Protocol Synthesis for Real-Time Applications

AHMED KHOUMSI
Department GEGI, University of Sherbrooke, Sherbrooke, Canada

GREGOR V. BOCHMANN
School of ITO , University of Ottawa, Ottawa, Canada

RACHIDA DSSOULI
Department IRO, University of Montreal, Montreal, Canada

Key words: Real-Time Protocol, Service, Synthesis, Assembly System, X.25 Protocol.

Abstract : This paper deals with 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 several modifications and additions. First, the number of messages exchanged between the protocol entities is minimized. Second, a less restrictive strategy for choosing between several service primitives is proposed. 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 examples of applications are described.

1. INTRODUCTION

Several methods for deriving a protocol from a service have been developed by various researchers [3,5-7,10,11]. 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. In [6] transit delays in the medium are supposed negligible, while in [7] they are bounded by a maximum value. In the present paper, our aim is to describe an approach for deriving protocols which guarantee both : **(a)** a correct ordering of service primitives; and **(b)** the satisfaction of given timing requirements between the executions of service primitives, in a more general case than in [6,7].

A protocol derivation approach presented in [10], which guarantees only (a), has been extended in [8] to the real-time case, i.e., for ensuring also (b). The timing requirements considered in [8] allow to specify certain constraints on the delays between consecutive service primitives. In the present paper, we consider the same problem but we provide a better

J. Wu et al. (eds.), *Formal Methods for Protocol Engineering and Distributed Systems*

solution. In fact, compared to [8] : **(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 7 deal with real-time systems. In Sect. 4, we describe how services and protocols are specified. 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 and 7, two examples illustrate our method. Finally, we conclude in Sect. 8.

2. PROTOCOL SYNTHESIS

We consider a distributed system (DS) consisting of several sites interconnected through a reliable communication medium (simply called *medium*). We assume that: (a) each pair of sites can communicate with each other through the medium; (b) 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 $Site_i$ corresponds a *protocol entity* PE_i.

In the *user's viewpoint*, the DS provides a service where only executions of primitives are visible. We assume that the specification of the desired service (provided to the user) defines the *ordering of* and *the timing requirements between* the occurrences of primitives. The timing requirements define the real-time properties of the DS.

The problem of the *designer* is then : *How* can he derive systematically specifications of the local 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.

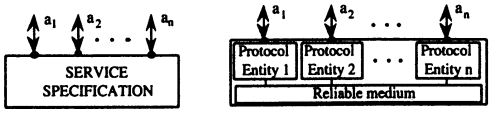


Figure 1. Service and protocol concepts

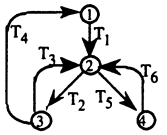
The approach we have selected for deriving protocols is called *Synthesis*, and the systems considered are assumed sequential. The basic principle we have used is the following: if in the specification of the service, a primitive

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 PE_b; (β) after PE_b receives m, it executes B.

3. PROTOCOL SYNTHESIS FOR NON-REAL-TIME APPLICATIONS

Service specification : A service desired by the user is described by a finite state automaton ($\mathcal{FSA}$), denoted SS, which specifies the sequences of primitives the user would like to observe at the various $\mathcal{SAP}$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 $T_p=[q,E_a,r]$. In a figure representing an $\mathcal{FSA}$, every transition $T_p=[q,E_a,r]$ may be simply represented by $T_p = E_a$ when 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. □



$T_1 = [1,A_1,2]$ $T_2 = [2,B_3,3]$ $T_3 = [3,C_2,2]$
$T_4 = [3,B_4,1]$ $T_5 = [2,B_1,4]$ $T_6 = [4,C_4,2]$

Figure 2. Service specification

Protocol specification : A PE_a is described by an $\mathcal{FSA}$, denoted PS_a (see for example Fig. 5), which specifies the sequences of events occurring at $Site_a$. There are three types of events in each PS_a.

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

Type S (for Send): The sending of a message is denoted $s_a^i(p)$, and means "message containing by 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 containing p and coming from PE_i is received by PE_a".

Let PE_1, PE_2,..., PE_n be 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 PEs*) The *combined behaviour* of PE_1, ..., PE_n is the behaviour of $\mathcal{DS}$. The specification DS of this behaviour can be computed from PS_1,..., PS_n. 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 : **(i)** $Proj_{\Lambda}(A)$ denotes the projection of an *FSA* A into an alphabet Λ; as an example, $Proj_{V_S}(DS)$ specifies the service provided to the user by *DS*; **(ii)** $A \cong B$ means that the *FSAs* A and B accept the same language; **(iii)** $A < B$ 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 service, generates specifications of protocol entities which are syntactically and semantically correct.

