Rapid prototyping of real-time reactive systems

Konrad Kułakowski
Institute of Automatics,
AGH University of Science and Technology,
al. Mickiewicza 30, 30-059 Kraków, Poland,
e-mail: kkulak@agh.edu.pl

Marek Kostrzewa
Deltavista AG,
Gustav-Siber-Weg 4, 8700 Küsnacht, Switzerland,
m.kostrzewa@deltavista.com

***Abstract*—The growing popularity and common usage of real-time reactive systems on various domains force activities that lead to the improvement of the development stage by applying several software engineering methods e.g.: *MDA (Model Driven Architecture)* or executable modelling, which could provide a working prototype of a system in the early stage of development. It is also required to have some tools that enable the designing, testing, simulation and running of the software preserving time constraints and interacting with its environment. *RAT (Reactive Appliance Toolkit)* is a kind of software that allows for the creation of an executable model of a real-time reactive system in *UML* and *Real-Time Java* and, simulates its behavior and verifies time constraints.**

I. Introduction

Reactive systems are more and more popular as a platform for building reliable applications. Many of them, by nature, are real-time systems. This leads to the need to define real-time reactive systems as a separate research subject. Correctness and a guarantee that all the deadlines are met for these systems are important and challenging. For this class of system, timeliness becomes a one of the key factors determining the level of quality of service. Assuring appropriate *QoS* might be hard since a traditional approach may require a lot of manual effort, like code reviews, writing unit tests and communication efforts. Errors might be introduced during elaboration on the design to a code. Examples of solutions that shorten the distance between design and ready to use software are *MDA (Model Driven Architecture)* and *Executable Modelling* [12]. This technique allows for the creation of a system as an architectural abstract *PIM (Platform Independent Model)*, and then allows them to be run or deployed in some executable form into the target environment *PSM (Platform Specific Model)*.

In this paper we would like to present a library *RAT (Reactive Appliance Toolkit)* [6] that facilitates *MDA* development of real-time reactive systems. On the *PIM* level we use *UML 1.4* with some real-time extensions, whilst the *PSM* is based on Real-Time Java.

The *RAT* presentation placed in section II is preceded by introduction (section I) covering some related topics and literature. Section III sums up the whole paper.

A. Related Works

There are various tools and methods for creating executable and translatable models of reactive and real-time systems. This section goes briefly through each area and concludes by showing where RAT fits in.

Tools for rapid prototyping are used by designers, developers and users as a way of viewing and checking a model at the early stage of the software development life cycle. An example of rapid prototyping tools is the language proposed by Luqi et al. [10]. Various other languages for modelling real-time systems have been proposed, e.g.: *RT-ASLAN* and its successor *ASTRAL* [2]. A separate group of specification languages of timed reactive systems is formed by various formal methods like: *Real-Time Petri Nets*, *RT coloured Petri nets* [16], process algebras: *Timed CCS*, *RT CSP*, *RT Lotos*, *RT ACP* and temporal logic. Most of these allow for simulation of a model and its automatic verification, but dedicated tools usually do not support code generation.

The most popular approach relies on using *the UML* modelling language to build a real-time system, which later on might be simulated and executed on the target platform. To support modelling real-time behavior in *UML,* a *UML Profile for Schedulability, Performance and Time* (SPT) [15] has been defined. Currently, there are plans to replace these with a *UML Profile for Modelling and Analysis of Real-Time Embedded Systems* [14].

A very promising direction seems to be *to* apply *the Model Driven Architecture* approach to modelling real-time software [9]. There are several examples of projects that use *MDA* and *UML* to build real-time software. One of these is the *CoSMIC (Component Synthesis using Model Integrated Computing)* project [4]. It is a tool suite for creation of *DRE (Distributed Real-Time and Embedded)* systems based mainly on *CORBA* architecture. It aims to support modelling and analyzing functionality and *QoS* requirements of *DRE* systems. Another example of a project employing *MDA* and *UML* technologies for real-time systems is *FUJABA* This tool supports incremental design and verification using *UML/RT*. The verification relies on transforming appropriate *RTSC*

