DERIVING PROTOCOL SPECIFICATIONS FROM SERVICE SPECIFICATIONS

Gregor von BOCHMANN, Reinhard GOTZHEIN
Département d'IRO, Université de Montréal, C.P. 6128, Succursale A
Montréal, Québec, H3C 3J7, Canada

Abstract - **The service concept has acquired an increasing level of recognition by protocol designers. Being an architectural concept, the service concept influences the methodology applied to service and protocol definition. Since the protocol is seen as the logical implementation of the service, one can ask the question whether it is possible to formally derive the specification of a protocol providing a given service.**

This paper addresses this question and presents an algorithm for deriving a protocol specification from a given service specification. It is assumed that services are described by expressions including operators for sequence, parallelism and alternatives and primitive service interactions. The expression defining the service is the basis for the protocol derivation process. The presented algorithm fully automates the derivation process. Future work focuses on the inclusion of parameters and the optimization of traffic between protocol entities.

1. INTRODUCTION

The service concept has acquired an increasing level of recognition by protocol designers (see e.g. [ViLo85]). Being an architectural concept, the service concept influences the methodology applied to service and protocol definition ([Chu84]). Since the protocol is seen as the logical implementation of the service, one can ask the question whether it is possible to formally derive the specification of a protocol providing a given service. Similar questions have been raised concerning the derivation of synchronization code from given specifications ([Lav79], [Mac83]).

A service definition is the specification with the highest degree of abstraction. Therefore, it should not contain explicit information associated with the protocol level. Depending on the OSI-layer being considered, this can mean that no information about the places where service primitives are to be executed is included. However, this is important for the derivation of the protocol entities and has to be added in our approach.

Further information is added during the derivation of the protocol entities. An algorithm is developed for that purpose which allows to fully automate the derivation process. Services are described by expressions including operators for sequence, parallelism and alternatives and primitive service interactions. The expression defining the service is the basis for the protocol derivation process. Currently, we are working on the inclusion of parameters which are not considered in the algorithm presented in this paper.

Other approaches toward the synthesis of protocol specifications can be found in [Zaf80], [Mer83] and [Gou84]. All have in common that they make use of the duality inherent in message exchange: For each message sent by a protocol entity, there must be a protocol entity prepared to receive it. Differences exist concerning the assumed properties of the transmission medium or the maximum number of protocol entities. The approach described in [Zaf80] starts from partly specified protocol entities and gives rules how to arrive at complete specifications ('complete' with respect to message reception). [Gou84] assumes the existence of the specification of one protocol entity and constructs a second one which remains in some sense synchronous with the former. [Mer83] employs the specification of n-1 protocol entities and the service specification for the synthesis of the remaining n^{th} protocol entity. The method introduced in this paper is more general in that only the existence of the service specification is required. In addition, an assignment of the different primitive service interactions to a finite number of service access points must be given, and the method provides specifications for all the protocol entities serving these access points.

The paper is composed as follows: Section 2 introduces concepts and notations on which our algorithm is based. Section 3 presents the algorithm in several steps, each representing successive improvements. It also contains a complete example, demonstrating some of the capabilities of the algorithm. Section 4 mentions further extensions, such as the inclusion of parameters and the optimization of traffic between system components.

2. CONCEPTS AND NOTATIONS

A service (see [BoSu80], [ViLo85]) in our approach is defined by an expression, consisting of service primitives and operators. The syntax of expressions is defined by production rules of a context-free grammar, where '**e**' is a non-terminal (and also the starting) symbol, and '{**a**,**b**,...} ∪ {;,||,|}' is a finite set of terminal symbols:

1. for each terminal symbol **x** ∈ {**a**,**b**,...}: **e** → **x**
2. **e** → **e** ; **e**
3. **e** → **e** || **e**
4. **e** → **e** | **e**

The operator ';' expresses that the service defined by the left subexpression must be terminated completely before execution of the service defined by the right subexpression may be started. '||' expresses that the services defined by the two subexpressions may be executed in parallel. The meaning of '|' is that either the service defined by the left subexpression or by the right subexpression is to be executed.

