The Journal of Systems and Software 92 (2014) 59–70

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Programmable context awareness framework

Bachir Chihani[a,b,*], Emmanuel Bertin[a,b], Noël Crespi[b]

[a] Orange Labs, 42 rue des Coutures, 14066 Caen, France
[b] Institut Mines-Telecom, Telecom SudParis, CNRS 5157, 9 rue Charles Fourier, 91011 Evry, France

A R T I C L E I N F O

Article history:
Received 28 September 2012
Received in revised form 9 July 2013
Accepted 22 July 2013
Available online 1 August 2013

Keywords:
Software engineering
Context-awareness
Privacy
Adaptation
XML

A B S T R A C T

Context-awareness enables applications to provide end-users with a richer experience by enhancing their interactions with contextual information. Several frameworks have already been proposed to simplify the development of context-aware applications. These frameworks are focused on provisioning context data and on providing common semantics, definitions and representations of these context data. They assume that applications share the same semantic, which limits the range of use cases where a framework can be used, as that assumption induces a strong coupling between context management and application logic. This article proposes a framework that decouples context management from application business logic. The aim is to reduce the overhead on applications that run on resource-limited devices while still providing mechanisms to support context-awareness and behavior adaptation. The article presents an innovative approach that involves third-parties in context processing definition by structuring it using atomic functions. These functions can be designed by third-party developers using an XML-based programming language. Its implementation and evaluation demonstrates the benefits, in terms of flexibility, of using proven design patterns from software engineering for developing context-aware application.

1. Introduction

Two distinctive architectural approaches (Gutheim, 2011) have been used to design context-aware applications: architectures that follow a broker model and those based on a point-to-point model. Both models contain two types of elements: context providers in charge of collecting contextual data, and context consumers (i.e., context-aware applications) that use contexts to adapt their behavior. In the first model, a context broker is used as an additional element to decouple context providers and consumers, limiting or eliminating the direct connections between them. This broker is thus, in most cases, in charge of context modeling and inference (Gutheim, 2011; Naudet, 2011; Filho and Agoulmine, 2011; Hamadache and Lancieri, 2010; van Sinderen et al., 2006). However, these tasks are performed in a pre-defined way and are barely customizable by context consumers. As a result, the generated information may not fully match the specific needs of consumers. In the second model, context consumers know the providers and send their requests to them directly (Oh et al., 2010). This model is less sophisticated; context consumers need to know which provider should be addressed for any given contextual information and should also be aware of their state (e.g., awake or asleep).

In both cases, consumers continuously request contextual information from their sources (either the context broker or directly from context providers) or subscribe to be notified with context updates. Upon reception, this information must then be processed by the context consumers to determine if it would impact their behavior. When the updating load increases, consumers become overloaded with messages, many of which are not at all relevant to them. Moreover, consumers have to store and handle context information locally to maintain a consistent vision of the user's situation and to adapt their behavior accordingly.

We have thus identified the need for a better separation within context-aware architectures. The application business logic and the context management operations should be kept separate, so that one of these logics can be modified without having to modify the other. To enable this separation, we propose to host all operations related to context management in the context broker, outside of context-aware applications. All of the context management operations to be executed in the broker will need to be specified with an appropriate language. For a given context-aware application, a script or a program written in this language will be executed by the context broker as the specified context events occur (e.g., events from context providers). The result of each execution is then sent back to the corresponding application, which does not have to maintain or retrieve this data.

* Corresponding author at: Orange Labs, 42 rue des Coutures, 14066 Caen, France. Tel.: +33 2 31 83 90 05.
E-mail addresses: bachir.chihani@orange.com (B. Chihani), emmanuel.bertin@orange.com (E. Bertin), noel.crespi@it-sudparis.eu (N. Crespi).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.07.046

We present a context management framework based on a broker architecture, which gives context consumers the ability to design, via an XML-based scripting language, their specific context processing that will be executed by the broker. This will allow context handling to be centralized into the broker, instead of being partially processed by the different context-aware services.

The rest of the paper is organized as follows. Section 2 presents our motivations and challenges for building effective context management architectures. Section 3 illustrates our framework architecture and the implementation details. Case studies showing the possible applications of the framework are presented in Section 4. In Section 5, we present an evaluation of the proposed framework through a real case study and performance evaluation in a simulated environment. Section 6 discusses related work and the proposed framework. Finally, we conclude the paper and present future work in Section 7.

