Some Software Design Issues for Realizing Internet-Scale Ubiquitous Computing

Hiro Ishikawa, Tatsuo Nakajima
Department of Information and Computer Science
Waseda University
3-4-1 Okubo Shinjuku Tokyo 169-8555 JAPAN
ishikaw@dcl.info.waseda.ac.jp

Abstract

In the near future, we will carry a variety of mobile and wearable appliances that are connected to the Internet. We require a lot of middleware components for integrating these appliances to make it easy to develop advanced future applications.

However, building such components is very hard, and needs to consider a lot of issues. In this paper, we describe several software design issues to develop complex middleware components for realizing *Internet-scale ubiquitous computing environments* that enable various appliances to be integrated on the Internet.

This paper first presents some design issues for building Internet-scale ubiquitous computing environments. Then, our plan to develop a new framework for building software for Internet-scale ubiquitous computing is described.

1 Introduction

Our daily life will be dramatically changed since computers will be embedded in various objects surrounding us such as shoes and pens. The objects will become intelligent and augment our daily life by monitoring our behaviors[5]. Also, traditional appliances such as TV, microwaves, and refrigerators will become more intelligent and communicate each other. Also, we carry various mobile wearable appliances that are connected to the Internet.

Recently, we are carrying a lot of personal appliances with us such as cellular phones, PDAs and MP3 players. In the near future, we will have a lot of home appliances in our houses. Also, various appliances in cars, trains, and public spaces will be appeared. These appliances will be connected to the Internet soon, and it is possible to share information from various appliances.

Ubiquitous computing will add several additional functionalities to these appliances. For example, ubiquitous computing applications need to change their behaviors according to their current situations such as location information, a user's emotion, and preference[1, 18]. Also, ubiquitous computing applications provide new interaction techniques[19, 7]. A variety of sensors monitor our behaviors, and a model that represents our real world in computers provides better context-awareness[3, 6].

In *Internet-scale ubiquitous computing environments*, the most important issue that needs to be taken into account is extreme heterogeneity. For example, future appliances will run on various types of processors, and they will have a variety of I/O devices. Also, there are connected by various types of networks and protocols. Therefore, it is necessary to take into account extreme heterogeneity when designing software for *Internet-scale ubiquitous computing environments*.

In these environments, one of the most important issues is to develop software at low cost. Thus, it is desirable to develop software that is portable for various platforms. Also, it is important to reuse existing software on various appliances. Especially, it is very expensive to develop platform software such as operating systems and middleware components, then our aim is to increase the portability of platform software. Also, COTS(Commercial Off The Shelf) software components such as Linux, Java, and CORBA should be used for building various ubiquitous computing applications because these COTS software components enable us to use a lot of existing software. For example, it is easy to find various software components written in Java such as XML parsers and CORBA runtimes. Also, there are a lot of Internet middleware components on Linux. These standard platform software components should be used in Internet-scale ubiquitous computing environments because using the standard interface makes an educational cost cheap, it is easy to develop a huge amount of software contributed from various communities, and to reuse various commer-

cial services and appliances.

This paper first presents some design issues for building Internet-scale ubiquitous computing. Then, our plan to develop a new framework for building software for Internet-scale ubiquitous computing is described.

2 Requirements for Realizing Internet-Scale Ubiquitous Computing

There are several requirements to realize Internet-scale ubiquitous computing environments. The following requirements are considered as architectural guidelines to build the environments.

- Extremely Portable
- Uniform Behavior
- High Level Abstraction
- Survival Systems

The software for Internet-scale ubiquitous computing environments should be extremely portable because the environments are extremely heterogeneous, but we do not expect to implement the software on every platform. However, the software should be optimized by using the characteristics of applications and platforms. For example, the resource management of a middleware component should be customized by using the requirements of application programs.

