NEW RESULTS ON DERIVING PROTOCOL SPECIFICATIONS FROM SERVICE SPECIFICATIONS+

Ferhat KHENDEK, Gregor von BOCHMANN,
Christian KANT*

Département d'IRO, Université de Montréal,
C.P. 6128, Succursale A
Montréal, Québec, H3C 3J7, Canada

Abstract

Previous papers describe an algorithm for deriving a specification of protocol entities from a given service specification. A service specification defines a particular ordering for the execution of service primitives at the different service access points using operators for sequential, parallel and alternative executions. The derived protocol entities ensure the correct ordering by exchanging appropriate synchronization messages, between one another through the underlying communication medium.

This paper presents several new results which represent important improvements to the above protocol derivation approach. First the language restriction to finite behaviors is removed by allowing for the definition of procedures which can be called recursively. Secondly, a new derivation algorithm has been developed which is much simpler than the previous one. Third, the resulting protocol specifications are much more optimized than those obtained previously.

* Christian Kant is with the Université de Moncton,

+ Partial support from the Natural Sciences and Engineering Research Council of Canada, and The Ministry of Education of Québec is gratefully acknowledged.

1. Introduction

The service concept has acquired an increasing level of recognition by protocol designers (see e.g. [ViLo 85]). This architectural concept influences the methodology applied to service and protocol definition. Since the protocol can be seen as the logical implementation of the service, one may ask the question whether it is possible to formally derive the specification of a protocol providing a given service.

An architectural model for both service and protocol specification is depicted in Figure 1. A service is realized by a service provider which - according to the principle of abstraction - is seen as a black box, and made available through, a certain number of service access points (SAPs, see Figure 1(a)), in the following also called "places". In the more detailed view of the protocol specification, some internal structure is given to the black box; several protocol entities linked by an underlying transmission medium may cooperate to provide the service (Figure 1(b)). We assume the communication medium to be reliable, to maintain the sending sequence of messages and to be connected to each entity by FIFO-queues for transmissions and receptions.

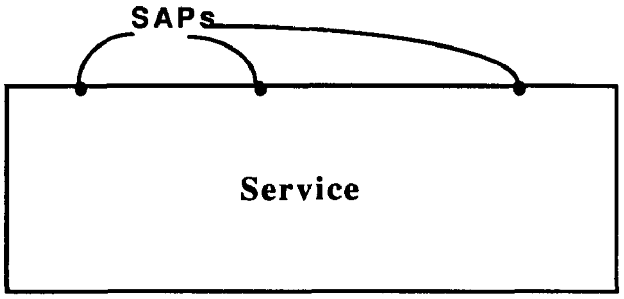

Figure 1(a): Service architecture

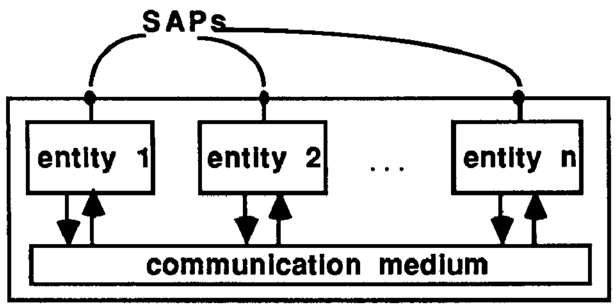

Figure 1(b): Protocol architecture

Based on this architectural model, we can phrase the above question in more precise terms as follows: Given a service specification e_S (see Figure 2(a)), is it possible to formally derive the specifications $T_i(e_S)$ for all protocol entities (see Figure 2(b)) ?

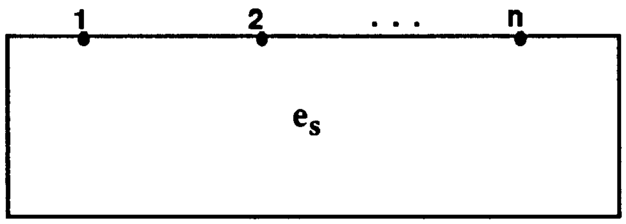

Figure 2(a): Service specification

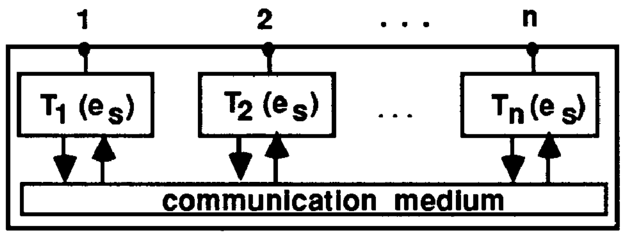

Figure 2(b): Derived protocol specification

In the area of communication protocols, analysis techniques have been developed and applied to detect design errors like deadlocks, unspecified receptions, non executable interactions, state ambiguities and non-conformance with the service specification. The best-known approach here seems to be reachability analysis, usually based on the specification of protocol entities as finite state automata (see for instance [Boch 87c]). Because the analysis of a sufficiently complex protocol specification usually reveals some of the above design errors, the specification has to be revised and the analysis to be repeated until no more errors are found.

