Using MDE to Build a Schizophrenic Middleware for Home/Building Automation

Grégory Nain, Erwan Daubert, Olivier Barais, Jean-Marc Jézéquel

IRISA/INRIA/university of Rennes1, Equipe Triskell, F-35042 Rennes Cedex

Abstract. In the personal or corporate spheres, the home/office of tomorrow is soon to be the home/office of today, with a plethora of networked devices embedded in appliances, such as mobile phones, televisions, thermostats, and lamps, making it possible to automate and remotely control many basic household functions with a high degree of accuracy. In this domain, technological standardization is still in its infancy, or remains fragmented. The different functionalities of the various appliances, as well as market factors, imply that the devices that control them communicate via a multitude of different protocols (KNX, LonWorks, InOne). Building a high level middleware to support all the appliances seems to be a reasonable approach. However, market factors has shown that the emergence of a unique and universal middleware is a dream. To solve this issue, we have built a new generation of schizophrenic middleware in which service access can be generated from an abstract services description. EnTiMid, our implementation of schizophrenic middleware, supports various services access models (several personalities): SOAP (Simple Object Access Protocol), UPnP and DPWS (Device Profile for WebServices). In this paper, we describe how these personalities are generated using a Model Driven Engineering approach and discuss the benefits of our approach in the context of a deployment of new services at the city level.

1 Introduction

Time after time, each building parts manufacturer has developed his own communication protocol, and this for two reasons. The first one is the increasing need of communication between the devices. Then, the belief that a close protocol is more secured, is still present in minds and make the second reason. As a consequence, devices of today are communicating through dozens of protocols, and most of them are private and protected. For example, X2D[1], InOne[2] or IO-homecontrol[3] are private protocols. Open protocols are emerging such as KNX, LonWorks or BacNet, but interconnections between each other and/or with private protocols are often made 'on demand'. Building a high level middleware to support all the appliances and allow the development of high level services seems to be a reasonable approach. However, building automation market factors has shown that the emergence of a unique and universal middleware is a dream. To solve this issue, we present in this paper a new generation of schizophrenic middleware [15] in which service access can be generated from an abstract services description. EnTiMid,

[1] Dela-Dore protocol (http://www.deltadore.com)
[2] Legrand protocol (http://www.legrand.fr)
[3] IO-homecontrol consortium protocol (http://www.io-homecontrol.com)

our schizophrenic middleware implementation, supports various services access models (several personalities): SOAP (Simple Object Access Protocol), UPnP [12] and DPWS (Device Profile for WebServices) [6]. In this paper, we describe how these personalities are generated using an Model Driven Engineering approach and discuss the benefits of our approach in the context of a deployment of new services at the city level.

The remainder of the paper is organized as follows. Section 2 presents an overview of EnTiMid, a middleware for home automation and presents the meta-model embedded into EnTiMid to export devices services at the business level. Section 3 presents the generative tool chain of personalities. Section 4 discusses the usage of EnTiMid in the context of a city deployment. Section 5 highlights some selected related works and Section 6 wraps up with conclusions and outlines some future work.

2 Overview of EnTiMid

EnTiMid is a middleware implementation developed in a house or building automation context. The aim of this middleware, is to offer a level-sufficient abstraction, making it possible for high level services to interact with physical devices (such as lamps, heater or temperature sensors).

2.1 A layered middleware based on services

Based on a service-oriented architecture [5], this middleware incites people to build their software as a set of services working together. Thus, each user can customize the services offered by the software, according to his needs.

The OSGi Alliance[9], 'consortium of technology innovators', has released a set of specifications which define a service-oriented platform[1], and its common services. The OSGi kernel is a standard container-provider to built service-oriented software. It implements a cooperative model where applications can dynamically discover and use services provided by other applications running inside the same kernel. It provides a continuous computing environment. Applications can be installed, started, stopped, updated and uninstalled without a system restart. It offers a remote management model for applications that can operate unattended or under control of a platform operator. Finally it embeds an extensive security model so that applications can run in a shielded environment. According to these specifications, an application is then divided on several bundles. A bundle is a library component in OSGi terms. It packages services that are logically related. It imports and exports Java packages and offers or requires services. Services are implementations of Java interfaces.

