Anna DEREZIŃSKA, Marian SZCZYKULSKI

Warsaw University of Technology, Institute of Computer Science

doi:10.15199/48.2015.11.43

Application of time concepts from the MARTE profile in a Model-Driven Development case study

***Abstract**. Behavior of a complex system can be designed using state machines of the system classes. Using a Model-Driven Development approach models are transformed into an executable code. Structural and behavioral models can be extended with time concepts from the Modeling and Analysis of Real-Time and Embedded Systems (MARTE) profile. The refined models are used in transformation. We presented a case study of a home alarm system that illustrates an application development methodology. It was used in verification of the approach implemented in Framework for eXecutable UML (FXU). This MDD tool, used for development of an C# application from UML classes and state machines, was extended with the support of MARTE time concepts.*

Streszczenie.** Złożone systemy są modelowane z użyciem maszyn stanowych. Transformacje modeli służą do budowy wykonywalnych aplikacji. Modele mogą być uszczegóławiane z wykorzystaniem pojęć czasowych zdefiniowanych w profilu MARTE. W pracy przedstawiono projekt systemu alarmowego ilustrującego modelowanie pojęć czasowych. Transformacja i realizacja aplikacji systemu była wykonana przy pomocy FXU - narzędzia do automatycznej generacji kodu z klas i maszyn stanowych, wspierającego transformacje modeli ze specyfikacją czasu z profilu MARTE.* ***Wykorzystanie pojęć czasu z profilu MARTE w wytwarzaniu oprogramowania opartym na modelach

Keywords: code generation, UML, state machines, MARTE profile, C#, time modeling. Model-Driven Engineering
Słowa kluczowe: generacja kodu, UML, maszyny stanowe, profil MARTE, C#, modelowanie czasu, inżynieria oparta na modelach.

Introduction

Model-Driven Development (MDD) assists in reliable transformation of modelled system concepts into an executable code [1]. Behavior of complex systems can be designed with state machines of system classes.

Non-functional requirements, especially time-related are important notions in a system modeling. Early versions of UML (1.x) were already supported by the standard OMG profile for Schedulability, Performance and Time (SPTP) [2]. A UML profile consists of a set of concepts specified with stereotypes that refine different UML meta-elements. The standard UML is extended, but its meta-model remains unchanged. Stereotypes are accompanied with tagged values, i.e. a kind of attributes that hold different data.

Starting from the 2.0 version of UML, simple concepts referring to time, its duration and observation were specified in the *SimpleTime* package [3]. This time model is very simple and was intended to be enhanced in specialized profiles. However, the previous SPTP profile was not compatible with the new package.

An extended OMG profile, compatible with UML 2.x and SysML, is MARTE: Modeling and Analysis of Real-Time Embedded Systems [4]. The profile consists of three main packages devoted to foundations, model design, and real time & embedded analysis. Time domain was specified in the *Time* package, which belongs to profile foundations. Referring to MARTE in this paper, we deal only with concepts of the MARTE::Time package.

In order to benefit from an MDD approach that uses refined models with time specification, an adequate tool support is needed. Although, much work has been done on modelling with MARTE [5-9], there are still gaps in an automatic code development. Therefore, we extended the Framework of eXecutable UML (FXU) [10] to cover elements of MARTE. FXU is a MDD tool for C# code generation and application development from UML classes and state machines.

In this paper, we present a case study to illustrate the idea of a model development with MARTE time concepts and its application in the MDD approach. The system was used in verification of the model to code transformation of MARTE elements realized in FXU.

The structure of the rest of the paper is as follows. In the next Section we recall a related work. Section 3 describes briefly FXU. Sections 4, 5, and 6 present a case study requirements, a system model and experimental verification of the approach. Finally, Section 7 concludes the paper.

Related work

The time elements of MARTE find a broad application in system modeling and analysis, such as design of control systems, HW/SW co-design of embedded systems, test generation, and other systems, not only real-time ones.