With protocol synthesis one wants to avoid errors a priori. Existing approaches (see e.g. [Zafi 80], [MeBo 83], [GoYu 84], [Rama 85] and [Rama 86]) take partly specified protocol entities or complete specifications of some protocol entities as a starting-point for the synthesis procedure. The synthesis procedure is based on the duality inherent in message exchange: For each message sent by a protocol entity, there must be a protocol entity prepared to receive it. However, several important limitations apply to each of these approaches:

- With the exception of [MeBo 83], the service specification is not taken into account. There is no formal requirement on which the synthesis is based, instead it starts with part of the solution which has to be provided in advance. It is clear that - without a formal service definition - conformance with the service is not guaranteed by the synthesis algorithm and has to be shown in a separate step.

- Again with the exception of [MeBo 83], only two-party protocols are considered. It seems to be difficult to extend the approaches to an arbitrary number of protocol entities. Thus they are not well-suited for high-level protocols involving more than two parties.

- [Zafi 80], [GoYu 84] and [Rama 85] all assume the existence of a reliable communication medium. The latter, however, is extended in [Rama 86] to cover noisy channels.

- None of the above approaches takes parameters into account. Only a distinction between different message types is possible.

- [Zafi 80] and [MeBo 83] do not avoid deadlocks by construction.

- All approaches assume the existence of (incomplete) protocol specifications. None is based solely on the service definition.

- [Zafi 80] and [MeBo 83] are quite expensive with respect to computation.

Our approach, introduced in [Boch 86g] and extended in [Gotz 89], is more general in that only the existence of the service specification (see Figure 2(a)) is required. It can handle an arbitrary number of protocol entities. Furthermore, input and output parameters are taken into account. Subsystem failures and unreliable channels, however, are not taken into account. A similar approach was also taken in [Chu 88a] in the context of finite state two party protocols and extended to operate in the presence of message loss [Chu 88b].

In our approach to the derivation of a protocol specification from a given service specification, an assignment of the service interactions to the different service access points must be given; the derivation algorithm then provides specifications of all protocol entities serving the different access points. The algorithm has been implemented in Prolog together with translations between a subset of the Lotos specification language [Bolo 87] and our service specification language [Khen 89].

However, this derivation method has also some limitations. One is the assumption of a reliable communication medium. The other is a certain restriction on the power of the language used for the specification of service specifications. Only finite behaviors have been considered so far. The language of [Boch 86g] and [Gotz 89] only includes the operators ";" for sequential execution, "[]" for alternatives, and "|||" for independent parallelism.

This paper presents several new results which represent important improvements to the above protocol derivation approach. First of all, the language restriction to finite behaviors is removed by allowing for the definition of procedures which can be called recursively. Secondly, a new derivation algorithm has been developed which is much simpler than the one presented in [Boch 86g]. Third, the resulting protocol specifications are much more optimized than those obtained previously. Section 2 presents the new derivation algorithms, including recursive definitions, which is much easier to understand than the algorithm described in [Boch 86g] and [Gotz 89]. The optimization of the protocol specifications is discussed in Section 3. Section 4 contains the conclusions.

2. The derivation algorithm

2.1. Specification language

The protocol derivation algorithm is defined in terms of the language constructs that can be used to write a service specification which is the starting point for the derivation of the protocol. For the discussion in this paper, we adopt the specification language defined by the following syntax rules:

(1)	Service-Def	$\rightarrow$	Proc-Def
(2)	Proc-Def	$\rightarrow$	PROC Proc-Id = e END Proc-Def_1
(3)	Proc-Def	$\rightarrow$	PROC Proc-Id = e END
(4)	e	$\rightarrow$	Proc-Id
(5)	e	$\rightarrow$	Event-Id
(6)	e	$\rightarrow$	e_1 ; e_2
(7)	e	$\rightarrow$	e_1 ||| e_2
(8)	e	$\rightarrow$	e_1 [] e_2
(9)	e	$\rightarrow$	(e_1)

(* $e_1 = e_2 = e$ and Proc-Def_1 = Proc-Def *)

As in Lotos[Bolo 87] for a behavior expressions, operator priorities are given as follow, in decreasing order : ";" , "|||" and "[]". Parentheses may be used (rule (9): e $\rightarrow$ (e_1)) to enforce different priorities, or stress the predefined ones. "PROC" and "END" are keywords.

The event identifiers (rule (5) e $\rightarrow$ Event-Id) are of the form "$\text{Identifier}^{\text{Place}}$" where "Identifier" represents a primitive service interaction with the user of the service at a given service access point, and "Place" characterizes that service access point. In the following we use "Places" in the form of integer numbers and single characters to represent the interactions. For example, the event a^2 represents the service interaction "a" at the service access point "2".

