Analysis of Event-Driven Real-Time Systems with Time Petri Nets

A Translation-Based Approach

Zonghua Gu and Kang G. Shin
RTCL/EECS, University of Michigan, Ann Arbor, MI 48109 {zgu,kgshin}@eecs.umich.edu

Abstract: The growing complexity of modern real-time embedded systems makes it imperative to apply formal analysis techniques at early stages of system development. This paper considers formal modelling of event-driven real-time systems with Time Petri Nets, and subsequent analysis via model-checking by a simple, fully automatable translation into Timed Automata. The proposed approach is applied to a small application scenario taken from Avionics Mission Computing.

Keywords: real-time, embedded, Time Petri Net, Timed Automata, UPPAAL, model checking, CORBA

1. INTRODUCTION

Real-time embedded systems are ubiquitous in modern society, many of which perform safety-critical functions, and therefore, it is imperative to have tools and techniques that can guarantee a high degree of system correctness. In this paper, we consider application of Merlin and Farber's *Time Petri Net* (TPN) [12] to model event-driven real-time systems, and formally define a translation procedure from a TPN model into a semantically equivalent Timed Automata [7] model in order to perform model-checking on the TA model. This translation procedure also gives a formal semantics for TPN in terms of TA, and clarifies a number of semantic ambiguities in the original TPN definition. For example, we clearly define the semantics of multiple-enabledness of a transition as freshly enabling a transition after each firing, which is intuitively the behaviour of a task serving multiple queued execution requests.

As an example of our approach, we model and analyze an application scenario taken from Avionics Mission Computing [8]. Using the model checker UPPAAL, we were able to check the system timing properties such as end-to-end latency. In case a system timing property is violated, UPPAAL gives us an error trace leading to the violation state and allows us to gain more insight into the cause of the violation.

This paper is structured as follows: Section 2 considers TPN modelling of real-time scheduling. Section 3 describes a simple algorithm for mapping TPN into TA. Section 4 considers modelling and analysis of an application scenario taken from Avionics Mission Computing. Section 5 describes related work, and the paper concludes with Section 6.

2. MODELLING OF REAL-TIME SCHEDULING WITH TPN

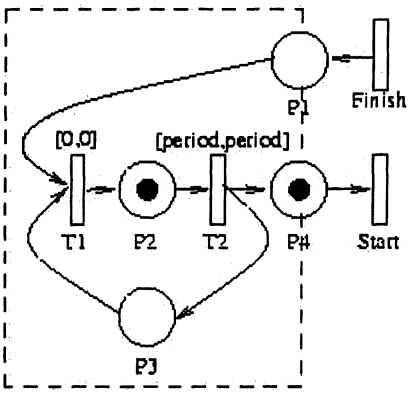

Figure 1. A periodic timer in TPN.

Figure 1 shows the TPN model for a periodic timer, and Figure 2 shows a TPN model for static priority, *non-preemptive* scheduling of two periodic tasks. The inhibitor edge from P11 to T22 models the fact that Task1 has priority over Task2: a non-empty P11 prevents T22 from firing.

