A Cross-Layered Diagnostician in OSGi Platform for Home Network

Pang-Chieh Wang, Yi-Hsuan Hung, and Ting-Wei Hou

Dept. of Engineering Science, National Cheng-Kung University,
Tainan, Taiwan
{pangchieh, elegance, hou}@nc.es.ncku.edu.tw

Abstract. The service gateway in a home network plays an important role as a centric service controller. New services can be updated and plugged by the operators or users of the gateways. But more services in the gateway bring more complex service relations and may bring some unexpected exceptions and conflicts. This research summarizes the requirements of diagnosis in open service platforms and integrates several techniques to develop a cross-layer approach for detecting the service conflict fault, handling the general exceptions, and diagnosing the device fault on OSGi platforms. This solution, called Diagnostician, helps users and operators handle the problems in OSGi and reconfigure the system.

Keywords: Service gateway, OSGi, diagnosis, cross-layered.

1 Introduction

There are a lot of emerging researches on the digital home and smart appliances. Based on the technologies of home networking, more and more smart appliances are connected in the home networks. They are able to communicate and cooperate with each other through the home gateway. The Open Service Gateway initiative Alliance [12] defines such a service-oriented and component-based platform called OSGi Service Platform. With the OSGi technology, one complicated job which requires controlling several devices can be done automatically.

With the increasing number of services for improving the user's daily life, the user would feel more convenient to enjoy the benefits of these services. Unfortunately, the complexity of operating these networked smart devices also increases significantly, which make users spend more time figuring out how to set them up, what functionality they can do, and how to keep them working [5]. Moreover, common users do not know how to troubleshoot the services when they encounter some faulty situations such as the network is jammed, the service works inappropriately, or the gateway seems to be broken, etc. Hence the robustness of the open service platform becomes one of the major concerns of residents in smart homes [3].

In addition, most faults except hardware problems are due to the inappropriate operations of the user while assuming that the services are trustworthy. Since all the members in a family will request to access the resources in the digital home, the operations of distinct services may conflict when they need to access a device at the same time. If these inappropriate operations are collected in advance and maintained by the service providers, this kind of faults can be prevented before the problem occur rather than diagnose such problems after the user has lost his/her patience.

The exceptions also occur in the service platform and require an exception handling mechanism for detecting and resolving them. Although each service should have its own exception handling mechanism, there should be a general exception handling mechanism at the higher level of a service platform. Thus the general exceptions such as installing service failed, stop service failed, and so on can be handled, even resolved automatically, by the service platform.

In order to resolve the above faulty conditions, we propose a cross-layer approach on the OSGi platforms for detecting conflict(s) between distinct services. The high level exception handling mechanism is also addressed on the OSGi platform. In addition, the on-demand diagnostician embedded in an OSGi platform is able to assist the user when he/she encounters some problems in operating the services.

In section 2, we categorize the problems about the diagnostician would encounter in the service gateway and some other researches are introduced, too. The design of the proposed diagnostician in the service gateway and some results are presented in section 3. The last is the conclusion and future work in section 4.

2 Related Works

Here we categorized the problems of services gateways and indicates the characteristics of each problem.

2.1 Challenges on Fault Diagnosis

The issues of software debugging and fault diagnosis are often taken into consideration as designing various kinds of computing systems like operating systems, embedded or distributed systems, and network computing, etc. Each of these computing systems would define different kinds of challenges it would face, respectively. To design a fault diagnosis model on an open service framework, the challenges of fault tolerant pervasive computing defined in [14] can be applied. The authors in [14] discuss four fault tolerant issues:

1. **Fault Detection:** It is a difficult problem to do an effective fault detection task in a pervasive computing environment. In a common way, the device or application can be periodically checked to see if it is alive such as heartbeat messages. The drawback is that plenty of devices and applications make the data traffic of the heartbeat messages increases significantly.
2. **Fault Containment:** The fault that has been detected needs to be solved or isolated right away to block the spread of more serious faults. This is a critical problem, especially in the pervasive computing environment, since there are many services work together. A job would not terminate and even get worse if one service is down causes the other services operate incorrectly.

