Strengthening Property Preservation in Concurrent Real-Time Systems*

Oana Florescu, Jinfeng Huang, Jeroen Voeten† and Henk Corporaal
Eindhoven University of Technology, Electrical Engineering Department
{o.florescu, j.huang, j.p.m.voeten, h.corporaal}@tue.nl

Abstract

To build a "correct" implementation from a model of a concurrent real-time system, it is important to understand the relation between the properties of the model and of its corresponding implementation. In this paper, the behaviour of a system is described with timed action sequences. Based on a notion of distance that expresses the observable property preservation between timed action sequences, we show that a stronger observable property preservation between model and implementation is obtained when urgency on the execution of observable actions is imposed over the execution of unobservable ones. Furthermore, we present a model synthesis approach and, by means of a case study, we show how it can be applied in the synthesis of real-time systems.

1. Introduction

From early stages of the design trajectory of real-time systems, it is important to predict realisation properties and to verify whether they will meet the requirements. To this end, the use of appropriate mathematical modelling techniques is required to reason about the properties of a real-time system, preventing expensive and time-consuming design iterations. Based on the results obtained from analysis of models, design and implementation decisions can be made to meet the functional and non-functional (timing) requirements of the system. However, with respect to timing behaviour, models can only approximate system implementations. Therefore, inevitable "time deviations" of the implementation from the model appear. Thus, it is desirable to understand what is the relation between properties analysed in the model and the ones present in the implementation.

One of the commonly used abstract (mathematical) structures for modelling real-time systems is the timed labelled transition system. Such a model regards the system as an entity having some state and, depending on that state, being able to engage in (action or time) transitions leading to other states. In this abstract view, the behaviour of the system is considered to be the set of all sequences of transitions that can be taken. These sequences are called *timed action sequences*. As concurrency is an important aspect, the interleaving semantics has been adopted to model simultaneous actions. Under the choice of this semantics, the timing behaviour is often formalised by a two phase execution model based on action urgency [9]. The phases of execution are: the state of the system changes either by asynchronously executing simultaneous atomic actions, without passage of time, or by letting time pass synchronously for all the components of the system when no action can be performed.

In [6], it was proved that all the properties of the implementation of a concurrent real-time system can be predicted from the properties of its model when their time deviation is bounded by some ϵ. Nevertheless, in general, real-time properties of interest for a system are related to its observable actions, which might be unnecessarily weakened by the unobservable ones. In this paper, we show how the observable property preservation can be strengthened. We define and use a notion of distance between timed action sequences that abstracts away from the unobservable actions to express preservation of observable properties between model and implementation. Furthermore, we show that an implementation that imposes urgency on the execution of the observable actions results in a smaller distance to the model than any other implementation of the same model. Based on this result, we extend an existing synthesis approach and we show on a case study that the improvement on the strength of property preservation was 80%.

The paper is organised as follows. Related research is presented in section 2. Observable property preservation is discussed in section 3, whereas section 4 shows how this can be strengthened. A model synthesis mechanism based on this result is provided in section 5, whereas experimental results obtained by applying this approach on a case study are given in section 6. Conclusions are drawn in section 7.

2. Related research

The timing semantics based on the two phase execution framework that treats system functionality and time progress in an orthogonal way is commonly adopted in design languages. Variations of this semantics have been used in different formal frameworks to model and analyse real-

*This work is being carried out as part of the Boderc project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Senter TS program.

†and the Embedded Systems Institute (*http://www.esi.nl*)

time systems, such as timed automata [2], time process algebra [10] and real-time transition systems [5]. Recently, this semantics was also integrated into design languages, such as SDL-2000 supported by TAU G2 [1], and POOSL supported by SHESim [4]. However, in current practice, the automatic generation of implementations from models lacks sufficient support to bridge the gap between the timing semantics of the modelling language and of the implementation language. As a result, the properties of the implementation cannot be deduced from the properties of the model. As an example, in the automatic implementation of an SDL-2000 model, the timing expressions rely on an asynchronous timer mechanism provided by the underlying platform. Hence, all expressions referring to some amount of time will refer to *at least* that amount of time. Timing errors are accumulated during execution, and this leads to timing failures and even functionality failures [7].

As we will show in this paper, the automatic generation of an implementation from a POOSL model overcomes the timing issue by the synchronisation of the model time with the physical time. By keeping an upper bound on the time deviation between model and implementation, properties of the implementation can be predicted from the properties of the model. An idea similar to our approach has been proposed in [13] where the notion of *Almost ASAP* semantics was introduced for timed automata. This semantics is useful when modelling real-time controllers and prevents the need for instantaneous reaction to events by allowing a certain time-bound which is left as a parameter of the model. However, in our approach, we look for an implementation that has the smallest time deviation from the model, whereas their approach sets a requirement on the deviation and gives a solution that satisfies it.