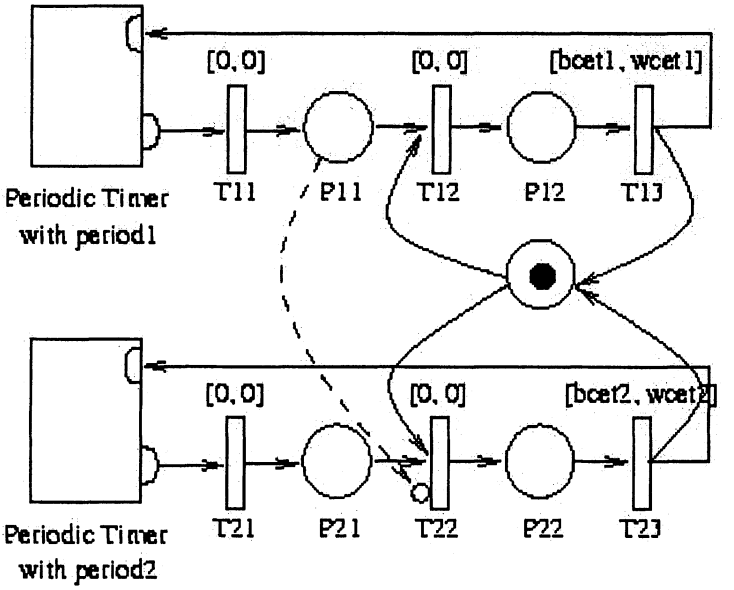

Figure 2. Static priority, *non-premptive* scheduling of two periodic tasks. The blocks marked *Periodic Timer* denote instantiations of the periodic timer model in Figure 1 with periods period1 and period2, respectively. The top part of the figure represents high-priority task Task1, and the bottom part represents low-priority task Task2. BCET stands for *best-case execution time* and WCET stands for *worst-case execution time.*

3. MAPPING TPN INTO TA

We formally define a translation algorithm for mapping a TPN model into a semantically equivalent TA model.

1. Declare a global urgent channel *go*. A transition with an urgent channel as its synchronization label is an urgent transition, and has to be taken whenever it is enabled without delay.
2. Create an automaton with a single location, and a transition with synchronization label *go!* starting and ending at that location, as shown in Figure 4.
3. For each TPN place $p \in P$, declare an integer global variable with the same name in the TA model.
4. Suppose a TPN transition $t \in T$ has an associated delay interval $[lb, ub]$, a pre-set of k input places $p_1^{in}, \ldots, p_k^{in}$, a post-set of m output places $p_1^{out}, \ldots, p_m^{out}$, and a set of n inhibitor arcs from places $p_1^{inh}, \ldots, p_n^{inh}$. Classify all the TPN transitions according to the number

of its input, output and inhibitor places. For example, all transitions with 1 input place, 2 output places, and 1 inhibitor place are put into the same class. For each transition class:

a) Define an automaton template with two locations *disabled* and *enabled*, one local clock c, and $k+m+n$ integer parameters named $p_1^{in},\ldots,p_k^{in},p_1^{out},\ldots,p_m^{out},p_1^{inh},\ldots,p_n^{inh}$.

b) Add an invariant condition $c \le ub$ at the location *enabled.*

c) Add an edge from *disabled* to *enabled* with guard condition $p_1^{in} \ge B(p_1^{in},t),\ldots,p_k^{in} \ge B(p_k^{in},t), p_1^{inh} == 0,\ldots,p_n^{inh} == 0$, synchronization label *go?,* and assignment label $c := 0$.

d) Add $k+n$ edges from *enabled* to *disabled* with guard condition $p_1^{in} < B(p_1^{in},t)$ on $edge_1$, ..., $p_k^{in} < B(p_k^{in},t)$ on $edge_k$, $p_1^{inh} > 0$ on $edge_{k+1},\ldots p_n^{inh} > 0$ on $edge_{k+n}$, and synchronization label *go?* on every edge.

e) Add an edge from *enabled* to *disabled* with guard condition $p_1^{in} \ge B(p_1^{in},t),\ldots,p_k^{in} \ge B(p_k^{in},t), p_1^{inh} == 0,\ldots,p_n^{inh} == 0, c \ge lb$, and assignment label
$p_1^{in} := p_1^{in} - B(p_1^{in},t),\ldots,p_k^{in} := p_k^{in} - B(p_k^{in},t),$
$p_1^{out} := p_1^{out} + F(p_1^{out},t),\ldots,p_m^{out} := p_m^{out} + F(p_m^{out},t)$

5. In the system configuration section, instantiate one automaton template for each TPN transition, with the appropriate global variables as parameters, representing the input, output and inhibitor places of that transition.

Figure 3. A dummy automaton with an urgent transition *go*.