3. **Transparent Fault Tolerance:** One of the key characteristics of pervasive computing is transparency [16]. The pervasive system should mask all the faults and try not to disturb the user. In other words, the system needs to decrease the user's awareness as much as possible.
4. **Good Fault Reporting Mechanisms:** The system needs to report the fault information and suggest a solution to the user whenever a fault has been detected regardless of whether this fault can be solved automatically or not. Something needs to be noticed is how to report to the user in some kind of non-intrusive ways.

2.2 Exception Handling

The conditions which are brought to the attention of the operation's invoker while attempting to perform some operations are called exception conditions [6]. Then the invoker needs to respond to the conditions. The operation of bringing an exception condition to an invoker's attention is called raising an exception. Handling the exception is defined as the invoker's response. An exception condition would make the system inconsistent and may result in system failure.

The services of open service framework are increasing in a significant speed. The number of exceptional conditions is incident to various services. In order to enhance the robustness, fault tolerant techniques are introduced. One of the most important schemes for detecting and recovering errors is the exception handling mechanism [4]. The exception handling mechanism could also be used for structuring fault-tolerant activities in a system. Exception handling [2] is an acknowledged technique to ensure fault tolerance in a system which is up against a variety of faults [1].

2.3 Avoidance of Service Conflicts

The services on the OSGi framework are able to access and control home appliances such as televisions, telephones, and surveillance cameras, etc. If there are multiple services invoked by different users trying to access the same resource at the same time, a service conflict problem would occur because the resource is not available. Neither OSGi framework nor the service itself can handle such an unexpected error. Thus service mediation is necessary because there are various services running on one gateway operated by multiple users, which results in service conflict problem [11].

A common solution is to set priority. A rule-based controlling framework for a context-aware system [9] avoids the conflict to each device by simply setting and checking the priority. A conflict occurs when multiple rules indicating that different actions are required for the same device. The framework provides a mechanism to detect a conflict among multiple rules automatically. Thus the appropriate operation is chosen according to the predefined priority.

Since the context used to determine the priority is fixed, the process of service control is uniform such as certain service always gets higher priority. This is not appropriate for avoiding device conflicts [10]. Thus Ogawara, Kobayashi et al. develop a service control platform for the user-oriented home network services called "Home Service Harmony" [10]. The system estimates the importance of each service according to the user's preference and the usage history. The resources are assigned to services in accordance with the importance of each service. Because the importance is changing during runtime, the conflicts among resources can be avoided dynamically. However, as they comment, the system is difficult to handle controls if there are several users trying to use the same service. This is a significant drawback for applying this mechanism to an open service environment with multi-users.

We separate the above researches into three groups in Table 1 and highlight the techniques applied in these researches. The first group, called Fault Diagnosis Group, contains the fault diagnosis models discussed in section 2.1. The second group, called Exception Handling Group, includes all the exception handling approaches discussed in section 2.2. The researches discussed in section 2.3 are contained in the Service Conflict Avoidance Group. If there is a check sign, it means that the works in this group have the functionality described in the left column of this row. Otherwise the functionality is not supported. In addition, there are several functionalities which are not supported by all the researches in a group, i.e. only some are able to do that job. A triangle sign is used to describe this kind of incomplete conditions.

Table 1. A comparison between the related researches

	Fault Diagnosis Group	**Exception Handling Group**	**Service Conflict Avoidance Group**
Fault Detection	√	N/A	N/A
Fault Correction	√	N/A	N/A
Fault Reporting	△	N/A	N/A
Exception Handling	N/A	√	N/A
Exception Reporting	N/A	√	N/A
Resource Access Conflict Avoidance	N/A	N/A	√
Human Behavior Conflict Avoidance	N/A	N/A	△
√: All the researches in this group support the functionality. △: Not all the researches in this group support the functionality.			

3 Design and Implementation

3.1 Design of Diagnostician

Diagnostician is designed on the OSGi framework to solve the problems discussed in previous sections. The architecture of the whole framework is shown in Fig. 1. The Original Component blocks refer to the components that the OSGi framework originally have. We embed the Automated Logging Framework (ALF, one of Major Component blocks) between OSGi framework and Java Virtual Machine (JVM) to catch all the method invocation events triggered from JVM. There is a Conflict Detector (one of Sub Component block) inside ALF to monitor the OSGi service activities.