Principle for deriving PEs : From an SS specifying a desired service, deriving a protocol consists of generating an *FSA* PS_i for each $Site_i$, which specifies the action sequences executed at $Site_i$. In order to provide the service, the different *PEs* exchange messages through a medium. The basic principle used for deriving a protocol (see Sect. 2) has been applied in [8] 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 two parameters: *p* which identifies the executed transition T_p; and *q* which identifies the chosen transition T_q to be executed next. All PE_{bi}s will receive m, but only one of them will execute the primitive corresponding to the selected transition T_q.

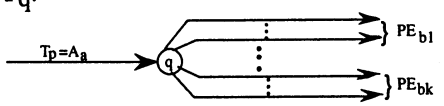


Figure 3. Choice between several actions

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 PE.* □

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

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

Compared to [8,10], 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 PE which, in turn, selects one of its primitives.

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

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 number of the transition in SS which is associated to the first of the two primitives. The obtained specification is denoted GPS.

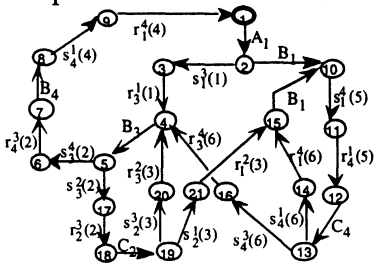
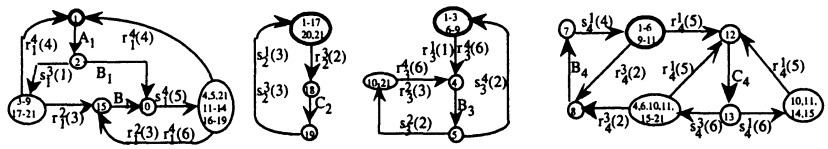


Figure 4. GPS obtained in Step 2 for the example represented in Fig. 2

Step 2 : From GPS, the specification PS_i of each PE_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 [8,10], besides being optimal. 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.



(a) PS_1 (b) PS_2 (c) PS_3 (d) PS_4

Figure 5. Obtained protocol specifications

Contrary to [10], the rules for deriving PEs do not depend on whether primitives are upward or downward. In our opinion, such a distinction

complicates uselessly the rules of Step 1. In the remaining part of this paper, we extend the procedure of protocol derivation to real-time systems.

4 TIMED AUTOMATA TO SPECIFY SERVICES AND PROTOCOLS

We use a *dense-time* model [1,8] where the time is viewed as a state variable that ranges over a dense domain and evolves indefinitely. The timed automata ($\mathcal{TA}$) model we propose here uses a variable **v** and a clock **c**.

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 $a \leq b$. 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; and **(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 *u*th 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 $Tr1=[q1,\sigma1,q;C1,v1]$, $Tr_2=[q2,\sigma2,q;C2,v2]$, $Tr_3=[q,\sigma3,r1;C3,v3]$, and $Tr_4=[q,\sigma4,r2;C4,v4]$. 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 $Tr=[q,\sigma1,r;C,v]$, 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, $C3=(I3_1,I3_2)$, $C4=(I4_1,I4_2)$, $I3_1=[1;2]$, $I3_2=[0;2]$,

$I4_1$=[1;3] and $I4_2$=[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 $I3_1$ (resp. $I4_1$);
  * 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 $I3_2$ (resp. $I4_2$);
  * If Tr_3 does not occur after a delay within [0;2], then Tr_4 *must* occur after a delay within [2;5].

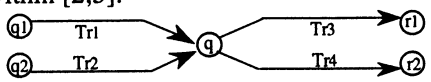


Figure 6. Incoming and outgoing transitions

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) 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 iff ($c \in I2_2$) and Tr_5 is enabled iff ($c \in I5_2$) (intervals $I2_2$ and $I5_2$ of $C2$ and $C5$ are used because **v**=2). Therefore, if "I $\leftarrow \langle Tr_a, Tr_b \rangle$" means "the delay of Tr_a and Tr_b falls within the interval I" then : $I1 \leftarrow \langle Tr_4, Tr_1 \rangle$ $I2_1 \leftarrow \langle Tr_1, Tr_2 \rangle$ $I2_2 \leftarrow \langle Tr_3, Tr_2 \rangle$ $I2_3 \leftarrow \langle Tr_6, Tr_2 \rangle$ $I3 \leftarrow \langle Tr_2, Tr_3 \rangle$ $I4 \leftarrow \langle Tr_2, Tr_4 \rangle$ $I5_1 \leftarrow \langle Tr_1, Tr_5 \rangle$ $I5_2 \leftarrow \langle Tr_3, Tr_5 \rangle$ $I5_3 \leftarrow \langle Tr_6, Tr_5 \rangle$ $I6 \leftarrow \langle Tr_5, Tr_6 \rangle$.