The services defined by such expressions have to be augmented by information about the location where a service primitive shall be executed. For this reason, we introduce identifiers referring to interaction points, called "places" in the following, and associate service primitives with places: The notation '$\mathbf{a}^{\mathbf{4}}$' means that the service primitive '**a**' is to be executed at place '**4**'.

We now can describe a constraint which applies to production rule 4 in an informal way (a precise definition is given in section 3.1.). In this case of alternative subexpressions, a decision has to be made which subexpression should be executed. We assume that this decision is taken at one place without the consultation of entities at other places (all actions at one place are associated with one entity). Therefore, we require that the places of the starting operations of the two subexpression be the same.

3. THE DERIVATION ALGORITHM

In this section, we introduce the derivation algorithm in several steps thus incrementally arriving at the desired result. For a given service specification (see section 2), this algorithm produces the specifications of all protocol entities.

The principle is to define the behavior of each protocol entity to be the projection (see [Mer83]) of the service specification onto the place (i.e. service access point) serviced by the respective entity. This is augmented by appropriate synchronization among the protocol

entities through the underlying communication medium such that the possible temporal order of operations being executed at different places satisfies the order implied by the service specification. Note that each protocol entity can determine directly only the order of operations at the place which it services. Therefore, communication among the protocol entities through an underlying communication medium is required and has to be introduced by the derivation algorithm.

Synchronization is required in all cases where the operator ';' is used in the service definition. Here, all terminating operations of the left subexpression of ';' have to send synchronization messages to all starting operations of its right subexpression. Similarly, all starting operations of the right subexpression have to receive synchronization messages from all terminating operations of the left subexpression.

In case of '||', no synchronization is needed. Also, with the constraint concerning the places of starting operations in production rule 4 (see section 2), no additional synchronization is required in case of '|'.

3.1. A first version of the derivation algorithm

In order to define the derivation algorithm, the formalism of attribute grammars ([Boc76]) is used. From the consecutive application of production rules 1 to 4, starting from the non-terminal symbol '**e**', we obtain a syntax tree for each service expression '$\mathbf{e_S}$', where service expressions only contain terminal symbols. For each node in this tree, synthesized attributes pass information upward (from the successor(s) of the node toward the root), and inherited attributes pass information downward. For the derivation algorithm, the attributes provide information between which places synchronization messages must be exchanged.

To define the attribute evaluation rules, we need a clear distinction between the left and the right side of a production rule and between the subexpressions on its right side. Therefore, we introduce indices referring to the number of a successor node. This notation does not affect the applicability of production rules, i.e. if a rule is applicable to the non-terminal symbol '**e**', then it can also be applied to '$\mathbf{e_1}$' or '$\mathbf{e_2}$'.

We rewrite the context-free grammar of section 2 as follows:

1. for each terminal symbol $\mathbf{x} \in \{\mathbf{a^i},\mathbf{b^j},...\}$: $\mathbf{e} \rightarrow \mathbf{x}$
2. $\mathbf{e} \rightarrow \mathbf{e_1}\ ;\ \mathbf{e_2}$
3. $\mathbf{e} \rightarrow \mathbf{e_1}\ ||\ \mathbf{e_2}$
4. $\mathbf{e} \rightarrow \mathbf{e_1}\ |\ \mathbf{e_2}$
   where the starting operations of '$\mathbf{e_1}$' and '$\mathbf{e_2}$' are located at one single place

The following attributes are defined for each node of a syntax tree:

- **S** (.) : send-operations associated with the 'starting places' (synthesized)
- **E** (.) : receive-operations associated with the 'ending places' (synthesized)
- **P** (.) : receive-operations from the 'preceeding places' (inherited)
- **F** (.) : send-operations to the 'following places' (inherited)

For production rule 1, the attributes **S** and **E** are synthesized as follows:

