A Clock-independent Model for Real-Time

F. Rößler, B. Geppert, R. Gotzhein

SFB 501 Report 18/96

A Clock-independent Model for Real-Time

F. Rößler, B. Geppert, R. Gotzhein
{roessler, geppert, gotzhein}@informatik.uni-kl.de

Report 18/96

Sonderforschungsbereich 501

Computer Networks Group
Computer Science Department
University of Kaiserslautern
P.O. Box 3049
67653 Kaiserslautern
Germany

A Clock-independent Model for Real-Time

F. Rößler, B. Geppert, R. Gotzhein

Computer Science Department, University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
{roessler, geppert, gotzhein}@informatik.uni-kl.de

Abstract

A new approach for modelling time that does not rely on the concept of a clock is proposed. In order to establish a notion of time, system behaviour is represented as a joint progression of multiple threads of control, which satisfies a certain set of axioms. We show that the clock-independent time model is related to the well-known concept of a global clock and argue that both approaches establish the same notion of time.

1 Introduction

Though there is no accepted definition of the term *real-time*, there are at least four characteristics of real-time *systems* commonly agreed on:

- Timeliness: Real-time systems have to perform their functions on time, i.e. have to be predictable in timing behaviour.
- Reactiveness: Real-time systems normally exhibit ongoing interaction with some environment, often occurring in an asynchronous and unpredictable manner.
- Concurrency: Real-time systems often require concurrent processing to keep pace with the simultaneous activities of the embedding environment.
- Distribution: Real-time systems, as in the case of communication systems, are often inherently distributed.

Throughout this report we exclusively focus on the timeliness aspect when talking about real-time. Existing Formal Description Techniques (FDTs) such as Estelle [13] or SDL [19] provide powerful modelling concepts to support distribution, concurrency, and reactiveness, the timeliness issue, however, is often neglected. Ideally one would wish for a possibility to track time utilization during system specification and design. Beginning with the global timing requirements, the impact of design decisions on timing behaviour should be understandable.

The broad variety of present techniques to cope with timeliness ranges from queueing [15] and scheduling theory [14] to simulative techniques. The problem with these approaches lies in the construction of a separate mathematical or computer-executable model. However, it should be possible to argue within the same formalism also used for system specification. [4], [11], and [10] are examples of simulative and stochastic approaches already dealing with that problem. The advantage is obvious, because then we have a better chance to be sure that no assumptions about the considered system are ignored or smuggled into the argumentation. We take that risk whenever an extra modelling of timing behaviour is necessary.

Though FDTs are sometimes extended by the means of tools and methodologies to alleviate the timeliness problem (e.g., [3], [21], and [20]) this is not achieved in a strictly formal but a pragmatical way.

As a prerequisite, a sufficient time semantics is needed for any formal treatment of real-time systems. Therefore several time extensions of existing models of computation (such as finite automata) were proposed in the literature. In [2], a real-time system is modelled as a tuple S = (S, P, μ , T) with a set S of system states, a set P of observables, a labelling function μ from S to 2^P, and a set T of timed state sequences. Each possible behaviour of the system is represented by a timed state sequence $\tau \in$ T, which associates system states with time instants. Timed automata [1] generalize finite state machines over infinite strings and are additionally restricted by timing constraints. Actually a timed automaton operates with a finite set of real valued clocks that proceed jointly as time elapses. A state transition may reset some of the clocks. The point is that each state puts certain constraints on the clock values and the automaton is only allowed to reside in a particular state as long as these constraints are met. As a result, valid state transition sequences are associated with time interval sequences, which determine the allowed duration of stay for individual states. Another definition of a timed automaton is given in [17]. There a timed automaton generalizes the formalism of an I/O automaton [18] by restricting the times at which state transitions may occur with lower-bound and upper-bound requirements. Informally speaking, the requirements assert that a state transition cannot be taken unless it has been continuously enabled for a number of time units equal to the lower bound and must have been taken before the upper bound expires. Consequently, state transition sequences are associated with sequences of time instants, determining when transitions are taken. In [12] the timed communicating state machine model equally applies these lower-bound and upper-bound requirements. Other approaches, based, for instance, on petri nets, process algebras, or temporal logic, are also discussed in the literature [5].

There is one point all these approaches have in common: they describe the system state as a function of time and therefore employ the concept of a clock. We call this kind of time modelling *clock-dependent.* This report investigates the question whether clock-dependence is the only possibility for time modelling, and actually proposes a time model, that does not rely on a clock.

This report is organized as follows. After a general motivation, Section 2 introduces a clock-independent model of time in three steps. As a starting point, a computational model is presented, which reflects the minimum set of requirements necessary to establish a notion of time. Afterwards we introduce *implicit coupling* as a general means for modelling time. Finally, we define a set of axioms, which specialize implicit coupling to a sound time model. Section 3 relates the clock-independent time model to the well-know concept of a global clock, and we argue that both approaches establish the same notion of time.