Inspired by the CBSE (Component Based Software Engineering) paradigm [11], each bundle is designed to reach the highest level of independence, giving the software enough modularity to allow partial services updates, adds or removes. This programming style allows software-builders, to deploy the same pieces of software for all of their clients, either professionals or private individuals, and then simply adapt the services installed. Moreover, the services running on the system can be changed during execution.

On top of this kernel, EnTiMid is structured around three layers as presented in Figure 1: a low-level layer which manages communication with the physical devices, a middle layer (the kernel), offering a message oriented middleware (MOM), and giving a first abstraction level of low-level layer, a high-level layer which publishes services and enables the device access through standard protocol. The Figure 1 gives an overview on how services are organized. EnTiMid defines a set of interfaces between the services provided by the low level layer and the services required by the middle layer to allow the access to the physical devices.

The low-level offers a common abstraction to the EnTiMid Kernel to access the different devices. It wraps existing library to support some protocols, for example, it wraps the Calimero toolkit to provide an access to KNX devices [4]. It provides also home-made drivers to access protocol such as X2D or X10. The EnTiMid kernel (middle layer) and the high level layer are described more precisely in Sections 2.3 and 3.

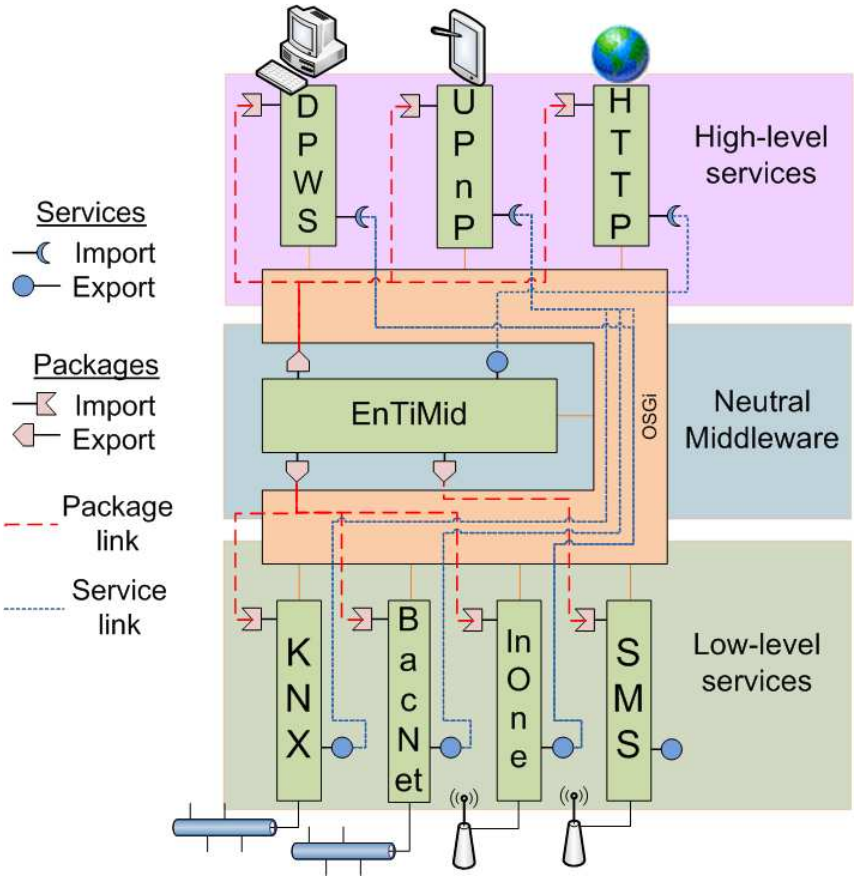

Fig. 1. EnTiMid architecture

[4] calimero.sourceforge.net

2.2 A schizophrenic middleware

High-level protocols, such as DPWS, SOAP or UPnP, are going to be more and more present in our everyday life. Each of them offers a different access to devices, according to their main goal. For example, UPnP has been developed to ease media sharing, whereas SOAP is a protocol to programmatically access services such as devices provided services. So, manufacturers choosing to implement a high-level access to their devices, will select the protocol offering the best accordance with the devices applications.

