A Dynamic SCA-Based System for Smart Homes and Offices

Thomas Calmant[1,2], João Claudio Américo[1], Didier Donsez[1] and Olivier Gattaz[2]

[1] Grenoble University, LIG ERODS Team, Grenoble, France
[2] IsandlaTech, Grenoble, France
{Joao.Americo,Didier.Donsez}@imag.fr,
{Thomas.Calmant,Olivier.Gattaz}@isandlatech.com

Abstract. We demonstrate in this paper the interoperability and the dynamism capabilities in SCA-based systems in the context of smart habitats. These capabilities are due to three developed tools: a Python-based OSGi runtime and service-oriented component model (Pelix and iPOPO, respectively); a tool to publish SCA services as OSGi services (NaSCAr) ; and a tool to publish UPnP devices as SCA services (UPnPServiceFactory). By this, we have developed a service robot and robot pilot agent, which can dynamically add and remove sensors and widgets. This use case follows and responds to the ubiquitous computing trend and the runtime adaptivity needed in such systems.

Keywords: SCA, Service-Oriented Architectures, Component-Based Design, Dynamic Adaptability, Smart Habitats.

1 Introduction

One of the biggest concerns in smart habitats is the integration of the often heterogeneous networked systems and devices that compose them. One of the principal solutions used in the industry nowadays to ease developers' burden concerning integration issues is the use of service-oriented architectures. Thus, service platforms, like UPnP [1], IGRS [2], Echonet [3] and DPWS [4], are very often used, despite their lack of support to ease software development, such as component models. Meanwhile, the Service Component Architecture (SCA) [5] is a technology-agnostic standard for developing service-oriented components. SCA has several runtime implementations, such as OW2 Frascati[6], Apache Tuscany[7], Oracle Tuxedo[8] and IBM WebSphere Application Server Feature Pack for SCA[9]. Nevertheless, these platforms (and the SCA specification itself) do not take into account another problem inherent to networked systems and devices: dynamism. In these highly dynamic scenarios, components (i.e. devices) have dynamic availability, and may appear and disappear several times during the execution time. In this demonstration, we present system infrastructure which enables both the dynamism and the interoperability between two service-oriented component models (SOCMs), one targeting Java/OSGi applications, and the other Python applications. This infrastructure consists of a robot whose devices can be dynamically added and removed. These devices' information collectors are implemented as components in Python and Java-based service-oriented component models.

A. Ghose et al. (Eds.): ICSOC 2012, LNCS 7759, pp. 435–438, 2013.

2 Used Tools

This demonstration software stack is based on four frameworks, the last three being developed by the authors.

The OSGi Service Platform[10] is the basis for most existing SOCMs, such as iPOJO [11] and Declarative Services [12], and it is considered as the standard *de facto* for modularity in Java. It also offers life-cycle management for its modules and enables them to interact by means of its service registry. SOCMs ease components development, by automatically managing SOA mechanisms, such as publication and discovery.

IPOPO is a Python-based service-oriented component model inspired of the iPOJO component model. As iPOJO turns on the top of an OSGi Service platform, iPOPO is executed on an Python-based OSGi container called Pelix.

NaSCAr[14] is a tool which transforms SCA composites into OSGi bundles and deploys them on an OSGi Service Platform. NaSCAr is also based on the iPOJO component model and includes a SCA binding extension that enables dynamic service publication, discovery and binding for SCA composites.

UPnPServiceFactory is a tool which exposes UPnP services in an OSGi Registry.

We intend to present in this demonstration the interoperability between service-oriented component models enabling software development with Java (OSGi) and Python, by composing iPOPO and NaSCAr/iPOJO components dynamically. The components in the system correspond to sensors (robots) and widgets, which are installed as plug-ins and used to display data [15].

3 RobAIR, a Telepresence Robot for Smart Habitats

The use of robotics for personal assistance (named *service robotics*) is growing. It differs from *industrial robotics* in that the service robot coexists and cooperates with humans. In Europe, this area is at the heart of very important societal issues, due to its aging population. A telepresence robot is a mobile service that allows persons to attend meetings, visit factories, warehouses, hospital rooms, museums and so forth. It can be used by technical experts as well as by elderly persons wishing to travel without leaving their houses. A basic implementation would be a video conferencing system mounted on a mobile robotic platform, with or without self-motion control.

In this demonstration, we present RobAIR (Robot for Ambient Intelligence Rooms, depicted in Figure 1), a telepresence robot which can be deployed in smart habitats.

Pilots can remotely control the robot by using an user-agent running on a PC or a tablet. The hardware platform of RobAIR is based on a Wifibot [16]. We have added sensors for piloting (*e.g.* lidar and pan-tilt webcam) and collecting environmental data (*e.g.* geiger counter). New sensors can be dynamically plugged to adapt RobAIR to a specific domain usage (*e.g.* safety inspection for monitoring elderly people or museum visits). Their corresponding components can be dynamically installed as well, without the need to restart the robot. While connected to the robot, the user agent queries the robots components in order to list the current sensors and then dynamically deploys and starts the widgets for the visualization of the sensors data (in this case, radiation level).

