Adaptive Component Management Service in ScudWare Middleware for Smart Vehicle Space

Qing Wu and Zhaohui Wu

College of Computer Science, Zhejiang University,
Hangzhou, Zhejiang, China, 310027
{wwwsin, wzh}@cs.zju.edu.cn

Abstract. Due to the complexities of increasing prevalence of ubiquitous computing, it poses a large number of challenges for middleware and component technologies. We believe that service-oriented component adaptation provides a principled means to achieve the flexibility and scalability required. The focus of this paper regards an adaptive component management service in the ScudWare middleware architecture for smart vehicle space. The contribution of our work is twofold. First, an adaptive component management service framework, including a resource abstract framework, is put forward to implement adaptive mechanism. Second, a component hook proxy is proposed in detail for adaptation. In addition, this service is validated by a series of experimental results.

1 Introduction

In recent years, many kinds of smart devices come into our life such as PDAs, mobile phones, and smart cameras. The physical world and information space integrate seamlessly and naturally. The computation is becoming embedded and ubiquitous [1], which provides more facilities for people. This computing environment demands plenty of computation resources for functional requests and performance requirements. However, the computation resources in environments are limited in terms of CPU computation capabilities, network bandwidth, memory size, and device power, etc. As a result, sometimes it cannot provide enough resources to execute applications successfully. In addition, changes of the heterogeneous contexts including people, devices, and environments are ubiquitous and pervasive. Therefore, it results in many problems in software middleware design and development. We consider "adaptation" is the key issue for software systems and applications to meet the different computing environments and the diverse run-time context. On the other hand, component-based and service-oriented software (CBSOS) architecture provides a novel infrastructure and a development platform for ubiquitous computing. Components are abundant, heterogeneous, autonomic, and multiple categories. Because ubiquitous computing aims at building a human-centric ideal world, all entities should communicate and cooperate with each other transparently and spontaneously. The CBSOS system provides a flexible and adaptive computing framework. Taking into account the influence of dynamic changes on computation adequately, we use the service-oriented, context-aware, and component-based methods for adaptation.

Vehicles play an important role in our daily life. People require more safety, comfort, and facilities in vehicles. We select a vehicle space[2] as a representative scene to study ubiquitous computing. Cho Li Wang[3] has proposed five types of software adaptation, consisting of data adaptation, network level adaptation, energy adaptation, migration adaptation, and functionality adaptation. Our current work focuses at the design-time and run-time adaptation including the computation resource, logic behavior semantic, and run-time context adaptation. We emphasize on adaptive component management at design-time and run-time in the ScudWare[4] middleware for smart vehicle space. An experiment prototype called "mobile music system" is built to demonstrate the feasibility and reliability of our methods and techniques. The paper brings forward the ScudWare middleware platform and an adaptive component management service framework. In addition, we have made experiments to test the performance of this service.

The rest of the paper is organized as follows. Section 2 describes the ScudWare middleware platform including smart vehicle space, CCM (CORBA Component Model) [5] specification overview, and the ScudWare middleware architecture. Then an adaptive component management service framework is proposed in Section 3. Specially, a resources abstract framework and the functions of this service are presented particularly. Section 4 gives a run-time component hook proxy mechanism. In Section 5, we give experiments study and evaluate the efficiency and performance of the service. Next, some related work is stated in Section 6. Finally, we draw a conclusion in Section 7.

2 ScudWare Middleware Platform

To implement smart vehicle space naturally and adaptively, we have built the ScudWare middleware platform conformed to the CCM (CORBA Component Model) specification. We use the ACE (Adaptive Communication Environment) [6] and the TAO (The ACE ORB) [7]. TAO is a real-time ORB (Object Request Broker) developed by Washington University. According to the application domain of smart vehicle space, we reduce the TAO selectively and add some adaptive services such as adaptive resource management service, context service, and notification service. ScudCCM, a part of ScudWare, is responsible for adaptive component management comprising component package, assembly, deployment, and allocation at design-time, and run-time component monitoring. As following, we introduce smart vehicle space, CCM specification and ScudWare architecture briefly.

