Introducing Context-Awareness in Applications by Transforming High-Level Rules

Carlos Andrés Parra, Maja D'Hondt, Carlos Noguera, Ellen Van Paesschen

INRIA Futurs - ADAM Team
Parc scientifique de la Haute-Borne
40, Avenue Halley
59650 Villeneuve d'Ascq Cedex
{carlos-andres.parra, maja.d-hondt, noguera, ellen.vanpaesschen}@lifl.fr

Abstract. In the last years, we have witnessed the increase in the popularity and capabilities of mobile technologies. This evolution has enforced the idea of smart environments, in which devices are aware and able to react to changes in their environment. In this position paper we describe a specific approach for the development of context-aware software. We propose to make existing applications context-aware by means of three main components: context models, high-level rules and code-generation processors. We present each component and analyze the issues related to the development of context-aware software following this strategy.

1 Introduction

The proliferation of mobile devices in everyday life is moving from fixed computers to a wide range of smart and mobile devices with wireless networking capabilities. Mobility opens the door for new kinds of challenges in software, one of such is the capability to adapt applications to be *aware* of the context and to react properly to its changes. By context we informally mean a set of situations in a given environment at a given time. In order to achieve this *awareness*, devices and applications should interact with each other to share context information, gathered mostly through physical sensors that can detect different data such as temperature, time or presence of other nearby devices. Context-aware applications require mechanisms to specify actions when context changes. One way to do this is through the definition of a *language* in which we are able to express specific behavior in the following form: *for some specific context conditions, perform an action*. This structure has two main parts, the conditions on the context and the action that has to be performed when the conditions are met. In this paper we discuss the issues related to guiding the behavior of applications based on the events produced by changes in the context.

Model-driven Architecture (*MDA*) [6] aims for a higher level of abstraction in software development, by separating the business elements from the specific details related to the platform and implementation issues. We intend to follow the same strategy with the definition of high-level *rules* to express the behavior and the preferences of users in a platform and implementation-independent way.

These rules can be used then as an entry point to a process of code transformation. This paper is organized as follows. In section 2 we present a brief summary of the related work. In section 3 we first introduce our approach to the development of context-aware software. Then we describe the three main components of our proposal: a context model that represents the basic structure of context information, a rule language to express the context-action pairs and a process that adapts the application based on the context model and the rules. Finally, in section 4, we present some conclusions.

2 Related Work

There are several related areas involved in the development of context-aware software. We introduce briefly some of the most relevant, classified in three different categories. Firstly, the middleware and the implementation details. Secondly, the representation of context information, and finally, the use of rules to express behavior in terms of high-level elements.

On the implementation side, the need for new middleware arises. As stated in [13], current middleware technologies are not adequate to handle the restrictions that mobility and smart environmental systems impose. These characteristics include: volatile connections, processing and memory restrictions on mobile devices, narrow communication channels, reduced screens, restricted input mechanisms, and the list goes on. A set of context-aware implementations can be found in the literature, which, represent working prototypes of context-aware applications. Some of these implementations are `Hydrogen` [5] a three layer architecture and a framework to support context-awareness, `Gaia` [11] a middleware infrastructure to support the development of mobile and ubiquitous systems, and `CybreMinder` [2] a prototype application implemented using the `Context Toolkit` [12], to handle reminders associated with context information. We assume as a basis for our approach that a context-aware middleware exists and that is able to propagate updated context information.

An approach to develop adaptive service-oriented is presented in [8]. In this approach, the authors aim for the generation of adaptation points for a given application, based on meta data. Then, at runtime and using these points of adaptation exposed as services, context-sensitive aspects can interact with the original application and adapt properly to the situation. In our approach, we also start from meta data about the applications, but we also use high-level rules that describe the desired behavior of applications to face a context change, and information about other devices and applications. With this elements, we propose to build code-generation processors to generate the parts that the application requires to be context aware. Our approach and all the elements on it, will be described in more detail in section 3.

A different field is concerned with the representation of context information. A generic ontology is presented in [10]. It is based on four main concepts: user, environment, platform and resources. Here, ontologies are mainly employed to enable communication across different devices in the same network. The Unified

Modeling Language (UML) can also be used to model context, as proposed by `ContextUML` [1]. This is a language for the model-driven development of context-aware web services. Here the models are used to separate the definition and information related to the context from the specific implementation. There are other characteristics that make context information difficult to model. As stated in [4] and [3], sometimes it is necessary to differentiate between static and dynamic information. Static information refers to data that does not change, for example, the name or date of birth of a user, whereas dynamic information refers to data that may change over time as for example the location of a user.

As we will show in detail in Section 3.1, we borrow some concepts from these and other approaches to construct our own context information model using an UML class diagram.