In a few years, new high and low level protocols will probably appear, and some will become useless. Existing protocol will evolve and EnTiMid, which aims to ease the device interconnection, has to support all the new protocols and new protocols versions. An interesting solution to face the need of protocol management flexibility (both low and high level), is the schizophrenic middleware, presented by Laurent Pautet[15]. A schizophrenic middleware offers several personalities to access services. Consequently, in EnTiMid, high-level protocols define application personalities. As the middleware has to support lots of application personalities, we propose in this paper to use a MDE approach to generate these application personalities from an internal representation of devices. This choice is driven by the features provided by a MDE approach (abstraction, transformation language) and by the improvement of the maturity of MDE tools (we use the Eclipse Modelling Framework [3] for the meta-model definition and Kermeta [8] for the transformation). Next subsection presents in depth the internal representation of the device services. Section 3 presents the details of the generative approach used for the UPnP and the DPWS personalities.

2.3 EnTiMid kernel

Figure 2 shows a part of the structure of the middle-level layer. Each manufacturer provides a *Bundle*. This bundle will use a *Gateway* contained in a *Zone* in order to access the devices. Each low-level bundle implements a method called *getAvailableProducts()*. This method returns a catalogue of the devices the bundle is able to control/provide. Then, users just have to instantiate the device they need to interact with. According to the type of the device, the actions offered are different. *Actuators* inform the system about the actions they are able to realise, giving a list of *CommonAction* (on, off, up, down...). Then, for each *CommonAction*, they produce a *HouseAction* containing all necessary information for the action to be done on a device. These *HouseActions* can be valued to specify light intensity for example.
Sensors are divided in different categories, and we focus on *ModularSensors*. They are composed of *Modules*, and those modules are only containers. At setup time, a sensor in charge to switch off a light, will ask the light actuator for the *HouseAction* to be sent to switch off the light, and this *HouseAction* is then stored in a list. Then, at use time, when the *push()* method is called on a *Module* of a *ModularSensor*, all the *HouseActions* stored in the *Module* list are published to all *HouseActionListeners*.

Figure 3 illustrates what append when push method is called on a *Module*. A *HouseEvent* containing a *HouseAction* is sent for each *HouseAction* to each *HouseEventListener*. Then, the bundle able to route the information on the destination

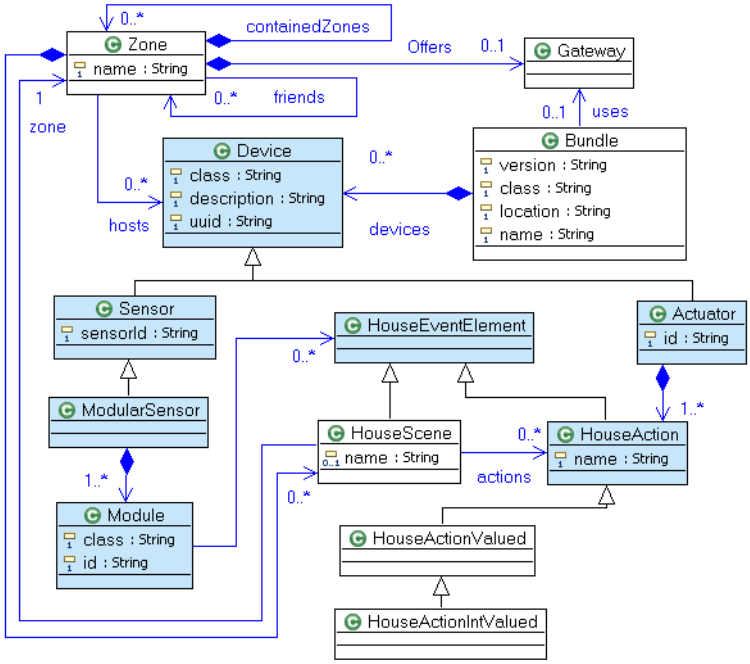

Fig. 2. Simple EnTiMid model

network (here the LonWorks network) will translate and send the message to the concerned device.

