Efficient Modelling of Highly Adaptive UbiComp Applications

Andreas Petter, Alexander Behring, Joachim Steinmetz
University of Technology Darmstadt
Telecooperation Group, Hochschulstr. 10, 64289 Darmstadt, Germany
$\{a\_petter, behring, joachim\}$@tk.informatik.tu-darmstadt.de

Adaptive applications in ubiquitous computing depend on large sets of parameters while requiring device independency. To efficiently develop such applications, we discuss a possible solution, we call "amending models".

1 Introduction

Ubiquitous computing introduces new challenges for software engineering, among these are support for a large variety of platforms and numerous different contexts. By leveraging modeling, a great degree of abstraction from the underlying platform is achieved. By interpreting models at runtime, the abstraction is further improved. But still there is the need to go into different contexts of usage, which would have to be modeled explicitely. Therefore an approach where models are adapted to changing contexts at runtime and contexts not known at development time can be dealt with is sought after. Ideally such an approach also reduces the development efforts associated with the development of ubiquitous computing applications.

Amending Models are a possible solution to this question. They especially allow modelling for applications adapting to numerous different contexts. Amending Models address this problem by enabling an application designer to model only parts of his model and let other model elements, due to introduction or change of context, be explored and modelled automatically during runtime.

In the next section, we briefly explain what an amending model is and argue against and for it in sections 3 respectively 4, also taking into account the effects of its application. Section 5 scetches a possible setup and section 6 concludes and gives further perspectives on this field.

2 The Thesis

Building on the ideas expressed in [1] and [7], we argue for utilizing and interpreting models at runtime and amending them by leveraging *Artificial Intelligence (AI)* concepts in the adaptation process. Further, we propose that

to develop highly adaptive software for ubiquitous applications in a cost effective manner, a standard modeling approach is not sufficient. The concept of amending models is a feasible approach to adress this problem.

An *Amending Model* hereby is a graph with nodes (e.g. tasks, UI elements, ...) carrying attributes and the possibility to link the nodes through edges (e.g., in a task model, UI model, ...). An initial setup (a start model) is given by the developer. During runtime, the application is adapted by amending its model(s). This is done through applying AI algorithms and thereby, e.g., changing the links between and the usage of nodes, as well as inserting and removing nodes. The adaptation process hereby takes into account the *Amendment Context*, which includes all information that could be relevant to amend the model.

By reducing the number of different Amendment Contexts (e.g., different available sensor systems) that are modeled explicitly the adaptation process must be able to cope with by then unknown parameters. Therefore, standard model transformations alone aren't suitable to transform Amending Models. AI algorithms are a possible way of dealing with such a situation.

In order to produce models compliant with the specification of the system, validation and evaluation become an essential part of the adaptation process. This compliance could be achieved by, e.g., choosing suitable amendment methods or validation of the model.

3 Arguing against Amending Models

Validation consequently is an integral part of amending models. Unlike in current *Software Engineering (SE)*, (partial) validation must be done automatically at runtime. This *introduces new complexity and the need for special knowledge* how to write validation rules. However, if validation rules can be generated automatically from e.g. a goal model, this drawback is greatly reduced.

To reduce the number of models explored at runtime for one adaptation step, heuristics seem most suitable. *Formulating heuristics can be difficult* [4] and might result in the need for AI experts. This drawback could be reduced by improved tool-support, potential reuse of heuristics and results of further research.

Modeling might lead to *unpredictability*, which in turn results in *reduced usability*, e.g., for user interfaces [9]. Amending Models intensifies these problems, because the Amendment Context is not modeled explicitly. Besides improving the adaptation process itself, *model templates* could provide predefined model parts for certain Amendment Contexts (e.g., UIs for the most used devices) to regain predictablity. Furthermore, there has been improvement in this area, e.g., [2], [8] and [7].

4 Arguing for Amending Models

Modeling highly adaptive applications with a traditional approach implies to model every possible Amendment Context (e.g., for User Interface Adaptation in [3]). Amending Models loosen this restriction by providing an abstraction and allowing to model only the key ingredients of an application.