2.1 Smart Vehicle Space

In recent years, a lot of developers have applied embedded, AI, and biology authentication technologies to vehicles. The drive capability, dependability, comfort, and convenience of the vehicle are improved greatly. When people go into smart vehicle space, they find many intelligent devices and equipments around

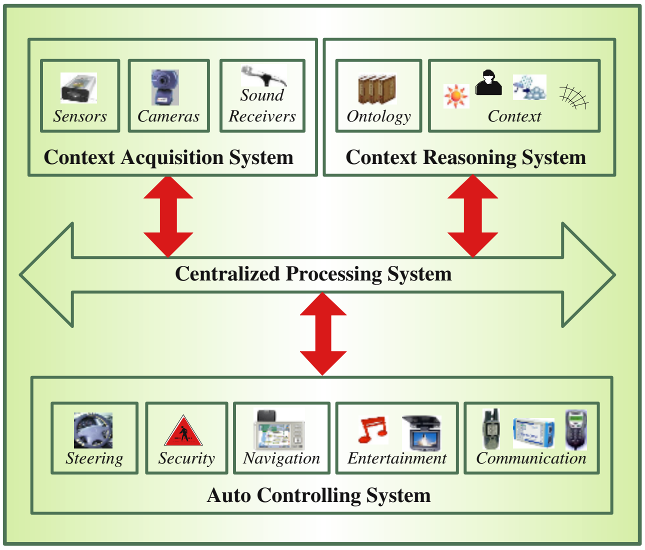

Fig. 1. Smart Vehicle Space

them. They communicate with these tools naturally and friendly. It forms a harmonious vehicle space where people, devices, and environments co-operate with each other adaptively.

Figure 1 describes the structure of smart vehicle space, which has four parts and is defined as SVS=(CA, CR, AC, CP). CA is a context acquisition system. $CA = ((\Delta State(pe, de, en), (sen, cam, sou))$ aims at sensing status changes of people, devices, and environments in the vehicle, including sensors, cameras, and sound receivers. CR is a context repository reasoning system. CR=(*context, ontology, domain, inference*) uses the correlative contexts and application domain ontology to make the manipulating strategy for adaptation. AC is an auto controlling system. AC=(*ste, com, ent, nav, sec*) consists of steering, communication, entertainment, navigation, and security subsystem. CP is a centralized processing system. Particularly, CP is the kernel of smart vehicle space, which controls above third parts co-operating effectively.

2.2 CCM Specification Overview

CORBA (Common Object Request Broker Architecture) is one of software middlewares, which provides language and operating system independences. CCM is an extension to CORBA distributed object model. CCM prescribes component designing, programming, packaging, deploying and executing stages.

CCM specification defines component attributes and ports. Attributes are properties employed to configure component behavior. Specially stated, component ports are very important, which are connecting points between components. There are four kinds of ports: facets, receptacles, event sources, and event sinks.

Facets are distinct named interfaces provided by component for client interaction. Receptacles are connection points that describe the component's ability to use a reference supplied by others. Event sources are connection points that emit events of a specified type to one or more interested event consumers, or to an event channel. Event sinks are connection points into which events of a specified type may be pushed.

In addition, CCM specification defines component home, which is a meta-type that acts as a manager for component instances of a specified component type. Component home interfaces provide operations to manage component lifecycle. CIF (Component Implementation Framework) is defined as a programming model for constructing component implementations. CIDL (Component Implementation Definition Language), a declarative language, describes component implementations of homes. The CIF uses CIDL descriptions to generate programming skeletons that automate many of the basic behaviors of components, including navigation, identity inquiries, activation, state management, and lifecycle management. The component container defines run-time environments for a component instance. Component implementations may be packaged and deployed. A CORBA component package maintains one or more implementations of a component. One component can be installed on a computer or grouped together with other components to form an assembly.