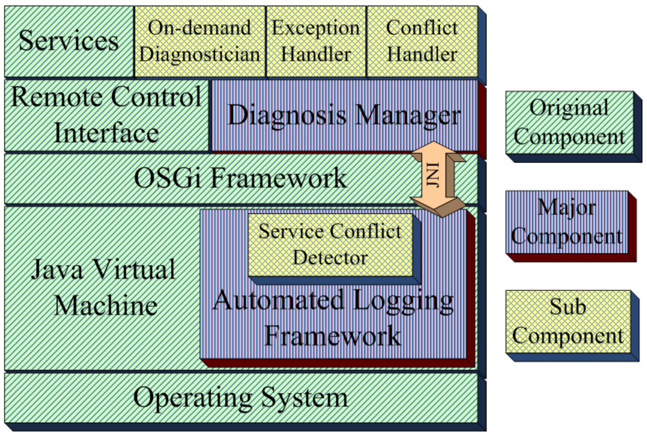


Fig. 1. The architecture of proposed Diagnostician

In the upper layer, the Diagnosis Manager (another Major Component block), which is an OSGi bundle, manages all the faults. It is the manager of the three sub components: On-demand Diagnostician, Exception Handler, and Conflict Handler. Each component handles one kind of faults. In addition, the On-demand Diagnostician is embedded in the original remote control interface and managed by the Diagnosis Manager. Diagnostician is able to assist the user when he/she encounters some problems like the occurrence of unexpected operations or device error. The following paragraphs show the details of the design concept and the implementation of each component.

- **Diagnosis Manager**
  The Diagnosis Manager bundle manages all the faulty situations with the upper sub-component as On-demand Diagnostician, Exception Handler, and Conflict Handler. This bundle is a two-way bridge between the OSGi framework and all the fault detectors. It could not only communicate with the fault detector in native JVM level, but also perform the administrative actions on the OSGi framework like stopping an illegal bundle, showing some human-readable messages, and so on.

- **Conflict Detector**
  As several services need to access and control the same device concurrently on the OSGi platform, it is like the 'race condition' which is a classical issue in process synchronization of operating systems [7]. We found that this problem would happen on OSGi platform. To take precautions against the race condition, we have to ensure that only one service/method at a time can access the device/resource. The solution to the critical-section problem [7] is taken into consideration. When the device is in use, no other related methods are allowed to get the control of the same device. Thus, this device controlled by one service is "mutually exclusive".

- **Modified ALF**
  The tool, Automated Logging Framework (ALF), is used to help catching the event of method entry. ALF is modified to have the conjunction of OSGi platform and itself. Thus ALF can notify the Diagnosis Manager to do some operations on OSGi platform when a mutually exclusive service is detected. In order to communicate between ALF (implemented in standard C++) and OSGi framework (implemented in JAVA), some JNI (Java Native Interface) calls are added. Thus ALF could set the bundle ID of the newly loaded class while handling a class load event.

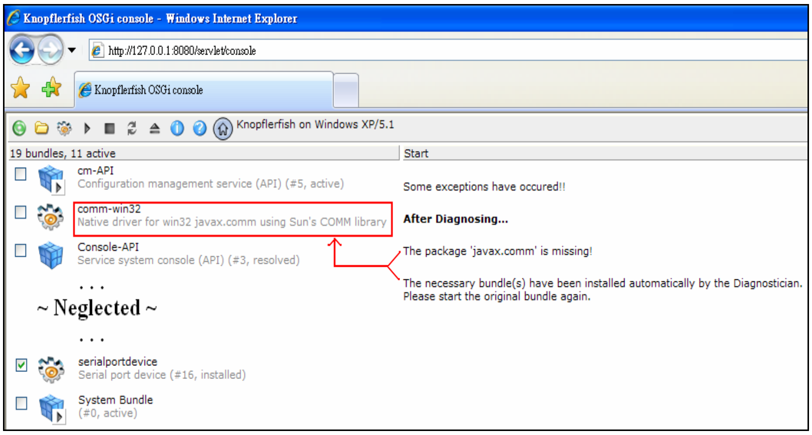
3.2 Implementation Result

We implemented the proposed Diagnostician on the OSGi framework to verify our solutions to handle the faults. The components of the prototype are listed in Table 2. The Knopflerfish 2 [7], the open source implementation of OSGi framework Release 4, is used. The version of JVM is J2SE 1.4.2 running on Windows XP. The remote control interface is modified from the httpconsole bundle developed by Knopflerfish. A user can access the remote control interface via Internet since a web server on the OSGi platform is activated. In addition, ALF is used as a tool for catching JVM events.