(*Real-Time Statecharts*) into *ExHTA* (*Extended Hierarchical Timed Automata*); then to *HTA*, which is an acceptable input for the Uppaal model checker. An *MDA* modelling system generating executable code in *RT Java* is *HIDOORS* (*High Integrity Distributed Object-Oriented Real-Time System*) [11]. The central part of *HIDOORS* is the *HIDOORS UML Profile*, which is a subset of OMG's *UML Profile SPT* [15]. It is used by all the tools included in *HIDOORS* software tool suite. This project supports automatic code generation from models written in *HIDOORS UML Profile* to source code in *RT Java*.

The *RAT* presented in this paper is similar in some aspects to *HIDOORS*. It supports *UML* based visual modelling and transformation from a *UML* model into source code in *RT Java*. Some new artefacts such as stereotypes: *appliance* or *sensors*, are introduced. A general concept of *performance annotation* (*PA*) is taken from the *UML SPT* profile [15]. *PA* is used as a comment containing time constraints assigned to a specified appliance's state. Despite introducing some new concepts, system modelling in *RAT* is done in pure *UML*. For this reason almost any UML tool might be used to build a reactive real-time system in *RAT*.

B. Reactive Systems

The term 'reactive system' was introduced in [5] by David Harel and Amir Pnueli. It is a self-operation entity that possesses an internal state and interacts with environments. The role of reactive systems is to maintain ongoing interaction with their environment, rather than produce a final value upon termination.

Very often reactive systems are also in fact real-time systems since the actions that they perform may have a strictly limited execution time and system resources.

C. Real-Time Java Platform

One of the more and more popular runtime environments for real-time reactive systems is *RT Java (Real-Time Java)* [7]. The idea of the RT extension of *Java* came about in 1998. The first version of *RTSJ (Real-Time Specification for Java)* was available in June 2000. The final *RTJ API* was approved in November 2001. Since then, several *RTJ* distributions have appeared on the market. The major *RTJ* providers are TimeSys Corp., Aicas Software [1] and Sun Microsytems. There is also a non-commercial distribution called jRate, which extends the standard *GNU Java* compiler *GCJ*.

The most important features introduced by *RTJ API* are: real-time threads, new memory management schemes of RTJ, asynchronous events handling, asynchronous transfer of control, high resolution time and timers, direct access to physical memory.

D. UML AND Execfutable Modelling

UML is a modelling language for specifying, constructing, documenting and visualising artefacts of a wide range of complex systems, including software/hardware computer aided systems. The popularity of *UML* has its origins in unification (it combines commonly accepted concepts and ideas from many methods), universality (it can be used during all phases of development, from requirements to deployment, with most development processes) and genericness (it can be applied to a wide range of domains). There is a need for rapid prototype techniques that can assure that a developed system is correct (or not) as soon as it is conceived, without having to wait for the design or implementation. Among the other prototype techniques, the concept of executable modelling applied to *UML* seems to be very promising. An executable model, can be constructed, run, tested and modified in short incremental, iterative cycles; thus, it provides the ability for early feedback. This technique also excellently suits *MDA* methodology. Translation between *PIM* and *PSM* is replaced by constructing an executable model, where transition between the design and a machine dependent bundle is done automatically. Thus, it eliminates the labor-intensive task of transforming the *PIM* into *PSM* [3].

The executable *UML* defines data structure (class diagram), the behavior of instances over time (statechart diagram), and precise computational behavior (actions diagram). In order to execute a model, the following requirement should be satisfied: *each class has to be assigned its own state machine*. Each state machine has to get a set of procedures, each of which in turn contains a set of actions. These actions are triggered by the state changes in the state machines that cause synchronization, data access and functional computations to be executed.

II. Solution Overview

A *Reactive Appliance Toolkit (RAT)* is dedicated to the early stage of requirement analysis and verification of some aspects of real-time reactive systems. We put emphasis on the simplification of modelling a continuous real-time synchronous and asynchronous interaction with the environment, reaction on input depending on the actual system state, the parallelism of execution and the simulative verification of time constraints. In the first step, some *UML* diagrams have to be modelled and some preliminary values of time constraints have to be set.

