Decoupling Personalization Aspects in Mobile Applications

Arturo Zambrano[1], Silvia Gordillo[1,2], and Luis Norberto Polasek[1]

LIFIA, Facultad de Informatica, Universidad Nacional de La Plata 50 y 115 1er Piso

1900 La Plata, Argentina

1 {arturo, gordillo, pola}lifia.info.unlp.edu.ar

2 CIC, Provincia de Buenos Aires

1 Introduction

Mobile computing is constantly evolving and it is evident that with the advance of technological issues this trend will grow. Mobile software, executing in small devices such as Personal Digital Assistants (PDAs) and cell phones, must provide the user with a great variety of information and services that will be even more complex in the future to come. In this context and, as explained in [10], efective use of information and services can only be carried out by using adequate personalization mechanisms to present the information and services in a way that is better suited to the user. Research on personalization issues has been quite important for Web software, but personalization on software running in mobile devices is still premature. In this direction we propose to address behavioral adaptation for mobile applications by using the Aspect-Oriented Programming paradigm, following the ideas presented in [11] to introduce adaptation and in [6] to identify concerns. In this paper we present an architecture in which components implementing functional applications' requirements are completely decoupled from those implementing personalization features in order to obtain independent evolution of both. Separation of those concerns is achieved by using aspects that model adaptation components, isolating them from the base application. This paper is organized as follows: In Section 2 the basic personalization concepts using throughout the paper are presented and related work in this subject is discussed. In Section 3 we describe the most important issues when realizing adaptation in mobile software. In Section 4, the basic concepts of Aspect Oriented

Software are introduced. Section 5 presents our architecture and in Section 6 we show how to map a concrete application onto the presented architecture. Finally, some concluding remarks and further work are discussed.

2 Personalization

According to [1] personalization is understood as the process that adapts functionality, interface or information contents to make it more relevant to a particular user. For an application to be personalized it must know the user's context, i.e. all those features that characterize the execution environment including user information and preferences. Personalized software should maintain models of the objectives, characteristics, preferences and knowledge of the intended user. These models are used to keep up-to-date information on each user (usually called user profile) to adapt services to his preferences, in order to satisfy his needs [8]. This adaptation will also consider usually other contextual elements and will involve presentation or management issues. The adaptation process consist usually in three tasks depicted in Figure 1.

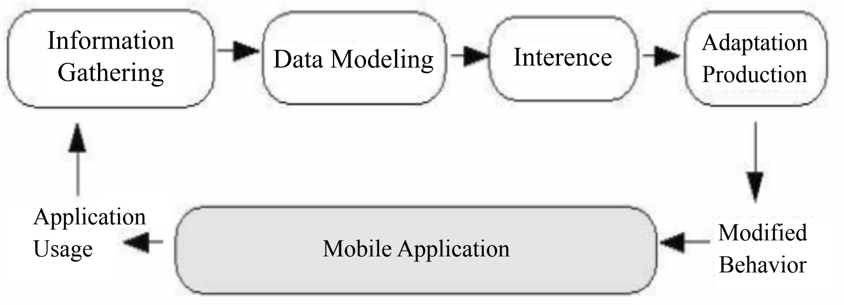

Fig. 1. Adaptation process for personalization

User data acquisition. In this process we identify available user data, such as his characteristics, behavior and environment. With this information, initial models of the user preferences are built.

User proÞle representation and secondary inference. That is, presumptions on the user and/or group of users, behavior and environment are elaborated.

Production or Adaptation. It generates content, presentation or behavior adaptations that are then introduced in the application.

It is interesting to note that, due to the previously mentioned characteristics, when an application is modified in order to support personalized behavior, code must be added in diferent modules of the application. This makes the task of application evolution difcult and error-prone. Among most usual actions, to adapt the information presented to the user, we can mention:

- Filtering: It consists in removing information or services that are not interesting to the user.
- Ordering or Priorization: It is achieved by reorganizing information according to the user preferences Suggestion It consists in giving spontaneous suggestions to the user, presenting information or suggesting tasks that are assumed of his interest.

3 Adaptation for Mobile Devices

Nowadays, mobile devices such as cell phones and PDAs allow the user to access information and services according to his geographical position and current activity. Changes in the geographical position and the situation where the device is used are inherent characteristics of such systems.