2.3 ScudWare Middleware Architecture

As Figure 2 shows, ScudWare architecture consists of five parts defined as $SCUDW = (SOSEK, ACE, ETAO, SCUDCCM, SVA)$. *SOSEK* denots SMART OSEK [8], an operating system of vehicle conformed to OSEK [9] specification developed by us. *ACE* denotes the adaptive communication environment, providing high-performance and real-time communications. *ACE* uses inter-process communication, event demultiplexing, explicit dynamic linking, and concurrency. In addition, *ACE* automates system configuration and reconfiguration by dynamically linking services into applications at run-time and executing these services in one or more processes or threads. *ETAO* extends ACE ORB and is designed using the best software practices and patterns on ACE in order to automate the delivery of high-performance and real-time QoS to distributed applications. *ETAO* includes a set of services such as the persistence service and transaction service. In addition, we have developed an adaptive resource management service, a context service and a notification service. Specially, the context service is based on semantic information [10]. *SCUDCCM* is conformed to CCM specification and consists of adaptive component package, assembly, deployment, and allocation at design-time. Besides, it comprises component migration, replacement, updating, and variation at run-time. In addition, the top layer is *SVA* that denotes semantic virtual agent [11]. *SVA* aims at dealing with application tasks. Each *sva* presents one service composition comprising a number of meta objects. During the co-operations of *SVA*, the SIP(Semantic Interface Protocol) [11] set is used including *sva* discovery, join, lease, and self-updating protocols. Due to the limited space, we don't detail *SVA* in this paper.

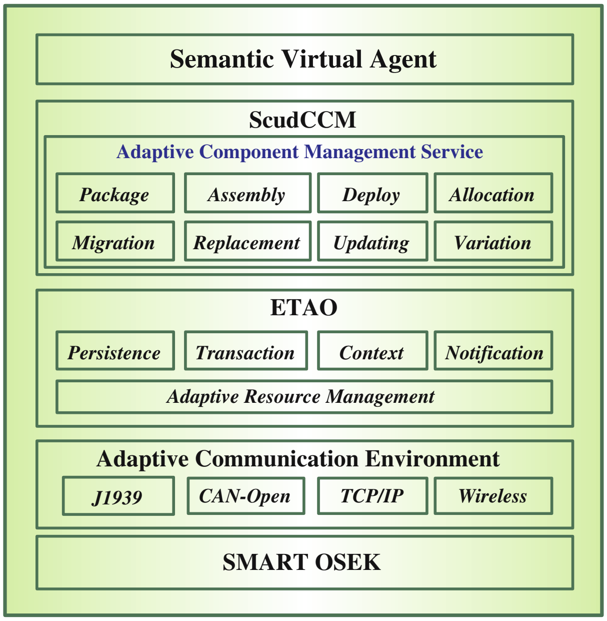

Fig. 2. ScudWare Architecture

3 Adaptive Component Management Service Framework

In this section, we describe the architecture of the adaptive component management service in a structural method. Because the component management is resource-constrained, we firstly give a resource abstract framework, and then we details this service.

3.1 Resource Abstract Framework

As Geoff Coulson [12] said, the goal of the resource abstract model is to support component adaptation. In refining this goal, two additional requirements have been identified. First, the framework must be extensible to capture diverse types of resources at different levels of abstraction, including CPU processing resources (e.g., threads, virtual processors), memory resources (e.g., ram, disk storage), communication resources (e.g., network bandwidth, transport connections), OS resources (e.g., Windows, Linux, Unix) and component container resource (e.g., CCM, EJB, .Net). Second, the framework must provide maximum control to applications according to resource adaptation.

A resource is a run-time entity that offers a service for which one needs to express a measure of quality of service. In ubiquitous computing environments, various smart devices provide amount of resources on deferent level. On the other hand, a large number of components are distributed on these devices, consuming computation resources when executing tasks. Due to the joinment and departure

