

Combining Components, Aspects, Domain Specific Languages and Product Lines for Ambient Intelligent Application Development

Lidia Fuentes, Daniel Jiménez

ETSI Informática Málaga, Blvd. Louis Pastour 35, 29071 Málaga, Spain.
Departamento de Lenguajes y Ciencias para la Computación
Universidad de Málaga, Spain
{lff, priego}@lcc.uma.es

Abstract. Currently, we can see how different kinds of experimental Ambient Intelligence (or AmI) applications are being developed to solve very diverse problems. These applications will soon be widely used by ordinary people making their lives easier and more comfortable. As time passes, more and more new portable and wearable computational devices will appear and will demand that AmI applications be adapted to them. The first challenge is therefore to define advanced mechanisms to perform a custom-made installation of AmI applications in devices with different capacities. The second one is to provide mechanisms to adapt the application when the technology, the environment or the user demands change. To fulfil this goal, we have developed an Ambient Intelligence platform, called AOPAmI¹ that is based on the use of Components, Aspects, Domain Specific Languages and Product Lines software technologies.

1 Challenges addressed

The development of Ambient Intelligence (or AmI) applications is complex and difficult due to several problems related to the nature of AmI applications [6].

The first problem is hardware heterogeneity that makes it difficult to develop and deploy AmI applications for new devices. Portable devices have different capabilities and constraints such as the amount of main memory, the kind and amount of storage memory, the communication capabilities, the computation capability or the CPU type. So, the first challenge is to define a mechanism for the custom-made installation of AmI applications on portable devices according to their specific hardware constraints.

The second problem addressed, is related to the application evolution management. AmI applications have to deal with change, since the hardware and software technologies for AmI is continuously evolving. For example, it should be easy to update existing applications with new multimedia formats, new communication protocols or

¹ This work is partially financed by IST-2-004349-NOE AOSD-Europe and the Spanish Ministry of Technology and Science, CICYT, under grant TIN2005-09405-C02-01.

location systems. Dealing with such quick evolution makes software reuse a key strategy.

Finally, the third problem is related to the dynamic nature of the environment where applications are executed. AmI applications should be designed to be able to react in a more or less automatic and unassisted way to changes in the environment. However, due to memory and execution restrictions, it is impossible to code the behaviours for each possible external event occurring in the environment. Thus, it is necessary to provide a mechanism that allows the dynamic adaptation of the application to new events and conditions. We propose the AOPAmI² platform (Aspect-Oriented Platform for AmI) that attempts to solve these problems combining several advanced software development technologies, described in the next section.

2 Technologies used by the AOPAmI platform

AmI applications are distributed applications, but with specific adaptability requirements due to the special nature and diversity of the hardware in which the applications are executed. On the other hand, AmI applications typically have to react to a multitude of external events inside a changing environment. So, a key issue is to achieve a functional decomposition of AmI functionality around truly autonomous components, which are able to evolve and be composed in an independent way.

The Component Based Software Development [9] promotes the encapsulation of application functionality in different independent modules called components. Thus, an application will be developed by selecting and gluing a set of house-made or commercial-off-the-shelf (COTS) components. However, we notice that the use of component technology alone is not enough to achieve proper software evolution management. The problem is that because some properties, which usually crosscut several components, are tangled with the components' functionality at the code level [4], it is very difficult to develop truly independent and reusable components

As a consequence in our work, we have try to improve software modularisation using a very promising technology known as AOP [4]. AOP proposes modelling cross-cutting concerns as a new entity called *aspect*, which is considered a first order entity inside the application. The distinguishing characteristic of AOP is that aspects are executed and *weaved* (composed) *obliviously* in relation to the base units, in our case components. Consequently, both aspects and components are able to evolve independently by themselves, in addition to being more reusable and easier to replace. AOP promotes the separation of concerns only at the code level [4], but the Aspect Oriented Software Development [1] proposes to use aspects at any stage of the software life cycle. For instance, in [3] an Architectural Description Language (DAOP-ADL) describes the weaving between components and aspects. In AOPAmI we continue with this approach specifying components, aspects and weaving information at the architectural level. By doing so, the software architect will perform changes affecting the application logic in the ADL, without modifying the application code.

² This work is partially financed by IST-2-004349-NOE AOSD-Europe and the Spanish Ministry of Technology and Science, CICYT, under grant TIN2005-09405-C02-01.