Compared to [Gotz 89], the definition of several procedures (rules (1) through (3)) as well as the statement calling the execution of a procedure (rule (4)) are the extensions which make it possible to define infinite behaviors, such as the following example:

Example 1:

Service specification :

PROC A = (a^1 ||| b^2 ||| c^3) ; B END
PROC B = (e^3 ||| (c^3 [] d^3)) ; A END

The service is defined by the behavior of the first procedure, procedure "A" in this example. The behavior of these procedures is described below :

Procedure A:
The interaction primitives a^1, b^2 and c^3 are executed independently (with interleaving) at the places 1, 2 and 3, respectively. When the execution of these primitives is finished, the behavior is as specified for procedure B.

Procedure B:
First, the interaction primitive e^3 is executed at place 3 in interleaving with the primitive c^3 or d^3 (the choice between the interaction primitives c^3 and d^3 is done at place 3). Then the behavior is as specified for procedure A.

At the abstraction level of the service specification, only the interactions with the service user are defined. At the more detailed level of protocol specifications, in addition the exchange of messages between the protocol entities must be specified. These messages serve for synchronization in order to ensure the correct ordering of the service interactions, as well as for the transfer of information about interaction parameters which are exchanged with the users of the service. The purpose of the protocol derivation algorithm is to determine the order of message exchanges and service interactions to be executed by each of the protocol entities. This paper concentrates on the synchronization messages; issues related to interaction parameters and required messages are discussed in detail in [Gotz 89].

The resulting protocol specifications are written in the same language as the service specification, except that addition primitives for the exchange of protocol messages are introduced. We write "$s_j(m)$" for the sending of a synchronization message m to the protocol entity at place j. If this statement is executed by the protocol entity at place i, then the protocol entity at place j may later execute the statement "$r_i(m)$", which represents the reception of the message m from the entity at place i.

The following restrictions are imposed on the form of service specifications, in order to simplify the protocol derivation. For each subexpression of the form "e_1 [] e_2", contained in the service specification, the following conditions must be satisfied:

R1: All starting interactions of e_1 and all starting interactions of e_2 must be associated with the same place.

R2: The set of ending interaction places of e_1 and the set of ending interaction places of e_2 must be equal, unless one of these is empty.

Restriction R1 was already introduced in [Boch 86g]. It simplifies the implementation of the decision which alternative should be selected. Instead of using a distributed algorithm for this selection (e.g. [Rana 83]), the choice can be made locally by the protocol entity at the place where the alternatives start. Restriction R2 is the basis for the simplification of the derivation algorithm (see below).

The restrictions R1 and R2 can be eliminated from the specification language by introducing a preliminary step in the protocol derivation process. During this step, a "starting" and "ending" places are selected for each subexpression of the form "e_1 [] e_2", and some dummy interactions at these places are added at the beginning and end of each of the alternatives, if the existing interactions are not associated with the right places.

2.2. The basis for the simplification of the algorithm

The basic idea of the protocol derivation algorithm [Boch 86g] is the observation that the exchange of synchronization messages is only required for the sequencing operator ";". In fact, the subexpressions e_1 and e_2 of an expression "e_1 ; e_2" may involve service interactions at different places. It is important that the interactions belonging to e_2 do not start before all interactions belonging to e_1 have been executed. This synchronization can be obtained by sending synchronization messages from the places where interactions of e_1 are executed to those places where interactions of e_2 are to be executed. In the case of the operator "e_1 ||| e_2" the subexpressions e1 and e_2 are executed in parallel and independently from one another; no synchronization is required.

In order to determine which synchronization messages must be exchanged for an expression "e_1 ; e_2", it is necessary to know the places where the last interactions of e_1 are executed, and where the first interactions of e_2 are executed. For each subexpression e in the service specification, we call these sets EP(e), the set of ending places, and SP(e), the set of starting places of the subexpression. As shown in Figure 3, these sets can be associated as attributes with the nodes of the syntax tree of a given service specification. As the figure suggests, synchronization messages must be sent from all places of EP(e_1) to all places of SP(e_2).

In the case of service specifications satisfying the restrictions R1 and R2, the sets EP and SP may contain more than one place only when the parallel operator "|||" is involved in the expression. The set SP for an alternate expression of the form "e_1 [] e_2" contain only a single place, due to restriction R1, while the set EP may contain more than one place if the operator "|||" is involved. With the restriction R2, no distinction must be made whether several places in that set are due to parallelism or alternatives. This distinction makes the protocol derivation algorithm described in [Boch 86g] and [Gotz 89] much more complicated than the one presented here.

Another difference of the derivation algorithm leads to more optimized protocol specifications. While the earlier algorithm foresees synchronization messages sent between the "ending events of e_1" and the "starting events of e_2" for each sequential subexpression of the form "e_1 ; e_2", the here described algorithm foresees synchronization messages at a higher level in the syntax tree, passed directly between "subexpression e_1" and "subexpression e_2" (see Figure 3). This is possible because of restriction R2 above. As shown in Section 3, this can lead to an important reduction of the number of exchanged messages.

