A Petri Net Based Language for Realtime Systems

Nikolaus Huber, Pontus Ekberg, and Wang Yi

Uppsala University, Uppsala, Sweden
firstname.lastname@it.uu.se

Abstract

This paper proposes a framework that shall aid in the design, development, analysis, and verification of realtime software. A new programming language is proposed, internally modeled by a special class of Petri nets. This new programming language allows full separation between the timing and functional aspects of the software system.

1 Introduction

We present the concept of a new programming language targeting the development of realtime systems. Realtime systems, in their broadest sense, are software systems in which the correctness of a computational result does not only depend on the correctness of its (numerical) value, but also on the time point at which this value becomes available. When developing and verifying software for such systems, two different methodologies have evolved: Some systems are developed in traditional programming languages such as C or C++ and a model is abstracted from the code, on which the timing aspects are verified. On the other hand, developers can use a modelling framework, such as TIMES [1], TASTE [7], etc. to model the system architecture, perform timing analysis on the model, and use automated code generation for the skeleton of the implementation. In this report we would like to make the case for an additional approach, combining both methodologies.

2 Background

Many realtime systems fall under the category of safety critical systems, which means that an error would either lead to great financial loss, or even the loss of life. Examples of such systems include medical equipment, cars, aeronautics, and so forth. It is therefore of the utmost importance that these systems are constructed in a way that allows verifying their functional

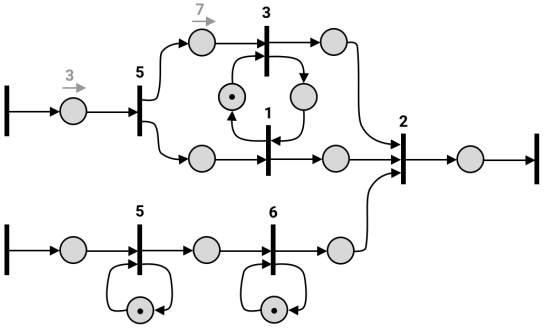

Figure 1: Example of a timed event graph.

as well as their timing behaviour. In many traditional programming frameworks for realtime systems, the functional and timing aspects influence each other.

The idea for this programming language arose during the first author's study of a special class of timed Petri nets. Figure 1 shows an example of a timed event graph (TEG), which is a Petri net in which each place (depicted as a gray circle) has only one upstream and one downstream transition (depicted as black bars). These types of networks are useful in the modelling of a variety of discreet event systems, such as public transport networks. While different forms of Petri nets have been used in the past to model software systems, to the best of the authors' knowledge, this has not been done with TEGs so far. Conceptually, the transitions (i.e. bars) model computations, that consume input and produce output. The inputs and outputs are FIFO buffers, modelled as places (i.e. gray circles). Dedicated data items are shown as tokens (i.e. small black circles in the places). A transition can fire (i.e. a computation can be performed), whenever there is at least one token in each incoming place. In addition to the rule that each place can only have one incoming and one outgoing edge, TEGs allow the explicit modelling of time. Small numbers above the bars illustrate response times of the corresponding computation, meaning the maximum amount of time between a transition becoming active (i.e. it having at least one token in each incoming edge) and that transition producing an output token on each outgoing edge. Times on the places model transition times, i.e. the amount of time a token needs to travel through the buffer. This can be used to explicitly express the time needed to forward a data item through a communication channel, for example when programming distributed systems. Transitions with no incoming edges are inputs to the system, likewise, transitions with no outgoing edges model outputs. In the following sections we discuss our thoughts as to why TEGs can be an interesting basis for a programming language for realtime systems.

3 Relationship to Kahn networks

First we need to declare some further restrictions on what kind of computation a transition can perform. Each computation has a set of inputs and outputs, which is fixed. Furthermore, the computation is stateless in the sense that any form of state (i.e. local memory) must be modelled as a self-edge with a place holding the state variable(s). With these restrictions, TEGs become a special case of Kahn networks [6]. As pointed out by Kahn, the output of these networks is independent of the concrete transition firing order, as long as the there is no stalling (i.e. if a transition is able to fire, it will fire at some point in the future). This means, that by using TEGs as the basis of our programming language, we can separate the functional and timing analysis completely.

