Explicit Language and Infrastructure Support for Context-aware Services

Malte Appeltauer and Robert Hirschfeld
Hasso-Plattner-Institut
Universität Potsdam, Germany
{malte.appeltauer, hirschfeld}@hpi.uni-potsdam.de

Abstract: Dealing with context information is an important concern for mobile service computing. However, modern frameworks and programming languages do not directly support context-dependent behavior with first class entities. In this paper, we present Context-oriented Programming for mobile applications and provide an overview of our context-oriented extension to the Java programming language.

1 Introduction

Modern mobile computing technologies open new fields for applications that consider the user's context information. Typically, these applications are implemented in a service-oriented approach and composed by services that provide context-dependent functionality. For instance, location-based services are aware of the user's position, health-and-fitness services monitor speed and heartbeat, while mood-based services

take care of personal dispositions.

The development of context-aware systems often requires programmers to consider an additional dimension in the software model. At programming language level, *if* conditions check the presence of a certain context and tangle the code with the new behavior instead of representing context-dependent behavior explicitly. Without appropriate means for representation, development, and evolution, context-sensitive programs are getting more complex for maintenance and further development. We propose an integral approach to the design, development, maintenance, and evolution of context-dependent systems. *Context-oriented Programming* (COP) [HCN08] enriches programming languages and execution environments with features to explicitly represent context-dependent behavioral variations.

In this paper, we employ COP for the development of context-aware services. Section 2 motivates and introduces context-aware services and presents a scenario for a context-dependent mobile application. Section 3 describes the context-oriented programming paradigm and gives a brief introduction to a context-oriented extension to the Java programming language. An example for the combination of COP and service-oriented computing is shown afterwards. Section 4 presents related work, while the last section concludes the paper.

2 Development of Context-aware Services

Context-aware Services are mostly implemented based on context-management systems. These systems process context-reasoning - e.g., based on ontologies - and announce context changes to applications. We focus on the implementation of such systems and discuss implementation and evolution issues of context-aware services in Section 2.2.

Section 2.1 exemplifies a service-based mobile application. As we want to investigate the interaction of service and context information, we describe a highly context-dependent scenario. The application not only gathers context information for implementation purposes, but also informs a community about context changes of users.

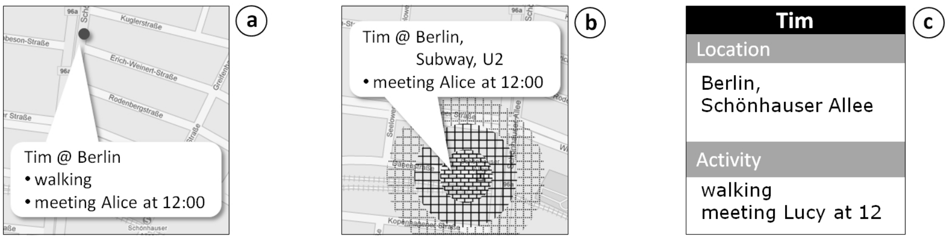

Figure 1: Lucy's application renders Tim's activity depending on Tim's location provider and Lucy's connectivity. (a) GPS localization; (b) Localization via phone cell IDs; (c) No connection to a map service.

2.1 Scenario

This subsection describes a scenario for a mobile community application. The client software allows users to observe the current activity of their friends.

Lucy and Tim have an appointment at Hasso-Plattner-Institut (HPI) at 12:00. When Lucy arrives at the HPI campus at 11:50, she checks the client software on her mobile phone to see Tim's whereabout. The application shows Tim's location in Berlin on a map.

As Tim enters the subway station, GPS data cannot be received anymore. Thus, the mobile device switches to cell-based location detection and now provides just vague location data based on the current mobile cell. Tim's change of the location provider propagated to Lucy's device. Thus, Tim's position is not shown as a point on a map anymore but as a circular area with different color intensities indicating the probability of Tim's real position (see Figure 1).

While checking Tim's whereabout, Lucy's phone battery lifetime decreases to 30%. To save energy, the system closes the Internet connection and stops the GPS device. Instead, context information is exchanged by SMS at regular intervals. The service responsible for context data exchange has to re-render its data for the SMS format while Tim still receives date in the default format.

2.2 Shortcomings of Context Representation

The community software that we describe in our scenario, can be implemented with common Web service and mobile application techniques. In fact, some mobile applications provide functionality that is similar to our example system (e.g., [KTL^{+}06]). Typically, context-dependent functionality requires system adaptations at several points. Figure 2 sketches some modules of our example application. The implementation of the *GPS* and *LowBattery* functionality is scattered over the client application and a service module. Because these *cross-cutting concerns* require adaptations of the modules at several points, they cannot be completely modularized with pure object-oriented techniques. Instead, they tangle other modules with additional functionality. For instance, *if*-conditions are injected, that care for behavioral change depending on context information. The more variations a system has, the harder it becomes for developers to maintain the source code. The lack of an explicit representation of behavioral variations hinders software development and evolution.

