technische universiteit eindhoven

Towards Stronger Property Preservation in Real-Time Systems Synthesis

Oana Florescu, Jinfeng Huang, Jeroen Voeten, Henk Corporaal

ES Reports
ISSN 1574-9517

ESR-2006-02
18 May 2006

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

http://www.es.ele.tue.nl/esreports
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

Towards Stronger Property Preservation in Real-Time Systems Synthesis*

Oana Florescu, Jinfeng Huang, Jeroen Voeten, Henk Corporaal

Eindhoven University of Technology
Electrical Engineering Department
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
o.florescu@tue.nl

Abstract

A key aspect in concurrent real-time system development is to build a model from which a "correct" implementation can be synthesised. Hence, it is important to understand the relation between the properties of a model and of its corresponding implementation. In this paper, we use timed action sequences to describe the behaviour of a real-time system. We first define a notion of distance as a metric to express the observable property preservation between timed action sequences. Furthermore, considering both model and implementation as sets of timed action sequences, we show that a smaller distance between them, and hence a stronger observable property preservation, is obtained when urgency on the execution of observable actions is imposed over the execution of unobservable ones. Based on this result, we extend a previous model synthesis approach to generate from a model an implementation with stronger property preservation. By means of a case study, we show how the proposed approach can be applied to the development of real-time systems.

1. Introduction

Embedded systems, found nowadays in cars, airplanes, printer/copier machines, or medical devices, employ different real-time software components to synchronise and coordinate different processes and activities. The correctness of real-time systems is related to both the value of the computations results, as well as on the moments in time when results are issued. Under the increasing time-to-market constraint, the intricate behaviour of such systems must meet various timing constraints, either for people's safety or simply to ensure things work correctly.

From early stages of the design trajectory, it is important to predict the properties of a system and to verify whether it will meet its requirements. To this end, use of appropriate mathematical techniques for modelling 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 such that the functional and non-functional (timing) requirements of the system are met. However, models are only approximations of system implementations with respect to timing behaviour. 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) structure for real-time system modelling 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

*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.

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. Moreover, the implementation of such a model is also viewed as a set of timed action sequences. In [9], it was proved that the properties of the implementation can be predicted from the properties of the model when their time deviation is bounded by some ϵ. This result was generalised in [8] for concurrent real-time systems with interleaving semantics and a property preserving model synthesis approach has been conceived. This approach relies on a mechanism that generates timed action sequences in the implementation with the same sequence of actions as their corresponding timed action sequences in the model. Moreover, a synchronisation of the model time with the physical time ensures that the time deviation between model and implementation is bounded by some ϵ. Thus, each property in the implementation is a 2ϵ-weakening of the corresponding property of the model.

Nevertheless, in general, real-time properties of interest for a system are related to its observable actions. As the model synthesis approach presented in [8] considers that all possible (observable and unobservable) actions of the system influence the strength of property preservation, observable properties might be unnecessarily weakened.

Contributions of the paper. In this paper, we show how the observable property preservation can be strengthened. We define a notion of distance between timed action sequences that abstracts away from the unobservable actions (which might also be time-intensive). We use this distance as a metric to express preservation of observable properties between timed action sequences, and hence 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 the existing synthesis approach to ensure strengthening of property preservation between model and implementation. Our technique is applicable to any kind of system and it is not restricted to particular types of real-time components. By means of a motion control system case study, which combines short observable actions with time-intensive unobservable computations, we show that the proposed approach improved the strength of property preservation with 80%. Moreover, the case study illustrates that there are models whose requirements could not be met by the implementation generated using the previous synthesis approach, whereas the synthesis approach presented in this paper can now correctly implement them (i.e. fulfilling the requirements).

The paper is organised as follows. Section 2 introduces the necessary mathematical preliminaries with respect to real-time systems models, real-time properties and property preservation. The observable property preservation is discussed in section 3. In section 4, we show that an implementation in which observable actions urgency is imposed over the execution of unobservable actions has the smallest time deviation from the model. A model synthesis mechanism based on this result is provided in section 5, whereas experimental results obtained by applying this approach are presented in section 6. Related research is presented in section 7 and conclusions are drawn in section 8.

2. Preliminaries

In this section, we provide the basic concepts needed to understand the contributions of this work. We present the mathematical structure used for modelling real-time systems in subsection 2.1. A definition of real-time properties is given in subsection 2.2, whereas a summary of the results obtained previously with respect to property preservation is provided in subsection 2.3.

2.1. Real-time systems models

To properly reason about properties of real-time systems, an abstract (mathematical) model of them is needed. A possible abstract view on the behaviour of a system is to regard it as an entity having some internal state and, depending on that state, is able to engage in transitions leading to other states. A mathematical structure that captures this abstract view on a real-time system is the timed labelled transition system. A timed labelled transition

system is a 5-tuple

$$\mathcal{T} = (S, Act, T^+, \xrightarrow[Act]{}, \xrightarrow[T^+]{})$$

consisting of a set S of states, a set Act of *actions*, a positive time domain T^+ which is included in the set of real positive numbers $\mathbb{R}^+$, an action transition relation $\xrightarrow[Act]{} \subseteq S \times Act \times S$ and a time transition relation $\xrightarrow[T^+]{} \subseteq S \times T^+ \times S$. An action transition $s_1 \xrightarrow[Act]{a} s_2$ denotes a possible transition of the system from state s_1 to state s_2 by performing action a. A time transition $s_1 \xrightarrow[T^+]{t} s_2$ denotes a transition of the system from state s_1 to state s_2 by letting an amount $t > 0$ of time to pass. In a timed labelled transition system model, action transitions ($\xrightarrow[Act]{}$) are assumed to be instantaneous (taking no time), and the passage of time is denoted by time transitions ($\xrightarrow[T^+]{}$). In the remainder, we often leave out the subscript identifying the transition relation as being the action transition relation or the time transition relation. It will be clear from the label (being from Act or T^+) which relations is intended.

As concurrency is an important aspect of real-time systems, the interleaving semantics has been adopted in many formal frameworks to model simultaneous actions. In this model, parallel actions are represented as interleaved sequential actions. This means that two parallel actions $a||b$ are represented by two sequential actions $a; b$ and $b; a$. Such sequentialisation of concurrent actions facilitates calculations and yields correctness proofs in a linear form [5].