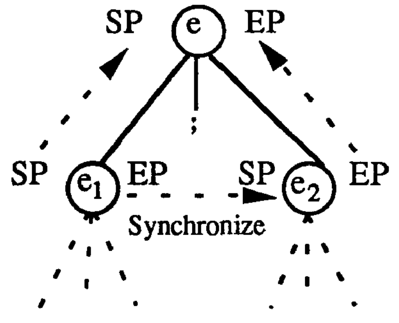

Figure 3

2.3. Attributes evaluation

The first phase of the protocol derivation algorithm is the evaluation of attributes defined on the derivation tree of the given service specification. The following attributes are associated with each expression node e in the syntax tree:

SP(e): the set of "starting places" as described above.

EP(e): the set of "ending places" as described above.

P(e): either NIL or a pair < PP, N > where PP is the set of "preceding" places, that is, the set of "ending places" EP of the subexpression "preceding" the expression e; and N is the number of the expression node which directly precedes the ";" operator in the service expression. This number is included in the synchronization messages and permits the receiving protocol entity to determine to which alternative a received message belongs [Boch 86g].

FP(e): the set of "following places", similar to the "preceding places" PP of the P attribute above.

The rules for evaluating the attributes of a node in the syntax tree depend on the syntactic rule applied at that node, as defined in the Table 1. The evaluation of the attributes can be considered in two phases. In the first phase, the attributes SP and EP are evaluated from to bottom towards the top of the tree (so-called synthesized attributes). For the leaf nodes generated by the rule (5) (e → Event-Id) the attributes SP and EP are both set to the value {place(Event-Id)}. Here "place" is a function from the set of events to the set of places: place(IdentifierP) = p. In the second phase, the attributes P and FP are first set, respectively, to NIL and {} at the root of each subtree corresponding to a behavior expression of a procedure (see rules (2) and (3)), then evaluated from the top down (so-called inherited attributes) for the intermediate nodes, using the values for SP and EP obtained during the first phase (see rule (6)).

Production rules			
(1)	Service-Def	→	Proc-Def
(2)	Proc-Def	→	PROC Proc-Id = e END Proc-Def$_1$
(3)	Proc-Def	→	PROC Proc-Id = e END
(4)	e	→	Proc-Id
(5)	e	→	Event-Id
(6)	e	→	e_1 ; e_2
(7)	e	→	e_1 ||| e_2
(8)	e	→	e_1 [] e_2
(9)	e	→	(e_1)

	SP	EP
(2)	SP(Proc-Id) := SP(e)	EP(Proc-Id) := EP(e)
(3)	SP(Proc-Id) := SP(e)	EP(Proc-Id) := EP(e)
(4)	SP(e) := SP(Proc-Id)	EP(e) := EP(Proc-Id)
(5)	SP(e) := SP(Event-Id)	EP(e) := EP(Event-Id)
(6)	**SP(e) := SP(e_1)**	**EP(e) := EP(e_2)**
(7)	SP(e) := SP(e_1) ∪ SP(e_2)	EP(e) := EP(e_1) ∪ EP(e_2)
(8)	SP(e) := SP(e_1) = SP(e_2)	EP(e) := EP(e_1) ∪ EP(e_2)
(9)	SP(e) := SP(e_1)	EP(e) := EP(e_1)

	P	FP
(2)	P(e) := NIL	FP(e) := {}
(3)	P(e) := NIL	FP(e) := {}
(4)	P(Proc-Id) := NIL	FP(Proc-Id) := {}
(5)	P(Event-Id) := NIL	FP(Event-Id) := {}

(6)	**P(e_1) := NIL**
	P(e_2) := <EP(e_1), N(e_1)>
		FP(e_1) := SP(e_2)
		FP(e_2) := {}

(7)	P(e_1) = P(e_2) := NIL	FP(e_1) = FP(e_2) := {}
(8)	P(e_1) = P(e_2) := NIL	FP(e_1) = FP(e_2) := {}
(9)	P(e_1) := NIL	FP(e_1) := {}

Table 1: Evaluation rules for the attributes SP, EP, P, FP

The above attribute evaluation rules are similar to those of [Boch 86g], however, the following differences can be noted. The attributes above are essentially sets of places with a straightforward meaning, while the attributes in [Boch 86g] represent statements for the sending and receiving of synchronization messages and which must satisfy certain syntactic properties. Effective values for the attributes P and FP are only used in relation with the sequential composition operator. The most important difference is the presence of recursive procedure calls which requires an iterative solution to the evaluation of the attributes during the first phase.

The attributes SP and EP of a leaf node corresponding to a procedure call (generated by rule (4)) can be considered variables. We equate the values of these attributes for a call of a particular procedure A with the attributes obtained for the node representing the definition of this procedure A, that is, the root of the subtree starting with the expression e generated by rule (2) or (3) with the value of Proc_Id equal to A. Therefore, the evaluation rules for SP applied to a given procedure subtree, give rise to an equation defining SP for that procedure in terms of constant places corresponding to the explicit events defined in that procedure, and in terms of variables representing the SP values of those procedures which are called; and similarly for EP. These equations, which are recursive in general, can be solved by applying the rule that the equation SP(A) := SP(A) $\cup$ X implies the equation SP(A):=X, where SP(A) is the value of the SP attribute for procedure identified by A.

