Implementation of Timed Automata in a Real-time Operating System

Pavel Kučera, Ondřej Hynčica, and Petr Honzík

***Abstract*— The goal of this paper is to present a tool for automatic implementation of Timed Automata model in a real-time operating system. The purpose of this tool is to make design, implementation and verification of real-time control systems easier because human resources will be concentrated more on the area of specification and verification than implementation issues. In the paper are described developed modules enabling automatic implementation of Timed Automata models from UPPAAL into the real-time operating system extension RTX. The software tool is being developed as an open-source project with support from the National Grant Agency at the Department of Control and Instrumentation, Brno University of Technology.**

***Index Terms*— Timed Automata, Temporal Logic, UPPAAL, RTX.**

I. Introduction

Control systems play an important role in everyday life. People and the systems themselves are surrounded by hundreds of different devices consisting of one or more control system: cellular phones, PDAs, televisions, cars, buildings etc. They usually help us, but in the case of incorrect operation they can be a threat to us or other systems [1], [2], [3]; therefore it is necessary to ensure their reliable and safe behaviour. To produce such reliable and safe systems via a standard (and proved) approach is and arduous task requiring a lot of time, skill, and human effort; however, the human work is mainly concentrated in the area of implementation (writing routines and programs, designing HW, debugging, etc.) and not in the area of specification and verification. Formal methods help to concentrate human resources on the specification and verification and it significantly improves final reliability and safety of the control systems.

Timed Automata are often used for description, modelling and verification of different systems in almost all branches of engineering. Today, they are an interdisciplinary tool facilitating work on product and system design from HW to SW parts of applications. Many articles and papers are published every year; they show Timed Automata as a useful tool in many areas of human activity such as aviation [5], economy [6], industry [7], communication [4], control systems [8], etc.

Complexity of today's control systems forces system developers to often utilize operating systems in the control engineering applications. It is because the complexity and computational power demands of the operating system becomes less important due to higher performance of the processors while their advantages are indisputable. However, developing control application with or without operating system support is a totally different task. Techniques, methods and strategies that are used in implementations without operating systems cannot be used if implementing the same control task in the target platform with the operating system support. This is because control system application must be separated into less or more independent tasks (processes, threads) and with such independent tasks difficulties arise around exclusive access to shared resources; memory locations and hardware elements. Moreover, a general-purpose operating system cannot be used because control task is a strictly time-constraint process and it's deadlines must be complied with at all time. This is why a real-time operating system must be used; however difficulties arise related to the time behaviour of the tasks.

II. Formal Approach

The formal approaches used for designing the real-time control systems such as: UML, ROPES, etc., concentrate predominantly on theoretical solutions of the appropriate interactions of the blocks to provide real-time behaviour in terms of time-constraints and synchronism, however, practical experiences show that a fair amount of failures in real-time control systems are not only caused by flaws in the design, but also by underestimation of the efficiency of the particular point-to-point connection. Having designed proper Timed Automata, synchronization points of processes, and employing further measures to diminish the compromising of timeliness is crucial if the control system is to be transferred into practice.

Utilization of Timed Automata and real-time operating system can solve many of the aforementioned problems. Timed Automata can be used not only for formal description of the control task but also for verification of it's correctness and time behaviour. Real-time operating systems can be used for proper implementation of this formal model in the target platform. Transformation from the Timed Automata specification into the real-time operating system application can be done automatically without human intervention; it

Manuscript received July 16, 2010.

This work has been supported in part by the Grant Agency of the Czech Republic GA1890030 (Implementation of Timed Automata into real-time operating systems), Ministry of Education, Youth and Sports of the Czech Republic Research Intent MSM0021630529 (Intelligent systems in automation), and Project 1M0567 (Centre for applied cybernetics).

Authors are with the Department of Control and Instrumentation, Brno University of Technology, Kolejní 4, 612 00 Brno, Czech Republic. (phone: +420-541141113; e-mail: {kucera, hyncica, honzikp}@feec.vutbr.cz).

