SSCP: An OSGi-based Communication Portal for Smart Space

Chang Cheng\textsuperscript{1}, Yue Suo\textsuperscript{1}, Yu Chen\textsuperscript{1}, Yuanchun Shi\textsuperscript{1}, Weikang Yang\textsuperscript{1}

\textsuperscript{1}Key Laboratory of Pervasive Computing, Ministry of Education Tsinghua National Laboratory for Information Science and Technology Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

c-cheng07@mails.tsinghua.edu.cn; \{suoyue, yuchen, shiyc, yangweikang\}@tsinghua.edu.cn

Abstract

Booming Internet and mobile computing technology require Smart Space to have stronger communication capability with varied devices, services and other Smart Space counterparts. In this paper, we propose SSCP, an OSGi-based communication portal to provide enhanced in-space and cross-space interactions. By adopting the well-modularized architecture of OSGi, SSCP features good scalability and extensibility. SSCP also enables communication with prevailed smart device protocols, such as UPnP. Furthermore, with an ECF Adaptor, communication with Smart Space counterparts, as well as the applications using message protocols like XMPP is enabled. A prototype of SSCP is developed and experimental scenario is undertaken, which demonstrate and evaluate our work.

Keywords: Pervasive Computing, Smart Space, OSGi, Communication Portal.

1. Introduction

As internet communication and mobile computing technologies are making their way into our daily life, we are seeing increasing necessity of developing platforms for building pervasive computing environment with stronger communication capability to the outside world. Smart Space, regarded as the test-bed for pervasive computing, has been calling attention to the researchers in recent years, such as Gaia [1], Interactive Workspace [2], iRoom [3], Smart Classroom [4], Open Smart Classroom [5], etc. Wherein, Agent-based, Component-based or Meta-based middleware has been developed on different levels of abstractions, enabling the distributed modules to easily discover, communicate and cooperate within the Smart Space, and thereby providing a solid software infrastructure for building Smart Space environment.

However, cross-space interaction through open network has not been fully addressed yet. With the development of the internet, it is highly necessary to bridge the gap between the islands of the distributed Smart Spaces and the services or devices among the different pervasive computing environment. There are several preliminary approaches to address this issue, such as a Web-Service-based solution [5], but further research and improvement are still needed. Also, almost all the communication protocol used in these Smart Spaces is relatively private that is not able to communicate with other prevailed protocols for smart devices, such as UPnP [6], IGRS [7]. Furthermore, a variety of communication protocols have emerged along with the prosperity of Internet, such as XMPP [8], MSN [9] and IRC [10]. It is highly valuable that Smart Space could “talk” with them without labored implementations for each protocol stack.

In this paper, we proposed a Communication Portal for Smart Space (SSCP) using OSGi [11] technology. By adopting OSGi architecture to Smart Space, we introduce a middleware framework where OSGi-based central management node facilitates better integration of multiple common services and provides better scalability. We choose OSGi because it is well-known as a mature architecture, validated by the success of Eclipse, which provides standard interface and service specifications. With SSCP, a generalized Event Bus for publish/subscribe messages between distributed modules is provided along with a variety of adaptors that enable the platform to communicate with multiple protocols. There are four adaptors in SSCP: 1) Directory Service (DS) in Open Smart Platform [5], which is our previous software infrastructure for Smart Space, is re-implemented that acts as an adaptor for Smart Space agents. 2) The UPnP adaptor enables Smart Space applications or services to communicate with UPnP devices and vice versa. 3) We also adopted Eclipse Communication Framework [12] as the adaptor to provide communication capability with the applications which uses different contemporary protocols outside of Smart Space. With the help of ECF adaptor, Smart Spaces could share its Event Bus to the outside. 4) By integrating previous SPAW and WSWA
the Web Service adaptor is built for Smart Space to communicate with applications or services using Web Service technology.

The rest of the paper is organized as follows: Section 2 motivates our research by reviewing existing software infrastructures for Smart Space and related technologies. Section 3 explores the architecture and implementation of the SSCP based on OSGi, which is validated by the experimental scenario introduced in Section 4. Section 5 compares the proposed SSCP with related work and presents future work. Finally, we conclude this paper in Section 6.