Another way of solving the recursive equations is by iteration. The bottom-up attribute evaluation pass over the procedure subtrees is performed several times, each representing a step of the iteration. For the first step, the values of the SP and EP attributes of the procedure call leaf nodes are set to the empty set. In each subsequent step, the values of these attributes are set equal to the corresponding values obtained at the procedure root nodes in the previous step. The iteration terminates when the attribute values of all procedure root nodes have not changed during the last step.

Based on the attributes SP and EP, we can now formally define the restrictions R1 and R2 for the rule (8)
$(e \rightarrow e_1 \; [] \; e_2)$:

R1 : $SP(e_1) = SP(e_2) = \{p\}$, where p is an arbitrary place
R2 : $EP(e_1) = EP(e_2)$ or $(EP(e_1) = \{\}$ or $EP(e_2) = \{\})$

For Example 1 described in section 2.1, the behavior of each procedure is described by a syntax subtree as is shown Figure 4. The application of the rules described in Table 1 leads to the values shown in the figure. Some of the attributes involve the variables SP(A), SP(B), EP(A) and EP(B). In order to determine these variables, we proceed as follows. The evaluation rule for SP at the root node of the A subtree reads SP(A) := SP(e) and by inspecting this subtree we see that SP(e) is equal to {1, 2, 3}. Therefore SP(A) = {1, 2, 3}. Similarly, we get EP(A) := EP(B), SP(B) = {3} and EP(B) := EP(A). Finally, this leads to the equation EP(A) := EP(A), which can be written
EP(A) := EP(A) $\cup$ {}, and therefore we set EP(A) = {}.

These are obviously the values we were expecting, because the two procedures A and B do not terminate.

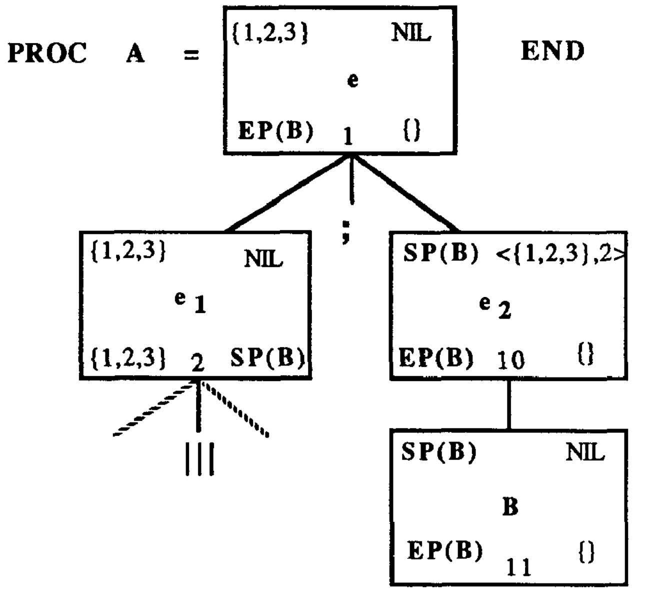

Figure 4(a): Syntactic tree of the procedure A

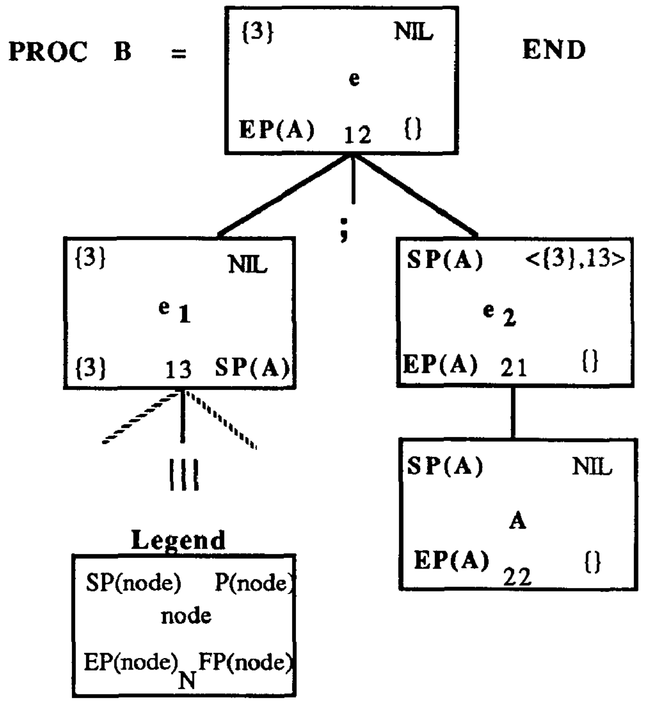

Figure 4(b): Syntactic tree of the procedure B

2.4. Protocol derivation

Once the attributes are evaluated, they are used to derive the specifications of the protocol entities as follows. For each service access point, identified by the place p, the specification of the protocol entity at that place is obtained by applying the function Tp, defined in Table 2, to the root node of the service specification. The result is a character string in the form of a specification containing a set of procedures with the same identifies as in the original service specification, however, the definition of their bodies is changed. Only the service interactions occurring at the place for which the protocol entity is derived will be included in the protocol specification (see rule (5)), and additional statements for the sending and receiving synchronization messages will be included (see rule (6)). The string "empty" represents no action and can be eliminated using the rules described in [Boch 86g].