This work was supported in part by the U.S. Department of Commerce under Grant BS123456

significantly improves reliability and safety parameters of the final application.

This transformation can be done by a development strategy that has been previously presented by the authors [9]. The simplified model of this strategy can be described by a state diagram shown in Fig. 1.

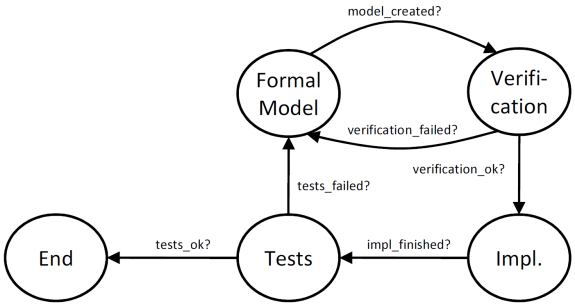

Fig. 1: State diagram of the development strategy

The state diagram of the proposed development strategy consists of 4 important steps. In the step: *Formal Model*, a Timed Automata model of the control task is created. As soon as the model is created it's verification phase can be carried out in the step: *Verification*. When the verification step is finished, the development procedure can either move back into the *Formal Model* step, because the verification failed, or into the step: *Implementation*, where automatic implementation of the model is built.

The formal model can be incorrect if deadlock or livelock is observed during the verification step. In this case, we must return to the *Formal Model* step and another appropriate Timed Automata model of the system must be created. The *Implementation* step is followed by the *Tests* step. In this step all necessary tests of the final application are made. If any of the performed tests fail, the development strategy must return back into the *Formal Model* and changes in the Timed Automata model must be carried out. If the tests of the system were successfully then the development procedure ends.

III. Timed Automata Model

For the purposes of the formal description of the task, timed automaton and temporal logic have been chosen as an appropriate description method. The main advantage of this formal method is that there exists an integrated tool environment for modelling, simulation, and verification of real-time systems called UPPAAL. UPPAAL was developed jointly by BRICS at Aalborg University and the Department of Computer Systems at Uppsala University [10]. It is the appropriate tool for a system that can be modelled as a collection of non-deterministic processes with a finite control structure and real-valued clocks, communicating through channels or shared variables. Typical application areas include real-time controllers, communication protocols and embedded systems control. UPPAAL consists of three main parts: a description language, a simulator, and a model-checker [18].

1. The Description Language

The description language is a non-deterministic guarded command language with simple data types; unbounded integers, arrays, etc. It serves as a modelling or design language to describe system behaviour as networks of automata extended with clock and data variables [18].

1. The Simulator

The simulator is a validation tool that enables examination of possible dynamic executions of a system during early design or modelling stages and thus, provides an inexpensive means of fault detection prior to verification by the model-checker, which covers the exhaustive dynamic behaviour of the system [18].

1. The description language

3. The model-checker is used to check invariant and bounded liveness properties by exploring the symbolic state-space of the system, i.e., reachability analysis in terms of symbolic states represented by constraints [18].

The theory of timed automaton is well described in [11]. A number of verification tools have been developed for timed systems in the past year. UPPAAL is one of them as described in [12], [13], or [10]. The other tools are well described in [14].

The goal of UPPAAL has always been to serve as a platform for the tool to provide a flexible architecture that allows experimentation. It should allow orthogonal features to be integrated in an orthogonal manner to evaluate various techniques within a single framework and investigate how they influence each other [15].

The formal model is represented by Timed Automata diagram(s) created in UPPAAL. The model is stored in a XML file. This formal model can be automatically converted into the objects that are easy to implement into real-time operating systems and these objects result in an executable code.

IV. Real-Time Operating System

There are many real-time operating systems available. Some of them are free, some are available under GPL or BSD license and some of them are proprietary. Decision process of choosing the right one is a painful work and many aspects must be considered [16]. However practical experiences show that the selection of the suitable real-time operating system is often the question of experiences of the developers.

Authors are very well experienced with the real-time extension RTX. It is not a stand-alone real-time operating system because it requires the presence of the Microsoft operating system on the target platform. But it provides almost all features and properties like standard real-time operating system for x86 platforms [17].