Currently, some CASE tools have been equipped with facilities to create models with UML profiles, including MARTE, e.g. IBM Rational Software Architect (since v. 7.0), Papyrus UML, MagicDraw, etc. However, an open issue is further transformation of a MARTE model and support for its concepts in a model or a programming code execution.

Many approaches are devoted to development of embedded systems with use of specialized environments, such as SystemC, OpenCL, or Pharaon [5,6]. They are suitable for HW/SW, but are limited to domain languages.

Another approaches are based on simulation, where models are directly simulated or transformed into a general simulation environments. In [7] authors used a Simulink platform to run models that are transformed from SySML models with MARTE specyfication.

One of solutions is transformation of a MARTE model into an existing formal description. Afterwards, using analysis tools we can verify dependability requirements defined in the model. For example, transformation rules to Object-Z were discussed in [8], while ATL-based transformation to FIACRE model was presented in [9]. MARTE models were also transformed to stochastic Petri nets, Timed Petri nets, VHDL models, Promela, etc.

In the contrary, the FXU approach deals with a general purpose language (C#) and its commonly used environment (MS Visual Studio) as a target of the model transformation.

Transformation of UML/MARTE models using FXU

Framework of eXecutable UML (FXU) focused on transformation not only structural models, but also complete models of state machines [10,11]. It was the first tool that supported model-driven development of state machine models towards the C# programming language. At the moment, some tools support model transformation to C# code, but limited to class models (e.g. IBM Rational

Software Architect), or to very simplified state machines (e.g. Sparx Enterprise Architect). Except FXU, no other tools dealing with full state machines in C# were mentioned in a systematic review [12].

Moreover, FXU takes into account one of the most comprehensive sets of state machine notions. The similar level of state machine complexity can be transformed by the IBM Rational Rhapsody tool, but not to the C# language.

FXU consists of two components: FXU Generator and FXU Runtime Library. The generator transforms UML class and state machine models into the corresponding programming code and creates a software project. The project can be supplemented with an additional code, e.g. detailed implementations of operations. The final application is built from this code linked with the FXU Runtime Library that includes implementation of all state machine notions.

One of FXU versions was extended with the capability for supporting a subset of MARTE profile. The following stereotypes of MARTE::Time, used in a class model and a state machine, are transformed into their corresponding code: *TimedDomain*, *ClockType*, *Clock*, *TimedProcessing*, *TimedEvent*, *TimedValueSpecification*. These stereotypes can be specified with sets of different tagged values that also were used in modeling and further taken into account in model transformation. The Runtime Library of FXU was extended with processing of MARTE events, as well as support for a logical clock and a chronometric clock.

Requirements of a modelled system

A model of a home alarm was designed in order to verify the approach. It was inspired by two models, an intrusion alarm [13] and a fire alarm [14]. The general idea of the system is a home facility that combines alarms preventing the home from a burglar intrusion and from a fire risk.

The main actor of the model is a home administrator that can switch on or switch off an intrusion alarm or a fire alarm. An administrator operates the system with an access code. At first, “ON” button is pressed, next all digits have to be provided. A correct input of a code is shown by a green color diode. After an intrusion alarm is switched on, a red diode is blinking. It indicates the time to leave the home.

The system is equipped with a smoke sensor. If a fire system is launched, the smoke sensor monitors the smoke level. If a high level of smoke is detected, a warning signal is triggered. A signal of the fire alarm is activated if smoke reaches its critical level.

Another sensor is devoted to detection of an unintended movement in the home. When an intrusion alarm is switched on, a movement detection launches an appropriate signal. Before the signal is activated a red diode is blinking. During this time the alarm can be deactivated by an access code.

Access code can be changed if all alarms are switched off. A successful change of the code is denoted by blinking of a green diode for a certain time.

The system behavior is controlled by a set of time requirements that should be met by the system. In brackets we provided the exemplary values used in experiments.