The functions trans and rec are used to obtain statements for the sending and receiving of synchronization messages from the set of "following" and "preceding" places, respectively. They are defined as follows:

rec (NIL) = "empty"
rec (<{i, j, ..., k}, N>) = "($r_i(N)$ ||| $r_j(N)$ ||| ... ||| $r_k(N)$)"

trans ({}, N) = "empty"
trans ({i, j, ..., k}, N) = "($s_i(N)$ ||| $s_j(N)$ ||| ... ||| $s_k(N)$)"

Production rules			
(1)	Service-Def	→	Proc-Def
(2)	Proc-Def	→	PROC Proc-Id = e END Proc-Def$_1$
(3)	Proc-Def	→	PROC Proc-Id = e END
(4)	e	→	Proc-Id
(5)	e	→	Event-Id
(6)	e	→	e_1 ; e_2
(7)	e	→	e_1 ||| e_2
(8)	e	→	e_1 [] e_2
(9)	e	→	(e_1)

Function Tp	
(1)	Tp(Service-Def) := Tp(Proc-Def)
(2)	Tp(Proc-Def) := "PROC" Proc-Id "=" Tp(e) "END" Tp(Proc-Def$_1$)
(3)	Tp(Proc-Def) := "PROC" Proc-Id "=" Tp(e) "END"
(4)	Tp(e) := Proc-Id
(5)	Tp(e) := if place(Event-Id) = p then Event-Id else "empty"
(6)	Tp(e) := Tp(e_1) ";" if $p \in$ EP(e_1) then **trans(FP(e_1),N(e_1))** else "empty" ";" if $p \in$ SP(e_2) then **rec(P(e_2))** else "empty" ";" Tp(e_2)
(7)	Tp(e) := Tp(e_1) "|||" Tp(e_2)
(8)	Tp(e) := Tp(e_1) "[]" Tp(e_2)
(9)	Tp(e) := "(" Tp(e_1) ")"

Table 2: Definition of the function Tp

Consider again Example 1 with the service specification:

PROC A = (a^1 ||| b^2 ||| c^3) ; B END
PROC B = (e^3 ||| (c^3 [] d^3)) ; A END

This derivation algorithm leads to the following protocol specification:

Protocol entity at place 1:
PROC A = a^1 ; $s_3(2)$; B END
PROC B = $r_3(13)$; A END

Protocol entity at place 2:
PROC A = b^2 ; $s_3(2)$; B END
PROC B = $r_3(13)$; A END

Protocol entity at place 3:
PROC A = c^3 ; $s_3(2)$; ($r_1(2)$ ||| $r_2(2)$ ||| $r_3(2)$) ;
B END

PROC B = (e^3 ||| (c^3 [] d^3)) ;
($s_1(13)$ ||| $s_2(13)$ ||| $s_3(13)$) ;
$r_3(13)$; A END

3. Optimizations

Different kinds of optimizations lead to different protocol designs depending on the performance objective to be optimized. In the context of the here described protocols, the following two objectives may be considered, as illustrated in the example Bellow:

(1) Minimization of the number of synchronization messages required for a typical execution of the service specification. In fact, different executions should be considered if the service specification allows for alternative choices.

(2) Minimization of the maximal number of synchronization messages transmitted in a sequential order during a typical execution of the service specification.

The first objective minimizes the message transmission "overhead", while the second minimizes the overall delay required for the execution of the service, assuming that the delay is solely due to message transmission, and the delays of all message transmissions are the same. In the following we only consider the first optimization objective.

3.1. Elimination of loopback messages

Let consider the derived protocol specification for Example 1, in particular the protocol entity for the place 3 above. One sees that the protocol specification includes synchronization messages sent by the entity at place 3 to itself. Clearly, such messages should be eliminated, since they do not provide any synchronization function. (Note that the sequencing relation is locally enforced by the ";" operator which is included in the specification of the entity).

This optimization is obtained by eliminating the place p of the entity to be generated, from the sets of places used for the generation of sending and receiving statements. More precisely, the expressions "rec ($P(e_2)$)" and "trans($FP(e_1)$, $N(e_1)$)" in Table 2 (rule (6)) should be replaced by "rec ($P'(e_2)$)" and "trans($FP'(e_1)$, $N(e_1)$)", respectively, where P' and FP' are defined as follows:

$P'(e) = NIL$ if $P(e) = NIL$
$P'(e) = < Set - \{p\}, N >$ if $P(e) = < Set, N >$

$FP'(e) = FP(e) - \{p\}$

In the case of Example 1 above, this modification leads to the specification :

PROC A = c^3 ; ($r_1(2)$ ||| $r_2(2)$) ; B END

PROC B = (e^3 ||| (c^3 [] d^3)) ;
($s_1(13)$ ||| $s_2(13)$) ;
A END

for the protocol entity at place 3, which exhibits no useless transmissions, while the specifications for the others entities are not changed.