2 Clock-independent modelling of time

The timing concepts referenced in Section 1 describe the system state as a function of time. In case of concurrent systems with multiple threads of control, it is generally possible to eliminate the time variable as the following analogy from mechanics illustrates. Imagine two physical processes, a pen-

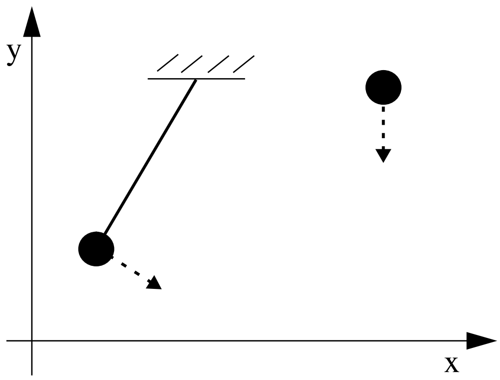


Fig. 1: two physical processes

dulum and the free fall of a mass (Figure 1). From physics, we may derive formulas to describe their state variables (x-, y-coordinates and velocity) as functions of time. This is analogous to the timing concepts referenced in Section 1. Alternatively, we may express the state (set of state variables) of the pendulum as a function of the state of the falling mass and vice versa, if we only know the initial or another arbitrary pair of coinciding states of the two physical processes. Though this transformation eliminates the time variable, the system description (the set of functions mentioned above) itself is certainly not timeless. The description only makes no use of an auxiliary measuring process, for instance, an oscillating crystal, which plays the role of a clock. While the clock-dependent description relates the progression of an imaginary clock with the progression of each physical process (e.g., after t_1 ticks of the clock, the falling mass will reach point (x_1, y_1)), the clock-independent description relates the progression of the two physical processes directly (e.g., when the pendulum reaches (x_1, y_1) the first time, this will coincide with the falling mass located at (x_2, y_2)).

In the following, we apply this analogy to yield a new semantic model for the specification of concurrent real-time systems.

2.1 Model of computation and implicit coupling

This report does not propose a time extension of an existing model of computation (e.g., communicating automata or petri nets). Rather, we investigate clock-independent modelling of time in general. Therefore we employ a very simple model of computation, which reflects the minimum set of requirements necessary to establish a notion of time. Note, that this model is certainly not suited or intended for *functional* aspects of concurrent systems.

Essentially the computational model provides an abstraction (called *execution* in this report) for the multiple threads of control executed by the processes of a concurrent system. Strictly speaking, executions only represent sections between possible waiting points of a thread of control, where the thread of control may be delayed because of causal dependencies on external events such as message arrival. Formally, an *execution* is defined as a totally ordered and dense set (continuum) of *states*. Apart from being uniquely named, states are not further characterized. Note, that in this context executions are threads of control with not only *discrete* observable states, as normally considered. Though irrelevant for the specification of the functional behaviour of computing systems[1], we need this concept for the foundation of asynchronous processing and dense time. It is worth mentioning, that the total order of an execution models the causal dependency relation between states.

A concurrent system comprises multiple processes, where each possible run of a process yields an element of a set of possible threads of control. Therefore we model a concurrent system as a set of executions, representing those sections of the possible threads of control, which are not delayed.

Definition 2.1 *A concurrent system C is given by a set of executions E_i, where an execution E_i is defined by a totally ordered and dense set of states with a start state $s_{init} = min\{E_i\}$. Additionally states are unique, i.e. the following holds: $\forall\ i, j, t, u.\ i \neq j \wedge t \in E_i \wedge u \in E_j \Rightarrow t \neq u$.*

For later usage we introduce some additional notations. We assume a function *e*, which yields the corresponding execution for any state, i.e. $e(t) = E_i$ iff $t \in E_i$ (as states are unique *e* is well-defined). Additionally $<_{e(s)}$ denotes the total order of the execution $e(s) \in C$.

As already mentioned the definition of a concurrent system (our model of computation) is quite simple, because we stripped off all the details of a real system, that are not necessary to establish a notion of time. The concurrent mechanical system of the pendulum and the falling mass, for instance, may be modelled by two executions representing the two trajectories through the state space of x- and y-coordinates. According to our analogy, a notion of time can be established if we introduce functions from the states of one execution to the states of the other, and vice versa. We call

