Protocol Synthesis for Real-Time Applications*

A. Khoumsi
Université de Sherbrooke, Département de génie électrique et de génie informatique
Sherbrooke(Quebec) Canada J1K 2R1 email : khoumsi@gel.usherb.ca

Gregor v. Bochmann
University of Ottawa, School of ITO, Ottawa, (Quebec), H3C 3J7

Rachida Dssouli
Université de Montréal, Département IRO, Montréal, (Quebec), H3C 3J7

February, 1999

ABSTRACT.

This paper deals with automatically deriving protocol specifications which provide a given service satisfying timing requirements. In previous work, we have developed an extension of a method proposed by Saleh and Probert, by considering timing requirements in a more general case than in other existing studies. In the present paper, we improve our method by the following modifications and additions. First, the number of messages exchanged between the protocol entities is minimized. Second, a less restrictive strategy for choosing between several possible service primitives is proposed, which allows that certain decisions are distributed among several sites. Third, we consider applications where the choice between several primitives of a single site can be made by the user, and not only by the system. Fourth, conditions of existence of solutions are weaker. Fifth the timing constraints of the synthesized protocols are weaker. Finally, two simple but concrete examples of applications are described.

KEYWORDS. Real-Time Protocol, Service, Synthesis, Dense Time, Assembly System, X.25 Protocol.

1. Introduction

Several methods for deriving a protocol specification from the specification of a desired service have been developed by various researchers [4,7,8,10-12,19,21]. These methods are not applicable for real-time applications, for which the correct ordering of service primitives alone does not always ensure the success of a task. In addition, certain delays must be respected between occurrences of services primitives. Timing constraints are dealt with in [10,11], but with restrictions. In fact, in [11] transit delays in the medium are supposed negligible, while in [12] they re bounded by a maximum value. In the present paper, our aim is to describe a systematic approach for deriving protocols which guarantee both : **(a)** a correct ordering for the execution of service primitives; and **(b)** the satisfaction of given timing requirements between the executions of service primitives, in a more general case than in [11,12].

An approach presented in [19], which guarantees only (a), has been previously extended in [13] for ensuring also (b). The timing requirements considered in [13] allow to specify certain constraints on the delays between consecutive service primitives. For instance, we can specify that the delay between a data transmission and its reception must be in an interval $[t_{min}, t_{max}]$. In the present paper, we consider the same problem but we provide a better solution. In fact, compared to [13] : **(a)** the number of messages exchanged between protocol entities for providing a desired service has been minimized; **(b)** the strategy for choosing between several possible service primitives is distributed among several sites, instead of being centralized in a single site; **(c)** we consider applications where the choice between several primitives of a single site can be made by the user, and not only by the system; **(d)** conditions of existence of solutions are weaker; **(e)** the temporal constraints to be respected by the protocol entities are weaker. We have also presented two concrete examples of application.

The remaining of this paper is organized as follows. In Sect. 2, we introduce the problem of protocol derivation and the principle used for deriving protocols. Sect. 3 deals with non-real-time systems. First, we show how services and protocols are specified, and then we introduce the method for deriving protocols. Sect. 4 to 6 deal with real-time systems. In Sect. 4, we describe how services and protocols are specified using timed automata. In Sect. 5, we explain the approach used for calculating temporal requirements for protocol entities from temporal requirements of a service to be provided. We also present some rules for deriving real-time protocols in a systematic way. In Sect. 6, three examples (one abstract and two concrete) illustrate our method. Finally, we conclude in Sect. 7.

2. Protocol synthesis

We consider a distributed system, denoted *DS*, consisting of several sites interconnected through a reliable communication medium, simply called *medium*. We assume that:

- Each pair of sites can communicate with each other through the medium;
- The environment can interact with the *DS* at the different sites through service access points (*SAP*).

These interactions correspond to the executions of service primitives (simply called *primitives*).

We may assume that to each site, identified by a number i and denoted $Site_i$, correspond a *protocol entity*, denoted PE_i. Intuitively, PE_i represents the local behaviour of the *DS* in $Site_i$.

In the *user's viewpoint*, the *DS* is a black box which provides a service where only executions of primitives are visible. We assume that the specification of the desired service (provided to the user) defines: **(a)** the *ordering* of the occurrences of primitives; **(b)** *the timing requirements* between the occurrences of primitives. The timing requirements define the real-time properties of the *DS*, i.e., the success of a task depends on the respect of certain delays.

The aim of the *designer* is then to derive specifications of the local protocol entities PE_i, for i=1,2, ..., n, from the specification of the desired service. In the case of a real-time system, the designer must also have a temporal model of the medium, which is assumed reliable. Intuitively, for generating protocol specifications, the designer must know: **(i)** what the user wants (the service specification), and **(ii)** bounds on the transit delays of messages in the medium (temporal model of the medium). The problem of protocol derivation is then: *How* can we derive systematically specifications of the local protocol entities (protocol specifications) which provide a given desired service ?

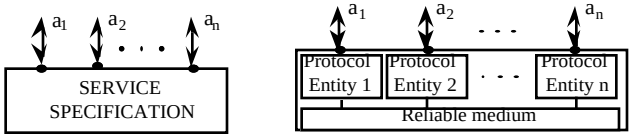


Fig. 1. Service and protocol concepts

The approach we have selected for deriving protocols is called *Synthesis*, and the systems considered are assumed sequential, i.e., if two events are consecutive then there is a causality relation between them. The basic principle we have used is the following: if in the specification of the desired service, a primitive A is executed by PE_a, and is followed by the execution of a primitive B by PE_b, then: **(α)** after PE_a executes A, it sends a message m to entity PE_b; **(β)** after PE_b receives message m, it executes B. We will see in the following sections how this principle is used for developing algorithms for synthesizing protocols. We note the terms "derivation" and "synthesis" are used as synonyms.

3. Protocol Synthesis for Non-Real-Time Applications

3.1. Service Specification

A service desired by the user is described by a finite state automaton (*FSA*), denoted SS, which specifies the sequences of service primitives the user would like to observe at the various *SAP*s. Every transition of SS (Fig. 2) is defined by $[q,E_a,r]$, where: **(1)** q and r are origin and destination states; and **(2)** E_a represents a primitive E executed by PE_a. Besides, every transition is identified by a number p and then denoted $T_p=[q,E_a,r]$. Henceforth, in a figure representing an *FSA*, any transition $T_p=[q,E_a,r]$ is simply denoted E_a, since q and r are explicitly represented by the transition diagram.

Definition 3.1. (*Incoming and outgoing transitions*).
An *outgoing* (resp. *incoming*) transition of a state q is a transition which *is executable from* (resp. *leads to*) q. □

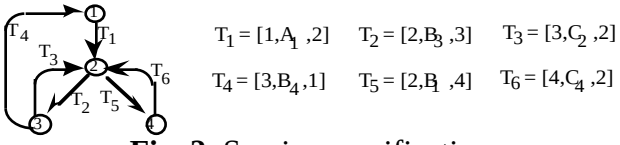


Fig. 2. Service specification

3.2. Protocol specification

A protocol entity PE_a is described by an *FSA*, denoted PS_a (see for example Fig. 5), which specifies the sequences of local events which occur at $Site_a$. There are three types of events in each PS_a.

Type P (for Primitive) : The execution of a service primitive E is denoted E_a.

Type S (for Send): The sending of a message is denoted $s_a^i(p)$, and means "message parameterized by p (i.e., with content p) is sent by PE_a to entity PE_i ".

Type R (for Receive): The reception of a message is denoted $r_a^i(p)$, and means "message parameterized by p and coming from PE_i is received by PE_a".

To clarify our syntax, we note that in SS and in any PS_a, every index specifies the site where the event is executed, and every exponent specifies the destination site for events of Type *S* and the sender site for events of Type *R*.

3.3. Correctness of a protocol

Let PE_1, PE_2,..., PE_n be *n* protocol entities, specified by PS_1, PS_2,..., PS_n respectively. Let $\mathcal{DS}$ be constituted by PE_1, PE_2,..., PE_n and by the medium, and specified by DS.

Definition 3.2. (*Combined behaviour of several protocol entities*)
The *combined behaviour* of PE_1, PE_2 , ..., PE_n is the behaviour of $\mathcal{DS}$. Intuitively, the specification of this behaviour, denoted DS, can be computed from PS_1, PS_2,..., PS_n by making a shuffled product of PS_1, PS_2, ..., PS_n, with the following constraint: in DS, the first event which follows $s_a^b(p)$ is $r_b^a(p)$, and reciprocally, the last event which precedes $r_b^a(p)$ is $s_a^b(p)$, for any a, b, p. We define this combined behaviour by the operator *Comb*: DS=*Comb*(PS_1, PS_2 , ..., PS_n). □

Definition 3.3. (*Projection, total and partial provision of a desired service*)
Let V_s and V_i be the alphabets of the service and PS_i, respectively. We also use the following concepts :

- $Proj_\Lambda(A)$ denotes the projection of an $\mathcal{FSA}$ A into an alphabet Λ. As an example, $Proj_{V_s}(DS)$ specifies the service provided to the user by $\mathcal{DS}$.
- A ≅ B means that the $\mathcal{FSA}$s A and B accept the same language (trace equivalence).
- A < B means that the language accepted by A is included in the one accepted by B.

