Monitoring your Lego Mindstorms™ with Giotto

Sébastien Saudrais - Olivier Barais – Noël Plouzeau - Jean-Marc Jézéquel

Projet Triskell/IRISA
Campus de Beaulieu.
F - 35 042 Rennes Cedex
prenom.nom@irisa.fr

ABSTRACT. *In the domain of soft real-time application design, the gap between component-specification models and their implementations often implies that the implementations cannot fully take advantage of the specification models. The component's time behaviour is not always used during the implementation process by the targeted programming languages because they cannot deal easily with time. In this presentation we propose a complete model transformation chain to generate monitors from time specification. We demonstrate the applicability of this approach for monitoring a Lego Mindstorms™ application with Giotto platform.*

KEYWORDS : *Timed specification , Model Driven Development, Giotto, Monitoring embedded system, Lego Mindstorms™, .*

1. Motivations

The MDE (Model-Driven Engineering) initiatives, most prominently advanced by IBM with EMF, the OMG (Object Management Group) with the MDA or by Microsoft with Software Factories promote models to be the primary artifacts of software development, making executable code a pure derivative. According to this development paradigm, software is generated with the aid of suitable transformations from a compact description (the model) that is more easily read and maintained by humans than any other form of software specification in use today. In the domain of system engineering, we argue that the techniques promoted by those initiatives can be used to better profit from specifications even if those specifications are partially ignored by the developers during the development process. Specifications can be used to generate monitors that check the real-behaviour of the systems against its specification. We present, first, the development process. We present next how we have applied this process to monitor an embedded systems developed with Lego Mindstorms™. Finally, we discuss the benefits of this process.

2. Development process

The development process aims at facilitating the use of formal methods in a traditional development process. In this way, the designer can choose his implementation techniques (programming language, programming paradigm …) without restrictions. Our approach augments the traditional development process with time specifications. Those time specifications serve to generate monitors. The generated monitors also decorate the system implementation to keep a check on it. This new process can be done in parallel with the traditional development. The augmentation is performed in two steps as presented in Figure 1. The first one is controlled by the designer and the second one is fully automatic:

- During the first step, the architect is assisted to introduce time into component's behaviour. To achieve this, we defined a set of time properties expressed with time patterns: response time, execution time, delay, period… The application architect selects the suitable patterns that must be introduced into his components. Then, the patterns are woven into the original behaviour. The resulting behaviour is a timed automaton for each component. A global timed automaton for the whole application can be computed from the components' automata. The full process is explained in [1]

- During the second step, a monitor is generated to check, at runtime, that the behaviour is correct against the specified behaviour. This monitor is generated from the timed behaviour of the application (that represents the expected behaviour of the components). A model transformation, defined in Kermeta [3], converts the timed automaton of the application into an implementation based on the Giotto framework [4]. The monitor observes whether the execution of the components is correct by keeping a check on the interactions between components. The monitor can be generated in Java or C code. The transformation is explained in [2].

3. Architecture

To evaluate the applicability of this approach, we present how to design and implement an embedded system using this development process. The target platform is a Lego Mindstorms™ NXT Robot. The operating system has been modified for the experiment. The Robot is now based on *leJOS*[1] (pronounced like the Spanish word "lejos" for "far"). It is a tiny Java-based operating system. *leJOS* was originally forked out of the *TinyVM* project. It contains a VM for Java bytecodes and additional software to load and run Java programs. *Lejos* supports features such as: Object oriented language (Java), Preemptive threads (tasks) and Synchronization.

