A Tool for Compositional Analysis of Timed Systems by Abstraction

Pavel Krčál, Leonid Mokrushin and Wang Yi

Dept. of Information Technology, Uppsala University, Sweden.
Email: {pavelk,leom,yi}@it.uu.se.

Abstract We present a tool for compositional timing and performance analysis of real-time systems modeled using timed automata and the real-time calculus [5]. It is based on an (over-) approximation technique in which a timed automaton is abstracted as a transducer of abstract streams described by arrival curves from network calculus [2]. As the main feature, the tool can be used to check the schedulability of a system and to estimate the best and worst case response times of its computation tasks. The tool is available for evaluation at www.timestool.com/cats.

1 Introduction

Real-time systems are often constructed based on a set of real-time tasks. These tasks may be scheduled and executed according to given release patterns. There have been a number of methods and tools developed for timing analysis to estimate the worst case (and also the best case) response times of computation tasks for such systems, e.g., Rate-Monotonic Analysis [4] for periodic tasks, Real-Time Calculus (RTC) [5] for tasks described using arrival curves [2], and Times [3] using Timed Automata (TA) [1]. Some of these techniques, e.g., implemented in Times can deal with systems with complex release patterns, but do not scale well with system size and complexity; the others are scalable but can not handle systems with complex structures. Our goal is to take the advantages of these existing techniques and develop a tool, that is scalable and also capable of handling complex systems.

We model the architecture of a system using data-flow networks in the style of the RTC, where the nodes stand for the building blocks or components of the system and edges for the communication links between the nodes. In the RTC, nodes represent either tasks or functions on arrival and service curves. To enhance the expressiveness of the task model, we also allow TA nodes as release patterns. The essential idea of our analysis technique is to abstract the TA nodes using arrival curves, which can be done modularly for each node, and to compose the analysis results in order to perform the system level analysis.

2 The Model

The basic concepts of our model are tasks, task arrival patterns and computational resources. Tasks are abstractions of programs that execute on a processing

unit and thus consume computational resources. The parameters of a task are its best and worst case execution times on a reference processor. A task arrival pattern describes the moments in time at which tasks are released for execution. The execution of a task is scheduled on a processing unit according to a preemptive fixed priority scheduling strategy. The capacity and availability of a processing unit form the model of a computational resource. We use arrival curves [5] and TA [1] to model task arrival patterns, and service curves [5] to model computational resources.

We shall introduce a notion of an abstract stream as a set of non-decreasing diverging sequences of timestamps ranging over positive reals. Each timestamp denotes an occurrence of an event. An abstract stream defined by a pair of upper and lower arrival curves is the greatest abstract stream such that all the sequences of events of this abstract stream comply with the constraints induced by the pair of arrival curves as in [5]. A timestamp with a name assigned to it is called an action. We define a timed trace to be a sequence of actions with non-decreasing diverging timestamps. A set of timed traces forms a timed language.

A timed language where action names are taken from a bounded (by the resource capacity) subset of non-negative rational numbers is called an abstract resource. These numbers represent the amount of computational resources in reference processor units until the next action. An abstract resource defined by a pair of upper and lower service curves is the greatest abstract resource such that all the sequences of actions of this abstract resource comply with the constraints induced by the pair of service curves as in [5].

The model analysed by the tool is a finite network of nodes interconnected with links. Nodes may have ports and links are directed edges connecting ports of the nodes. Each port has three parameters: name, direction (input or output) and type (*event* or *resource*). We distinguish the following types of nodes:

- **Task node**, a node with a task assigned to it; it has two input ports: *release* (an arrival pattern of the task) and *demand* (computational resource available for the task execution) and two output ports: *finish* and *rest* representing the pattern of task finishing times and the remaining computational resource respectively; the ports *release* and *finish* are of the *event* type whereas *demand* and *rest* are ports of the *resource* type,
- **Task arrival pattern node**, a node with either a pair of upper and lower arrival curves or a TA assigned to it; in the first case the node has only one output port and no input ports, and in the second case – the input and output ports corresponds to the input and output letters of the TA; all the ports are of the *event* type,
- **Resource node**, a node with one output port of the *resource* type and a pair of upper and lower service curves assigned to it,
- **Function node**, a node containing a function of the real-time calculus [6]; the input ports correspond to the parameters of the function and there is only one output port; all the ports have the same type – *event* or *resource*.

The links between the node ports must always connect an output port to an input port and loops are not allowed. Moreover, it is only possible to connect

ports of the same type. Intuitively, the links model the control flow and the flow of computational resources. Task priorities are defined by the order of the task nodes in the flow of computational resources.