3. Observable property preservation

To appropriately analyse the properties of a concurrent real-time system, a mathematical structure is usually used to represent the system. One of the commonly used mathematical models for real-time is the timed labelled transition system. Both model and implementation of a concurrent real-time system are viewed as timed labelled transition systems whose behaviours are given as sequences of action transitions that are taken at certain timestamps. These sequences of pairs, written as

$$\xi = (a_0, t_0)\ (a_1, t_1) \ldots\ (a_n, t_n) \ldots = (\overline{a}, \overline{t})$$

where a_i is the label of an action transition and t_i is the timestamp when this transition is taken, are called timed action sequences. To differentiate from the observable action transitions, a special label is used for the internal (unobservable) transitions, τ.

The properties of interest in the analysis of a real-time system are related to which actions can be observed in the system and at which moments in time. A system satisfies a real-time property φ, formalised using temporal logic MTL [8], if and only if each timed action sequence ξ in the behaviour satisfies it ($\xi \models \varphi$). As models are approximations of system implementations with respect to timing behaviour, time deviations appear between the behaviour of a model and the behaviour of the implementation. To investigate the property preservation between them, we define a metric over timed action sequences to measure the observable deviation between model and implementation.

The ***observable distance between two timed action sequences***, $\xi = (\overline{a}, \overline{t})$ and $\xi^{'} = (\overline{a^{'}}, \overline{t^{'}})$, is defined as

$$d^*(\xi, \xi^{'}) = \begin{cases} \sup\{|t_i - t_i^{'}| \mid a_i \neq \tau\}, & \text{if } \overline{a} = \overline{a^{'}} \\ \infty, & \text{otherwise.} \end{cases}$$

The observable distance between two equivalent timed action sequences is the least upper bound of the absolute difference between the corresponding timestamps of observable actions. This metric is used to express observable property preservation between timed action sequences: $\xi \models \varphi \Rightarrow \xi^{'} \models \varphi_{2*d^*(\xi,\xi^{'})}$. Hence, $\xi^{'}$ satisfies a $2{*}d^*(\xi, \xi^{'})$ weakening of φ. Due to space limitations, the proof for the improvement of property preservation by abstracting from the unobservable actions can be found in [3].

4. Strengthening property preservation

As a model is an abstraction of the implementation of a system in which instantaneous execution of actions is assumed, different actions may have the same timestamp. However, in the implementation, because the execution of each action takes physical time, no two actions can have the same timestamp. Therefore, we define

$$T_{Exec} : Act \rightarrow \mathbb{R}^+$$

as a function that associates to each action a positive real number representing the time it takes to execute that action on a target platform.

Moreover, let $\xi_M = (\overline{a}, \overline{t})$ be a timed action sequence in the model of a system. Then, its corresponding timed action sequence in the implementation on a target platform is $\xi_I = (\overline{a}, \overline{t^{'}})$. If in the model there exists i, j such that $t_i < t_{i+1} = \ldots = t_j < t_{j+1}$ and assuming that there exists a mechanism of synchronisation between physical time and model time (as fig. 2 will show), then $t_{i+1}^{'} > t_{i+1}$ and

$$\begin{aligned} t_{i+2}^{'} &= t_{i+1}^{'} + T_{Exec}(a_{i+2}) \\ &\ldots \\ t_j^{'} &= t_{i+1}^{'} + \textstyle\sum_{k=i+2}^{j} T_{Exec}(a_k) \end{aligned}$$

Therefore, the observable distance between ξ_M and ξ_I is induced by the implementation timestamp of the last observable action executed at the same model timestamp. Thus

$$d^*(\xi_M, \xi_I) = sup_{i,j}\{t_{i+1}^{'} - t_{i+1} + \sum_{k=i+2}^{l} T_{Exec}(a_k) \mid$$

$$t_{i-1} < t_i = \ldots = t_j < t_{j+1}, a_{l+1} = \ldots = a_j = \tau\}$$

Assume that the value of the observable distance between model and implementation is induced at a model timestamp for which exists some i, j such that $t_i < t_{i+1} = \ldots = t_j < t_{j+1}$. Then, there exists $i < l$ such that $a_l \neq \tau$ and $a_{l+1} = \ldots = a_j = \tau$, hence

$$d^*(\xi_M, \xi_I) = t_{i+1}^{'} - t_{i+1} + \sum_{k=i+2}^{l} T_{Exec}(a_k)$$

Moreover, assume there exists some $i < m < l$ such that $a_m = \tau$, an unobservable action which is independent from the following observable actions. Then, it is allowed to postpone this τ such that the sequence of actions becomes

$$a_0 \dots a_i \dots a_{m-1} a_{m+1} \dots a_l a_m a_{l+1} \dots$$

whereas the model timestamps remain the same. Under these circumstances, the observable distance is reduced because in the implementation

$$t_l^{'} = t_{i+1}^{'} + \sum_{k=i+2}^{m-1} T_{Exec}(a_k) + \sum_{k=m+1}^{l} T_{Exec}(a_k)$$

This means that by postponing one unobservable action, the observable property preservation is strengthened.