[1] http://lejos.sf.net

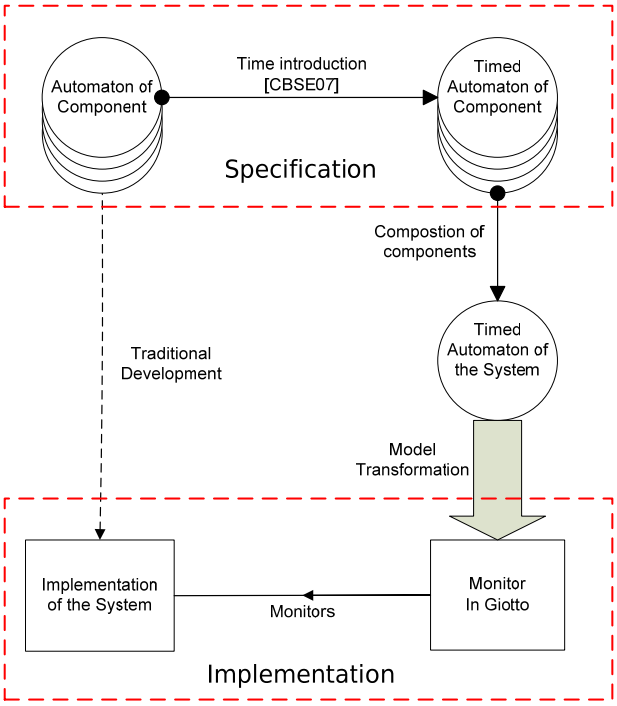

Figure 1: Generation process

The generation of the monitors is based on the Giotto framework [4]. Giotto is a real-time framework for embedded control systems running on possibly distributed platforms. A Giotto program explicitly specifies the exact real-time interaction of software components with the physical world. A Giotto program is generated from the time specification. This program is in charge checking on the components' interaction. The Giotto compiler automatically generates timing code (*ecode*) that ensures the specified behaviour on a given platform. By default, this *ecode* is generated for a complete Java Virtual Machine. In the case of the Robot Mindstorms™, some parts of the Java Standard Library are removed: no File support, no ClassLoader support, no *Hashtable* support, no *switch* statement support. To embed Giotto into *leJOS*, those shortcomings have been overcome. As *leJOS* supports preemptive threads, the main program and the monitors are executed in two different threads. The main program is instrumented to notify the monitors for each of message sent and received.

As an example, we consider a system of a self driving car in which the safe distance between two cars should not be violated more than a specific time denoted *x*. The system behaviour is defined with a timed automaton that contains a clock *x*. The generated monitor warns the driver about violations of this constraint via the Robot Mindstorms™ Liquid Crystal Display.

4. Conclusion and Discussion

This approach aims to monitor Quality of Service properties. The generated monitor does not control the implementation but only checks if the designed behaviour is correct during execution. If not, the user/system is notified who may choose to modify the QoS level. This approach can be justified for a set of systems in which the time is only one of the concerns of the system. In those cases, developers do not necessarily want to use a time control framework like Giotto to implement their system.
We are currently working on a connection between the generated monitor and BIP [5]. This connection allows supervision of a BIP execution with a monitor generated for C.

[1] S. Saudrais, N. Plouzeau and O. Barais -- Integration of Time Issues into Component-Based Applications -- In *Component-Based Software Engineering (CBSE 07)*, Juillet 2007.
[2] S. Saudrais, O. Barais, and L. Duchien -- Using Model-Driven Engineering to Generate QoS Monitors from a Formal Specification -- In *Proceedings of the Aquserm 2006*, Hong Kong, Chine, Octobre 2006.
[3] P-A. Muller, F. Fleurey, and J-M Jézéquel. -- Weaving executability into object-oriented meta-languages. -- In S. Kent L. Briand, editor, *Proceedings of MODELS/UML'2005*, volume 3713 of *LNCS*, pages 264--278, Montego Bay, Jamaica, October 2005. Springer.
[4] T.A. Henzinger, C.M. Kirsch, and B. Horowitz. Giotto: A time-triggered language for embedded programming. In *Proceedings of the IEEE*, 91(1):84–99, January 2003.
[5] A. Basu, M. Bozga, J. Sifakis, "Modeling Heterogeneous Real-time Components in BIP, In *Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM'06)*, 2006, pp. 3-12