1. adequate observable parts of the universe abstract from physical continuity

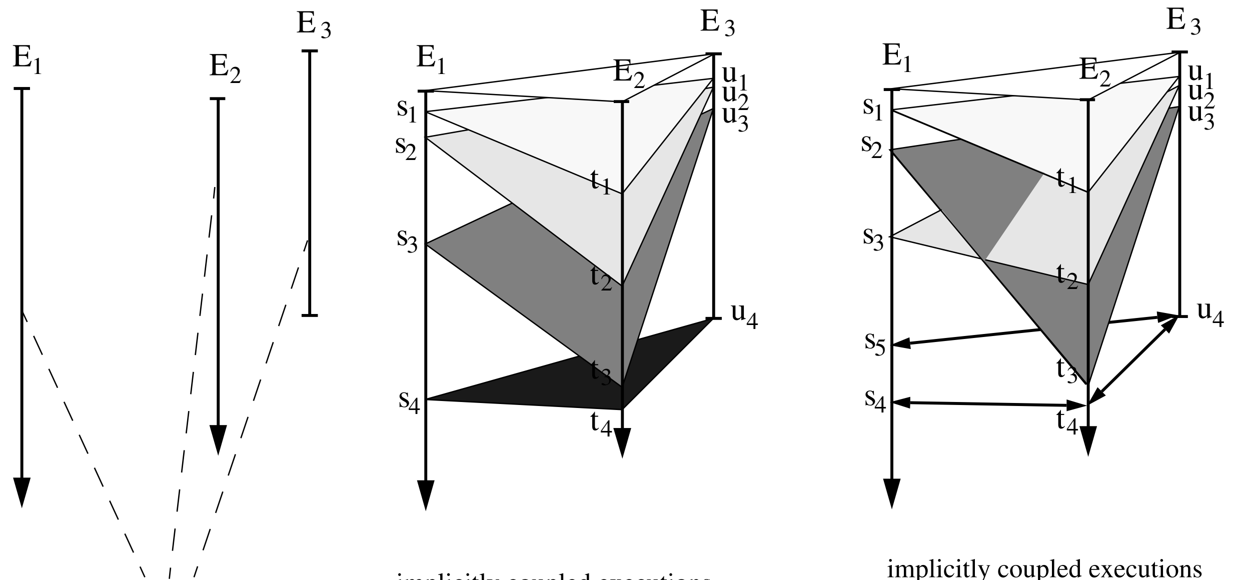
two executions with those functions defined *implicitly coupled* and generalize implicit coupling to arbitrary concurrent systems (as defined in Definition 2.1).

Definition 2.2 *An implicitly coupled concurrent system is given by a pair (C, M), where C denotes a concurrent system according to Definition 2.1, and M denotes a set of coupling functions* $m_{s,\,t}: e(s) \rightarrow e(t),\ \exists\, E_i,\, E_j \in C\,.\ s \in E_i \wedge t \in E_j \wedge i \neq j$.

The reader may wonder whether the functions $m_{s,\,t}$ should better be indexed by executions instead of states. We employ the latter version, because we need to identify more than one function for each pair $(E_i\,,\, E_j)$ of executions in C. As Definition 2.2 even allows an arbitrary set of functions for each pair of executions, implicit coupling of a concurrent system is not sufficient for establishing a sound notion of time. In addition, the set M of functions $m_{s,\,t}$ has to be constrained (Section 2.2).

2.2 Axiomatic description of a real-time system

Roughly speaking, implicit coupling must enforce a *joint progression* of a set of executions as illustrated in Figure 2, where executions are vertical lines with the minimum state located at the top.



concurrent executions

implicitly coupled executions which establish a notion of time

implicitly coupled executions which fail to establish a notion of time

Fig. 2: joint progression of three executions

Nonterminating executions (i.e. executions where no maximum state exists) are indicated by an arrowhead. Therefore Figure 2 shows one terminating and two nonterminating executions on the left side. In the middle the same executions are shown together with some selected argument-value pairs of certain coupling functions. These coupling functions make the three executions to *jointly progress* in the following way:

- The executions start together.
- Eventually E_1, E_2, and E_3 will jointly reach the states s_1, t_1, and u_1, respectively.
- Afterwards E_1, E_2, and E_3 will jointly reach s_2, t_2, and u_2 , and so on.
- Finally E_3 will finish at u_4 , just when E_1 and E_2 coincide with s_4 and t_4 .
- It is not shown how E_1 and E_2 will continue.

This kind of joint progression is suitable to establish a notion of time, while the behaviour shown on the right side of Figure 2 is counterintuitive: on the one hand does the coincidence of the states s_2, t_3, and u_3 in conjunction with the coincidence of the states s_3, t_2, and u_2 violate causal dependency. On the other hand does the coincidence of the states s_4, t_4, u_4, and s_5 imply that E_1 proceeds without time consumption from s_5 to s_4. In the following, we present a necessary and sufficient set of axioms constraining the possible sets of coupling functions of an implicitly coupled concurrent system, thus leading to a sound notion of time. Note, that we use the following conventions: any variables that appear with no quantifiers are assumed to be universal variables with global scope, and undefined predicates evaluate to false.

Definition 2.3 *A real-time system T is an implicitly coupled concurrent system (C, M), where M meets the following axioms:*

1.1: $m_{s,\,t} \in M \Rightarrow \exists\, u.\ (m_{s,\,t}(u) = min\{e(t)\} \vee m_{s,\,t}(min\{e(s)\}) = u)$

1.2: $m_{s,\,t} \in M \wedge (max\{e(s)\}\downarrow$ [1] $\vee\ max\{e(t)\}\downarrow) \Rightarrow \exists\, u.\ (m_{s,\,t}(u) = max\{e(t)\} \vee m_{s,\,t}(max\{e(s)\}) = u)$

1.3: $u <_{e(s)} v <_{e(s)} w \wedge m_{s,\,t}(u)\downarrow$ [2] $\wedge\ m_{s,\,t}(w)\downarrow \Rightarrow m_{s,\,t}(v)\downarrow$