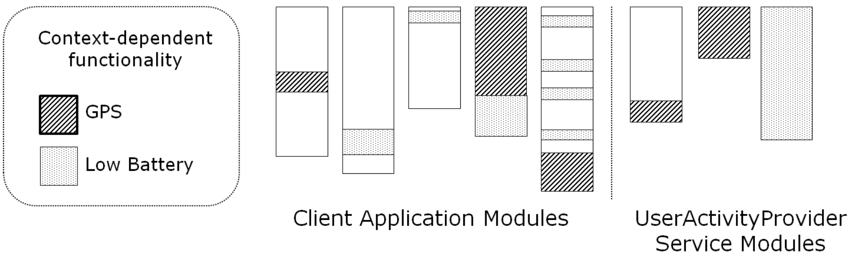

Figure 2: Context-dependent behavior is scattered over the system and is tangled with module core functionality.

3 Context-oriented Programming

Context-oriented programming [HCN08] is a paradigm supporting the development of behavioral variations that depend on execution context. COP languages provide an explicit representation of context-dependent behavioral variations as first-class program entities. Thus, COP features a new dimension of software modularization. With dynamic composition of behavioral variations, software evolution can become more flexible and accessible.

A program's context could be everything within its execution and environment, such as personalization, sharing, location-awareness, activity, and more. Context change can trigger behavioral variation. A broad introduction to COP is provided by [HCN08].

COP has been successfully implemented in the form of several language extensions, such as for Lisp, Smalltalk, Python, and Ruby. In the following section, we sketch ContextJ, our COP extension for Java.

3.1 ContextJ

For the implementation of our scenario, we want to benefit from the abilities of dynamic layer activation. We chose Java as base language mainly for two reasons: Firstly, Java is a widely accepted language proven many times to be a stable and secure platform for service-oriented programming. Secondly, Java would be the first statically typed programming language to be extended with context-oriented features. We investigate, how the dynamic properties of COP are feasible for statically typed languages. The main language features are described in the following:

Layer. Java is extended with a new block construct called `layer` which can contain partial class or method declarations. Next to new composition means, layers are code structures that make behavioral variations explicit and combine them in a dedicated modularity construct.

Partial method declarations. A partial method declaration contains instructions which are being executed (while the layer is active) before, after or instead of the original method instructions. This feature is also implemented by other languages such as Common Lisp and AspectJ.

Layer activation. Layers can be activated and deactivated, depending on the current context at run-time. While activated, method calls are dispatched to the layered methods. The construct `with(Layer)` controls activation.

Figure 3 shows two classes belonging to the application introduced in Section 2.1. To extend our application with the *LowBattery* concern of our scenario, we need to adapt our system both on the client and server sides. For instance, `UserActivityProvider` has to reduce or compress data since SMS are length-restricted. Among other adaptations, the client application changes user representation. Thus, the `renderUser(User)` method needs to be rewritten. The figure represents a layer containing partial method declarations. This layer will be activated by a `with` command when battery power decreases below 30%.

The development of ContextJ is work in progress. A first prototype of ContextJ, ContextJ* [HCN08], has been developed previously. ContextJ*, a plain Java library, covers a limited set of COP features implemented as a library.

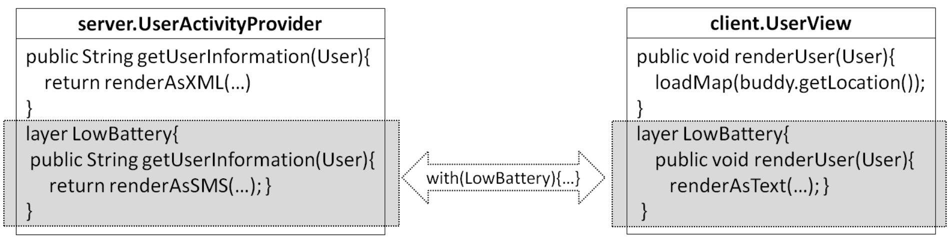

Figure 3: Behavioral variation of a service implemented with ContextJ

3.2 A ContextJ-based Infrastructure for Context-aware Services

To validate the use of ContextJ for service development, we are setting up an infrastructure to implement an application like the one described in Section 2.1. Such an application contains services written in a context-oriented language supported by a community platform and a mobile client which is able to announce context change. We evaluate *IYOUIT* as service platform and *Android* as client system.

IYOUIT [KTL+06] is a community platform for mobile users. It manages users' buddy lists and propagates their context information to friends. Users are informed about their buddies activity, location, mood, and more. It is possible to create new context data, such as local weather information or blog entries and photos that are tagged with the author's context data. For the implementation of our scenario, we extend the IYOUIT system with ContextJ-based services. Later on, we will additionally implement services with ContextS and the Smalltalk based Seaside [PTB+08] web framework. With this heterogeneous infrastructure we will evaluate COP.