We say that the service is totally (resp. partially) provided if $Proj_{V_s}(DS) \cong SS$ (resp. $Proj_{V_s}(DS) < SS$). □

Definition 3.4. (*Semantic and syntactic correctness*)
We say that the protocol is semantically correct if the desired service is totally provided. The protocol is syntactically correct if DS is deadlock-free and livelock-free and no unspecified reception error is possible (we assume that the desired service SS is deadlock-free and livelock-free). □

Our aim is therefore to propose a synthesis method which, from the specification of a desired service, generates specifications of protocol entities which are syntactically and semantically correct.

3.4. Principle for deriving protocol entities

From an SS specifying a desired service, deriving a protocol consists of generating as many $\mathcal{FSA}$s as there are sites. Each of these $\mathcal{FSA}$s is denoted PS_i and specifies the action sequences executed at $Site_i$. In order to provide the desired service, the different $\mathcal{PE}$s exchange messages through a reliable medium. The basic principle used for deriving a protocol is explained in the last paragraph of Sect. 2. This principle has been applied in [13] as follows. If the execution of a primitive A by PE_a is followed by a choice between primitives executed by other PE_{bi}s, for i=1, ...,k, (Fig. 3) then, after the execution of A, PE_a decides which transition should follow. It therefore sends a message m to all PE_{bi}s (i=1, ...,k, and bi≠a) which contains the following two parameters: **(1)** the identifier p of the executed transition T_p; **(2)** the identifier q of the chosen transition T_q to be executed next. All PE_{bi}s will receive the message m, but only one of them will execute the primitive corresponding to the selected transition T_q. The other PEs ignore the message. In the particular case where A is followed by a choice of primitives executed by the same protocol entity PE_b, the choice can be made by PE_a or PE_b, depending on the application.

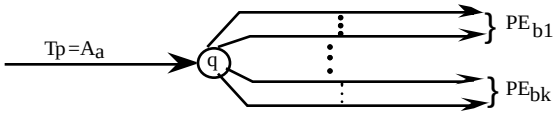


Fig. 3. Choice between several actions

This principle implies that the following three restrictions must be satisfied.

Restriction 1. The transitions which may occur in the initial state of SS are all executable by *a single*

protocol entity. □

Restriction 1 is necessary because the choice between several primitives executed at different sites is made by the PE which has just executed a primitive. With this approach, the choice cannot be made in the initial state, since no primitive has been executed.

Restriction 2. The choice between several primitives is made by the system and not by the user. □

In other terms, when there is a choice between several primitives, the latter must be outputs.

Restriction 3. The choice between primitives executed by a given PE_{bi} is made by PE_a. □

Restriction 3 implies that the choice between several primitives executed by a given PE_{bi} may not depend on some processing executed by PE_{bi}, after the reception of message m from PE_a.

Compared to [13,19], the following three improvements are made in the subsections below:

(a) *Restriction 2 is weakened* as follows: the choice between PE_{bi}s is made at $Site_a$ by the system, but the choice between several primitives of the selected PE_{bi} may be made at the selected site by the system (for upward primitives) or by the user (for downward primitives).

(b) *Restriction 3 is removed* as follows: PE_a is not necessarily required to select the following primitive; it may decide to select only the following protocol entity which, in turn, selects one of its primitives.

(c) PE_a sends a message only to the selected protocol entity.

3.6. Derivation procedure

SS being the input of the problem, the derivation procedure consists of the following two steps.

Step 1 : This step consists of completing SS by the insertion of a message exchange between each pair of consecutive primitives which are executed in different sites. In order to avoid any ambiguity, every message contains the identifier of the state which is reached in SS after the execution of the first of the two primitives. This tranformation implements the relation of causality between consecutive primitives.

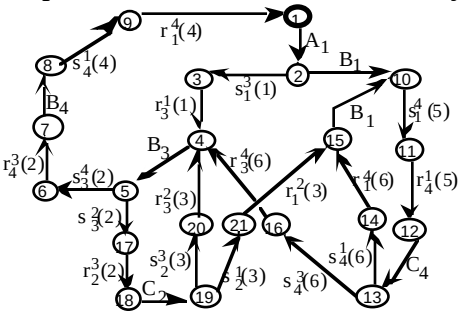
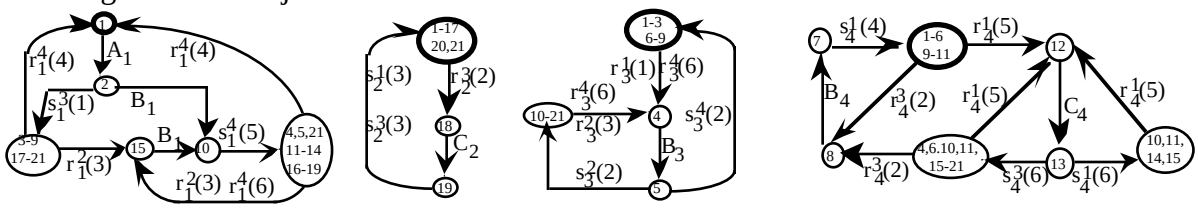


Fig. 4. GPS obtained in Step 2 for the example represented in Fig. 2.

Step 2 : From GPS, we compute the specification PS_i of each PE_i which must be implemented in $Site_i$ as follows. Each PS_i is obtained by projecting GPS into the alphabet of events which occur in $Site_i$. Then, the PS_i obtained are minimized and determinized.

We note that this two-step procedure is simpler than the procedures proposed in [13,19], besides being more optimal. The simplicity comes from the fact that, contrary to [13,19], each of the two steps of the procedure is intuitively understandable and justified. Procedures in [13,19] consists of a set of abstract transformations most of which are not intuitively explained.

For our example of Fig. 2, we obtain the specifications of Figures 4 and 5, after the first and second steps, respectively. To make the projections of GPS (Fig. 4) for obtaining PS_i (i=1 to 4) (Fig. 5) more directly visible, the states of PS_i are named according to their corresponding states in GPS, where "i-j" means all integers from i to j.



5.a. PS_1 **5.b.** PS_2 **5.c.** PS_3 **5.d.** PS_4

Fig. 5. Obtained protocol specifications

For lack of space, syntactic and semantic correctness of the protocol synthesized in not presented.

Contrary to [19,20], the rules for deriving protocol entities do not depend on whether primitives are upward or downward. In our opinion, such a distinction complicates uselessly the rules of Step 1. In fact, we must keep in mind that the aim of the messages generated is to guarantee the order of primitives implied by the service specification, independently whether they are upward or downward. Recall that the actors of choices are implicitly specified by the three improvements of Sect. 3.4. In the remaining part of this paper, we extend the procedure of protocol derivation to real-time distributed systems.

4. Timed automata for specifying services and correct protocols with temporal requirements

Two approaches have been used to model time: *Discrete-time* models which use the domain N of integers to model the time [3,17,18], and *Dense-time* models which use a dense domain to model the time [1,2,5,13,14,16]. In this study, we have used a dense-time model where the time is viewed as a state variable that ranges over a dense domain and evolves indefinitely.

4.1. Timed automata ($\mathcal{TA}$)

The timed automata ($\mathcal{TA}$) model we propose here uses a variable **v** and a clock **c**. $\mathcal{TA}$s are inspired from the model presented in [1]. Here are a few definitions which are necessary for a formal definition of a $\mathcal{TA}$.

Definition 4.1. (*Clock* **c**, *variable* **v**)

c has a positive real value which : (1) is set to zero at the occurrence of every transition; and (2) is equal to the time elapsed since the last instant it was set to zero.

v has a strictly positive natural value which can be updated at the occurrence of any transition. □

Let $A=(Q,\Sigma,\delta,q_0)$ be an $\mathcal{FSA}$ where Q is a set of states, Σ is an alphabet, q_0 is the initial state, and $\delta \subseteq Q\times\Sigma\times Q$ defines the transitions. Let us see how a $\mathcal{TA}$ can be defined from the $\mathcal{FSA}$ A.

Definition 4.2. (*Timed transition, and Timed automaton*)

Let I=[a;b] be an interval, where a and b are positive real numbers and a≥b.

A *timed transition* is defined by $[q,\sigma,r;C, v]$ where : **(a)** $[q,\sigma,r]$ defines a transition of the $\mathcal{FSA}$ $\mathcal{A}$; **(b)** $C=(I_1, I_2,...,I_m)$ is a *m*-tuple of *non-empty* intervals, where *m* is a strictly positive natural number; **(c)** v is a value of variable **v**.

