

Toward an OSGi-based Infrastructure for Smart Home Applications

Szu-Chi Wang, Kai-Peng Chen, and Tung-


Yan Lin

View metadata, citation and similar papers at core.ac.uk

*Information Engineering,
National I-Lan University, Yilan, Taiwan
wsc@niu.edu.tw*

Chi-Yi Lin

Department of Computer Science and

brought to you by  CORE

*provided by Tamkang
Tamkang University, Taipei, Taiwan
chiyilin@mail.tku.edu.tw*

Abstract

In this paper we show the steps to implement the prototype of a smart home environment, with emphasis on the OSGi framework and first-order logic based inference engine. Due to the inherent extensibility of the underlying system, either the interfaces of sensors/home appliances or the inference rules can be adjusted and reloaded during runtime. Moreover, we illustrate several sensor-driven services that may help develop intriguing smart home applications in the near future. Related works and future directions are also addressed to complete our current implementation afterwards.

1. Introduction

The concept and practice of smart homes have greatly advanced recently. A smart home environment consists of collaborating ICT (Information and Communication Technology) facilities interconnected by the home network and provides us with a more convenient, autonomous, and secure living space. Inspired by [1], in this work we focus on exploring the issue of improving the living experience with the aid of various modern appliances. Also, we adopt Open Server Gateway Initiative (OSGi), the main-stream software platform for developing the smart home software architecture. OSGi is standardized by OSGi Alliance [2], with objective to propose an open service platform, i.e. OSGi Service Platform (OSGiSP), as a common and unified middleware upon which all smart-home devices could easily interoperate with each other.

Generally speaking, an intelligent system includes three building blocks: 1) Database; 2) Knowledge-base; and 3) Inference Engine. In this paper we present a prototype implementation of a smart home and discuss the future directions and challenges. In regard to a living space, the major contents stored in the Database

are generated by various in-house sensors. It should be noted that these constituents correspond to automatic/periodic sensor readings, and thus are vital for on-line data analysis and context-aware computing. In our implementation, since the inference engine is built from scratch, only rule-based inference in first-order logic has been employed. Nonetheless, we show that some reasoning useful in daily life can still be carried out. The rest of this paper is structured as follows. In Section 2 we introduce the background knowledge. Then our main results, as well as some discussion on practical issues and our future directions, are given in Section 3. Finally, Section 4 concludes this paper.

2. Preliminaries

2.1. UPnP

UPnP [3] is one of the most promising technologies to support easy-to-use, flexible, standards-based connectivity for building smart-home applications. It leverages the existing Internet protocols including IP, TCP, UDP, HTTP, SOAP [4], and XML, to enable seamless proximity networking as well as control and data transfer among networked devices. In 2007, the UPnP specifications have been approved as international standards under the ISO/IEC JTC 1, numbered DIS 29341 [5].

UPnP devices interact with each other via standard methods specified in the UPnP Device Architecture [6]. The interactions of UPnP devices are peer-to-peer; as a consequence, an UPnP network scales well and exhibits robustness. To support quality of service (QoS) in the UPnP environment, the UPnP Forum also defines the UPnP QoS Architecture [7]. In the following, we briefly describe the UPnP Device Architecture and the UPnP QoS Architecture.

2.1.1. UPnP Device Architecture. Two general classifications of devices are defined in the UPnP Device Architecture: *controlled devices* (or “*devices*” for short) and *control points*. Devices offer services requested by control points. Interactions between devices and control points are in six phases:

1. *Addressing*: Devices obtain an IP address from a DHCP server or use the Auto IP mechanism to get an address.
2. *Discovery*: Control points acquire the existence of devices by searching the network or through unsolicited advertisements sent by devices.
3. *Description*: Control points learn the detailed information about the embedded devices or services in a device.
4. *Control*: Control points invoke actions on the devices;
5. *Eventing*: Devices notify control points on changes of state by sending event messages;
6. *Presentation*: Devices present more information through URLs to allow users to control/view the device states.

Figure 1 shows the UPnP protocol stack, in which the SSDP (Simple Service Discovery Protocol) is a multicast discovery and search mechanism used in the discovery phase, and the GENA (General Event Notification Architecture) is an event subscription and notification protocol. Both SSDP and GENA are defined in [6]. SOAP, one of the W3C Recommendations, is the Simple Object Access Protocol on the basis of XML-based remote-procedure calls [4].