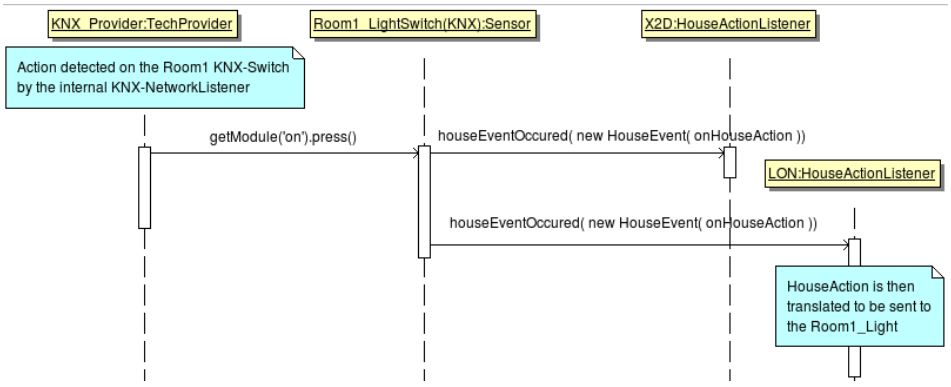

Fig. 3. Message transmission on action detection

3 A MDE approach to generate middleware personality

3.1 From PIM to PSM

The Object Management Group (OMG) promotes an approach to achieve adaptability at the application level with the Model-Driven Architecture (MDA) [10,4]. The concepts involved in MDA are based on the definition of two kinds of model: Platform Independent Models (PIM) and Platform Specific Models (PSM). A PIM provides an abstract representation of an application independently of any standards and aims at capturing only the business concerns. A PSM is the result of the transformation of a PIM to apply it on a particular technology or a particular standard. PIM and PSM are defined by specific meta-models which constrain the expression of the models. The EnTiMid internal device services model can be considered as a PIM, due to the fact that all devices types are specified in the framework, and specifications are common to all technologies. On the opposite, UPnP or DPWS personalities, for example, are to be considered as PSMs, because the device description mode is very specific for each device and dependent of the protocol.

3.2 UPnP personality generation

UPnP Meta-model UPnP [12] is based on a discovery-search mechanism. As an UPnP-Device joins the UPnP network, it sends an XML description file to all UPnP-ControlPoints, presenting itself with information such as manufacturer, device type, device model or uuid.
Most of times a UPnP-Device is self-contained, describing itself and publishing all services it can offer. The UPnP specification allows devices to contain other devices (called embedded devices); if so, the *rootDevice* (the container) have to publish its self-description and the description of each embedded device.

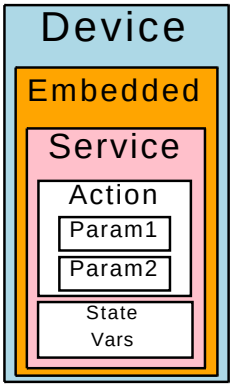

Fig. 4. UPnP-Device structure

Moreover, UPnP-Devices (embedded or not) can offer UPnP-Services, as shown in Figure 4. Each service type provided by an UPnP-Device, must be described in a separated file. The description explicits all the UPnP-Actions the service renders, and all the UPnP-StateVariables, used by these actions, in a UPnP-StateTable. Indeed, UPnP-Actions can be parametered, and sens (in or out), name and related StateVar are specified for each parameter. UPnP-StateVars are used to offer more precise information about parameters, such as value type or allowed value list.