Finally, rule languages are used in some cases to achieve context-awareness. `CRIME` [7] for example, is a prototypical implementation of the `Fact Space Model`, which is a coordination language that provides the applications a view of their environment. The rules in `CRIME` describe the behavior of the applications according to the context information. `CRIME` also deals with disconnection by invalidating the facts and the conclusions that are drawn from devices that are no longer available in the environment. As we describe and argue in Section 3.3, it is our intention to *compile* context rules, rather than to *interpret* them. A result of our approach is that we will not be able to invalidate inferred facts when the context changes.

3 Context-Aware Applications

In this section, we present our approach to the development of context-aware applications. We define high-level rules to express software adaptation to context. Using a rule language, it is possible to specify the preferred actions to follow changes in context. The rules are then transformed into software assets that are merged with the existing application.

To illustrate this approach, let us consider for example a basic jukebox application. This application has a standard functionality: play a song, surf a list of songs, and turn the volume up and down. If we wanted to introduce context-awareness, we would need to transform the application to add some extra behavior, for example, to detect if there is a cellphone nearby and, subsequently, to turn the volume down automatically. In such a transformation we do not want to create new functionality to turn the volume down, but to use such functionality whenever a cellphone is present in the context. To achieve this functionality we propose a process like the one shown in Figure 1.

In this diagram, there are three main parts. First, we need a model to express context information (`ContextModel`). That way, we are able to represent information related to users, devices, location, time, etc. The rules(`High-level Rules`), on the other side, represent the actions to be followed by the context-aware application when a certain change in the context occurs. Finally, there is

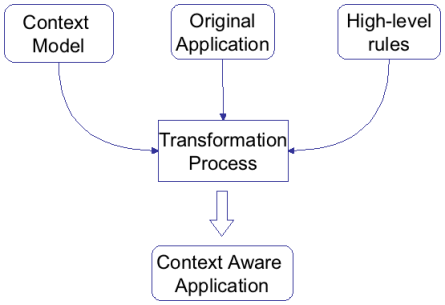

Fig. 1. Process to transform non-context-aware applications

a transformation process in charge of extending the (non-context-aware) original application, taking as input the context model and the rules.

3.1 Context Model

Regardless of the variety of devices, there should be a common understanding between all the participants about the context information they want to share. To achieve this, we first design a context meta-model (left side of Figure 2), to describe the classes and relationships in a context model for a given situation. The first element is the `Context` which stands as the root of the meta-model. A `Context` contains `ContextDescriptor` and `Entity` elements. The `Entity` elements represent all the actors of an environment, for example the users and their devices. `Entity` as well as `ContextDescriptor` elements may have a group of `Attribute` elements. Finally the `Entity` offers functionality represented by the `Service` element. On the right side of the (Figure 2), there is a context model that conforms to the context meta-model. This model represents a particular context in which there are `Jukebox` and `Cellphone` elements, contained in an `Environment`. Here we use the stereotypes to represent the relation between the elements in the model and their corresponding meta-class in the meta-model. The context model should be used as the language to express behavior and actions to follow in the high-level rules.

Aside from having a structure to represent context information, it is necessary to define the responsibilities for populating the context model. In context-aware software development, where a wide variety of devices and applications interact in the same environment, fundamental questions have to be addressed such as, what part is responsible for keeping the context information up to date, and what update strategy is going to be used. For example, when someone holding a cellphone arrives, it is possible that the jukebox application retrieves the updated context every once in a while and detects the new device, but it is also possible that an underlying middleware triggers an event when the cellphone is detected. In our proposal we assume the presence of a middleware that generates an event every time a change in the context is detected.

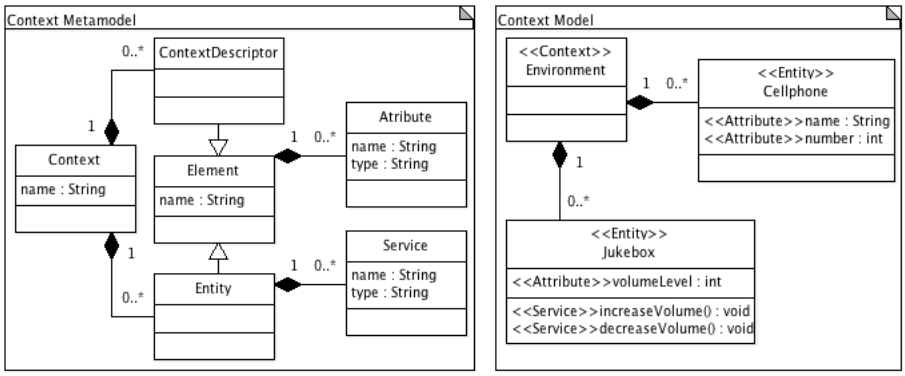

Fig. 2. Metamodel and model of context information

3.2 Rule Language

As we mentioned in Section 1, rules play an important role in our approach. By writing rules in terms of the context model, we think it is possible to express the desired behavior of the applications. We consider that a scheme where rules express conditions of the type `IF (condition) THEN (action)` is appropriate for our purposes. The condition as well as the action can be written in terms of the context model as is shown in the following example:

```
IF(Cellphone.name = MyCellphone)
DO Jukebox.decreaseVolume
```