1.4: $\neg\, max\{e(s)\}\downarrow \wedge \neg\, max\{e(t)\}\downarrow \wedge \exists\, u.\ (m_{s,\,t}(u)\downarrow \wedge u <_{e(s)} v) \Rightarrow m_{s,\,t}(v)\downarrow$

2.1: $m_{s,\,t} \in M \Rightarrow m_{s,\,t}(s) = t$

2.2: $e(s) \neq e(t) \Rightarrow m_{s,\,t} \in M$

2.3: $m_{s,\,t}(u) = v \Rightarrow m_{s,\,t} = m_{u,\,v}$

3: $u <_{e(s)} v \wedge m_{s,\,t}(u)\downarrow \wedge m_{s,\,t}(v)\downarrow \Rightarrow m_{s,\,t}(u) <_{e(t)} m_{s,\,t}(v)$

4: $m_{s,\,t}(u) = v \Rightarrow m_{t,\,s}(v) = u$

5: $m_{s,\,t}(u) = v \wedge m_{t,\,r}(v) = w \Rightarrow m_{s,\,r}(u) = w$

Note, that coupling functions between executions neither need to be total nor surjective, as Figure 3 illustrates. But Axioms 1.1 - 1.4 assert certain other properties concerning the domain and the range

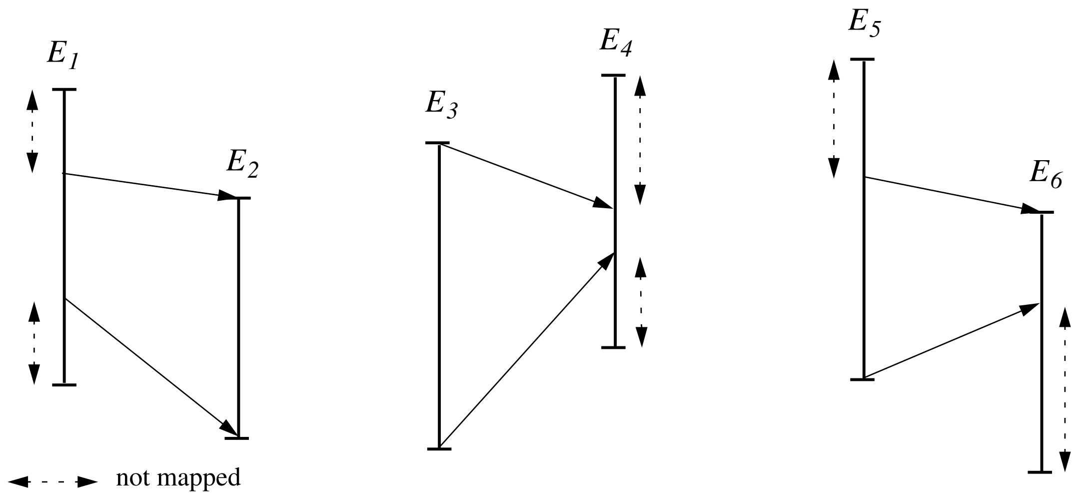


Fig. 3: legal coupling functions

of coupling functions. In particular, at least one of the two start states has to be mapped (Axiom 1.1).

1. Means: for *e(s)* a maximum state is defined. In other words, the execution *e(s)* terminates.
2. Means: $m_{s,t}\,(u)$ is defined.

The same applies to the final states, if existent (Axiom 1.2). Because of Axiom 1.3, domains have to be dense, where Axiom 1.4 asserts that domains are unbounded in the case of nonterminating executions. Together with Axioms 3 and 4, domains and ranges have to be left-closed intervals in the case of nonterminating executions and closed intervals otherwise. The end-points of an interval are given by the argument-value pairs containing the mapped start and final states (see Figure 3 for the case of terminating executions).

