ON SPECIFYING REAL-TIME DISCRETE EVENT SYSTEMS : AN APPLICATION FOR DESIGNING REAL-TIME PROTOCOLS

A. Khoumsi , G.v. Bochmann and R. Dssouli

Université de Montréal
Faculté des arts et des sciences
Département d'informatique et
de recherche opérationnelle
C.P. 6128, Succursale A
Montréal, (Quebec)
H3C 3J7

Abstract. In this paper, we firstly propose a detailed model for specifying real-time discrete event systems. This model uses a global clock, and several fictitious timers and counters. It can be used for applications in different areas, such as telecommunications or process control, which can be modeled as concurrent and real-time discrete event systems (DES). Next, we propose an application of this model for deriving protocol and medium specifications from service specifications for real-time applications. Compared to [KBD93], the application field is much broader, because two important restrictions are removed. Firstly, temporal requirements are between events which are not necessarily consecutive. Secondly, the systems considered can be concurrent .

1. Introduction

A discrete event system (DES) is a dynamic system where events are executed instantaneously, causing a discrete change of the state of the system. If sequences of events are a regular language, the system can be specified by a finite automaton. A first example of DES is a telecommunication network; an event can then be the transmission of a packet of data. Another example is a communication protocol, and an event can be execution of a service primitive. For some DES it is not enough to represent the ordering of events. We must also specify temporal requirements between events. This class of DES are called Real-time DES. For specifying such DES, we use timed automata (TA) which are defined by using a global clock, and several fictitious timers and counters. And for studying them, we use the approach which consists of transforming a real-time problem to an untimed problem ([AD90,BW92]). In comparison with [KBD93], two extensions are made: **a)** temporal requirements are not only between consecutive events; **b)** concurrent systems are considered. Henceforth, DES means Real-time discrete event system.

This paper is organized as follows. In section 2, we introduce in detail the model we have developed for specifying and studying a real-time DES. In section 3, we propose an application of our model for designing real-time protocols in a systematic way. Both sequential and parallel protocols are considered. And at last, we conclude in section 4. We will notice that the possible concurrency in the parallel systems, and the timing requirements cause a problem of state space explosion and of complexity.

Here is a table of some of the most important notations used in this paper. They will be explained in the following sections.

N is the set of positif integers, and N^* is the set of strictly positive integers, i.e., $N^*=N-\{0\}$

DES means : Real-time discrete event system

|E| is the cardinal of a set E, and 2^E is the set of subsets of E

uct is an abbreviation for : unit of clock time

trc = $\bullet\sigma_1,\tau_1$®... $\bullet\sigma_n,\tau_n$® is a finite timed trace, and Trc = $\bullet\sigma_1,\tau_1$®... $\bullet\sigma_i,\tau_i$® ... is an infinite timed trace

TRC = $\alpha_1\alpha_2 \dots \alpha_j \dots$ is an infinite untimed trace

T is the set of timers $T_1, T_2, \dots, T_{Nt}$, where Nt is the number of timers

ts= $(t_1, t_2, \dots, t_{Nt})$ is the Nt-uplet representing the current values of all the timers (ts = timer state)

$\mathcal{T}$ is the set of all possible values of the Nt-uplet ts

E_T is the set of enabling boolean functions, w.r.t. $T=\{T_1, \dots, T_{Nt}\}$, i.e., depending on ts=$(t_1, \dots, t_{Nt})$

C is the set of counters $C_1, C_2, \dots, C_{Nc}$, where Nc is the number of counters.

Vc_i is an alphabet associated to counter C_i.

cs= $(c_1, c_2, \dots, c_{Nc})$ is the Nc-uplet representing the current values of all the counters (cs = counter state)

$\mathcal{C}$ is the set of all possible values of the Nc-uplet cs

E_C is the set of enabling boolean functions, w.r.t. $C=\{C_1, \dots, C_{Nc}\}$, i.e., depending on cs=$(c_1, \dots, c_{Nt})$

TA means: a timed automaton. Such TA is defined by $A^t=(Q,V,T,\mathcal{V},\delta, q_0)$

$\mathcal{L}_{A^t}$ is the timed language accepted by the TA A^t

Tr =[q1;σ;q2;E(ts),R;K(cs)] defines a timed transition of A^t, where σ[V, E(ts)[E_T, R⁄T, and K(cs)[E_C

V^* is the set of finite sequences of events over the alphabet V

Mt_i is the maximum value a timer T_i is compared to.

Mt is the maximum value any timer is compared to, i.e., : $(\forall\ Mt_i : Mt_i \leq Mt)$ and $(\exists Mt_i : Mt=Mt_i)$.

Mc_i is a bound on the counter C_i, and $Mc=\sup(Mc_i)$, i.e., : $(\forall\ Mc_i : Mc_i \leq Mc)$ and $(\exists Mc_i : Mc=Mc_i)$.

$Ext_W(A^t)$ is the extension of A^t to the alphabet W, with V⁄W (see def. 20)

$A^t\infty B^t$, A^t□B^t and $A^t||B^t$ are three types of products of two TA A^t and B^t, (def. 18, 21, 22)

tick is the event representing the passing of one uct

$A^{ut}=(Q^{ut},V\cup\{tick\}, \delta^{ut}, q_0^{ut})$ is a FSM called untimed automaton (UA) over the alphabet $V\cup\{tick\}$

$\mathcal{L}_{A^{ut}}$ is the untimed language accepted by A^{ut}

tr = [q1;σ;q2] defines a transition of A^{ut}, where σ [$V\cup\{tick\}$ and q1, q2 [Q^{ut}

pr(T) is the set of sequences which are prefixes of the sequence T

$P_W(A^{ut})$ is the projection of A^{ut} on alphabet $W\cup\{tick\}$, with W⁄V

$A^{ut}\times B^{ut}$ is the synchronized product of two untimed automata A^{ut} and B^{ut}.

UntimeT and *UntimeL* are operators for untiming respectively timed traces and timed languages

UntimeA is the operator for untiming timed automata, i.e., $A^{ut}=$*UntimeA*(A^t) .

Table 1. Notations

2. Real-Time systems specifications

For specifying a real-time discrete event system (DES), we use a global digital clock, and the set N of natural numbers is our domain of time. The time is then modeled by a global variable, noted τ and called *discrete time* : τ is initially equal to zero and is incremented by one after the passing of each unit of clock time (uct) ([Os90, BW92, OW90]).

2.1. Timed traces and Timed languages

A *finite timed trace* trc over an alphabet V is a finite sequence of pairs •σ_i,τ_i®, where σ_i is an event of V, and τ_i is an integer such that τ_{i+1} ε τ_i . Such trace is represented by trc = •σ_1,τ_1®...•σ_n,τ_n® and contains all events that have occurred before time τ_n+1. Each •σ_i,τ_i® means that the event σ_i has occurred when the discrete time is equal to τ_i. It is clear that there is an inaccuracy of one uct on the exact delay of event occurrences.

An *infinite timed trace* Trc over an alphabet V is an infinite sequence of pairs •σ_i,τ_i® ; any finite prefix of Trc is called a finite timed trace over V. Such infinite trace is represented by Trc= •σ_1,τ_1®...•σ_i,τ_i®... Each pair •σ_i,τ_i® defined in Trc is called a component of Trc which is noted : •σ_i,τ_i® □Trc. Since a τ_i may be equal to τ_{i+1}, several consecutive events may occur at the same discrete time, i.e., during one uct.

Definition 1. (Finiteness property)

An infinite timed trace respects the finiteness property (FP) if the number of events executed during one uct is bounded by an arbitrary constant Mc. Formally, Trc = •σ_1,τ_1®...•σ_i,τ_i® ... respects the FP if and only if : ¢ $i > 0$, ¡ $j > i$ such that $\tau_{j-1} = \tau_i < \tau_j$ and $j \leq i+Mc$. The FP is differently defined in [TH92], where it only requires that a finite number of events occur in any finite time interval.

■

Example 1. Let Trc be the following infinite trace Trc=•σ_1,2®•σ_2,4®...•σ_i,2i® ... Trc respects the finiteness property because one event occurs when τ is even, and no event occurs when τ is odd.

■

Example 2. Let Trc be the following infinite trace Trc=•σ_1,1®^{1} •σ_2,4®^{2}...•σ_i,2i®^{i}..., where •σ,τ®^{p} means that σ occurs p times when the discrete time is equal to τ. Trc does not respect the FP because the number of events during one uct is not bounded.

■

Definition 2. (Timed trace and timed language)

In this paper, we consider only *infinite* timed traces. Such traces, will be simply called *timed traces***.**

A *timed language* $\mathcal{L}$ over an alphabet V is a set of infinite timed traces over V.

We say that $\mathcal{L}$ respects the finiteness property (FP) if all its timed traces respect the FP. ■

Infinite timed traces, which will be simply called *timed traces* , are executed by non terminating processes. This is not really a restriction. In fact, a terminating process which may be executed infinitely often, can also be considered as a non terminating process.

Definition 3. (Projection of a timed trace)

Let V be a subset of an alphabet W, and let Trc= •σ_1,τ_1®...•σ_i,τ_i®... be a timed trace over W. The *projection* of Trc on V, noted Proj_V(Trc), is obtained by removing from Trc all •σ_i,τ_i®, where σ_i□V.

■

Definition 4. (Projection and Extension of a timed language)
Let V be a subset of an alphabet W. Let $\mathcal{L}_1$ be a timed language over W. The projection of $\mathcal{L}_1$ on V, noted $\mathrm{Proj}_V(\mathcal{L}_1)$, is defined by : $\mathrm{Proj}_V(\mathcal{L}_1)$= {Trc, over V | ¡ Trce □ $\mathcal{L}_1$ with Trc=Proj_V(Trce)};
Let $\mathcal{L}_2$ be a timed language over V. The extension of $\mathcal{L}_2$ to W, noted $\mathrm{Ext}_W(\mathcal{L}_2)$, is defined by :

$\mathrm{Ext}_W(\mathcal{L}_2)$={Trc , over W | Proj_V(Trc)□ $\mathcal{L}_2$}. ■

Remark 1 : a) if W=V then $\mathrm{Proj}_V(\mathcal{L})=\mathrm{Ext}_W(\mathcal{L})=\mathcal{L}$; **b)** $\mathrm{Proj}_V(\mathrm{Ext}_W(\mathcal{L}))=\mathcal{L}$ and $\mathcal{L}\,\Pi\,\mathrm{Ext}_W(\mathrm{Proj}_V(\mathcal{L}))$.

2.2. Timers and counters

A DES may be specified by a timed automaton, or simply a TA, which is an extended FSM accepting a timed language (def. 2). For defining a TA, we use several fictitious timers and counters.

Definition 5. (Timer)
A fictitious timer T_i is a conceptual entity associated to a variable t_i belonging to the set N of natural numbers . t_i is automatically incremented after the passing of one uct, and is called the current value of timer T_i. The operations we can do on the timer are :

- Reset : a timer T_i, whose value t_i is increasing regularly by one after each uct, can be set to zero. t_i represents therefore the time elapsed from the last reset of timer T_i.
- Comparison : the value t_i of timer T_i can be compared to a constant integer. The comparison operators are =, > and ≤ . Other operators < and ε are not necessary because timer values are integers.

Initially, when the discrete time τ is equal to zero, t_i also is equal to zero. ■

We deduce that if several timers T_1, T_2, ..., T_{Nt} are used, then their current values t_1, t_2, ..., t_{Nt} are automatically and *simultaneously* incremented after the passing of one uct, i.e. when the discrete time τ is incremented. Therefore, all the timers are synchronized on the digital global clock.

Definition 6. (Timer state)
Let Nt (or |T|) be the number of timers T_1, T_2, ..., T_{Nt}. The Nt-uplet ts=(t_1,..., t_{Nt}), where t_i is the current value of timer T_i, is called the *current timer state* . ■

Definition 7. (T_Condition, set E_T)
Let T={ T_1, T_2, ..., T_{Nt}} be a set of timers. A T_Condition E(ts), w.r.t. T, is a boolean function depending on the current timer state ts=(t_1,..., t_{Nt}). E(ts) is formed from : **a)** canonical boolean functions t_i~k, where t_i is the current value of a timer T_i , k□N^*, and ~ is =, ≤ or > ;
b) operators AND(□), OR(Δ), and NOT(□) on these canonical boolean functions.
The set of all T_Conditions, w.r.t. T, is noted E_T.

■

Definition 8. (Counter)

A fictitious counter C_i, w.r.t. an alphabet Vc_i is a conceptual entity associated to a variable c_i belonging to N. c_i is called the current value of C_i, and is automatically : **a)** incremented after the occurrence of any event of Vc_i; **b)** set to zero after the passing of one uct, i.e.,when τ is incremented. ■

Definition 9. (counter state)
Let Nc (or |C|) be the number of timers C_1, C_2, ..., C_{Nc}. The Nc-uplet cs=(c_1,..., c_{Nc}), where c_i is the current value of counter C_i, is called the *current counter state* .

■

Definition 10. (F_Condition, set E_C)
Let C={ C_1, C_2, ..., C_{Nc}} be a set of counters. A F_Condition K(cs), w.r.t. C, is a boolean function depending on the current counter state cs=(c_1,..., c_{Nc}). K(cs) is formed from : **a)** canonical boolean functions $c_i < Mc_i$, where c_i is the current value of a counter C_i, and Mc_i□N^*; **b)** operator AND(□) on these canonical functions. The set of all F_Conditions, w.r.t. C, is noted E_C.