The environments should allow a user to behave in a uniform way when he moves to any spaces such as his/her house, an airport, and an office. For example, a user should control a television in the same way, in any places such as his/her house or an airport. The uniform behavior is very important to design the user interface for Internet-scale ubiquitous computing.

Also, the system software for building Internet-scale ubiquitous computing environments should provide high level abstraction to make us to build application program easily. The existence of high level models is very important to build the application programs in a systematic way since the lack of right abstraction makes the structure of the application programs ad-hoc.

Lastly, Internet-scale ubiquitous computing environments should survive against security attacks, system crashes, and natural disaster since our daily life will heavily rely on the environments. However, it is very difficult that a usual user takes into account survivability issues when implementing programs. Therefore, it is important to develop a methodology to convert a program that does not take into account survivability to survival software automatically.

3 Software Design Issues

In this section, we describe two software design issues to develop software for Internet-scale ubiquitous computing environments.

3.1 Transparent QOS Evolution

As describe in the previous section, we need to reuse a variety of COTS software components to realize Internet-scale ubiquitous computing environments. However, these components must take into account various properties such as security, reliability, and predictability, but usual programmers do not know what are real problems to build survival software. Therefore, these properties may be sometimes ignored in COTS software components.

One of solutions is to add these properties into COTS software components by post hoc. This means that a COTS component is transparently translated to a component that takes into account non functional properties such as security, reliability and predictability. The solution may not provide these properties completely, but the approach improves the quality of software drastically.

We are considering several approaches to realize the goal. One approach is to use aspect-oriented programming(AOP) techniques[2, 9, 17] to add these non functional properties. When using AOP techniques, these properties are defined as aspects, and the aspects are merged into base COTS software components at runtime. Therefore, this enables the COTS software components to be adapted according to the characteristics of underlying platforms. Also, there are several proposals to translate Java binary codes that are used for adding these properties by post hoc[4].

However, we have to take into account several issues to overcome the problems of current proposals. The first issue is whether the QOS evolution can be achieved transparently from a client program. If some assumptions of the components are changed by adding these features, the correctness of a client application may be violated. We need to define rigorous API semantics for middleware components, and need to check the assumptions when adding the features. The second issue is that current AOP techniques require to understand the internal structure of the base COTS components to define aspects. However, it is not easy to understand complex COTS software components such as Linux, Java or CORBA. It is important to export high level abstract structure of the components to define various aspects.

3.2 Portability Issues

In Internet-scale ubiquitous computing environments, we need to take into account a variety of platforms. If

we like to use COTS software components, it is important to consider how to exploit advanced characteristics provided by these underlying platforms. This requires to add meta level interface[10] or QOS parameters to control the internal algorithms of the COTS software components. However, it is not easy to export generic interface to control underlying platforms because the generic interface usually hides some low level characteristics of the underlying platforms. We believe that it is desirable that the interface should be customized to respective underlying platforms for enabling us to use the full power of the platforms. Our research aims are how to provide platform specific meta interface or QOS parameters in a systematic way, and how to build portable applications that access the platform specific meta interface and QOS parameters. We are considering to exploit AOP techniques and design patterns to implement COTS software components.

When building software, we need to take into account various tradeoffs among many metrics. For example, a programmer needs to consider several metrics such as timeliness, precision, accuracy, and consistency to build mobile applications. It is impossible to satisfy all requirements so he/she must consider which requirements we need to focus on, and the decision affects the program's structure dramatically.

For example, distributed applications should take into account three metrics: consistency, availability, and network partition. If we like to improve the application available, we need to select either consistency and network partition. If we need to assume that network partition occurs, it is impossible to ensure complete consistency. Therefore, it is desirable to adopt optimistic protocols to satisfy consistency. On the other hand, if we require complete consistency, a system should not assume network partition.

Building portable software requires to make explicit its assumptions because the assumptions are necessary to ensure the correctness of a program. Combining several components that provide different assumptions to their client programs allows us to use the combined component as a component that supports a wide range of assumptions. This requires to switch components when the assumption is changed. However, the duration to change the component may cause inconsistency, so it is important that the change is executed atomically.