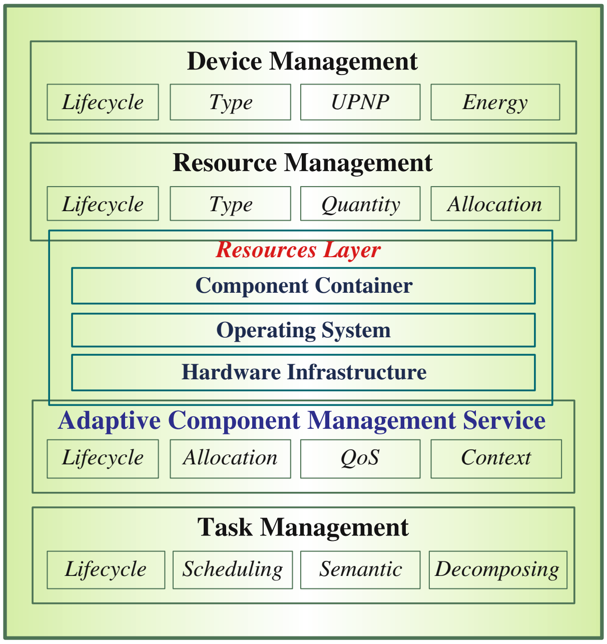

Fig. 3. Resource Abstract Framework

of the smart devices and components are dynamic, it forms a relationship between the producer and consumer based on computation resources. For instance, when a new smart device d goes into a system s, the components in s can use the resources provided by d. In addition, when a new component c enters a system s, it will be decided that how to allocate c automatically and adaptively. Specially, component c can migrate from the one device to another device.

The resource abstract framework $RAF = (DM, RM, CM, TM, PS)$ shows in Figure 3. DM is a smart device manager that monitors the device lifetime, type, and energy. In addition, it provides a mechanism for devices to UPNP (Universal Plug and Play). RM is a resource manager, administering resource lifecycle, type, quantity, and allocation. CM is an adaptive component management service, which is responsible for component lifecycle, allocation, QoS, and context management. To emphasize CM, we will give a detailed description in Section 3.2. TM is a task manager. When an application comes, TM will decompose it into several tasks based on semantic information. TM monitors tasks lifetime and schedules them in an adaptive way. Besides, PS is a set of management policy sets for these four parts, which can be well defined and reconfigured dynamically.

3.2 Adaptive Component Management Service

In terms of the resource abstract framework, we have developed an adaptive component management service. According to the different run-time contexts, this service is responsible for allocating and re-allocating the components in an appropriate way. In addition, it monitors component lifetime and is responsible for the QoS of component execution. Importantly, this service uses a run-time component hook proxy, described in Section 4.

In one component lifecycle, there are two kinds of key behaviors: component migration and component replication. These two behaviors are essential for adaptive component management. Because the components are distributed in such dynamic and discrete system, this service should adaptively take measures about when and how to migrate or replicate components.

Components are installed in different smart devices. On on hand, component migration means moving one component from one device to another device. The former device will not hold that component, and the latter device becomes the new resource carrier for that component. Emphatically stated, the latter device should have suitable resources for that component, including necessary hardware resources, OS resources, and component container resources. On the other hand, component replication means copying one component from one device to another device. In component replication, different from component migration, the former still has that component. As a result, there are two same components in two different devices. In the same way, two different devices should have same suitable resources for that component. To illustrate two behaviors, we give the cases shown in Figure 4. At first, component c_1 is distributed in smart device d_1, and d_1 also has other components such as c_2 and c_3. Assume that c_2 is exe-