Android [Ope] is an open-source platform for mobile devices based on the Linux kernel. For application development, Android provides a Java-based framework. Android directly supports location-based services such as GPS but does not come with a context-management system. Since IYOUIT only supports a client software for Nokia S60 cell phones, we develop an IYOUIT client for Android.

4 Related Work

4.1 Context-Management Frameworks

Most context-aware applications are endured with a *context-management framework* (CMF) which supplies applications with context information. IYOUIT is one example for CMF-based applications.

Another representative is the mobile tourist application *COMPASS* [vSPK04]. It provides users with context-dependent points of interests. Application modules can subscribe to a notification manager to be announced whenever context changes. The application then has to care for behavioral variations depending on the new context.

Context-management frameworks provide a representation of context and announce context change to applications. However, they do not support modularization of the crosscutting context-dependent concerns for application development. We will investigate, if the combination of context-management at application level and context-oriented programming at language level could ease the development of these systems.

4.2 Distributed Aspect-oriented Programming

Aspect-oriented Programming (AOP) provides means to encapsulate cross-cutting concerns. Some languages and systems support distributed AOP, such as *AWED* [NSV+06] and *ReflexD* [TT06]. Distributed AOP can be used to encapsulate cross-cutting context-specific concerns. However, common pointcut models have no explicit means for the representation of context-dependent functionality. For a feature comparison of distributed AOP approaches, see [DBC07].

CSLogicAJ [RSC06] features context-sensitive service aspects to adapt service behavior. The extensible join point model supports context change. Special pointcuts can be used to describe context. An asynchronous advice type executes service adaptation triggered by these context-aware pointcuts. The service interception and adaptation is done by Ditrios, an OSGi-based middleware. In the current version, CSLogicAJ is restricted to local OSGi service bundles. Server-side services can not be intercepted.

AOP systems support the encapsulation of cross-cutting concerns such as context-specific behavior. However, they isolate context-dependent functionality from the base program into an aspect structure. In contrast, COP integrates it features with the existing ones.

5 Conclusion

Although context-awareness is handled by context-management systems at application level, it is poorly supported by programming languages. The implementation of context-dependent concerns tangles application and service modules and hinders software maintenance and evolution.

In this paper, we propose to apply COP to support the development of context-aware services. With COP languages, such as ContextJ, context-awareness is handled consistently in the entire service development process.

Acknowledgments

The authors thank Jens Lincke, Michael Perscheid, and Michael Haupt for comments on drafts of this paper.

References

[DBC07] Francisco Dantas, Thais Batista, and Nelio Cacho. Towards Aspect-Oriented Programming for Context-Aware Systems: A Comparative Study. In *SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, page 4, Washington, DC, USA, 2007.

IEEE Computer Society.

[HCN08] Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. Context-oriented Programming. *Journal of Object Technology*, 7(3):125–151, March-April 2008.

[KTL+06] Johan Koolwaaij, Anthony Tarlano, Marko Luther, Petteri Nurmi, Bernd Mrohs, Agathe Battestini, and Raju Vaidya. Context Watcher – Sharing context information in everyday life. In J.T. Yao, editor, *Web Technologies, Applications, and Services*, Calgary, Canada, July 17-18 2006. ACTA Press, Proceedings of The IASTED International Conference on Web Technologies, Applications, and services.

[NSV+06] Luis Daniel Benavides Navarro, Mario Südholt, Wim Vanderperren, Bruno De Fraine, and Davy Suvée. Explicitly distributed AOP using AWED. In *AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development*, pages 51–62, New York, NY, USA, 2006. ACM.

[Ope] Open Handset Alliance. Android. http://www.code. google.com/android.

[PTB+08] Michael Perscheid, David Tibbe, Martin Beck, Stefan Berger, Peter Osburg, Jeff Eastman, Michael Haupt, and Robert Hirschfeld. *An Introduction to Seaside*. Software Architecture Group, Hasso-Plattner-Institut, April 2008.

[RSC06] Tobias Rho, Mark Schmatz, and Armin B. Cremers. Towards Context-Sensitive Service Aspects. In *Workshop on Object Technology for Ambient Intelligence and Pervasive Computing, co-located with ECOOP 06*, Nantes, France, July 3-7 2006.

[TT06] Éric Tanter and Rodolfo Toledo. A Versatile Kernel for Distributed AOP. In *Proceedings of the IFIP International Conference on Distributed Applications and Interoperable Systems*, volume 4025 of *Lecture Notes in Computer Science*, pages 316–331, Bologna, Italy, Jun 2006. Springer-Verlag.

[vSPK04] Mark van Setten, Stanislav Pokraev, and Johan Koolwaaij. Context-aware Recommendations in the Mobile Tourist Application COMPASS. In Paul De Bra and Wolfgang Nejdl, editors, *AH*, volume 3137 of *Lecture Notes in Computer Science*, pages 235–244. Springer, 2004.