4 A New Framework for Building Internet-Scale Ubiquitous Computing Programs

In this section, we propose a new framework for building Internet-scale ubiquitous computing programs. The goal of the framework is to exploit the characteristics of underlying platforms without sacrificing the portability of a program. In this section, we introduce our current project towards the goal. In the framework, we assume to adopt Java to write programs.

4.1 Why Real-Time?

Our current project focuses on how real-time aspects can be added to existing non real-time programs. The reasons to choose real-time issues is that we have a lot of experiences with real-time computing[12, 13, 14, 15, 16]. Therefore, we will be able to apply our experiences to our new project. Also, adding real-time aspects offers several challenging problems. For example, real-time programs require careful structuring of programs for ensuring their timing constraints, and various QOS issues can be considered as useful examples for future computer science researches. Finally, Internet-scale ubiquitous computing environments require various resource constraints that have been discussed in real-time research communities.

4.2 Real-Time Programming in Java

Recently, the Java programming language becomes popular for building complex software. Internet-scale ubiquitous computing environments need to build a lot of complex software so Java seems to be a practical choice to build Internet-scale ubiquitous environments because Java provides automatic memory management, object-orientation, and multithreading that are desirable to build large-scale software.

We believe that providing real-time properties is a very important issue for building Internet-scale ubiquitous computing environment. For example, a variety of devices in embedded systems such as multimedia devices and mobile devices require to satisfy real-time properties. Multimedia programs such as MPEG video players should change video frames without violating their timing constraints.

There are two issues that should be taken into account when we consider real-time constraints. The first issue is that respective real-time applications require different resource management strategies. The most important experiences from our previous work is that there are many approaches to satisfy resource constraint problems, and respective solutions require to offer suitable API for respective application programs. Thus, it is necessary to provide different APIs for respective resource management strategies. Moreover, it is important to export the characteristics of underlying platforms directly because abstracting them may hide the real power of the underlying platforms.

The second issue is that a middleware component such

as Java provides standard interface to application programs to ensure portability. For example, JVM hides real-time properties provided by operating systems even if the operating systems provide the real-time resource management. Thus, it is important that middleware should not hide the useful characteristics of the underlying platforms as shown in Figure 1.

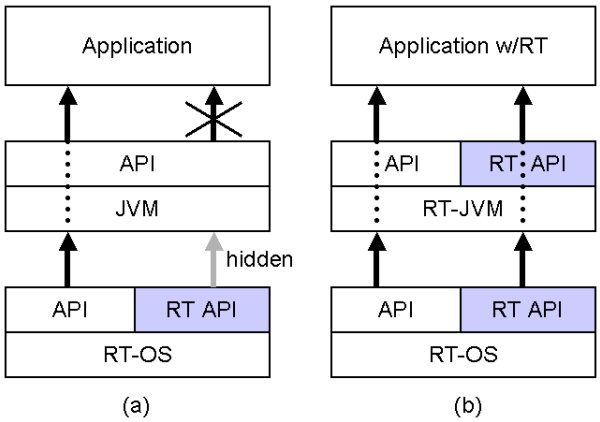

Figure 1: (a)Application without RT. (b)Application with RT.

Therefore, we believe that standardizing real-time API for Java is not useful for Internet-scale ubiquitous computing applications. Especially, it is not easy to standardize complex QOS specifications required for various applications. Recently, RTSJ[21] has been published to add API for programming real-time applications in Java. However, the specification contains traditional real-time supports that are not enough to build Internet-scale ubiquitous computing applications.

We believe that it is important to provide real-time API that is suitable for each application. The real-time API is customized for each application to take into account the requirements for its resource management strategy. However, as described in the previous section, it is necessary not to degrade its portability.