■

Example 3. if K(cs)=($c_1 < Mc_1$) must be always true, and C_1 is w.r.t. Vc_1, then no more than Mc_1 events of Vc_1 may occur during one uct.

■

2.3 Timed Automata for real-time processes

For defining a TA, we use in general:
- a global digital clock which informs about the passing of one uct,
- a finite set of fictitious digital timers (def. 5), for specifying the timing requirements,
- a finite set of counters (def. 8), for respecting the finiteness property (def. 1).

Definition 11. (Timed transition, and Reset)
Let A=(Q,V,δ,q_0) be a FSM where Q is a set of states, V is an alphabet, q_0 is the initial state, and $\delta \Pi Q \infty V \infty Q$ defines the transitions, i.e., a transition of A can be represented by [q_1;σ;q_2].
Let T={T_1, ..., T_{Nt}} be a set of timers, and let C={C_1, ..., C_{Nc}} be a set of counters, w.r.t $Vc_i \Pi V$, for i=1,2, ..., N_c. Let E_T (resp. E_C) be the sets of T_Conditions (resp. F_Conditions), w.r.t. T (resp. C).
A *timed transition* , w.r.t. T and C, is defined by Tr=[q_1;σ;q_2;E(ts);R;K(cs)], with σ□V, q_1,q_2□Q, E(ts)□E_T, K(cs)□E_C, and RΠ T. R is called *Reset* of the transition Tr. The semantics of Tr is the following. Let q_1 be the current state : **(1)** σ may occur only if E(ts) (def.7) *and* K(cs) (def. 10) are true; **(2)** after the occurrence of σ : **a)** the state q_2 is reached, timers of R are set to zero, *and*
b) c_i is incremented if σ □ Vc_i, for i=1,2, ..., N_c.
Besides, K(cs)=(c_{i1}<Mc_{i1})□...□(c_{ip}<Mc_{ip}), where c_{i1},..., c_{ip} are all counters respectively w.r.t. Vc_{i1},..., Vc_{ip}, such that σ□$Vc_{i1} \leftrightarrow ... \leftrightarrow Vc_{ip}$.

■

Informally, the event σ of V in Tr=[q_1;σ;q_2;E(ts);R;K(cs)] may occur only if the T_Condition E(ts) is true. Besides, if σ □ Vc_i (ΠV), then (c_i<Mc_i) also must be true for occurrence of σ.

Definition 12. (Enabled and Eligible timed transition)

A timed transition $[q_1;\sigma;q_2;E(ts);R;K(cs)]$ is *enabled* if conditions for occurrence of σ are true (def.11).
A timed transition $Tr=[q_1;\sigma;q_2;E(ts);R;K(cs)]$ is *eligible* if :

Tr is enabled or will become enabled with the passing of time (without occurrence of any event). ■

A timed automaton (TA) A^t can then be constructed from the FSM $A=(Q,V, \delta,q_0)$, the finite sets T (of timers $T_1,...,T_{Nt}$) and C (of counters $C_1,...,C_{Nc}$). For that, we transform each transition $tr=[q_1;\sigma;q_2]$ of A into a timed transition Tr (def.11) by associating to it, a T_condition E(ts), a Reset, and a F_Condition.

In this paper, we consider only TA which accept (def.14) a timed language, i.e., a set of infinite timed traces. Here is a simple example, where we see that a TA is convenient for specifying a DES.

Example 4. Let's consider a communicating system which executes the three following service primitives : connect.request, connect.confirm, and disconnect.indication. These primitives are respectively abbreviated by *cr* , *cc* , *di* . The informal desired behaviour is the following. The primitive *cr* is first executed. It can be accepted and followed by *cc* , or refused and followed by *di* . And this process is repeated indefinitely. Between two consecutive *cr* , there must be at most 9 uct. After *cc* or *di* , we must wait at least 3 uct before the next *cr* . After its execution, if *cr* is not refused (i.e., not followed by *di*) 2 uct after its occurence, it will be inevitably accepted (i.e., followed by *cc*) within 3 uct after its occurence. With this informal specification, the finiteness property (def.1) is automatically respected, because of the minimum 3 uct between *cc* or *di* and *cr* .

This desired behaviour is formally specified by the TA of figure 1, which uses two timers T_1 and T_2. T_1 is used for defining timing requirements between : two *cr* , *cr* and *cc* , *cr* and *di* . T_2 is used for defining timing requirements between : *cc* and *cr* , *di* and *cr* . We may also use one counter C_1, w.r.t. $Vc_1=V=\{cr, cc, di\}$ with $Mc_1=2$, but in this example, the counter is not really necessary. In fact, the timing requirements ensure that the finiteness property is respected. But in general, they do not.

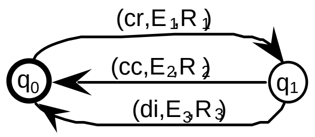


Figure 1. Timed automaton

In this example, Nt=2, $T=\{T_1,T_2\}$, $ts=(t_1, t_2)$, C_1 is w.r.t. V, the F_Condition of all timed transitions is $K(c_1)=(c_1 < 2)$, the T_Conditions are $E_1(ts)=((t_1 \leq 9)\ 3\ (t_2 > 2))$, $E_2(ts)=(t_1 \leq 3)$, $E_3(ts)=(t_1 \leq 2)$, and the Resets are $R_1=\{T_1\}$, $R_2=\{T_2\}$, $R_3=\{T_2\}$. Let's now give a formal definition of a timed automaton. ■

Definition 13. (Timed automaton)
A timed automaton $A^t=(Q,V,T,\mathcal{V},\delta,q_0)$ is defined as follows. Q is the set of states, q_0 is the initial state, V is the alphabet, T is the set of timers $T_1, T_2,...,T_{Nt}$, $\mathcal{V}=\{Vc_i \mid \text{for } i=1,2,...,Nc\}\ 1\ 2^V$, where each Vc_i is associated to one counter C_i, and $\delta\prod Q\infty V\infty Q\infty E_T\infty 2^T\infty E_C$ defines the timed transitions (def.11), where E_T and E_C are the sets of T_Conditions and F_Conditions (def.7 and 10).
Besides, A^t *accepts only infinite timed traces* (def. 14), and is called a sequential TA. ■

In example 4, the TA of figure 1 is defined by A^t=(Q,V,T,{V},δ,q_0) where: Q={q_0, q_1}, V={*cr*,*cc*,*di*}, T={T_1,T_2}, C={C_1}, Mc_1=2, δ={[q_0;*cr*;q_1;E_1;R_1;K], [q_1;*cc*;q_0;E_2;R_2;K], [q_1;*di*;q_0;E_3;R_3;K]}.

Definition 14. (Acceptance of a timed trace and of a language, equivalence, partial order relation)
Let A^t be a TA (Q,V,T,𝒱,δ,q_0) and let Trc= •σ_1,τ_1®...•σ_i,τ_i®... be an infinite timed trace.

- Trc *is accepted by* A^t , is formally defined by :
∀ i□N^* : ∃ Tr_i=[q_{i-1};α_i;q_i;E_i(ts);R_i;K_i(cs)]□δ with : (α_i=σ_i) 3 ((τ=τ_i) □ (E_i(ts)3K_i(cs)=True)).

Informally, a system specified by A^t may execute a trace accepted by A^t.
- A timed language, noted $\mathcal{L}_{At}$, *is accepted by* A^t if it contains all and only the traces accepted by A^t.
- A^t_1 and A^t_2 are *equivalent* , and noted A^t_1≅A^t_2, if and only if $\mathcal{L}_{A^t_1}$ =$\mathcal{L}_{A^t_2}$.
- A^t_1 *is smaller than or equal to* A^t_2, and noted A^t_1≤A^t_2, if and only if $\mathcal{L}_{A^t_1}$ ∏$\mathcal{L}_{A^t_2}$. ■

Property 1. Let A^t=(Q,V,T,𝒱,δ,q_0) be a timed automaton specifying a non terminating system, with 𝒱={Vc_1, Vc_2,...,Vc_{Nc}}∏2^V. If Vc_1≈...≈Vc_{Nc}=V, then the language $\mathcal{L}_{At}$ accepted by A^t (def.14) respects the finiteness property. In this case, we say that A^t respects the finiteness property.
Proof : *See Appendix A .* ■

Definition 15. (set 𝒯 of timer states)
Let T={T_1, ..., T_{Nt}} be a set of timers used for defining a TA A^t, and let Mt_i be the maximum value a timer T_i is compared to, for defining the T_Conditions (def.7) of all the transitions of A^t. In this case, the value t_i of T_i does not need to be incremented as soon as t_i=Mt_i+1. In fact, in this case the incrementation would have no influence on truths of the T_Conditions. Therefore, we can limit the value of t_i by Mt_i+1, for i=1, 2, ..., Nt, and the set 𝒯 of timer states ts=(t_1, ... , t_{Nt}) is equal to or included in •0;Mt_1+1®∞...∞•0;Mt_{Nt}+1®, where •0;Mt_i+1® is the set of integers belonging to the interval [0;Mt_i+1]. ■

In example 4, Mt_1=9, and Mt_2=2, and then 𝒯 ⊆ •0;10®6•0;3®

Definition 16. (Addition between 𝒯 and N)
Let T={T_1, ..., T_{Nt}} be a set of timers used for defining a TA A^t. The addition between 𝒯 and N is defined as follows : if ts=(t_1, ..., t_{Nt})□𝒯 and p□N, then ts+p=(inf(t_1+p,Mt_1+1),..., inf(t_{Nt}+p, Mt_{Nt}+1)).
Where inf is defined by : inf(A,B) □{A,B} and ((inf(A,B)=A) □ (A≤B)). ■
Intuitively, if ts is the current timer state, then ts+p is the futur timer state after the passing of p units of clock time (uct). In example 4, if ts=(4,1) and p=3, then ts+3=(inf(4+3;10), inf(1+3;3))= (7,3) □ (7,4).

Definition 17. (set 𝒞 of counter states)
Let C={C_1, ..., C_{Nc}} be a set of counters used for defining a TA A^t, and let Mc_i be the maximum value which bounds the value c_i. Therefore, the set 𝒞 of counter states cs=(c_1, ... , c_{Nc}) is equal to or included in •0;Mc_1®∞...∞•0;Mc_{Nc}®, where •0;Mc_i® is the set of integers belonging to the interval [0;Mt_i]. ■

In example 4, Mc_1=2, and then 𝒞 ⊆ •0;2®.

2.4. Product of two timed automata over the same alphabet

Let A_1^t and A_2^t be two TA (def.13) defined over the same alphabet V. An intuitive definition of the synchronized product of A_1^t and A_2^t, noted $A_1^t6A_2^t$, is the following. $A_1^t6A_2^t$ is a TA specifying a system which may execute *all and only* the infinite timed traces accepted by both A_1^t and A_2^t.

Definition 18. (Product over a same alphabet)
Let A_i^t=(Qi,V,Ti,$\mathcal{V}$i,δi,qi_0), for i=1,2, be two TA over a same alphabet V, with T1↔T2=□, and $\mathcal{V}$i={Vci_1,...,Vci_{Nci}}. Each A_i^t uses then a set Ti={Ti_1,...,Ti_{Nti}} of timers and a set Ci={Ci_1,...,Ci_{Nci}} of counters, where each Ci_j is w.r.t. Vci_j. The product, noted A^t=$A_1^t6A_2^t$, is defined by A^t=(Q,V,T,$\mathcal{V}$,δ,q_0), with $\mathcal{V}$=$\mathcal{V}$1≈$\mathcal{V}$2, T=T1≈T2, Q∏Q16Q2, q_0=•$q1_0$,$q2_0$®□Q, and :
<u>Definition of δ</u> : Let E_{T1}, E_{T2} and E_T be the set of T_Conditions (def. 7), respectively w.r.t. T1, T2 and T=T1≈T2. Let E_{C1}, E_{C2} and E_C be the set of F_Conditions (def. 10), respectively w.r.t. C1, C2 and C=C1≈C2. Then ∀ •q1,q2®, •r1,r2®□Q, ∀ σ□V, ∀ E□E_T, ∀ R∏T, ∀ K□E_C :
([•q1,q2®,σ,•r1,r2®,E,R,K] □δ) □ (∃ E1□E_{T1}, ∃ E2□E_{T2}, ∃ R1∏T1, ∃ R2∏T2, ∃ K1□E_{C1}, ∃ K2□E_{C2},)
(with : R=R1≈R2, E=E1□E2, K=K1□K2, and
)
([q1,σ,r1,E1,R1,K1]□δ1, and [q2,σ,r2,E2,R2,K2]□δ2.)

■

Theorem 1. If $\mathcal{L}A_1^t$ and $\mathcal{L}A_2^t$ are respectively the timed languages accepted by A_1^t and A_2^t over the same alphabet, then: $\mathcal{L}A_1^t{}_{6A_2^t}$=$\mathcal{L}A_1^t$ ↔ $\mathcal{L}A_2^t$. (**Proof :** *See Appendix A*). ■

Property 2. In def. 18, if $Vc1_1$≈...≈$Vc1_{Nc1}$=$Vc2_1$≈...≈$Vc2_{Nc2}$=V, then A_1^t, A_2^t, and $A_1^t\infty A_2^t$ respect the finiteness property. (**Proof :** *See Appendix A*).