S (**e**) := "s"$_{\text{place}(\mathbf{x})}$	for each terminal symbol $\mathbf{x} \in \{\mathbf{a^i},\mathbf{b^j},...\}$
E (**e**) := "r"$_{\text{place}(\mathbf{x})}$	for each terminal symbol $\mathbf{x} \in \{\mathbf{a^i},\mathbf{b^j},...\}$

'place' is a function from the set $\{\mathbf{a^i},\mathbf{b^j},...\}$ to the set of places: place($\mathbf{x^p}$):=**p**. The values of 'place' are interpreted as strings. Subsequent strings are implicitly concatenated. Thus, we get string values for the attributes **S** and **E** which later are incorporated into protocol expressions: "s_p" or "r_p" means that a synchronization message has to be sent to, or received from, place '**p**', respectively.

The heuristics for the attribute evaluation rule above is the observation that synchronization messages have to be transmitted to 'place(**x**)' from all 'preceeding places' and to be received by all 'following places'. For the other production rules, the attributes are evaluated as follows:

production rule	S	E
2	$S(e) := S(e_1)$	$E(e) := E(e_2)$
3	$S(e) := S(e_1)$ "||" $S(e_2)$	$E(e) := E(e_1)$ "||" $E(e_2)$
4	$S(e) := S(e_1)$	$E(e) := E(e_1)$ "|" $E(e_2)$

The heuristics for the attribute evaluation rules concerning production rules 2 to 4 is just the same as for production rule 1. In case of production rule 2, for example, *the send-operations associated with the* 'starting places' for the father-node in the syntax tree are the same as for the left subexpression of the operator ';', the receive-operations associated with 'ending-places' are the same as for the right subexpression.

We are now capable of precisely defining the constraint for production rule 4:

$$e \rightarrow e_1 \mid e_2 \quad \text{where } S(e_1) = S(e_2) = "s_p" \text{ for some place 'p'}$$

This also explains why we can simplify the definition of the attribute evaluation rule for **S** in this case.

After having synthesized attributes **S** and **E**, we can now evaluate the inherited attributes **P** and **F**, starting at the root. Initializing '**P**(**e**)' and '**F**(**e**)' as '**empty**' at the root, the following evaluation rules are used:

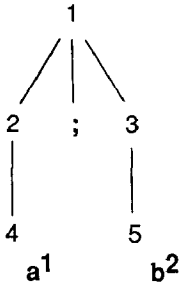
production rule	P	F
1	$P(x) := P(e)$ for all $x \in \{a^i, b^j, ...\}$	$F(x) := F(e)$ for all $x \in \{a^i, b^j, ...\}$
2	$P(e_1) := P(e)$ $P(e_2) := E(e_1)$	$F(e_1) := S(e_2)$ $F(e_2) := F(e)$
3	$P(e_1) := P(e_2) := P(e)$	$F(e_1) := F(e_2) := F(e)$
4	$P(e_1) := P(e_2) := P(e)$	$F(e_1) := F(e_2) := F(e)$

The heuristics is that we want to arrive at expressions which define for each operation associated with a leaf of the syntax tree which receptions have to be performed *before* the execution of the operation (attribute **P**) and which transmissions are necessary afterwards (attribute **F**). This is done by making use of the synthesized attributes **S** and **E** (see definition covering production rule 2).

The attributes defined above can now be used to derive, from a service specification, the specification of the protocol entities. Let '**p**' be an arbitrary place, then the following rules, applied recursively to the syntax tree of service expressions, provide a specification for the entity serving the place '**p**', which is given by 'T_p' applied to the root node of the service specification.

production rule	T_p
1	$T_p(e) :=$ if place(x) = "p" then P(x) "; x ;" F(x) else "empty" for all $x \in \{a^i, b^j, ...\}$
2	$T_p(e) := T_p(e_1)$ ";" $T_p(e_2)$
3	$T_p(e) := T_p(e_1)$ "||" $T_p(e_2)$
4	$T_p(e) := T_p(e_1)$ "|" $T_p(e_2)$