3.2. Other optimizations

As mentioned in Section 2.2, the derivation algorithm presented here gives rise to more optimized protocol specifications than the previous algorithm [Boch 86g]. The difference becomes visible for service specifications such as the following:

Example 2:
PROC A = (a^1 ||| b^1) ; (c^2 ||| d^2) END

The algorithm of [Boch 86g] gives rise to four synchronization messages, as indicated in Figure 5(a). An optimization described in [Khen 89] reduces this number to two, as shown in Figure 5(b). Note that this optimization is also applicable in the case that restriction R2 is not satisfied. For the simplified derivation algorithm described in this paper (which requires restriction R2) the resulting protocol includes only a single synchronization message, as indicated in Figure 5(c). As indicated in the figure, the message relates the higher nodes of the syntax tree, instead of the leaf nodes, as is the case of the earlier algorithm. This is the reason for the optimization.

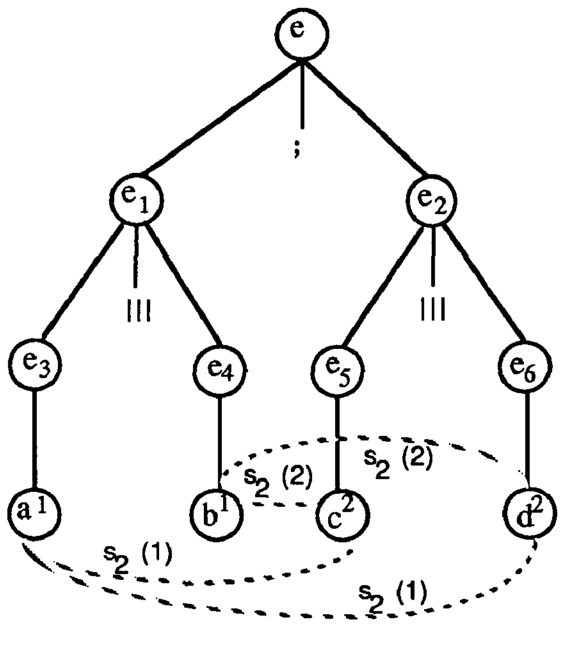

Figure 5(a)

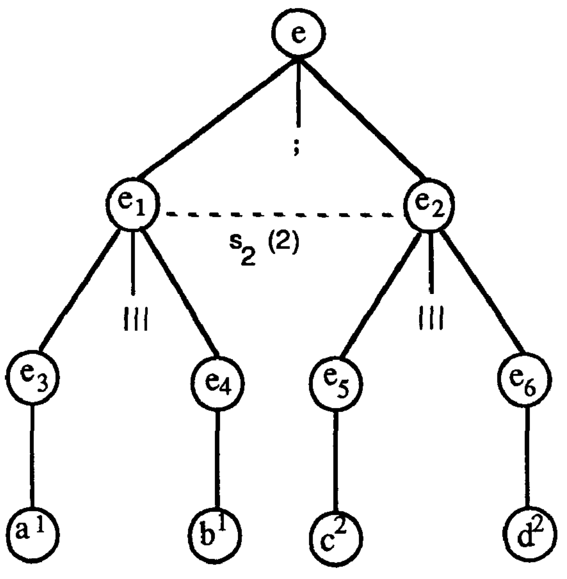

Figure 5(c)

When interaction parameters are taken into account, another optimization may be performed [Khen 89]. In this case, instead of sending a synchronization message and a data message together from the place p_1 to the place p_2, the synchronization message may be eliminated and the protocol entity at place p_1 has to send just the data message to place p_2. This data message will also serve for synchronization.

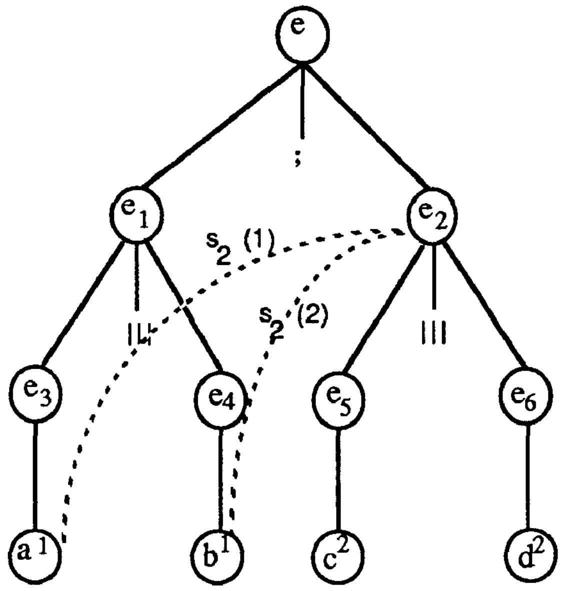

Figure 5(b)

4. Conclusions

This paper considers the automatic derivation of a protocol specification from a given service specification. The derived protocol automatically satisfies the usual "nice properties", such as absence of deadlocks, unspecified receptions, etc. In addition, it provides the service interactions at the different service access points in the order specified by the given service specification. The necessary synchronization messages between the different protocol entities are determined automatically.