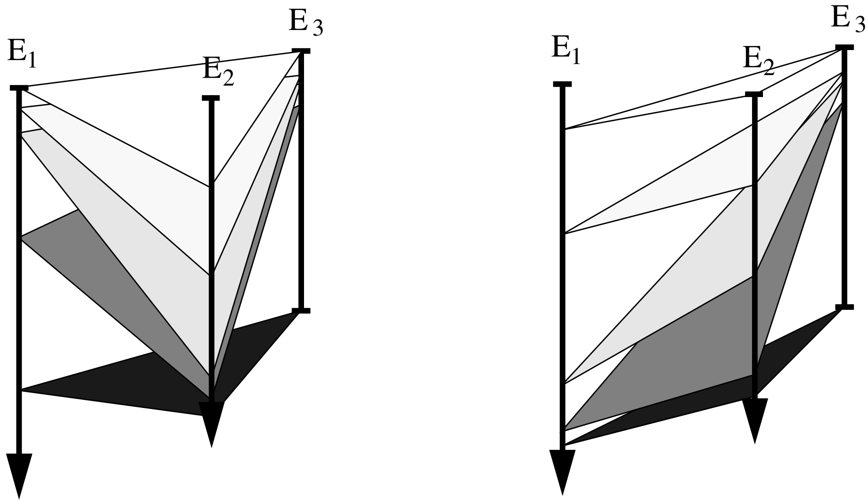


Fig. 4: different phase displacements between the executions of Figure 2

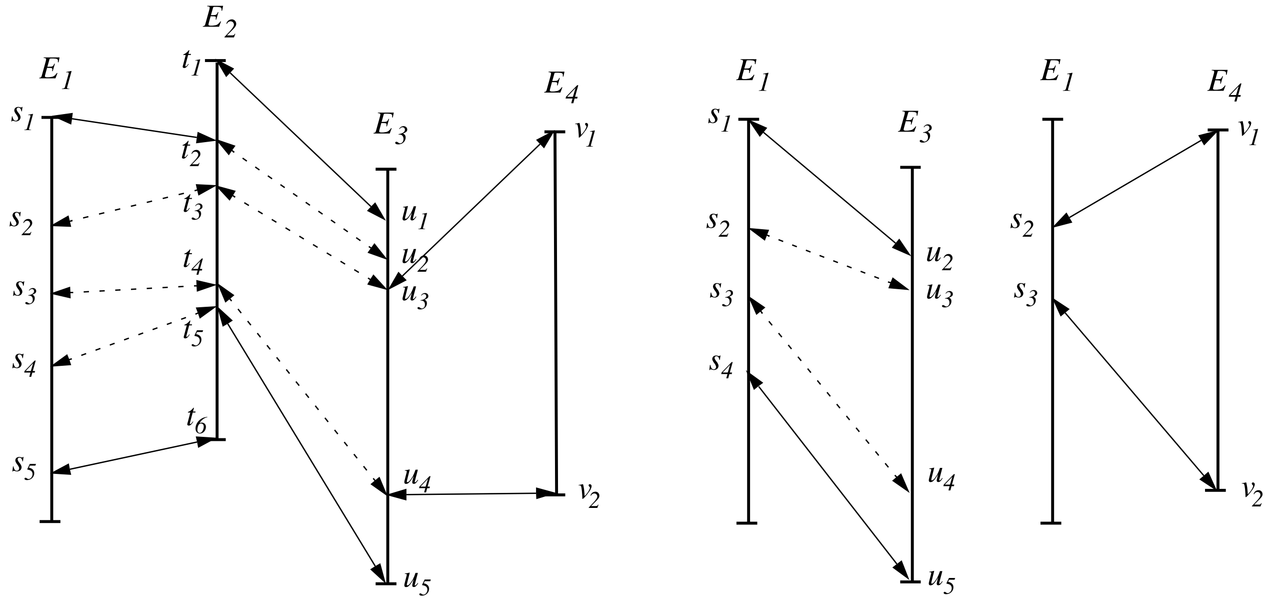
Axioms 2.1 - 2.2 allow an infinite number of different coupling functions between any pair of executions E_i and E_j , where each pair of states must be mapped by at least one of these coupling functions. Each pair of states will even be mapped by at most one coupling function, as Axiom 2.3 identifies two functions, if they share at least one mapped pair of states. If the start states do not coincide (i.e. they are not mapped) for a certain coupling function, we call the corresponding executions to be *out of phase* for that coupling function. As a consequence, each coupling function between the same pair of executions represents another phase displacement in the joint progression of the two executions. Different phase displacements for the three executions of Figure 2 are shown in Figure 4.

It is important that implicit coupling preserves causal dependency, which is guaranteed by Axiom 3. Additionally this axiom implies injective functions such that the inverse function exists, where Axiom 4 even states that the inverse coupling function must be contained in *M*. That is, functions between the same executions (no matter, which execution is domain or range) are equal (modulo inversion), if they share at least one coincident pair of states. Therefore coincident pairs may be illustrated by bidirectional arrows in Figure 3.

This kind of equivalence is generalized for the case of more than two executions by the Transitivity-axiom 5. Figure 5 gives an example. Part (a) shows a selection of mapped states for a sample set of executions. The domain and range end-points are mapped by solid lines. The dashed lines are additional coincident pairs, consistent with the above mentioned axioms. The application of the transitivity-axiom demands the existence of other mapped states partly shown in Figure 5(b). As a result we have the following joint progression of the executions E_1 - E_4:

- E_3 starts first.
- When E_3 reaches u_1, E_2 begins processing with t_1.
- E_2 and E_3 jointly reach t_2 and u_2 respectively, when E_1 starts work with s_1.
- Similarly E_4 is set in motion at state v_1, just when E_1, E_2 and E_3 coincide with s_2, t_3, and u_3, respectively, and so on.

The point is, that given one coincident tuple of states (e.g. (s_2, t_3, u_3, v_1)) together with the validity of Axiom 5, M enforces a unique joint progression of the executions E_1 - E_4.



(a) *(b)*

Fig. 5: illustration of Axiom 5

As mentioned above, for every pair of executions M contains several functions. However, those functions only differ in phase displacement, while the joint progression of the executions was carefully constrained to establish a sound notion of time. The joint progression is dependent, for instance, on the chosen implementation or hardware platform. Any change of these conditions results in a new set of coupling functions.

With a given set of coupling functions any possible behaviour of the real-time system is given by a joint progression of the executions with a fixed phase displacement.