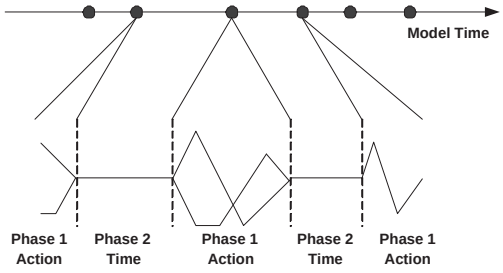

Figure 1. Two phase execution model

Under the choice of the interleaving semantics, the timing behaviour of a concurrent system is often formalised by a two phase execution model based on action urgency [14]. As depicted in fig. 1, the phases of the execution are: the state of the system changes either by asynchronously executing simultaneous atomic actions based on the interleaving semantics, without passage of time (phase 1), or by letting time pass synchronously for all the components of the system when no action can be performed (phase 2). As soon as an action becomes available, the first phase is resumed. Such a model assumes an abstract notion of time, called *virtual* or *model* time.

Example. *For a system consisting of two independent parallel components P (fig. 2.a) and Q (fig. 2.b), under the interleaving semantics and based on action urgency, the timed labelled transition system that can be constructed for the system $P||Q$ is shown in fig. 2.c. Actions $in1$, $in2$, $out1$ and $out2$ are considered communication actions, whereas $computation1$ and $computation2$ represent some internal calculations. Note that due to the action urgency, as component P outputs its result at time 1, the value of the time transition between states $s7$ and $s8$ is 1 such that component Q can output its result at time 2.*

Path through a transition system. A (possibly infinite) sequence

$$s_0 \xrightarrow{\gamma_0} s_1 \xrightarrow{\gamma_1} s_2 \ \ldots \ s_{m-1} \xrightarrow{\gamma_{m-1}} s_m \ldots$$

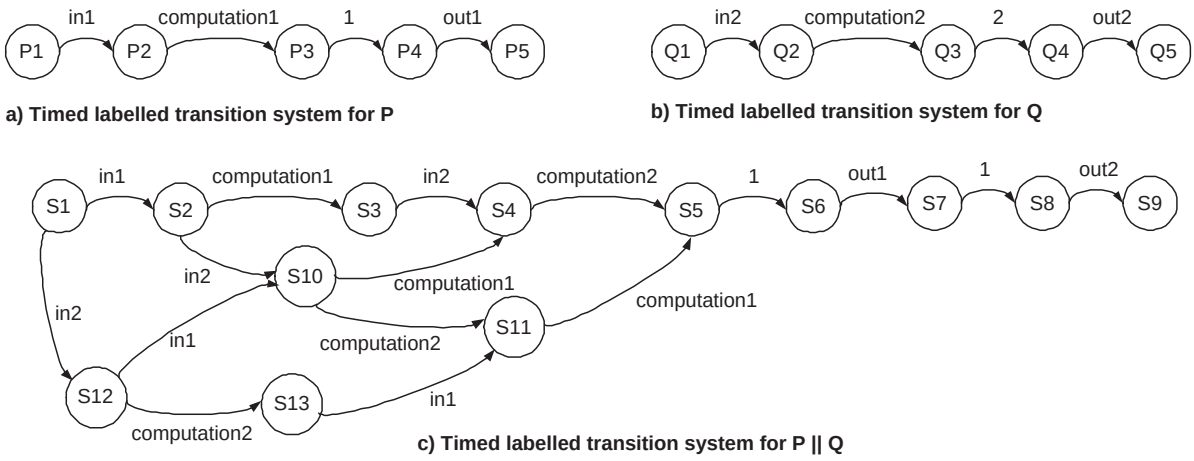

Figure 2. Timed labelled transition systems

where for all i, $s_i \in S$, $\gamma_i \in Act \cup T^+$, and $s_i \xrightarrow{\gamma_i} s_{i+1} \in \xrightarrow[Act]{} \cup \xrightarrow[T^+]{}$, is called a path from s_0 through the timed labelled transition system $\mathcal{T}$. If a path cannot be extended anymore because it ends in a state without outgoing transitions, it is called a *full path*.

Example.

$$s_1 \xrightarrow{in1} s_2 \xrightarrow{computation1} s_3 \xrightarrow{in2} s_4 \xrightarrow{computation2} s_5 \xrightarrow{1} s_6 \xrightarrow{out1} s_7 \xrightarrow{1} s_8 \xrightarrow{out2} s_9 \qquad \text{(path.1)}$$

is a full path through the timed labelled transition system presented in fig. 2.c. Another possible full path is

$$s_1 \xrightarrow{in1} s_2 \xrightarrow{in2} s_{10} \xrightarrow{computation1} s_4 \xrightarrow{computation2} s_5 \xrightarrow{1} s_6 \xrightarrow{out1} s_7 \xrightarrow{1} s_8 \xrightarrow{out2} s_9 \qquad \text{(path.2)}$$

Timed action sequence. From a path $s_0 \xrightarrow{\gamma_0} s_1 \xrightarrow{\gamma_1} s_2 \ldots s_{m-1} \xrightarrow{\gamma_{m-1}} s_m \ldots$ through a timed labelled transition system, a timed action sequence

$$\xi = (a_0, t_0)\ (a_1, t_1) \ldots\ (a_n, t_n) \ldots$$

is derived by picking up the action transitions labels together with the moments in time when they are taken according to the interleaving semantics, where:

- $\forall i, a_i \in Act$ and $\exists j\ a_i = \gamma_j \in Act$ if $\forall k < j$ with $\gamma_k \in Act,\ \exists l < i, a_l = \gamma_k$
- $t_0 = \sum_{j=0}^{k-1} \gamma_j$ iff $a_0 = \gamma_k$ and $\forall j < k,\ \gamma_j \in T^+$
- $\forall\ i > 0, t_i = t_{i-1} + \sum_{l=j+1}^{k-1} \gamma_l$, where $a_{i-1} = \gamma_j$, $a_i = \gamma_k$ and $\forall j < l < k,\ \gamma_l \in T^+$.

A timed action sequence derived from a full path is called *maximal* timed action sequence. A maximal timed action sequence is finite.

Example. *The timed action sequence derived from* (path.1) *in the previous example is*

$$\xi = (in1, 0)\ (computation1, 0)\ (in2, 0)\ (computation2, 0)\ (out1, 1)\ (out2, 2) \qquad \text{(seq.1)}$$

This is a maximal timed action sequence because it is derived from a full path.