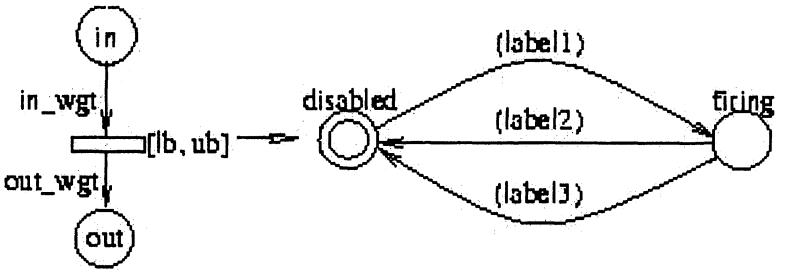

Figure 4. TA model of a TPN transition *t* with 1 input place *in*, 1 output place *out*, and time bounds *[lb,ub]*. The process template has argument list (int *in, out*; const *in_wgt, out_wgt*; const *lb, ub*), and a local clock *c*. The transition labels are: (label1) is *((in >= in_wgt; go? ;c := 0)*; (label2) is *(in < in_wgt; go?)*; (label3) is *(in >= in_wgt, c >= lb; in := in – in_wgt, out := out + out_wgt)*.

Figure 4 shows the mapping for a TPN transition *t* with 1 input place *in* and 1 output place *out*. The urgent channel *go* ensures that the automaton changes its state from *disabled* to *enabled* as soon as $in \geq in\_wgt$, that is, the input place in contains *in_wgt* or more tokens. The TPN transition's delay interval *[lb, ub]* is modelled by the state *enabled* in the TA model, which has an invariant condition $c \leq ub$, and a guard condition $c \geq lb$ on the lower outgoing transition that represents transition firing. The resulting semantics is that the automaton has to take the lower transition from *enabled* to *disabled* if it has been staying in state *enabled* continuously for at least *lb* time units, and at most *ub* time units. The automaton can also be disabled by condition $in < in\_wgt$, meaning that some other conflicting transition has been fired and removed one or more tokens from *t*'s input place *in* so that the number of tokens is now less than *in_wgt*.

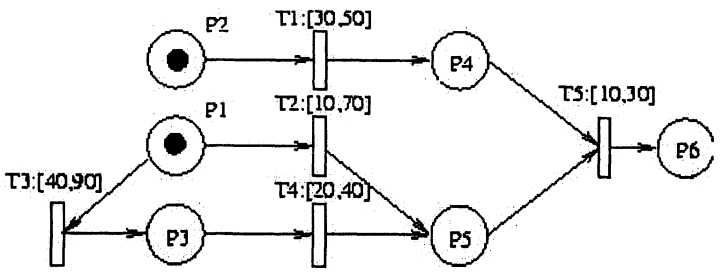

Figure 5. Simple TPN modeling concurency, competition and synchronization.

Figure 5 shows a simple TPN taken from [11]. In order to translate this TPN model into a TA model, it is simply a matter of instantiating the TA templates for TPN transitions with 1 input/1 output, and 2 input/1 output,

which happen to be the only two types of transitions present, as shown below:

```
int p1 := 1,p2 := 1,p3 := 0,p4 :=0,p5 := 0,p6 := 0;
urgent chan go;
T1 := T1in_1out(p2, p4, 1, 1, 30, 50);
T2 := T1in_1out(p1, p5, 1, 1, 10, 70);
T3 := T1in_1out(p1, p3, 1, 1, 40, 90);
T4 := T1in_1out(p3, p5, 1, 1, 20, 40);
T5 := T2in_1out(p4, p5, p6, 1, 1, 1, 10, 30);
System Dummy, T1, T2, T3, T4, T5;
```