In order to obtain the specifications for all protocol entities, ' T_p' has to be applied for each place '**p**'. Let us consider a first example: the operations $\{a^1, b^2\}$ and the service expression '$a^1; b^2$'. The syntax tree for this service and its attributes can be depicted as follows ("-" represents "**empty**"):



node \ attribute	S	E	P	F
1	"s_1"	"r_2"	-	-
2	"s_1"	"r_1"	-	"s_2"
3	"s_2"	"r_2"	"r_1"	-
4			-	"s_2"
5			"r_1"	-

The derivation of the protocol specifications for the places 1 and 2 leads to the following result:

$$\begin{aligned} T_1(e_S) &= T_1(a^1; b^2) \\ &= T_1(a^1) ";" T_1(b^2) \\ &= P(a^1) "; a^1 ;" F(a^1) "; empty" \\ &= "empty ; a^1 ; s_2 ; empty" \\ &= "a^1 ; s_2" \end{aligned}$$

$$\begin{aligned} T_2(e_S) &= \ldots = "empty ;" P(b^2) "; b^2 ;" F(b^2) \\ &= "r_1 ; b^2" \end{aligned}$$

This is obviously the result we were expecting: the protocol entity at place '1' first executes operation 'a^1' and then sends a synchronization message 's_2' to place '2', while the protocol entity at place '2' first receives this message from place '1' (see 'r_1') and then executes operation 'b^2'.

It should be noted that certain simplifications of expressions obtained during the process of derivation are permitted. Semantically, the following expressions are equivalent:

$$\begin{aligned} &e \;;\; empty \approx e \\ &empty \;;\; e \approx e \\ &e_1 \;||\; e_2 \approx e_2 \;||\; e_1 \\ &e \;||\; empty \approx e \\ &empty \;|\; empty \approx empty \end{aligned}$$

for arbitrary expressions e, e_1 and e_2.

3.2. Improvement of the derivation algorithm

The algorithm presented so far still contains some flaws. One flaw can be illustrated by the following example. The service expression

$e_S = (a^1 \mid b^1) \;;\; (c^2 \mid d^2)$ leads to
$T_1(e_S) =$ "$(a^1; s_2) \mid (b^1; s_2)$"
$T_2(e_S) =$ "$((r_1 \mid r_1) \;;\; c^2) \mid ((r_1 \mid r_1) \;;\; d^2)$"

In the case of $T_1(e_S)$, the result is exactly what one expects. In the case of $T_2(e_S)$, it should be

$$T_2'(e_S) = \text{"}(r_1 \mid r_1) \;;\; (c^2 \mid d^2)\text{"}$$

which better reflects the fact that the choice between 'c^2' and 'd^2' is made on place '2' *after* a reception from place '1'. (This may also be seen as an optimization.)

The information required is already contained in the attribute **P**. Therefore, the following revised transformation rule 4 could be applied:

4') $T_p(e)$ = if $(S(e_1) = S(e_2) = \text{"}s_p\text{"})$
then $P(e)$ "; (" $T_p(e_1)$ "|" $T_p(e_2)$ ")"
else $T_p(e_1)$ "|" $T_p(e_2)$

The distinction made by the condition assures that the attribute '**P**(**e**)' is only included in the result of 'T_p(**e**)' if the place of the starting operation of 'e_1' is '**p**', i.e. the place for which the specification of the protocol entity is currently derived. By constraint, '$S(e_1)$' and '$S(e_2)$' are identical (see production rule 4). In order not to get the value for '**P**(**e**)' a second time by applying transformation rule 1 later in the derivation process, the attribute evaluation rule for production 4 has to be changed, too:

4') $P(e_1) := P(e_2) :=$ "empty"

The reader may check that the changes lead to the result $T_2'(e_S)$.