Semantically, every time an event arrives to the *release* port of a task node, the task associated with the node is released for execution. During its execution a task consumes computational resources entering the *demand* port of a task node. The task completes its execution after being computed for a period of time between the best and the worst case times specified in task parameters and issues an event to the *finish* port. The resources not used by the task are passed through to the *rest* port. Task arrival pattern nodes and resource nodes with assigned pair of curves generate events triggering task releases and computational resources respectively. Task arrival pattern nodes with an associated TA transform input abstract streams into the output abstract streams as follows. First, the input abstract streams are converted into a timed language. Then, the TA interpreted as a transducer computes the output timed language. Finally, this language is approximated by an abstract stream for each output port of the node. Function nodes transform abstract streams or resources from incoming ports according to the real-time calculus functions assigned to them.

3 Tool Architecture and Features

The tool implementation (as shown in Fig. 1) consists of two parts: model construction and model analysis. The first part contains internal system model representation and the editors, which allow to define the topology of a system, the timed automata assigned to TA nodes, the functions assigned to the RTC nodes, and pairs of arrival and service curves.

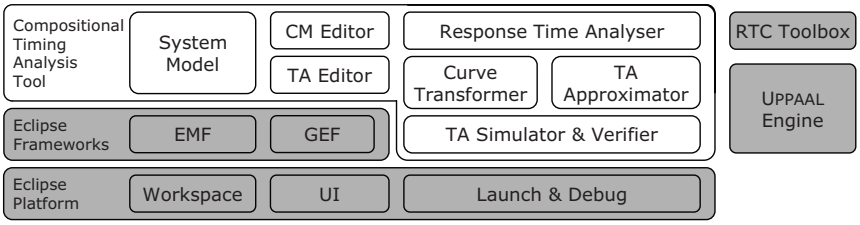

Figure1. The architecture of the tool.

The tool computes the best and the worst case response times of every task for a given task release pattern and a set of computational resources. It does this by analysing abstract streams and resources appearing on the links of the model network. Depending on the type of the node the tool assembles an appropriate operation at the evaluation time. For example, for a TA node an evaluation operation consists of the encoding of the input abstract streams into TA, computing

the output of the TA node using UPPAAL verification engine, and decoding the results back into the form of the abstract streams.

The screenshot of the tool is shown in Fig. 2. The tool is implemented as a set of plugins built on top of the Eclipse Development Platform. The implementations of the internal models are based on the Eclipse Modeling Framework. A script language is used for specification of the model together with a dedicated text editor. The graphical editor assists designers in TA modelling and is built on top of the Eclipse Graphical Editing Framework. The runtime part of the tool extends Eclipse Launch&Debug functionality and provides a set of specialized views for monitoring and interpreting the results. We use Real-Time Calculus Toolbox [6] to evaluate RTC functions.

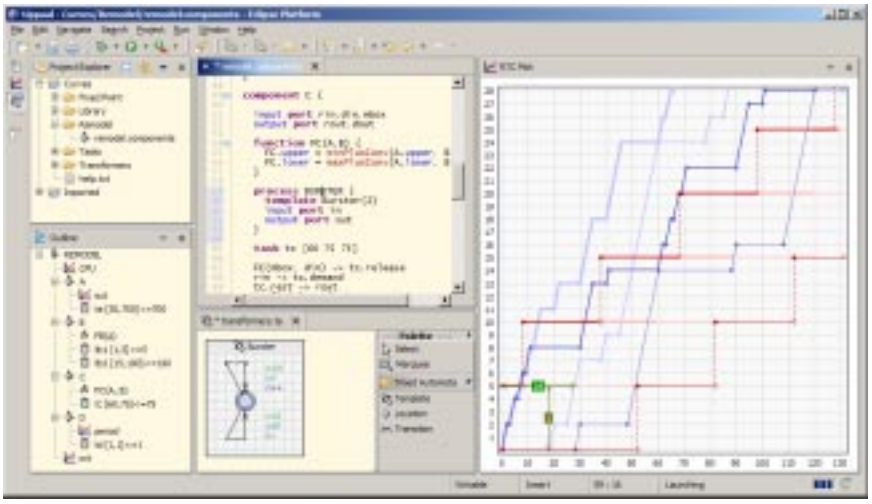

Figure2. The screenshot of the tool.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. R. Cruz. A calculus for network delay. *Proc. of IEEE Trans. Information Theory*, 37(1):114–141, 1991.
3. E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theoretical Computer Science*, 354:301–317, March 2006.
4. M. Joseph and P. Pandya. Finding response times in a real-time system. *BSC Computer Journal*, 29(5):390–395, October 1986.
5. L. Thiele, S. Chakraborty, M. Gries, A. Maxiaguine, and J. Greutert. Embedded software in network processors - models and algorithms. In *Proc. of EMSOFT'01*, pages 416–434. Springer-Verlag, 2001.
6. E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.