In the remainder, for the simplicity of reading, we write $\overline{a} = a_0 a_1 \dots a_n \dots$ to denote the sequence of actions and $\overline{t} = t_0 t_1 t_2 \dots t_n \dots$ to represent the sequence of timestamps in a timed action sequence. Thus, $\xi = (\overline{a}, \overline{t})$. Moreover, we define $\mathcal{N}(\overline{t})$ to represent the number of timestamps in a sequence. $\mathcal{N}(\overline{t})$ is finite, in case of maximal timed action sequences, or countable infinite. Note that, due to the two-phase execution model, there exists indexes $i, j < \mathcal{N}(\overline{t})$ such that $t_i < t_{i+1} = \dots = t_j < t_{j+1}$.

Real-time system behaviour. Given a real-time system modelled by a timed labelled transition system $\mathcal{T}$, the behaviour of the system is the set of all timed action sequences derived from all possible paths through $\mathcal{T}$.

2.2. Real-time properties

The goal of the analysis of a real-time system is the verification of its real-time properties. These properties are related to which actions (events) happen and when. Properties of real-time systems are often formalised by temporal logics. In this paper, we assume that the properties are formalised using temporal logic MTL [11]. It is said that the behaviour of a system satisfies a real-time property φ expressed as an MTL formula if and only if for all timed action sequences ξ in the behaviour of the system, ξ satisfies φ (written as $\xi \models \varphi$).

Example. *For the system shown in fig. 2.c, a real-time property satisfied by the behaviour of the system is*

$$\varphi = p \rightarrow \Diamond_{[1,1]} q \qquad \text{(prop.1)}$$

where p denotes the property that action transition $in1$ is taken, q denotes the property that action transition $out1$ is taken, $\Diamond_{[1,1]}$ means that, eventually*, after p becomes true, q becomes true within the time interval $[1, 1]$ (which in this case contains a single point). Assuming that the environment is always ready to interact with the system, then, eventually, if the first input is received, after 1 unit of time, the corresponding output data is sent. Note that, in all paths through the transition system, $in1$ is taken at moment 0 whereas $out1$ at 1 due to the assumption of instantaneous actions. Hence all timed action sequences of the system satisfy property* (prop.1)*, thus the behaviour of the system satisfies it.*

Furthermore, a notion of weakening of properties can be used (see [9] for a formal definition). For example, a 0.1-weakening of property (prop.1) is

$$\varphi_{0.1} = p \rightarrow \Diamond_{[0.9,1.1]} q$$

whereas a 0.9-weakening of it would be

$$\varphi_{0.9} = p \rightarrow \Diamond_{[0.1,1.9]} q$$

Furthermore, property $\varphi_{0.9}$ is weaker than $\varphi_{0.1}$ since the satisfaction of $\varphi_{0.1}$ by a timed action sequence always implies the satisfaction of $\varphi_{0.9}$. An ϵ-weakening of property (prop.1) can be written as

$$\varphi_\epsilon = p \rightarrow \Diamond_{[1-\epsilon,1+\epsilon]} q$$

Note that in case $1 - \epsilon < 0$, the ϵ-weakening of the formula is $p \rightarrow \Diamond_{[0,1+\epsilon]} q$.

In [9] it was proved that if a timed action sequence satisfies a real-time property, then it also satisfies an ϵ-weakening of it. The larger the value of ϵ, the more weakened the real-time property is.

2.3. Property preservation

The model and the implementation of a concurrent real-time system can be viewed as timed labelled transition systems. Thus, their behaviour is given by a set of timed action sequences. 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, a metric over timed action sequences was defined in [9] to measure the deviation between model and implementation.

Two timed action sequences which have the same sequences of actions (written as $\overline{a} = \overline{a'}$) are called equivalent.

Distance between timed action sequences. Let $\xi = (\overline{a}, \overline{t})$ and $\xi^{'} = (\overline{a'}, \overline{t'})$ be two timed action sequences. The distance between them is

$$d(\xi, \xi^{'}) = \begin{cases} \sup\{|t_i - t_i^{'}| \mid i < \mathcal{N}(\overline{t})\}, & \text{if } \overline{a} = \overline{a'} \\ \infty, & \text{otherwise.} \end{cases}$$

The distance between two equivalent timed action sequences is the least upper bound of the absolute difference between the corresponding timestamps of any action.

Example. *Consider the timed action sequence* (seq.1) *from the model in fig. 2.c and assume that its corresponding timed action sequence in the implementation is*

$$\xi^{'} = (in1, 0.01)\ (computation1, 0.4)\ (in2, 0.41)\ (computation2, 0.8)\ (out1, 1.1)\ (out2, 2.1) \qquad \text{(seq.2)}$$

As they are equivalent, the distance between them is calculated as follows

$$d(\xi, \xi^{'}) = \sup\{|0 - 0.01|, |0 - 0.4|, |0 - 0.41|, |0 - 0.8|, |1 - 1.1|, |2 - 2.1|\} = \\ = \sup\{0.01, 0.4, 0.41, 0.8, 0.1, 0.1\} = 0.8$$

Two timed action sequences whose distance is bounded by some ϵ are called ϵ-close. In [9], it was proved that when two timed action sequences, ξ and $\xi^{'}$, are ϵ-close and a certain real-time property is satisfied by one of the sequences ($\xi \models \varphi$), the other one satisfies a 2ϵ-weakening of it ($\xi^{'} \models \varphi_{2\epsilon}$).

Example. *From the action sequences in the previous example, we have that property* (prop.1) *is weakened up to* 1.6 *in the implementation, thus the implementation* $\xi^{'}$ *satisfies*

$$\varphi_{1.6} = p \rightarrow \Diamond_{[0,2.6]} q$$

To predict the properties of an implementation based on the properties of a model of a concurrent real-time system, a notion of distance between them is used.

Distance between model and implementation. As model and implementation are viewed as sets of equivalent timed action sequences, M and I respectively, the distance between them is

$$d(M, I) = \sup_{\xi \in M, \xi^{'} \in I} \{d(\xi, \xi^{'}) \mid \xi = (\overline{a}, \overline{t}) \text{ and } \xi^{'} = (\overline{a}, \overline{t'})\}$$

Assuming that the distance is bounded by some ϵ (*ϵ-hypothesis*) and the model satisfies a real-time property, then the implementation satisfies a 2ϵ-weakening of it. The smaller the value of ϵ, the stronger the preservation of properties between model and implementation is.