This requires the use of any *UML* editor. We decided to used *ArgoUml* [1] as an *UML* editor, but there is no limitation to any one specific editor. Any editor can be used if it supports the following *UML* features:

- A metamodel of at least version 1.4;
- *XMI* (version 1.2);
- Stereotypes and profiles;
- Class diagram;
- Statechart diagram and comment;
- Activity diagram.

A ready *UML* model can be executed by the use of a *Reactive Appliance Toolkit*. During simulation, the assumed time constraints can be verified against execution delays introduced to the model.

[1] argouml.tigris.org

This toolkit consists of three components:

- API library;
- Code generator;
- Management console.

A *RAT* requires the installation of standard *Java* [2] and *Real-Time Java* (e.g. *Jamaica VM* [3], or *Sun RT Java*[4]).

A. API library

The *API (Abstract Programming Interface)* library is a core element of the *'Reactive Appliance Toolkit'*. It provides a set of concepts that could be useful for designers in modelling and executing reactive systems independent of the targeting software platform.
From an architectural point of view, the library can be split into three layers:

- Interfaces;
- Common implementation (independent of the real-time Java vendor's implementation);
- Dedicated implementation (the real-time Java vendor's specific implementation).

The separation of common implementation from dedicated ones simplifies the usage of different distributions of *Real-Time Java* implementations.

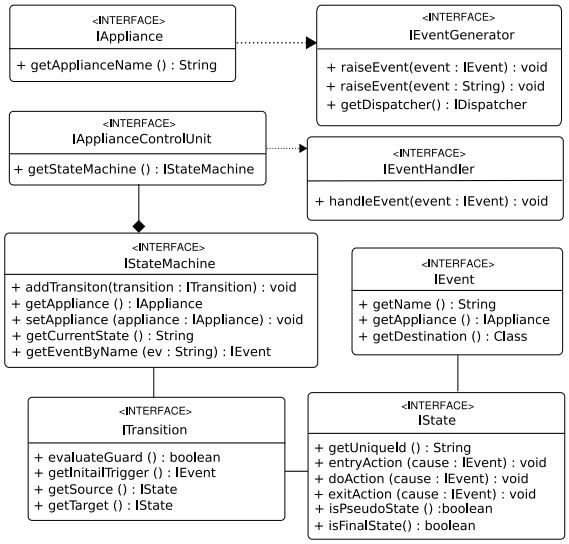

Fig. 1. Reactive Appliance Toolkit

Figure 1 presents an API's class diagram. The basic abstract concept is *appliance*, and this designates any device. It could influence its own environments or other *appliances* by sending *events* or *signals*. An *appliance* could be *static* or *dynamic*. Static appliances do not change their own state. An *appliance control unit* controls the behavior of *dynamic appliances*. The control unit has a state machine; and is assigned to each instance of the *appliance*. The state machine consists of a set of states and transitions. The appliance's state is identifiable by its name. Each state could have a defined dedicated action on entry, exit and process. A pair of source and target states define the transition. An *appliance control unit* can receive events or signal from the environment and thus trigger a state transition. The initial trigger could fire a transition if the evaluation of the transition's guard is successful. A state machine assigned to an appliance runs in its own real-time thread. It corresponds to the situation in the real world, where every *dynamic appliance* has its own processing unit (in the same way as in [2]). Thus, in the context of *RAT,* we do not discuss scheduling issues [8] but we rather focus on interactions between different processes.

Code generator: The code generator plays the role of the model compiler. It takes the *UML* model as input and generates Java classes as a result. In order to generate classes from models, *UML* artefacts have to be "serializing" textually into an *XMI* format.
XMI (XML Metadata Interchange) [13] is an open industry standard that applies *XML* as the stream-based interchange format for models and meta-models. Nowadays, most available *UML* editors support the *XMI* format. The code generator engine uses freely available libraries of *Argo UML* to parse the input file (*XMI* and *zargo* format) and template technique to generate the result code. UML elements such as class, attribute, operation, parameter, state machine are handled by code generation engine.