A PE_a is described by a $\mathcal{TA}$ denoted PST_a, which specifies: **(a)** the sequences of 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*. Examples of $\mathcal{TA}$s specifying PEs are given in Sect. 6 and 7.

We consider PE_1, PE_2 , ...,PE_n which are specified by PST_1, PST_2, ..., PST_n, respectively. Let $\mathcal{DS}$ be 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 $0 < t_1 < t_2 < ... < t_i < ...$ 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 defined similarly to the projection of an $\mathcal{FSA}$. 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*) 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.

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 $Tr=[q1,\sigma,q;C,v]$ and $Tr_k=[r,\sigma k,q2;C_k,v_k]$ 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 $Ik_v=[\gamma k_v;\delta k_v]$ of C which, for simplicity, is denoted $I_k=[\gamma_k;\delta_k]$.

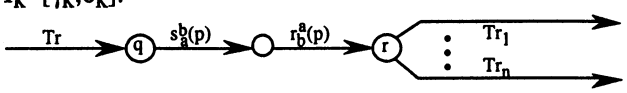
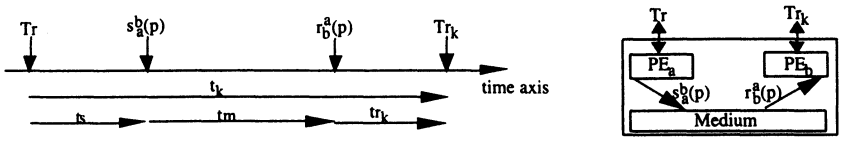


Figure 7. Outgoing transitions on a state of GPST



(a) Representation in function of time (b) Representation by entities

Figure 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].

Condition for 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, we must have :

$$\text{for } k= 1, 2, ..., n : \quad \delta_k - \rho_{a,b} \geq \sup(\gamma_k - \mu_{a,b}; 0) \qquad (1)$$

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 [8]. For resolving the timing constraints of the *PEs*, we consider the following three cases :

Static case : messages transmitted by *PEs* contain no temporal information;

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

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

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

Static case : we assume that the intervals S and R_k 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 :

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

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

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

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

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

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

Intuitively, modifying ψ and ξ allows to "move" some timing constraints between two communicating entities.

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 :

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

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

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.

Second dynamic case : in comparison with the first dynamic case, we assume 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 the previous case; ω_k and τ_k are calculated by PE_b as follows :

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

$$\tau_k(ts+tm) = \sup(\gamma_k - (ts+tm); 0) \quad (11)$$

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.

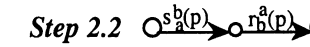
We note that ts and ts+tm , which are transmitted in the dynamic cases, are a *relative* temporal information. This implies that a global clock is not necessary. We also note that the temporal requirements of the protocol obtained using the approach in [8] are more restrictive than those derived by our improved approach.

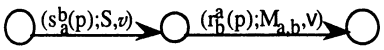
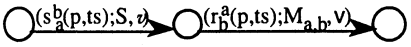
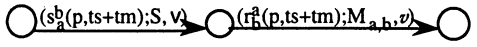
Derivation Procedure : it consists of three steps. Step 1, which generates a specification GPST, is similar to step 1 of the non-real-time case.

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. 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;
* Intervals $X_k=I_k-M_{a,b}$, k=1,...,n, in the first dynamic case;

Step 2.2  becomes: (v being the value of **v** which is set by the transition preceding $s^b_a(p)$)

- In the static case : 
- In the first dynamic case : 
- In the second dynamic case : 

Informally: - the delay between Tr and $s^b_a(*)$ falls within $S=[\theta;\phi]$;

- the delay between $s^b_a(*)$ and $r^a_b(*)$ falls within $M_{a,b}$.

Step 2.3 For each k=1, ...,n, the vth interval I_k of $\mathcal{C}_k$ 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 GST obtained at Step 2.3 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 $r^a_b(p)$ and Tr_k must belong to: - the interval R_k in the static case;

- an interval $R_k(ts)$ which depends on the interval $X_k=I_k-M_{a,b}$ and on ts ;
- an interval $R_k(ts+tm)$ which depends on the 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 $[0;\infty]$.

In Sect. 6 and 7, ψ and ξ are taken to be equal to 0.5, which means that the temporal constraints are equally distributed between the two sites.