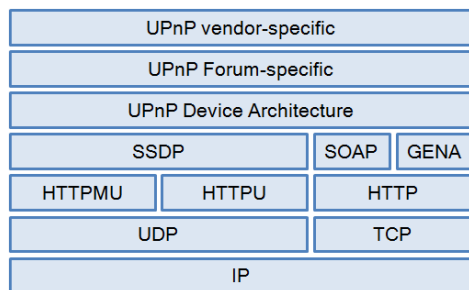


Figure 1. UPnP protocol stack

2.1.2. UPnP QoS Architecture. Figure 2 shows the UPnP QoS Architecture [7]. There are three types of services, namely *QosPolicyHolder* (QPH) services, *QosManager* (QM) services, and *QosDevice* (QD) services.

- The QPH service is a repository of QoS policies that specifies the treatment of traffic on the home network.

- The QM service discovers and controls QD and QPH services, in order to request, update, and release the QoS for various traffic streams.
- The QD service is responsible for managing the device’s network resources for traffic streams.

The QoS operations are described as follows. The control point constructs a *TrafficDescriptor* structure according to its knowledge of source, sink, content to be streamed, traffic specification (Tspec), and then appoints the QM to setup QoS for the traffic stream. The QM in turn requests the QPH to provide appropriate policy for the traffic stream described by the *TrafficDescriptor*. After a traffic policy is returned by the QPH, the QM then configures the involving QDs based on the policy. Upon successfully setting up the resources on all the involved QDs, the source device can start transmitting the traffic stream to the destination device. When the transmission completes, the control point instructs the QM to release the allocated resources in the involving QDs.

Regarding the Layer 2 priority of a specific traffic stream, the QPH can specify the relative importance of a traffic stream by assigning a *TrafficImportanceNumber* in the traffic policy, which is then used by the QD to derive the technology-specific Layer 2 priorities (e.g. IEEE 802.1p, HomePlug, HPNA, and DSCP).

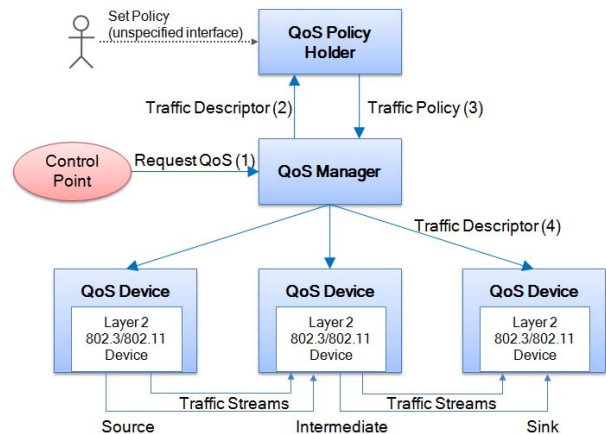


Figure 2. Overview of the UPnP QoS Architecture

2.2. OSGi

OSGi is standardized by the OSGi Alliance [1], whose mission is to create open specifications for the network delivery of managed services to local networks and devices. The OSGi Service Platform specification delivers an open, common architecture for service providers, developers, software vendors, gateway operators and equipment vendors to develop, deploy and manage services in a coordinated fashion

[8]. The targeted devices of the OSGi specifications include PCs, set-top boxes, service gateways, cable modems, mobile phones, consumer electronics, and many more. With devices that implement the OSGi specifications, service providers such as telcos, cable operators, and utilities are able to deliver smart home services to their customers.

The OSGi framework provides general-purpose, secure, and managed Java framework that supports deploying extensible and downloadable Java-based service applications known as *bundles*. An OSGi service platform is an instantiation of a Java virtual machine, an OSGi framework, and a set of bundles [9]. The OSGi framework manages the installation and update of bundles in an OSGi environment in a dynamic and scalable fashion by managing the dependencies between bundles and services in detail.

