Design Patterns for Context-Aware Adaptation

Gustavo Rossi[1,4], Silvia Gordillo[1,3], Fernando Lyardet [2]

[1] Lifia. Facultad de Informática. Universidad Nacional de La Plata, Argentina
Fernando D. Lyardet. Telecooperation, Darmstadt University of Technology, Germany
[3]Also at CICPBA, [4]Also at Conicet
{gordillo,gustavo}@lifia.info.unlp.edu.ar, fernando@tk.informatik.tu-darmstadt.de

Abstract

In this paper we show how the judicious use of design patterns can improve the design of context-awareness in software systems. We first review well-known problems in the development of context-aware applications. Next, we present our view on the design process of context-aware software; we introduce design patterns and explain why they can be useful to improve the quality of this kind of applications. We finally present some design patterns we mined by researching on successful context-aware approaches. Some concluding remarks are finally presented together with further work we are pursuing.

1. Introduction

An interesting research area related with context-aware software involves finding the most appropriate design structures to represent context and to provide context-awareness. In this paper, we focus particularly on those design issues that developers should consider when dealing with context-aware adaptation. We have identified a set of design micro-architectures that allow customizing services (application behaviors perceived by the user) according to the user's current context. However, we will not focus solely on our work, but we will reflect on the work of others: those successful builders of context-aware software. We will use patterns as tools to record and convey good design decisions in the field of context-aware software. The contributions of this paper are twofold:

-We show the importance of recording recurrent problems related with context-awareness and their solutions in the form of patterns.

-We present a set of patterns to illustrate our point of view.

The rest of the paper is organized as follows. We first present our (critical) view on how design issues are dealt with in the field of context-aware software. Next, we introduce the concept of patterns and analyze why they are helpful in our domain. Then, we present some concrete patterns, explaining how they have been used by successful designers. Finally, we present some further research we are pursuing.

2. Design Issues in Context-Awareness

As cleverly pointed out in [7], most of the recent research on context-awareness has adopted an infrastructure-centered, rather than a conceptual view, presenting specific frameworks for gathering, managing and disseminating context information.

We can go further in this analysis and state that, even while context representation and acquisition is clearly understood by the community, context-aware adaptation is still designed using ad-hoc practices; we are far from having an engineering view of this aspect of context-awareness.

It is somewhat clear which abstraction constructs are necessary to implement context information acquisition. Context widgets as presented in [12] represent a good approach for decoupling sensors from more abstract context data. Aggregating and transforming contexts according to the needs of applications has also been discussed in [12] and elsewhere. These ideas follow well-known practices for decoupling of concerns, in particular extensions to the popular MVC architectural style as explained in [9]. In the same way, the process of notifying the application about changes in the current context has been also studied in the literature, and it is reasonably clear when push or pull approaches are to be used [2].

However, the task of context-aware adaptation is seldom addressed in the literature, or just left as an application concern. One could argue that, once acquired and transformed, context information is not different to other kind of application's data, and thus the process of adaptation should follow existing approaches in more general software systems.

Surprisingly however, most existing applications use rules for adaptation, i.e. applications implement some kind of "if-then" structure that, according to the actual values of contextual variables, decides which action must be performed. These program structures may appear as production rules as in [3], or rule objects as in [14]. While easy to program (even by final users), rule sets can become monolithic and make evolution a nightmare. We show in the rest of the paper, that there are other design structures that can (and should) be used to implement context-aware adaptation.

If context-awareness becomes mainstream, hundreds of developers will create their own applications. They will not use existing frameworks or infrastructures, unless they finally become standard or widespread used (as Jakarta Struts for the "old" field of Web applications). So, how can we help them in the process of building those applications?

We need to show them which problems they will surely face and how to solve those problems. We must do that, by taking into account that their applications will be novel, i.e. new kinds of contexts and context-aware adaptation will arise. In the following section we introduce design patterns, and show how to apply them to record and convey good solutions to context-aware adaptation problems.

3. Patterns for Context Aware Adaptation

Design patterns [4] describe problems that occur recurrently, and specify the core of the solution to those problems in such a way that we can (re)use this solution many times in different contexts and applications. Design patterns complement design methods as they show solutions that go beyond the naive use of primitive constructs. Patterns act as high-quality micro-architectures in a system; they usually help to improve system modularity and ease of extension. Knowing about patterns in a domain helps us know what experience designers do when facing a non-trivial problem.

Some authors have already used patterns to record design experience in different aspects of ubiquitous and mobile computing. For example, in [8] different strategies for using ubiquitous computing are recorded as patterns; [11] presents some patterns for organizing components in a mobile setting. In the field of context-aware computing there are different design concerns and certainly we could mine patterns related with each of them. For example, if we address the problem of context acquisition and abstraction we will find that Widgets as in [12] certainly express a pattern: the idea of decoupling sensors from the rest of the software by using intermediate objects is independent of a particular implementation framework. In other words, we can re-write the idea in such a way that (if necessary) a novice programmer can implement his own widgets library just by grasping the abstract idea behind the concept. Similarly, we can focus on how an application becomes aware of changes in the actual context and discover some relevant patterns.