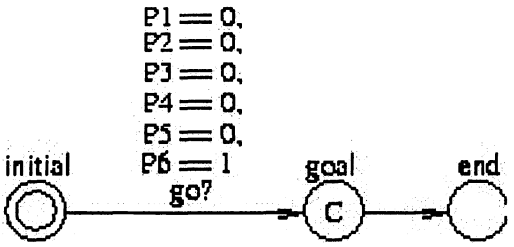

Figure 6. Observer automaton

Figure 6 shows an observer automaton that records the time t it takes to reach the goal state (0, 0, 0, 0, 0, 1) from the initial state (1, 1, 0, 0, 0, 0), where the state vector denotes marking of the TPN (p1, p2, p3, p4, p5, p6). Using the model checker UPPAAL we can prove that t falls in the time interval [40, 140]. In order to verify that this is a tight bound, it is necessary to perform three queries, due to UPPAAL's lack of parametric analysis [6] capability:

1. A[] observer.goal imply observer.c $\geq$ 40 and Observer.c $\leq$140. This is checked to be true.
2. A[] observer.goal imply observer.c $\geq$41. This is checked to be false.
3. A[] observer.goal imply observer.c $\leq$139. This is checked to be false.

4. MODELLING AND ANALYSIS OF AN APPLICATION SCENARIO FROM AVIONICS MISSION COMPUTING

Software for Avionics Mission Computing [8] is the embedded software onboard a military aircraft for controlling mission critical functions, such as,

navigation, target tracking and identification, weapon firing, etc. The software provided to us by Boeing as part of the DARPA MoBIES (Model-based Integration of Embedded Software) project is modelled with *Unified Modelling Language* (UML) [3], and runs on a distributed hardware platform on top of real-time CORBA TAO [4]. Its software architecture is publish/subscribe, using Real-Time CORBA Event Service [5] as its underlying communications substrate. Event publishers push events through the event channel to event consumers, whose execution is triggered by the arrival of events. The system runs at a number of different rates driven by timer event publishers, such as 40Hz, 20Hz, 10Hz, 5Hz, and 1Hz.

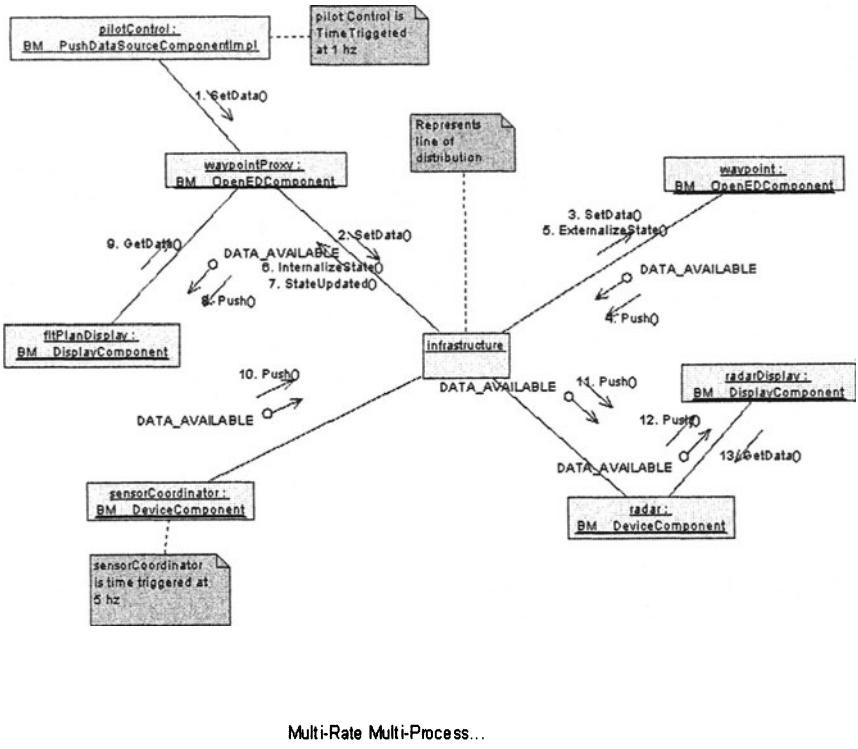

Figure 7. *UML Collaboration Diagram for a multi-rate, multi-processor scenario.*