The information delivered through mobile devices is bound to the usage context, the activity in which the user is involved, and user preferences. On the other hand, the minimal resources available in mobile devices impose constraints regarding information processing. Wireless communication is expensive and not reliable. Storage capacity is very limited and processing is not powerful (typically less than 200MHz). Furthermore, graphics displays are not always available, or they are low resolution ones. In this context adaptation is a fundamental tool needed to cope with these issues.

Furthermore, these non-functional requirements must be envisaged from the early design phase. But they have a negative effect on the design and implementation. The inclusion of such non-functional concerns tends to complicate the design making application modules hard to understand.

At the same time it is very diffcult to trace where those requirements are implemented, since they are spread along several modules.

Then, the resulting designs and implementations of functional and non-functional requirements are coupled. In the worse case, depending on the cou-pling level, the final implementation of non-functional requirements can be em-bedded into the functional requirements' implementation.

4 An Overview of Aspect Oriented Programming

In the application development process, it is common to find a set of concerns that affect many objects beyond their classes which constitute (in object-oriented programming) the natural units to define functionality. They are called crosscutting concerns. A crosscutting concer*n* is one that is spread along many of the modules of a system. Typical crosscutting concerns are persistence*,* synchronization*,* erro*r* handling*.* etc*.* As it is said in [3]: "...existing software formalisms support separation of concerns only along a predominant dimension neglecting other dimensions... with negative effects on reusability, locality of changes, understandability...". These secondary dimensions correspond to crosscutting concerns. In our case, secondary dimensions are represented by context-awareness related concerns.

Aspect-Oriented Programming (AOP for short) [5] is one of many technologies resulting from the effort to modularize crosscutting concerns. The goal of AOP is to decouple those concerns, so that the system's modules can be easily maintained, evolved and seamlessly integrated. To do that AOP introduces a set of concepts:

- Join Point is a well-defined point in the program flow (for instance a method call, an access to a variable, etc)
- Point-Cut selects certain join points and values at those points.
- Advice: Advises define code that is executed when a point-cut is reached.

The program whose behavior is affected by aspects is called base program. A join point specifies a point in the execution of the base program that will be affected by an aspect. One or more of these join points (from one or different classes) are identified by a pointcut in the aspect layer, associating it with an advice. In this way, when one join point, referred in a pointcut, is reached in the program execution, the additional code, defined in the proper advice is executed. The aspect's code is composed of advises and the pointcuts where those advises must be applied.

5 Our Approach

Considering the negative effects of embedding adaptation code into the core application code, it is necessary to define an architecture which enables the separation between system modules and those that realize the adaptive personalization functionality. At same time, this separation is useful as it allows a correct integration of the different system's views, ideally in a transparent manner from the core application point of view.

Such an architecture will provide a set of advantages, among them we found:

- Extensibility: since each view of the system is independent from one another, they can evolve independently.
- High abstraction level: since the personalization features are isolated from the rest of the system, it allows the designer/programmer to focus in the core application, regardless secondary views such as personalization features.

In this work we propose the use of aspect oriented techniques in order to properly separate the core application components from those aimed to personalization. In order to get such a separation we have identified the main components, their roles and relationships, and defined the foundations of a software architecture that combines both objects and aspects.

5.1 Architecture's Main Components

A personalized mobile application can be divided into two dimensions or views. The first one is where the base application belongs to, that is to say, where the functional requirements are implemented. The second one comprises the non-functional requirement of personalization and its implementation. More views can be modelled as needed but, as far as this work is concerned, two views or dimensions will be enough.

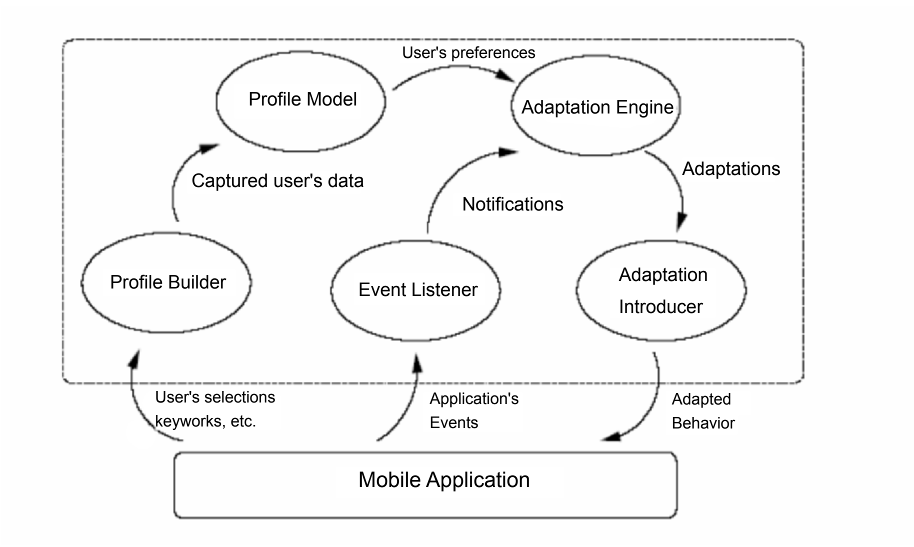

Fig. 2. Separation between the two main dimensions, base application and personalization