The functionality of the OSGi framework is divided into five layers (shown in Figure 3):

1. *Security Layer*: It defines a secure packaging format as well as the runtime interaction with the Java 2 security layer.
2. *Module Layer*: It defines a modularization model for Java, with strict rules for sharing Java packages between bundles or hiding packages from other bundles.
3. *Life Cycle Layer*: It provides an API to manage the bundles in the Module Layer. The API provides a runtime model for bundles and defines how bundles are started and stopped as well as how bundles are installed, updated and uninstalled.
4. *Service Layer*: It provides a dynamic, concise and consistent programming model for Java bundle developers, simplifying the development and deployment of service bundles by decoupling the service’s specification (Java interface) from its implementations.
5. *Actual Services*: The smart-home applications/ services.

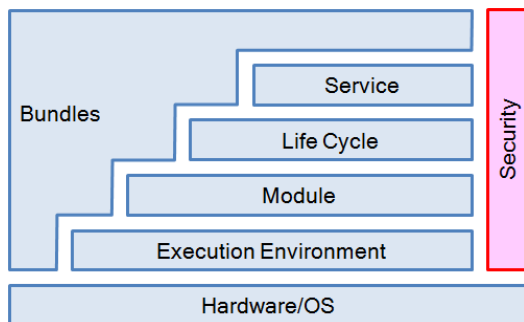


Figure 3. OSGi framework layering

The bundles are allowed to select an available implementation at run-time through the framework service registry, in which bundles register new services, receive notifications about the state of services, or look up existing services to adapt to the current capabilities of the device. The benefit of the framework design is that new bundles can be installed for added features or existing bundles can be modified and updated without requiring the system to be restarted.

The way that the layers interact with each other is shown in Figure 4.

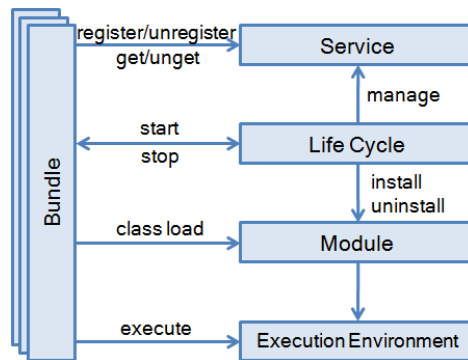


Figure 4. Interactions between layers

2.3. OSGi-to-UPnP Transformation

The UPnP base driver can add virtual UPnP devices to the UPnP network by exporting OSGi services. The system response and flexibility can be enhanced in this way. For example, provided that the internal control center is an UPnP control point and regulates various environment parameters through the OSGi gateway, we can construct an OSGi bundle for conciliation service that communicates with and integrates the involved external controllers, as a virtual UPnP device; such design may provide better coordination.

3. Main Results

3.1. System Description

First, we employ the OSGi platform as the basis of a service-oriented infrastructure. As mentioned above, such framework in a smart home environment supports dynamic load and update of services. Family members can login via the Internet. After authentications are confirmed by the residential service gateway, they can monitor and manipulate the services and bundles via a Web-based GUI. The sensors and appliances over the home network can also access the Internet through the residential service gateway. Figure 4 illustrates the major components of our system infrastructure.