Figure 7 describes an execution scenario that is multi-rate (1Hz and 5Hz) and multi-processor. The 1Hz thread is initiated by the *pilotControl* component, which calls *SetData* on the *waypointProxy* component. The *waypointProxy* component, in turn, forwards the *SetData* call through the network to the master component *wayPoint* on another processor. Upon its wakeup, the *waypoint* component pushes a DATA_AVAILABLE event

through the network, updating *waypointProxy* with fresh data and notifying *fltPlanDisplay* that fresh data is now available at the *waypointProxy* component. Upon notification, *fltPlanDisplay* calls *GetData* on *waypointProxy* to get the new waypoint data.

The 5Hz thread is initiated by the *sensorCoordinator* component, which pushes a DATA_AVAILABLE event through the network to the *radar* and *radarDisplay* components. Upon its wakeup, *radarDisplay* calls *GetData* on *radar* to get fresh data.

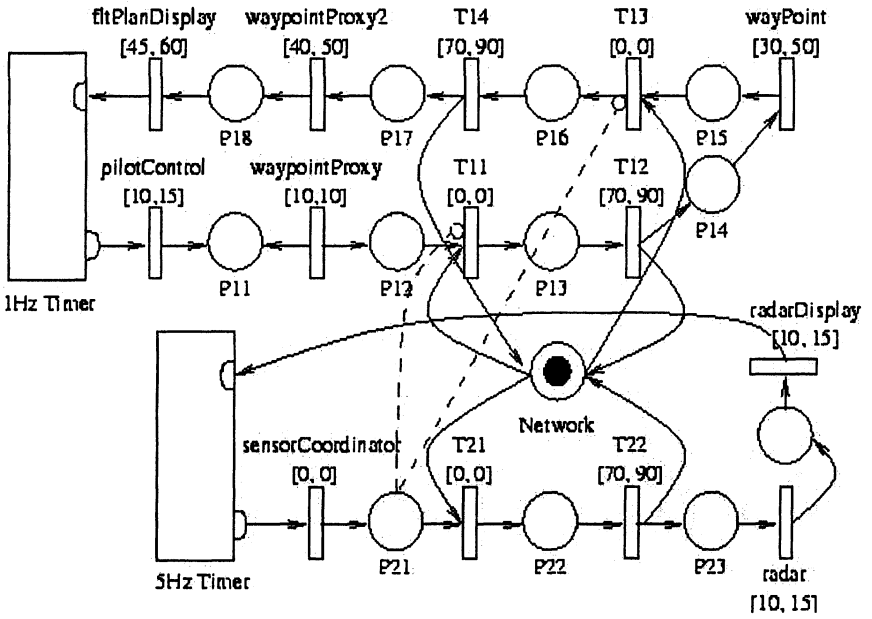

Figure 8. TPN model of the UML scenario in Figure 7.

The TPN model in Figure 8 is largely self-explanatory, with a few notable points. The three groups of transitions and places, *(T11, P13, T12), (T13, P16, T14), (T21, P22, T22),* model RT-CORBA infrastructure that processes message transmission across the network. The place *Network* models the non-preemptively scheduled network resource. The inhibitor arcs connecting *P12* to *T11* and *T13* express priorities in network resource arbitration.

By translating the TPN model into TA, we are able to use UPPAAL to check the following properties:

- The 5Hz thread has frame overrun. UPPAAL can give a diagnostic trace that shows the execution scenario that leads to the error condition, which is omitted due to space limitations.

- The end-to-end latency of the 1Hz transaction lies within [275,525]ms. UPPAAL can give execution scenarios for achieving the smallest and largest response times of the 1Hz transaction.