RTX is a real-time extension based on Win32 Application Interface (API). It provides precise control of IRQs, I/O, and memory. It operates in Ring 0 with the highest performance and sustained interrupt rates of 30 KHz with an average IST latency of less than one microsecond. This is why RTX becomes more and more popular in control systems, especially in the area of diagnostic systems and real-time control systems. Another advantage of this real-time extension is a close cooperation with the Windows operating system. It makes possible to create outstanding graphics user interface (GUI) without any additional costs or effort because this GUI is created under standard Windows API and with the

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

real-time extension core is connected by a shared memory mechanism. Moreover, RTX is available for free in the evaluation edition.

Basic concept of RTX is based on so-called processes slots. Slot is an execution unit (task) that can be started or stopped independently to the other RTX processes. Each task consists of at least one thread. Threads in the same task run in the same environment (memory, CPU core) however threads from different tasks are planned by the same scheduler. Priority system consists of 128 independent priority levels and two or more threads can share the same priority level.

V. Implemetation Tool

Implementation of the Timed Automata model in RTX can be carried out automatically in the step: *Implementation* in Fig. 1. The relationship between UPPAAL, implementation tool, RTX, x86 target platform and control technology is shown in Fig. 2.

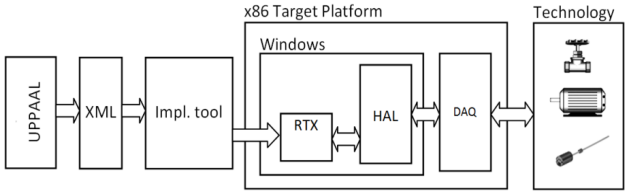

Fig. 2: Relationship between the designing tool, target platform and technology

The Model of the control system is created and verified in the UPPAAL tool. If the verification is passed then the implementation tool reads the Timed Automata model saved in XML file and creates a Visual C++ project for the RTX application. The RTX application creates one task consisting of one or more threads. Each thread in the RTX process is equivalent to UPPAAL process assignment. The RTX process has a direct connection to the Hardware Abstraction Layer (HAL) of the Target Platform and it can directly control hardware in the Technology. Mapping of the model variables into the hardware must be completed by a human in Visual C++ project. However internal behaviour of the RTX process and its threads are generated automatically by Timed Automata model.

Many problems must be solved before the Timed Automata model can be successfully transferred into the RTX structures. The most important one is a time concept. UPPAAL time is an integer value passing from $-\infty$ to $+\infty$. RTX time is based on a HAL timer period that is a minimum time quantum that can be used for timers and sleep functions. This HAL timer period can be one of the fixed value; 100, 200, 500 and 1000 μs and this value is the same for all processes and its threads. The corresponding value must be chosen before the implementation of the Timed Automata in RTX as the implementation tool should be aware of it.

Another problem can be with the Urgent Channels. Urgent channels are similar to synchronisation channels, except that it is not possible to delay in the source state if it is possible to trigger a synchronisation over an urgent channel. It cannot be successfully realized in symmetric architecture with one available CPU (with just one core) in the target platform. Therefore, urgent synchronization is carried out by a supervising thread using the maximum priority. This thread is attached to the control process and controlled via the HAL timer period.

VI. Example

Let us consider a simple system that represents the JK flip-flop device:

$$Q_{n+1} = J\overline{Q_n} + \bar{K}Q_n \quad (1)$$

Where J and K are inputs of the device and Q_n and Q_{n+1} are its outputs before and after the synchronization clock. A simplified model of this device in UPPAAL is shown in Figure 3.