A model synthesis approach that complies with the ϵ-hypothesis guarantees that all properties of the model are preserved by the generated implementation up to twice the distance between model and implementation. Based on the metric defined for expressing the property preservation between timed action sequences, all actions specified in the model influence the preservation of properties in the implementation.

3. Observable property preservation

Usually, the relevant real-time properties of a system are the ones related to which *observable* actions a user can see by interacting with it. Thus, in order to express observable property preservation between model and implementation, we need to consider the observable behaviour of the system which is defined with respect to an

observer. Depending on what properties are of interest, at a coarse level of granularity, an observer can "see" the inputs and the outputs of the whole system. At a finer level of granularity, properties related to the communication taking place between the components of the system might also be interesting, thus the related actions are considered observable. However, typically the computations that different components of the system perform are not observable, and they can be labelled with τ, a typical label for internal transitions of a transition system. Nevertheless, the synthesis approach that will be presented in this paper is not restricted to a certain level of granularity with respect to the discrimination between observable and unobservable actions. Thus, a formal definition of the observable and the unobservable actions is outside the scope of this paper. In this work, we assume that based on the properties of interest for a system, the designer defines which are considered observable and which are unobservable actions.

Labelling of actions. Let L be a labelling function defined on the set Act of actions in a timed labelled transition system $\mathcal{T}$ as

$$L(a) = \begin{cases} a, & \text{if } a \text{ is observable} \\ \tau, & \text{otherwise.} \end{cases}$$

Based on this labelling of actions, a timed action sequence ξ can be written as

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

Example. *As* $computation1$ *and* $computation2$ *are considered unobservable actions in the system from fig. 2.c, the timed action sequence* (seq.1) *can be written as*

$$\xi = (in1, 0)\ (\tau, 0)\ (in2, 0)\ (\tau, 0)\ (out1, 1)\ (out2, 2)$$

In a model synthesis approach complying with the ϵ-hypothesis, the metric defined in the previous subsection considers that all actions, both observable and unobservable, affect property preservation between model and implementation. Thus, observable properties get, possibly unnecessarily, weakened. Therefore, we define a new distance metric based on the labelling of actions to express observable property preservation between them.

Observable distance between timed action sequences. Let $\xi = (\overline{a}, \bar{t})$ and $\xi^{'} = (\overline{a^{'}}, \overline{t^{'}})$ be two timed action sequences. The observable distance between them is defined as

$$d^*(\xi, \xi^{'}) = \begin{cases} \sup\{|t_i - t_i^{'}| \mid i < \mathcal{N}(\bar{t}) \text{ and } L(a_i) \neq \tau\}, & \text{if } \overline{a} = \overline{a^{'}} \\ \infty, & \text{otherwise.} \end{cases}$$

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. In the following, we show that, by abstracting from the unobservable actions, the strength of property preservation may be improved.

Formally, given $\xi = (\overline{a}, \bar{t})$ and $\xi^{'} = (\overline{a^{'}}, \overline{t^{'}})$ two timed action sequences, if $d(\xi, \xi^{'}) = \epsilon \in \mathbb{R}^+$, then $d^*(\xi, \xi^{'}) \leq \epsilon$.

The proof comes directly from the definitions. Because $d(\xi, \xi^{'}) = \epsilon \in \mathbb{R}^+$, it means that $\overline{a} = \overline{a^{'}}$, thus ξ and $\xi^{'}$ are equivalent. Moreover, $\overline{L(a)} = \overline{L(a^{'})}$. If exists some $i < \mathcal{N}(\bar{t})$ with $L(a_i) = \tau$, then

$$\{|t_i - t_i^{'}| \mid i < \mathcal{N}(\bar{t}) \text{and} L(a_i) \neq \tau\} \subset \{|t_i - t_i^{'}| \mid i < \mathcal{N}(\bar{t})\}$$

From this,

$$d^*(\xi, \xi^{'}) = \sup\{|t_i - t_i^{'}| \mid L(a_i) \neq \tau\} \leq \sup\{|t_i - t_i^{'}|\} = d(\xi, \xi^{'}) = \epsilon$$

Example. *In the model from fig. 2.c,* $in1$*,* $in2$*,* $out1$ *and* $out2$ *are considered observable actions and property* (prop.1) *is an example of a real-time observable property of the system. Consider the timed action sequence* (seq.1) *from the model and its corresponding timed action sequence* (seq.2) *in the implementation. Based on the labelling function, they can be written as*

$$\xi = (in1, 0)\ (\tau, 0)\ (in2, 0)\ (\tau, 0)\ (out1, 1)\ (out2, 2)$$
$$\xi^{'} = (in1, 0.01)\ (\tau, 0.4)\ (in2, 0.41)\ (\tau, 0.8)\ (out1, 1.1)\ (out2, 2.1)$$

The observable distance between them is calculated as follows

$$d^{*}(\xi, \xi^{'}) = \sup\{|0 - 0.01|, |0 - 0.41|, |1 - 1.1|, |2 - 2.1|\} =$$
$$= \sup\{0.01, 0.41, 0.1, 0.1\} = 0.41$$

Thus, the property (prop.1) *is weakened actually only up to* 0.82 *in the implementation, thus the implementation* $\xi^{'}$ *satisfies*

$$\varphi_{0.82} = p \rightarrow \Diamond_{[0.18, 1.82]} q$$

All property preservation results proved in [9] and that are based on the notion of distance between timed action sequences also hold for the observable distance metric defined here. As proved above, with this new metric that discriminates between observable and unobservable actions, observable property preservation is less affected by the unobservable actions.

4. Strengthening property preservation

In order to show how observable property preservation can be strengthened even further, we need to introduce a notion of observation equivalence between timed action sequences in subsection 4.1. Then, we show in subsection 4.2 that by postponing, if possible, the execution of a τ action in favour of an observable action, the property preservation is strengthened. Based on observation equivalence, we present in subsection 4.3 a synthesis strategy to obtain stronger property preservation between model and implementation.

4.1. Observation equivalence between timed action sequences

Based on the labelling of actions, we define observation equivalence between timed action sequences.