Table 2. A prototype

Software	Name or Version	Hardware	Information
Operating System	Windows XP Pro.	**CPU**	Intel Pentium M
Java Virtual Machine	J2SE 1.4.2	**Clock**	1.73 GHz
OSGi Framework	Knopflerfish 2 (OSGi Release 4)	**Hard Disk**	60 GB 4200 rpm
Network	Ethernet 1Gbps	**Memory**	1G ram
Web Server	Knopflerfish Http Server		
Additional Tool	Automated Logging Framework		

The exceptions may occur anytime. In a scenario, we assume that the user had successfully installs the 'Serial Port Device' service on the prototype OSGi framework. The remote control interface shows the messages which are responses from the OSGi framework. After starting this bundle, the messages shown in Fig. 2 indicating that an exception occurs because of missing a package called 'javax.comm'. The exception handler then finds and installs the necessary bundle automatically. Thus the user can start the Serial Port Device bundle again successfully without the occurrence of exceptions.


Knopflerfish OSGi console - Windows Internet Explorer

http://127.0.0.1:8080/servlet/console

Knopflerfish OSGi console

Knopflerfish on Windows XP/5.1

19 bundles, 11 active

- cm-API
  Configuration management service (API) (#5, active)
- comm-win32
  Native driver for win32 javax.comm using Sun's COMM library
- Console-API
  Service system console (API) (#3, resolved)

. . .

~ Neglected ~

. . .

- serialportdevice
  Serial port device (#16, installed)
- System Bundle
  (#0, active)

Start

Some exceptions have occured!!

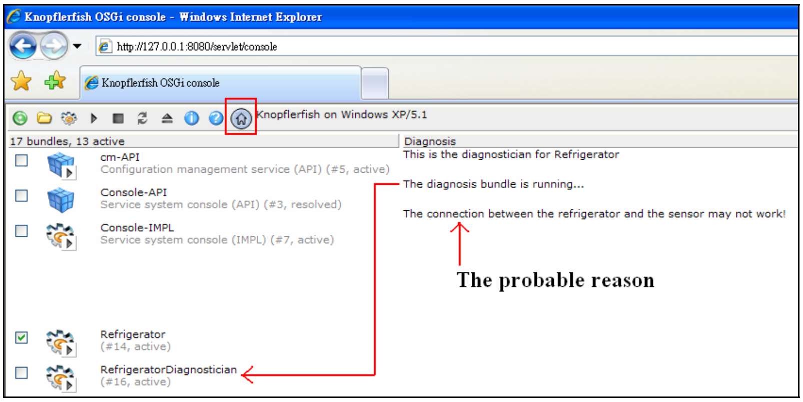
After Diagnosing...

The package 'javax.comm' is missing!

The necessary bundle(s) have been installed automatically by the Diagnostician.
Please start the original bundle again.

Fig. 2. The snapshot after diagnosing the exception

If the user finds out that the milk box is actually empty and doubts that the Refrigerator service may have some problem, he/she can click the button in the square of Fig. 3 to activate the On-demand Diagnostician. After the diagnosing has been done by downloading and starting a corresponding diagnosing bundle, the final result of the diagnosing process is shown on the right side of the screen as illustrated in Fig. 3. Therefore, the user can figure out what may be broken.