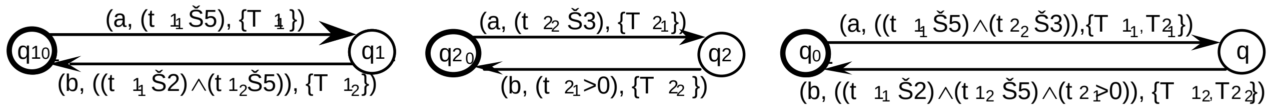
■

Remark2 : **a)** In def.18, if there exist i≤Nc_i and j≤Nc_j such that $Vc1_i$=$Vc2_j$, then counters $C1_i$ and $C2_j$ may be a same counter for defining A_1^t, A_2^t , and $A_1^t6A_2^t$. In fact, the values $c1_i$ and $c2_j$ are incremented and set to zero simultaneously. Therefore, one counter, for example $C1_i$, is sufficient.

b) From theorem 1, we deduce that if A_1^t and A_2^t specify two sequential processes over the same alphabet, then *their synchronized product also specifies a sequential process*.

Example 5. A_1^t and A_2^t are respectively represented on figures 2.a and 2.b. A_1^t=(Q1,V,T1,$\mathcal{V}$,δ1,$q1_0$) and A_2^t=(Q2,V,T2,$\mathcal{V}$,δ2,$q2_0$), with $\mathcal{V}$={Vc1}={Vc2}={V}, Q1={$q1_0$,q1}, T1={$T1_1$,$T1_2$}, Q2={$q2_0$,q2}, T2={$T2_1$,$T2_2$}, V={a,b},and Mc1=Mc2=10. The values of timers $T1_1$,$T1_2$, $T2_1$ and $T2_2$ are respectively $t1_1$,$t1_2$, $t2_1$ and $t2_2$. The values of counters C1 and C2 are respectively c1 and c2. δ1={[$q1_0$,a,q1,$E1_1$,{$T1_1$},K1], [q1,b,$q1_0$,$E1_2$,{$T1_2$},K1]}, with : $E1_1$=($t1_1 \le 5$), $E1_2$=($t1_1 \le 2$) □ ($t1_2 \le 5$), and K1=(c1<10). δ2={[$q2_0$,a,q2,$E2_1$,{$T2_1$},K2], [q2,b,$q2_0$,$E2_2$,{$T2_2$},K2]}, with : $E2_1$=($t2_2 \le 3$), $E2_2$=($t2_1 > 0$), and K2=(c2<10). Since V=Vc1=Vc2, only one counter, for example C1, is used (remark 2.a), and transitions of A_1^t, A_2^t , and $A_1^t6A_2^t$ are enabled only if (c1<10). The synchronized product of A_1^t and A_2^t is represented on figure 2.c.

■


(a, (t 11 Š5), {T 11})
q10
q1
(b, ((t 11 Š2)∧(t 12Š5)), {T 12})
(a, (t 22 Š3), {T 21})
q20
q2
(b, (t 21>0), {T 22 })
(a, ((t 11 Š5)∧(t 22 Š3)),{T 11,T21})
q0
q
(b, ((t 11 Š2)∧(t 12 Š5)∧(t 21>0)), {T 12,T22})

2.a. A_1^t **2.b.** A_2^t **2.c.** $A^t = A_1^t \infty A_2^t$

Figure 2. Synchronized product over the same alphabet

2.5. Product of two timed automata over alphabets V_1 and V_2 with $V_1 \Pi V_2$

Before defining the product over alphabets V_1 and V_2, with $V_1 \Pi V_2$, let's give two definitions.

Definition 19. (Operator H on En)

Let $E_1(ts)$, $E_2(ts)$, ..., $E_k(ts)$ be k T_Conditions (def. 7), depending on a set of timers $\{T_1, T_2, ..., T_{Nt}\}$. We define $E(ts)=E_1(ts)HE_2(ts)H...HE_k(ts)$ as follows.

$$(E(ts)=\text{false}) \square \{ \forall i \square \{1, ..., k\}, \forall p \square N : E_i(ts+p)=\text{false} \}$$

■

Informally, $E_1(ts)H...HE_k(ts)$ is false if and only if all $E_i(ts)$ are false and remain false with the passing of time. If for instance $T=\{T_1\}$,$E_1(t_1)=(t_1\leq5)$, $E_2(t_1)=((t_1>2)\square(t_1\leq6))$, then $E(t_1)=E_1(t_1)HE_2(t_1)=(t_1\leq6)$.

Definition 20. (Extension of a timed automaton)

Let $A^t=(Q,V,T,\mathcal{V},\delta,q_0)$ be a TA over an alphabet V with $T=\{T_1,...,T_{Nt}\}$, $\mathcal{V}=\{Vc_1,...,Vc_{Nc}\}$, and then $C=\{C_1,...,C_{Nc}\}$. Let E_T (resp. E_C) be the set of T_Conditions w.r.t. T (resp. F_Conditions w.r.t. C). Let W be an alphabet such that $V \Pi W$. The *extension* of A^t to the alphabet W, noted $Ext_W(A^t)$, is a TA defined by $(Q,W,T,\mathcal{V},\delta_{ext},q_0)$, where $\delta_{ext} \Pi Q \infty W \infty Q \infty E_T \infty 2^T \infty E_C$ is such that :

(1) $\forall$ q1,q2$\square$Q, $\forall \sigma \square V$, $\forall E \square E_T$, $\forall R \Pi T$, $\forall K \square E_C$: $[\bullet q1,\sigma,q2,E,R,K] \square \delta \square [\bullet q1,\sigma,q2,E,R,K] \square \delta_{ext}$.

(2) $\forall q \square Q$: Let $E_i \square E_T$, for i=1,..., k, be all the T_Conditions of E_T such that : $\exists q_i \square Q$, $\exists \sigma_i \square V$,

$\exists R_i \square 2^T$, $\exists K_i \square E_C$, with $[\bullet q,\sigma_i,q_i,E_i,R_i,K_i] \square \delta$, and let then $E= E_1HE_2H... HE_k$.

Then $\forall \sigma \square W\text{-}V$: $[\bullet q,\sigma,q',E',R,K] \square \delta_{ext} \square$ (q'=q, E'=E, R=$\square$, K=True).

If $B^t=Ext_W(A^t)$, then A^t is called projection of B^t in the alphabet V, and is noted $A^t=Proj_V(B^t)$. ■

Informally, $Ext_W(A^t)$ is obtained by adding selfloops of all events of W-V to each state of A^t . The resets of these selfloops are empty, and their T_conditions are defined as follows. The T_Condition of the added selfloops at a state q of A^t is true if at least one of the transitions defined in A^t from q is eligible. The F_Condition for events of W-V is always true, and then $Ext_W(A^t)$ does not necessarily respect the finiteness property (property 1).

Intuitively, let $\mathcal{P}_{ext}$ and $\mathcal{P}$ be two non terminating processes respectively specified by $Ext_W(A^t)$ and A^t, where A^t is defined over the alphabet V. An external agent who can observe all and only the events of V, cannot differentiate the two processes. If the T_Conditions of the added selfloops in $Ext_W(A^t)$ were always true, the external agent may see $\mathcal{P}_{ext}$ as a terminating process. In fact in this case, it is possible that a selfloop of an event of W-V is indefinitely executed. In Example 6 (next section 2.6), the two timed automata of figures 3.a. and 3.b. are extended into the two timed automata of figures 4.a and 4.b.

Lemme 1. If $\mathcal{L}_{A^t}$ is the timed language accepted by a TA A^t over an alphabet V, and if W is an alphabet such that $V \Pi W$, then : $\mathcal{L}_{Ext_W(A^t)}=Ext_W(\mathcal{L}_{A^t})$. (see def. 4 for $Ext_W(\mathcal{L}_{A^t})$) (**Proof :** *See Appendix A*). ■

Before defining formally the product over V1 and V2 with V1∏V2, let's give an intuitive definition. Let A_1^t and A_2^t be two TA defined over V1 and V2 with V1∏V2. The product of these two TA is a TA specifying a system which may execute *all and only* the infinite timed traces which both :

are accepted by A_2^t, and whose projections (def. 3) on V1 are accepted by A_1^t .

Definition 21. (Product over V1 and V2 with V1∏V2)

Let A_i^t=(Qi,Vi,Ti,$\mathcal{V}i$,δi,qi_0), for i=1,2, be two TA (def.13) over alphabets V1 and V2, with V1∏V2, T1↔T2=□, and $\mathcal{V}i$={Vci_1,...,Vci_{Nci}}, i.e., each A_i^t uses a set Ci={Ci_1,...,Ci_{Nci}} of counters where each Ci_j is w.r.t. Vci_j. Their synchronized product, noted A_1^t□A_2^t, is defined by :

A_1^t□A_2^t =(Q,V2,T1≈T2,$\mathcal{V}1$≈$\mathcal{V}2$, δ,q_0)= $Ext_{V2}(A_1^t)$∞A_2^t (See def.18 and 20 for ∞ and $Ext_{V2}(A_1^t)$).

■

Theorem 2. If $\mathcal{L}A_1^t$ and $\mathcal{L}A_2^t$ are respectively the timed languages accepted by A_1^t and A_2^t respectively over alphabets V1 and V2, with V1∏V2, then: $\mathcal{L}A_1^t$□A_2^t=$\mathcal{L}$ $Ext_{V2}(A_1^t)$↔$\mathcal{L}A_2^t$. (**Proof :** *See Appendix A*).

■

Property 3. Let A_1^t and A_2^t be two TA, respectively over alphabets V1 and V2 with V1∏V2. If A_2^t respects the finiteness property (FP), then A_1^t□A_2^t respects the FP. (**Proof :** *See Appendix A*). ■

Remark3 : **a)** in definition 21, if V1=V2, then A_1^t∗A_2^t = A_1^t∞A_2^t (def. 18), because $Ext_{V2}(A_1^t)$=A_1^t ;

b) From theorem 2, we deduce that if A_1^t and A_2^t specify two sequential processes respectively over alphabets V1 and V2 with V1∏V2, then *their synchronized product also specifies a sequential process.*

2.6. General parallel product of two timed automata

Before defining formally the parallel product of two TA A_1^t and A_2^t, respectively over alphabets V1 and V2, let's give an intuitive definition. The product of A_1^t and A_2^t is a TA specifying a parallel system which may execute *all and only* the timed traces over the alphabet V1≈V2: **a)** whose projections (def.3) on V1 are accepted (def.14) by A_1^t *and* ; **b)** whose projections on V2 are accepted by A_2^t.

Definition 22. (Parallel product of two TA)

Let A_i^t=(Qi,Vi,Ti,$\mathcal{V}i$,δi,qi_0), for i=1,2, be two TA over alphabets V1 and V2, with T1↔T2=□, and $\mathcal{V}i$={Vci_1,...,Vci_{Nci}}.Their parallel product, noted A_1^t||A_2^t, is defined by :

A_1^t||A_2^t= (Q,V1≈V2,T1≈T2,$\mathcal{V}1$≈$\mathcal{V}2$,δ,q_0) =$Ext_{V1≈V2}(A_1^t)$∞$Ext_{V2≈V1}(A_2^t)$. ■

Remark 4 : in definition 22, if V1∏V2 then A_1^t||A_2^t=A_1^t∗A_2^t, and if V1=V2 then A_1^t||A_2^t=A_1^t6A_2^t

Theorem 3. If $\mathcal{L}A_1^t$ and $\mathcal{L}A_2^t$ are the timed languages accepted by two TA A_1^t and A_2^t over alphabets V1 and V2, then : $\mathcal{L}A_1^t$||A_2^t=$\mathcal{L}$ $Ext_{V1≈V2}(A_1^t)$↔$\mathcal{L}$ $Ext_{V1≈V2}(A_2^t)$ (**Proof :** *See Appendix A*).

■

Property 4. If two TA A_1^t and A_2^t, respectively over alphabets V1 and V2, respect the finiteness property, then A_1^t||A_2^t respects the finiteness property. (**Proof :** *See Appendix A*).

■

Example 6. Let A_1^t=(Q1,V1,T1,$\mathcal{V}$1,δ1,q1$_0$) and A_2^t=(Q2,V2,T2,$\mathcal{V}$2,δ2,q2$_0$) (figure 3), with $\mathcal{V}$i={Vci$_1$}={Vi}, Qi={qi$_0$,qi}, Ti={Ti$_1$,Ti$_2$}, Mc=Mci$_1$=10, for i=1,2. V1={*a*,*b*}, V2={*a*,*c*}. The values of timers T1$_1$, T1$_2$, T2$_1$ and T2$_2$, are respectively t1$_1$, t1$_2$, t2$_1$ and t2$_2$, and the values of counters C1$_1$ and C2$_1$, are respectively c1$_1$ and c2$_1$.

δ1={[q1$_0$,*a*,q1,E1$_1$,{T1$_1$},K1],[q1,*b*,q1$_0$,E1$_2$,{T1$_2$},K1]}, E1$_1$=(t1$_1$≤5), E1$_2$=(t1$_1$≤2)□(t1$_2$≤5), K1=(c1$_1$<10).