Observation equivalent timed action sequences. Two timed action sequences $\xi_1 = (\overline{a^1}, \overline{t^1})$ and $\xi_2 = (\overline{a^2}, \overline{t^2})$ are called observation equivalent if

- for all $i < \mathcal{N}(\overline{t^1})$ where $L(a_i^1) \neq \tau$, there exists $j < \mathcal{N}(\overline{t^2})$ with $L(a_i^1) = L(a_j^2)$ and $t_i^1 = t_j^2$
- for all $j < \mathcal{N}(\overline{t^2})$ where $L(a_j^2) \neq \tau$, there exists $i < \mathcal{N}(\overline{t^1})$ with $L(a_j^2) = L(a_i^1)$ and $t_j^2 = t_i^1$
- for any $i < j < \mathcal{N}(\overline{t^1})$ exists $k < l < \mathcal{N}(\overline{t^2})$ such that $L(a_i^1) = L(a_k^2) \neq \tau$ with $t_i^1 = t_k^2$ and $L(a_j^1) = L(a_l^2) \neq \tau$ with $t_j^1 = t_l^2$.

Observation equivalent timed action sequences have the same timing and ordering of observable action. Therefore, as properties are related to the observable actions of a system, two observation equivalent timed action sequences have the same observable properties.

4.2. Changing action ordering

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 time, no two actions can have the same timestamp. Therefore, we define

$$ExecTime : Act \rightarrow \mathbb{R}^+$$

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 $i, j < \mathcal{N}(\overline{t})$ 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, then $t'_{i+1} > t_{i+1}$[1] and

$$\begin{aligned} t'_{i+2} &= t'_{i+1} + ExecTime(a_{i+2}) \\ \ldots & \\ t'_j &= t'_{i+1} + \textstyle\sum_{k=i+2}^{j} ExecTime(a_k) \end{aligned}$$

Therefore, the distance between ξ_M and ξ_I is induced by the amount of execution time of all the actions found at the same timestamp in the model:

$$d(\xi_M, \xi_I) = sup\{|t_i - t'_i| \mid i < \mathcal{N}(\overline{t})\} = sup_{i,j<\mathcal{N}(\overline{t})}\{|t_j - t'_j| \mid t_i < t_{i+1} = \ldots = t_j < t_{j+1}\} =$$

$$= sup_{i,j<\mathcal{N}(\overline{t})}\{t'_{i+1} - t_{i+1} + \sum_{k=i+2}^{j} ExecTime(a_k) \mid t_{i-1} < t_i = \ldots = t_j < t_{j+1}\}$$

Correspondingly, the observable distance between the timed action sequences is induced by the implementation timestamp of the last observable action executed at the same model timestamp. Thus

$$d^*(\xi_M, \xi_I) = sup_{i,j<\mathcal{N}(\overline{t})}\{t'_{i+1} - t_{i+1} + \sum_{k=i+2}^{l} ExecTime(a_k) \mid$$
$$t_{i-1} < t_i = \ldots = t_j < t_{j+1}, L(a_{l+1}) = \ldots = L(a_j) = \tau\}$$

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 $i < l \leq j < \mathcal{N}(\overline{t})$ such that $L(a_l) \neq \tau$ and $L(a_{l+1}) = \ldots = L(a_j) = \tau$, hence

$$d^*(\xi_M, \xi_I) = t'_{i+1} - t_{i+1} + \sum_{k=i+2}^{l} ExecTime(a_k)$$

Moreover, assume there exists some $i < m < l$ such that $L(a_m) = \tau$ and $L(a_{m+1}), \ldots, L(a_l) \neq \tau$, thus the execution time of this τ action influences the observable distance between model and implementation. If this action is independent from the following observable actions, then it is allowed to impose urgency on the execution

[1] The exact relation between them depends on the time synchronisation mechanism. In the ideal case, $t'_{i+1} = t_{i+1} + ExecTime(a_{i+1})$. Nevertheless, the following action that has the same timestamp will start immediately at t'_{i+1}.

of the observable actions and postpone τ for after their execution. This translates into a permutation of the actions in ξ_M and respectively in ξ_I with one position to the left, thus obtaining

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

in the sequence of actions, whereas the absolute values of the timestamps remain the same.

The new timed action sequence in the model is denoted with $\xi_M^{'}$, whereas the one in the implementation with $\xi_I^{'}$. Based on the definition in the previous subsection, ξ_M and $\xi_M^{'}$ are observation equivalent, hence they have the same properties. Obviously, the distance between model and implementation remains the same ($d(\xi_M, \xi_I) = d(\xi_M^{'}, \xi_I^{'})$). However, the observable distance is reduced because in the implementation

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

thus, $d^*(\xi_M, \xi_I) > d^*(\xi_M^{'}, \xi_I^{'})$. This means that by postponing one unobservable action, the 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.

4.3. Synthesis strategy

By abstracting from unobservable actions, observation equivalence between systems is defined. Two systems are considered equivalent if an observer cannot distinguish them by interacting with them. The formalisation of observation equivalence is established by means of a relation on states of the timed labelled transition systems called *weak bisimulation relation* [12]. The idea behind weak bisimulation is to regard two states of two timed labelled transition systems as equivalent if they can engage in the same observable action or time transitions and end up in equivalent states again. As an example, fig. 3 shows a timed labelled transition system which is observation equivalent with the one presented in fig. 2.c for the $P||Q$ system. From the definition, it results that for each timed action sequence derived from one of the transition systems, there exists in the other transition system an observation equivalent timed action sequence[2]. Thus, two observation equivalent systems have the same observable properties.

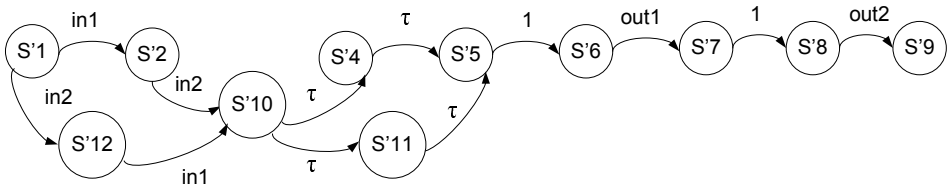

Figure 3. Observation equivalent timed labelled transition system for $P||Q$

Based on the interleaving semantics which is assumed in this paper for reasoning about concurrent real-time systems, urgency of observable actions execution over the unobservable actions is often possible. For example, for

[2]Paths through timed labelled transition systems travelling through equivalent states induce timed action sequences in which the ordering of observable actions is preserved, only the relative position of unobservable actions differs.