Here we present a single rule that evaluates a condition and describes an action to follow, if the condition is met. In this case, the jukebox application is asked to decrease the volume if a cellphone with the name *MyCellphone* exists in the context model. One important question to answer about the rules, is when they need to be evaluated. This is directly related to the mechanisms we use to keep the context information updated. As we mentioned en the previous section, we assume the existence of an underlying middleware in charge of populating the context and of notifying the application whenever a change in the context is detected and the model needs to be updated.

3.3 Program Transformation

Given that in our approach we propose to extend existing applications so that they are sensible to changes in the ambient context, a module that transforms these applications is necessary. Such transformation must insert into the application the concepts described in the context model, and it must modify its control-flow to reflect the behavior specified by the rules.

We opt for a compile-time adaptation of the program rather than a dynamic one to better cater for the limitations of current ambient execution environments. Indeed, mobile devices deal with restricted resources in terms of available memory and processing power. By relying on source code transformation, our approach does away with the overhead of additional libraries or custom execution environments (for example, a modified Java VM) that would be needed for dynamic adaptation. In addition to this, by including the context model description and rule set definition at compilation time, it becomes possible to *compile* rather than *interpret* the context rules. This will allow the transformation module to optimize their implementation, although, to what degree remains to be seen.

To perform the transformation, programmer inserted annotations in the base application, as well as the context model and rule set, will be consumed by a source code processor. This processor will be responsible for the weaving of the rules and context entities in the base program, while leaving the application's original function intact. To implement the transformation for programs based on Java, we use the Spoon framework [9], since it provides a fine-grained representation of the program and offers several facilities for the processing of annotations.

Nevertheless, by using a compile-time approach, we cannot offer certain capabilities to users as to create or modify rules at runtime. To minimize the impact of such restrictions, it is imaginable to create means to automatically compile and deploy new rules. However, the development of this kind of features remains as future work and is out of the scope of this paper.

4 Conclusions

Despite the recent wave of attention in context-aware systems, we think that there is still a gap in order to bring context modeling to a real implementation. First, we believe that models ought to be used not only to exchange design ideas, but also as a core of the development of context-aware applications. We envision context models being used in transformation and generation processes. As such, code will be generated in order to adapt applications according to variations on their context.

This paper presents our approach to making existing applications context-aware by means of context models, high-level rules and code-generation processors. This approach is currently mostly in the conceptual phase, although most parts have been implemented or can be largely based on our previous work and experiences. The largest challenge is without doubt determining the locations in the existing application where the transformed high-level rules and context information need to be inserted. Another fundamental question is to what extent we will be able to generalize our approach, independently of the application that we wish to transform.

References

1. *ContextUML: a UML-based modeling language for model-driven development of context-aware Web services*, 2005.
2. A. K. Dey and G. D. Abowd. Cybreminder: A context-aware system for supporting reminders. In *HUC '00: Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 172–186, London, UK, 2000. Springer-Verlag.
3. K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems, 2005.
4. K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems, 2002.
5. T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger. Context-awareness on mobile devices - the hydrogen approach. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, Washington, DC, USA, 2003. IEEE Computer Society.
6. J. Miller and J. Mukerji. Mda guide version 1.0.1. Technical report, Object Management Group (OMG), 2003.
7. S. Mostinckx, C. Scholliers, E. Philips, and C. Herzeel. Fact spaces: Coordination in the face of disconnection, 2007.
8. H. Mugge, T. Rho, and A. B. Cremers. Integrating aspect-orientation and structural annotations to support adaptive middleware. In *MAI '07: Proceedings of the 1st workshop on Middleware-application interaction*, pages 9–14, New York, NY, USA, 2007. ACM Press.
9. R. Pawlak, C. Noguera, and N. Petitprez. Spoon: Program analysis and transformation in java. Technical Report 5901, INRIA, may 2006.
10. D. Preuveneers, J. Van den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. De Bosschere. *Towards an Extensible Context Ontology for Ambient Intelligence*. 2004.
11. M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. Gaia: a middleware platform for active spaces. *SIGMOBILE Mobile Computing and Communications Review*, 6 (4):65–67, 2002.
12. D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM Press.
13. T. Strang and C. L. Popien. A context modeling survey, September 2004.