6. SYNTHESIS OF A TIMED X.25 PROTOCOL

We consider the X.25 service [2]. 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 [4].

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

Specification of the Simplified X.25 Service : Our specification contains two blocs $S_{1,2}$ and $S_{2,1}$, where $S_{i,j}$ (see Fig. 9) models the service when $Site_i$ and $Site_j$ are the sender and the receiver, respectively. The specification of the simplified X.25 is schematized in Fig. 10. The event $Token_i^j$ means that "$Site_i$ gives to $Site_j$ the possibility to establish a connection". We assume that State $1_{1,2}$ is the initial state of the service.

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. 9 and 10, with the intervals defining the temporal constraints.

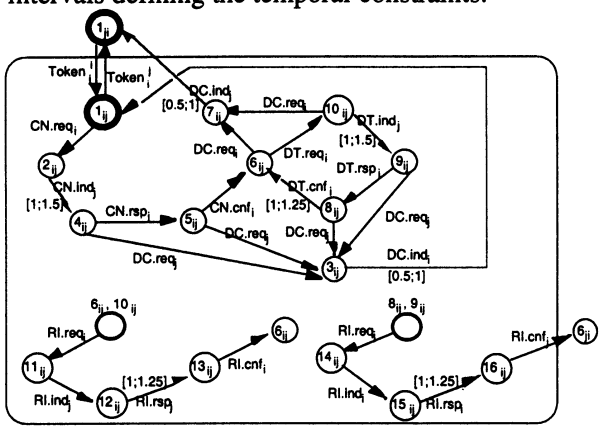


Figure 9. Block $S_{i,j}$

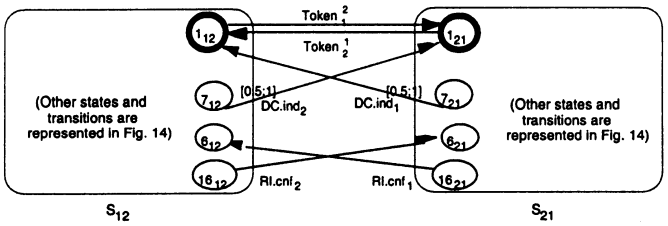


Figure 10. Service specification of the simplified X.25

Protocol Synthesis : The synthesized specifications of the two PEs can be represented by the single $\mathcal{TA}$ of Fig. 11, and where i identifies the PE described 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. 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 $Site_1$. To simplify, we give only the results of the static case. The synthesized temporal constraints, which are defined by interval for the transitions represented in grey in Fig. 11, are the following.

$\underline{Site_1}$: 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]; constraint for $s_1^2(p7_1)$ is [0.25;0.25]; constraint for $s_1^2(p8_1)$ and $s_1^2(p9_1)$ is [0.25;0.375]; constraint for $DT.cnf_1$ is [0.375;0.375]; constraint for $DT.ind_1$ is [0.375;0.5]; constraint for $DC.ind_1$ is [0.125;0.25]; constraint for $RI.rsp_1$ is [0; 0.5].

Site$_2$: 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]; constraint for $s_2^1(p7_2)$ is [0.375;0.375]; constraint for $s_2^1(p8_2)$ and $s_2^1(p9_2)$ is [0.375;0.5]; constraint for $DT.cnf_2$ is [0.25;0.25]; constraint for $DT.ind_2$ and $CN.ind_2$ is [0.25;0.375]; constraint for $DC.ind_2$ is [0;0.125]; constraint for $RI.rsp_2$ is [0;0.5].

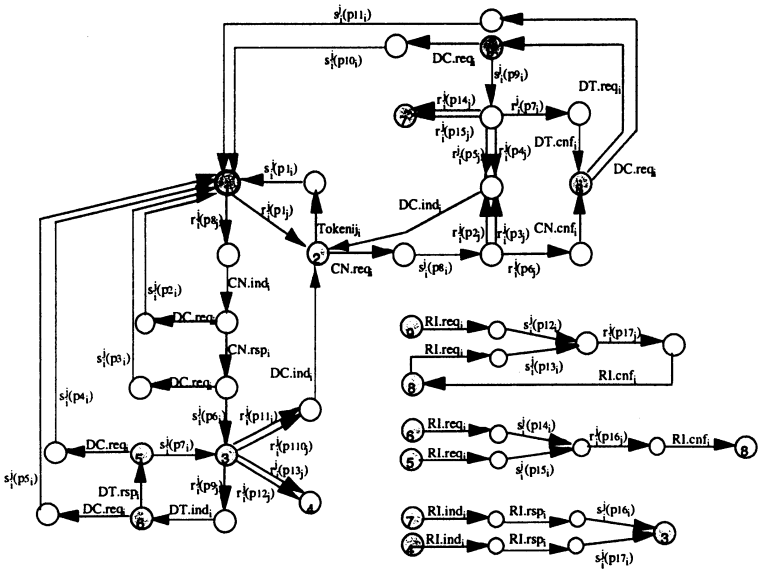


Figure 11. Specification of the synthesized X.25 protocol at Site$_1$

7. EXAMPLE ASSEMBLY SYSTEM

The following example 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 [9].

