

# Policy-Based Context-Management for Mobile Solutions

Caroline Funk<sup>1</sup>, Björn Schiemann<sup>2</sup>

<sup>1</sup> Ludwig-Maximilians-Universität München  
Oettingenstraße 67, 80538 München  
caroline.funk@nm.ifi.lmu.de

<sup>2</sup> Siemens AG, CT SE 2  
Otto-Hahn-Ring 6, 81730 München  
bjoern.schiemann@siemens.com

**Abstract:** Nowadays our world becomes more and more connected, and sensors and any kind of mobile (e.g. cell phones, PDAs) and fixed computational devices are linked together through wired and wireless networks. Accordingly the number of context-aware services will grow and more and more context will have to be managed. Thus this paper outlines an architecture for the policy-based management of context data. The architecture, its implications and benefits are discussed.

## 1 Introduction

Mobile devices and networks have become a commodity, and many people use them everyday. First data services became available, such as SMS and by now more sophisticated services like restaurant finders appeared on the market. The latter fall in the category of context-aware services which came into the focus of attention in order to adapt even better to a person's demand. Recently, RFID-tags as well as sensor networks have received growing attention in research and media. Combined with mobile user devices there is a trend towards pervasive systems, where devices with computational capabilities are present almost everywhere in order to be at our hands the best adapted way possible.

Facing the huge and still growing variety of mobile devices and the increasing number of context-aware services, it is obvious, that the provisioning and management of context, such as time, location or activity, for a heterogeneous dynamic environment and customer base is quite a challenge. At the same time, the provided services should be easy to use and therefore the details of the underlying architecture and technology shall be hidden. This document gives an outline, how this challenge can be solved in a form, which is applicable to mobile devices. The solution key are policies.

In the following, we will first give an overview of related work. Then we will introduce our architecture and outline its innovations. We will conclude with an overview of our next steps.

## 2 Related Work

A widely accepted definition of context is the following from Dey [De00]: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves."

Different researchers have developed architectures and concepts for context management in mobile systems. Dey [De00] introduces context widgets, that are derived from GUI widgets and build a layer between applications and context sensors. Their purpose is to hide the sensing of context and to abstract context information. They offer query and notification functions and are reusable by different applications. The key drawback of context widgets is the complex configuration of new widgets. Each widget stores a state together with a set of attributes and call-backs for notification of changes. These attributes and call-backs have to be determined by the developer. Furthermore, applications that request context from context widgets have to deal with failures themselves. Context widgets don't provide any failure management, that hides request failures from applications.

Solar, another system for managing context is developed at Dartmouth College (cf. [CK01]) as a distributed system that resides on different network nodes. Context sources or context refiners deliver context as events. Applications interested in certain context have to subscribe to the associated event queue. This is realized by a library residing on the same system as the application and through standard network protocols. The application is detached from the Solar system and runs on any platform. Solar is not capable of supporting disconnected operation, as it always requires contact to the context queue. Besides this, no mechanisms are provided that deal with non-available context.

In this paper, we focus on the most important related work, but [BP03] list a few more approaches to context management. Summarized one can see, that existing systems lack some important features. The most relevant, which we're going to provide solutions for, are convenient configuration for easy maintenance, support of disconnected operation, as well as failure management, in case the relevant context is not available or outdated.

To overcome the problems mentioned above we suggest the usage of policies to manage context. Policies are similar to event-condition-action rules. They are easy to write down, human-readable and easy to adjust. Policies divide a large management problem into sets of small rules. A policy system is stateless and event-based and may execute policies in parallel. Policies may also be adjusted at runtime. A centralized policy repository (PR) assures that all management activities are consistent. In order to complete a policy-based system, the following components are required (cf. [IET]): Policy Decision Point (PDP), where the decision is made, if a certain policy action will be executed or not and Policy Enforcement Point (PEP), a unit that takes care of the enforcement of the PDP's decision.

## 3 Architecture

Starting with the functional building blocks of a typical mobile middleware we identified those components, which have to be added in order to provide context data (for context-

aware applications) and for their policy-based management. Figure 1 gives an overview of this innovative architecture. The standard components for middleware such as session management or access control are left out in figure 1 and only new and components needed for context management are shown. The numbers in the following refer to the numbers in figure 1.