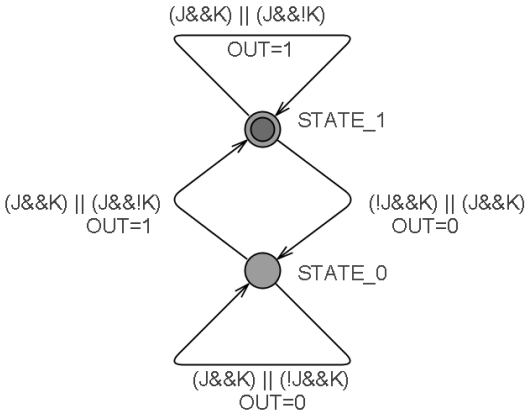

Fig. 3: Simplified model of the JK flip-flop

The system is a Mealy state machine with two states and four transitions. Anytime the system enters the state: STATE_1, its output is set to 1 and if the system enters the state: STATE_0, the output is set to 0. Transition between the states corresponds with the equation 1. For simplicity the time and invariant conditions are not considered here.

The implementation of this system in RTX is carried out by the process JK_TEST consisting of 3 threads. The structure of the process is shown in Fig.fi 4.

Thread *sync* is responsible for the proper synchronisation of signals J and K. Function mutex ensures an exclusive access to the signals J and K because the threads *inout* and *state_machine* also share them. Function event then represents calling of the corresponding event handling of the synchronization channel; standard, urgent or broadcast channel. This thread has the highest priority in the process. This thread is completely generated by the implementation tool.

Thread *inout* is responsible for communication with the HAL. Implementation is strictly dependent on the target platform hardware and it cannot be generated automatically from the Timed Automata model. The implementation tool creates only a skeleton and the developer must implement the correct read and write function. However the developer must not solve problems with synchronisation or unique access to the shared variables; it is already carried out by the skeleton and the other thread(s) in the process. Priority of this thread must be lower than the priority of the sync thread.

The thread *state_machine* is a core of the task. It generates the logical behaviour of the system. It consists of one or more case-switch statements where transitions among states are performed. All the conditions that fire the transitions for each

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

state are encapsulated in a Condition to Leave function. For instance, function CL_STATE_1() has an implementation:

```
CL_STATE_1() {
   if ((J&&K) || (J&&!K)) {
      Q = 1;
      state = STATE_1;
      return;
   }
   if ((!J&&K) || (J&&K)) {
      Q = 0;
      state = STATE_0;
      return;
   }
}
```

This thread is also completely generated by the implementation tool and its priority can be the same as *inout* thread but must be lower than *sync* thread.

Process: *JK_TEST*

```
thread: sync

do
  {
    mutex(J);
    event(J);
    mutex(K);
    event(K);
  }
while (!terminated)
```

```
thread: inout

do {
    mutex(J);
    read(J);
    mutex(K);
    read(K);
    mutex(Q);
    write(Q);
} while (!terminated)
```

```
thread: state_machine

do {
    case (state) {
       switch STATE_1: CL_STATE_1();break;
       switch STATE_0: CL_STATE_0();break;
    }

} while (!terminated)
```

Fig. 4: Implementation of the JK flip-flop in RTX

VII. CONCLUSION

A suggested tool for automatic transferring of the Timed Automata model into the real-time operating system brings several advantages into the area of system control design.

Routine implementation work of the software engineer will be replaced by automatic implementation; it significantly increases safety and reliability of the control systems, while its verification demand will be lower.

Verification of the system will be based on its formal specification; it brings the possibility to verify the project during a specification process and final verification can be automated or semi-automated: without human intervention.

Total design time will be reduced, whilst safety and reliability parameters will be preserved or improved with lower costs.

However, several issues remain unresolved regarding the tool. As was aforementioned, the challenging problem with urgent channels was solved by a special thread *sync* that controls the synchronisation between channels. The minimal delay between synchronization events rely on the HAL timer period. This period can be in the range from 100 to 1000 μs; and it can be too long for time-critical application. Thus another approach must be found.

Another issue deals with Committed Locations. The Committed Location is a location (state), where no delay pass is allowed. If a broadcast synchronisation is sent to different locations (state) then it is difficult to ensure constant delay pass among the states.

Finally, the invariant conditions can be used in the future version of this tool for time-checking behaviour of the model, during real-time execution of the control system.