A $\mathcal{TA}$ A^t can therefore be constructed if we transform every transition $tr=[q_1,\sigma,q_2]$ of A into a timed transition Tr by associating to it an *m*-tuple C of intervals and a value v of **v**. The semantics of a timed transition $Tr=[q,\sigma,r;C, v]$ of A^t depends on the current state q and on the current values of **v** and **c** as follows. If u is the current value of **v** then : (1) Tr is enabled (i.e., may occur) only if the current value of **c** falls within the uth interval I_u of C; and (2) after the occurrence of Tr, **v** is set to v and **c** is set to zero. Intuitively, the temporal constraint of a transition may depend on how the current state has been reached (this information is given by **v**). □

Henceforth, every timed transition is simply called transition, and $Tr=[q,\sigma,r;C, v]$ may be simply represented by $Tr=[\sigma;C, v]$ if there is no ambiguity about q and r. An example of a part of $\mathcal{TA}$ is given in Fig. 6. State q has two incoming (Tr_1 and Tr_2) and two outgoing transitions (Tr_3 and Tr_4), with $Tr_1=[q_1,\sigma_1,q;C_1,v_1]$, $Tr_2=[q_2,\sigma_2,q;C_2,v_2]$, $Tr_3=[q,\sigma_3,r_1;C_3,v_3]$, and $Tr_4=[q,\sigma_4,r_2;C_4,v_4]$. With the representation of Fig. 6, we can define v_1 and v_2, and C_3 and C_4. In fact, for a timed transition $Tr=[q,\sigma_1,r;C,v]$, the definition of v necessitates to know all the incoming transitions of r, and the definition of C necessitates to know all the incoming transitions of q. For example, $v_1=1$, $v_2=2$, $C_3=(I_{3_1},I_{3_2})$, $C_4=(I_{4_1},I_{4_2})$, $I_{3_1}=[1;2]$, $I_{3_2}=[0;2]$, $I_{4_1}=[1;3]$ and $I_{4_2}=[2;5]$. The informal specification is then the following, with q being the current state:

- if **v**=1, i.e., q has been reached by transition Tr_1, then :
  * Tr_3 (resp. Tr_4) may occur after a delay within the interval $I_{3_1}=[1;2]$ (resp. $I_{4_1}=[1;3]$);
  * If neither Tr_3 nor Tr_4 occurs after a delay within [1;2], then Tr_4 *must* occur after a delay within [2;3] (in order to avoid a deadlock).
- if **v**=2, i.e., q has been reached by the transition Tr_2, then :
  * Tr_3 (resp. Tr_4) may occur after a delay within the interval $I_{3_2}=[0;2]$ (resp. $I_{4_2}=[2;5]$);

* If Tr_3 does not occur after a delay within [0;2], then Tr_4 *must* occur after a delay within [2;5].

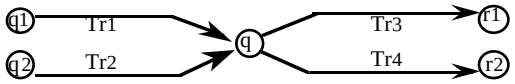


Fig. 6. Incoming and outgoing transitions

Our $\mathcal{TA}$ model can be transformed into the model presented in [1] as follows. Instead of using variable **v** which informs when necessary about how the current state q has been reached, we may define another state space in which the states are pairs (q, R), where q represents the current state and R a subset of the previous states. In this case, every enabling condition C becomes a single interval, but each state q is splitted into k states (q,R_i), for i=1,...,k, such that the original transitions (in our $\mathcal{TA}$ model) from the states of R_i to q set **v** to the same value i.

4.2. Service and protocol Specifications

A desired service is described by a $\mathcal{TA}$ denoted SST, which specifies: **(a)** the required sequences of primitives; and **(b)** certain temporal requirements between consecutive primitives. In any state q of SST, we can express some temporal constraints on the primitives which are executable from state q. These temporal constraints may depend on how q has been reached. As an example, the $\mathcal{FSA}$ SS of Fig. 2 (Sect. 3.1) is transformed into a $\mathcal{TA}$ SST by replacing transitions T_i of SS into the following timed transitions Tr_i, i=1,...,6, respectively : $Tr_1=[1,A_1,2;C1,1]$, $Tr_2=[2,B_3,3;C2,1]$, $Tr_3=[3,C_2,2;C3,2]$, $Tr_4=[3,B_4,1;C4,1]$, $Tr_5=[2,B_1,4;C5,1]$, $Tr_6=[4,C_4,2;\ C6,3]$, and $C1=I1$, $C2=(I2_1,I2_2,I2_3)$, $C3=I3$, $C4=I4$, $C5=(I5_1,I5_2,I5_3)$, $C6=I6$, where $I1$, $I2_1$, $I2_2$, $I2_3$, $I3$, $I4$, $I5_1$, $I5_2$, $I5_3$, and $I6$ are intervals. For example, if the current state 2 is reached by Tr_3, then **v** is set to 2, Tr_2 is enabled if and only if $(\mathbf{c} \in I2_2)$, and Tr_5 is enabled if and only if $(\mathbf{c} \in I5_2)$ (intervals $I2_2$ and $I5_2$ of $C2$ and $C5$ are used because **v**=2). Therefore, if we denote the sentence "the delay of Tr_a and Tr_b falls within the interval I" by " $I \leftarrow \langle Tr_a,Tr_b \rangle$" then :

$I1 \leftarrow \langle Tr_4,Tr_1 \rangle$ $\quad I2_1 \leftarrow \langle Tr_1,Tr_2 \rangle$ $\quad I2_2 \leftarrow \langle Tr_3,Tr_2 \rangle$ $\quad I2_3 \leftarrow \langle Tr_6,Tr_2 \rangle$ $\quad I3 \leftarrow \langle Tr_2,Tr_3 \rangle$

$I4 \leftarrow \langle Tr_2,Tr_4 \rangle$ $\quad I5_1 \leftarrow \langle Tr_1,Tr_5 \rangle$ $\quad I5_2 \leftarrow \langle Tr_3,Tr_5 \rangle$ $\quad I5_3 \leftarrow \langle Tr_6,Tr_5 \rangle$ $\quad I6 \leftarrow \langle Tr_5,Tr_6 \rangle$

A PE_a is described by a $\mathcal{TA}$ denoted PST_a, which specifies: **(a)** the sequences of local events which occur at $Site_a$; **(b)** certain temporal constraints to be satisfied between consecutive events. Similarly to the non-real-time case, the events may be of the three types *P*, *S* and *R* (see Sect. 3.2). Examples of $\mathcal{TA}$s specifying protocol entities will be given in Sect. 6.

4.3. Correctness of a protocol

We consider PE_1, PE_2 , ...,PE_n which are specified by PST_1, PST_2, ..., PST_n, respectively. Let $\mathcal{DS}$ be the distributed system constituted by PE_1, PE_2,...,PE_n and by the medium, and specified by a $\mathcal{TA}$ DST.

Definition 4.3. (*Timed sequence of events, Timed language, Acceptance*)

A timed sequence T is represented by $\langle \sigma_1,t_1 \rangle \langle \sigma_2,t_2 \rangle ... \langle \sigma_i,t_i \rangle ...$ and means that events σ_1, σ_2, ..., σ_i, ... occur at instants t_1, t_2, ..., t_i ..., respectively, where $t_1 < t_2 < ... < t_i < ...$ and each t_i is a positive real value.

A timed language is a set (possibly infinite) of timed sequences. Let A be a $\mathcal{TA}$, and L_A be the set of sequences which can be executed by A. Then we say that A *accepts* the language L_A. □

Definition 4.4. (*Projection, total and partial provision of a desired service with temporal requirements*)

The projection of a $\mathcal{TA}$ into a subalphabet Λ can be intuitively defined similarly to the projection of an $\mathcal{FSA}$ (see Def. 3.3). Therefore, $Proj_{V_s}(DST)$ specifies the service provided to the user by $\mathcal{DS}$, and each $Proj_{V_i}(DST)$ specifies PE_i, for i=1,..., n.

For the comparison of timed languages, we use the symbols $\cong_T$ and $<_T$, i.e., $A \cong_T B$ means that $L_A=L_B$, and $A <_T B$ means that $L_A \subset L_B$. Total and partial provisions of a real-time service are defined like in Def. 3.3, but by using symbols $\cong_T$ and $<_T$ instead of $\cong$ and $<$. □

5. Protocol Synthesis for Real-Time Applications

Definition 5.1. (*Reliable medium*)

A temporal model of the medium is necessary to compute temporal requirements for the PEs. Besides not

altering messages, in the real-time case a reliable communication medium must be such that the transit delay t_m of a message sent at $Site_a$ and received at $Site_b$, belongs to a finite interval $M_{a,b}=[\mu_{a,b};\rho_{a,b}]$ which depends on $Site_a$ and $Site_b$. □

The synthesis of the real-time PEs uses the same Step 1 of the non-real-time case, where we obtain GPST from SST (the last T indicates the presence of temporal constraints). In this step, the timed transitions are processed like simple transitions, while C and v are kept unchanged. GPST specifies the correct ordering of primitives, but it does not specify the correct temporal requirements of the service. In a subsequent step (see Sect. 5.4), we will use GPST and the model of the medium in order to compute temporal constraints of the PEs which guarantee the temporal requirements of the service.

5.1. Approach for the Problem of Computing Timing Requirements (PCTR)

To compute temporal constraints for the PEs, we consider every pair of states q and r of GPST which are connected by two consecutive events $s_a^b(p)$ and $r_b^a(p)$ (see Fig. 7). Let Tr be the single incoming transition of q (which corresponds to a primitive executed at a $Site_a$) and let $Tr_1,...,Tr_n$ be the outgoing transitions of r (which correspond to primitives executed at the same $Site_b$). After Tr, PE_a sends a message to PE_b (written $s_a^b(p)$); when PE_b receives the message (written $r_b^a(p)$), it executes one of the n Tr_k. The sequencing of events between $Tr=[q_1,\sigma,q;C,v]$ and $Tr_k=[r,\sigma_k,q_2;C_k,v_k]$ is represented as a function of the time in Fig. 8.a. The delay between Tr and Tr_k must belong to the vth interval $I_{k_v}=[\gamma_{k_v};\delta_{k_v}]$ of C which, for simplicity, is denoted $I_k=[\gamma_k;\delta_k]$.

