MOBY/PLC – A Design Tool for Hierarchical Real-Time Automata*

Josef Tapken**

University of Oldenburg, Germany

Abstract. MOBY/PLC is a graphical design tool for PLC-Automata, a special class of hierarchical real-time automata suitable for the description of distributed real-time systems. Besides the modelling language in use and some features of MOBY/PLC, like several validation methods and code generation, the implementational basis which is built up by the C++ class library MCL is sketched. MCL serves for a rapid development of hierarchical editors for different graphical formalisms by providing a modular hierarchical graph editor.

1 Introduction

MOBY/PLC is a graphical design tool for distributed real-time systems which is based upon a formal description technique called PLC-Automata [3]. It is part of the MOBY-workbench which provides several methods to model and analyse distributed systems, e.g. Petri-net based validation methods like simulation and model-checking for SDL-specifications [4].

The kernel of the workbench called MCL (**MOBY Class Library**) is an application independent C++ class library which comprises, among others, classes for building Motif based hierarchical interactive graph editors [6].

PLC-Automata, a language developed in the UniForM-project [5] and applied to a real-world case study of the industrial partner, are a class of hierarchical real-time automata suitable (but not restricted) to the description of the behaviour of *Programmable Logic Controllers* (PLC) that are often used to solve real-time controlling problems. The automata are tailored to a structural compilation into runnable PLC-code and they are provided with a formal semantics in *Duration Calculus* (DC) [1] to allow formal reasoning about their properties.

This paper gives a survey of MOBY/PLC by introducing its modelling language (PLC-Automata, Sec. 2), by describing its features (Sec. 3) and by making some remarks about its implementational basis (MCL, Sec. 4).

2 PLC-Automata

Programmable Logic Controllers (PLC) are cyclic real-time controllers that frequently poll input values from sensors or other PLCs and compute output values

* This research was partially supported by the Leibniz Programme of the Deutsche Forschungsgemeinschaft (DFG) under grant No. Ol 98/1-1.

** e-mail: tapken@informatik.uni-oldenburg.de

for actuators (or other PLCs). To deal with real-time problems, PLCs are enriched by a convenient timer concept.

A PLC-Automaton describes the behaviour of a PLC by an extended finite state machine with three categories of variables, namely input, local, and output variables. A transition is labelled by a condition and by a list of assignments to local and output variables. In every cycle a PLC-Automaton updates its input variables from the environment and performs (exactly) one transition according to the actual state and values of variables. The execution of a transition may be prohibited by a state label which consists of a time value d ($\in \mathbb{R}_{\geq 0}$) and a Boolean expression over the input variables. A state can only be left if it is held for longer than for d time units or the state expression evaluates to false.

In order to increase expressiveness as well as for the purpose of structuring PLC-Automata are enhanced by a hierarchy concept which is based on state refinement, i.e. a state can represent a set of substates and its label can also restrict the outgoing transitions of the substates.

The whole system specification consists of a network of PLC-Automata which communicate through channels with each other. Therefore, a channel links an output variable of one automaton to an input variable of another, i.e. communication is performed implicitly by updating every cycle the input variables of a PLC-Automaton with the current values of the corresponding output variables. The system network may also be structured hierarchically.

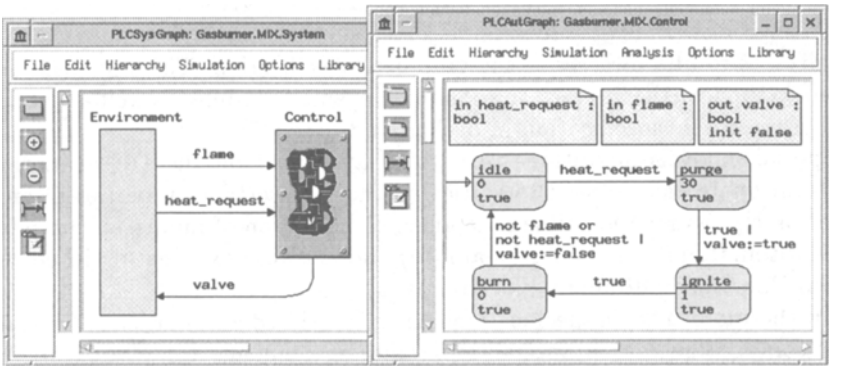

Fig. 1. PLC-automata of the gasburner specification

Fig. 1 shows on the left a system diagram describing a network of two automata (Environment and Control). On the right the Control-automaton is shown in detail. The example is taken from a specification of a control unit for a gasburner, which gets two Boolean inputs from the environment, namely heat_request representing the state of a thermostat changed by the user and flame indicating the status of a sensor monitoring the flame. The control unit delivers one Boolean output valve controlling the gas valve.