5. RELATED WORK

Cortes [10] proposed a mapping algorithm from PRES+ model, which is a variant of TPN with additional data handling capabilities, into HyTech [6] models. Out mapping is simpler and more compositional because we take advantage of UPPAAL's capability of having guard conditions on urgent transitions, which is not present in Hytech. Cortes' mapping algorithm can only deal with 1-safe TPNs (where each place can contain at most one token), while our algorithm can deal with non-1-safe TPNs (each place can contain more than one token) and multiple-enabledness of transitions, which are required to model task queuing and preemptive scheduling.

Wang [11] described a reachability analysis algorithm for TPN that enables computation of end-to-end system timing properties. Our TPN-to-TA translation algorithm can perform verification of more complex system properties in the form of temporal logic specifications, not just reachability. Furthermore, tool support for the algorithms described in [11] is not available.

6. CONCLUSIONS AND FUTURE WORK

In this paper we consider modelling of event-driven real-time systems with Time Petri Nets, and subsequent analysis via model-checking by defining a simple and fully-automatable mapping from TPN into Timed Automata.

A common complaint against formal methods in industry is that they are too hard to use for people without background in formal logic and mathematics. Even though graphical formalisms, such as Petri-Nets and Timed Automata, are generally easier to understand than text-based formalisms, they are usually not *broad-spectrum* models that can be applied throughout the system development life cycle. In order to use them, the designer has to manually map the regular software model into one of the *analysis specific* models, perform analysis, and then map the results back into the regular model. It is not realistic to expect industry to accept this pattern of usage in view of increasingly shorter time-to-market windows and product life-cycles, except in safety-critical industries such as the avionics and automotive industries. It also creates problems in maintaining

consistency between multiple models of the same underlying system. In order to achieve broader acceptance of formal methods, we plan to investigate integration of formal analysis techniques with informal, widely-adopted techniques such as the Unified Modeling Language (UML) [3], by using UML as the user-visible modeling formalism, and use the formal techniques as back-end analysis engines that are largely invisible to the designer. As shown in this paper, it is natural to mapping UML Collaboration Diagrams into TPN models. We are also investigating adding timing annotations to UML *Activity Diagrams* and mapping them into TPN models.

REFERENCES

[1] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, Wang Yi, "UPPAAL - A Tool Suite for Automatic Verification of Real-Time Systems", *Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, 22-24 October, 1995. LNCS 1066.

[2] S.Yovine. "Kronos: A Verification Tool for Real-Time Systems", *Springer International Journal of Software Tools for Technology Transfer*, Vol. 1, Nber. 1/2, October 1997.

[3] http://www.omg.org/uml

[4] D. Schmidt, D. Levine, S. Mungee, "The Design and Performance of Real-Time Object Request Brokers", *Computer Communications*, Volume 21, No. 4, April, 1998.

[5] D. Schmidt, D. Levine, T. Harrison, "The Design and Performance of a Real-time CORBA Object Event Service", *Proceedings of OOPSLA*, pp.434-763, 1997.

[6] T. Henzinger, P. Ho, H. Wong-Toi, "HYTECH: A Model Checker for Hybrid Systems" *Software Tools for Technology Transfer, special issue on timed and hybrid systems*, pp. 110-112, 1997.

[7] R. Alur, D.L. Dill, "A Theory of Timed Automata", *Theoretical Computer Science* 126:183-235, 1994.

[8] D. Sharp, "Object-Oriented Real-Time Computing for Reusable Avionics Software", *Proceedings of Fourth International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 185-192, 2001.

[9] H. Storrle, "An Evaluation of High-End Tools for Petri-Nets", *Technical Report, University of Munich*, June, 1998.

[10] L. Cortes, P. Eles, Z. Peng, "Verification of Embedded Systems Using a Petri Net Based Representation." *Proceedings of ISSS*, 2000, pp. 149-155

[11] J. Wang, Y. Deng, "Reachability Analysis of Real-Time Systems Using Time Petri Nets"' *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 30, No. 5, October 2000.

[12] P. Merlin, D. Farber, "Recoverability of Communication Protocols - Implication of a Theoretical Study." *IEEE Transactions on Communications*, Vol. COM-24, pp.1036-1043, Sept. 1976.