2. Related Work and Technology

2.1. Smart Spaces

Smart Space is regarded as the representative test-bed for pervasive computing. Several Smart Spaces projects have been developed or is still on-going in recent years [1][2][3][4][5]. To summary the common issues lying in different Smart Spaces, we separate the development of Smart Space into three phases: the Individual Smart Space (ISS), the Open Smart Space (OSS), and the Smart Community (SC) [13]. In ISS phase, building the smart human-computer interactive environment with varied distributed devices and applications in one Smart Space is the key issue, which has been achieved by most of related projects. OSS phase aims at bringing more openness into Smart Space. Roaming with users in or out of Smart Space, mobile devices and handheld devices brought by the users are able to seamlessly interact with Smart Space. Services and the applications in the internet are also able to contribute to the task. Moreover, OSS enables the communication interfaces for outside systems, such as other Smart Spaces, to access and collaborate with each other. In SC phase, multiple Smart Spaces connect with each other by specific social attribution or organized hierarchy. Not only the mechanism of sharing services and devices, but also enabling the users to seamlessly move among different Smart Spaces are the key issues of SC. Most of the related projects are still focusing on the OSS phases, only very few of them are trying to address the issue of SC. In this paper, we focus on providing better communication mechanism of OSS.

There are several related projects that could be referred to. iRos [2] and iCrafter [14] by Stanford is the software infrastructure for building their Interactive Workspace, which ties diverse devices with their own low-level OS and enables the users to flexibly interact with the devices and services together. Whereas, iRos/iCrafter lack of providing communication interfaces for the devices or services that does not belong to Smart Space, not to mention for other Smart Spaces. Metaglu [15] and Hyperglue [16] by MIT involve the multiple Smart Spaces resource management. Meanwhile, it totally relies on Java RMI technology, hindering other applications with it to join in. Gaia [1] is a middle infrastructure with resource management and provides the user-oriented interfaces for such physical spaces populated with network-enabled computing resources. Gaia provides well-supportive of services, however, lacks of emphasizing on providing communication mechanism to other Smart Spaces. Smart Platform [4] developed by Tsinghua University provides three different communication schemes and loose-coupled multi-agent encapsulation architecture. Its extended Open Smart Platform [5] addresses the services communication both inside and outside of Smart Space based on Web-Service technology, as well as provides the lightweight web-based interface for mobile devices to access and communicate with Smart Space. However, limited by the Web-Service technology, it lacks of supporting for other prevailed networking protocols for devices such as UPnP, IGRS, or protocols for applications, such as XMPP, MSN or IRC.

2.2. Related Technologies

OSGi (Open Service Gateway Initiative) is the module system for Java. It defines a way to create true modules and a way for those modules to interact at runtime.” [17] The original objective of OSGi was to build home service gateways. As it develops, OSGi has been extended to a much boarder area, including automotive industry, software development environments like Eclipse IDE, mobile devices and networking applications.

OSGi provides good dynamic modularization specification for Java runtime. Any framework which implements the OSGi standard specification is able to provide modularization mechanism. The module is called bundle in OSGi, which is basically a Jar file with additional metadata allowing monolithic applications to be split into multiple bundles. Also, it defines an application life-cycle management model and a service registry mechanism to facilitate the service-oriented platform. Service can be retrieved from the service registry in terms of the class name.

There are several services in OSGi that facilitate the communications in Smart Space. EventAdmin in OSGi provides event-based communication channel with topic feature. The latest OSGi specification has also integrated UPnP as one of its device specification [18] and there have been several implementations on it [19]. Therefore, it is possible for OSGi to communicate with UPnP-enabled devices.

In short, OSGi architecture provides modularization framework with extensibility and runtime management, which is the basis for our proposed approach in this paper. Among different implementations for OSGi, Equinox [20] is used because it is considered as the relatively mature OSGi implementation, which is...
famous for “fostering the vision of Eclipse as a landscape of bundles.”[20]

ECF (Eclipse Communication Framework) [12] is the framework for supporting the development of distributed Eclipse-based tools and applications. Recently it releases its latest version 3.0 along with Eclipse Galileo. The implementation of RFC 119 [22] is included in this version, which supports remote OSGi service using R-OSGi [22]. The elastic ECF supports varied protocols like XMPP, ActiveMQ / JMS, ECF generic, Skype, JavaGroups, etc.

The core connection structure of ECF is called the Container. From Container, different adapters are available to support different functions or services, which facilitate the communications to be easily maintainable and extensible. We use ECF to build one adapter in SSCP to enable the communications with varied protocols aforementioned.

3. Architecture

The SSCP is based on OSGi architecture, therefore developers could make use of the features provided by OSGi such as modularization, services registry and lifetime management. Components are made into OSGi as standard bundles which are of easy deployment and management.

Figure 1. Overall Architecture of SSCP.

Figure 1 illustrates the overall architecture of SSCP. There are four adapters in the bottom, which are responsible for the specific types of communications. They all share the same objectives of transforming between messages of specific protocols and OSGi Events with the help of the corresponding Abstract Services, while their underlying implementations are different. Both DS (adapter for Smart Space Agents) and UA (adapter for UPnP devices) implement in terms of the Importer and Exporter (refer to Figure 3), while the EA (adapter for applications of protocols supported by ECF) wrappers ECF, and WS (adapter for Web-Services) is built by integrating WSWA and SPAW in Open Smart Platform [5]. Generally speaking, the implementations of the adapters only need to fit the event model of communication required by SSCP, thus new adapters could be added flexibly without prior knowledge of existing adapters.