Meta-binding he Figure 5 shows how the binding between EnTiMid-Devices and UPnP-Devices is done. An EnTiMid-Device is exported as a UPnP-RootDevice, beeing Sensor or Actuator.
In the case the device is a Sensor, and more precisely, a ModularSensor, all the modules it contains are exported as UPnP-EmbeddedDevices. The RootDevice do

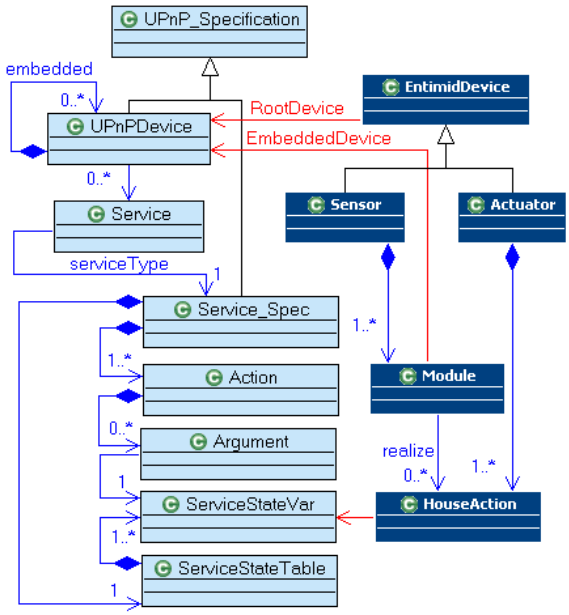

Fig. 5. UPnP mapping example

not offer any services: only modules are offering such a thing. But, actuators do not contain any Module, and so, can not be exported as simply as sensors.
In order to allow users to generate actions on actuators, a new UPnP-Device is created and is given as *BasicModule* as it offers *CommonActions*. For example, if a simple light actuator offers "on" and "off" *CommonAction*, a *BasicModule* will be created for each action, and added to the RootDevice. By this way, it will be presented as a UPnP-Device containing two Modules (one "on" ans one "off") each one offering a *push()* method.
As a consequence, EnTiMid-Devices are mapped to UPnP-RootDevices; the Modules are mapped to UPnP-EmbeddedDevices and *CommonActions* are specified as *ServiceStateVariables*.

Files As previously said, each UPnP-RootDevice and each UPnP-Service must be defined in an XML description file. During its life, the UPnP-Device will frequently be asked to send its description file to other UPnP-Devices, and services descriptions are consulted each time a service is likely to be used.
In a commercial point of view, those description files are defined once (for a proper device), and embedded in the product. But the dynamics of EnTiMid, its abstraction level and its modularity, implies that information, such as device type or description, about the devices installed on the system, are never known in advance. Moreover, the EnTiMid implementation of a product can offer different services, and those services are implementation dependent. That is why the description files are generated at runtime.

Service description file generation Our choices of implementation has led to the fact that only Modules offer services. UPnP-Actions, offered by a UPnP-

Service, are an UPnP-Export of some methods of the module class. Methods to be exported are signaled to be UPnP-Compliant, thanks to the presence of an UPnP-Method annotation. This annotation offers some interesting information. The first information is that the annotated method has to be exported in the service; then, the annotation gives, at runtime, a semantic name to the parameters and the returned value of an action. More precisely, without annotation, the only information one can get on a parameter at runtime is its place on the method signature and its type. In this case, the user can not have the information that the first string parameter is for the name, and the second for the age. The annotation brings these information.
So, if the BasicModule service description file does not exist yet, a simple reflexive research on the class's methods make it possible for the system to generate it, and all devices offering this service will then be able to point to this file.

