MDA-BASED MANAGEMENT OF UBIQUITOUS SOFTWARE COMPONENTS

Franck Barbier & Fabien Romeo
PauWare Research Group – Université de Pau
Av. de l'université, BP 1155, 64013 Pau CEDEX – France
Franck.Barbier@FranckBarbier.com, Fabien.Romeo@univ-pau.fr

Introduction: the Model-Driven Architecture (MDA) [1] software engineering paradigm aims at considering "models" as first-class products within a software development process. More precisely, this new approach is based on model transformation in which the generated code (in a fairly abstract or detailed form) is just an "implementation model". Concretely, MDA is strongly influenced by the Unified Modeling Language even if other modeling techniques are acceptable. MDA advocates Platform-Independent Models (PIMs) and Platform-Specific Models (PSMs), the later resulting from transformations of the former. In this optic, it becomes natural to focus on PSMs which specifically target ubiquitous applications (see for instance [2]).

In a MDA approach, the need for model checking is often based on "executability" [3]. So, the modeling language used, if "executable", enables model simulation. Such an activity occurs at development time and encompasses, of course, model checking activities, but also testing activities if models include technical details which closely refer to deployment platform properties. In the area of ubiquitous computing, deployment platforms have special features. A relevant research statement is therefore the look for MDA concepts, techniques and tools that comply with the development and the inner nature of ubiquitous applications. For instance, if one is able to provide different executable models which correspond to distinct software component types, how then to deploy and run these models/components in wireless and mobile devices? How to protect these models/components from instable communication, a key characteristic of wireless and mobile platforms? How to endow these models/components with autonomic features (self-managing, self-healing, dynamical reconfiguration…) since controlling runtime conditions is more difficult in ubiquitous systems compared to common distributed systems? Etc.

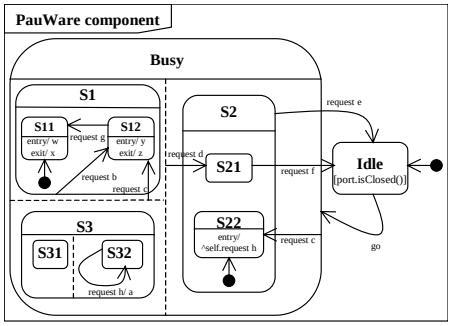
This experiment paper proposes a MDA-compliant execution engine called *PauWare*. (www.PauWare.com/PauWare_software/). This tool is mainly composed of a Java library which enables the simulation of *UML 2 Sequence Diagrams* (*i.e.*, scenarios) and *UML 2 State Machine Diagrams*, a variant of Harel's Statecharts. The *PauWare.Velcro* sub-library is a J2ME-compliant (Java 2 Mobile Edition) tool which supports the design of the inner workings of software components by means of Statecharts.

We stress in this paper the problem of remote management of software components embedded in wireless and mobile devices and, in certain cases, the possibility of equipping such components with self-managing characteristics [4]. We think that ubiquitous software components and applications require larger management capabilities. Management relies on dedicated infrastructures like, for instance, Java Management eXtensions (JMX) [5]. Despite the availability of management standards, there are few techniques that explain how to instrument the dynamical reconfiguration of components running in remote devices. What could mean self-healing and how one may implement it? Etc.

In *PauWare*, supporting dynamical reconfiguration leads to forcing the state of a component to a well-known stable consistent "situation". For instance, in the figure below, one may go (or go back) to the *Idle* state in bypassing the "normal" behavior of the component. The stable consistent nature of a statechart is a modeled state[1], *Idle* here plus some invariants that can be

[1] Several nested and/or parallel states are also possible in conformance with all of the power offered by the Statecharts modeling technique.

checked at runtime (a port must be closed). States of components are modeled at development time but are also explicit at deployment time since models persist at runtime. The execution model of UML 2 is a run-to-completion model, meaning that component clients' requests are queued and cannot interrupt the current processing, if any, of a request.
Management services may therefore be incorporated into a configuration interface. For concrete reasons, such a facility currently relies in *PauWare* on JMX and on the Wireless Message API (WMA) for communication. Self-management is a more tricky problem. A component may aim at itself deciding to launch a self-configuring operation, "reset" for instance, namely in case of fault recovery: this is typically self-healing. We propose a (parameterized) rudimentary mechanism which is a kind of "undo". If the "autonomic" flag is turned on, a component automatically tries to roll back the current transaction (a global transition from the stable consistent context raising a problem to the immediately prior one) in case of fault detection. Roll back may succeed but it may also fail because many internal business operations (see for instance *x*, *y*, *z* and *w* in the figure below) are executed within a run-to-completion cycle. Canceling the effect of such operations is not always possible. State invariants therefore help to establish if roll back succeeded: all state invariants attached to all nested and/or parallel states must be true when returning to these states.
Conclusion: we think that ubiquitous applications and components require self-adapting capabilities. We nevertheless observe that a gap between theory and practice still remains. While the notions of self-management, self-adaptation are evident and may have several formal shapes, few means currently exist for supporting these concepts in ubiquitous platforms. In the global world of software engineering, MDA put models forward. In such a context, we exploit the power of reputable models like Statecharts to implement self-adaptation.



1. Mellor, S., Scott, K., Uhl, A., Weise, D.: *MDA Distilled – Principles of Model-Driven Architecture*, Addison-Wesley (2004)
2. Grassi, V., Mirandola, R., Sabetta, A.: *A UML Profile to Model Mobile Systems*, Proc. «UML» 2004, LNCS #3273, Springer, pp. 128-142, Lisbon, Portugal, October 11-15 (2004)
3. Mellor S., Balcer, S.: *Executable UML – A Foundation for Model-Driven Architecture*, Addison-Wesley (2002)
4. Barbier, F., Romeo, F.: *Administration of Wireless Software Components*, Proc. ETSI/MOCCA Open Source Middleware for Mobility Workshop, European Telecommunications Standards Institute, Sophia-Antipolis, France, April 6 (2005)
5. Kreger, H., Harold, W., Williamson, L.: *Java and JMX – Building Manageable Systems*, Addison-Wesley (2003)