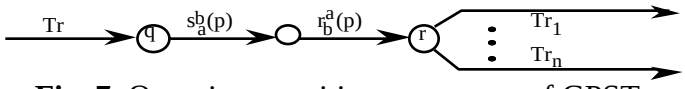
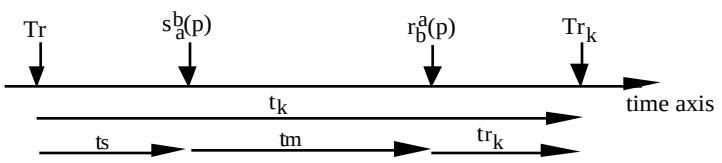
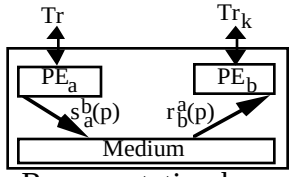


Fig. 7. Outgoing transitions on a state of GPST.



8.a. Representation in function of time



8.b. Representation by entities

Fig. 8. Representation of events between Tr and Tr_k

From Fig. 8.a, we see that the service requires that the time t_k, between the executions of Tr and Tr_k, falls within $I_k=[\gamma_k;\delta_k]$. The model of the medium implies that the transit delay tm of a message sent by PE_a and received by PE_b, falls within $M_{a,b}=[\mu_{a,b};\rho_{a,b}]$.
The aim of the temporal requirements derivation for the protocol entities is the following.

> From requirements $tm \in M_{a,b}=[\mu_{a,b};\rho_{a,b}]$ and $t_k \in I_k=[\gamma_k;\delta_k]$ (k=1, 2,..., n), we must compute constraints on ts and tr_k (k=1,2, ..., n) which ensure that requirements $t_k \in I_k$ on the service will be respected. These derived constraints are written in the form $ts \in S=[\theta;\phi]$, and $tr_k \in R_k=[\tau_k;\omega_k]$, k= 1, 2,..., n. This computation must be made for each occurrence of the structure in Fig. 7 within GPST.

Notations: operators $\subseteq$, $\cup$ or $\cap$ will be used on intervals, and [a;b]+[c;d]=[a+c;b+d], [a;b]-[c;d]=[a-c;b-d]. For the lack of space, correctness of the solutions given in the remaining of Sect. 5 is not proved here.

5.2. Conditions for the existence of solutions

We consider two consecutive transitions Tr_1 and Tr_2 wich are executed at $Site_i$ and $Site_j$, respectively. After Tr_1, $Site_i$ sends a message to $Site_j$ to inform it that it may execute Tr_2. If the delay between Tr_1 and Tr_2 must be greater than x and smaller than y, then the transit delay of the message must be smaller than y. Besides, the difference between the biggest delay and the smallest delay of the message in the medium must be smaller than y-x. Formally, for each occurrence of the structure in Fig. 7 within the GPST, wemust have : : for k= 1, 2, ..., n : $\delta_k - \rho_{a,b} \geq \sup(\gamma_k - \mu_{a,b}; 0)$ (1)

where sup(a;b) is equal to the biggest of a and b.
Therefore, for each occurrence within GPST of the structure in Fig. 7, we must check if (1) is respected. If the checking is positive then we must compute : **(a)** the interval S representing the constraint on ts,

and **(b)** intervals R_k, k=1, 2,..., n , representing the constraints on tr_k, k=1, 2,..., n.

We note that condition (1) is less restrictive than the conditions for the existence of solutions of [13].

5.3. Resolution

For resolving the timing constraints of the *PE*s, we consider the following three cases :

Static case : the messages transmitted by *PE*s contain no temporal information;

First dynamic case : the *PE*s include some temporal information in the messages they send;

Second dynamic case : the temporal information included by the *PE*s is completed by the medium.

In the following, ξ and ψ are any real values which fall within the interval [0;1].

5.3.1. Static case

We assume that the intervals S and R_k (see Sect. 5.1) are constant. When PE_a executes a transition Tr and decides to send a message to PE_b, the time ts between Tr and the transmission of the message falls within a constant interval S. When PE_b receives the message from PE_a, it can execute a transition Tr_k, among n possible transitions (k=1, 2, ..., n), in a time tr_k belonging to a constant interval R_k.

The interval $S=[\theta;\phi]$ must satisfy the following equations :

$$\phi = \psi * \min_{k=1 \text{ to } n} (\delta_k - \rho_{a,b}) \quad (2)$$

$$\theta = \sup(U, 0) + (\phi - \sup(U, 0)) * \xi \quad (3)$$

$$\text{with } U = \max_{k=1 \text{ to } n} (\phi + (\rho_{a,b} - \mu_{a,b}) - (\delta_k - \gamma_k)) \quad (4)$$

Afterwards, we choose the less restrictive solutions for $R_k=[\tau_k; \omega_k]$:

$$\text{for } k= 1, 2, ..., n : \quad \omega_k = \delta_k - \rho_{a,b} - \phi \quad (5)$$

$$\tau_k = \sup(\gamma_k - \mu_{a,b} - \theta; 0) \quad (6)$$

Let us see intuitively how the values of ψ and ξ may influence the synthesized system. Taking ψ as small as possible and ξ as large as possible, implies to have ϕ and θ as small and as close as possible. In this case, ω_k and τ_k will be the less constrained possible. Therefore, the sender entity will be more constrained and the receiving entity will have as much time as possible to provide the service. Generally, modifying ψ and ξ allows to "move" some timing constraints between two communicating entities.

5.3.2. First dynamic case

We assume that PE_a sends to PE_b a message containing ts (Fig. 8.a), and PE_b calculates dynamically R_k as a function of ts when it receives the message from PE_a.

The interval $S=[\theta;\phi]$ must satisfy the following equations :

$$\phi = \psi * \min_{k=1 \text{ to } n} (\delta_k - \rho_{a,b}) \quad (2)$$

$$\theta = \phi * \xi \quad (7)$$

The interval $R_k(ts)$ is computed as follows. If ts, which belongs to $[\theta;\phi]$, is the delay when the message is sent after the execution of Tr_p, the receiving entity knows it and can choose :

$$\text{for } k= 1, 2, ..., n : \quad \omega_k(ts) = \delta_k - \rho_{a,b} - ts \quad (8)$$

$$\tau_k(ts) = \sup(\gamma_k - \mu_{a,b} - ts; 0) \quad (9)$$

Intuitively, with the information ts, the receiving entity PE_b can use the time allocated to it to provide the service more efficiently than in the static case. Let us, for instance, assume that some optional tasks are achieved by PE_b, in order to provide a better quality of service, only if PE_b has enough time. In the static case, PE_b may estimate that it has not enough time to execute its optional tasks, while in the dynamic case optional tasks will be executed. In other terms, sometimes in the static case PE_b has to "hurry up" when in the dynamic case it does not have to.

5.3.3. Second dynamic case

Compared to the first dynamic case, we assume in this case that the medium modifies ts into the more accurate information ts+tm. In this case, PE_b receives the message with information ts+tm, and it calculates dynamically the interval R_k, as a function of ts+tm.

$S = [\theta;\phi]$ is resolved as in Sect. 5.3.2; ω_k and τ_k are calculated dynamically by PE_b as follows :

$$\text{for } k= 1, 2, ..., n : \quad \omega_k(ts+tm) = \delta_k - (ts+tm) \tag{10}$$

$$\tau_k(ts+tm) = \sup(\gamma_k - (ts+tm); 0) \tag{11}$$

Intuitively, with the information ts+tm the receiving entity PE_b knows that Tr has been executed ts+tm before the reception of the message, which is a more accurate information than in the first dynamic case. Due to this fact, in the second dynamic case PE_b can use the time allocated to it to provide the service more efficiently than in the first dynamic case.

We note that ts and ts+tm , which are transmitted in the dynamic cases, are a *relative* temporal information. This is an advantage since it implies that a global clock is not necessary.

We also note that the temporal requirements of the protocol obtained using the approach in [13] are more restrictive than those derived by our improved approach.

5.4. Derivation Procedure

The derivation procedure consists of three steps. Step 1, which generates a specification GPST, is similar to step 1 of the non-real-time case.

Step 2: The aim of this step is : **(a)** to compute and insert into GPST the static temporal constraints and, in the dynamic cases, some constant parameters which allow to compute the dynamic temporal constraints; **(b)** to insert ts and tm into the exchanged messages. The $\mathcal{TA}$ obtained is denoted GST. Therefore, for every structure represented in Fig. 7 and contained in GPST, the following three substeps are performed to transform GPST into GST.