Device description file generation In order to generate the description file of a device only once, the name of file is the device class name. Device des s. The first one gives general information about the device: manufacturer, description, model type, model number or uuid; some are statically completed by the EnTiMid-System (such as manufacturer), others are retrieved from the device itself (model type, model number). The second part contains the types, identifiers and links of the services offered by the device. For example, the first BasicModule declared will complete its services list with:

```
<serviceType>urn:www.entimid.org:service:BasicModule:1</serviceType>
<serviceId>urn:www.entimid.org:serviceId:BasicModule1</serviceId>
<SCPDURL>/service/BasicModule/description.xml</SCPDURL>
<controlURL>/service/BasicModule/control</controlURL>
<eventSubURL>/service/BasicModule/eventSub</eventSubURL>
```

The last description file part contains the description of each embedded devices. Those descriptions are composed of the two previous part, for each embedded device.

UPnP events management A different listener is created for each service of the device, to simply manage UPnP events. By this way, a listener is linked with a unique module, and side-effects with other modules or devices are avoided. Each listener is then attached to each action of the service.
When an action event is received, the first work is to identify the method that has been actionned. Once done, the second work is to cast the UPnP-Action parameters into the real method argument types. Then the method is invoked, and, if necessary, the result is translated into a UPnP-VarType and sent.

3.3 DPWS personality generation

The Device Profile for Web Services (DPWS) [6] defines a minimal set of implementation constraints to enable secure Web Service messaging, discovery, description, and eventing on resource-constrained devices. Its objectives are similar to those of Universal Plug and Play (UPnP) but, in addition, DPWS is fully aligned

with Web Services technology. Moreover, it includes numerous extension points, allowing for seamless integration of device-provided services in enterprise-wide application scenarios. From a conceptual point of view, the DPWS meta-model is closed to the UPnP meta-model described in Figure 4. Consequently, building the abstract model of the service to export, follows the same way: we use the Java annotation in the low level layer to infer the model. However, the generation process is different. To build the DPWS layer, we use the WS4D project [16]. This project proposes a programming model to create DPWS services. This model is based on the concept of *service*, *device*, *operation* and *parameter*.

A DPWS *service* provides an implementation of one or more WS (Web Services) port types to DPWS clients. The messages, a service receives and sends, are specified by its WS port types. The DPWS services definition is different of the standard definition of the term *service* in the WSDL specification. A *device* hosts one or more services and provides common functions like messaging or discovery. It is classified by its port types. According to the DPWS specification a device is a target service of the WS-Discovery specification. The basic functionality of a device is implemented by the class HostingService. An *Operation/Action* is a named message exchange pattern of a port type. It can consist of an input and output message, and multiple fault messages. The appearance and order of the input and output messages in the WSDL determine the type of the operation (request-response (in-out), solicit-response (out-in), notification (out), one-way (in)).

For each device, service and operation, a Java class has to be generated. This class must extend respectively HostedService, HostingService and Action. For each parameter, an instance of the class Parameter has to be implemented. Consequently, the generation process can be done automatically. To achieve that we use the JET Framework to create generation template for DPWS. We embed the JDT compiler provided by the eclipse project to compile the generated code. Finally, we programmatically create a new bundle containing all generated classes. Once loaded, this new bundle provides all generated classes, and allow them to be used.

4 Use case

4.1 Context: Application to a city-level project

EnTiMid is currently used in a Brittany project to allow old persons to hold in their home as long as possible. Two associations of the Rennes metropolis, the "*CODESPAR*" and the "*ASSAD*", are working together in a project called "*Maintiens à domicile et habitat évolutif*". With the support of industrial partners, they are conceiving an environment around health professionals and old people, composed of new information technologies. As previously said, the main goal of the project is to help people to stay at home as long as possible, but this can not be done without helping health professionals in their everyday work.

From March to October 2007, an initial study has permitted to obtain a set of recommendations. The second phase of the project aims to find technical answers to these recommendations. However, technical solutions are often multiples, and the probability to install this technical environment over, or mixed with, an already installed technologies is not null.

Consequently, the software used to manage the technologies and ease the access to the house for health professionals, will have to be fully adaptable to the in-place technology, and require a short development time to reach new technologies or new protocol versions.