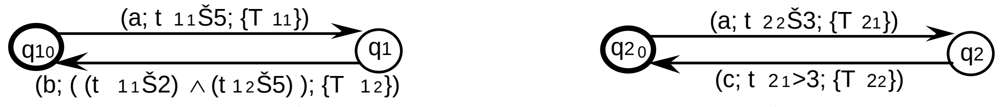
δ2={[q2$_0$,*a*,q2,E2$_1$,{T2$_1$},K2],[q2,*c*,q2$_0$,E2$_2$,{T2$_2$},K2]}, E2$_1$=(t2$_2$≤3), E2$_2$=(t2$_1$>3), and K2=(c2$_1$<10).

Ext$_{V1\approx V2}$(A_1^t) and Ext$_{V2\approx V1}$(A_2^t) are on figure 4, and the product of the two parallel TA is on figure 5. The F_Conditions (def. 9) of transitions in $A_1^t||A_2^t$ (fig.5) are as follows.

Transitions with event *a* are enabled only if both (c1$_1$<10) and (c2$_1$<MA) are true (*a*□Vc1$_1$↔Vc2$_1$).

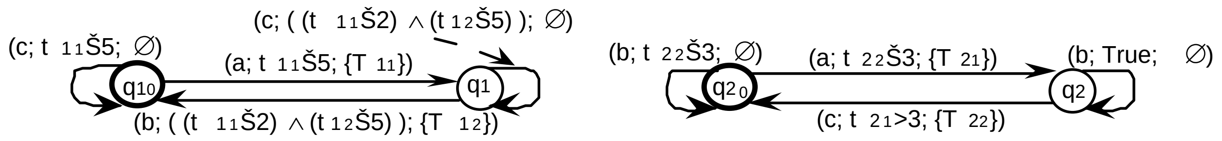
Transitions with event *b* are enabled only if (c1$_1$<10) is true (because *b* □ Vc1$_1$).

Transitions with event *c* are enabled only if (c2$_1$<10) is true (because *c* □ Vc2$_1$).



3.a. A_1^t **3.b.** A_2^t

Figure 3. Two concurrent automata



4.a. Ext$_{V1\approx V2}$(A_1^t) **4.b.** Ext$_{V2\approx V1}$(A_2^t)

Figure 4. Extensions of the two concurrent automata of figure 3

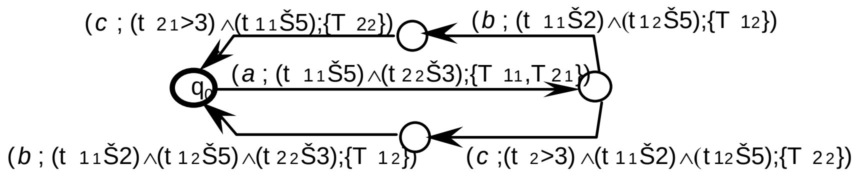

(c ; (t 21>3) ∧(t 11Š5);{T 22})
(b ; (t 11Š2) ∧(t 12Š5);{T 12})
(a ; (t 11Š5) ∧(t 22Š3);{T 11,T 21})
q0
(b ; (t 11Š2) ∧(t 12Š5) ∧(t 22Š3);{T 12})
(c ;(t 2>3) ∧(t 11Š2) ∧(t 12Š5);{T 22})

Figure 5. Synchronized product $A_1^t||A_2^t$

Definition 23. (Independent and concurrent DES)

Let A_i^t be two TA over alphabets Vi, for i=1, 2, specifying two processes.

If V1↔V2=□, the two processes are independent with each other.

If V1↔V2□□, the two processes are concurrent. In fact, they may run in parallel by executing respectively events of V1 - V2 and V2 - V1 , but they must execute conjointly events of V1↔V2. ■

2.7. Untimed traces and untimed languages

So far, an infinite sequence of events has been represented by a timed trace Trc=•σ_1,τ_1®...•σ_i,τ_i®...

If we introduce a fictitious event *tick* which represents the passing of one uct, the same sequence can be represented by an untimed trace TRC= $\alpha_1\alpha_2...\alpha_j$ ..., where each α_j for j=1,2,..., is equal to *tick* or to one of σ_1, σ_2, ...

Example 7. The timed Trc=•σ_1,2®...•σ_i,2i®... can equivalently be represented by the untimed :

TRC=*tick tick* σ_1*tick tick* σ_2...σ_{i-1}*tick tick* σ_i *tick tick* σ_{i+1}...

A formal definition of the untimed trace corresponding to a timed trace is the following.

Definition 24. (Untimed trace and operator *UntimeT*)
Let Trc = •σ_1,τ_1®... •σ_i,τ_i®... be a timed trace. We define the operator *UntimeT* , for obtaining the untimed trace TRC corresponding to Trc, by : TRC=*UntimeT*(Trc)=$\alpha_1\alpha_2 \dots \alpha_j \dots$
with : ($\alpha_{i+\tau_i}=\sigma_i$) and (α_j=*tick* , if ™ k>0 such that j=k+τ_k) , for i,j=1,2, ...
If Trc respects the finiteness property (def.1), we also say that TRC respects the finiteness property.
Let's notice that the operator *UntimeT* is a **bijection**.

■

Property 5. Let TRC= $\alpha_1\alpha_2 \dots \alpha_j \dots$ be a infinite untimed trace respecting the finiteness property .
∃ Mc>0 such that : ∀ k>0, ∃ l_1>k, ∃ l_2>k with α_{l_1} □*tick* , l_2-k ≤Mc+1, and α_{l_2}=*tick* .
Proof : *See Appendix A*

■

More informally, property 5 means that the untimed TRC corresponds to an *infinite* timed trace (by α_{l_1}□*tick*) which respects the *finiteness property* (by l_2-k ≤Mc+1 and α_{l_2}= *tick*).

Definition 25. (Untimed language and operator *UntimeL*)
Let $\mathcal{L}$ be a timed language. $\mathcal{L}^u$, which is called untimed language and noted $\mathcal{L}^u$=*UntimeL*($\mathcal{L}$), is defined by : $\mathcal{L}^u$=*UntimeL*($\mathcal{L}$)={TRC | ∃ Trc □ $\mathcal{L}$ with TRC=*UntimeT*(Trc) } ■

Theorem 5. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two timed languages over a same language.
UntimeL($\mathcal{L}_1$↔$\mathcal{L}_2$)=*UntimeL*($\mathcal{L}_2$)↔*UntimeL*($\mathcal{L}_2$). (**Proof** : *See Appendix A*

■

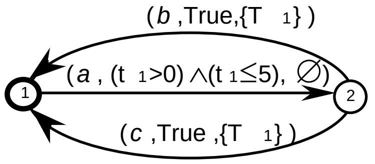
2.8. Untimed automata

Definition 26. (Untimed automaton and operator *UntimeA*)
Let A^t be a TA over an alphabet V which accepts (def. 14) a timed language $\mathcal{L}$, and let $\mathcal{L}^u$=*UntimeL*($\mathcal{L}$).
A^{ut} is the minimal FSM over the alphabet V≈{*tick* }, called untimed automaton (UA) which accepts the untimed language $\mathcal{L}^u$. In other words : $\mathcal{L}_{A^{ut}}$ =*UntimeL*($\mathcal{L}_{A^t}$), (def. 25, for *UntimeL*)
A sufficient condition of existence of A^{ut} is the finiteness of the set of timers.
We also define the surjective operator *UntimeA* such that : A^{ut}=*UntimeA*(A^t). ■

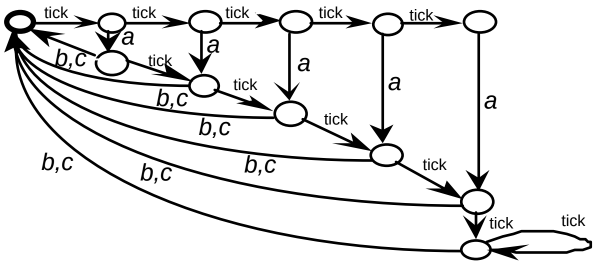
Example 8. Let's consider the timed A^t =(Q,V,T,{V},δ,q_0) on figure 6.a, where we use one timer T_1, and one counter C_1 w.r.t. V, with Mc_1=5 . In this specification, the value c_1 of C_1 is smaller than or equal to c_1, and since Mt_1=5 (def. 15), the value t_1 of T_1 is smaller than or equal to 6. The obtained untimed A^{ut} is on figure 6.b, each state being defined by •q,t_1,c_1®, where q is a state in A^t.

Remark 5: Since untimed traces accepted by A^{ut} correspond to infinite timed traces accepted by A^t, then A^{ut} accepts only infinite untimed traces, and does not contain indesirable states. An indesirable state is either a deadlock state or a state from which only a selfloop *tick* is executable.
- A deadlock in A^{ut} is indesirable, because it has no sense. In fact, a deadlock state means that the event *tick* is not executable. Therefore, the passing of time is stopped!

- A state from which only a selfloop *tick* is executable is indesirable, because it implies that A^{ut} accepts a trace TRC=*UntimeT*(Trc) where Trc is a **finite** timed trace!



6.a. A^t



6.b. A^{ut}

Figure 6. Timed and untimed automata

Informally, A^{ut} allows to represent a real-time system specified by A^t, as a system without timing requirement, but where a new event *tick* is added. This event, which models the passing of one unit of clock time (uct), is processed like any other event.

Let's give an idea of how A^{ut} is obtained from A^t over an alphabet V, when only one counter C_1, w.r.t. Vc_1=V is used. This implies that A^t respects the finiteness property (theorem 1). Let T={T_1,..., T_{Nt}} be a set of timers used for defining a TA A^t, let Mt_i be the maximum value a timer T_i is compared to, for defining the T_Conditions (def. 7) of all the transitions of A^t. In this case, the value t_i of T_i does not need to be incremented as soon as $t_i=Mt_i+1$ (def. 15). A state of A^{ut} is defined by •q1,ts,c_1®, where q1 is a state of A^t, ts=(t_1,..., t_{Nt}) is a timer state (def.6), and c_1 is a value of C_1. The passing of one uct is represented in A^{ut} by the event *tick* . Execution of *tick* from state •q1,ts,c_1® leads to state •q1,ts+1,0®, i.e., timers are incremented and the counter is set to zero. Execution of an event σ≠*tick* from state •q1,ts,c_1® of A^{ut} leads to state •q2,ts',c_1+1®, where q2 is a state of A^t which is reached by a transition tr=[q1,σ,q2,E,R,K] from state q1 of A^t (with E true for the current timer state ts, and $c_1<Mc_1$), and ts' is obtained from ts by setting to zero timers belonging to R. Besides, A^{ut} is minimal.

Remark 6 : a) if ts=(Mt_1+1, ... , Mt_{Nt}+1) then ts+1=ts. In this case, an event *tick* is a selfloop in A^{ut} ; **b)** Since two A_i^{ut} over alphabets V_i≈{*tick* }, for i=1,2, are FSM, we can use the classic synchronized product between them, noted $A_1^{ut}\infty A_2^{ut}$, where events of (V_1↔V_2)≈{*tick*} are executed conjointly.

c) The product *UntimeA*(A_1^t)∞*UntimeA*(A_2^t) may contain deadlocks, therefore it does not correspond to a real DES. In fact, a deadlock prevents the event *tick* , i.e., the passing of time is stopped.

Lemmes 2. Let $A^t=A_i^t$=(Q,V,T,$\mathcal{V}$,δ,q_0) be a TA and A^{ut}=*UntimeA*(A^t). Let's remind some notations :
a) Nt and Nc are the numbers of timers and counters; **b)** Mc bounds all the Mc_i, for i=1,...,Nc; **c)** Mt is the maximum constant any timer is compared to; **d)** |Q| and |δ| are numbers of states and of transitions.

2.a. The number $|Q^{ut}|$ of states of A^{ut} is bounded by : $\mathbf{|Q|*(Mt+2)^{Nt}*(Mc+1)^{Nc}}$

2.b. The number $|\delta^{ut}|$ of transitions of A^{ut} is bounded by : $\mathbf{(|Q| + |\delta|)*(Mt+2)^{Nt}*(Mc+1)^{Nc}}$

2.c. The complexity for calculating A^{ut} is in : $\mathbf{O(|Q^{ut}|^2)=O(|Q|^2*(Mt+2)^{2\infty Nt}*(Mc+1)^{2\infty Nc})}$ **.**

$|Q^{ut}|$, $|\delta^{ut}|$ and the complexity for calculating A^{ut} are then exponential in the numbers of timers and of counters. (**Proof** : *See Appendix A*).

■

Remark 7 : a) The number of counters is not really a problem. In fact, in general one counter is sufficient, for ensuring the finiteness property. Therefore, the complexity is essentially due to the number of timers. **b)** If the timing requirements are only between consecutive events, one timer is sufficient for specifying temporal constraints. In this case, the complexity is no more exponential.

Properties 6. Let A_1^t and A_2^t be two TA respectively over alphabets V_1 and V_2.