ACKNOWLEDGEMENT

This work has been supported in part by the Grant Agency of the Czech Republic GA1890030 (Implementation of Timed Automata into real-time operating systems), Ministry of Education, Youth and Sports of the Czech Republic Research Intent MSM0021630529 (Intelligent systems in automation), and Project 1M0567 (Centre for applied cybernetics).

REFERENCES

[1] NTSB, “Annual Report to Congress“, National Transportation Safety Board, NTSB, 1 Jul 2010. Web. 1Jul2010, <http://www.ntsb.gov/Publictn/2010/SPC1001.pdf>.
[2] Leveson, N. “Computers and Risk”. Chapter 2 of “Safeware: System Safety and Computers”, Addison-Wesley, 1995.
[3] Neumann, P., G., „Computer Related Risks“, Reading, Mass.: Addison-Wesley, 1995.
[4] Petalidis, N, “Verification of a fieldbus scheduling protocol using timed automata”, Computing and Informatics, Volume 28, Issue 5, 2009, Pages 655-672.
[5] Glässer U., Rastkar S., Vajihollahi M., “Modeling and validation of aviation security“, Studies in Computational Intelligence, Volume 135, 2008, Pages 337-355.
[6] Kristoffersen, K. J., Pedersen, C., Andersen, H. R., “Runtime verification of timed LTL using disjunctive normalized equation systems“, Electronic Notes in Theoretical Computer Science, Volume 89, Issue 2, October 2003, Pages 215-230.
[7] Zhou, M., He, F., Gu, M., Song, X., “ Translation-based model checking for PLC programs“, Proceedings - International Computer Software and Applications Conference Volume 1, 2009, Pages 553-562.
[8] Wardana, A.N.I., Folmer, J., Vogel-Heuser, B., “Automatic program verification of continuous function chart based on model checking“, IECON Proceedings (Industrial Electronics Conference), 2009, Pages 2422-2427.
[9] Kucera, P., Honzik, P., “Automation of Real- time Embedded System Design“, The13th World Multi-Conference on Systemics, Cybernetics and Informatics. WMSCI. Orlando: WMSCI, 2009, Pages 23-26.
[10] Larsen, K.G., Pettersson, P., Yi, W., ”UPPAAL in a Nutshell”. Interantional Journal on Software Tools for Technology Transfer, Vol. 1, Number 1-2, 1998, Pages 134-152.
[11] Alur, R., Dill, D.L., ”A theory of timed automata”, Theoretical Computer Science, Vol. 126, 1994, Pages 183-235.
[12] Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W. “Uppaal - a Tool Suite for Automatic Verification of Real Time Systems”, Proceedings of Workshop on Verification and Control of Hybrid Systems III, number 1066 in Lecture Notes in Computer Science, Springer Verlag, October 1995.
[13] Hune, T., Larsen, K.G., Pettersson, P., ”Guided Synthesis of Control Programs Using UPPAAL”, Proceedings of the IEEE ICDCS International Workshop on Distributed Systems Verification and Validation, April 2000, Pages 15-22.

[14] Yovine, S., “A verification tool for real time systems” International Journal on Software Tools for Technology, Vol 1, October 1997, Pages 123-133.
[15] David, A., Behrmann, G., Larsen, K.G., Yi, W., “A tool Architecture for the next generation of UPPAAL”, Technical report, Uppsala University, Sweden, February 2003.
[16] Barr, M., “How to Choose a Real-Time Operating System“, Netrino The Embedded Systems Experts, 1Jul 2010. Web. 1Jul2010, < http://www.netrino.com/Embedded-Systems/How-To/RTOS-Selection/>.
[17] IntervalZero, “ Hard Real-Time with IntervalZero RTX on the Windows Platform“, IntervalZero, 10Jun2010. Web.10Jun2010, < http://www.intervalzero.com/pdfs/RTXWhitePaper-6-09.pdf>.
[18] UPPAAL, “About”, Department of Information Technology at Uppsala University, 1 Jul 2010. Web. 1Jul2010, <http://www.it.uu.se/research/group/darts/uppaal/about.shtml>.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)