Its unified technology management, provided by the middleware abstraction of the underlying protocols, and its multiple access personalities, inherited from its schizophrenic aspect, have led EnTiMid to be a privileged candidate to be deployed as the access point to the equipped houses.

4.2 Advantages of a schizophrenic middleware in this context

The schizophrenia of this middleware and its generative capabilities are advantages in two dimensions.

In space The city scale deployment of the project necessarily implies that the technologies used will sometimes be different, due to some physical constraints, or because a technology is already installed, and people do not want to change. EnTiMid will then propose an abstraction of the deployed devices technologies; it will expose different personalities of these devices for high level application developers. Consequently, services provider associated to the project will be able to develop high level services directly on top of DPWS or UPnP. Finally, the management capabilities provided by the OSGI gateway will also help to update and reconfigure the gateway.

In time Software, technologies and protocols constantly evolves and versions change with, sometimes, some compatibilities problems. That is to say that during its life, the OSGi gateway will have to implement new protocols, or different versions of a given protocol. Moreover, protocols can be used in different versions, at a given time, in different places of the city. Once again, the different personalities make it possible for EnTiMid to gain multiple version compliance, for different protocols.

5 Related work

Xie Li [7] has developed a residential service gateway, which aim is to rely "inside-of-house" system to a "outside-of-house" system, allowing users to connect their residential gateway from a centralized point "outside-of-house". The connection is recommended to be done by a VPN solution, and offers an HTTP interface to control devices through the Lonworks PLC technology and they have planned in future work to export their services to UPnP. The "inside-of-house" system, developed on an OSGi platform, implements algorithm giving Plug&Play facilities to the system for "pre-defined devices". Compared to this system, our implementation is designed to be Plug&Play. Our system also eases the interoperability and can access several technologies.

The paper [13] presents a "Service Oriented Pervasive Applications Based On Interoperable Middleware". As EnTiMid, the described middleware is composed of three layers: a drivers layer, in charge of the connexion between the devices and the "Unified Service" layer. Then a bridges layer links the Unified Service instances to diverse "service technologies (UPnP, WS,...)". The solution is similar, but they do not use the OSGi technology. Besides, we made the choice in this paper to use

an MDE based generative approach to propose several personalities for the diverse "service technologies (UPnP, WS,...)".

Valtchev *et al.*[14] have developed a gateway to control a smart house. This gateway defines an abstract layer to manage the hardware protocol used to communicate with physical devices. Moreover it is defined to manage many services gateways. But it does not define a high level abstraction to offer services through protocols like DPWS or UPnP. This abstraction could be done using their gateway but for practical reason we have choosen to define our own implementation, because their implementation offers many things like we don't want to use now. Even if, later, EnTiMid could be bigger and need to manage many gateways for examples.

Bottaro *et al.* have developed a service platform[2] to offer service abstraction like DPWS over device communication. Into this platform, each device has to register on the OSGi context, for each high level protocol it implements. At runtime, the high level protocol 'manager' gets all registered devices and publishes them on the network. The main difference between this platform and EnTiMid comes from the service registration. Indeed, for each high level protocol a device want to offer, it has to implement a set of specific interfaces(API) and register to the OSGi context. The generative approach used in EnTiMid simplifies the development of devices. All installed devices are natively exported toward high level protocols, thanks to their EnTiMid-Device implementation.

6 Conclusions and perspectives

The plethora of networked devices embedded in appliances, such as mobile phones, televisions, thermostats, and lamps, makes possible to automate and remotely control many basic household functions with a high degree of accuracy. Consequently, a new breed of technologies is needed to address the challenges of the development and deployment of building automation applications over an evolving, large-scale distributed computing infrastructure. The approach and the tools, provided by EnTiMid, and described in this paper, are an example of such a technology.