2. Motivation and challenges

In this section we highlight the benefits of a modular design of context management frameworks by separating the main features of the framework into different non-overlapping functions, thereby making it much simpler to build different types of context-aware applications.

2.1. Context-awareness

Context-awareness aims to enhance services (e.g., tracking, navigation, information, communication, and entertainment services) by making them sensitive to users' situations, and therefore making them more adapted to user's needs (Chihani et al., 2011a). Context management frameworks are used to empower the development of such services by gathering context-related functions in a common component and exposing its functionalities to third-party services looking for context awareness.

Many of the efforts in building context management frameworks are focused on specific domains (e.g., IPTV (Song et al., 2010), IMS (IP Multimedia Subsystem) environments (Simoes and Magedanz, 2011), and document management systems (Balinsky et al., 2011)) making their proposed frameworks virtually impossible to re-use for building context-aware applications in other domains. Other works have proposed more generic frameworks (Gutheim, 2011; van Sinderen et al., 2006; Plesa and Logrippo, 2007) enabling different types of applications to coexist and to be built on top of these frameworks. However, these frameworks do not offer customization capabilities – a serious lack since applications often have different requirements.

Our contribution is a generic framework, based on the Observer design pattern (Gamma et al., 1994), which is flexible enough to be customized for building context-aware applications in different domains. Herein, flexibility refers to dynamic adaptation, customization and component reusability. The use of the Observer pattern allows the framework core component (i.e., the context broker) to be observed by third party applications context consumers. In addition, these consumers can define, through a specification document, the set of internal states of the broker they want to observe in order to be notified of any specific changes. Furthermore, this specification defines how the framework should process context data, as well as the application callback address on which to be notified when an event of interest occurs. When the broker processes new published context data, it may change its internal state and as a result notifies the consumers interested in these events by sending messages to their callback addresses.

2.2. Design considerations

Contextual information is dynamic by nature. The measurements change over time (e.g., temperature, location) with a variable rate. Also, a context may lose consistency (become outdated or incorrect) because the sensing operation may fail or the sensing environment may become too noisy. These characteristics bring a high degree of complexity to context management, and consequently to context-aware applications, as context management is usually coupled with application logic.

To address the dynamic characteristics of contextual information and its complicated management issues, we identify the following requirements for a 'developer-friendly' context management framework:

- The framework should support the rapid development of context-aware applications through the reduction of context management complexity.
- To facilitate the re-use of the architecture in different environments (e.g., Telcos, Machine-to-Machine) the communication protocol used to transport information between all components of the architecture should be generic and not specific to a given technology.
- Information representation should be very flexible so that application developers are not forced to respect strict formatting rules in order to circulate data among the architecture's components. Applications should be able to subscribe to a subset of all context events generated by updates from providers for given contextual information.
- The mechanisms provided to application developers for customizing context processing should be kept as simple as possible to reduce the learning curve for developers.
- The framework should provide appropriate mechanisms allowing the respect of user privacy enabling users to grant or prohibit third-party applications to access his/her context data and as a result controlling the adaptation level of these applications.

3. Context management framework

3.1. Framework architecture

In order to meet these requirements, we propose a programmable framework (Fig. 1) for processing contextual information, based on six primitive functions related to context management: *produce*, *filter*, *abstract*, *select*, *aggregate*, and *consume*. Each of these functions corresponds to a specific action from the well-known layered approach (Schmidt, 2006; Chihani et al., 2011a) in context management.

The "*produce*" function consists of producing raw contextual information. It is implemented by context providers that wrap sensors to comply with the framework API for publishing context.

The "*filter*" function provides signal processing functionalities that aim to eliminate or at least reduce noise in contextual

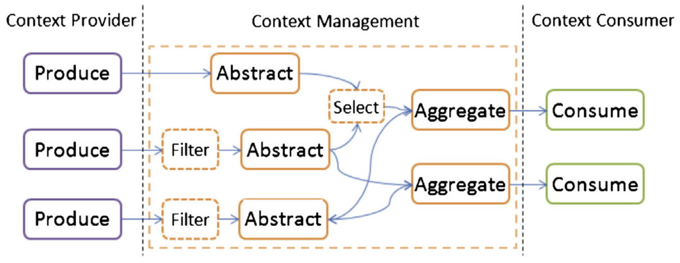

Fig. 1. A layered framework for context management.