Patterns can be described in different levels of abstraction: for example we can express the solution in terms of communicating objects, or architectural modules. We can instead use a notation such as the one in [7] to describe patterns related with conceptual modeling of context-aware software.

In this paper we will focus on design patterns for context-aware adaptation. We chose this field because, as previously explained, we feel that it is quite fertile for showing good and somewhat ignored design solutions which in some cases, result from the judicious application of well-known design structures. As usual with patterns, some problems must be faced using combination of them, i.e. patterns are not supposed to be applied in isolation. See for example 4.3 and 4.4 for an idea of possible combinations.

For the sake of conciseness we will use a reduced template as the basis for representing patterns. The template contains a Problem section that briefly shows the (recurrent) abstract problem with an example. The Solution section shows how expert designers face the problem. Diagrams are presented in UML and, similarly to [4], we indicate the role of classes in the pattern solution; when possible we enclose in parenthesis the example class. The Applicability section indicates when we must use this solution; finally we include a Known Uses section as examples of use of the pattern. We purposely present Rule-Based Adaptation as a pattern to clearly indicate when rules should be used.

3.1. Typified Context Element

Problem: You want to adapt your application according to the actual user's device (Palm, Cell Phone, etc). For example, you want to present different information or use different interface styles according to the device.

Solution: Describe devices in a class hierarchy and define polymorphic methods in each class for providing the expected result. Instead of representing the context variable as a string (e.g. "Palm"), define it as an instance of the corresponding class in the hierarchy. Delegate the corresponding adaptation into that context object. In this way, you avoid having a rule for each possible type of device. In Figure 1 we

present the generic structure of this pattern in which Context elements are modeled as a class hierarchy with a polymorphic behavior invoked by the application when adapting to context.

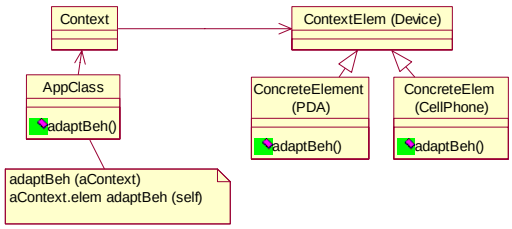

Figure 1. Treating context variables as types

Applicability: Use this pattern when i) a context value can be representing as an instance of a set of polymorphic types, e.g. user roles, types of activities, moods, devices, etc. and ii) the user is not supposed to add new type of values during run-time.

Known Uses: This pattern is being largely used in [13]

3.2. Active Context Element

Problem: Suppose that your software must provide different services according to the place you are; for example in an augmented reality Museum, you can see additional information when you are in front of an artwork; different artworks allow you different kinds of interactive experiences

Solution: Model each place as a full-fledged object and assign a set of command objects [4] with corresponding services to that object. As in 3.1 define the corresponding context attribute as a reference to this object, instead of solely a string with its name. Delegate the selection of services to the context object, which in turn will use its commands. Context elements in Figure 2 can exhibit different sets of behaviors, configured as commands.

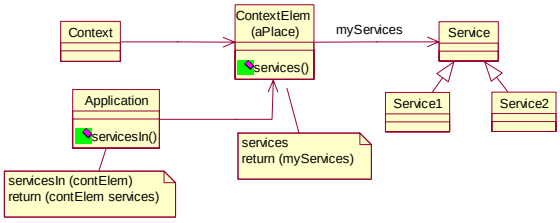

Figure 2. Context as Active Objects

Applicability: Use this pattern when the context itself (or a relevant context element) can be viewed as an object; the most usual case occurs with outstanding physical places in the application. Notice that 3.1 is a particular case of this pattern in which each context can be modeled as a class. For example when a tourist has a set of services when entering any Church we use 3.1; when each Church (e.g. Notre Dame, Sacre-Coeur, etc.) provides a specific context-aware behavior, we use *Active Context.* Notice that in 3.1 the commands are replaced by class´ methods.

Known Uses: A nice example can be found in [5]. Physical Hypermedia applications [6] also use extensively this pattern.

3.3. Rule-based Adaptation

Problem: You are building a system that must adapt its behavior according to different time intervals. For example the response should vary between 2 and 3 pm and between 4 and 5 pm. A completely different behavior is expected at night.

Solution: Use condition/action rules to implement the behavioral adaptations. This solution has several variants. If the user is expected to add his own rules (e.g. “I don’t want to be bothered between 4 and 5”), use Rule Objects in which conditions and actions are full-fledged objects [14]. If the set of rules might include contradictory rules, use either meta-rules to define the adaptation strategy or production-rules algorithms to choose which rule is to be executed.