EnTiMid offers a first solution to manage the multiplicity and the evolution of communication protocols through a layered schizophrenic middleware. This solution consists of offering a common abstraction of the home device topology and provides a generative approach to offer an access to the devices through different personalities. To improve the flexibility of this middleware, the high level protocols are generated and loaded at runtime. It enables a dynamic reconfiguration of the application and the high-level protocol bundle without any system restart.

EnTiMid have been implemented to form a complete middleware for home automation[5]. Future work includes technical improvement and new scientific investigations. As a technical improvement of the platform, the AMIGO European project designed a 3D application called VantagePoint, in order to model a room with objects and devices. Moreover, the JDT compiler embedded to compile the DPWS personalities is heavyweight for implementation in small commodity set-top box. The DPWS generation tool chain has to be technically improved. This could be a really good way to generate the configuration file of a house, or to provide a 3D device management application. As a scientific future work, we will work on

[5] http://house-manager.gforge.inria.fr

the definition of a context-aware service composition operator in order to provide users the relevant high level services. In this context, we will follow the work of the S-Cube project in particular the work on the adaptation of service-oriented applications.

Acknowledgments The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube). (http://www.s-cube-network.eu).

References

1. The OSGi Alliance. Osgi service platform core specification, release 4, avril 2007.
2. Stéphane Seyvoz André Bottaro, Eric Simon and Anne Gérodolle. Dynamic web services on a home service platform. In *22nd International Conference on Advanced Information Networking and Applications*, pages 378–385, mar 2008.
3. ECore. The eclipse modeling framework project home page. http://www.eclipse.org/emf.
4. Lidia Fuentes, Mónica Pinto, and Antonio Vallecillo. How mda can help designing component- and aspect-based applications. In *EDOC '03: Proceedings of the 7th International Conference on Enterprise Distributed Object Computing*, page 124, Washington, DC, USA, 2003. IEEE Computer Society.
5. F. Jammes and H. Smit. Service-oriented paradigms in industrial automation. *IEEE Trans. Industrial Informatics*, 1(1):62–70, 2005.
6. François Jammes, Antoine Mensch, and Harm Smit. Service-oriented device communications using the devices profile for web services. In *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–8, New York, NY, USA, 2005. ACM.
7. Xie Li and Wenjun Zhang. The design and implementation of home network system using osgi compliant middleware. *IEEE Transactions on Consumer Electronics*, 50, mai 2004.
8. Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In S. Kent L. Briand, editor, *Proceedings of MODELS/UML'2005*, volume to be published of *LNCS*, pages –, Montego Bay, Jamaica, octobre 2005. Springer.
9. Osgi alliance. http://www.osgi.org/About/HomePage.
10. R. Soley and the OMG Staff. Model-Driven Architecture. OMG Document, novembre 2000.
11. Clemens Szyperski. Component technology: what, where, and how? In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 684–693, Washington, DC, USA, 2003. IEEE Computer Society.
12. The UPnP Forum. http://www.upnp.org.
13. Aitor Uribarren, Jorge Parra, J.P.Uribe, Kepa Makibar, Ione Olalde, and Nati Herrasti. Service oriented pervasive applications based on interoperable middleware. In *Workshop on Requirements and Solutions for Pervasive Software Infrastructure (RSPSI2006)*, 2006.
14. Dimitar Valtchev and ProSyst Software AG Ivailo Frankov. Service gateway architecture for a smart home. *IEEE Communications Magazine*, 40:126–132, apr 2002.
15. Thomas Vergnaud, Jérôme Hugues, Laurent Pautet, and Fabrice Kordon. Polyorb: A schizophrenic middleware to build versatile reliable distributed applications. *LNCS*, 3063:106–119, 2004.
16. Elmar Zeeb, Andreas Bobek, Hendrik Bohn, Steffen Prueter, Andre Pohl, Heiko Krumm, Ingo Lück, Frank Golatowski, and Dirk Timmermann. Ws4d: Soa-toolkits making embedded systems ready for web services. In *3rd International Conference on Open Source Systems, Embedded Workshop on Open Source Software and Product Lines*, Limerick, Ireland, 2007.