3.3. Further improvements

The revised algorithm of section 3.2. still has some shortcomings which are illustrated by the following examples:

a) alternative

i) $e_S = (a^1; b^2) \mid (c^1; d^2)$ results in
$T_1(e_S) =$ "$(a^1; s_2) \mid (c^1; s_2)$" and
$T_2(e_S) =$ "$(r_1; b^2) \mid (r_1; d^2)$"
Thus the result is semantically equivalent to the protocol derivation for the service

ii) $e_S' = (a^1 \mid c^1) \;;\; (b^2 \mid d^2)$ leading to
$T_1(e_S') =$ "$(a^1; s_2) \mid (c^1; s_2)$" and
$T_2(e_S') =$ "$(r_1 \mid r_1);\; (b^2 \mid d^2)$"
The protocol derived from e_S is obviously not what one expects, since the sequence of operations defined in the service is not always maintained by the derived protocol.

b) parallelism

$e_S = (a^1 \,||\, b^1) \;;\; (c^2 \,||\, d^2)$ results in
$T_1(e_S) =$ "$[a^1; (s_2 \,||\, s_2)] \,||\, [b^1; (s_2 \,||\, s_2)]$" and
$T_2(e_S) =$ "$[(r_1 \,||\, r_1); c^2] \,||\, [(r_1 \,||\, r_1); d^2]$"
This allows e.g. c^2 to be executed after two receptions from place '1', but before completion

of both a^1 and b^1. As in case (a.i), the derived protocol is not correct.

The problem seems to be that different send-operations cannot be distinguished by the receiver. A means for overcoming the deficiencies illustrated above therefore is the addition of a message parameter which 'identifies' a synchronization message.

It is noted that synchronization is always linked to the sequence operator ';': The service defined by the left subexpression must be completely executed before the service given by the right subexpression may be started. Therefore, synchronization messages have to be sent from all places of 'terminating' operations of the left subexpression to all places of 'starting' operations of the right subexpression.

Messages related to different 'terminating' operations must be distinguishable at the receiving places, and therefore we introduce a consecutive numbering applied to groups of synchronization messages: For each 'terminating' operation, all messages indicating its completion form a group. Groups can have more than one element, because the completion of a 'terminating' operation may have to be communicated to more than one place (or to the same place, but for different 'starting' operations) of the right subexpression.

The following modifications of the algorithm overcome the deficiencies illustrated above: An attribute **N**(.) is introduced which defines a unique numbering of all leaves of the syntax tree. This attribute can be obtained by parsing the syntax tree from left to right. Now we can modify the definition of the synthesis of the attributes **S** and **E** for production rule 1:

1') $\mathbf{S(e)} := \text{"s"}_{\text{place(x)}}\text{"(z)"}$
for each terminal symbol $x \in \{a^i, b^j, \ldots\}$
$\mathbf{E(e)} := \text{"r"}_{\text{place(x)}}\text{"(N(x))"}$
for each terminal symbol $x \in \{a^i, b^j, \ldots\}$

This means that we add the parameter value **N**(**x**) to receive-operations associated with the operation **x** (attribute **E**). Furthermore, a parameter 'z' is added to send-operations which is replaced by a specific value later in the derivation process, according to the following modified derivation rule:

1') $\mathbf{T_p(e)}$:= if place(**x**) = "**p**"
then **P**(**x**) "; **x** ;" **F**(**x**)[z/**N**(**x**)]
else "**empty**"
for all $x \in \{a^i, b^j, \ldots\}$

Here, '**F**(**x**)[z/**N**(**x**)]' denotes that all occurrences of 'z' in '**F**(**x**)' are to be substituted by the value of '**N**(**x**)'.