Figure 2 presents a layout of these components, the mobile system is divided into two parts. The core mobile application itself which is operative, independently from the personalization layer, it implements the functional requirements of the system. Modules located at this layer provide the main system functionality.

The second layer corresponds to a metalevel, where the personalization feature is reificated. This metalevel is the part of the system in charge of gathering user preferences, storing them in a proper way for later retrieval, and instrumenting the execution of the underlying mobile application, adapting its behavior to meet those preferences.

It is important to note that, since the base mobile application is completely functional and independent from the personalization metalevel, there is no interaction from the application towards the personalization level. This independence is a benefit that starts at the design phase, since it allows the designer to concentrate in the core functionality, abstracting him from those details related to personalization.

It is worth to note that in our approach we suggest to settle the personalization functionality on the client side, that is, in the mobile device. This characteristic makes this approach different from other works on personalized services for mobile devices. Client side personalization makes the system *network failsafe*, since it does not rely on the server to provide the personalized behavior. Due to instability of wireless connections in mobile contexts it is common to face situations where there is no connectivity. That's why we argue that personalization should be located at the client side. Since information gathering, storing and adaptation mechanisms regarding personalization are implemented at the client side, it is possible to cope with offline situations, and keep providing personalized response to the user.

Back to the architecture, the personalization dimension is formed by the following components:

- Profile Model: This component is in charge of storing user preferences.
- Adaptation Engine: The engine is responsible for inferring the kind of adaptation that should be done, it is done using the information stored in the profile model.
- Profile Builder: This component is in charge of intercept certain application execution points in order to feed the profile model with information about the user,
- Event Listener: This component comprises a set of aspects that detect the occurrence of certain application events. These events can be seen as triggers of adaptive actions.
- Adaptation Introducer: Once an interesting event has been detected, and the proper kind of adaptation identified, this component controls the application behavior adding the planned adaptation. This is done through aspects that can introduce behavior in the application.

Figure 3 shows the mapping between architecture components and a potential application design. Graphical notation is an UML variant [7], which denotes aspectual concepts through stereotypes. Advised methods are pointed by <<pointcut>> relationships. As it is shown in Figure 3 the base application is intercepted in those methods related with the user interaction by using the proper pointcuts. This interception is performed by the ProfileBuilder component, which gathers information about the user profile and passes it to the ProfileModel component. At the same time, the EventListener catches those events that can trigger some kind of adaptive behavior, and notifies the AdaptationEngine, which decides the adaptation type to be done. These adaptations are introduced in the application by the AdaptationIntroducer, where pointcuts are defined on those application parts where adaptation make sense to be done. Generally, suitable joint points are user interface events.

5.2 On the application and the personalization model

The link between the application model and the personalization one is done transparently by an aspectual layer. Aspects located in this layer are responsible for three key personalization activities:

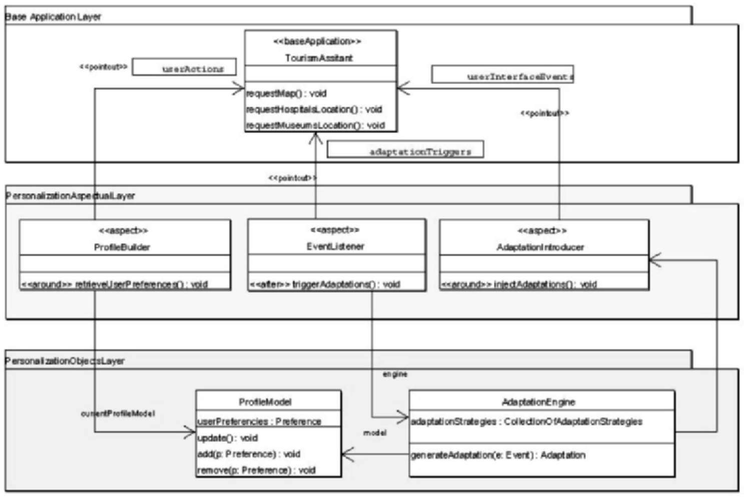
Base Application Layer
<<baseApplication>>
TourismAssitant
requestMap() : void
requestHospitalsLocation() : void
requestMuseumsLocation(): void
<<pointcut>>
userActions
userInterfaceEvents
<<pointcut>>
adaptationTriggers
<<pointcut>>
PersonalizationAspectualLayer
<<aspect>>
ProfileBuilder
<<around>> retrieveUserPreferences() : void
<<aspect>>
EventListener
<<after>> triggerAdaptations() : void
<<aspect>>
AdaptationIntroducer
<<around>> injectAdaptations(): void
PersonalizationObjectsLayer
engine
currentProfileModel
ProfileModel
userPreferencies : Preference
update(): void
add(p: Preference) : void
remove(p: Preference) : void
model
AdaptationEngine
adaptationStrategies : CollectionOfAdaptationStrategies
generateAdaptation(e: Event) : Adaptation