Step 2.1. We compute the interval $S=[\theta ; \phi]$ and:

- Intervals R_k, k=1,...,n, in the static case (Sect. 5.4.1);
- Intervals $X_k=I_k-M_{a,b}$, k=1,...,n, in the first dynamic case;

Step 2.2 $\circ \xrightarrow{s_a^b(p)} \circ \xrightarrow{r_b^a(p)} \circ$ becomes: (v being the value of **v** which is set by the transition preceding $s_a^b(p)$)

- In the static case : $\circ \xrightarrow{(s_a^b(p);S,v)} \circ \xrightarrow{(r_b^a(p);M_{a,b},v)} \circ$
- In the first dynamic case : $\circ \xrightarrow{(s_a^b(p,ts);S,v)} \circ \xrightarrow{(r_b^a(p,ts);M_{a,b},v)} \circ$
- In the second dynamic case : $\circ \xrightarrow{(s_a^b(p,ts+tm);S,v)} \circ \xrightarrow{(r_b^a(p,ts+tm);M_{a,b},v)} \circ$

Informally: - the delay between occurrences of Tr and $s_a^b(*)$ falls within $S=[\theta;\phi]$;

- the delay between occurrences of $s_a^b(*)$ and $r_b^a(*)$ falls within $M_{a,b}$.

Step 2.3 For each k=1, ...,n, the vth interval I_k of C_k is replaced by the interval: **(i)** R_k in the static case; **(ii)** $X_k=I_k-M_{a,b}$ in the first dynamic case; **(iii)** I_k (i.e., it is not replaced) in the second dynamic case.

We note that the $\mathcal{TA}$ obtained at Step 2.3, which we call GST, is defined by constant intervals. In dynamic cases, some of these constant intervals do not directly represent timing constraints, but they are used for a dynamic calculation of the time requirements. In fact, for each k=1,...,n, the delay between occurrences of $r_b^a(p)$ and Tr_k must belong to: - the *constant* interval R_k in the static case;

- a *variable* interval $R_k(ts)$ which depends on the *constant* interval $X_k=I_k-M_{a,b}$ and on ts ;
- a *variable* interval $R_k(ts+tm)$ which depends on the *constant* interval I_k and on ts+tm.

Step 3 : This step consists of generating the protocol specification PST_i by projecting GST onto the alphabet of events which occur at $Site_i$, i=1,...,n. This step is similar to the second step of the non-real-time case, with the difference that intervals $M_{a,b}$ are replaced by $[0;\infty]$. Informally, timing constraints for the receptions of events do not need to be explicitly specified, since they are implicitly specified by the model of the communication medium. Replacing $M_{a,b}$ by $[0;\infty]$ is mandatory, because $M_{a,b}$ is a timing constraint between two consecutive events $s_a^b(*)$ and $r_b^a(*)$ of GST which, after the projections, will be in two different timed automata PST_a and PST_b.

For the lack of space, syntactic and semantic correctness of the protocol synthesized in not presented.

Remarks. About the passage from the non real-time case to a real-time case:

(a) The passage to the static case necessitates that the protocol entities and the medium must satisfy the required temporal constraints;

(b) The passage to the first dynamic necessitates, besides the requirement in (a), a modification of the protocol which must add some temporal information in its messages (ts).

(c) The passage to the second dynamic case necessitates, besides the requirements in (a) and (b), a modification of the medium which must modify the temporal information ts into ts+tm.

These remarks illustrate the price to pay for each of the three alternatives.

6. Examples

We have developed a tool called PROSYN which implements our synthesis method. The application of this tool is illustrated in the following three examples, where ψ and ξ (Sect. 5.1, 5.2) are taken to be equal to 0.5, which means that the temporal constraints are equally distributed between the two sites.

6.1. A Pedagogical Example

We consider the SST with four states and six transitions presented in Sect.4.2. Let us for instance take :

- $I1=[3;6]$, $I2_1=[5;10]$, $I2_2=[4;8]$, $I2_3=[4;10]$, $I3=[3;8]$, $I4=[4;9]$, $I5_1=[1;3]$, $I5_2=[4;8]$, $I5_3=[3;8]$, $I6=[4;10]$.
- $M_{a,b}=[2;4]$ for all pairs of sites (Def. 5.1).

If we apply the derivation procedure, we obtain the PST_is in each site. In the static case, the derived PST_is are represented in Fig. 9, with $\mathcal{D}1=R1$, $\mathcal{D}2=(R2_1,R2_2,R2_3)$, $\mathcal{D}3=R3$, $\mathcal{D}4=R4$, $\mathcal{D}5=(R5_1,R5_2,R5_3)$, $\mathcal{D}6=R6$. Using formulae (2-6), we calculate :

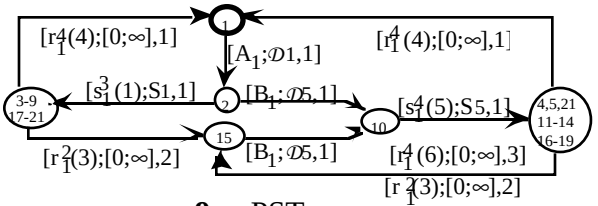
In PST_1 : $S1=[1.5;3]$, $S5=[1.5;3]$, $R1=[0.5;1]$, $R5_1=[1;3]$, $R5_2=[1;2]$, $R5_3=[0;2]$,
In PST_2 : $S3_1=[1;2]$, $S3_2=[1;2]$, $R3=[0;2]$,
In PST_3 : $S2_1=[1;2]$, $S2_2=[1.25;2.5]$, $R2_1=[1.5;3]$, $R2_2=[1;2]$, $R2_3=[0.5;3]$,
In PST_4 : $S4=[0.5;1]$, $S6_1=[1.5;2]$, $S6_2=[1;2]$, $R4=[0.75;2.5]$, $R6=[0.5;3]$.

In the first dynamic case, we must :

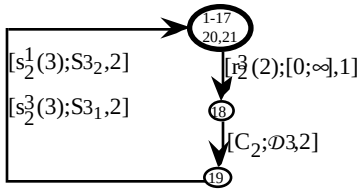
- replace every $s_a^b(p)$ and $r_b^a(p)$ respectively by $s_a^b(p,ts)$ and $r_b^a(p,ts)$;
- compute every Si_j by formulae (2,7);
- compute the constant intervals X_i which are used to compute dynamically $R_i(ts)$ with formulae (8,9);
- replace intervals R_i by intervals X_i in the specifications of Figure 9.

In the second dynamic case, we must :

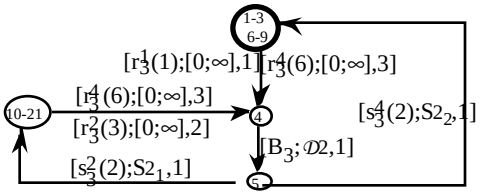
- replace every $s_a^b(p)$ and $r_b^a(p)$ respectively by $s_a^b(p,ts)$ and $r_b^a(p,ts+tm)$;
- compute every Si_j by formulae (2,7);
- replace intervals R_i by intervals I_i in the specifications of Figure 9.



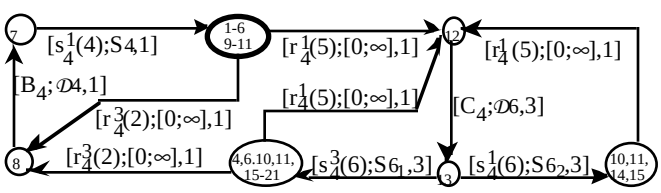
9.a. PST_1



9.b. PST_2



9.c. PST_3



9.d. PST_4

Fig. 9. First example of protocol specifications with timing requirements

6.2. Synthesis of a Simplified X.25 Protocol with Temporal Requirements

X.25 [6] is a communication protocol consisting of the lowest three levels of the OSI reference model. Services are offered to the user through the network layer. Globally, the X.25 protocol allows two sites $Site_i$ and $Site_j$ of the network to communicate. After the two sites have established a connection, they can exchange data. The communication between them is stopped when one of the two sites initiates a disconnection. A site can send a message at any moment without waiting for an acknowledgement. Two kinds of data are supported: normal and express data. Each of the two kinds of data are transmitted

according to a FIFO discipline. But the FIFO discipline is not respected between the two kinds of data since express data may be received before normal data which were sent before.

In order to apply our synthesis method, the X.25 service is made sequential by assuming that the service primitives are ordered and executed sequentially. For that purpose, the following assumptions are made: **(i)** a new message cannot be sent before the last one is received; and **(ii)** express data are not supported. The simplified X.25 service obtained will be extended by adding certain temporal requirements between consecutive primitives.

In order to give the possibility to both sites to establish a connection, we have used a mechanism of tokens to realize a distributed choice. The following description of this example is based on [9].