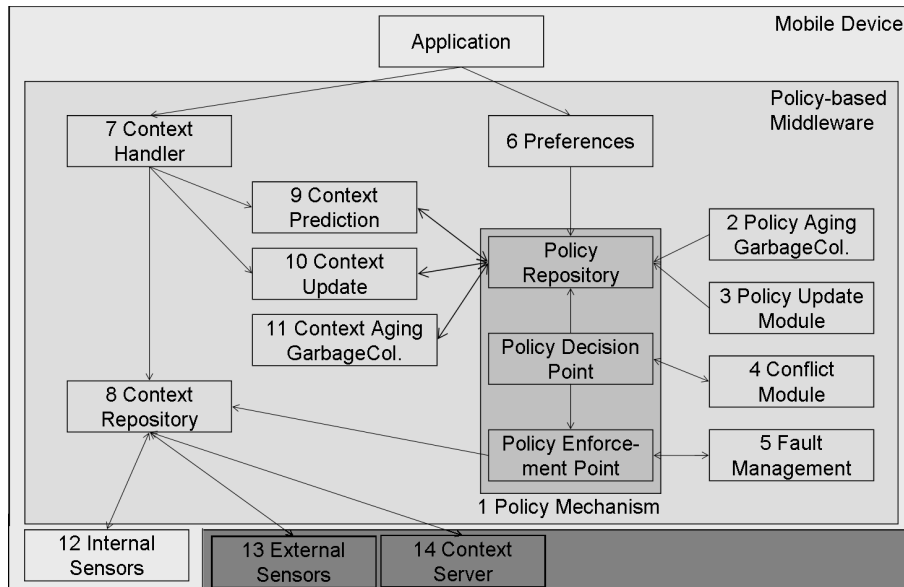


Figure 1: Architecture concept

In our concept we place all necessary components on the mobile device. Thus we are able to continue operation even if we are disconnected from the network. First a set of policy components (1) namely a PR, PDP and PEP (in the following referred to as *policy mechanism*) are necessary. In addition to the policy mechanism on the mobile device, a distributed and partially hierarchical structure with policy mechanisms on different levels and at different locations within the network is necessary. It is obvious that with limited capabilities of mobile devices not all policies can be held on the mobile device. Therefore we provide two complementary components on the mobile device, the policy aging and garbage collection (2) and the policy update (3) components. Old or unused policies will be deleted from the mobile device or any other policy repository by the policy aging and garbage collection component. To come to the decision, whether a policy may be deleted, the age, domain and other information of policies have to be taken into account. For example, the mechanism has to take care, not to delete policies generated locally by the user or the application as preferences (6). As a fallback, we provide the second component, the policy update module. It enables the download of policies from a policy repository in the network. To accomplish this, it compares existing policies with policies available in the network and downloads needed policies. To decide on downloading, the current user

context is taken into account. The policy aging and policy update module have to take each others decisions into account to avoid increased data transmission over the wireless link by alternately deleting and downloading policies again.

Challenges with the processing of policies are the resolution of policy conflicts and policy enforcement. After a certain event occurred the PDP decides which policies are triggered and which ones of these have to be enforced. Sometimes the order of execution of the policies has an impact on the result. Therefore, if the PDP evaluates more than one policy condition to true, the conflict module (4) decides on the order of execution. This decision will be based on information like priority, domain and other context. If problems like unavailable resources (e.g. sensors) occur during the enforcement of a policy, the fault management (5) is notified and triggers further actions.

The context handler (7) is the interface to applications that want to use context. The context handler tries to retrieve the desired context from its associated context repository (8). The current context as well as its history may be stored there, which is necessary for disconnected operation. If the specified context is not available in the context repository, the context handler generates events triggering policies that realize context update (10) or context prediction (9). In this way it delegates the request for context to the policy mechanism, that comprises policies for decision making and prediction. An exemplary process is, that first an event is generated to update the desired context by retrieving the context from an internal (12) or external sensor (13) or from some kind of context server (14). If this fails e.g. due to disconnection from all context servers, another event may be generated, that starts prediction of a feasible value by using a context history and other context information in the context repository. For example, if it is known, that half an hour ago the temperature was 10 degree and according to the calendar-context the location didn't change (or the integrated GPS sensor says so), it is likely that the temperature is still the same.

In addition a context aging and context garbage collection (11) component are necessary, to cope with the restrictive hardware on mobile devices, similar to the policy aging component (2). Besides that, context may be outdated e.g. location context of London is not interesting any more, if one flies to Munich. So this context can be removed from the mobile device.

Our middleware concept for mobile data management contains multiple innovations outlined in the following. First the middleware concept is applicable to mobile devices and even *small* mobile devices such as cell phones or PDAs. Due to context and policy aging their limited hardware capabilities can be satisfied. In addition it provides a policy mechanism adapted for usage on mobile devices. As all components are available on the mobile device, this mechanism may work partially independent from network connections. Thus it is an enabler for autonomic computing. The policy mechanism enables a highly scalable architecture through a distributed and hierarchical organization. Context management is implemented through management by policies. Due to human-readable and automated policy enforcement, the administration is less error-prone and easier than existing context management architectures. Furthermore, we provide a means for coping with context data failures: context will be updated or predicted in case of non available context. These are new features not used so far which support data management even in case of disconnected operation.

## 4 Summary and Future Steps

Based upon traditional approaches for mobile middleware we introduced a new platform architecture, which allows for easy administration of context data even on mobile devices. We exploit policies for the purpose of lifecycle management and prediction of context data. Such automated management of context data is less error-prone and thus may help to reduce total cost of ownership. Our policy-based middleware may be deployed on resource-constrained mobile devices like cellular phones or PDAs. Due to a hierarchical server and policy infrastructure, devices only carry dedicated policies. This helps to overcome implementation challenges like footprint, low bandwidth and disconnected operation.

The architectural design is in a stable state although it needs some additional work in particular in the areas of which policy language will be suited best and which prediction heuristics will be implemented. Our future work will comprise the development of a prototype, which will show the overall feasibility and serve as a platform for performance measurements. A distributed rule-base for event-condition-action rules has been implemented in Java for portability reasons and tested. It will serve as a starting point for the comprehensive policy-based approach.

## 5 Acknowledgements

Parts of this work have been funded by the German Bundesministerium für Bildung und Forschung (BMBF) within the framework of the project "wireless Internet - cellular systems". Further fruitful suggestions stem from a research cooperation between Siemens corporate technology and the institute of computer science at the LMU Munich.

## References

- [BP03] Barrett, K. und Power, R. State of the art: Context management. March 2003.
- [CK01] Chen, G. und Kotz, D.: Solar: Towards a flexible and scalable data-fusion infrastructure for ubiquitous computing. In: *Workshop on Application Models and Programming Tools for Ubiquitous Computing at Third International Conference on Ubiquitous Computing (UbiComp 2001), Atlanta, GA.* 10 2001.
- [De00] Dey, A.: *Providing Architectural Support for Building Context-Aware Applications.* PhD thesis. Georgia Institute of Technology. November 2000.
- [IET] IETF working group: Policy framework (policy).