4.3 Design Issues

Adding real-time properties to existing non real-time programs require program translation. This section presents the following three approaches to realize the transparent modification of existing programs.

- Recompiling after Source Code Modification.
- Bytecode Translation.
- Just-in-time Compilation.

A traditional way to modify existing programs is to translate source codes, and recompile them before their execution. Additional codes are merged to existing programs at the source code level. The approach has the following two problems. The first problem is requiring to obtain source codes to add new features. Usually, it may not be access source codes of commercial programs. The second problem is that recompiling large software takes very long time. Therefore, the overhead to modify programs may cause a serious problem.

Bytecode translation is able to modify programs without obtaining source codes. Therefore, program translation does not need recompilation.

For example, Java bytecode translators modify bytecodes when loading class files into Java virtual machines. There are many techniques to modify bytecodes, but it is important to provide high level abstraction to modify the bytecodes for making it easy to add non functional properties.

Lastly, JIT, Just-in-time compilation, is similar to the bytecode translation. The difference between these two approaches is that the approach to adopt JIT does not require to modify existing Java virtual machines because a JIT compiler generates a machine code that satisfies real-time requirements automatically, and the code is directly executing on a processor without the help of virtual machines.

We believe that the byte code translation is appropriate for building Internet-scale ubiquitous computing applications because source codes may not be obtained usually, and JIT cannot be used for resource constrained small devices.

4.4 Our Approach

In this section, we describe our approach to build Internet-scale ubiquitous computing applications. To build extremely portable applications, we first write a program in a very generic way. Then, additional properties such as real-time aspects will be added at run-time by using Javassist[22] that is a bytecode translator providing high level reflection interface. Figure 2 shows the overview architecture of our approach.

The approach has the following characteristics.

- Separating a portion of timing constraint management as an aspect.
- The aspect may invoke a special API for achieving special requirements on respective platforms. Therefore, the aspect may be rewritten for respective platforms.
- The aspect is merged at run-time by Javassist.

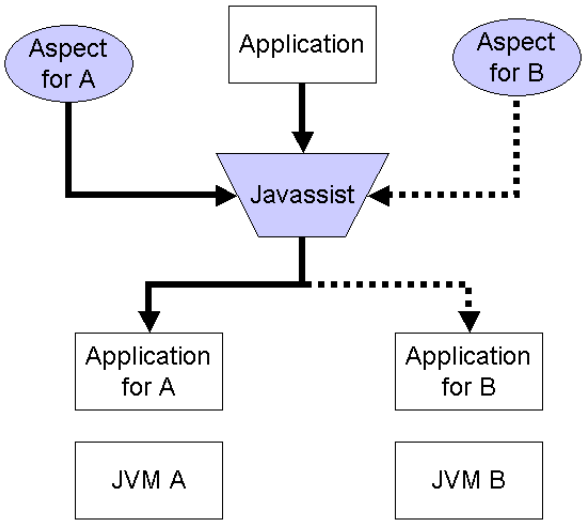

Figure 2: Our Prototype

The advantage of our framework allows us to adopt ad-hoc extensions provided by underlying platforms without sacrificing the portability of an application program. If a user likes to port an application program, only aspect parts should be rewritten to take into account platform specific characteristics. Currently, we have just started our project, and the design of our framework has not been completed.

In embedded system communities, System-on-Chip(SoC) is a hot topic. In the future, very small intelligent devices[8] will be embedded in our world. However, we already have the large amount of software that is written on COTS software. The COTS software provides standard API, and does not take into account the characteristics of the new devices. Therefore, our approach is very effective for building future embedded systems to exploit platform specific characteristics.

Also, in the future, we like to work on other aspects such as security in our framework because providing security requires very careful programming to avoid security holes, but it is not easy to hire expert programmers for security to implement future Internet-scale ubiquitous computing applications.