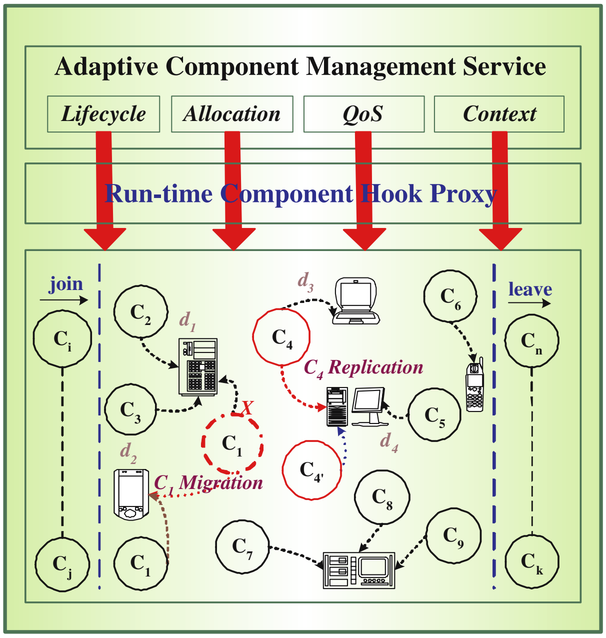

Fig. 4. Adaptive Component Management Service

cuting and occupies more hardware resources of d_1, it induces that c_1 cannot be executed for hardware resources limited when one invocation comes. Under this condition, component management service will decide to migrate c_1 to another device d_2. Next, in one appropriate time, the component migration of c_1 will take place. Following that, c_1 will execute on d_2 successfully. Here is another case of component replication. At beginning, c_4 is distributed in smart device d_3. Different form the former case, the number of invocations of c_4 is very large. For load balance, the component manger will decide to copy c_4 to another device. Assume that d_4 is satisfied with resource demands of c_4 and is not busy at that time, c'_4, the backup of c_4, will be distributed in d_4 to decrease the invocations of c_4 in d_3.

4 Run-Time Component Hook Proxy Mechanism

In the framework of adaptive component management service, we introduce a proxy mechanism called run-time component hook proxy that plays an important role in component management. This section firstly presents a component interdependence graph, and then proposes an architecture of this hook proxy.

4.1 Component Interdependence Graph

During component management, the relationships among components are very important. In order to describe the interdependent relationships among components, we introduce a component interdependence graph composed of component nodes and link paths.

For each component, we associate a node. In addition, the link paths are labelled with a weight. We define the component interdependence graph $A_{ig} = (CN, LP, W)$. (1) $CN = \{cn_i\}_{i=1..n}$ denotes a set of component nodes. (2) $LP = \{l_{i,j}\}_{i=1..n,j=1..m}$ denotes a set of component links, describing the dependent targets. $l_{i,j}$ is the link between the nodes cn_i and cn_j. (3) $W = \{w_{i,j}\}_{i=1..n,j=1..m}$ denotes a set of interdependent weight. $w_{i,j}$ is a non-negative real number, which labels $l_{i,j}$. In addition, $w_{i,j}$ reflects the importance of the interdependence between the associated components. These weights used, for instance, to detect which links becomes too heavy or if the systems rely too much on some components. In terms of this weight, we can decide which component should be allocated preferentially. Extremely, this graph changes according to the different contexts. Therefore, this interdependence is not static. It can be modified when a new component is added, or one component disappears. Moreover, based on the different application domain contexts and the run-time environments, the interdependent relationships will change.

For example, Figure 5 shows a case of the component interdependence graph. The dependence weight of component c_2 on component c_5 is 0.8, and component c_5 on component c_2 is 0.6. We call component c_2 and c_5 are mutual and direct component interdependent. Besides, we can also calculate indirect component dependent weight by decomposing each direct dependent relationship. In this case, we can conclude the indirect dependence weight of component c_1 on

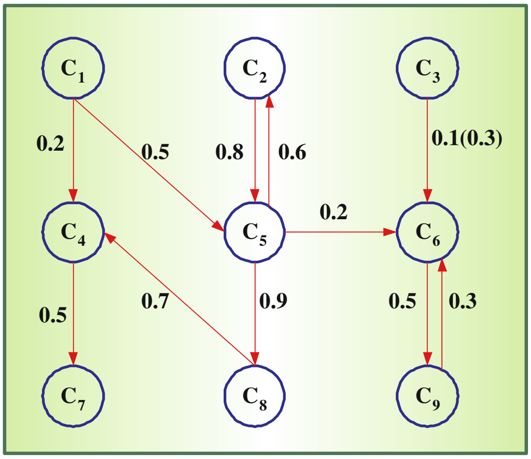

Fig. 5. A case of Component Interdependence Graph

component c_7 is a sum of weight of component c_1 on component c_4, and weight of component c_4 on component c_7. As a result, the weight is 0.7. In addition, we can also see dependence weight of component c_3 on component c_6 is 0.1 in one context, while this weight changes to 0.3 in another context. Therefore we should consider the effect of context variety on component interdependence in component allocation design.