6.a. If $V_1=V_2$, then : $UntimeA(A_1^t \infty A_2^t) \leq UntimeA(A_1^t) \infty UntimeA(A_2^t)$

6.b. If $V_1 \prod V_2$, then : $UntimeA(A_1^t □ A_2^t) \leq UntimeA(A_1^t) \infty UntimeA(A_2^t)$

6.c. If V_1-V_2 □ □ and V_2-V_1 □ □, then : $UntimeA(A_1^t \| A_2^t) \leq UntimeA(A_1^t) \infty UntimeA(A_2^t)$

6.d. If $V_1=V_2$, then : $\mathcal{L}_{A_1^t} \prod \mathcal{L}_{A_2^t}$ □ $\mathcal{L}_{UntimeA(A_1^t)} \prod \mathcal{L}_{UntimeA(A_2^t)}$

6.e. If $V_1 \prod V_2$, then : $UntimeL(Proj_{V_1}(\mathcal{L}_{A_2^t})) = Proj_{V_1}(UntimeL(\mathcal{L}_{A_2^t}))$

(where $A \leq B$ means $\mathcal{L}_{A^{ut}} \prod \mathcal{L}_{B^{ut}}$). **Proof** : *See Appendix A*

■

2.9. Why untimed automata are useful.

Problem of using timed automata (TA) :

a) Respecting the timing requirements does not ensure to avoid deadlock states.

b) Timing requirements between events are specified by using some fictitious timers. Therefore, if we project a TA into an alphabet, a few events may disappear. In this case, we have to respecify temporal requirements between events, and then we have to redefine new fictitous timers. This is not self-evident.

Interest of using untimed automata (UA) :

A UA is a FSM. Therefore, all known methods used for FSMs can be used. Let's see two examples :

a) States respecting in general a certain "indesirable" property, in particular deadlocks states, may be removed; **b)** a UA defined over an alphabet $W'=W \approx \{tick\}$ can be projected in any alphabet $V \prod W'$.

Thus, before making some processings, a TA is untimed for obtaining a UA. But after the processing, it is convenient to transform the processed UA into an equivalent TA. This the object of the next section.

2.10. Timing untimed automata

Timing an untimed automaton is not self-evident, because for a UA A^{ut}, there are an infinite number of TA A_1^t, A_2^t, ..., A_i^t, ..., such that $UntimeA(A_1^t)=UntimeA(A_2^t)=...=UntimeA(A_i^t)=A^{ut}$. We propose an operator *TimeA* which, from a UA A^{ut}, generates a timed automaton with a new model different than the model previously defined. A logic question arises : why two different models are used for specifying a DES ?

The first model, previously defined, is used because it is more intuitive. In the case where a TA must specify a desired behaviour, it may be constructed manually by a user. In fact, the timers and counters are convenient fictitious entities which may be defined intuitively.

The second model is less intuitive, but it can be automatically and easily constructed from a UA. It uses only timers. Informally, if the alphabet of A^{ut} is $V \approx \{tick\}$, $TimeA(A^{ut})$ is obtained as follows.

a) A^{ut} is projected into the alphabet V, for obtaining $Proj_V(A^{ut})$.

b) For each state q of $Proj_V(A^{ut})$, several timers Tq_i are defined, and their values tq_i are incremented at each *tick*..

c) For each transition Tr of $Proj_V(A^{ut})$ which is executable from a state q1 and leads to a state q2 :

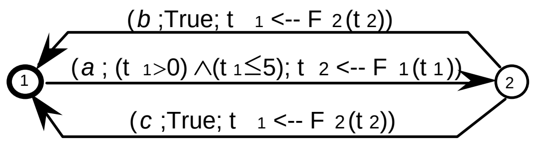
- Several enabling conditions $E_i(tq1_i)$ are defined. Each $E_i(tq1_i)$ depends on a timer $Tq1_i$ of state q1.
- Several initializations $tq2_j$ ♦ $F_i(tq1_i)$ are defined. Each initialization consists in initializing a timer $Tq2_j$ of q2 in function of the value $tq1_i$ of a timer $Tq1_i$ of q1. The index j also depends on $tq1_i$.

The semantics of the enabling conditions and of the initializations is the following.

- When a timer is initialized by a transformation, it becomes the **active timer**.
- The transition Tr may be executed only if its *enabling condition* depending of the active timer is true.
- When Tr is executed, a timer $Tq2_j$ is *initialized* in function of the value of the current active timer. $Tq2_j$ becomes the new active timer.

Let's give an idea of how the timers are defined. Each state q of *TimeA*(A^{ut}) (and of $Proj_V(A^{ut})$) corresponds to a group Gq of states of A^{ut} closed under the event tick. The group Gq may be composed by several sequences Sq_i of states. Each Sq_i contains a state, called first state, without ingoing tick, and all other states of Sq_i are reachable from this first state by executing a few ticks. To each Sq_i, we associate a timer Tq_i whose value is equal to zero in the first state of Sq_i. That is why several timers may be associated to each state of *TimeA*(A^{ut}).

If we consider the untimed A^{ut} of figure 6.b, *TimeA*(A^{ut}) is represented below. One timer T1 (resp. T2) is defined for state 1 (resp. state 2). *For the transition from state 1 to state 2* : Its enabling condition is $E1(t1)=(t1\leq5)$□$(t1>0)$, and its initialization is t2 ♦ F1(t1)=t1-1. *For the two transitions from state 2 to state 1* : Their enabling condition is True, and their initialization is t1 ♦ F2(t2)=0.


(*b* ;True; t 1 <-- F 2(t 2))
(*a* ; (t 1>0) ∧(t 1≤5); t 2 <-- F 1(t 1))
(*c* ;True; t 1 <-- F 2(t 2))
1
2

We will see in the next section, that the untiming and timing operations may be convenient to resolve a real problem, such as designing real-time protocols.

3. Deriving protocols specifications providing real-time services

Let's firstly give a table of the main notations in the present section.

RTDS	: Real-time distributed system
PE_i	: Protocol entity identified by number i
SAP_i	: Service access point associated to PE_i
SS^t	: Timed automaton (TA) specifying a desired sequential real-time service
$SupMed^t_{i,j}$	: TA specifying the supremal model of the medium for a pair (PE_i , PE_j)
PS^{ut}_i	: Untimed automaton (UA) specifying PE_i, which contributes for providing SS^t
$ReqMed^{ut}_{i,j}(q)$	: UA specifying timing requirements for the medium between PE_i and PE_j
SS^{ut} , $PrSS^{ut}$	: Two UA specifying respectively the desired and the provided sequential services
$SS[j]^t$, for j=1, 2	: Two sequential TA which compose a concurrent desired service
PE_c	is the protocol which makes choices in a distributed system

$PS_i^{ut}[j]$, for j=1,2	: Untimed automata specifying PE_i, which contributes for providing $SS^t[j]$, for j=1, 2

Table 2. Notations in section 3

3.1. Problem of the protocol derivation of real-time systems

In a real-time distributed system (RTDS, fig.7), n protocol entities (with n>1) communicate : **a)** with the user of the system through several service access points (SAP); **b)** with each other through a medium assumed reliable. To each SAP corresponds one protocol entity.

In the *user's viewpoint*, the RTDS is a black box where only interactions with the user are visible. These interactions correspond to the executions of service primitives (or simply primitives). Therefore, the specification of the service desired by (or provided to) the user defines the *ordering and timing requirements* between the executed primitives.

But in the *designer's viewpoint*, it is necessary to compute the specifications of the local real-time protocol entities PE_i, for i=1,2, ..., n, which may provide the service desired by the user. The designer must also compute timing requirements which must be respected by the medium. In order to avoid the computation of timing requirements impossible to respect by the medium, the designer may refer to a *supremal model* (def.27) of the medium, and compute only timing requirements which respect this supremal model. Informally, if for instance we know that the medium needs at least two units of clock time (uct) to carry messages between two protocol entities, this information is contained in the supremal model. In this case, the designer will not compute timing requirements such as : some message must be carried in one uct. We will see that the medium not only carries a message, but it also adds an information about the transit delay of the message in the medium.

The problem for desigining protocols is then : how can we derive systematically the different local protocol specifications and the timing requirements on the medium, from : **a)** a global specification of the service desired by the user ; **b)** a supremal model of the medium.

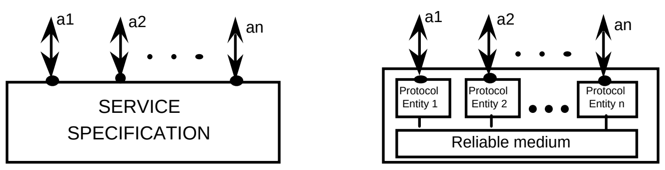


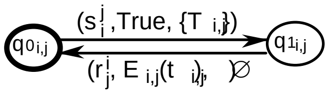
Figure 7. Service and protocol concepts

3.2. Approach of the problem of protocol derivation

The approach used for deriving protocols is *synthesis* ([BG86, KBK89, SP90, KHB92, KBD93]). Timing requirements are considered in [KBD93], but they are only between consecutive events, and the systems considered are sequential. In the present study, these two constraints are removed. For the sake of simplicity, we explain the basic principle of protocol derivation only for sequential systems. But parallel systems also are considered, farther in this paper (section 3.4). The principle is then : if a primitive A is executed by a protocol entity PE_a, and is followed by execution of a primitive B by PE_b, then after execution of A by PE_a, this one sends a message to PE_b to inform it that it may execute B. If after execution of A by PE_a, there is a choice between several primitives executed by different PE_{bi}, for i=1,2,..., p, then PE_a selects one PE_{bi} and sends a message to it to inform it that it may execute one of its primitives. Let's now formally define the supremal model of the medium, which is one of the two starting points of protocol derivation (sect.3.1).

Definition 27. (Supremal model of the medium)
Let's firstly remind that the supremal model of the medium is used in order to avoid the derivation of timing requirements impossible to respect by the medium (sect.3.1). The medium is assumed reliable and its supremal model is the following :
when PE_i sends a message m to PE_j, then the transit delay of m in the medium belongs to an interval $I_{i,j}=[t_{i,j}^{min} ; t_{i,j}^{max}]$, where $t_{i,j}^{min}$ and $t_{i,j}^{max}$ are constant integers such that $1 \leq t_{i,j}^{min} \leq t_{i,j}^{max} <$ □ . Therefore, we suppose that there is at least one *tick* (sect.2.7) during the transmission of a message. For each pair (PE_i,PE_j), this supremal model can be represented by a TA $Sup^{Med^t_{i,j}}$ (fig.8) defined below.
For each pair (PE_i,PE_j), the TA $Sup^{Med^t_{i,j}}$ (fig.8) is defined by $(Q_{i,j},V_{i,j},\{T_{i,j}\},\{V_{i,j}\},\delta_{i,j},q0_{i,j})$, where $Q_{i,j}=\{q0_{i,j},q1_{i,j}\}$, $V_{i,j}=\{s_i^j,r_j^i\}$, $\delta_{i,j}=\{[q0_{i,j},s_i^j,q1_{i,j},True,\{T_{i,j}\},True]$, $[q1_{i,j},r_j^i,q0_{i,j},E_{i,j}(t_{i,j})$,□$,True]\}$, with $E_{i,j}(t_{i,j})=(t_{i,j} > (t_{i,j}^{min} -1))$□$(t_{i,j} \leq t_{i,j}^{max})$.
$Sup^{Med^t_{i,j}}$ uses one timer $T_{i,j}$ (def.5) for defining timing requirements between s_i^j and r_j^i, where the event s_i^j means "PE_i sends a message to PE_j", and the event r_j^i means "PE_j receives a message coming from PE_i". A counter is not necessary, because timing requirements ensure the finiteness property (def.1) due to $(t_{i,j}>(t_{i,j}^{min} -1))$ in the T_Condition $E_{i,j}(t_{i,j})$. Therefore, the F_Condition is True.

■



$(s_i^j,True, \{T_{i,j}\})$
$q0_{i,j}$
$q1_{i,j}$
$(r_j^i, E_{i,j}(t_{i,j}), \emptyset)$

Figure 8. Supremal model $Sup^{Med^t_{i,j}}$ of the medium for a pair (PE_i,PE_j)

Remark 8 : timing requirements on $Sup^{Med^t_{i,j}}$ and $Sup^{Med^t_{j,i}}$ may be different.

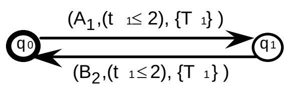
3.3. Protocol derivation for sequential real-time systems

3.3.1. Service specification

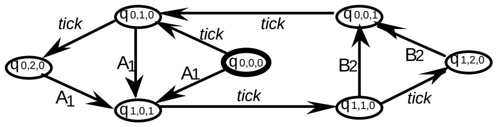
The desired service is, with the supremal model of the medium, one of the two starting points of the protocol derivation. It is described by a TA, noted SS^t and defined by $(Q,V,T,\{V\},\delta,q0)$, where V is the set of interactions with the user, and T is the set of timers (def.5) used for defining timing requirements between these interactions. Only one counter C , w.r.t. V (def.8), is used, and the finiteness property (def.1) is respected (property 1). Informally, no more than Mc service primitives are executed during one unit of the global clock time (uct). Each event of V is represented by A_i, where A is the name of the primitive executed, and i identifies the protocol which executes A.

Example 9 : Here is a very simple service specified by $SS^t=(Q,V,T,\{V\},\delta,q0)$ (fig.9.a), with $Q=\{q0,q1\}$, $V=\{A_1,B_2\},T=\{T_1\},\delta=[q0,A_1,q1,E(t_1),\{T_1\},K(c_1)],[q1,B_2,q0,E(t_1),\{T_1\},K(c_1)]\}$, with $E(t_1)=(t_1\leq 2)$, and $K(c_1)=(c_1<1)$. Informally, SS^t uses one timer T_1 and one counter C_1, and specifies that : **a)** events A_1 and B_2 are executed alternatively; **b)** at most one event occurs between two ticks (F_Condition $K(c_1)$);

c) at most two ticks occur between two consecutive events (T_Condition E(t_1)). SS^{ut}=*UntimeA*(SS^t) (def.26) is represented on figure 9.b, and its states are •q,t_1,c_1® where q is a state of SS^t.