A) A time limit for introducing the whole access code (10 sec.) After elapsing of this time, the inputted digits are invalided and the code has to be re-written.

B) Activation time of the home alarm after an intrusion detection (20 sec.). If the correct code was not written after this time, an alarm signal is launched.

C) Leaving time of the home after activation of the intrusion alarm (30 sec.)

D) Starting time of access code input after pressing “ON” button (3 sec.) After this time the button has to be pressed once again.

E) Time interval, in which an access code can be changed (30 sec.) If this time is exceeded, the code change process has to be repeated.

F) Maximal interval between pressing of buttons on a keyboard during an input of an access code (3 sec.).

G) Interval between consecutive flashes of red and green signal diodes.

H) Time for stopping a signal of a critical smoke level (30 sec.). If after this time an acceptable smoke level was detected, the signal is deactivated.

I) Time interval between displaying of consecutive messages about alarm statuses (2.5 sec)

J) Maximal time of reloading a message on a display (0.5 sec.).

K) Maximal time of displaying all messages (7 sec.)

System model with MARTE profile

The home alarm system was designed with UML class models and behavioural state machines. Time requirements described in the previous section were specified using stereotypies and tagged values from MARTE::Time.

System elements that manage elapsing of time are provided in the *Clock* package (Fig. 1). It is labelled with the MARTE *TimedDomain* stereotype, as elements of the time domain specification are referenced by stereotypes of the profile used in other system models.

In Figure 1, stereotyped classes with their MARTE tagged values are shown. Two clock types are specified in the system. *CodeEnteringClock* is a logical clock type, which manages time by events. Whereas *AlarmClock* specifies a chronometric clock type, associated with the physical time. Defined clocks use these clock types. During the model to code transformation, this time domain is searched for all instances of clocks used in other diagrams.

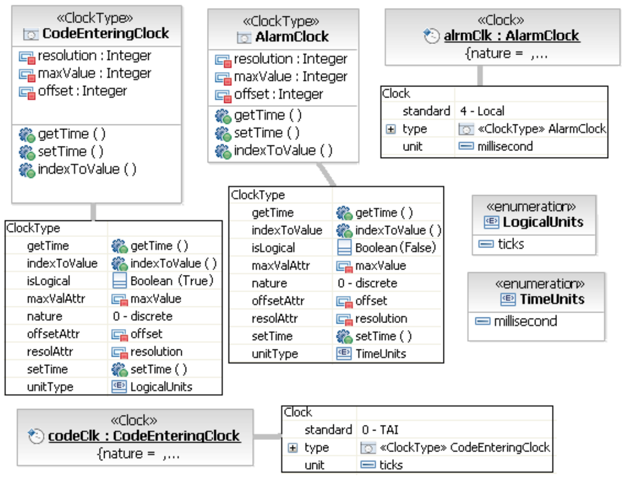

Fig.1. Clock types and clocks of the home alarm system

The main classes that realize the system logic are placed in the *Controller* package (Fig. 2). The system process and communication between components is controlled by the *AlarmController* class. Processing and verification of an access code is managed by the *CodeController* class. Another classes correspond to different system components, like sensors, keypad, LED indicators, and a display.

All classes from the *Controller* package were specified with their state machines labelled with MARTE stereotypes. As an example, a state machine of *SmokeSensor* is shown (Fig. 3). If a higher smoke level or a critical level encountered an object of *SmokeSensor* delivers information about the level to the alarm controller.