Fig. 3. Object-Aspect Oriented Architecture

1. Interception of user actions in order to gather information regarding his preferences, usual actions and so on.
2. Detection of contextual changes (through application events) that require personalization actions.
3. Application behavioral adaptation, based on the user profile, in order to provide a better response for the user.

The first activity is aimed to collect information regarding user preferences and common activities. The interception is done by aspects that extract parameters entered by the user, for instance the words used in an web search, the preferred order in a listing, or the selected option from a set. With this information it is possible to build the user profile. For instance, in an application that allows the user to search information using keywords, they can be extracted and used to build a semantic web, relating them with other terms. Keywords can be also used to analyze frequency and to find information patterns. Then, the derived information feeds the user profile. This kind of activities also includes:

– Measures of time spent in deferent parts of the application: this can be an indicator of special interest in some service or information offered by the system. The frequency of some kind of input can be also used to detect interest.

– Detection of common user choices. When some option is selected among several ones, in a repetitive way, it is an indicator of interest in that information or service. Frequency and repetition are indicators of user preference.

The adaptation process then consist in the instrumentation of the system in order to intercept those events that trigger internal personalization mechanisms. Eventually, these mechanisms will produce some behavioral changes in the application. For example, given a tourism mobile application, which aids the user showing the list of interesting places to visit, there will be events automatically triggered by the GPS. It is possible to react to those events showing the user new places to visit, basing this suggestion on the information gathered regarding his preferences.

All this behavioral alteration is introduced by means of aspects which can affect the normal control flow of the application. In this way the application is oblivious regarding adaptations, since they are introduced transparently in any application join point.

6 Example

To illustrate these ideas, we present as an example a mobile application that will be personalized using aspects. We will show how the adaptation is made, once the user's profile is built.

6.1 Definition

The application is a kind of tourism guide, for a tourist in a Buenos Aires journey, using a PDA as his assistant. This user is interested in local folklore, food, music and traditional dances. He is familiarized whit Argentina's history and also is a sport fan, specially soccer.

This person has a tour of places to visit and wants to be notified when he is close to some place that may interest him, but do not belongs to the original tour.

The tour crosses the neighbourhood of La Boca, the Plaza de Mayo, and the Obelisco. The tour starts in La Boca, so his PDA will show him information about this neighbourhood.

As he walks the La Boca's streets, he get close to the well known soccer stadium La Bombonera, that is not registered in his tour. Nevertheless, the system recognizes that the user is interested in soccer and notifies him that he is close to the famous stadium.

6.2 Functionality Distribution

In this case, the tourism guide is the base application on which the adaptation of the user's preferences will be made. This application is fully functional independently of the personalization capabilities.

The profile modeling has been already discussed in [4] and [2]. These models can be adapted to fulfill the Profil*e* Mode*l* role in Fgure 2, so that we do not analyze this topic, and concentrate in the adaptation topic. The Adaptation Engin*e* decides which adaptations have to be introduced in the application when an event occurs. The engine can use different technics to infer the adaptation such as semantic nets, neuronal nets, agents, etc. These topics have already been discussed in [9], that work shows different ways to filter information considering user's preferences. Since filtering technics are not the objective of this paper, we only will focus on the role that fulfills the aspects, how they relate with the base application and the components that implements the personalization (*ProfileModel* and *Adaptation Engine*)

The aspects that implements the *Profile Builder* component intercepts execution points, as those described in Section 5.2. From the keywords collected from searches and user chosen options, information that defines his preferences is captured. This information that is captured automatically without disturbing the user, is known as implicit construction. The aspect is in charge of intercepting the methods that implements the search and the selections, in order to inspect the values entered by the user. Since this information capture is done using AOP provided constructions (join points, point cuts, and advises), the base application does not need to implement any behavior related to information capture in order to build the profiles.