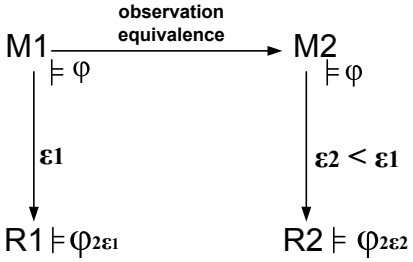

Figure 4. Synthesis strategy based on observation equivalence

two parallel actions $a||\tau$, two representations are possible: $a;\tau$ and $\tau;a$, and either can be found in a model $M1$ of the system. By imposing observable actions urgency, an observation equivalent model $M2$ may consider only $a;\tau$. Thus, viewing a model as a timed labelled transition system, an observation equivalent model can be built by considering only paths in which observable actions are taken before τ. As both $M1$ and $M2$ satisfy the same properties, as shown in fig. 4, 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.

5. A predictable development approach

To guarantee preservation of model properties in the implementation of a system, the ϵ-hypothesis must be complied with, which requires that: the timed action sequences of the implementation should be the same as those in the model; moreover, the time deviations between activations of the corresponding observable actions in the implementation and the model should be less than or equal to ϵ units of time.

In this section, we present a model synthesis approach that is based on a design language adequate for modelling concurrent real-time systems, called the Parallel Object-Oriented Specification Language (POOSL). POOSL consists of a *process* part and a *data* part. The process part is used to specify the behaviour of active components in the system, the processes, and it is based on a real-time extension of the Calculus of Communicating Systems (CCS) [12]. The data part is based on traditional concepts of sequential object-oriented programming. It is used to specify the information that is generated, exchanged, interpreted or modified by the active components. The semantics of POOSL is based on the two phase execution model, where the state of a system can change either by asynchronously executing atomic actions, such as communication and data computation without time passing (phase 1) or by letting time pass synchronously without any action being performed (phase 2).

The implementation of the two phase execution model in a simulation tool, SHESim, is achieved by adopting the process execution trees (PETs). 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. For example, fig. 5.a shows a system consisting of two parallel processes P and Q. The PETs of $P||Q$ are shown in fig. 5.b. During the evolution of the system, PETs send *action requests* and/or *delay requests* to the scheduler. The PET scheduler, whose behaviour is described by the algorithm in fig. 6, asynchronously grants all eligible atomic actions until there are no other actions available at the current model time moment. The internal state of each PET is dynamically changed according to the choices made by the PET scheduler and new requests may be sent to the scheduler. When no action is possible, based on the shortest delay request, time passes synchronously for all PETs until an action becomes eligible again. More details about PETs can be found in [16]. The correctness of PET with respect to the semantics of the POOSL language was formally proved in [6].

Based on the strengthening of observable property preservation results we have proved in section 4, we have extended the PET scheduler such that it can make a distinction between observable and unobservable actions.

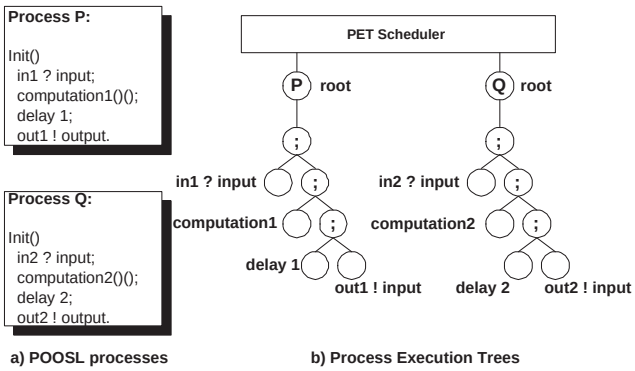

Figure 5. The $P||Q$ system in POOSL

```
PETSCHEDULER()
 list actions;
 list delays;
 while true do
  while actions.notEmpty() do
     actions.getAsynchronously()–>grant();
  if delays.nonEmpty() then
     modelTime = modelTime + delays.getFirst()–>amountOfTime();
     continue;
  else
     DEADLOCK();
     return.
```

Figure 6. The PET scheduler

Moreover, it can enforce urgency on the execution of observable actions over the unobservable ones. With a proper annotation of the model made at design time, observable and unobservable action requests that arrive at the scheduler are placed into different lists. Whenever observable actions are available for execution, they are granted by the PET scheduler. When no observable action is eligible, an unobservable action request is granted, and then the observable action requests list is checked again. The algorithm for the extended PET scheduler is presented in fig. 7. 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.

Rotalumis is a tool that takes a POOSL model and generates the executable code for the target platform. The ϵ-hypothesis is incorporated into Rotalumis. Each PET in the model is directly transferred into a C++ structure called PET 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, the implementation of a system needs to interact with the outside world and its behaviour has to be interpreted in physical time domain. Since the progress of model time is monotonic 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. That is, the scheduler of PETs ensures that the implementation always has the same event order as observed in the POOSL model.

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 physical time during the running of the implementation, as shown in fig. 8. This ensures that the execution of the implementation is always as close as possible to a trace in the model

```
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();
     continue;
  else
     DEADLOCK();
     return.
```

Figure 7. The extended PET scheduler

with respect to the observable distance between timed action sequences. In the variable $epsilon$, the observable distance between model and implementation is updated every time when no observable action is available. Due to the physical limitation of the platform, the scheduler may fail to guarantee that the implementation is ϵ-close to the model, for a fixed ϵ value. In this case, designers can get the information about the missed actions from the scheduler. Correspondingly, they can either change the model to reduce the computation cost at a certain model time moment, or replace the target platform with a platform of better performance.

6. Experimental results

To illustrate the improvement that can be obtained in the strength of property preservation of a system by using the approach proposed in this paper, we considered as case study the control of a motion system made of two devices running in parallel (see fig. 9). Such a system is representative, for example, for the control of a part of a printer where several motors must be controlled concurrently. In our setup, each device is made of two rotating masses connected through a flexible shaft. One mass is excited by a motor, while the position of the second mass can be measured. The movement of the motors, and accordingly of the first masses, follows a linear law $r = v * t$, where the position r is given as a function of the velocity v and the current time t. One motor runs at $50 rotations/s$, and the other one at $30 rotations/s$. The control algorithms that ensure stability of the system were designed by control engineers such that they run at $1KHz$ and at $500Hz$ respectively. The goal of this experiment was to correctly synthesise the control part of the system such that its stability is guaranteed.