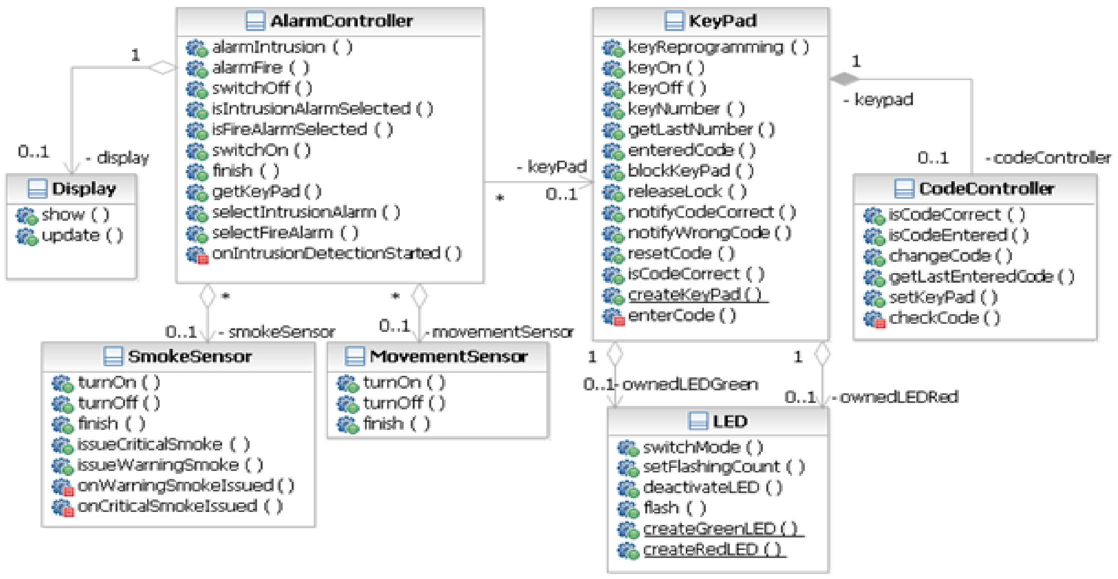

Fig.2. Main structural model of the system - classes from the *Controller* package

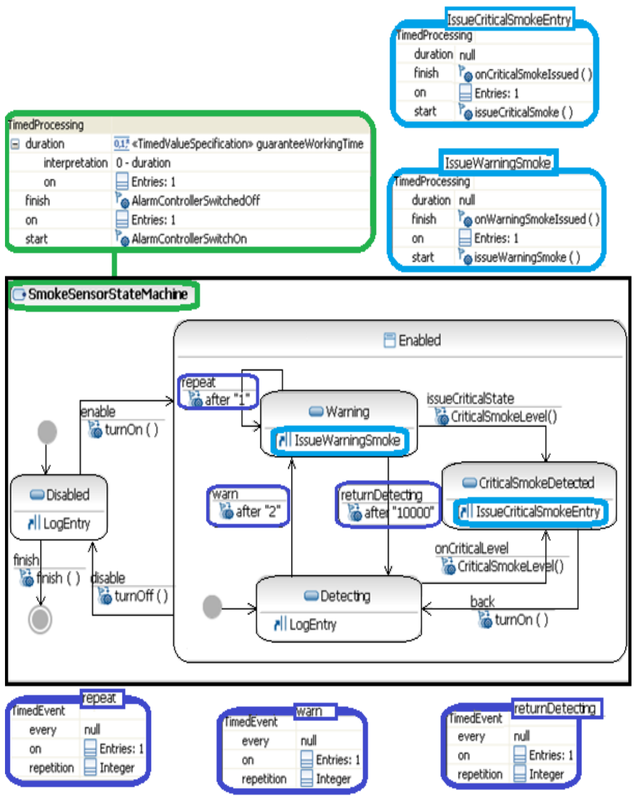

Fig.3. UML/MARTE state machine of the *SmokeSensor* class

The sensor operates in two main modes: disabled and enabled, in accordance to the status of the alarm system. In the enabled state, the sensor can be in one of substates: detecting - while smoke is monitored, warning - when a higher smoke level was detected, and a critical level of smoke. Apart from the states, their internal operations and transitions with triggers, different MARTE elements are used in the behavioral specification of the smoke sensor.