By induction, the smallest distance between model and implementation is obtained when all possible unobservable actions in all the timed action sequences of the model are postponed to be executed after the execution of the observable actions which appear at the same model timestamp. Thus, strengthening of property preservation between model and implementation is obtained by imposing, where possible, urgency on the execution of observable actions over the unobservable actions.

Based on the interleaving semantics which is assumed in this paper, urgency of observable actions execution over the unobservable actions is often possible. For a model $M1$, an observation equivalent model $M2$ can be built by considering only those paths in which observable actions are taken before τ actions. As $M1$ and $M2$ are observational equivalent and thus satisfy the same properties, as shown in fig. 1, the property preservation is weakened less in the implementation $R2$ than in $R1$ ($\epsilon_2 < \epsilon_1$). A synthesis strategy that, given a model $M1$, is able to generate such an observation equivalent model $M2$ and implement it, enables strengthening of observable property preservation between model and implementation.

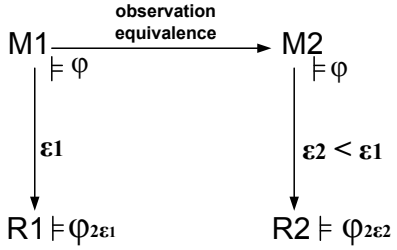

Figure 1. Synthesis strategy based on observation equivalence

5. A predictable development approach

To guarantee preservation of model properties in the implementation of a system, it is required that the timed action sequences of the implementation are the same as those in the model and that the time deviations between activations of the corresponding observable actions in the implementation and the model are bounded from above.

In this section, we present a model synthesis approach that is based on a design language suitable for modelling concurrent real-time systems, called the Parallel Object-Oriented Specification Language (POOSL) [11] whose semantics is based on timed labelled transition system. The implementation of the two phase execution model for the timed labelled transition system in a simulation tool, SHESim, is achieved by adopting the process execution trees (PETs) [12]. The state of each process is represented by a tree structure, where each leaf is a statement and internal nodes represent compositions of their children. The correctness of PET with respect to the semantics of the POOSL language was formally proved in [4]. Details about PETs implementation and behaviour can be found in [12].

Based on the results proved in this paper, with a proper design time annotation of the model, during the evolution of the system, PETs can send *observable action requests*, *unobservable action requests* and/or *delay requests* to the scheduler. The PET scheduler, whose behaviour is described by the algorithm in fig. 2, asynchronously grants all eligible atomic observable actions. When no observable action is eligible, an unobservable action request is granted, and then the observable action requests list is checked again. When no action request of any kind is available, time passes synchronously for all PETs until some action becomes eligible again. In this way, an observation equivalent system of the original model is built by always choosing the path that contains eligible observable actions in front of unobservable actions.

```
PETSCHEDULER()
  list observableActions;
  list unobservableActions;
  list delays;
  while true do
    while observableActions.notEmpty() do
      observableActions.getAsynchronously()–>grant();
    if unobservableActions.nonEmpty() then
      unobservableActions.getAsynchronously()–>grant();
      continue;
    else
    if delays.nonEmpty() then
      modelTime = modelTime + delays.getFirst()–>amountOfTime();
      /* synchronisation between model and physical time */
      wait_until physicalTime == modelTime;
      continue;
    else
      DEADLOCK();
      return.
```

Figure 2. The PET scheduler

Rotalumis is a tool that takes a POOSL model and generates the executable code for the target platform. Each PET in the model is directly translated into a C++ structure whose behaviour is the same as the PET implemented in SHESim. As a result, the generated implementation exhibits exactly the same behaviour as the model, if interpreted in model time domain. On the other hand, since the progress of model time is monotonicly increasing, which is consistent with the progress of physical time, the action order observed in model time domain is consistent with that in physical time. To obtain the same (or similar) quantitative timing behaviour in physical time as in model time, the PET scheduler tries to synchronise model time with phys-

ical time during the running of the implementation. This ensures that the execution of the implementation is always as close as possible to a trace in the model with respect to the observable distance between timed action sequences.

6. Experimental results

A case study of the control of a motion system made of two devices running in parallel, whose model is presented in fig. 3, was considered in this paper. Such a system is representative, for example, for the control of a part of a printer where several motors must be controlled concurrently. The goal of this experiment was to correctly synthesise the control part of the system such that its stability is guaranteed.

