properties. In Figure 3, tTemp Temp.In express the variable for temperature inside the home and its type.

All appliances deployed in the HNS are declared in multiple APPLIANCE sections (blocks), each of which defines an appliance object. An APPLIANCE block comprises of definitions of properties and methods of the appliance. The appliance properties are specified in the same way as in the ENVIRONMENT section. Each method is described in a METHOD subsection.

In the case of setTemperature method of AirConditioner, return value is void and argument is temp (its type is tAC.Temp). Each method has PRE, POST, ENV.R, ENV.W, RETURN as its attributes. setTemperature method requires that power property value is ‘ON’ before its execution. This attribute is defined as PRB (pre condition). POST (post condition) describes property name and its value which changes after method execution. Namely, invocation of setTemperature changes value of tempSetting to value of temp (argument of setTemperature). ENV.R shows environment property monitored by a sensor device (in the case of Thermometer_inside, measureTemp method monitors Temp.In in environment property). On the contrary, ENV.W shows environment property affected by appliance method. The method of setTemperature affects Temp.In. RETURN specifies the value to be returned, which can be specified by a property or an expression (e.g., currentTemp (This property is defined as an appliance property)).

Figure 4 illustrates a service model example of the HVAC service. In this example, HVAC service has an argument userTemp (the type is tAC.Temp). In VAR section and APPLIANCE section, variables and appliances used in the service are declared. In CONTENT section, actual integrated service statements are shown. Basically, the statements are sequentially executed one-by-one from top to bottom, as in the ordinary procedural programming language.

end() and exit() are pseudo functions. end() returns true (1) when the user terminates the service (e.g., with a termination signal from the user agent). exit() models a system call by which the system terminates the service. These allow the developer to specify explicitly when or by whom the service is terminated.

These system and service model can be translating into the well-known SMV language. Once translated, the SMV tool automatically and exhaustively verifies the integrated service against any properties specified in CTL (Computational Tree Logic). Thus, we can effectively detect design flaws in integrated services.
2.2 Application Development of Integrated HNS Services

As we stated in the beginning of this section, we proposed an application architecture and models. Our application architecture make it possible to use together the appliances implemented by various standards. Our models realize integrated service verification at design level. However, there is no relationship between models and implementation in integrated HNS services. So, it is not guaranteed that a HNS application is exactly developed according to the models.

In order to implement the artifacts with a validated model, it is important to support application development by a development framework such as code generation. Moreover, it is expected that this kind of support is effective in the field of the appliance component in which the feature of the same specification is implemented by multiple appliances in many cases. For example, the method of ON/OFF is implemented by various kinds of appliance. Such common feature specifications are often used by same kinds of appliance.

Implementation of integrated services needs to be updated corresponding to variation of appliances and methods. So, the more complicated service scenarios become, the more deteriorated their quality, such as maintainability and reusability, is.

To improve these problems, we adopt MDD technologies for integrated HNS service development.

3 Model-Driven Development of Integrated HNS Services

Model-Driven Development (MDD) is a promising approach to address platform complexity and express domain concepts effectively [19]. MDD combines the following two technologies:

(1) DSML (Domain Specific Modeling Languages): whose type systems formalize the application structure, behavior, and requirements within particular domains.

(2) Transformation Engines and Generators: analyze certain aspects of models and then synthesize various types of artifacts, such as service interface, simulation inputs etc.

MDA [2], Software Factories [6], and MCSD [21] are representative MDD technologies which generate implementation artifacts from models.

In this article, we propose MDD framework for integrated HNS services.

3.1 Key Idea

Figure 5 illustrates a process model of the proposed MDD (Model-Driven Development) framework for integrated HNS services. The system model defines home environment and exhibited features of appliance in HNS. Service model shows the scenarios of integrated HNS services. These models are input to the MDD framework. Our framework consists of integrated service verification process and integrated service composition process. In verification process, SMV program generator transforms a service model and a system model to SMV program and SMV checks its program [12]. If behavioral anomalies of the models are discovered by model checking, the developer corrects the bugs and re-verifies them with SMV, and finally gets the validated service model.

Next process is an integrated service composition process. In the composition process, service component generator generates service components from the system model, integrated service generator generates integrate service implementations from the validated service model and the system model.

We use Java Web services for implementation of service components, and Perl script for composing services. In the following section we define transformation rules for integrated service composition process. As shown in Section 2.1, system model and service model are platform-independent and are defined as a set of arbitrary constructs. Implementation artifacts (called ARTIFACTS) such as service components and integrated services, are
generated from this PIM (Platform Independent Model). Moreover, if ARTIFACTS is a set of arbitrary constructs in the implementation artifacts, the transformation to ARTIFACTS from PIM is defined as a map \( M : PIM \rightarrow ARTIFACTS \). Henceforth, each transformation rule which constitutes the map \( M \) is explained.

In Section 3.2, transformation rules to service components (Java Web services) from the system model are shown. In Section 3.3, transformation rules to integrated service (Perl script) from the validated service model and the system model are shown.

3.2 Transformation Rules from System Model to Service Component

Service component generator generates service component implementation with Java Web services, from system model. Table 1 shows transformation rules \( T \) for type definition of property used in HNS. Type transformation rules \( T \) transform three type definitions: Boolean, int, enumeration to three type definition in int. Boolean is transformed to \{true=1, false=0\}. In enumeration type, its enumerating concrete elements are transformed to ordered integer value from zero (for example, \{COOLING, FAN\} is \{0, 1\}).