Knopflerfish OSGi console - Windows Internet Explorer

http://127.0.0.1:8080/servlet/console

Knopflerfish OSGi console

Knopflerfish on Windows XP/5.1

17 bundles, 13 active

- cm-API
  Configuration management service (API) (#5, active)
- Console-API
  Service system console (API) (#3, resolved)
- Console-IMPL
  Service system console (IMPL) (#7, active)
- Refrigerator
  (#14, active)
- RefrigeratorDiagnostician
  (#16, active)

Diagnosis

This is the diagnostician for Refrigerator

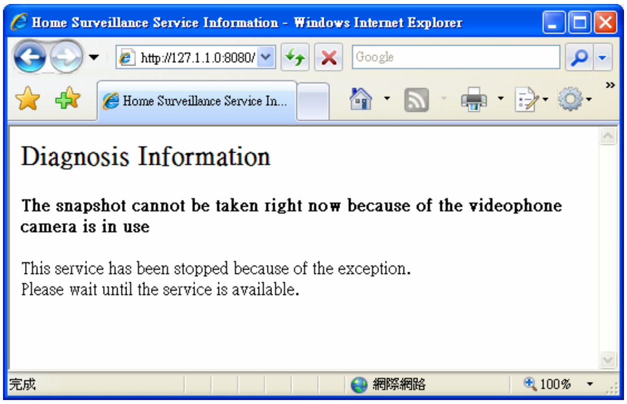
The diagnosis bundle is running...

The connection between the refrigerator and the sensor may not work!

The probable reason

Fig. 3. The snapshot after performing the On-demand Diagnostician

For another example, a service conflict may occur as the mother wants to take a picture by the camera, controlled by Home Surveillance Service, via the link of remote control interface. If there is not any conflict, the snapshot would be taken and shown on the web page. Otherwise the conflict fault would be detected and this service becomes unavailable because the Diagnosis Manager would stop it. The picture is unavailable and the messages are shown in Fig. 4.


Home Surveillance Service Information - Windows Internet Explorer
http://127.1.1.0:8080/
Google
Home Surveillance Service In...
完成
網際網路
100%

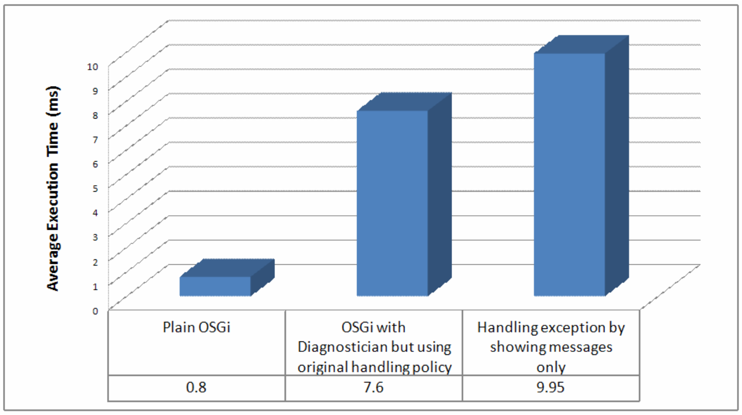
Diagnosis Information

The snapshot cannot be taken right now because of the videophone camera is in use

This service has been stopped because of the exception.
Please wait until the service is available.

Fig. 4. An example of diagnosis information

We evaluate the execution time when handling exceptions in different conditions and show the result in Fig. 5. The plain OSGi is without our Diagnostician and the time means an exception occurs then the service stopped. After installing it, an exception occurred with our Diagnostician catching and handling it. It takes 7.6ms and 9.95ms while showing the message on the user interface. The performance of the proposed solution seems not good because the ALF takes time in filtering and analyzing the information. In realistic, the time is still short enough to make user accept it. If there is not a Diagnostician in the service gateway, the plain OSGi cannot suggest and locate the possible reason of the exception, which would require a user or an operator spend a lot of time to find it out.



Plain OSGi	OSGi with Diagnostician but using original handling policy	Handling exception by showing messages only
0.8	7.6	9.95

Fig. 5. The performance evaluation of the Diagnostician

4 Conclusion and Future Work

The robustness of an open service platform is a critical issue. End users applying the smart home technology to their current houses look forward to improving their lives in a more convenient way, rather than bringing more troubles in dealing with the complex services and faulty situations frequently like some fault scenarios in this research. A cross-layer approach is proposed for detecting the fault(s) of service conflict(s) and handling the general exceptions on OSGi platforms. The OSGi platform with the proposed cross-layer approach is able to detect the service conflict fault in advance while the operations of distinct services called by different users conflict with each other. The general exceptions, caused by some inappropriate operations of a user, are handled as well. Both of the faults are resolved automatically without the user's awareness. The user would receive the human-readable information, which describes the faulty situation and what solutions have been done, shown on the user interface such as the web page. Furthermore, an on-demand Diagnostician module can be triggered by the user when he/she encounter some problems in the reaction of some probably faulty devices. The on-demand Diagnostician then installs the corresponding diagnosis bundle and begins to analyze the specified service and/or device. The results and suggestions are provided to the user for figuring out what service is crashed or what device is broken.