9.a. SS^t



9.b. SS^{ut}

Figure 9. Timed and untimed sequential service specification

Let's notice that the timers and counters, used for specifying a desired service and for defining the supremal model of medium, are **fictitious**. For example, the desired service of figure 9.a just means that the user wants that there must be at most two uct between primitives A1 and B2. But the timers do not really exist. Therefore, a question such as, *how can we use a same timer in a distributed system*, has no sense. If the timers were real, there would be a problem of using a same timer in different sites.

Definition 28. (outgoing, ingoing, out(q), in(q), $outst_i$(q), nbrout(q))

Let SS^t be a TA specifying a timed sequential service, and let q be one of its states.

Outgoing (resp. *ingoing*) transitions of q are transitions which are executable from (resp. lead to) q.

out(q) (resp. in(q)) contains identifiers of SAP where outgoing (resp. ingoing) transitions of q occur.

$outst_i$(q) is the set of states of SS^t reachable from q by transitions executed by PE_i.

nbrout(q) is the number of transitions executable from q.

■

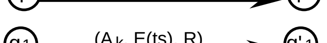
Example 10: for SS^t of Example 9 (figure 9.a), in(q0)={2}, in(q1)={1}, out(q0)={1},out(q1)={2}, $oust_1$(q0)={q1}, $oust_2$(q0)=□, $oust_1$(q1)=□, $oust_2$(q1)={q0}, nbrout(q0)=nbrout(q1)=1. ■

Since the starting points of the protocol derivation, i.e., the supremal model of the medium and the specification of the desired service, have been defined, we can propose a systematic method for deriving the specifications of : **a)** the local protocol entities; **b)** the necessary timing requirements on the medium, (sect. 3.1). These derived specifications are first untimed automata (def.26), and then are timed by using operator *TimeA* (sect.2.10).

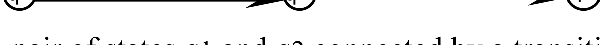
3.3.2. Transformation of the service specification

The first thing to do is to transform SS^t into another timed automaton TSS^t (Transformed SS^t) with the following rules.

First step : each timed transition of SS^t:



is replaced by :



A new state q'1 is then inserted between each pair of states q1 and q2 connected by a transition.

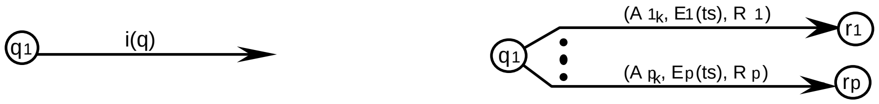
q1 and q'1 are connected by an internal transition i(q2) parameterized by q2.

After this first step, we obtain a TA noted TS^t. Let's notice that if a state of TS^t is reachable by an internal transition i(q), then its outgoing transitions are not internal.

Second step : The specification TS^t is transformed into an equivalent (def.14) TSS^t, such that every state q1 of TSS^t respects either condition C1 or condition C2, defined below.

C1 = only an internal transition i(q) is executable from q1 (fig. 10.a),

C2 = no internal transition is executable from q1, and all outgoing transitions (def.28) of q1 are executable by a same protocol entity, i.e., cardinal of out(q1) is equal to one (|out(q1)|=1) (fig.10.b). On figure 10.b., out(q1)={k} and $outst_k(q1)$={r1, ... , rp}.



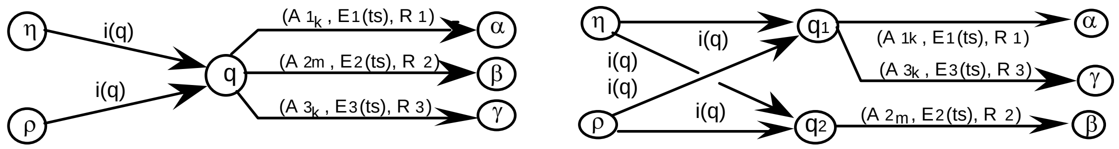
10.a. internal outgoing transition **10.b.** non internal outgoing transitions

Figure 10. Outgoing transitions in a state of the transformed specification TSS^t.

The way for obtaining TSS^t from TS^t is the following. Every state q of TS^t reachable by internal transition(s) (fig.11.a), is replaced by as many states qi as the cardinal of out(q) (fig.11.b) . Outgoing transitions of states qi (which are not internal) must respect the preceding condition C2, and the following condition C3. Ingoing transitions of states qi must respect the following condition C4.

C3 : Outgoing transitions of two different states qi and qj of TSS^t (fig.11.b), generated from a same state q of TS^t (fig.11.a), are executed by two different protocol entities.

C4 : The sets of ingoing transitions (which are internal) of two different states qi and qj of TSS^t, generated from a same state q of TS^t, are equal to the set of ingoing transitions of state q (fig.11).



11.a. State e in TS^t **11.b.** Transformation of e in TSS^t

Figure 11. Example of transformation from TS^t to TSS^t

Remark 9 : a) if two states r1 and r2 of TSS^t are connected by a transition i(q) then |in(r1)|=|out(r2)|=1; **b)** if TSS^t□TS^t, then TSS^t is non deterministic; **c)** if for every state q of SS^t, |ou(q)|=1, then TSS^t=TS^t.

Definition 29. (Operator *Transf*)

Operator *Transf* is simply defined by : TSS^t=*Transf* (SS^t). ■

Example 11 : SS^t of example 9 (fig.9.a.) is transformed into TSS^t of figure 12. In this example, only the first step of the transformation is used, because |ou(q0)|=|ou(q1)|=1 (remark 9.c).

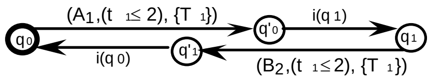


Figure 12. Transformation of SS^t of figure 9.a.

3.3.3. Procedure of protocole derivation for a sequential system

Considering a TA SS^t (sect.3.3.1) , and a TA $Sup^{Med^t_{i,j}}$ (def.27) for each pair (PE_i,PE_j),the proposed procedure of protocol derivation, is called *Der_Seq_Prot* and consists of eight steps.

Step 1 : SS^t is transformed into TSS^t, i.e., TSS^t=*Transf* (SS^t) (sect.3.3.2, def.29).

Step 2 : From TSS^t and the different $Sup^{Med^t_{i,j}}$, we generate $MedSS^t_\varepsilon$ with the following rules :

- A not internal transition remains unchanged.

- An internal transition i(q) $q1 \xrightarrow{i(q)} q2$ is replaced by :

Case a : if in(q1)=out(q2) (def.28), the transition becomes : $q1 \xrightarrow{\varepsilon} q2$

Case b : if in(q1)={i}≠out(q2)={j}, the transition becomes: $q1 \xrightarrow{(s_i^j(q),True,\{T_{i,j}\})} \bigcirc \xrightarrow{(r_j^i(q),\ E_{i,j}(t_{i,j})\ )\ \varnothing} q2$

This transformation uses $Sup^{Med^t_{i,j}}$ (def.27), but with s_i^j and r_j^i, (def. 27) parameterized by q.
Informally, i(q) consists in : **a)** doing nothing, if it connects two consecutive transitions of SS^t executed by a same PE_i ; **b)** sending a message from PE_i to PE_j, if it connects two consecutive transitions of SS^t respectively executed by PE_i and PE_j. The message is parameterized by q.

Step 3 : Transitions ε of $MedSS^t_\varepsilon$ are removed by projection for obtaining $MedSS^t$. An algorithm for removing these ε is proposed in [BC79].

Step 4 : $MedSS^t$ is untimed (def.26) for obtaining $MedSS^{ut}$=*UntimeA*($MedSS^t$). $MedSS^{ut}$ is a minimal FSM containing the event *tick*. Let's notice that the three remaining steps process FSMs with event *tick* .

Step 5 : we generate an untimed automaton GPS^{ut} (global protocol specification), by adding a second parameter to each event $s_i^j(q)$ or $r_j^i(q)$ in $MedSS^{ut}$, with the following rule :
A transition $q1 \xrightarrow{s_i^j(q)} q2$ is replaced by a transition $q1 \xrightarrow{s_i^j(q,q2)} q2$. The same transformation is made on transitions $r_j^i(q)$. This transformation allows to differentiate two transitions $s_i^j(q)$ (or $r_j^i(q)$) which do not lead to the same state in $MedSS^{ut}$.
Informally, a message is sent from a PE_i with two parameters q and q2 (event $s_i^j(q,q2)$), and may be received by a PE_j with a different second parameter q'2 (event $r_j^i(q,q'2)$). This means that the medium not only carries messages, but it also modifies their second parameters. This modification informs the receiving protocol entity about the transit delay of the message in the medium.

Step 6 : For each PE_i, the untimed automaton PS_i^{ut} is derived by projecting GPS^{ut} in the alphabet V_i≈{*tick* }, where V_i contains all events in GPS^{ut} executed by PE_i. An event of V_i may correspond to : **a)** execution of a primitive by PE_i; **b)** an event $s_i^j(q,q2)$; **c)** an event $r_i^k(q,q2)$, with j,k □ i.

Step 7 : For each pair (PE_i, PE_j) and each q, where PE_i sends to PE_j a message whose first parameter is q (i.e., events $s_i^j(q,*)$ and $r_j^i(q,*)$ exist in GPS^{ut}), the untimed automaton $ReqMed^{ut}_{i,j}(q)$ is generated by projecting GPS^{ut} in the alphabet $V_{i,j}(q)$≈{*tick* }. An element of $V_{i,j}(q)$ may be any event $s_i^j(q,*)$ and $r_j^i(q,*)$ of GPS^{ut}. The obtained $ReqMed^{ut}_{i,j}(q)$ specifies the behaviour of the medium when it carries, from PE_i to PE_j, a message whose first parameter is q.

The informal semantics of the different PS_i^{ut}(step 6) and $ReqMed^{ut}_{i,j}(q)$ (step 7) is the following. Let n be the number of protocol entities PE_i, for i=1,..., n. If PE_i are specified by PS_i^{ut}, and if the medium respects the specifications $ReqMed^{ut}_{i,j}(q)$, then the service SS^t is totally or partially provided (def.30 and 31).

Step 8 : The untimed specifications PS_i^{ut} and $ReqMed^{ut}_{i,j}(q)$ obtained at steps 6 and 7 are timed, by using the operator *TimeA* (sect.2.10). **End of** ***Der_Seq_Prot***
■

Remark 10: a) In steps 1, 2, 3, SS^{t}, TSS^{t}, $MedSS^{t}_{\varepsilon}$, and $MedSS^{t}$ use the same counter C_1, w.r.t. the alphabet V of SS^{t}; **b)** $MedSS^{t}_{\varepsilon}$ and $MedSS^{t}$ use timers of SS^{t} and a timer $T_{i,j}$ for each pair (PE_i,PE_j) where PE_i sends a message to PE_j.

Definition 30. (Provided service $PrSS^{ut}$)
For obtaining an untimed automaton (with event *tick*) specifying the service provided to the user, and noted $PrSS^{ut}$, one only has to project $MedSS^{ut}$ (step 4) in $V \approx \{tick\}$, where V is the alphabet of SS^{t}. Informally, this projection consists in keeping visible, in sequences accepted by $MedSS^{ut}$, only events of SS^{ut}.

■

Definition 31. (Service totally or partially provided)
Let SS^{ut} and $PrSS^{ut}$ be untimed automata specifying respectively the desired and the provided service.
The service is said *totally provided* if and only if : $SS^{ut} \cong PrSS^{ut}$, i.e., $\mathcal{L}_{PrSS^{ut}} = \mathcal{L}_{SS^{ut}}$,
The service is said *partially provided* if and only if : $SS^{ut} < PrSS^{ut}$, i.e., $\mathcal{L}_{PrSS^{ut}} \wp \mathcal{L}_{SS^{ut}}$.