The paper presents a simplified derivation algorithm, which is much easier to understand than an earlier one [Boch 86g]. In addition, it provides protocol specifications which are better optimized. Another important different is the extension of the algorithm to handle service specifications with loops and recursive procedure calls. Together with the other operators ";" (sequential execution), "[]" (alternatives), and "|||" (independent parallelism), this allows for a language power similar to Lotos or CCS [Miln 80].

The inclusion of interaction parameters in the context of the earlier derivation algorithm is discussed in [Gotz 89]. It involves additional messages exchanged between the protocol entities for the transmission of interaction parameters. The same approach can be used for handling interaction parameters in the context of the here described derivation algorithm.

The protocol derivation algorithm assumes that the underlying communication is reliable. This assumption is usually satisfied for protocols above the level of the OSI Transport protocol. We believe that this algorithm could be useful for automating the design of application protocols, in a context where the service specifications change frequently.

For the case that the protocol should be reliable in the case of message losses, it is conceivable to use an approach where the protocol for the reliable case is systematically transformed into a protocol that recovers from message loss. We think that it may be possible to adapt the approach of [Rama 86] to the here described context. More research is still required to work out the details.

References

[Boch 86g] BOCHMANN, G.v. and GOTZHEIN, R. Deriving protocol specifications from service specifications. in: Communications, Architectures & Protocols, Proceedings of the ACM SIGCOMM '86 Symposium, Vermont, USA, 1986.

[Boch 87c] BOCHMANN, G.v. Usage of protocol development tools : the result of a survey. 7-th IFIP Symposium on Protocol Specification, Testing and Verification, Zurich, May 1987.

[Bolo 87] BOLOGNESI, T. and BRINKSMA, E. Introduction to the ISO Specification Language Lotos. Computer Networks and ISDN Systems, vol. 14, no.1, pp. 3- , 1987.

[Chu 88a] CHU, P.M. and LIU, M.T. Synthesizing Protocol specifications from service specification in FSM Model. in Proc. Computer Networking Symp.'88,pp 173-82,April 1988.

[Chu 88b] CHU, P. M. and LIU, M.T. Protocol Synthesis in a State Transition Model. in Proceedings IEEE COMPSAC ' 88, pp. 505-512.

[GoYu 84] GOUDA, M. and YU, Y. Synthesis of communicating finite-state machines with guaranteed progress. IEEE Transactions on Communications, COM-32, No.7, July 1984, pp.779-788

[Gotz 89] GOTZHEIN, R. and BOCHMANN G.v. Deriving protocol specifications from service specifications including parameters. to be published in ACM TOPLAS.

[Khen 89] KHENDEK, F. Dérivation de protocoles à partir de services de communication écrits dans un sous-ensemble de LOTOS. M.Sc. Thesis, Université de Montréal, 1989.

[MeBo 83] MERLIN, P. and BOCHMANN, G.v. On the construction of submodule specifications and communication protocols. ACM Trans. on Programming Languages and Systems, No.1, Jan.1983, pp.1-25

[Miln 80] MILNER,R. A calculus of communicating systems. Lecture Notes in Computer Science 92, Springer-Verlag, Berlin 1980, 171p.

[Rama 85] RAMAMOORTHY, C.V., DONG, S.T., USUDA,Y. An Implementation of an Automated Protocol Synthesizer (APS) and its Application to the X.21 Protocol. IEEE Transactions on Software Engineering, Vol. SE-11, No.9, Sept.1985, pp. 886-908

[Rama 86] RAMAMOORTHY, C.V., YAW, Y., AGGARWAL, R., SONG, J. Synthesis of Two-Party Error-Recoverable Protocols. in: Communications, Architectures & Protocols, Proceedings of the ACM SIGCOMM '86 Symposium, Vermont, USA, 1986, pp. 227-235.

[Rana 83] RANA, S. P. A Distributed Solution of the Distributed Termination Problem. Information Processing Letters 17, 1983, pp. 43-46.

[ViLo 85] VISSERS, C.A. and LOGRIPPO, L. The importance of the service concept in the design of data communications protocols. in: M. Diaz (ed.), Protocol Specification, Testing, and Verification, V, Proc. of the IFIP WG 6.1 Workshop, Toulouse-Moissac, France, June 10-13, 1985, North-Holland, Amsterdam 1986, pp. 3-17.

[Zafi 80] ZAFIROPULO, P., WEST, C.H., RUDIN, H., COWAN, D.D. and BRAND, D. Towards analyzing and synthesizing protocols. IEEE Transactions on Communications, Vol. COM-28, No.4, April 1980, pp.651-661.