The whole state machine diagram is annotated with the *TimedProcessing* stereotype, shown with its tagged values in the left top corner of Figure 3. It was used for the specification of an event that activates the whole state machine (*AlarmControllerSwitchOn* in the *start* tag), and an event generated after the state machine end (*AlarmControllerSwitchOff* in the *finish* tag*)*. The time limit of the entire state machine activity is specified with the *duration* tag. In case of exceeding this time, a special MARTE time exception is raised.

The state machine of the smoke sensor uses a logical time. Events given in the complex state *Enabled*, such as after "1", after "2", after "10000", are defined using a logical clock and specified with the *TimedEvent* stereotypes (shown in the bottom part of Figure 3).

Activities performed on entry to states *Warning* and *CriticalSmokeDetected* are annotated with the *TimedProcessing* stereotypes. Events that are generated at the beginning and just after the finish of these activities are defined in appropriate tags of the stereotypes (tags given in the right top corner in Figure 3).

In an analogous way, state machines of the remaining classes were created and specified with the MARTE stereotypes and tagged values.

Code generation and application verification

Structural and behavioral models of the system refined with the MARTE specification were prepared using IBM Rational Software Architect. Further, the models were transformed into the corresponding source code by the FXU Generator. The final application was built in the Visual Studio using the generated C# project and the FXU Runtime Library. Additional details of the operations, which were not specified in the model, were added to the source code of methods. The final application was run and tested in experiments, in which the system behavior was compared with the model specification.

Test scenarios of system behavior were designed in correspondence to the state machine diagrams. The scenarios took into account a typical and critical usage of the system. Then, they were verified against application behavior observed by a user with tool support. An exemplary test scenario is shown in Table 1. Expected results are specified in terms of the modeled behavior.

Four mechanisms were used in verification of the application behavior: application logs, application traces, measured time relations, and timing logs.

Table 1. Test Scenario

No	Step	Expected results
1	Start of the home alarm	The alarm controller starts working in the detection state. Several objects launch their activities: keypad in its idle state and display in a starting place. Sensors of movement and of smoke are disabled.
2	Selection of the intrusion alarm only	Display class comes to the *IntrusionAlarmSelected* state.
3	Pressing of the "ON" button	The alarm controller proceeds to a detection substate in which an intrusion alarm is armed. Keypad moves on to its position accepting arming or disarming of an alarm. Controller of an access code begins to work.
4	Input of a correct access code (4-digit)	The code controller moves to the *EnteringCode* state, and reads the consecutive digits. After accepting of a number of correct digits, it realises an activity in the *Correct* state, waits for the prescribed time delay, and finishes its behaviour via the final steps. Keypad returns to its idle state. Alarm controller moves on to *IntrusionAlarmActivationDelay* and after the appropriate time delay activates an intrusion alarm. Movement sensor transits to the *MovementnotDetected* state.
5	Detection of a signal that indicates movement	The object of a movement sensor is enabled and reaches its movement detection state. The alarm controller goes to its state of movement detection.
6	Access code missing after a 40 sec. time delay	The alarm controller goes over to its *AlarmState* that is a substate of the intrusion detection functionality.

Application logs consist of information generated directly by the application and outputted in a display or into a file. Different application phases were observed: with intrusion and smoke alarms being switched on or off, different stages of alarm activities, processing of an access code, reports of sensors after a movement detection etc. We could confirm the behaviors specified in various test scenarios.

Traces of the application were also analyzed using FXU Tracer [15]. Traces are collected during an application run, and can be further studied together with the UML models. We can re-run an application trace in a step mode, considering all states of the original state machines and verifying their transitions.

Time relations in the application were examined with assistance of the stopper mechanism of the .NET environment (*System.Diagnostics.Stopwatch* class). Current times were measured by stoppers, added to the application logs, and displayed during the application run. For example, selected measured time relations referred to: time intervals between consecutive data showed in the system display, times of access code inputs, times of launching of the alarm signal after an intrusion detection. It was assumed that the time relations can deviate of maximum 1% of the values specified in the requirements. All measured times fulfilled this condition.