6.2.1. Primitives of the Simplified X.25 Service

Let U_1 and U_2 be two users of the network who are located in $Site_1$ and $Site_2$, respectively. The following service primitives are defined :

- *Connection* : A connection may be established between U_1 and U_2 if one of them, for instance U_1, sends a *Connect request* (CN.req) to U_2. When the latter receives a *Connect indication* (CN.ind), he may answer either by a *Disconnect request* (DC.req) to reject the connection request, or by a *Connect response* (CN.rsp). In the first case, U_1 receives a *Disconnect indication* (DC.ind), while in the second case U_1 receives a *Connect confirm* (CN.cnf).
- *Disconnection* : A disconnection primitive can be used either to reject a Connect request (see above) or to terminate an existing connection. For instance, U_1 may send a *Disconnect request* (DC.req) and then U_2 will receive a *Disconnect indication* (DC.ind).
- *Data Transfer* : This primitive allows to transfer data in both directions between two sites linked by a connection. To simplify the example, we assume that only the party which has initiated the connection can send data. The sending of a message is generated by a *Data request* (DT.req) and its reception by a *Data indication* (DT.ind)
- *Reinitialization* : The Reinitialization procedure allows to restore the synchronization between two parties. When a *Reinitialization request* (RI.rqt) is generated, for instance by U_1, then all the data being transmitted in the medium is removed. The next element to be received by U_2 is a *Reinitialization indication* (RI.ind). U_2 answers by a *Reinitialization response* (RI.rsp) and then U_1 will receive a *Reinitialization confirm* (RI.cnf). We assume that the party which requests the reinitialization is the sender of data.

6.2.2. Specification of the Simplified X.25 Service

Our specification contains principally two blocs $S_{1,2}$ and $S_{2,1}$, where $S_{i,j}$ (see Fig. 10) models the service when $Site_i$ and $Site_j$ are the sender and the receiver, respectively. The specification of the simplified X.25, which contains the two blocks, is schematized in Fig. 11. The event $Token_i^j$ means that "$Site_i$ gives to $Site_j$ the possibility to establish a connection". The other events have been defined in Sect. 6.2.1. State $1_{i,j}$ is the initial state of Block $S_{i,j}$; in this state, $Site_i$ has the possibility to request a connection but may also give this possibility to $Site_j$ (by the transition $Token_i^j$). State $6_{i,j}$ is the state where the connection has been established by $Site_i$ which is therefore ready to send data. We assume that State $1_{1,2}$ is the initial state of the service to be provided (Fig. 11).

6.2.3. Temporal constraints added to the simplified X.25 service

- The delay between $CN.req_i$ and $CN.ind_j$ belongs to the interval [1;1.5];
- The delay between $DC.req_i$ and $DC.ind_j$ belongs to the interval [0.5;1];
- The delay between $RI.ind_i$ and $RI.rsp_i$ belongs to the interval [0;0.5];
- The delay between $DT.req_i$ and $DT.ind_j$ belongs to the interval [1;1.5];
- The delay between $DT.rsp_i$ and $DT.cnf_j$ belongs to the interval [1;1.25].

Transitions with temporal constraints are represented in grey in Fig. 10 and 11. Intervals defining the temporal constraints are also represented on these transitions.

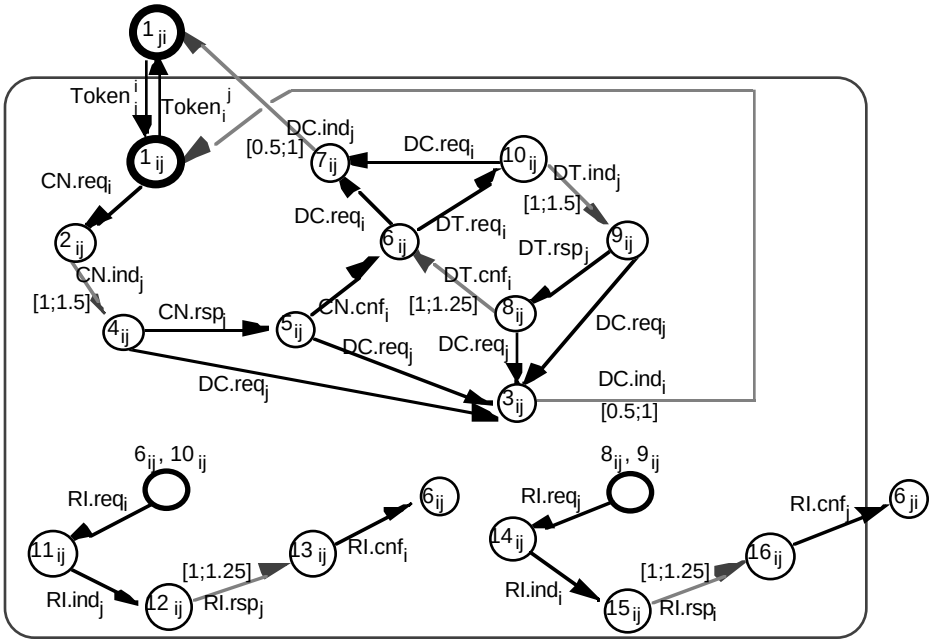
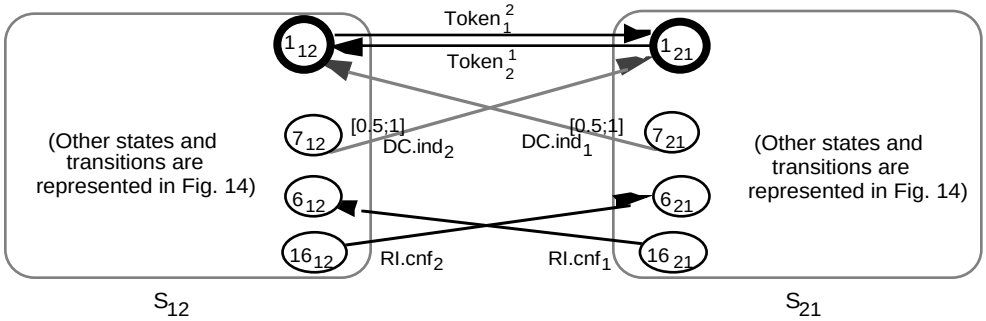


Fig. 10. Block $S_{i,j}$



S_{12} S_{21}

Fig. 11. Service specification of the simplified X.25

6.2.4. Protocol Synthesis

Our synthesis tool PROSYN has been applied to the service represented in Fig. 10 and 11, in the static case and the two dynamic cases. Due to the symmetry of the service, the synthesized specifications of the two protocol entities are quite similar and can be represented by only one parameterized $\mathcal{TA}$ which is represented in Fig. 12, and where i identifies the PE represented and j identifies the other PE. Messages sent by each PE_i contain the parameters p_i^k, k=1,..., 17, with $p_i^r \neq p_i^s$ if $r \neq s$. The initial states of PE_1 and PE_2 are identified by 1 and 2, respectively. Like in the service specification, the transitions with temporal constraints are represented in grey in Fig. 12. To generate these real-time PEs, the following temporal model of the medium has been used: the transit delay of a message falls within [0.5;0.75] when it is transmitted from $Site_1$ to $Site_2$, and within [0.25;0.5] when it is transmitted from $Site_2$ to $Site_1$. For simplicity, we give only the results of the static case. The synthesized temporal constraints, which are defined by an interval for the transitions represented in grey in Fig. 12, are the following.

In $Site_1$: the constraint for $s_1^2(p2_1)$, $s_1^2(p3_1)$, $s_1^2(p4_1)$, $s_1^2(p5_1)$, $s_1^2(p10_1)$ and $s_1^2(p11_1)$ is [0.0625; 0.125];
the constraint for $s_1^2(p7_1)$ is [0.25; 0.25]; the constraint for $s_1^2(p8_1)$ and $s_1^2(p9_1)$ is [0.25; 0.375];
the constraint for $DT.cnf_1$ is [0.375; 0.375]; the constraint for $DT.ind_1$ is [0.375; 0.5];
the constraint for $DC.ind_1$ is [0.125; 0.25]; the constraint for $RI.rsp_1$ is [0; 0.5].

In $Site_2$: the constraint for $s_2^1(p2_2)$, $s_2^1(p3_2)$, $s_2^1(p4_2)$, $s_2^1(p5_2)$, $s_2^1(p10_2)$ and $s_2^1(p11_2)$ is [0.125; 0.25]
the constraint for $s_2^1(p7_2)$ is [0.375; 0.375]; the constraint for $s_2^1(p8_2)$ and $s_2^1(p9_2)$ is [0.375; 0.5];
the constraint for $DT.cnf_2$ is [0.25; 0.25] ; the constraint for $DT.ind_2$ and $CN.ind_2$ is [0.25; 0.375] ;
the constraint for $DC.ind_2$ is [0; 0.125]; the constraint for $RI.rsp_2$ is [0; 0.5].