3 Relation to the concept of a global clock

In the following, we construct a global clock from a real-time system (as defined in Definition 2.3). Therefore we introduce an operator $seq(E_1, E_2)$, which yields a new execution by connecting E_1 and E_2 in series. The point is, that iterative application of this operator yields a cyclic execution, that can be used for „time measuring“.

Definition 3.1 *Let E_1 and E_2 denote executions, where E_1 has a final state. The sequence operator $seq(E_1, E_2)$ yields a new execution E_3, where:*

- $E_3 = E_1' / \{ max\{E_1'\} \} \cup E_2'$ [1]
- $\forall s', t' \in E_3 \,.\, s' <_{E3} t' \Leftrightarrow (s \in E_1 / \{ max\{E_1\} \} \wedge t \in E_2) \vee s <_{E1} t \vee s <_{E2} t$

According to Definition 3.1 the states of E_3 are given by joining the states of E_1, E_2 , and identifying $max\{E_1\}$ with $min\{E_2\}$. Furthermore the total order $<_{E3}$ preserves the orders $<_{E1}$, $<_{E2}$ and $s' <_{E3} t'$

1. In order to preserve uniqueness of states, naming conflicts must be resolved. Throughout this report we will use primed identifiers for this purpose. Therefore E_i' denotes the set of primed states in E_i .

holds for all states s', t' corresponding to $E_1 / \{ max\{E_1\} \}$ and E_2, respectively. In our computational model a state is just a conceptual entity, and each pair of executions may be sequenced. When applying the model, the states are more specifically characterized, and only executions E_i and E_j , where the final state of E_i and the start state of E_j are „semantically identical", are suited to be sequenced.

With Definition 3.1 we may extend a given real-time system $T= (C, M)$ by adding sequences of executions. Thereby the coupling functions concerning a sequenced execution $E_3 = seq(E_1, E_2)$ are already determined by the coupling functions between E_1 and E_2, respectively and the other executions in C.

Lemma 3.2 *Given a real-time system* $T = (C, M)$ *and an execution* $E_3 = seq(E_1, E_2)$, *where* $E_1, E_2 \in C$, *it follows that* $T' = (C \cup \{E_3\}, M \cup Y)$ *is a real-time system iff the following holds:*

$y \in Y$ *iff*

- *y implicitly couples* E_3 *with some execution in C*
  and
- *if the mapped interval of* E_3 *consists of states all corresponding to either* E_1 *or* E_2, *y is identical (modulo renaming) with a coupling function* $m_{r,s}$ *of the execution, which is exclusively contained in the mapped interval of* E_3
  and
- *if the mapped interval of* E_3 *consists of states corresponding to both* E_1 *and* E_2, *y is the union (modulo renaming) of a coupling function* $m_{t,u}$ *of* E_1 *and a coupling function* $m_{v,w}$ *of* E_2 , *where* $m_{t,u}(max\{E1\})$ *equals* $m_{v,w}(min\{E2\})$.

Figure 6 gives an example for a derived coupling function between an execution E_i and a sequenced execution E_3 = seq(E_1, E_2). We illustrate the two cases, where the mapped interval of E_3 only consists of states corresponding to E_1 (Figure 6(a)) and where the mapped interval of E_3 contains states corresponding to both E_1 and E_2 (Figure 6(b)).

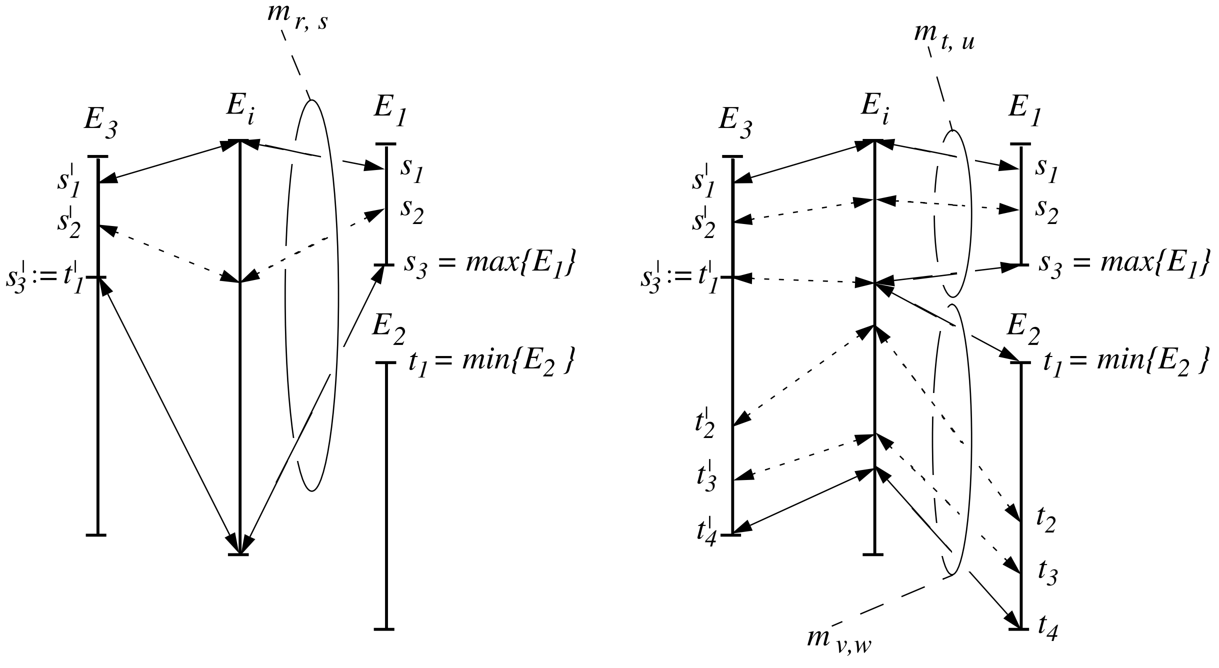
With Lemma 3.2 we can incorporate a global clock into a given real-time system $T = (C, M)$. As a starting point, we select an arbitrary execution in C as a reference which we call *tick*. Furthermore, we inductively define an execution $clock_n$ for every natural number n, which we add to T (according to Lemma 3.2 we yield a new real-time system).