Finally, a detailed verification of time relations was realized using logs generated by the *log4net* library. In a recommended place, a log file was created. The file included all information about traversal of a state machine associated with its timestamps. Therefore, a comprehensive time analysis is possible. We defined various time constraints to be verified based on these logs. The analyzed logs confirmed the system behavior specified by UML models with MARTE::Time.

Conclusions

We presented the application of MARTE time concepts in the modelling and development of an application. The modelling of time relations and time requirements was beneficial due to the tool support that allow us to automatically convert a time specification into its corresponding code in a final application. This MDD approach gives the advantage to concentrate on the time relations in the early phases of a system development. For more complex scenarios, the approach of direct model to code transformation can be combined with formal verifications [8,9]. However, specifying all necessary MARTE stereotypes and their tagged values could be wearisome. Another limitation may be incorporation of low level specification of time support architecture into higher level models.

REFERENCES

[1] Liddle S.W., Model-Driven Software Development, Embley, D.W., Thalheim, B. (eds.) *Handbook of Conceptual Modeling*, Springer (2011), 17-54

[2] Object Management Group, UML Profile for Schedulability, Performance, and Time Specification, version 1.1. (2005), http://www.omg.org/spec/SPTP/

[3] Object Management Group, OMG Unified Modeling Language, www.omg.org/spec/UML/

[4] Object Management Group, UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. version 1.1. (2011), http://www.omg.org/spec/MARTE/

[5] Rodrigues A.W.O.; Guyomarch F.; Dekeyser J.-L., An MDE Approach for Automatic Code Generation from UML/MARTE to OpenCL, *Computing in Science & Engineering*, 15 (2013), No 1, 46-55

[6] Posadas H.; Peñil, P.; Nicolaas, A.; Villar E., System synthesis from UML/MARTE models: The PHARAON approach, Electronic System Level Synthesis Conference, (2013), 1-8

[7] Morelli M., Di Natale M., An MDE approach for the design of platform-aware controls in performance-sensitive applications, *Emerging Technology and Factory Automation*, IEEE (2014), 1-8

[8] Haiyang X., Zhuang Y., A formal transformation approach of MARTE model, *2nd Inter. Conf. on Information and Control Engineering*, IEEE, (2015) 550-554

[9] Zhang T. Jouault F,et al. MDE-Based Model Transformation: From MARTE Model to FIACRE Model, *Journal of Software*, 20 (2009), No.2, 214-233

[10] Pilitowski R., Derezinska A., Code generation and execution framework for UML 2.0 classes and state machines, T. Sobh (eds.*) Innovations and Advanced Techniques in Computer and Information Science and Engineering,* Springer (2007), 421-427

[11] FXU Framework for eXecutable UML , http://galera.ii.pw.edu.pl/~adr/FXU/

[12] Dominguez E., Perez B., Rubio A.L., Zapata M.A, A systematic Review of Code Generation Proposals from State Machine Specifications, *Information & Software Technology,* 54 (2012), No. 10, 1045 – 1066

[13] Richardson M.W., Designing a Home Alarm using the UML and implementing it using C++ and VxWorks, I-Logicx UK Ltd., http://www.uml.org.cn/oobject/vxworks.pdf

[14] Graf S., Ober I., Ober I., A real-time profile for UML, *International Journal on Software Tools for Technology Transfer*, 8 (2006), No. 2, 113-127

[15] Derezińska A., Szczykulski M., Tracing of state machine execution in model-driven development framework, *Proc. of the 2nd International Conference on Information Technologies*, ICIT'2010, IEEE Soc. (2010), 109-112

***Authors**: dr inż. Anna Derezińska, Warsaw University of Technology, Institute of Computer Science, ul. Nowowiejska 15/19, 00-665 Warszawa, E-mail: A.Derezinska@ii.pw.edu.pl; mgr inż. Marian Szczykulski, E-mail: marian.s27@gmail.com*

The correspondence address is:
e-mail: *A.Derezinska@ii.pw.edu.pl*