The decision of adopting the AOP technology was proven to be advantageous, but it is not enough to build AmI applications, because we need to address variability event at runtime. In order to be able to reason about the entities executed in the application and to take decisions we have decided to use the Domain Specific Languages [2] (or DSLs). These languages are used to describe entities modelling concepts of specific domains and the operations available to these entities. Using these DSLs, we simplify the programming of specific application features such as communication, security or resource management and express them using a high level language. For example, in AOPAmI we use a DSL to define policies for choosing which communication protocol should be used when several alternatives exist, or component replacement strategies when a particular event occurs [7].

Finally, regarding how to handle the development of several versions of the same application for different devices with heterogeneous capabilities, we use the Software Product Lines (or SPLs) [2] technology. A software product line defines the possible variability points inside the application and the commonalities. In these variability points, components and aspects are combined in order to compose a new application version.

3 Previous and Ongoing Work

Since some AmI platforms already use the component technology, our first task was to refactor a well known component-oriented AmI platform, specifically [8]. This work helped us to identify particular aspects of AmI applications and to validate the benefits of applying aspect technologies to this application domain [5]. Additionally, this work show us the needed of providing event more flexibility to the AmI applications. This flexibility should be accomplished both at the design and at the execution level by using respectively SPLs and DSLs.

Regarding the use of DSLs, we presented in [7] an initial version of the platform configuration language, and we also outlined how the SPLs can help to address the custom-made installation problem in portable devices. In [10] we presented the definition of the AOPAmI platform and validated our contribution by an exhaustive comparison with the PCOM platform. Currently we are planning to refactor an urban traffic management application (iTransit) of Dublin in order to validate more extensively our proposal.

4 Conclusions

We address the problems presented in the first section by the combined used of the four technologies mentioned above.

Firstly, the hardware heterogeneity problem could be alleviated by the use of components and aspects, and by defining a product line of AmI platforms. By doing so, we define a homogeneous platform that hides the hardware dependences in the applications. Consequently, it is possible to develop different versions of the same application for each device, but maintaining only one application design.

Secondly, the platform adaptation to changes in requirements or in technology is addressed again by the combined use of components, aspects and SPLs. The SPL describes the application variability points and the components and aspects provide a specific implementation and/or composition for these variability points.

Finally, the dynamic adaptation of the application to the environment is achieved in two ways. First, the use of aspects allows us to enable or disable application features such as trace or security, even at runtime. Secondly, the use of ADLs and DSLs in XML will allow the programmer or even the user modify the application behaviour using a high level language that is interpreted by the device. The rules coded by this language can also be changed at runtime.

The main reason behind the development of this platform is to provide a way to handle the complexity and evolution of AmI applications. This subject will almost certainly become critical with the widespread use of these applications. Much effort has been put in making the last technology AmI application, but very little effort has been put in reusing the designs and patterns used to build them in a coherent and homogeneous way. As a consequence this work is a practical application of an old lesson learned from software engineering in the past. It is better to use the time in perform a good application design instead of using it in coding the final application.

References

1. AOSD Web Site. <http://www.aosd.net>
2. Batory, D., et Al.: Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study. *ACM Transactions on Software Engineering and Methodology*, Vol. 11, n° 2, (2002) 191-214
3. Pinto, M., Fuentes, L. and Troya J.M.: "A Dynamic Component And Aspect-Oriented Platform", *Computer Journal*, 48(4). 401-420
4. Kiczales, G., et Al.: Aspect-Oriented Programming. *European Conference on Object Oriented Software Development (ECOOP'97)*, Jyväskylä Finland (1997)
5. Fuentes, L., Jimenez, D. and Pinto, M.: Experiences Refactoring Ambient Intelligence Applications with Aspects. *LATE workshop, AOSD (2005)*
6. Fuentes, L., Jimenez, D.: An Aspect-Oriented Ambient Intelligence Middleware Platform, In *proceedings of 3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2005)* Grenoble France (2005)
7. Fuentes, L., Jimenez, D., and Pinto, M.: An Ambient Intelligent Language for Dynamic Adaptation, *Object Technology for Ambient Intelligence workshop (OT4AmI)* Glasgow Uk (2005)
8. Becker, C., et Al.: PCOM – A Component System for Pervasive Computing. *Second IEEE International Conference on Pervasive Computing and Communication. (PerCom'04)*, Orlando USA (2004)
9. Szypersky, C.: *Component Software. Beyond Object-Oriented Programming*, Addison-Wesley, (2002)
10. Fuentes, L., Jimenez, D., and Pinto, M.: Development of Ambient Intelligence Applications using Components and Aspects. *I Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI'2005)*, Thomson, ISBN 84-9732-442-0, Granada Spain (2005)