There also exists what is known as explicit profile construction, where the user express his preferences by filling forms. This information complements and feeds the profile with the information captured automatically by the aspects.

Once the information is captured within the *profile model,* the personalization layer is ready to make the adaptations. This adaptations begin with the detection of some event by the *Event Listener* component which is, in fact, an aspect. This aspect intercepts the application's control flow in order to detect the events that launch the adaptations. In the example, an aspect can intercept the geographic position change notification, notifying the *Adaptation Engine* that the user's geographic position has changed.

The *Adaptation Engine* finds that the position is close to the soccer stadium and since the *Profile Model* holds information that allows to establish that the user is a sport fan, the adaptation engine decides to launch an adaptation of the suggestion kind.

The aspects that implements the *Adaptation Introducer,* intercepts, through pointcuts, all the interface actions, so when a user generated event occurs, the suggestion to visit the stadium is presented to the user. Closing the adaptation process cycle.

7 Conclusions and Future Work

The application of technology that allows the advanced separation of concerns, like *Composition Filters, Subject Oriented Programming* and like this case Aspect Oriented Programming, in general produce higher levels of modularity. The benefits of modularity, well known in computer science, includes flexibility, maintainability, design and implementation clarity.

In this work have presented general foundations of an architecture that allows to isolate in a effective way the application core from such not functional concerns related with the personalization. We have defined the essential aspects and join points that allows to establish the connections between the architecture components. The presented proposals have foundation in previous experiences and are in implementation phase, we trust that the result will give support to the exposed ideas. We have also analyzed the impact of doing the adaptations in mobile gadgets, considering hardware limitations present in mobile computing.

The use of aspects to adapt the behavior of mobile applications is novel, and follows the ideas presented in [11]. Still remains the study of the integration of the architecture presented in this work in a way that incorporates the elements of context awareness as they were explained.

The implementation of prototype applications that materialize this ideas, will help to make a concrete evaluation of the benefits reached by applying aspect oriented programming in mobile systems.

References

1. J. Blom. Personalization a taxonomy. In CHI 2000 Workshop on Designing Interactive Systems for 1-to-1 Ecommerce, 2000.
2. W. W. W. Consortium. Composite capabilities/preference profiles, 2001.
3. S. Herrmann and M. Mezini. PIROL: A case study for multidimensional separation of concerns in software engineering environments. In OOPSLA, pages 188–207, 2000.
4. G. Kappell, B. Prll, W. Retschitzegger, and W. Schwinger. Customisation for ubiquitous web applications. In Int. Journal of Web Engineering and Technology (IJWET), Inaugural Volume, Inderscience, volume 2299. Publishers 2003, 2002.
5. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loingtier, and J. Irwin. Aspect-oriented programming. In Mehmet and S. Matsuoka, editors, 11th Europeen Conf. Object-Oriented Programming, volume 1241 of LNCS, pages 220–242. Springer Verlag, 1997.
6. C. Mesquita, S. D. J. Barbosa, and C. J. P. de Lucena. Towards the identification of concerns in personalization mechanisms via scenarios. In AOSD 2002, Workshop on Early Aspects, 2002.
7. R. Pawlak, L. Duchien, G. Florin, F. Legond Aubry, L. Seinturier, and L. Martelli. A uml notation for aspect-oriented software design. In AO modeling with UML workshop at the AOSD 2002 conference. Proceedings, 2002.
8. L. A. R. Rui Alexandre P. P. da Cruz, Francisco J. García Peñalvo. Perfiles de usuario: En la senda de la personalización. Technical report, Departamento de Informática. Universidad de Salamanca, 2003.
9. S. Stewart and J. Davies. User profiling techniques: A critical review. In 19th Annual BCSIRSG Colloquium on IR. Springer Verlag, 1997.
10. M. Wagner, W.T. Balke, R. Hirschfeld, and W. Kellerer. A roadmap to advanced personalization of mobile services. In 10th International Conference on Cooperative Information Systems, 2002.
11. A. Zambrano, S. Gordillo, and I. Jaureguiberry. Aspect based adaptation for ubiquitous software. In International Workshop on Information Retrieval. Mobile HCI, 2003.