The advantages of the proposed approach is that the end users would be pleased without being bothered about the faulty situation, the service providers would discover that the requests for technical supports are reduced, and the service designers would take their ease with developing new services without caring about the operations may conflict with other existing services.

The current approach only focuses on the faults of service conflict and exception. Other fault tolerance techniques like auto reconfiguration and security issues would be considered in the future. More complete exceptions would be included in the exception handler rather than the general exceptions only. The interface for presenting the detailed information would be improved in a more user-friendly way and applied in various devices such as TV display, mobile phone, and so on.

References

1. Budi, A., Alexei, I., Alexander, R.: On Using the CAMA Framework for Developing Open Mobile Fault Tolerant Agent Systems. In: Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, pp. 29–36. ACM Press, Shanghai, China (2006)
2. Cristian, F.: Exception Handling and Software Fault Tolerance. Transactions on Computers C-31, 531–540 (1982)
3. Edwards, W.K., Grinter, R.E.: At Home with Ubiquitous Computing: Seven Challenges. In: Abowd, G.D., Brumitt, B., Shafer, S. (eds.) Ubicomp 2001: Ubiquitous Computing. LNCS, vol. 2201, pp. 256–272. Springer, Heidelberg (2001)
4. Garcia, A.F., Rubira, C.M.F., Romanovsky, A., Xu, J.: A Comparative Study of Exception Handling Mechanisms for Building Dependable Object-Oriented Software. Journal of Systems and Software 59, 197–222 (2001)
5. Grinter, R., Edwards, W., Newman, M., Ducheneaut, N.: The Work to Make a Home Network Work. In: ECSCW 2005. Proceedings of the 9th European Conference on Computer-Supported Cooperative Work, pp. 469–488. Springer, Netherlands (2005)
6. John, B.G.: Exception Handling: Issues and a Proposed Notation. Communications of the ACM 18, 683–696 (1975)

7. Knopflerfish Project: Open Source OSGi Framework Implementation, http://www.knopflerfish.org/
8. Leslie, L., Robert, S., Marshall, P.: The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems (TOPLAS) 4, 382–401 (1982)
9. Nishigaki, K., Yasumoto, K., Shibata, N., Ito, M., Higashino, T.: Framework and Rule-based Language for Facilitating Context-Aware Computing using Information Appliances. In: 25th IEEE International Conference on Distributed Computing Systems Workshops, Columbus, Ohio, USA, pp. 345–351 (2005)
10. Ogawara, M., Kobayashi, E., Yoda, I.: Home Network Service Management Technologies. NTT Technical Review 3, 17–21 (2005)
11. Okugawa, T., Masutani, H., Yoda, I.: A Home Network Service Environment for Wide-Area Communications. In: Proceedings of 2005 Asia-Pacific Conference on Communications, Perth, Western Australia, pp. 14–18 (2005)
12. OSGi Alliance: Open Service Gateway Initiative Technology, http://www.osgi.org/
13. Sara, B., Bill, S., David, W.M., Barbara, R., Ylian, S.-H.: Broken Expectations in the Digital Home. In: CHI 2006. Conference on Human Factors in Computing Systems extended abstracts on Human factors in computing systems, pp. 568–573. ACM Press, Montréal, Québec (2006)
14. Shiva, C., Anand, R., Campbell, R.: Towards Fault Tolerance Pervasive Computing. IEEE Technology and Society Magazine 24, 38–44 (2005)
15. Utton, P., Scharf, E.: A Fault Diagnosis System for the Connected Home. IEEE Communications Magazine 42, 128–134 (2004)
16. Weiser, M.: The Computer for the 21st Century. Scientific American 265, 94–104 (1991)