Abstract Services are actually OSGi services; they could be OSGi Event Listener, Event publisher, or even virtual Agent instance. They are responsible for publishing or subscribing messages to the Event Bus.

Event Bus is the core module providing publish/subscribe mechanism. Basically, the messages that published to the Event Bus will only be received by listeners with corresponded interest, namely those who subscribed to the event topic.

High-Level Common Services provides curial common services in Smart Space, such as Context Awareness or Experience Record, by exchanging high-level events of the environment and overall system status with Event Bus.

In the rest of the section, we will explain detailed implementations about DS, EA and Event Bus. UA and WS are omitted, because UA is similar with DS and WS is built on previous work. The only difference between UA and DS is that UA implements Importer and Exporter by using Felix [19].

3.1. DS (Directory Service)

DS is the central node for Agents in Smart Space to provide publish/subscribe communication mechanism. Previous DS in Smart Platform [4] is implemented as a particular Agent written in C++ on Windows Platform. DS is re-implemented in Java to be integrated into OSGi architecture. The auto-discovery based on multicast and publish/subscribe mechanism are kept the same in new DS, which is transparent to existing Smart Space Agents.

Figure 2. Architecture of Directory Service.

The architecture of DS is shown in Figure 2. Multicast server and client are responsible for auto-discovery. The Blackboard and Container Manager are responsible for maintaining the publish/subscribe mechanism and managing connections of containers respectively (Note that in Smart Space, DS and Agents...
are connected indirectly through multiple Containers in each computing devices in order to provide easier management and disconnect recovery).

DS also acts as the adaptor for Smart Space Agents. Generally, the Importer and Exporter are used to achieve the transformation between OSGi Events in Event Bus and Agent messages.

Figure 3 illustrates the architecture of Importer and Exporter. Importer is responsible for converting Agent messages into OSGi Event. It retrieves the message group in the message, assigning it to topic property in OSGi Event, and uses Event Publisher to publish the converted event on the service registry. Any interested Listener can retrieve the message with the same topic. Exporter listens to OSGi Event and transforms it into Agent message by similar mechanism. Event Listener and Event Publisher are on the level of abstract services. Moreover, the Importer and Exporter needs to deal with XML-based messages. We chose Dom4j to be the helper of converting between Agent messages and SSCP Events.

3.2. ECF Adaptor

We wrapped ECF into the ECF adaptor in SSCP, shown in Figure 4. Since ECF focuses on Eclipse-based tools and platforms and is implemented on OSGi architecture, it is feasible to integrate it into SSCP. As ECF provides communication for different protocols, it enables SSCP to communicate with the applications in these protocols, of which the r-OSGi and XMPP are taken examples here.

3.3. Event Bus

Event Bus is the rendezvous of OSGi Events. OSGi services with Event Listener could receive related events they are interested in. It is based on the EventAdmin specification in OSGi; therefore, it is well-fit in OSGi Architecture.

By making use of Remote Service adaptor of ECF, we implement a RemoteEventAdmin to enable Event Bus to publish and subscribe event from remote Smart Space which has SSCP supported. The RemoteEventAdmin could publish events to local event listeners as well as the remote counterparts.

3.4. High-level Common Services

High-level common services of Smart Space are also designed to be integrated into SSCP, which rely on the Event Bus to communicate with other devices, services
and applications in Smart Space. For instance, Context Awareness service acquires event of context-data from a variety of sensor agents. User Experience Recorder captures all the events in Event Bus for further review and analysis.

4. Scenario

A scenario as remote meeting shown in Figure 6 is designed to demonstrate and evaluate SSCP.

Two distant meeting rooms as Smart Spaces are deployed in our building: One is in the 3rd floor; another is in the 5th floor. Student Li in 3rd floor is present a PPT which is interested about by laboratory in 5th, therefore two meeting room is connected by SSCP. Displays on both rooms are synchronized during the presentation. When Student Zhao has a question, he put it on the text discussion group. Students on both sides could join the discussion.

Figure 6. Two Meeting rooms connected by SSCP: 1) Central node with SSCP running upon; 2) Synchronized projectors; 3) Gtalk in HTC G2 as an XMPP client; 4) Node running with other Agents and virtual UPnP device.

Another important participant, Professor Wang, is caught by traffic jam and could not attend the meeting. Using cell phone with Gtalk, she could communicate with the meeting room remotely. By posting message to the Meeting Room in the contact list, which is emulated by EA, she could join in the online discussion group. By posting text of specific format, she could even make command on remote meeting room such as retrieving attendants’ name list.