4.2 Run-Time Component Hook Proxy Architecture

In the large and complex ubiquitous computing environments, the multiple resources are restricted. Software components are distributed, and connected with each other. They compute and communicate frequently under different conditions. As a result, they are interdependent. However, some relationships are casually and others are perpetual, which means the components do not always depend on special components for co-operations. Therefore, we use the component interdependence graph to describe run-time component self-adaptation. In executing time, the dynamic interdependence graph is generated automatically by the components hook proxy that is responsible for acquiring information to analyze and update interdependence graph to manage the components lifetime. Under the changes of the contexts, the components hook proxy uses the different strategies. Here gives a simple example. For a mobile music program, there are four components: c_1, c_2, c_3, c_4. c_1 is responsible for acquiring music information. c_2 is responsible for playing music with stereo tune. c_3 is responsible for playing music with mono tune. c_4 is responsible for outputting the music. At first, the network bandwidth is enough, the component hook proxy selects c_1, c_2, c_4 to deal with this task, and forms a component interdependence graph. However, when the component hook proxy finds the network bandwidth is scarcity, and cannot to transmit stereo track successfully, it will stop the actions of c_2,

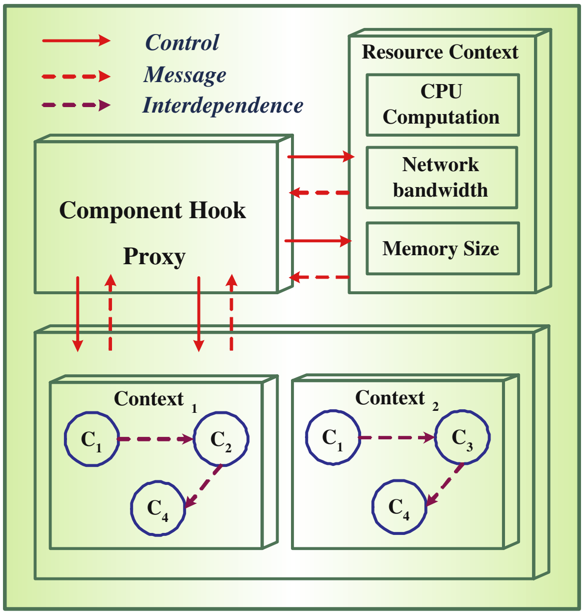

Fig. 6. A case of Component Hook Proxy

and then choose c_3 to work. As a result, the component interdependence graph changes. Figure 6 shows this case.

5 Experiment Study and Evaluation

We have made some preliminary experiments using the adaptive component management service to build the mobile music system of smart vehicle space. A large number of components are distributed on the various platforms to acquire, play, transmit, and output the music information. These components interact with the request and reply process. If one component sends the request for some music information, the component management service will select one appropriate component to work and reply to the demander. Since the context of the application is very dynamic, the strategy of component allocation should be done automatically and dynamically. Our experiments are tested on the following platforms, as shown in Table 1. The iPAQ is connected to the PC via the wireless LAN using 802.11b protocol. The middleware platform uses the ScudWare.

Mobile music system runs on some PDAs and PCs. Many components are distributed on the PDAs and PCs randomly. The functions of these components consist of acquiring the music source information, transmitting the music, and playing the music. To illustrate this, we give a case. First, the playing component c_1 on the PDA_1 is playing the music with stereo tune. When the component hook proxy finds the network bandwidth is not enough, it will stop the component c_1.

Table 1. Experiment Test Bed

	HP iPAQ Pocket PC H5500	Personal Computer
CPU	400 MHz Intel, XScal-PXA255	Intel Pentium IV 2.4G
Memory	128 MB RAM + 48 MB Flash ROM	256 MB RAM
Network	Wireless LAN 802.11b	LAN 100MB/s
OS	Familiar Linux v0.8.0-rc1	RedHat Linux 9.0 (2.4.20)
Middleware	ScudWare	ScudWare
Dev-Language	g++, QT	g++

Then the proxy finds another music playing component c_2 on the PDA_2, which plays music with mono tune and adapts to low network bandwidth. Because the current playing frame is No. 168, it will start the component c_2, and play music from the No. 168 frame with the mono tune. In this way, the system can continue successfully without more delays and provide comparative satisfaction for users.

In order to test the performance of the adaptive component management service, we have made many simulations and evaluations. The results show that our method is flexible and has little negative influence to the systems.