Reconsidering the examples from the beginning of section 3.3., the derivation now leads to correct protocol specifications:

a.i) $\mathbf{T_1}(e_S)$ = "$(a^1; s_2(1)) \mid (c^1; s_2(3))$"
$\mathbf{T_2}(e_S)$ = "$(r_1(1); b^2) \mid (r_1(3); d^2)$"

a.ii) $\mathbf{T_1}(e_S')$ = "$(a^1; s_2(1)) \mid (c^1; s_2(2))$"
$\mathbf{T_2}(e_S')$ = "$(r_1(1) \mid r_1(2)); (b^2 \mid d^2)$"

b) $\mathbf{T_1}(e_S)$ =
"$[a^1; (s_2(1) \parallel s_2(1))] \parallel [b^1; (s_2(2) \parallel s_2(2))]$"
$\mathbf{T_2}(e_S)$ =
"$[(r_1(1) \parallel r_1(2)); c^2] \parallel [(r_1(1) \parallel r_1(2)); d^2]$"

3.4. A complete example

We give in the following a derivation of the protocol for the service defined by

$$e_S = ([(a^1 ; (b^2 ; c^3)) \mid (d^1 ; e^5)] \parallel f^6) ; (g^7 \parallel h^8)$$

The syntax tree of this service expression is shown in figure 1. For each node of the tree, the attribute values of **S**, **E**, **P**, **F** and **N** are given.

The result of the derivation process is the following:

$\mathbf{T_1}(e_S)$ = "$(a^1; s_2(1)) \mid (d^1; s_5(4))$"
$\mathbf{T_2}(e_S)$ = "$[r_1(1); b^2; s_3(2)] \mid$ **empty**"
$\mathbf{T_3}(e_S)$ = "$[r_2(2); c^3; (s_7(3) \parallel s_8(3))] \mid$ **empty**"
$\mathbf{T_5}(e_S)$ = "**empty** $\mid [\, r_1(4); e^5; (s_7(5) \parallel s_8(5))\,]$"
$\mathbf{T_6}(e_S)$ = "$f^6; (s_7(6) \parallel s_8(6))$"
$\mathbf{T_7}(e_S)$ = "$[\, (r_3(3) \mid r_5(5)) \parallel r_6(6)\,]; g^7$"
$\mathbf{T_8}(e_S)$ = "$[\, (r_3(3) \mid r_5(5)) \parallel r_6(6)\,]; h^8$"