For the interface between the computer and the motion system, we used a TUeDACS [3] data acquisition system. It operates in real-time, without buffering of data and provides a fast link to the computer. Moreover, the operating system running on the laptop is LinuxRT, the real-time version of Linux, to enable real-time behaviour of the software components.

An analysis model of the motion system is shown in fig. 10. The control part of the system is made of two parallel processes, $MotorController\_1$ and $MotorController\_2$, whereas the environment consists of $Motor\_1$ and $Motor\_2$. The code shown in fig. 10 is the POOSL model for each $MotorController$. The method $controlAlgorithm$ models the actual control algorithm for a motor. From the design of the system, $ControllerPeriod$, which is an instantiation parameter of each process, is $1ms$ for the first motor and $2ms$ for the second one. To ensure the stability of the control of the two motors system, two required real-time properties must be satisfied. For the first motor, the message $actuatorOutput$ must *always* be sent between 0.9 and $1.1ms$ after the message

```
PETSCHEDULER()
 list observableActions;
 list unobservableActions;
 list delays;
 float epsilon = 0;
 while true do
  while observableActions.notEmpty() do
    observableActions.getAsynchronously()–>grant();
  /* re-compute the observable distance */
  if an observableAction was granted &
          epsilon < physicalTime - modelTime then
    epsilon = physicalTime - modelTime;
  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 8. The PET scheduler in Rotalumis

$sensorInput$ is received. The MTL formula corresponding to this property is

$$\Box(p_1 \rightarrow \Diamond_{[0.9,1.1]} q_1)$$

where p_1 represents the receiving of $sensorInput$ message and q_1 the sending of $actuatorOutput$ message for motor 1. For the second motor, with similar notations, the property that needs to be satisfies is

$$\Box(p_2 \rightarrow \Diamond_{[1.9,2.1]} q_2)$$

Since the model is very 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 generation of the implementation of the control part of the system from the model, a synthesis model was developed. In this model, the environment part was removed and all the communication with the environment was replaced with a synthesisable interface. The model presented in fig. 11 shows how the communication with the environment model was replaced with calls to the $readSensor$ and $writeActuator$ methods from a data class called DAS - Data Acquisition System. This data class provides only virtual methods for the synthesis model. Its actual implementation for the communication with the real motors via the TUeDACS device is provided by Rotalumis in C++ code (fig. 12 shows as an example the $readSensor$ method). The implementation of the interface should not be blocking because its timing behaviour can affect the deviation between model and implementation.

First, we have synthesised the model of the motor controllers using the previous synthesis approach, without making a distinction between observable and unobservable actions. We have recorded the time deviations obtained in the implementation for several hours of continuous behaviour and the maximum deviation was $0.213ms$. This implies that the real-time properties are weakened up to

$$\Box(p_1 \rightarrow \Diamond_{[0.574,1.426]} q_1)$$
$$\Box(p_2 \rightarrow \Diamond_{[1.574,2.426]} q_2)$$

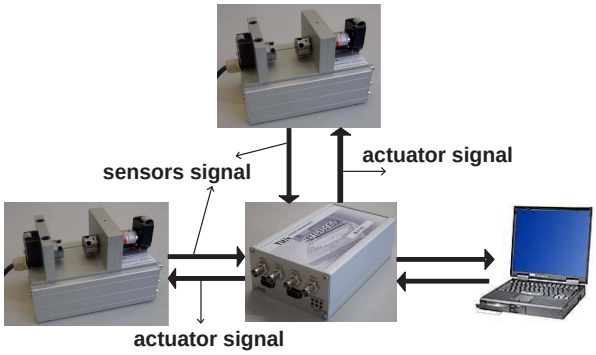

Figure 9. The setup of the case study

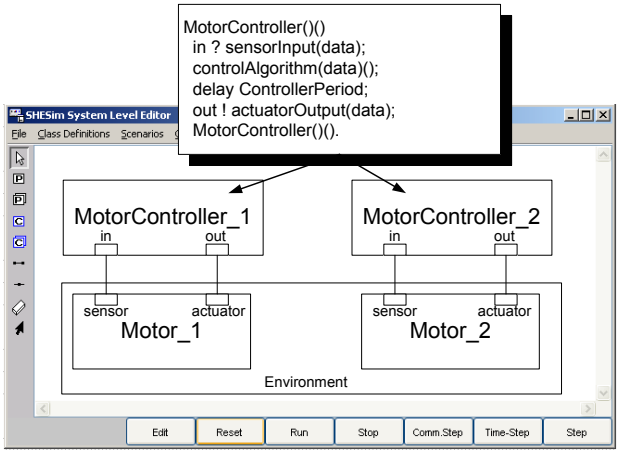

Figure 10. Model of the system

thus not meeting the requirements. This large weakening explains the unstable behaviour of the system that could be noticed during execution. A solution to obtain a smaller time deviation is to use a processor of a higher frequency.

In a second experiment, we have used the synthesis approach presented in this paper. We have labelled each action in the model, as shown in fig. 13, such that Rotalumis could identify which is considered observable or unobservable. Actions like reading from the sensor and writing to the actuator are observable activities of the system, whereas the control algorithm computation is 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

$$\Box(p_1 \rightarrow \Diamond_{[0.916,1.084]} q_1)$$
$$\Box(p_2 \rightarrow \Diamond_{[1.916,2.084]} q_2)$$

which fulfill the requirements. Hence, the control strategy designed is implementable using the proposed approach.

From this experiment, we could observe how significant the impact of the new synthesis approach on the strength of property preservation may be. In our case, the time deviation decreased with 80% because the computations turned out to be much more time-intensive than the reading and the writing actions.

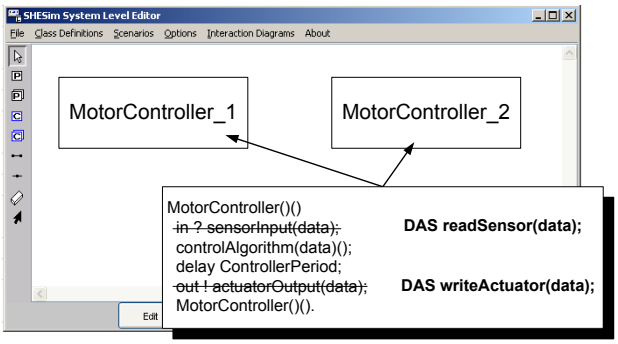

Figure 11. Synthesis model of the system