Due to adaptive component migration and replication, it must induce the execution performance cost through component manager monitoring. As a result, we focus on the performance test for measuring the cost. Because components in ubiquitous computing environments form a large, complex and rich world, the number of components in the tests is a key issue. In our tests, we choose the component number n in this way: the first value is 50, the last is 500, and the

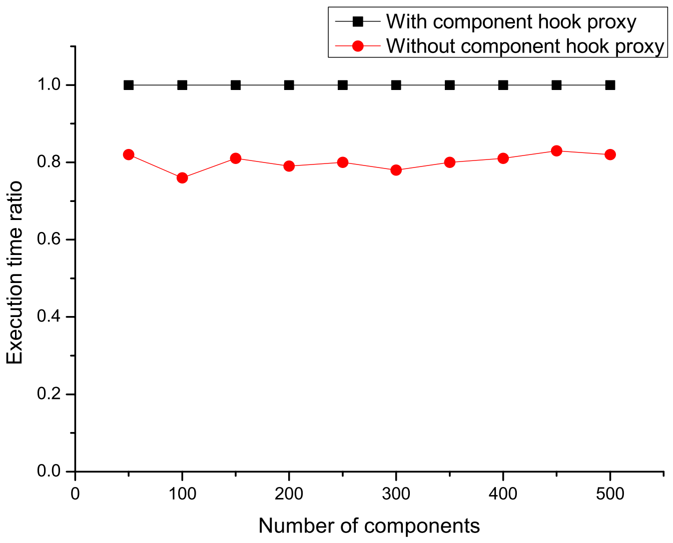

Fig. 7. Performance Cost

step is 50. We have done the experiments 10 times, and each time we use $n/25$ PCs and $n/50$ iPAQs. As shown in Figure 7, the average execution time for each n is given about two kinds: one is a mobile music system without the component hook proxy monitoring, and the other is with it. As a whole, the difference is small and the execution time is acceptable.

6 Related Work

Service-oriented adaptive middleware plays an important role in software engineering. It has the large potential for enhancing the system's flexibility and reliability to a very wide range of factors. Many efforts are put in this research area. For instance, Philip K. Mckinley has made a lot of research on adaptive software. He considers the compositional adaptation enables software to modify its structure and behavior dynamically in response to changes in its execution environment. He also gives a review of current technology comparing how, when, and where re-composition occurs [13]. In addition, he describes Petrimorph [14], a system that supports compositional adaptation of both functional and non-functional concerns by explicitly addressing collateral change. Kurt Wallnau and Judith Stafford [15] discuss and illustrate the fundamental affinity between software architecture and component technology. They mainly outline criteria for the component integration. Jiri Adamek and Frantisek Plasil [16] discuss the problem of defining a composition operator in behavior protocols in a way, which would reflect false communication of the software components being composed. Besides, we have proposed a semantic and adaptive middleware for data management in smart vehicle space [2]. In component adaptation, we should consider both the task decomposing completely at design-time and the executing effectively and reliably at run-time [17]. However, many researches consider incompletely, ignoring some aspects. Additionally, it needs an integrated computation model to describe adaptive component management. The goal of our research is to overcome this deficiency. Smita Bakshi and Daniel D. Gajski [18] present a cost-optimized algorithm for selecting components and pipelining a data flow graph, given a multiple implementation library. This method focuses on performance analysis, which is short of the run-time adaptive mechanism. Belaramani and Cho Li Wang [3] propose a dynamic component composition approach for achieving functionality adaptation and demonstrate its feasibility via the facet model. However, they do not integrate the design-time adaptation to form a synthetical computation model. Shige Wang and Kang G. Shin [19] give a new method for component allocation using an informed branch-and-bound and forward checking mechanism subject to a combination of resource constraints. Nevertheless, their method is static, and focuses on design-time instead of runtime adaptation.

7 Conclusions and Future Work

Now, adaptive component management is playing a more important role in ubiquitous computing environments, which is a significant research issue. In this paper, we firstly analyze the problem area caused by dynamic characters of ubiquitous computing, and give a short introduction of the software adaptation. Next, we mainly present an adaptive component management service, which is integrated into the ScudWare middleware. In addition, we have made a large number of experiments to test the performance cost of this service.