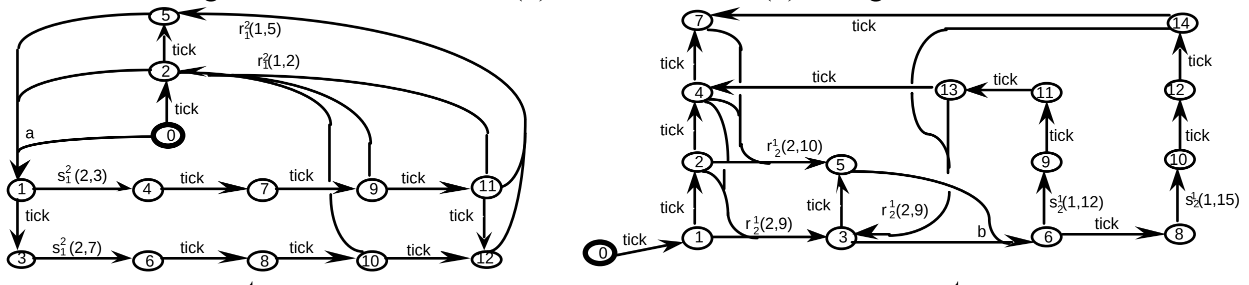
■

Theorem 6. If SS^{t} specifies a desired service, let SS^{ut}=*UntimeA*(SS^{t}) (def.26), and let $PrSS^{ut}$ be the specification of the provided service (def.30). Then : $PrSS^{ut} \leq SS^{ut}$ (i.e., $\mathcal{L}_{PrSS^{ut}} \prod \mathcal{L}_{SS^{ut}}$).
The safety is then ensured. (**Proof :** *See Appendix A* **.**

■

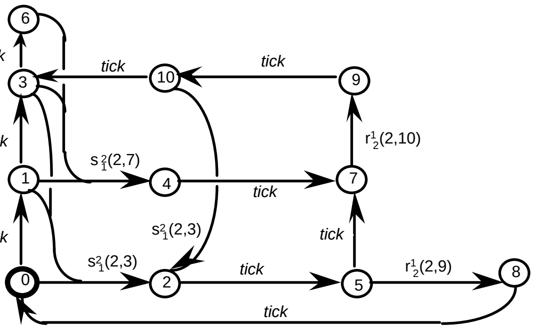
3.3.4. Example

We consider the SS^{t} of example 9 (fig.9.a), with q_0=1 and q_1=2. The supremal model of the medium is defined by $Sup^{Med^{t}_{1,2}}$ and $Sup^{Med^{t}_{2,1}}$ (def.27, fig.8), with $t^{min}_{1,2} = t^{min}_{2,1} = 1$ and $t^{max}_{1,2} = t^{max}_{2,1} = 2$. Informally, during the transmission of a message, one or two ticks of the global clock may occur.
By using the procedure *Der_Seq_Prot* (sect.3.3.3), we obtain, after the seventh step, the specifications PS^{ut}_1 and PS^{ut}_2 of figure 13, and $ReqMed^{ut}_{1,2}(2)$ and $ReqMed^{ut}_{2,1}(1)$ of figure 14.

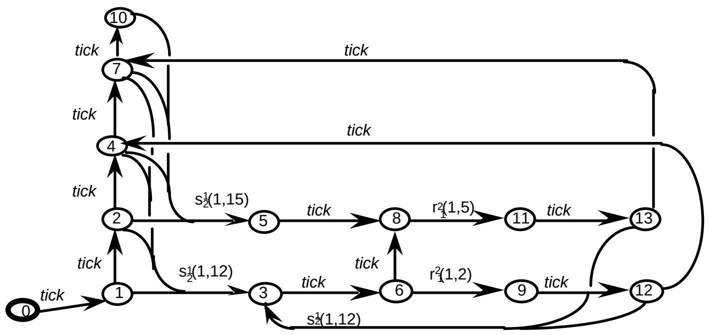


13.a. PS^{ut}_1 **13.b.** PS^{ut}_2

Figure 13. Obtained untimed protocol



14.a. $\mathrm{Req}^{\mathrm{Med}^{ut}_{1,2}(2)}$



14.b. $\mathrm{Req}^{\mathrm{Med}^{ut}_{2,1}(1)}$

Figure 14. Obtained untimed medium

Partial interpretation of the obtained specifications :

a) On PS_1^{ut} : From state 0, PE$_1$ executes the event *a* after 0, 1 or 2 ticks. Then it sends to PE$_2$ a message parameterized by 2 (by $s_1^2(2, *)$). A second parameter is added to the message. If the latter is sent 1 tick after execution of event *a* (from state 3), then the second parameter is 7 (by $s_1^2(2,7)$). Afterwards, PE$_1$ can receive from PE$_2$ a message parameterized by 1 (by $r_1^2(1,*)$). If the latter is received two ticks after $s_1^2(2,7)$ (in state 10), then the second parameter is 2 (by $r_1^2(1,2)$). In this case, PE$_1$ can execute the event *a* immediately (from state 2) or after one tick (from state 5)...

b) On PS_2^{ut} : From state 0, PE$_2$ may receive a message parameterized by 2 (by $r_2^1(2,*)$) after 1,2,3 or 4 ticks. The message contains a second parameter. If, for example, the message is received after 4 ticks (in state 7), the second parameter is 10 (by $r_2^1(2,10)$). In this case, PE$_2$ must execute the event *b* immediately (from state 5 to state 6). Afterwards, PE$_2$ sends to PE$_1$ a message parameterized by 1 (by $s_2^1(1,*)$). If the latter is sent one tick after execution of event *b* (from state 8), then the second parameter of the message is 15 (by $s_2^1(1,15)$) ...

c) On $Med^{ut}_{1,2}(2)$: From state 0, the medium may send, from PE$_1$ to PE$_2$, a message parameterized by 2 (by $s_1^2(2,*)$). If, for example, the message is sent after 3 ticks, the second parameter of the message may be 7 (by $s_1^2(2,7)$ from state 6 to state 4). In this case, if the message reaches its destination (i.e., PE$_2$) after one tick, the medium changes the second parameter from 7 to 10 (by $r_2^1(2,10)$ from state 7) ...

d) On $Med^{ut}_{2,1}(1)$: From state 0, the medium may send, from PE$_2$ to PE$_1$, a message parameterized by 1 (by $s_2^1(1,*)$). If, for example, the message is sent after 2 ticks, the second parameter of the message may be 12 (by $s_2^1(1,12)$ from state 2 to state 3). In this case, if the message reaches its destination (i.e., PE$_1$) after two ticks the medium changes the second parameter from 12 to 5 (by $r_1^2(1,5)$ from state 8) ...

By using the operator *TimeA* (sect. 2.10), we obtain the specifications detailed below and represented on figure 15. In the description below, EC means Enabling condition, and In means Initialization.

a) *TimeA*(PS_1^{ut}) : Timers T1$_1$ and T1$_2$ are respectively associated to states 1 and 2.
Timers T13$_1$ and T13$_2$ are associated to state3.

Transition Tr1$_1$: Event *a*; EC : t1$_1$≤2; -- In : t1$_2$ ♦ 0.

Transition Tr1$_2$: Event $s_1^2(2,3)$; EC : (t1$_2$=0); -- In : t13$_1$ ♦ 0.

Transition Tr1$_3$: Event $s_1^2(2,7)$; EC : (t1$_2$=1); -- In : t13$_2$ ♦ 0.

Transition Tr14 : Event $r^2_1(1,2)$; ECs : $(t13_1>1)$□$(t13_1\le3)$, $(t13_2=1)$; -------------- In : t11 ♦ 1.

Transition Tr15 : Event $r^2_1(1,5)$; EC: $(t13_1>2)$□$(t13_1\le4)$, $(t13_2=3)$; ---------------- In : t11 ♦ 2.

b) *TimeA*(PS^{ut}_2) : Timers $T21_1$, $T21_2$ and $T21_3$ are associated to state 1.

Timers T22 and T23 are respectively associated to states 2 and 3.

Transition Tr21 : Event $r^1_2(2,9)$; ECs : $(t21_1>0)$□$(t21_1\le3)$, $(t21_2>1)$□$(t21_2\le3)$, $(t21_3=2)$; In: t12 ♦ 0.

Transition Tr22 : Event $r^1_2(2,10)$; ECs : $(t21_1>1)$□$(t21_1\le4)$, $(t21_2>1)$□$(t21_2\le4)$, $(t21_3=3)$; In: t12 ♦ 0.

Transition Tr23 : Event *b*; EC : $(t22\le1)$; --- In : t23 ♦ 0.

Transition Tr24 : Event $s^1_2(1,12)$; EC : $(t23=0)$; --- In : $t21_2$♦ 0.

Transition Tr25 : Event $s^1_2(1,15)$; EC : $(t23=1)$; --- In : $t21_3$♦ 0.

c) *TimeA*($Req^{Med^{ut}_{1,2}(2)}$) : Timers $T31_1$ and $T31_2$ are associated to state 1.

Timers $T32_1$ and $T32_2$ are associated to state 2.

Transition Tr31 : Event $s^2_1(2,3)$; ECs : $(t31_1>0)$□$(t31_1\le3)$, $(t31_2>0)$□$(t31_2\le2)$; In $t32_1$ ♦ 0.

Transition Tr32 : Event $s^2_1(2,7)$; ECs : $(t31_1>1)$□$(t31_1\le4)$, $(t31_2>1)$□$(t31_2\le3)$; In $t32_2$ ♦ 0.

Transition Tr33 : Event $r^1_2(2,9)$; EC : $(t32_1=1)$; ------------------------------------- In : $t31_1$ ♦ 0.

Transition Tr34 : Event $r^1_2(2,10)$; ECs : $(t32_1=2)$, $(t32_2=1)$; ------------------------- In : $t31_2$ ♦ 0.

d) *TimeA*($Req^{Med^{ut}_{2,1}(1)}$) : Timers $T41_1$, $T41_2$ and $T41_3$ are associated to state 1.

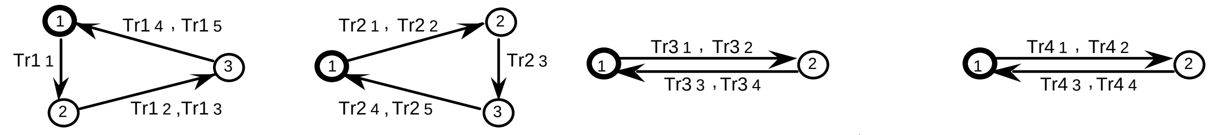
Timers $T42_1$ and $T42_2$ are associated to state 2.

Transition Tr41 : Event $s^1_2(1,12)$; ECs : $(t41_1>0)$□$(t41_1\le4)$, $(t41_2>0)$□$(t41_2\le3)$; $(t41_3>0)$□$(t41_3\le2)$;
In : $t42_1$ ♦ 0.

Transition Tr42 : Event $s^1_2(1,15)$; ECs : $(t41_1>0)$□$(t41_1\le5)$, $(t41_2>1)$□$(t41_2\le4)$; $(t41_3>1)$□$(t41_3\le3)$;
In : $t42_2$ ♦ 0.

Transition Tr43 : Event $r^2_1(1,2)$; EC : $(t42_1=1)$; --- In : $t41_2$ ♦ 0.

Transition Tr44 : Event $r^2_1(1,5)$; ECs : $(t42_1=2)$, $(t42_2=1)$; ----------------------------------- In : $t41_3$ ♦ 0.



15.a. *TimeA*(PS^{ut}_1) **15.b.** *TimeA*(PS^{ut}_2) **15.c.** *TimeA*($Req^{Med^{ut}_{1,2}(2)}$) **15.d.** *TimeA*($Req^{Med^{ut}_{2,1}(1)}$)

Figure 15. Obtained timed specifications

3.4. Protocol derivation for parallel and concurrent real-time systems

3.4.1. Introduction

For the sake of simplicity, we only consider a parallel system composed by only two sequential systems. A desired parallel service is then specified by two TA (def.12) $SS^t[i]$ over alphabets V[i], for i=1,2. Each $SS^t[i]$ specifies a sequential desired service. Let's consider three cases :

a) V[1]∏V[2] **:** $SS^t=SS^t[1]$□$SS^t[2]$ (def.21) is a sequential service (remark 3.b), and we may use the procedure *Der_Seq_Prot* (sect.3.3.3) for deriving the protocol providing the service specified by SS^t.

b) V[i]□□ and V[i]↔V[j]=□, for i,j=1, 2, and i□j **:** $SS^t[1]$ and $SS^t[2]$ are independent and compose a parallel system (def.23). We may process each sequential service separately, i.e., for each $SS^t[i]$, we use *Der_Seq_Prot* for deriving the sequential protocol which provide $SS^t[i]$.

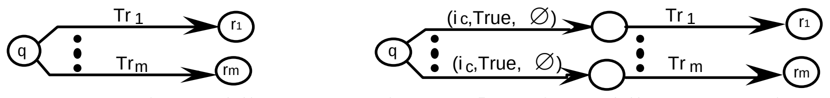
c) V[i]-V[j]□□ and V[i]↔V[j]□□, for i,j=1, 2, and i□j : $SS^t[1]$ and $SS^t[2]$ are dependent and compose a concurrent system (def.23). This case is studied in detail in the present section 3.4.

3.4.2. Solution for the problem of the choice

In a concurrent system, we think that one of the main problems consists in avoiding possible deadlocks. For that, we use the following approach, already proposed in [KHB92] :

During the running of a concurrent distributed system, when a choice is possible between execution of different transitions, this choice is made locally by a same protocol entity.

Let PE_i, for i=1, ... , n, be the n protocol entities which execute the transitions of $SS^t[1]$ and $SS^t[2]$. For the sake of simplicity, we suppose that all choices are made by a same protocol entity PE_c, with c>n. In other words, if there is a choice to make, the protocol entities PE_i, for i=1, ... ,n, "pass the buck" to PE_c. Such constraint seems too restrictive, and we intend to weaken it in a next version. To enforce explicitly this choice, we must add to $SS^t[1]$ and $SS^t[2]$, some timed events (def.10) noted (i_c,True,□), where i_c is executed by PE_c. These timed events are added as follows : for each state q of $SS^t[i]$, for i=1, 2, where nbrout(q) >1 (def.28), its ougoing transitions Tr_1, ..., Tr_m represented in figure 16.a. are replaced by the structure of figure 16.b. The obtained specifications are noted $SS^t_c[1]$ and $SS^t_c[2]$. Let's now propose a procedure of protocol derivation for concurrent systems.