```
static PDO *PDM_readSensor (PDO **LV)
{
  PD_DAS *pd = (PD_DAS *)PRIMDATA(LV[0]);
  double y;
  /* read the sensor data via TUeDACS API */
  td_enc_read_chan(&y, pd–>channel, pd–>link, TD_DIRECT);
  return Real(y);
}
```

Figure 12. The implementation of $readSensor$

7. Related research

The timing semantics based on the two phase execution framework that treats system functionality and time progress in an orthogonal way, namely, system actions are timeless (without any time progress) and the time progress is actionless (without preforming any action), is commonly adopted in design languages. Variations of this semantics have been used in different formal frameworks to model and analyse real-time systems, such as timed automata [4], time process algebra [13] and real-time transition systems [7]. Recently, this semantics is also integrated into design languages, such as SDL-2000 supported by TAU G2 [1], or POOSL supported by SHESim [6], as we have seen in section 5. 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 [10].

As shown 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.

One of the formal approaches widely used for modelling and analysis of real-time systems is timed automata. TIMES [2] is a tool for design of real-time systems models based on timed automata that can describe concurrency and synchronisation of periodic, sporadic, preemptive or non-preemptive real-time tasks with or without precedence constraints. An automaton is schedulable if there exists a scheduling strategy such that all possible sequences of events accepted by the automaton are schedulable (all associated tasks can be computed within their required deadlines). From such a real-time system model, the TIMES compiler generates a scheduler, based on

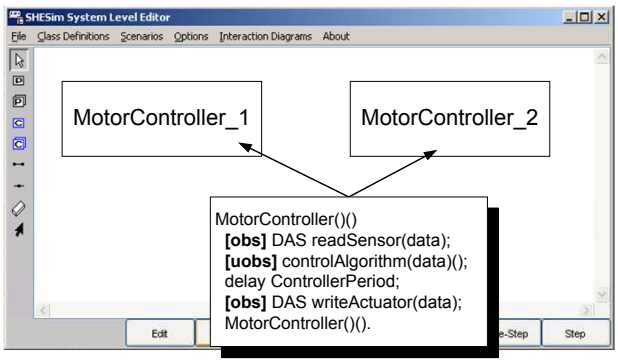

Figure 13. Annotated synthesis model

fixed priority assignment, and computes the worst case response time for all tasks. For this, it relies on synchrony hypothesis, assuming that the time for handling system functions on the target platform can be ignored compared with the execution times and deadlines of tasks which are considered pre-specified. This issue may be overcome by integrating the ϵ-hypothesis in their code generation. Moreover, the type of real-time properties TIMES focusses on refer to deadlines of tasks, which is at a coarser level of granularity than the properties we are are looking for preserving in the implementation.

A relaxation of the synchrony hypothesis is proposed in [17] 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. This idea is in-line with the ϵ-hypothesis that we use and it was shown that control strategies modelled with this semantics are robust and implementable. However, our approach differs from theirs in the sense that we look for an implementation that has the smallest time deviation from the model, whereas they set a requirement on the deviation and give a solution that satisfies it.

In [15], a programming model for real-time embedded controllers called Giotto is presented. Giotto is a methodology designed specifically for embedded control software development. Giotto is an embedded software model that can be used to specify a solution to a given control problem independent from a target platform. However, it is closer to executable code than a mathematical model. Giotto is restricted to periodic non-preemptive real-time control tasks. For model synthesis, worst case execution times of all tasks on the target CPU have to be provided, together with a jitter tolerance of the model. The Giotto compiler determines a schedule of the tasks that realises the execution on the target platform conforming to the tolerance. Moreover, Giotto can be seen as an intermediary step between mathematical models like timed automata and real execution code.

Compared with Giotto, our approach is fully based on a mathematical structure, the timed labelled transition system, and on a metric to express property preservation. During automatic code generation and execution, the time deviation (which is equivalent with the jitter tolerance) between the implementation and the model is determined for that specific target platform.

8. Conclusions

In this paper, we have shown how the strength of observable property preservation for concurrent real-time systems can be improved. We defined a notion of distance that abstracts away from the internal unobservable actions. This distance was used as a metric to express the strength of observable property preservation between model and implementation. We proved that an implementation in which observable action transitions can be taken before unobservable transitions 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. By the means of a motion control system case study, we showed 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.

Acknowledgments. The authors would like to thank Bjorn Bukkems for his support with the setup of the case study.

References

[1] Telelogic TAU generation 2. `http://www.telelogic.com/corp/products/tau/index.cfm`.

[2] TIMES Tool. `http://www.timestool.com/`.

[3] TUeDACS, TU/e Data Acquisition and Control System. `http://www.tuedacs.nl/`.

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

[5] Jos C. M. Baeten. The total order assumption. In: *NAPAW '92: Proceedings of the First North American Process Algebra Workshop*, pp. 231–240. Springer-Verlag, London, UK, 1993.

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

[7] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. An Interleaving Model for Real Time. In: *Proc. of the 5th Jerusalem Conference on Information Technology*, pp. 717–730, 1990.

[8] Jinfeng Huang, Jeroen Voeten, and Marc Geilen. Real-time property preservation in concurrent real-time systems. In: *Proc. of 10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, 2004.

[9] Jinfeng Huang, Jeroen P.M. Voeten, and Marc C.W. Geilen. Real-time property preservation in approximations of timed systems. In: *Proc. of 1st Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pp. 163–171, 2003.

[10] Jinfeng Huang, Jeroen P.M. Voeten, Andre Ventevogel, and Leo J. van Bokhoven. Platform-independent design for embedded real-time systems. In: *Proc. of Forum on Specification and Design Languages (FDL)*, pp. 318–329, 2003.

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

[12] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

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

[14] Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In: *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pp. 526–548. Springer-Verlag, London, UK, 1992.

[15] Benjamin Horowitz Thomas A. Henzinger and Christoph M. Kirsch. Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE*, 91: pp. 84–99, 2003.

[16] Leo J. van Bokhoven, Jeroen P.M. Voeten, and Marc C.W. Geilen. Software synthesis for system level design using process execution trees. In: *Proc. of 25th Euromicro Conference*, pp. 463–467, 1999.

[17] Martin De Wulf, Laurent Doyen, and Jean-Franois Raskin. Systematic implementation of real-time models. LNCS 3582, pp. 139–156. Springer, 2005.