Protocol entities : There are six PEs, R1, R2, R3, C1, C2 and C3, which correspond to the three robots and the three carpets, respectively. Table T is not considered as an entity since it is passive. A piece is denoted Pi.

Service primitives : They are the following (i=1, 2, 3, and j=1, 2):
$MOVE.Carpet_{Ci}$: Ci is actuated; $ARRIVED.Piece_{Cj}$: Ci has detected that a

Pi has reached its destination ; STOP.Carpet$_{Ci}$: Ci is stopped ; CHECK.Piece$_{Rj}$: Rj begins to check Pj ; TAKE.Piece$_{Rj}$: Rj takes a Pj from Cj ; TAKE.Piece1$_{R3}$: R3 takes a P1 from C1 ; TAKE.Piece2$_{R3}$: R3 takes a piece P2 from C2 ; TAKE.AssPieces$_{R3}$: R3 takes an assembled piece from table T ; PUT.Piece$_{R1}$: R1 puts a P1 on T ; PUT.AssPieces$_{R2}$: R2 puts an assembled piece on C3; ASS.Pieces$_{R2}$: R2 assembles P1 and P2.

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

Step 1 : C1 is actuated (MOVE.Carpet$_{C1}$)
Step 2 : P1, which is on C1, reaches its destination (ARRIVED.Piece$_{C1}$)
Step 3 : C1 is stopped (STOP.Carpet$_{C1}$)
Step 4 : 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 : C2 is actuated (MOVE.Carpet$_{C2}$)
Step 9 : P2, which is on C2, reaches its destination (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.*

Temporal constraints added to the service

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

- Between TAKE.Piece1$_{R3}$ (or TAKE.AssPieces$_{R3}$ or STOP.Carpet$_{C3}$) and MOVE.Carpet$_{C1}$: [6;10]; (i=1,2)

The service specification is represented in Fig. 12, where transition Tr_i corresponds to Step i of the scenario. The temporal constraint of each transition is defined by a single interval (which is shown in Fig. 12).

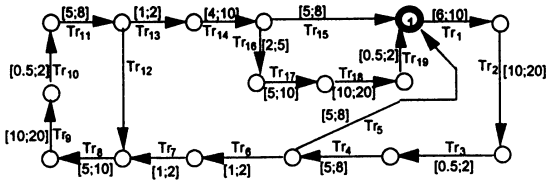
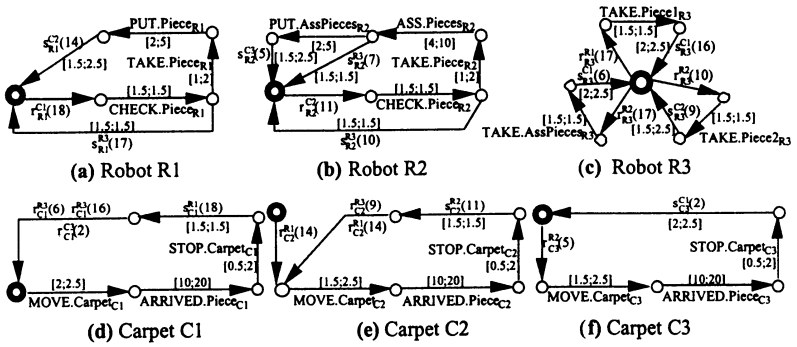


Figure 12. Service specification of the assembling system

Protocol Synthesis : The specifications synthesized in the static case are represented in Fig. 13, and consist of six $\mathcal{TA}$s modeling the 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].



(a) Robot R1 (b) Robot R2 (c) Robot R3

(d) Carpet C1 (e) Carpet C2 (f) Carpet C3

Figure 13. Synthesized specifications of the robots and the carpets in the static case

8. CONCLUSION

A method for deriving real-time protocols [8] is improved and extended in the present paper. Our method imposes requires that the service is sequential. We are presently investigating an approach using the following three steps : (a) the given service specification S is transformed into a sequential service, called S^{seq}; (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; (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 and 7 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] R.J. Deasington, "*Protocoles X.25 pour les réseaux à commutation de paquets,*" Masson, 1987.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.