Definition 3.3 *Given an execution tick of a real-time system,* $clock_n$ $(n \in N)$ *is defined as follows:*

$$clock_n = \begin{cases} tick, & n = 1 \\ seq(tick, clock_{n-1}), & n > 1 \end{cases}$$

Actually, an execution $clock_n$ of the real-time system plays the role of a global clock with a lifetime of n ticks. For „time measuring" we determine a certain *time function* $time_{Ei}(n)\colon N \rightarrow E_i$ for each execution $E_i \in C$. A time function $time_{Ei}(n)$ yields the reached state of E_i „after *n ticks*" of some $clock_m$ ($m \geq n$), assumed E_i and $clock_m$ start together. Figure 7(a) shows the details for determining $time_{E1}(1)$ - $time_{E1}(6)$, $time_{E2}(1)$ - $time_{E2}(5)$, *and* $time_{E3}(1)$ - $time_{E3}(4)$ for the concurrent system of Figure 2. In order to determine $time_{E1}(i)$ and $time_{E2}(i)$ for $i > 6$ some $clock_m$ with $m > 6$ is needed.

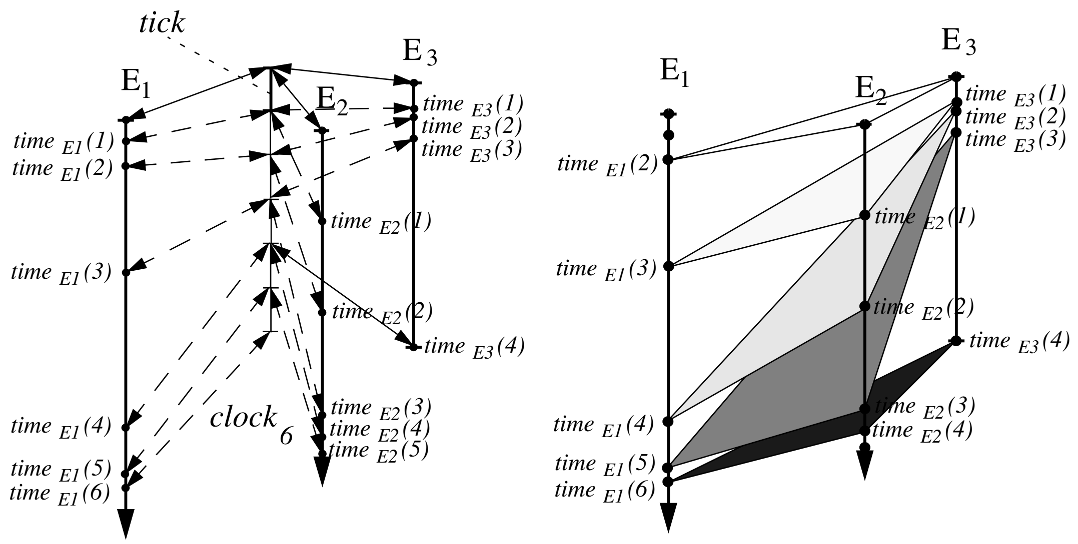
The point is, that the time functions $time_{Ei}$ even are a *clock-dependent* substitute for the *clock-independent* coupling functions M of a real-time system (at least to some extent). That is, if we are only interested in argument-value pairs and phase displacements lying apart an integer multiple of a *tick*, we have the full information at hand. For the case of the concurrent system of Figure 2 coupling functions can be derived from time functions as illustrated in Figure 7(b). It follows, that coupling functions must only map pairs of states that lie an equal number of *ticks* apart. For instance, in



(a) (b)

Fig. 6: implicit coupling between E_i and a sequenced execution $E_3 = seq(E_1, E_2)$

Figure 7 $time_{E1}(3)$ and $time_{E2}(1)$ as well as $time_{E1}(6)$ and $time_{E2}(4)$ are mapped, because the number of *ticks* between $time_{E1}(3)$ and $time_{E1}(6)$ equals the number of *ticks* between $time_{E2}(1)$ and $time_{E2}(4)$. For a valid coupling function this must hold for each pair of argument-value pairs.