16.a. Before adding events i_c **16.b.** After adding events i_c

Figure 16. Adding events i_c

3.4.3. Procedure of protocol derivation for a concurrent system

Let two TA $SS^t[i]$ over alphabets V[i], for i=1,2, and a TA $SupMed^t_{u,v}$ for each pair (PE_u,PE_v), the procedure of protocol derivation for concurrent systems, called *Der_Conc_Prot*, consists of nine steps.

Step one : $SS^t[i]$ are modified into $SS^t_c[i]$, for i=1,2, (sect.3.4.2.). Besides, any two states of respectively $SS^t_c[1]$ and $SS^t_c[2]$ must be identified differently. This is necessary for not confusing exchanged messages, which are parameterized by identifiers of states (see *Der_Seq_Prot* in sect.3.3.3).

Step two : Steps 1 to 5 of *Der_Seq_Prot* are applied to each $SS^t_c[i]$ for obtaining $GPS^{ut}_c[i]$, for i=1,2, but with the following difference. At the third step of *Der_Seq_Prot* , not only transitions ε , but also transitions (i_c,True,□), are removed. Let $V_g[i]\approx\{tick\}$ be the alphabet of $GPS^{ut}_c[i]$, then $V[i]\prod V_g[i]$.

Step three : The synchronized product $GPS^{ut}_c=GPS^{ut}_c[i]\infty GPS^{ut}_c[i]$ is computed (remark 6.b).

Step four : Indesirable states are removed from GPS^{ut}_c for obtaining GPS^{ut}. A state is indesirable if it is either a deadlock or only a selfloop *tick* is executable from it (remark 5). For removing indesirable states, we may use a fixpoint method similar to the one used in the control theory for computing controllable languages ([WR87,KBD94]).

Step five : The protocol specification PS^{ut}_c of PE_c (sect.3.4.2) is obtained by projecting GPS^{ut} in alphabet $V_c\approx\{tick\}$. V_c contains all events of GPS^{ut} executed by PE_c, and these events are of the form $s^*_c(*,*)$ and $r^*_c(*,*)$ (see def.26, and step two of *Der_Seq_Prot*), where * may be any parameter.

Step six : the sequential $GPS^{ut}[i]$ are obtained by projecting GPS^{ut} in alphabets $V_g[i] \approx \{tick\}$ of $GPS_c^{ut}[i]$, (step 2), for i=1, 2. The sequential processes specified by $GPS_c^{ut}[i]$, for i=1,2, interact with PE_c specified by PS_c^{ut} and do not lead to an indesirable state.

Step seven : For each $GPS^{ut}[i]$ (for i=1,2), we apply **step 6** of *Der_Seq_Prot* for obtaining the untimed automata (UA) $PS_j^{ut}[i]$ corresponding to PE_j (j=1, ... ,n).

Step eight : For each $GPS^{ut}[i]$ (for i=1,2), we apply **step 7** of *Der_Seq_Prot* for obtaining the UA $ReqMed_{j,k}^{ut}(q)$. Each $ReqMed_{j,k}^{ut}(q)$ depends implicitly on i, because q identifies a state of $SS_c^t[i]$, and states of $SS_c^t[1]$ and $SS_c^t[2]$ are identified differently (see step one).

The informal semantics of PS_c^{ut} (step 5), of $PS_j^{ut}[i]$ (step 7), and of $ReqMed_{j,k}^{ut}(q)$ (step 8) is the following. If each PE_j, for j=1, ... ,n, is a parallel system specified by two $PS_j^{ut}[i]$, for i=1,2, and if the medium respects the specifications $ReqMed_{i,j}^{ut}(q)$, then the desired concurrent service specified by $SS^t[1]$ and $SS^t[2]$ (step one), is totally or partially provided by the help of PE_c specified by PS_c^{ut}(step 5).

Step nine : The untimed specifications obtained at steps 5, 7 and 8 are timed, by using the operator *TimeA* (sect.2.10). **End of *Der_Conc_Prot***

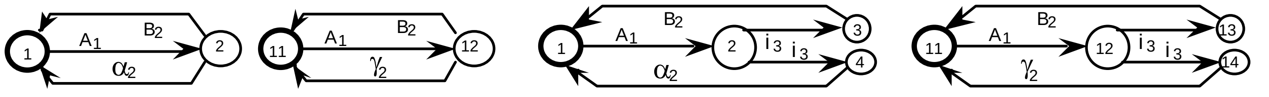
■

3.4.4. Example

Since the problem of concurrency exists even for systems without timing requirements, let's give an example for such systems. In this case, the T_Conditions (def.7) of timed transitions (def.11) are True, and their Resets are □. The untiming operation (def.26) consists just in adding a selfloop *tick* to every state. For these reasons : **a)** timed events (A_i,True,□) are represented just by A_i ; **b)** event *tick* is not represented, therefore A^t and A^{ut}=*UntimeA*(A^t) are not differentiated, and are refered to by A;
c) the messages exchanged contain only the first parameter. The second parameter which implicitly contains only temporal informations, is not necessary.

Then : a) Step 2 of *Der_Conc_Prot* is only composed by steps one to three of *Der_Seq_Prot* .
b) Step 4 of *Der_Conc_Prot* just consists in removing deadlocks.
c) Steps 8 and 9 of *Der_Conc_Prot* are not necessary.

The desired concurrent service is represented on figures 17.a and 17.b, and is specified by SS[1] and SS[2] respectively over alphabets $V[1]=\{A_1, B_2, \alpha_2\}$ and $V[2]=\{A_1, B_2, \gamma_2\}$. SS[1] and SS[2] are then synchronized on A_1 and B_2. After the first step, we obtain $SS_c[1]$ and $SS_c[2]$ on figures 17.c. and 17.d.



17.a. SS[1] **17.b.** SS[2] **17.c.** $SS_c[1]$ **17.d.** $SS_c[2]$

Figure 17. Example of concurrent desired service without timing requirements

If we apply *Der_Conc_Prot* , we obtain :

a) at step 5, the specification PS_c of PE_c, with c=3, is represented on figure 18.

b) at step 7, the specifications $PS_1[1]$, $PS_2[1]$, $PS_1[2]$ and $PS_2[2]$, are represented on figure 19.

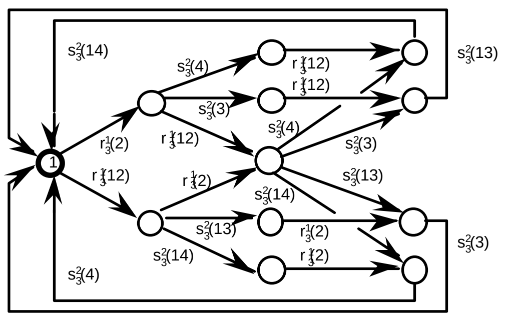


Figure 18. Specification of PE_c

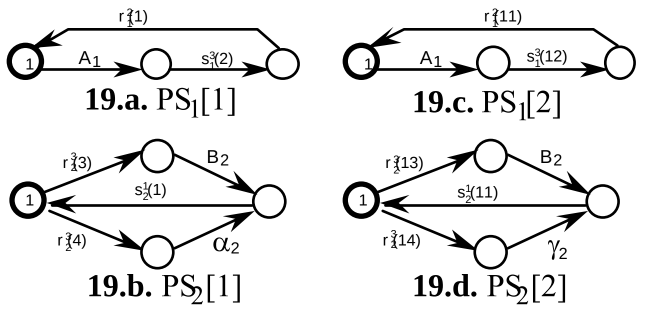


Figure 19. Specifications of the protocol entities

4. Conclusion

In this paper, we present a model we have developed for specifying real-time discrete event systems. An application of the model for designing real-time protocols is also proposed. The synthesis approach used for deriving a real-time protocol providing a desired service is inspired by other works, but our main contribution has been to consider *real-time* systems, i.e., systems containing timing requirements. We conclude this study by making an informal and succint comparison between our model for specifying DES, and the two models which have mainly inspired us. A few extensions are also proposed.

First model ([Os90, BW92, OW90]) : For defining a TA A^t, a global clock and a set of timers are used. For each transition Tr_i of A^t corresponds one timer T_i. This timer is reset only when a state q1, from which Tr_i is executable, is reached. This same timer is not used in another state q2□q1. The enabling condition of Tr_i is that the value of T_i must belong to an interval. We think that our model is more general because (in our model): **a)** the enabling condition of a transition may depend on several timers; **b)** a same timer may be used in the enabling conditions of several transitions; **c)** a timer may be reset at the occurrence of any transition; **d)** the finiteness property may be ensured by using counters;
e) the operator *TimeA* is defined.

Second model ([AD90, TH92]) : For defining a timed automata, a dense time and a set of clocks are used. These clocks are used as we use the timers in our model, but the semantics is quite different, because the clocks are not synchronized. Besides : **a)** Our operators *UntimeA* and *TimeA* are different than operators *Untime* and *Time* proposed in [TH92]. In fact in [TH92], from a TA A^t, the operator *Untime* is used for obtaining a UA A^{ut}. Some processing is then made on A^{ut} for obtaining a UA B^{ut}. And then the operator *Time* can be applied on B^{ut} only if the processing for obtaining B^{ut} from A^{ut} makes no projection, i.e., A^{ut} and B^{ut} have a same alphabet; **b)** The finiteness property is supposed respected in [TH92], but it is not ensured. Another advantage with our model is then that we can ensure the finiteness property. **c)** In [TH92] the composition is defined only when the two TA have a same alphabet, and in [AD90], authors only specify how events executed conjointly by the two composed systems are processed. Our limitation is that the time is discrete.

Extensions : The exponential complexity in the number of timers imposes to investigate how to choose classes of systems which avoid this computational blow-up. The simplest but also the most restrictive class contains systems respecting the following condition : timing requirements are only between consecutive events. With such systems, only one timer is necessary, therefore the complexity becomes polynomial. In the presented model, the enabling conditions use only operators $\leq$, > and = for defining canonical boolean functions. We are also investigating how to extend this model by using arithmetic operators + and - in the canonical boolean functions. For instance, a canonical boolean function can be $t_1+t_2-t_3 \leq k$. We are also investigating how we can modify systematically several existing protocol entities, which provide an old service, for providing a new desired service. For that, we intend to use control theory of the discrete event systems .

REFERENCES

[AD90] R. ALur and D.Dill, "Automata for Modeling Real-Time Systems." In Lecture Notes in Computer Science 443, editor Proceedings of the 17th Intern. Coll. on Automata, Languages and Programming, Warwick, UK, 1990. Springer-Verlag.

[BC79] W.A.Barrett and J.D.Couch,"Compiler Construction:Theory and Practice" Ed.:Science Research Associates, Inc. 1979

[BG86] G.v. Bochmann and R. Gotzhein, "Deriving protocols specifications from service specifications." Proceedings du Symposium ACM SIGCOM' 86, Vermont, USA, pp.148-156, 1986.

[BW92] B. Brandin and W.M. Wonham, "The supervisory Control of Timed Discrete-Event Systems." Proceedings of the 31rst Conf. on Decision and Control, Tucson, Arizona, Dec.92.

[Di89] D. Dill, "Timing assumptions and Verifications of Finite-State Concurrent Systems.", In Lecture Notes in Computer Sciences 407, editor Automatic Verification Methods For Finite State Systems, Intern. Workshop, pp.197-212, Grenoble France, 1989. Springer-Verlag.

[KBD93] A. Khoumsi, G.v. Bochmann and R. Dssouli, "Dérivation de spécifications de protocoles à partir de spécifications de services avec contraintes temporelles." Colloque Francophone pour l'ingénierie des protocoles (CFIP), Montreal, September 1993.

[KBD94] A. Khoumsi, G.v. Bochmann and R. Dssouli, "Contrôle et extension des systèmes à événements discrets totalement et partiellement observables." Third Maghrebian Conference on Software Engineering and Artificial Intelligence, Rabat, April 1994.

[KBK89] F. Khendek, G.v. Bochmann and C. Kant, "New results on deriving protocol specifications from services specifications", Proceedings of the ACM SIGCOMM'89, pp.136-145, 1989.

[KHB92] C. Kant, T. Higashino and G.v. Bochmann, "Deriving protocol specifications from service specifications written in LOTOS." Rapport interne No 805, Département d'Informatique et de Recherche Opérationnelle. Faculté des arts et des sciences, Université de Montréal, January 1992.

[Os90] J.S. Ostroff, "Deciding Properties of Timed Transitions Models." IEEE Transactions on Parallel and Distributed Systems, Vol.1, No.2, pp.170-183, April 1990.

[OW90] J.S. Ostroff and W.M. Wonham, "A framework for real-time discrete event control." IEEE Transactions on Automatic Control, Vol.35, No.4, pp.386-397, April 1990.

[SP90] K.Saleh and R. Probert, "A service-based method for the synthesis of Communications protocols." International Journal of Mini and Microcomputers, Vol.12, No 3, 1990.

[TH92] H.W-Toi and Gérard Hoffmann, "The Control of Dense Real-Time Discrete Event Systems." Report submitted to IEEE Transactions on Automatic Control, February 1992.

[WR87] W.M. Wonham and P.J. Ramadge, "On the Supremal Controllable sublanguage of a Given Language." SIAM J.Control and Optimization, Vol.25, No.3, May 1987.