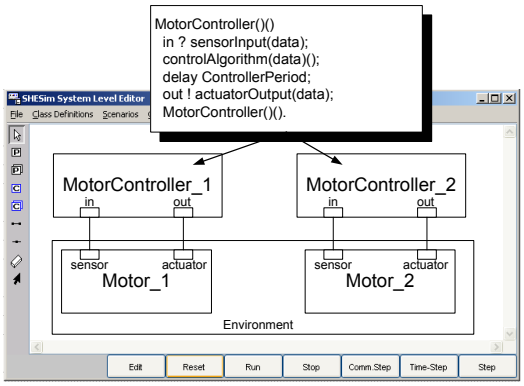

Figure 3. Model of the system case study

To ensure the stability of the control of the two motors system, two required real-time properties must be satisfied. The MTL formulas corresponding to these properties are $\Box(p_1 \rightarrow \Diamond_{[0.9,1.1]} q_1)$ and $\Box(p_2 \rightarrow \Diamond_{[1.9,2.1]} q_2)$, where p_1, p_2 represent the receiving of $sensorInput$ message and q_1, q_2 the sending of $actuatorOutput$ message for motor 1 and 2 respectively. These properties refer to sending a message to the actuator within a certain amount of time after each sensor message. Since the system is simple, we could manually check that the control part in the model satisfies $\Box(p_1 \rightarrow \Diamond_{[1,1]} q_1)$ and $\Box(p_2 \rightarrow \Diamond_{[2,2]} q_2)$.

To enable automatic model synthesis, the analysis model was transformed into a synthesis model (detailed explanations can be found in [3]). A first experiment was done using the original POOSL model synthesis approach which makes no distinction between observable and unobservable actions. The maximum time deviation obtained during several hours of continuous behaviour was $0.213ms$. This implies that the real-time properties are weakened up to $\Box(p_1 \rightarrow \Diamond_{[0.574,1.426]} q_1)$ and $\Box(p_2 \rightarrow \Diamond_{[1.574,2.426]} q_2)$ respectively, thus not meeting the requirements.

In a second experiment, we have applied the synthesis approach presented in this paper. Reading from the sensor and writing to the actuator were labelled as observable actions of the system, whereas the control algorithm computations were labelled as unobservable from outside the system. During several hours of continuous behaviour, the maximum obtained observable distance between model and the generated implementation was $0.042ms$. Thus, the properties satisfied by the implementation are $\Box(p_1 \rightarrow \Diamond_{[0.916,1.084]} q_1)$ and $\Box(p_2 \rightarrow \Diamond_{[1.916,2.084]} q_2)$, which fulfill the requirements. Due to the decrease with 80% in the time deviation, the control strategy designed is implementable using the newly proposed approach.

7. Conclusions

In this paper, we have used timed action sequences to describe the behaviour of a real-time system. We defined a notion of distance as a metric to express the strength of observable property preservation between model and implementation. We proved that an implementation in which observable actions can be executed before unobservable ones has a smaller distance to the model than any other implementation of the same model. Moreover, we have incorporated this result into an existing predictable development method and showed on a case study that the proposed approach improved the strength of property preservation with 80% and succeeded in generating an implementation that fulfilled the requirements.

As future work, we aim at integrating scheduling policies in the synthesis approach to deal with concurrent time-intensive computations. Moreover, as the synthesis strategy is conceived for a single processor architecture, we want to extend it for distributed systems.

References

[1] Telelogic TAU2. `http://www.telelogic.com/corp/products/tau/index.cfm`.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] O. Florescu, J. Huang, J. Voeten, and H. Corporaal. Towards stronger property preservation in real-time systems synthesis. Technical Report ESR-2006-02, 2006.

[4] M. G. Geilen. *Formal Techniques for Verification of Complex Real-Time Systems*. PhD thesis, Eindhoven University of Technology, 2002.

[5] T. A. Henzinger, Z. Manna, and A. Pnueli. An Interleaving Model for Real Time. In *Proc. of the 5th Jerusalem Conference on Information Technology*, pages 717–730, 1990.

[6] J. Huang, J. Voeten, and M. Geilen. Real-time property preservation in concurrent real-time systems. In *Proc. of 10th RTCSA*, 2004.

[7] J. Huang, J. P. Voeten, A. Ventevogel, and L. J. van Bokhoven. Platform-independent design for embedded real-time systems. In *Proc. of FDL*, 2003.

[8] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[9] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *Proc. of the Real-Time: Theory in Practice, REX Workshop*, 1992.

[10] X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.

[11] POOSL. `http://www.es.ele.tue.nl/poosl`.

[12] L. J. van Bokhoven, J. P. Voeten, and M. C. Geilen. Software synthesis for system level design using process execution trees. In *Proc. of 25th Euromicro Conference*, 1999.

[13] M. D. Wulf, L. Doyen, and J.-F. Raskin. Systematic implementation of real-time models. LNCS 3582. Springer, 2005.