Applicability: Rules should be used when the adaptation is performed on values that can not be typified (e.g. times, dates, weights, temperatures, etc.). We also use rules when they involve complex logical conditions between context elements: e.g. “when not at home”, “when busy or when using the cell phone”. Rules should be also used when they are defined at run-time by the user. Notice that many AND conditions involving types or objects can be solved by using a combination of delegation to context as in 4.1 and 4.2 and *Rule-Based Adaptation.*

Known Uses: This pattern is predominant in the literature. For example, it is used in the UWA project [14] and in [3].

3.4. Context Wrapper

Problem: You are extending existing software to add context-aware behaviors; for example your legacy academic information system must adapt its services to the role and context of use. You have identified one particular behavior that must be adapted and you have

decided to use 3.1, 3.2 or 3.3. How do you introduce this "new" code?

Solution: Wrap the corresponding class with an object that delegates the request to the component implementing the adaptation (e.g. a rule object or rule manager). This solution uses the Decorator pattern [4] to add new code unobtrusively. In Figure 3, if we need that *operation* (in *ApplicationClass)* behaves in a context-aware way, we use a context wrapper object that intercepts messages sent by any client, forwarding them to the corresponding adaptor object. We thus avoid rewriting *operation*. Notice that, if *ApplicationClass* may be seen as a context (e.g. an Artwork, a building, etc), the context wrapper might implement itself the adaptation, for example using 3.2.

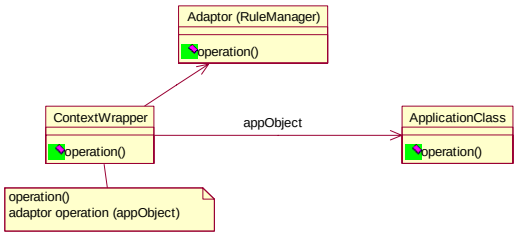

Figure 3. Wrapping Legacy Classes

Applicability: Use this pattern when you want to extend a legacy system (or class) and you do not want to modify existing code

Known Uses: The use of this pattern for adaptation to the individual user (i.e. personalization) has been discussed in [1]

4. Concluding Remarks and Further Work

We have discussed the use of design patterns to record and convey different strategies for context-aware adaptation. We have shown that in spite of the extensive use of rules to implement context-awareness there are other simple and effective strategies to adapt object behaviors to changing contexts.

We believe that, in order to improve the process of building context-aware software, we need a design "culture" that is independent of particular frameworks or infrastructures. However, we must learn from those successful projects, and extract the underlying software structures that can be used once and again in new endeavors; patterns are an elegant and effective tool for this. We are currently working in the definition of a complete pattern language that covers not only context-aware adaptation but also application's notification and context representation. The language not only contains individual patterns but also the design decisions and rationale for choosing one instead of others.

5. References

[1] J. Cappi, G. Rossi, A. Fortier, "Personalization policies need more than Rule Objects", In Proceedings of OOIS 2002, Springer Verlag LNCS, 2425, pp 117-123.

[2] K. Cheverest, K. Mitchell, N. Davies, "Exploring context-aware information push", Personal and Ubiquitous Computing, Springer, Vol 6, 2002, pp 276-281

[3]A. Fitzpatrick, G. Biegel, S. Clarke, V. Cahill, "Towards a Sentient Object Model", Proceedings of the Workshop on Engineering Context-Aware Object Oriented Systems and Environments, 2002.

[4] R. Gamma, R. Helm, R. Johnson, and J. Vlissides "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, 1995.

[5] T. Kanter, "Attaching Context-Aware Services to Moving Locations", IEEE Internet Computing V.7, N.2, pp 43-51, 2003.

[6] F. Hansen, N. Bouvin, B. Christensen, K. Gronbaek, T. Pedersen, "Integrating the web and the world: contextual trails on the move" In Proceedings of ACM Hypertext 2004,

[7] K. Henricksen, J. Indulska: "A Software Engineering Framework for Context-Aware Pervasive Computing". Proceedings of IEEE PerCom 2004, Orlando, USA. pp 77-86.

[8]J. Landay, G. Borriello, "Patterns for ubiquitous computing". IEEE Computer 36(8): 93-95 (2003).

[9] J. Pascoe, "Context-Aware Software", PHD Thesis, University of Canterbury, August 2001.

[10]D. Petrelli, E. Not, M. Zancanaro, C. Strapparava, O. Stock, "Modeling and Adapting to Context", Personal and Ubiquitous Computing Sringer, Vol 5, 2001, pp 20–24.

[11]J. Roth, "Patterns of Mobile Interaction", Personal and Ubiquitous Computing 6(4) 282-289 (2002)

[12] D. Salber, A. Dey, G. Abowd, "The Context Toolkit: Aiding the Development of Context-Enabled Applications" Proceedings of ACM CHI 1999, pp 434-441.

[13] SmartKom Project, http://www.smartkom.org/

[14] UWA Project. www.uwaproject.org