Design flexibility is gained through opening up a trade off between the effort of modelling elements and computational power at runtime. Closely connected to this is the decision of the developer to spend effort in modeling a specific Amendment Context (e.g., UIs for a certain device) or relying on the adapatation process.

By reducing the sheer number of elements to model, development time is reduced. By the same effect, a better overview over the application model is gained. This can improve the effectiveness of the development and reduce bugs. Consequentially, *development costs are reduced* through the application of Amending Models. Sometimes this cost reduction could be the prerequisite to start developing in the first place.

In industry software projects change requests (in all phases of the software lifecycle) are common. Especially changes affecting wide parts of an application are difficult to implement. Amending Models reduce the effort needed to implement a change request by abstraction from the Amendment Context. This abstraction also results in improved reusability. Consequently, in fortunate cases, a change request could be solved by writing a single transformation rule. Additionally, since Amending Models are interpreted at runtime, such a change might even be implemented after deployment of the application. Overall, *maintainance costs are reduced*.

Recapitulating, through Amending Models the *adaptivity of applications is improved*. An important aspect of this is improved *Plasticity* [11], denoting the size of the Amendment Context an UI stays usable in.

5 Action Example

In [1] an software architecture reference model for UI adaptivity is discussed. In the following, a brief description of a set of development methods and tools that could be used is given:

- adapted MDA (see [6]) concept and pattern
- MOF (see [10]) compliant metamodel similar to UML
- specific tools for abstract and concrete UI design
- integration of domain specific languages

A setting like this would leverage the spread of UML wihin software development companies and the concepts of MDA. The metamodel could be downsized and taylored to the specific needs of the software company using it. At the same time, it enables the usage of powerful domain specific languages. Finally Myers concerns [9] are adressed by integrating special UI design tools to reduce unpredictability. Thus setting is aimed to facilitate cost-effective development of context-aware, adaptive ubiquitous computing applications.

6 Conclusion and Outlook

We argued for *Amending Models*, which are modified and interpreted at runtime in order to adapt to different Amendment Contexts. By using this concept, typical challenges of Ubiquitous Computing are addressed and

- *development costs are reduced*, by reducing development time and improving effectiveness,
- *design flexibility is gained*, by opening up a trade-off between invested effort for detailed modeling of possible Amendment Contexts versus more abstract modeling, and
- *adaptivity of applications is improved* by reducing direct dependence of models on parameters.

Further research within this area may adress:

- identification of possible transformation processes,
- characterization of tools used for modelling with this paradigm, and
- detailed characterization of relations between different elements in this concept.

Some of these topics will be addressed within the eMode project [5].

Bibliography

[1] L. Balme, A. Demeure, N. Barralon, J. Coutaz, and G. Calvary. Cameleon-rt: A software architecture reference model for distributed, migratable, and plastic user interfaces. In *EUSAI*, pages 291–302, 2004.

[2] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, N. Souchon, L. Bouillon, M. Florins, and J. Vanderdonckt. Plasticity of user interfaces: A revised reference framework. In *TAMODIA 2002*, pages 18–19, Publishing House Bucharest, Romania, 2002. INFOREC.

[3] B. Collignon. Dégradation harmonieuse d'interfaces utilisateur. Master's thesis, Universite Catholique De Louvin, 2004.

[4] R. S. Engelmore (ed.). Knowledge-based systems in japan. Technical report, JTEC, Baltimore, MD, USA, 1993.

[5] EMODE. `http://www.emode-projekt.de`.

[6] J. Mukerji (ed.) J. Miller (ed.). Mda guide version 1.0.1. Technical report, OMG, 2003.

[7] J.-S.Sottet, G. Calvary, and J.-M. Favre. Towards model-driven engineering of plastic user interfaces. In *MDDAUI*, 2005.

[8] G. Mori, F. Paternò, and C. Santoro. Tool support for designing nomadic applications. In *IUI '03*, pages 141–148, New York, NY, USA, 2003. ACM Press.

[9] B. Myers, S. E. Hudson, and R. Pausch. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3–28, 2000.

[10] OMG. Mof specification, 2002.

[11] D. Thevenin and J. Coutaz. Plasticity of user interfaces: Framework and research agenda. In *Interact'99*, volume 1, pages 110–117. IOS Press, 1999.