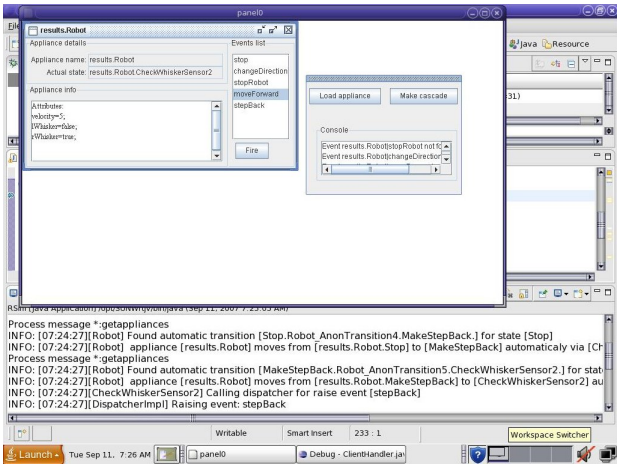

Fig. 2. Management console

B. Management console

The last element of our toolkit is the management console. This is a tool which enables monitoring, verification and direct interaction with simulated appliances. An example screenshot is present in figure 2. Management console uses 'on the fly' introspection; thus, it could receive all the information about each appliance, like its name, actual state, supported receiving or sending events and fired transitions. The basic functionality of the console allows a user to:

- fire a supported event or signal,

[2]www.sun.com
[3]www.aicas.com
[4]http://java.sun.com/javase/technologies/realtime/index.jsp

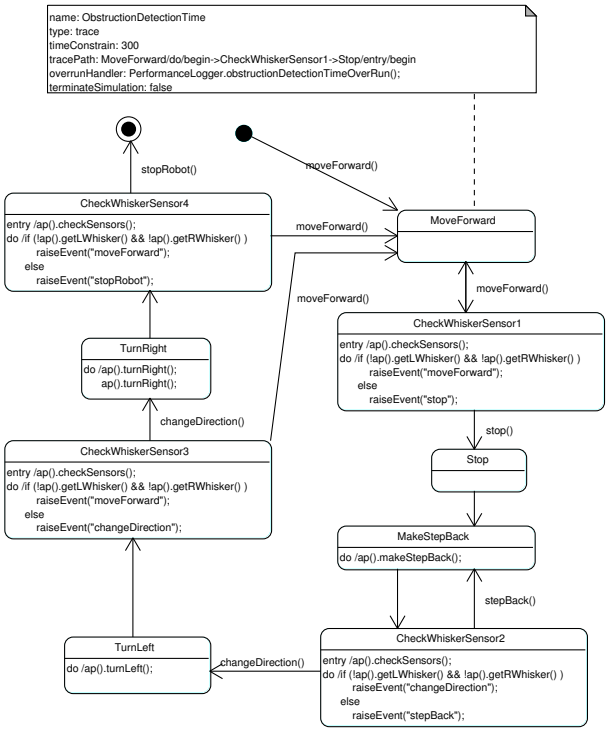

Fig. 3. Robot's state machine

- set a value for a specified parameter/attribute,
- watch the current value of an appliance's attributes.

C. Executable model

This section contains a description of usage of the RAT toolkit. For test purpose, let us see a simple model of a robot modelled in *UML*. The *robot* is able to move forward and backward. It also can turn some specific degrees to the right and left. It has two whiskers – sensors that inform the robot whether it is close to an obstruction. In our simplified model, the whiskers are passive appliances - i.e. they have no state machines assigned to them. The *robot* is an *active appliance*, thus having its own state machine determining its behavior (figure 3). In short, it might be summarized: try to move forward, and if this is impossible, take a step back and change direction. As passive appliances, the sensors (whiskers) do not have their own state machines. Thus, they are accessed by the *Robot* via calling internal methods *getLWhisker()* and *getRWhisker()*.