4 Algebraic analysis of TEGs

As outlined in [2], the type of TEG illustrated in this report allows modelling the timing of the network within a special kind of (max,+)-algebra. Intuitively, maximisation models synchronisation between different inputs, and addition models delays. As described thoroughly in [2], a TEG such as the one shown in figure 1 can be modelled as a system matrix with entries in a special kind of semi-ring. When trying to compute the end-to-end delays of system functions (i.e. the time it takes from firing one or more input transition(s) of the network to the time the corresponding output transition(s) are fired), one multiplies this system matrix with an input vector modelling the input timing. This enables analysis on the network without

fixing the input, and allows, for example, to calculate bounds on the input timings.

While the basics of this algebraic analysis have been known since the nineties, recent advances in this field make TEGs even more appealing as a basis for our proposed programming language. An algebraic approach to maximizing the timing of individual places while preserving output performance is shown in [4]. In [5], the model is extended to allow expressing ranges of time, e.g. for modelling uncertainties in the response time. An extension to model partial synchronisation, allowing to fix the firing times of transitions to multiples of a given period is illustrated in [3]. We believe, that all these and future advances in the analysis of TEGs will increase the expressiveness of the proposed language.

5 Conclusion and Future Work

In this report we presented our ideas for a new programming language framework which shall aid in the development and verification of sound realtime software. By utilizing a version of timed Petri nets, we can guarantee functional determinism, and separate the timing verification. The language presented here is limited in its expressiveness. Each routine presented inside a node is required to produce an output for every input, and the outputs of two different nodes cannot be merged. The proposed language is to be seen as a baseline, which we hope on further extending in the future. By using response times on the transitions, we were able to separate the scheduling problem from the current framework. However, for any real implementation, the scheduling of TEGs on real (multi-core) hardware has to be looked at.

Finally, this project came into existence as an idea for a platform for experimentation for another project, conducted at the group of the authors. Said project, tentatively called MIMOS [8] (Multiple Inputs Multiple Outputs) is an ongoing effort of simplifying the implementation of realtime software for heterogeneous hardware platforms.

References

[1] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times: a tool for schedulability analysis and code generation of real-time systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 60–72. Springer, 2003.

[2] François Baccelli, Guy Cohen, G.J. Olsder, and Jean-Pierre Quadrat. Synchronization and linearity - an algebra for discrete event systems. *The Journal of the Operational Research Society*, 45, 01 1994.

[3] X. David-Henriet, J. Raisch, L. Hardouin, and B. Cottenceau. Modeling and Control for Max-Plus Systems with Partial Synchronization. *IFAC Proceedings Volumes*, 47(2):105–110, 2014.

[4] Xavier David-Henriet, Laurent Hardouin, Jorg Raisch, and Bertrand Cottenceau. Holding Time Maximization Preserving Output Performance for Timed Event Graphs. *IEEE Transactions on Automatic Control*, 59(7):1968–1973, July 2014.

[5] Laurent Hardouin, Bertrand Cottenceau, Mehdi Lhommeau, and Euriell Le Corronc. Interval systems over idempotent semiring. *Linear Algebra and its Applications*, 431(5-7):855–862, August 2009.

[6] Gilles Kahn. The semantics of a simple language for parallel programming. In *IFIP Congress*, 1974.

[7] Maxime Perrotin, Eric Conquet, Julien Delange, André Schiele, and Thanassis Tsiodras. Taste: A real-time software engineering tool-chain overview, status, and future. pages 26–37, 01 2011.

[8] Wang Yi, Morteza Mohaqeqi, and Susanne Graf. Mimos: A deterministic model for the design and update of real-time systems, 2020.