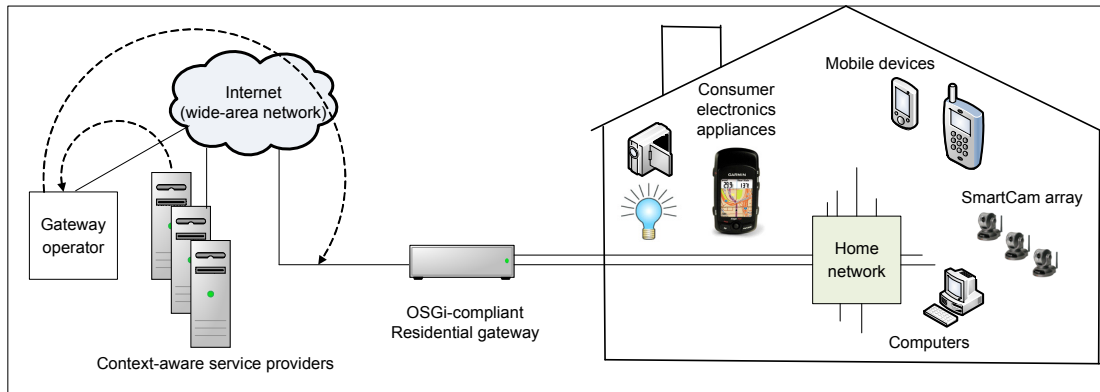


Figure 5. System overview

On the other hand, to make our living space more intelligent, we deploy various inter-networked sensors and appliances in the smart home environment. These equipments and the corresponding services should collaborate seamlessly to perform automatic tasks. The main sensors and home appliances in our current implementation include: 1) handheld PDAs equipped with GPS and 802.11b; 2) in-house temperature, sound level, and light sensors; 3) wireless smart cameras; 4) RFID tags and transponders. Since all sensors, in some sense, can capture the sub-set of environmental states, they should serve as context providers to augment the underlying computer system with context awareness. More specifically, these sensor datum can be further utilized to make the behavior of applications more “intriguingly” to the family members. See Section 3.2 for further details.

In regard to reasoning and decision making, in our implementation we simply consider inference in first-order logic. More specifically, we adopt Prolog, a declarative language that is widely used for logic programming, as the core of our inference engine. The inference rules are preloaded into the reasoning service. To increase the extensibility of our rulebase, however, the administrator can modify the inference rules and/or create new ones afterwards. By virtue of the properties of OSGi framework, these operations can be made over the Internet, and more importantly, in a dynamic manner without system rebooting.

We use MySQL to store both the various sensor datum and the user-defined rules. It is to be noted that the sensor samples are low-level, i.e. they are typically represented by streams of numeric values. Therefore, they should be transformed into the high-level “facts” for our Prolog-based inference engine. Moreover, one of our prime future works lies in developing an on-line algorithms that can synthesize higher-level knowledge (such as ontology-based context information) from these low-level sensor datum.

To somewhat verify the feasibility of our design, we consider the scenario as follows. As shown in Figure 6, a PDA-equipped family member’s identity, location, and trajectory can be (separately) recorded by the in-house database system. Additionally, the in-house control center can not only provide GPS-aided outdoor navigation but also deliver instant message in case some alarm has been derived by the rule-based inference engine. Note that due to lack of space, Figure 7 only provides a simplified version of alarm reasoning. Since these “facts” are automatically and continually generated from sensor datum, the reasoning process should be executed periodically to make sure that alarms are triggered based on the current environment.