3 The MOBY/PLC-Tool

This section gives an overview of the main components which are currently implemented in MOBY/PLC (see Fig. 2).

The central part of the tool is an interactive graphical editor for specifying a real-time system (i). Since the architectural part as well as the behavioural part of a specification may be structured hierarchically the editor comprises several different subeditors, e.g. system editors to describe the network of PLC-Automata or editors to specify automata and subautomata (see Fig. 1).

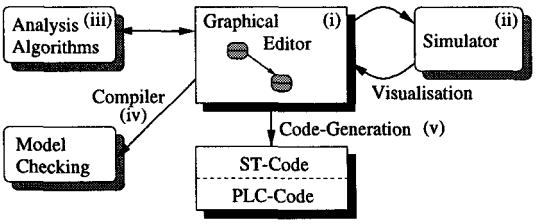

Fig. 2. Components of MOBY/PLC

In MOBY/PLC there are three ways to validate a given specification (ii, iii, iv). A simulator (ii) is able to execute a single or a set of PLC-Automata and to visualize its results directly in the graphical specification. The simulator is designed to support the interactive simulation of small modules as well as extensive tests of the whole specification in background mode [7].

Special analysis algorithms (iii) which are based on the DC-semantics of PLC-Automata can be used to statically calculate certain properties of an automaton, e.g. its reaction time on a given combination of inputs. A compiler of PLC-Automata into timed automata (iv) allows to use existing model checking systems for timed automata, like Kronos [2].

Furthermore, a given specification can be translated automatically by a structural compilation into a special programming language for PLCs called ST (**S**tructured **T**ext)(v). By the use of commercial compilers the ST-code can be transformed into runnable source code for PLCs.

4 The MOBY Class Library

The MOBY Class Library (MCL) is a powerful (Motif based) C++ class library for building graphical applications under the X Window System. MCL adopts several classes and concepts of the SMALLTALK-80 programming environment, e.g. many collection classes and the model view controller concept. Furthermore, it provides runtime class information and dynamic object creation for a given class object or class name.

MCL comprises an editor for hierarchical graphs which is tailored to build a common basis for several implementations of graphical formal description techniques (gFDT), like SDL, statecharts or PLC-Automata. This editor provides a strong modularity by a loose coupling between different hierarchy levels and it serves for a rapid development of editors for gFDTs by a generic description of hierarchy. This description, which is e.g. used for dynamic menu creation, prevents the programmer to reimplement the hierarchy control structure for each gFDT. The programmer only has to define classes of graphs and vertices and to describe what kind of vertices could be inserted into a certain graph and what kind of graphs could serve for a refinement of a certain vertex.

5 Conclusion

In this paper the modelling language, the features and the implementational basis of the design tool MOBY/PLC have been sketched.

Although MOBY/PLC is already usable there are several extensions we are planning to implement. E.g. we want to evaluate and visualize the results of background runs of the simulator. We have to enhance the prototypical implemention of the compiler of PLC-Automata into timed automata. Furthermore, it seems to be promising to expand the static analysis by further algorithms which calculate interesting properties based on the structure of a PLC-automaton.

Currently a graphical editor for Object-Z specifications is developed based on the hierarchical graphs of MCL. This editor should be integrated into MOBY/PLC in order to use Object-Z for the description of data aspects in PLC-Automata.

Acknowledgements. The author thanks H. Fleischhack, H. Dierks, E.-R. Olderog, and the other members of the "semantics group" in Oldenburg for fruitful discussions on the subject of this paper.

References

1. Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A Calculus of Durations. *Inform. Proc. Letters*, 40/5:269–276, 1991.
2. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In *Hybrid Systems III*, volume 1066 of *LNCS*, pages 208–219. Springer Verlag, 1996.
3. Henning Dierks. PLC-Automata: A New Class of Implementable Real-Time Automata. In *ARTS'97*, LNCS. Springer Verlag, May 1997.
4. Hans Fleischhack and Josef Tapken. An M-Net Semantics for a Real-Time Extension of μSDL. In *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods*, volume 1313 of *LNCS*, pages 162–181. Springer Verlag, 1997.
5. B. Krieg-Brückner, J. Peleska, E.-R. Olderog, et al. UniForM — Universal Formal Methods Workbench. In *Statusseminar des BMBF Softwaretechnologie*, pages 357–378. BMBF, Berlin, 1996.
6. Josef Tapken. Implementing Hierarchical Graph-Structures. Technical report, University of Oldenburg, 1997.
7. Josef Tapken. Interactive and Compilative Simulation of PLC-Automata. In W. Hahn and A. Lehmann, editors, *Simulation in Industry, ESS'97*, pages 552 – 556. SCS, 1997.