UPnP device is discovered automatically in the meeting room. During the meeting, Student Cheng wants to print some material. He finishes the job by using UPnP enabled printer without labored configure and knowledge of local network. It is also easy for him to use other legacy UPnP device such as Media Center and Media Render.

Figure 7 shows the runtime architecture of this scenario. To enable synchronized display in two classrooms, we build two agents running: PPT Controller Agent and Display Agent. A PPT Controller Agent runs for the presenter on the main meeting room. It publishes PPT controlling message to the Event Bus in terms of the presenter’s command. It also detects and publishes the status of the slideshow, such as the current page of the slideshow. A PPT Display Agent is responsible for controlling the PPT presentation on its host, receiving the controlling messages from PPT Controller Agent and making corresponding reactions.

We build several virtual UPnP devices on CyberLink [23], since real UPnP devices are not available in our Lab right now. These virtual UPnP devices fully comply with UPnP standard and actually act the same as the real UPnP devices. An UPnP Control Point is developed as an abstract service using SSCP UA. The Control Point controls other UPnP devices and delegates the control for other Agents. A Talk agent is a simple group discussing agent running on the attendants’ laptops. With the EA’s remote event channel, two Smart Spaces virtually share the Event Bus and facilitate convenient cross-space communication.

Figure 7. Runtime architecture of Meeting Rooms Scenario.

HTC G2 with Gtalk client is used as the mobile phone in the scenario. A simulated Gtalk account is created online to represent the meeting rooms. The EA service collects events from the Talk Agent and sends XMPP message to those on the friends list, and vice versa. By communicating with this account, users could publish and receive message to other Talk Agent in both of the Meeting room. Although the message is limited to the plain text because of the Gtalk client on the Cell Phone, by communicating with the XMPP Command agent in the meeting room, user could virtually accomplish some task by sending commands to the agent. For example, the user could obtain who have attended the meeting by making a query to the Guest Service. For example, the command “Guest Service: Is *” will be turn into the Event on the OSGi Event Bus, and a list of attendant’s names will be returned by XMPP interface.

5. Discuss and Future work

The SSCP has been developed to facilitate communication among different devices, services and applications in Smart Space. Messages in different protocols are transformed by adaptors and converged into the Event Bus. Although this mechanism is loosely coupled and provides low-leveled generalization, this
event-based mechanism brings some limitation on high-level applications and complicated scenarios.
Semantic information to the OSGi registry is able to be added in order to enable the client to access the Semantic Service. It requires further research on how to integrate Semantic Language into OSGi architecture, e.g. using standard OWL ontology language on OSGi services properties. By using shared vocabularies defined by the specification, agents requiring semantic information could use other semantic information of other services automatically, which is better than the Ad-hoc way to deal with Semantic information.

Also, we adopt ECF to build the EA in SSCP. Our approach is able to communicate with remote event by using Remote Services adaptor, as well as with XMPP protocol by using Presence adaptor, which are both compatible with the Event model in Event Bus. In fact, ECF supports other communication adaptors for different protocols such as File Transportation, Discovery, VOIP, etc, which could be further integrated into our SSCP. For example, File Transportation is adopted by Eclipse P2 [24] provisioning platform. Within ECF, it is possible to build our provisioning platform for Smart Space application, which will greatly contribute to the overall usability.

As ECF is still relatively developing framework in Eclipse projects, more supported protocols are expected to be emerging quickly. Recently building SSCP ECF adaptor in SSCP is only an ad-hoc way. It is planned to provide a Common ECF provider for Smart Space, whose benefit is obvious: when new protocols is supported by ECF, it will automatically contributes to the SSCP and Smart Space.

6. Conclusion

SSCP provides a communication portal for Smart Space and allows different protocols to communicate with Smart Space. By adopting OSGi architecture, SSCP enables better integration of multiple common services and provides better scalability. A generalized Event Bus for publishing/subscribing messages is created and different adaptors are presented for converting messages from/to different protocols to/from the Event in Event Bus. Directory Service (DS) Adaptor, UPnP Adaptor and ECF Adaptor are implemented, which enable the Smart Space to communicate with other remote Smart Space, local UPnP devices and remote applications using prevail message protocol, respectively.

While SSCP provides fundamental channel for event-based communication, it has limitations on accomplishing complex tasks. Further research on extending other types of communication scheme (e.g. File Transportation, Multimedia Streaming) into SSCP is still an open issue. Nevertheless, SSCP is designed as the adaptor-based and modularized architecture, therefore new adaptors and services are able to be easily integrated and to contribute to the Smart Space. Consequently, SSCP is an attractive approach for building communication portal for Smart Space.

References