(a) (b)

Fig. 7: incorporation of a discrete clock for the executions of Figure 2

As a result we conclude that implicit coupling and the concept of a global clock establish the same notion of time. Note, that we only constructed a discrete clock. However, one can extend this concept to a continuous clock, where a refinement of a *tick* is achieved by „iterative bisection“. As a result we get analogous functions to $time_{Ei}$, however, with the real numbers as domain. Additionally the continuous time functions build an equivalent substitution for the coupling functions *M* of a real-time system.

4 Summary

We proposed implicit coupling as a general means to establish a notion of time. Thereby a system is viewed as a set of executions with the ability to jointly progress, following certain laws. These laws are represented by a set of coupling functions between each pair of executions. It is a crucial point, that those coupling functions only differ in phase displacement and keep consistent otherwise. Any change in mutual progression speed, possibly caused by varying hardware platforms or faster algorithms, has to be modelled with a different set of mappings. We have shown that the clock-independent time model is related to the well-known concept of a global clock, and argued that both approaches establish the same notion of time.

The suggested time model is intended to be applied to the development of real-time communication systems with formal methods. Though there are enormous research activities in the field of Quality of Service (QoS) architectures, QoS provision, QoS control, and QoS management (e.g., [6], [9], [8], [7], [16]) description techniques lack an adequate support to employ these concepts. The proposed time model has to demonstrate its value when integrating these concepts.

References

[1] R. Alur and D. Dill, **Automata for Modeling Real-Time Systems**, LNCS 443, Springer, 1990

[2] R. Alur and T.A. Henzinger, **Logics and Models of Real Time: A Survey**, LNCS 600, Springer, 1992

[3] R. Bræk and Ø. Haugen, **Engineering Real Time Systems**, Prentice Hall, 1993

[4] G. v. Bochmann and J. Vaucher, **Adding Performance Aspects to Specification Languages**, International Conference on Protocol Specification, Testing, and Verification, Atlantic City, 1988

[5] G. Bucci, M. Campanai, and P. Nesi, **Tools for Specifying Real-Time Systems**, Real-Time Systems, 8: 117--172, 1995

[6] A. Campbell, C. Aurrecoechea, and L. Hauw, **A Review of QoS Architectures**, Proceedings of the 4th International IFIP Workshop on Quality of Service, Paris, 1996

[7] David D. Clark et al., **Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism**, Proceedings of ACM SIGCOMM'92, Baltimore, Maryland, pp 14--26, August 1992

[8] Luca Delgrossi, Ralf Guido Herrtwich, Carsten Vogt, and Lars C. Wolf, **Reservation Protocols for Internetworks: A Comparison of ST-II and RSVP (Extended Abstract)**, 4th International Workshop on Network and Operating System Support for Digital Audio and Video, United Kingdom, 1993

[9] Domenico Ferrari, **Client Requirements for Real-Time Communication Services**, IEEE Communications Magazine, 28(11): 65--72, 1990

[10] N. Goetz, H. Hermanns, U. Herzog, V. Mertsiotakis, and M. Rettelbach, **Stochastic Process Algebras - Constructive Specification Techniques Integrating Functional, Performance**

and Dependability Aspects, Chapter 1 of: Baccelli and Mitrani (ed.), Quantitative Modelling in Parallel Systems, Springer, 1995

[11] M. Hendaz, S. Budkowski, **A New Approach for Protocols Performance Evaluation Using Annotated Estelle Specifications**, Proceedings of the 8th International Conference on Formal Description Techniques, Canada, 1995

[12] C.-M. Huang and S.-W. Lee, **Timed Protocol Verification for Estelle-Specified Protocols**, Computer Communication Review, 25(3), 1995

[13] ISO, **Estelle: A Formal Description Technique Based on an Extended State Transition Model**, International Standard ISO/IS 9074, 1989

[14] M.H. Klein, T. Ralya, B. Pollak, R. Obenza, M. G. Harbour, **A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems**, Kluwer Academic Publishers, 1993

[15] L. Kleinrock, **Queueing Systems; Volume 2: Computer Applications**, Wiley, 1976

[16] E.W. Knightly and P. Rossaro, **Improving QoS through Traffic Smoothing**, Proceedings of the 4th International IFIP Workshop on Quality of Service, Paris, 1996

[17] V. Luchangco, E. Soeylemez, S. Garland, and N. Lynch, **Verifying Timing Properties of Concurrent Algorithms**, Proceedings of the 7th International Conference on Formal Description Techniques, Berne, 1994

[18] N. Lynch and M. Tuttle, **An introduction to input/output automata**, CWI-Quarterly, 2(3), 1989

[19] A. Olsen, O. Færgemand, B. Møller-Pedersen, R. Reed, and J.R.W. Smith, **Systems Engineering Using SDL-92**, North-Holland, 1994

[20] B. Selic, G. Gullekson, and P.T. Ward, **Real-Time Object-Oriented Modeling**, Wiley, 1994

[21] **SDT 3.0 Reference Manual & User's Guide**, TeleLogic, 1995