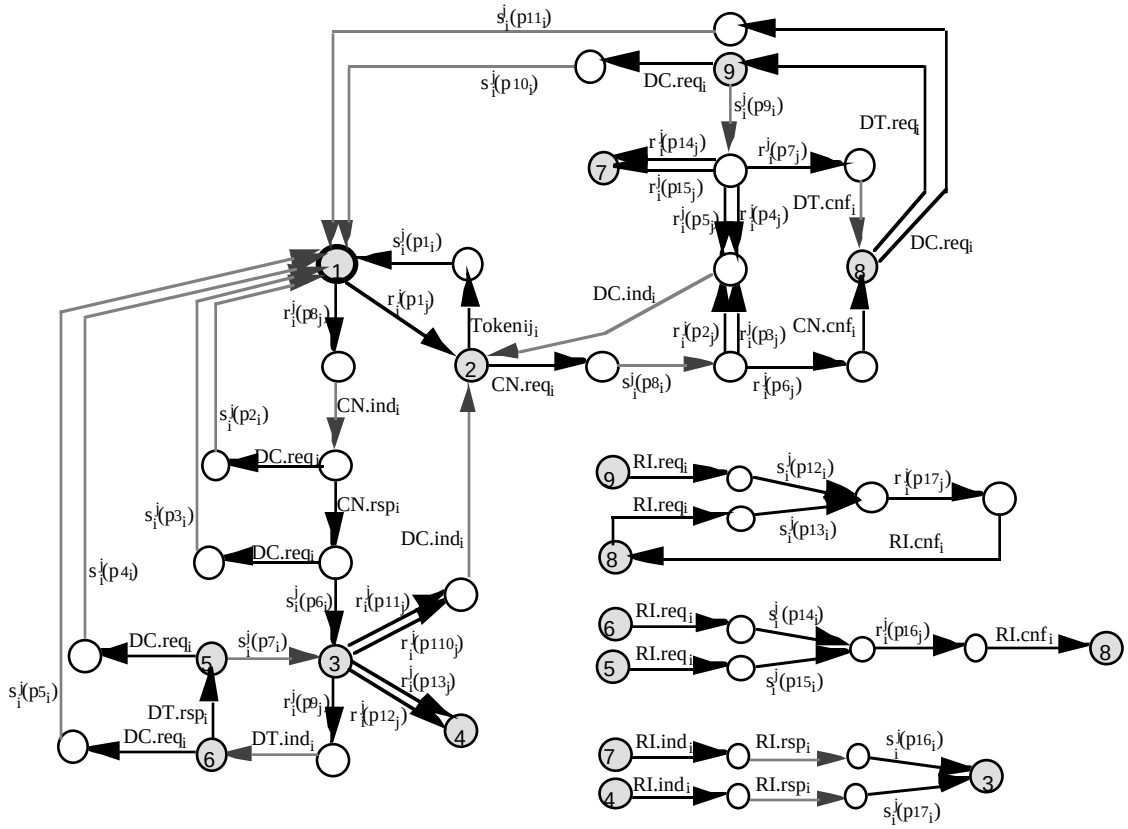


Fig. 12. Specification of the synthesized X.25 protocol at $Site_i$

6.3. Controlling Several Robots to Assemble Pieces

The following example is interesting in the sense that it illustrates the application of our synthesis method in another area than telecommunications. We consider an assembly system consisting of three robots R1, R2 and R3 and three carpets C1, C2 and C3. The carpets C1 and C2 bring pieces of type P1 and P2, respectively, and carpet C3 takes away the assembled pieces. Robot R1 takes a piece P1, and puts it on a table T for the assembly. Robot R2 takes a piece P2 and assembles it with the piece P1 which is on the table T. Robot R3 removes the defective pieces. The details of this example are given in [15].

6.3.1. Protocol entities

There are six PEs which correspond to the three robots and the three carpets. They are identified by R1, R2, R3, C1, C2 and C3, respectively. We consider that each carpet consists of the carpet itself, of an actuator which moves and stops the carpet, and of a sensor which indicates whether a piece carried by the carpet has reached its destination. Table T is not considered as an entity since it is passive.

6.3.2. Primitives of the service

The service primitives are the following (with i=1, 2, 3, and j=1, 2) :

- MOVE.$Carpet_{Ci}$: Carpet Ci is actuated (it begins to move),
- ARRIVED.$Piece_{Ci}$: Carpet Ci has detected that a piece Pi has reached its destination,
- STOP.$Carpet_{Ci}$: Carpet Ci is stopped,
- CHECK.$Piece_{Rj}$: Robot Rj begins to check a piece Pj,
- TAKE.$Piece_{Rj}$: Robot Rj takes a piece Pj from carpet Cj,
- TAKE.$Piece1_{R3}$: Robot R3 takes a piece P1 from carpet C1,
- TAKE.$Piece2_{R3}$: Robot R3 takes a piece P2 from carpet C2,
- TAKE.$AssPieces_{R3}$: Robot R3 takes an assembled piece from table T,
- PUT.$Piece_{R1}$: Robot R1 puts a piece P1 on table T,
- PUT.$AssPieces_{R2}$: Robot R2 puts an assembled piece on carpet C3,
- ASS.$Pieces_{R2}$: Robot R2 assembles pieces P1 and P2.

6.3.3. Scenario of the Service

From the initial state where the whole system is stopped, the scenario of the service is the following:

Step 1 : Carpet C1 is actuated (primitive MOVE.Carpet$_{C1}$)
Step 2 : Piece P1, which is on C1, reaches its destination, which is detected by C1 (ARRIVED.Piece$_{C1}$)
Step 3 : Carpet C1 is stopped (STOP.Carpet$_{C1}$)
Step 4 : Robot R1 checks P1 (CHECK.Piece$_{R1}$):
Step 5: If P1 is bad then R3 takes it off (TAKE.Piece1$_{R3}$), and *goto Step 1.*
Step 6: If P1 is good then R1 takes it from C1 (TAKE.Piece$_{R1}$) and
Step 7: R1 puts C1 on the table T (PUT.Piece$_{R1}$)
Step 8 : Carpet C2 is actuated (MOVE.Carpet$_{C2}$)
Step 9 : Piece P2, which is on C2, reaches its destination, which is detected by C2 (ARRIVED.Piece$_{C2}$)
Step 10 : Carpet C2 is stopped (STOP.Carpet$_{C2}$)
Step 11 : Robot R2 checks P2 (CHECK.Piece$_{R2}$):
Step 12 : If P2 is bad then R3 takes it off (TAKE.Piece2$_{R3}$), and *goto Step 8.*
Step 13 : If P2 is good then R2 takes it from C2 (TAKE.Piece$_{R2}$) and
Step 14 : R2 assembles P2 with P1 on the table T (ASS.Pieces$_{R2}$).
Step 15 : If the assembly is bad, which it is detected by R2, then R3 takes it off (TAKE.AssPieces$_{R3}$), and *goto Step 1.*
Step 16 : If the assembly is good then R2 takes it and puts it on carpet C3 (PUT.AssPieces$_{R2}$)
Step 17 : Carpet C3 is actuated (MOVE.Carpet$_{C3}$)
Step 18 : The assembled pieces reach their destination, which is detected by carpet C3 (ARRIVED.Piece$_{C3}$).
Step 19 : Carpet C3 is stopped (STOP.Carpet$_{C3}$), and *goto Step 1.*

6.3.4. Temporal constraints added to the service

- The delay between MOVE.Carpet$_{Ci}$ and ARRIVED.Piece$_{Ci}$ belongs to [10;20]; (i=1,2,3)
- The delay between ARRIVED.Piece$_{Ci}$ and STOP.Carpet$_{Ci}$ belongs to [0.5;2]; (i=1,2,3)
- The delay between STOP.Carpet$_{Ci}$ and CHECK.Piece$_{Ri}$ belongs to [5;8]; (i=1,2)
- The delay between CHECK.Piece$_{Ri}$ and TAKE.Piecei$_{R3}$ belongs to [5;8]; (i=1,2)
- The delay between ASS.Piece$_{R2}$ and TAKE.AssPieces$_3$ belongs to [5;8];
- The delay between CHECK.Piece$_{Ri}$ and TAKE.Piece$_{Ri}$ belongs to [1;2]; (i=1,2)
- The delay between TAKE.Piece$_{R1}$ and PUT.Piece$_{R1}$ belongs to [1;2];
- The delay between TAKE.Piece$_{R2}$ and ASS.Pieces$_{R2}$ belongs to [4;10];
- The delay between PUT.Piece$_{R1}$ (or TAKE.Piece2$_{R3}$) and MOVE.Carpet$_{C2}$ belongs to [5;10];
- The delay between PUT.AssPieces$_{R2}$ and MOVE.Carpet$_{C3}$ belongs to [5;10];
- The delay between ASS.Pieces$_{R2}$ and PUT.AssPieces$_{R2}$ belongs to [2;5];
- The delay between TAKE.Piece1$_{R3}$ (or TAKE.AssPieces$_{R3}$ or STOP.Carpet$_{C3}$) and MOVE.Carpet$_{C1}$ belongs to [6;10]; (i=1,2)

The specification of this service is represented in Fig. 13, where transition Tr$_i$ corresponds to Step i of the scenario (Sect. 6.3.3). As in the previous example, the temporal constraint of each transition is defined by a single interval (which is shown in Fig. 13).