5 Conclusion

In this paper, we have described several software design issues to develop middleware components to realize Internet-scale ubiquitous computing environments. The first issue is that programs running in the environments should take into account extreme heterogeneity. The second issue is that these programs should take into account survivability. Also, we have described a new framework for building Internet-scale ubiquitous computing environments.

References

[1] G.D. Abowd, E.D. Mynatt, "Charting Past, Present, and Future Research in Ubiquitous Computing", ACM Transaction on Computer-Human Interaction, 2000.

[2] M. Aksit, B. Tekinerdogan, "Aspect-Oriented Programming Using Composition-Filters", Position Paper for the Aspect Oriented Programming Workshop, Springer-Verlag, LNCS1543, 1998.

[3] B. Brumitt, J. Krumm, B. Meyers, S. Shafer, "Ubiquitous Computing and the Role of Geometry". IEEE Personal Communications, August 2000.

[4] Shigeru Chiba, "Load-time Structural Reflection in Java", In Proceedings of ECOOP 2000 – Object-Oriented Programming, LNCS 1850, Springer Verlag, page 313-336, 2000.

[5] N. Gershenfeld, "When Things Start to Think", Owl Books, 2000.

[6] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, Paul Webster, "The Anatomy of a Context-Aware Application", In Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, 1999.

[7] H. Ishii, B.Ullmer, "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms", In Proceedings of Conference on Human Factors in Computing Systems,1997.

[8] J. M. Kahn, R. H. Katz and K. S. J. Pister, "Mobile Networking for Smart Dust", ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99), 1999.

[9] G. Kiczales, et. al., "Aspect Oriented Programming", In Proceedings of the European Conference on Object-Oriented Programming, Springer-Verlag, 1997.

[10] G.Kiczales, J.Lamping, C.V.Lopes, C.Maeda, A.Mendhekar, G.Murphy, "Open Implementation Design Guidelines ", In proceedings of the 19th International Conference on Software Engineering (ICSE), 1997.

[11] T.Nakajima, "Technical Challenges for Building Internet-Scale Ubiquitous Computing", In Preparation.

[12] T.Nakajima, T.Kitayama, H.Tokuda, "Experiments with Real-Time Servers in Real-Time Mach", USENIX 3rd Mach Symposium, 1993.

[13] T.Nakajima, et. al., "Integrated Management of Priority Inversion in Real-Time Mach", IEEE Real-Time Systems Symposium, 1993.

[14] T.Nakajima, H.Tezuka, A Continuous Media Application supporting Dynamic QOS Control on Real-Time Mach, ACM Multimedia'94, 1994.

[15] T.Nakajima, H.Tokuda, "User-level Real-Time Network System on Microkernel-based Operating Systems", Kluwer Real-Time Systems Journal Vol.14 No.1, 1998.

[16] Tatsuo Nakajima, "Practical Explicit Binding Interface for supporting Multiple Transport Protocols in a CORBA System", In Proceedsing of IEEE International Conference on Network Protocols, 2000.

[17] H. Ossher, P.L. Tarr, "Hyper/J: Multi-Dimensional Separation of Concerns for Java", In Proceedings of the International Conference on Software Engineering, 2000.

[18] R. Picard, "Affective Computing", The MIT Press, 1997.

[19] J.Rekimoto and M.Saitoh, "Augmented Surfaces: A Spatially Continuous Workspace for Hybrid Computing Environments", Proceedings of CHI'99, 1999.

[20] Mark Weiser, "The Computer for the 21st Century", Scientific American, Vol. 265, No.3, 1991.

[21] G.Bollella, B.Brosgol, S.Furr, D.Hardin, P.Dibble, J.Gosling, M.Turnbull, "The Real-Time Specification for Java", Addison-Wesley, 2000.

[22] S.Chiba, "Javassist–A Reflection-based Programming Wizard for Java", In Proceedings of the ACM OOPSLA'98 Workshop on Reflective Programming in C++ and Java, 1998.