Figure 6 and Figure 7 are transformation rules for property and method of appliances, respectively. In a system model, an appliance consists of a set of properties and a set of methods. A set of properties are transformed to appliance status Java bean, and a set of methods are transformed to the methods of the appliance class to invoke the corresponding status Java bean. The status java bean have prop-
property declaration and initialization denoted by the section of APPLIANCE and PROPERTY in the system model. And Getter/Setter methods for accessing the properties are declared.

In the appliance class, instantiation of the corresponding status Java bean and the method declaration defined by the METHOD in the APPLIANCE section are denoted. Argument and return type and value of the method are written, and PRE/POST attributes are inserted in the method as a set of pre-condition and post-condition of assertion. As you see in the Figure 7, the appliance class is transformed to the skeleton of appliance component from the models. Developer can implement the appliance component only coding device-dependent part in the method.

3.3 Transformation Rules from Service Model to Integrated Service

Integrated Service Generator generates integrated service implementation in Perl script from the validated service model and the system model. In Figure 8, instantiation part of the integrated service implementation is shown. In the Perl script for Web services integration requires to include SOAP::Lite module and instantiation of WSDL for each service component. Then, arguments of the integrated service (formal_param), and procedure for pseudo function end() are declared. These declarations express the section of VAR in the SERVICE and APPLIANCE section. CONTENT section is transformed by statement transformation rules S (in Table 2).

4 Case Study

As a case study, we developed HVAC service. The detailed HVAC service is as follows.

HVAC Service : HVAC serves energy-saving air-conditioning service to keep a room at the set temperature. For the simplicity the discussion here, we focus on its cooling function. If the room is warmer than the set temperature, the HVAC service turns the air-conditioner to the cooling mode. To efficiently cool down the room, if the room temperature is warmer than the outside, the ventilator is also turned on to provide fresh outside air. In this case the ventilator will keep working until the room temperature reaches the outside temperature. If the room temperature is below the set temperature, on the other hand, HVAC turns the air-conditioner to the fan mode. This service requires four appliances: AirConditioner, Thermometer_inside, Thermometer_outside, and Ventilation.

A part of system model is shown in Figure 3. In Figure 4, service model of HVAC service is defined. Device itself is realized virtually as software functions.

Development environments are as follows.

- Apache Tomcat 5.5.17
- JDK 5.0 Update 7
- Apache Axis 1.4
- Active perl 5.8.8 (with SOAP::Lite and strict module)
import java.io.*;
public class AirConditioner{
    private AirConditionerStatus status = new AirConditionerStatus();
    public void ON() throws Exception{
        /*DEVICE_METHOD_INVOCATION*/
        assert status.getPower()==0 : status.getPower();
    }
    public void OFF() throws Exception{
        /*DEVICE_METHOD_INVOCATION*/
        assert status.getPower()==1 : status.getPower();
    }
    public void setTemperature(int temp) throws Exception{
        assert status.getPower()==0 : status.getPower();
        /*DEVICE_METHOD_INVOCATION*/
        assert status.getTempSetting()==temp :
    }
    public void setMode(int mode) throws Exception{
        assert status.getPower()==0 : status.getPower();
        /*DEVICE_METHOD_INVOCATION*/
        assert status.getModeSetting()==mode :
    }
}

Figure 9. AirConditionerClass

Figure 9 and Figure 10 are AirConditioner class and AirConditionerStatus Java bean generated by the service component generator, respectively. Interface of the AirConditioner class is exhibited as Web services. In this development, we coded only virtual appliance procedure in the generated program skeleton. Table 3 shows LOC of developed each artifact. LOC of generated code means size of the code generated automatically. LOC of added code means size of the code added by us. Moreover, the whole of integrated service implementation (HVAC.pl) for HVAC service was generated automatically. We confirmed that these implementation artifacts perform HVAC service.

5 Discussion

As shown in Section 4, the proposed MDD framework realizes the following processes.

1. Platform-independent service and system model are verified to create validated service model by SMV.

2. Appliance component skeleton of Java Web service is generated from the system class.

3. The whole of integrated service implementation (Perl script) is generated from the validated service model and the system model.

Verification process and composition process enables developers to make more reliable and safe integrated HNS services. Program skeleton of the appliance component realizes separation of exhibited interface to network and device/middleware-dependent procedure. So, developers code only device/middleware-dependent parts. Moreover, in the program skeleton, assertion is inserted for evaluation of \textit{PRE/POST} attribute in the exhibited method. These assertion statements enable the developers to test the appliance component program [5]. As a result, these development support capabilities in our MDD framework are expected enhance productivity and quality of integrated HNS services in various development environment such as multi-platform.

In conventional research, several MDDs for Web services are proposed [1], [17]. These researches realizes transformation from standardized modeling language such UML or proprietary models to Web services implementation. However, these researches lack separation of concerns between exhibited interfaces of appliance and device/middleware-dependent procedure, and verification the services.

In the domain of verification in HNS application, DLNA [3] uses CTT (Conformance Test Tool) for networked appli-
ance) to verify fixed integrated services thoroughly. However, new flexible services cannot be verified and developers are not supported.

6 Conclusion

This paper presents a MDD framework for integrated HNS services. In the domain of HNS application, safety, reliability, portability and maintainability are important factors, because HNS applications are closely involving our daily life. So our framework enables developers to verify the services and generate implementation artifacts. Generated artifacts use our proposed application architecture which can use multiple HNS platform together, based on Web service. As a result, developers can create reliable and safe integrated services at low-cost.

We are planning to create more various generation rules corresponding to more practical HNS applications creation.

Acknowledgments

This research was partially supported by Pache Research Subsidy I-A-2,2006 of Nanzan University.

References


[22] X-10, http://www.x10pro.com/