Verification of timing properties is done via simulative checking of timing constraints. A timing constraint is written as a *UML* comment assigned to a specified state. It has the form of a performance annotation which contains information like: name, value of time constraint, sequence of states that have to be passed by within a specified time, and an overrun handler. In the presented example, there is one timing constraint assigned to the state *MoveForward* (fig. 3). To every state action (entry, do and exit), some delay might be assigned. Such delay should correspond to the actual time needed by subsystems used by actions to get the job done. If, during simulation, a passing by sequence of states takes too much time, the appropriate overrun handler is called. State actions might be specified directly as expressions of some subset of Java language; or in the form of activity diagrams.

III. Summary

The solution presented above supports the rapid development of real-time-reactive systems by enabling the programmer to design and run applications without writing typical, non-functional code. It uses *MDA* with an executable modelling approach. This technique promotes early stage feedback from customers, thus improving quality - especially user perspective. Since the created system is based on *RT Java*, all the benefits of this platform are also available (e.g. it is possible to write a systems that accesses hardware drivers directly). In future, authors plan to extend the solution with formal model verification methods. Writing a plug-in for a commercial tool supporting *RT UML* is also being considered.

Acknowledgments

The paper is partially supported by the HeKatE Project funded from 2007–2009 resources for science as a research project.

References

[1] aicas GmbH. *JamaicaVM – User Documentation*. Aicas GmbH, Haid-und-Neu-Strasse 18, 76131 Karlsruhe, Germany, 2006.
[2] A. Coen-Porisini, C. Ghezzi, and R. A. Kemmerer. Specification of realtime systems using ASTRAL. *IEEE Transactions on Software Engineering*, TRCS96-30, 1997.
[3] S. Flint and C. Boughton. Executable/translatable UML and systems engineering. In *Practical Approaches for Complex Systems (SETE 2003)*, 2003.
[4] A. S. Gokhale, D. C. Schmidt, T. Lu, B. Natarajan, and N. Wang. CoSMIC: An MDA generative tool for distributed real-time and embedded applications. In *Middleware Workshops*, pages 300–306. PUC-Rio, 2003.
[5] D. Harel and A. Pnueli. On the development of reactive systems. In *Logic and Models of Concurrent Systems*, pages 477 – 498. Springer Verlag, 1985.
[6] M. Kostrzewa and K. Kułakowski. A practical approach to the modelling, visualising and executing of reactive systems. In *MIXed DESign of integrated circuits and systems*, pages 705–710, 2006.
[7] K. Kułakowski. Real Time Java, Software Platform for Real Time Systems. In Piotr Gaja, editor, *Systemy Czasu Rzeczywistego, Projektowanie i Aplikacje*, volume 2 of *SCR'05*, pages 143–152. WKŁ, 2005.
[8] J.W. Liu. *Real-time systems*. Prentice Hall, 2000.
[9] S. Lu, W. A. Halang, and L. Zhang. A component-based UML profile to model embedded real-time systems designed by the MDA approach. In *RTCSA*, pages 563–566. IEEE Computer Society, 2005.
[10] Luqi, V. Berzins, and R. T. Yeh. A prototyping language for real-time software. *IEEE Transactions on Software Engineering*, 14(10):1409–1423, October 1988.
[11] J. N. Meunier, F. Lippert, R. Jadhav, and N. Harding. MDA and Real-Time Java: The HIDOORS project. *Technical Report — University of Kent at Canterbury Computing Laboratory*, 17:89–95, 2004.
[12] J. Miller and J. Mukerji. MDA guide version 1.0.1. Technical report, Object Management Group (OMG), 2003.
[13] OMG. Xml metadata interchange version 1.3. Technical report, OMG, 2003.
[14] OMG. UML Profile for Modeling and Analysis of Real-Time and Embedded Systems. Technical report, OMG, august 2007.
[15] OMG. UML Profile for Schedulability, Performance and Time. Technical report, OMG, August 2007.
[16] M. Szpyrka. Analysis of RTCP-nets with reachability graphs. *Fundam. Inform*, 74(2-3):375–390, 2006.