Our future work is to improve the adaptive component management service including some algorithm analysis. In addition, we will take other methods to realize more component management flexibility and reliability both at design-time and run-time.

Acknowledgments

This research was supported by 863 National High Technology Program under Grant No. 2003AA1Z2080, 2003AA1Z2140 and 2002AA1Z2308.

References

1. Weiser M: The Computer for the 21st Century. Scientific American, pp.94-100 (1991)
2. Qing Wu, Zhaohui Wu, Bin Wu, and Zhou Jiang: Semantic and Adaptive Middleware for Data management in Smart Vehicle Space. In proceedings of the 5th Advances in Web-Age Information Management, LNCS 3129, pp. 107-116 (2004)
3. Nalini Moti Belaramani, Cho-Li Wang, and Francis C.M. Lau: Dynamic Component Composition for Functionality Adaptation in Pervasive Environments. In proceedings of the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems, (2003)
4. Zhaohui Wu, Qing Wu, Jie Sun, Zhigang Gao, Bin Wu, and Mingde Zhao: ScudWare: A Context-aware and Lightweight Middleware for Smart Vehicle Space. In proceedings of the 1st International Conference on Embedded Software and System, LNCS 3605, pp. 266-273 (2004)
5. http://www.omg.org/technology/documents/formal/components.htm (2005)
6. http://www.cs.wustl.edu/ schmidt/ACE.html (2005)
7. http://www.cs.wustl.edu/ schmidt/TAO.html (2005)
8. Mingde Zhao, Zhaohui Wu, Guoqing Yang, Lei Wang, and Wei Chen: SmartOSEK: A Dependable Platform for Automobile Electronics. In proceedings of the first International Conference on Embedded Software and System, LNCS 3605, pp. 437-442 (2004)
9. OSEK/VDX: OSEK/VDX Operating System Specification Version 2.2.2. http://www.osek-vdx.org (2005)
10. Qing Wu and Zhaohui Wu: Integrating Semantic Context Service into Adaptive Middleware for Ubiquitous Computing. In "Advances in Computer Science and Engineering Series", Imperial College Press, London, UK, to appeare (2005)
11. Qing Wu and Zhaohui Wu: Semantic and Virtual Agents in Adaptive Middleware Architecture for Smart Vehicle Space. In proceedings of the 4th International Central and Eastern European Conference on Multi-Agent Systems, LNAI 3690, pp. 543-546 (2005)
12. Hector A. Duran-Limon, Gordon S. Blair, Geoff Coulson: Adaptive Resource Management in Middleware : A Survey. IEEE Distributed System 5(7), (2004)

13. Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, Betty H.C. Cheng: Comosing Adaptive Software. IEEE Computer Society, pp. 56-64 (2004)
14. E.P. Kasten, P.K.McKinley: Perimorph: Run-time Composition and State Management for Adaptive Systems. In proceedings of the 4th International Workshop on Distributed Auto-adaptive and Reconfigurable Systems, pp. 332-337 (2004)
15. Kurt Wallnau, Judith Stafford, Scott Hissam, Mark Klein: On the Relationship of Software Architecture to Software Component Technology. In proceedings of the 6th International Workshop on Component-Oriented Programming (2001)
16. Jiri Adamek, Frantisek Plasil: Component Composition Errors and Update Atomicity : Static Analysis. Journal of Software Maintenance and Evolution: Research and Practice (2005)
17. Qing Wu and Zhaohui Wu: Adaptive Component Allocation in ScudWare Middleware for Ubiquitous Computing. In proceedings of the 2005 IFIP International Conference on Embedded And Ubiquitous Computing, LNCS, to appeare (2005)
18. Smita Bakshi, Daniel D.Gajski:A component Selection Algorithm for High-Performance Pipelines. In proceedings of the conference on European design automation, ACM, pp. 400-405 (1994)
19. Shige Wang, Jeffrey R. Merrick, Kang G. Shin: Component Allocation with Multiple Resource Constraints for Large Embedded Real-time Software Design. In proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (2004)