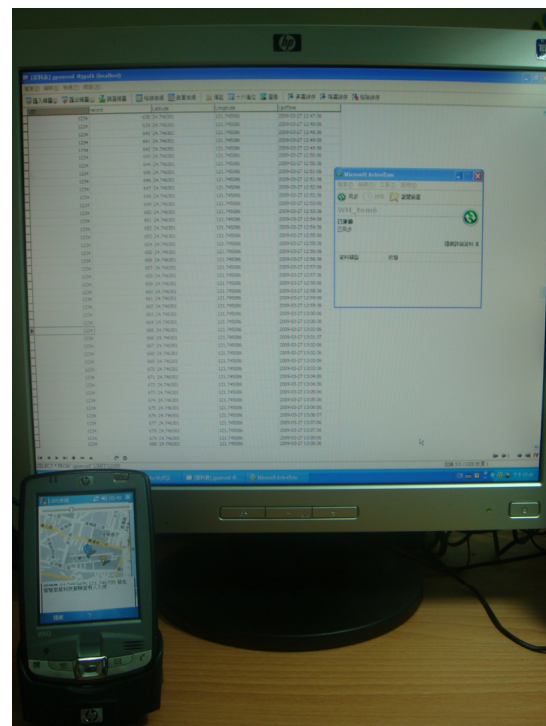


Figure 6. Illustrative example

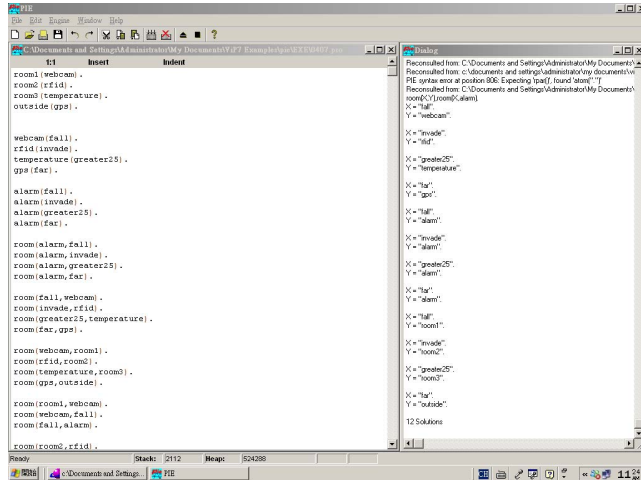


Figure 7. Inference in first-order logic

The update interval is usually application-specific and relates to sophisticated optimization techniques, hence the relevant issues are beyond the scope of this paper.

3.2. Discussion

As mentioned above, uncertainty roots in sensor outputs due to the noise in sensor inputs and the partial observations of events inherent in sensor networks. Moreover, we lack uncertain representation techniques and queries are carried out by the adopted database system via SQL expressions, which is not fit for uncertain data management [11][12]. Also in practice, queries evaluated with respect to specifications in a sensor network should return likelihood values instead of simply yes/no answers [13]. Therefore, it is crucial to tackle uncertainty associated with the stored sensor datum in a smart home environment.

A declarative programming language Snlog has been presented recently [14]; note that Snlog is the function-free subset of Prolog. As a consequence, the most important future directions include introducing this declarative programming language into our Database/Knowledge-base systems and developing the corresponding in-network evaluation services. We believe that putting all these cutting-edge techniques together can contribute to next-generation smart home technologies.

4. Conclusions

In this paper, we demonstrate how to implement a prototype smart home environment by virtue of the OSGi framework and simple rule-based inference in first-order logic. The extensible and flexible software

architecture of our system eases adjusting and reloading the interfaces of sensors/appliances and the rulebase during runtime. Moreover, we show several sensor-driven services that may bring about the future smart home applications.

References

- [1] T. Gu, H. K. Pung, and D. Q. Zhang, "Toward an OSGi-based infrastructure for context-aware applications," *IEEE Pervasive Computing*, Vol. 3(4), Oct.-Dec. 2004, pp. 66-74.
- [2] Open Services Gateway Initiative (OSGi) Alliance, <http://www.osgi.org/>.
- [3] Universal Plug and Play (UPnP) Forum, <http://www.upnp.org/>.
- [4] Simple Object Access Protocol (SOAP), <http://www.w3.org/TR/soap/>.
- [5] ISO/IEC DIS 29341, http://www.iso.org/iso/catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45415.
- [6] UPnP Device Architecture Version 1.0, <http://www.upnp.org/specs/arch/UPnP-DeviceArchitecture-v1.0.pdf>.
- [7] UPnP QoS Architecture Version 2.0, <http://www.upnp.org/specs/qos/UPnP-qos-Architecture-v2-20061016.pdf>.
- [8] OSGi Service Platform Release 4, <http://www.osgi.org/Specifications/HomePage>.
- [9] C. Lee, D. Nordstedt, and S. Helal, "Enabling smart spaces with OSGi," *IEEE Pervasive Computing*, Vol. 2(3), July-Sept. 2003, pp. 89-94.
- [10] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggle, "Towards a better understanding of context and context-awareness," in *Proc. Int'l Symp. Handheld and Ubiquitous Computing*, Springer-Verlag, pp. 304-307, 1999.
- [11] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and simple relational processing of uncertain data," in *Proc. IEEE ICDE*, Apr. 2008.
- [12] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong, "Approximate data collection in sensor networks using probabilistic models," in *Proc. IEEE ICDE*, pp. 48-60, Apr. 2006.
- [13] A. Singh, C. R. Ramakrishnan, I. V. Ramakrishnan, D. S. Warren, and J. L. Wong, "A methodology for in-network evaluation of integrated logical-statistical models," in *Proc. ACM SenSys*, pp. 197-210, Nov. 2008.
- [14] D. Chu, L. Popa, A. Tavakoli, J. M. Hellerstein, P. Levis, S. Shenker, and I. Stoica, "The design and implementation of a declarative sensor network system," in *Proc. ACM SenSys*, pp 175-188, Nov. 2007.