The software components inside RobAIR were developed using a SOA approach. They were modeled as SCA components. These components were deployed on an OSGi

Fig. 1. RobAIR - platform and sensors (geiger, toxic gaz)

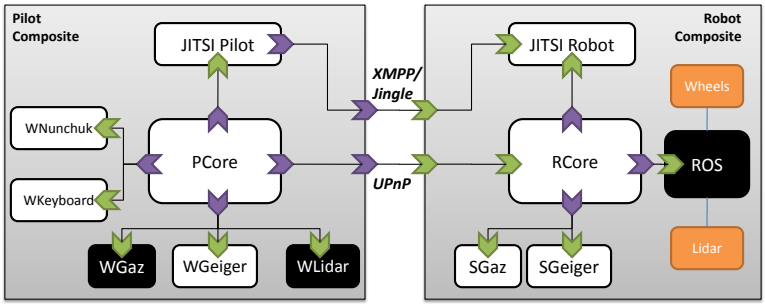

Fig. 2. SCA Modelisation of RobAIR's Robot and Pilot components

platform in both pilot and robot, thanks to NaSCAr and UPnPServiceFactory capabilities. The complete architecture (containing the robot and the pilot's user-agent) is represented by the Figure 2. Black boxes represent Python components, whereas white boxes correspond to Java components.

The robot and the pilot's user agent composites are bound to each other at runtime. The robot composite publishes its services using the UPnP protocols. Pilot's user agent can discover this service and get its description inside a network, like Small-office/Home-Office (SOHO). We have implemented a set of UPnP Device Control Protocols containing three distinct services: a *RobotPilot* service, for piloting the robot; a *SensorCollect* service, to collect the robot sensors; and a *XMPPSession* service, to allow a XMPP audio/video session between the robot and pilot agents.

The two main components are implemented in Java. Inside the *Robot composite*, the sensor components (*i.e. SGaz*, *SGeiger* and *SLidar*) collect the data from their corresponding sensors. They are instantiated and bound to the component *RCore* when the sensor is plugged to the robot's main board USB connectors, by the means of the UPnPServiceFactory tool. The component *RCore* is also connected to the *JITSI* component in order to exchange audio/video between the robot and the pilot. In turn, inside the *Pilot composite*, the widget components (*i.e. WGaz*, *WGeiger* and *WLidar*) display collected sensors data. They are instantiated and bound to the component *PCore* when *RCore* notifies changes in the robot's sensors configuration. In addition, controllers (such as Nintendo Wii's Nunchuk and keyboards) can be dynamically bound to *PCore* for piloting the robot.

4 Conclusions

This demonstration paper presents the dynamic and opportunistic composition of heterogeneous services in the highly-variable context of smart habitats. Most of the implemented components aimed to dynamically integrate heterogeneous legacy and off-the-shelfs components, like ROS[17] (250 KLoC in C++) and JITSI[18] (750 KLoC in Java). More information about RobAIR can be found at `http://air.imag.fr/mediawiki/index.php/RobAIR-Wifibot`.

References

1. UPnP: Universal Plug and Play, `http://www.upnp.org/`
2. IGRS: Information Device Intelligent Grouping and Resource Sharing QoS Specification for Wireless UWB networks (2008)
3. Matsumoto, S.: Echonet: A Home Network Standard. IEEE Pervasive Computing 9(3), 88–92 (2010)
4. Zeeb, E., Bobek, A., Bohn, H., Golatowski, F.: Service-oriented architectures for embedded systems using devices profile for web services. In: Proceedings of the 2nd Int'l IEEE Workshop on SOCNE 2007, pp. 956–963 (2007)
5. Open Service-Oriented Architecture Collaboration: Service Component Architecture Specifications (2007), `http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications`
6. Seinturier, L., et al.: Recongurable SCA Applications with the FraSCAti Platform. In: Proceedings of the 6th IEEE Int'l Conference on Service Computing, pp. 268–275 (2009)
7. Apache Foundation: Apache TuSCAny, `http://tuscany.apache.org`
8. Oracle Corporation: Oracle Tuxedo, `http://www.oracle.com/technetwork/middleware/tuxedo/overview/index.html`
9. International Business Machines Corporation: IBM WebSphere Application Server Feature Pack for SCA, `http://www-01.ibm.com/software/webservers/appserv/was/featurepacks/sca`
10. The OSGi Alliance: OSGi service platform core specification, release 4.3 (2011), `http://www.osgi.org/Specifications`
11. Escoffier, C., Hall, R., Lalanda, P.: iPOJO: An Extensible Service-Oriented Component Framework. In: Proc. IEEE Int'l Conf. Services Computing (SCC 2007), pp. 474–481 (2007)
12. Cervantes, H., Hall, R.: Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model. In: ICSE 2004, pp. 614–623 (2004)
13. Calmant, T., Américo, J.C., Gattaz, O., Donsez, D., Gama, K.: A dynamic and service-oriented component model for Python long-lived applications. In: Proceedings of the 15th ACM SIGSOFT Symposium on Component-Based Software Engineering, pp. 35–40 (2012)
14. Américo, J.C., Donsez, D.: Service Component Architecture Extensions for Dynamic Systems. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) ICSOC 2012. LNCS, vol. 7636, pp. 32–47. Springer, Heidelberg (2012)
15. Gama, K., Pedraza, G., Lévêque, T., Donsez, D.: Application management plugin-ins through dynamically pluggable probes. In: Proceedings of the 1st Workshop on Developing Tools as Plug-ins (ICSE Workshop), pp. 32–35 (2011)
16. Wifibot EURL: Wifibot, `http://www.wifibot.com/`
17. Robot Operating System, `http://www.ros.org/`
18. JITSI Community, `https://www.jitsi.org/`