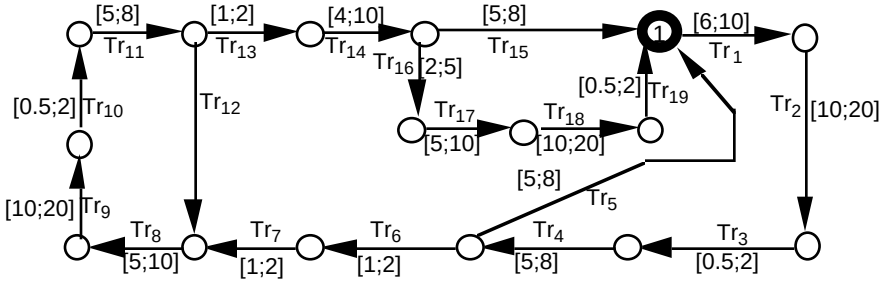
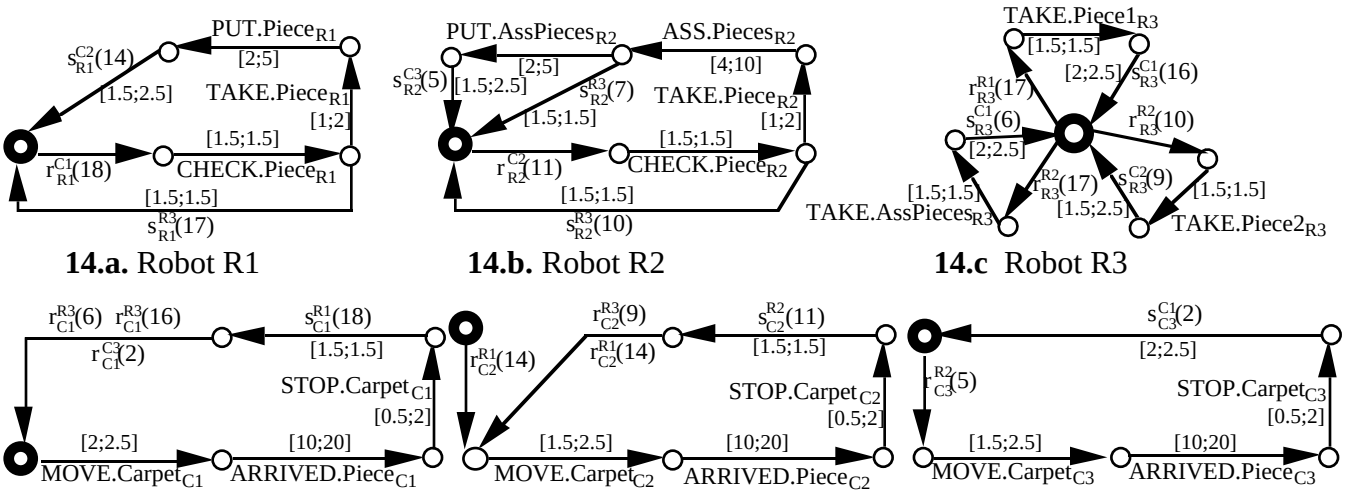


Fig. 13. Service specification of the assembling system

6.3.5. Protocol Synthesis

Our synthesis tool PROSYN has been applied to the service specification represented in Fig. 13 for the static and the two dynamic cases. The specifications synthesized in the static case are represented in Fig. 18, and consist of six $\mathcal{TA}$s modeling the behaviour of the three robots and the three carpets, respectively. To generate these real-time PEs, the transit delay of all messages has been assumed to belong to [2;5].



14.a. Robot R1 **14.b.** Robot R2 **14.c** Robot R3

14.d. Carpet C1 **14.e.** Carpet C2 **14.f.** Carpet C3

Fig. 14. Synthesized specifications of the robots and the carpets in the static case

6.4. Conclusion about the two concrete examples

Our synthesis method can be applied, not only for the design of communication protocols, but also in other areas such as robotics. Our method can be applied to realistic applications, provided that the latter are made sequential. The modifications which make a system sequential must be reasonable, in the sense that the service provided by the simplified system must be useful.

7. Conclusion

A method for deriving real-time protocols, which has been proposed in [13], is improved and extended in the present paper. In comparison with [13], the main advantages of the present approach are : (1) the number of exchanged messages is minimized; (2) the choice between several primitives can be made either by the user or by the system; (3) the conditions for the existence of a solution and the derived temporal constraints are weaker; (4) two simple but realistic applications of our method are proposed.

As in [13], the timing requirements of the synthesized protocols can be calculated statically or dynamically. The dynamic case is interesting because the receiving protocol entities use more efficiently the time allocated to them to provide the service.

Our method imposes two restrictions: (r1) the service specification is *not concurrent*; and (r2) timing requirements are only between *consecutive* primitives. Several methods of protocol synthesis for parallel systems, i.e., without restriction (r1), have been developed (e.g. [10]), but most of them do not consider timing requirements. A simplified version of the method in [10] has been extended to deal with timing constraints [11,12], but with restrictions.

We are presently investigating an approach using the following three steps :

Step (a) : the given service specification S is transformed into a sequential service, called S^{seq}.

Step (b). A protocol P^{seq} providing the sequential service S^{seq} is synthesized by using our method; the obtained protocol is a kind of "skeleton" of a protocol P providing S.

Step (c). P^{seq} is transformed in order to obtain a protocol P which provides S.

We note that Step (b) is realized automatically by the method presented in the present paper, and that Step (a) has been applied manually in the examples of Sect. 6.2 and 6.3 to make our method applicable. We therefore try to find a systematic way to achieve steps (a) and (c).

REFERENCES

[1] R. ALur and D.Dill, "Automata for Modeling Real-Time Systems," in *Proceedings of the 17th Intern. Coll. on Automata, Languages and Programming, Lecture Notes in Comp. Sci. 443*, Ed. Springer-Verlag, Warwick, UK, 1990.

[2] B. Bertomieu and M.Diaz, "Modeling and verification of time dependant systems using Petri nets," *IEEE Transactions of Software Engineering*, Vol.17, N° 3, pp. 259-273, March 1991.

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

[4] P.Y.M. Chu and M.T.Liu, "Synthesizing Protocol specifications from service specifications in FSM model," in *Proceedings of the IEEE Computer Networking Symposium,* pp. 173-182, April 1988.

[5] J-P. Courtiat, M. S. de Camargo, D-E. Saidouni, "RT-LOTOS: LOTOS temporisé pour la spécification de systèmes temps réel," in *Proceedings of Colloque Francophone pour l'ingénierie des protocoles (CFIP),* Montreal, September 1993.

[6] R.J. Deasington, "*Protocoles X.25 pour les réseaux à commutation de paquets,*" Masson, 1987.

[7] R. Gotzhein and G. v. Bochmann, "Deriving Protocol Specifications from Service Specifications Including parameters," *ACM Transactions on Computer Systems,* Vol. 8, N° 4, pp. 255-283, 1990.

[8] T. Higashino, K. Okano, H. Imajo and K. Taniguchi, "Deriving protocol specifications from service specifications in Extended FSM models," in *Proceedings of the 13th Intern. Conf. on Distributed Computing Systems,* pp. 141-148, 1993.

[9] Y. Iraqi, "Synthèse du protocole X.25 simplifié avec contraintes de temps. Utilisation de l'outil de Khoumsi," *Report of a project realized at the University of Montreal,* April, 1996.

[10] C. Kant, T. Higashino and G.v. Bochmann, "Deriving Protocol Specifications from Service Specifications Written in LOTOS," *Distributed Computing,* Vol. 10, N° 1, pp. 29-47, 1996.

[11] M. Kapus Kolar, "Deriving protocol specifications from service specifications with heterogeneous timing requirements." in *Proceedings of the IEEE Int. Conf. on Software Engineering for real time systems,* United-Kingdom, 1991.

[12] M. Kapus Kolar and J. Rugelj, "Deriving protocol specifications from service specifications with simple relative timing requirements," in Proceedings of *ISMM Int. Workshop on parallel computing,* Italy, 1991.

[13] A. Khoumsi, G.v. Bochmann, and R. Dssouli, "Dérivation de spécifications de protocole à partir de spécifications de service avec des contraintes temps-réel," *Revue Réseaux et informatique répartie (RIR),* Vol.4, N° 1, April 1994.

[14] G. Leduc and L. Léonard, "A Timed LOTOS supporting a dense time domain and including new timed operators," in *Proceedings of the Int. Conf. FORTE,* Perros-Guirec, France, Oct. 1992, Ed. North-Holland, Amsterdam, 1993.

[15] E. Madja, "Dérivation de protocoles pour applications temps réeel. Application au système d'assemblage," *Report of a project realized at the University of Montreal,* April, 1996.

[16] P. Merlin and D.J. Faber, "Recoverability of communication protocols," *IEEE Transactions on Communication,* Vol. 24, N° 9, September 1976.

[17] J.S. Ostroff, "Deciding Properties of Timed Transitions Models," *IEEE Transactions on Parallel and Distributed Systems,* Vol.1, N° 2, pp.170-183, April 1990.

[18] J.S. Ostroff and W.M. Wonham, "A framework for real-time discrete event control," *IEEE Transactions on Automatic Control,* Vol.35, N° 4, pp.386-397, April 1990.

[19] K.Saleh and R. Probert, "A service-based method for the synthesis of Communications protocols," *International Journal of Mini and Microcomputers,* Vol. 12, N° 3, pages 97-103, December 1990.

[20] K.Saleh and R. Probert, "An extended service-based method for the synthesis of protocols," in *Proceedings of the Sixth Intern. Symp. on Comp. and Inform. Sciences,* pp. 547-557, October 1991.

[21] H. Yamaguchi, K. Okano, T. Higashino and K. Taniguchi, "Synthesis of protocol entities specifications from service specifications in a Petri Net model with registers,", in *Proceedings of IEEE Parallel and Distributed Computing Systems,* 1995.