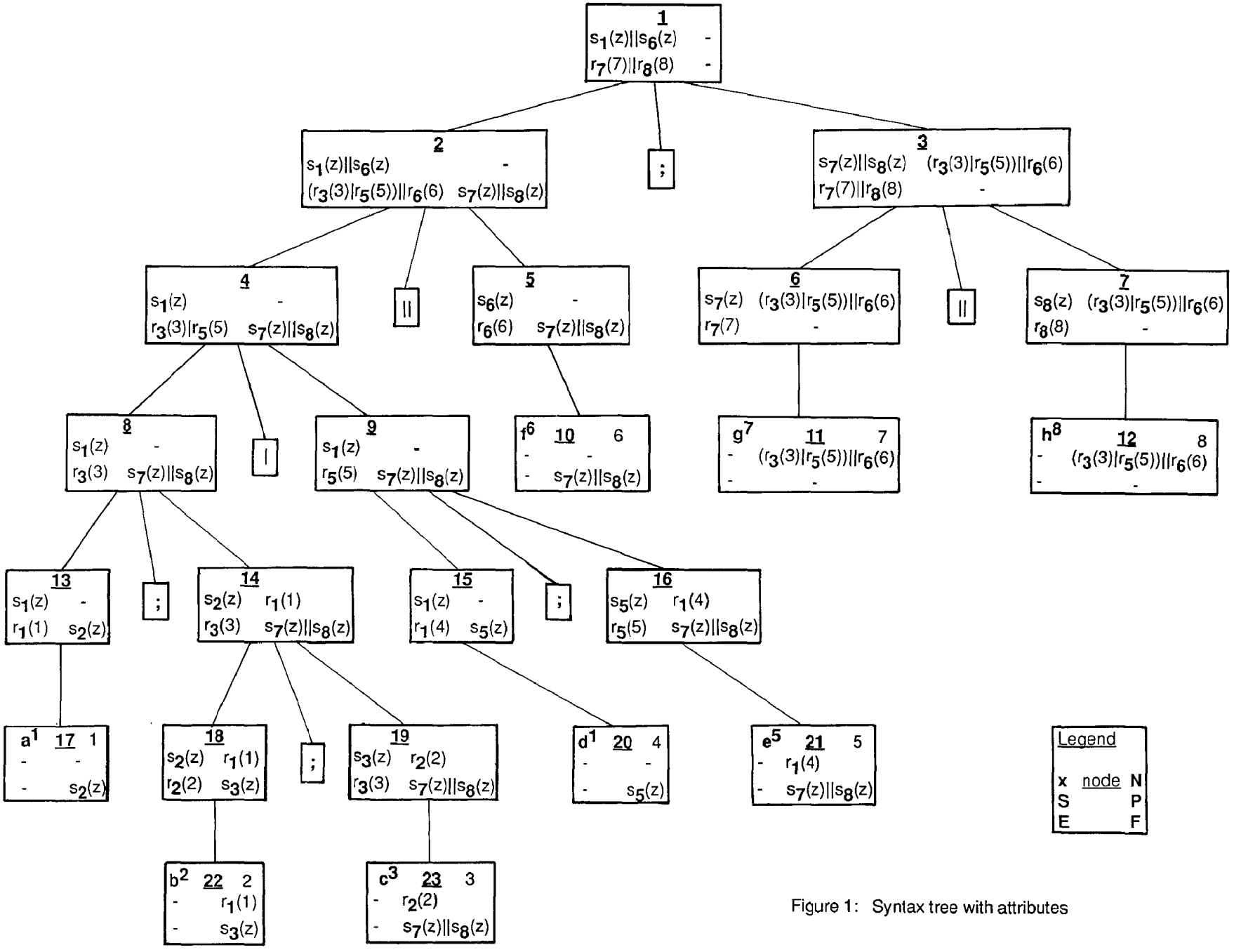


Figure 1: Syntax tree with attributes

4. DISCUSSION, EXTENSIONS AND APPLICATIONS

We have presented an algorithm which allows to fully automate the derivation of protocol specifications from service expressions. To define a service, we used operators for sequence, parallelism and alternatives. Such operators can also be found in FDTs like LOTOS ([Bri85]), CCS ([Mil80]) or CSP ([Hoa78]). The exchange of messages between the protocol entities is assumed to be provided by reliable FIFO-queues. The protocol specifications obtained by applying the algorithm are unique, since the derivation process is based on the (unique) syntax tree of the service expression and the defined attribute evaluation rules are deterministic.

It would be desirable to formally define the semantics of the language used to specify services and protocols in order to prove that the presented algorithm yields correct results. So far we believe that the flaws eliminated in sections 3.2. and 3.3. represent all shortcomings present in the algorithm as described in section 3.1.

A current limitation which we expect to remove in the next version concerns parameters which will have to be added to the service primitives. Since we deal with distributed systems, inputs of a service primitive may have to be obtained from different places, they are possibly results of the execution of other service primitives. First of all, such dependencies between inputs and outputs impose constraints on the set of valid service expressions. Secondly, additional message exchange becomes necessary to communicate outputs to the places where they are needed, which requires an extension of our derivation algorithm.

Additional extensions should concern the optimization of traffic necessary to synchronize operations and to pass parameter values. It is for instance not necessary to pass messages to synchronize subsequent operations at the same place. Also, synchronization messages and data messages may be combined.

Furthermore, it could be useful to include more powerful elements like levels of hierarchy and recursion into our language for the specification of services. The impact of such extensions on the derivation algorithm must be carefully examined.

The described protocol derivation algorithm may be applied in different areas. It is noted that we assume the availability of a reliable message transmission service between participating protocol entities. Usually, logical connections would be established between these entities before the derived protocol is executed. Within the OSI reference model, this situation can be satisfied for the application layer, it is therefore expected that the algorithm could be useful in areas such as distributed data bases, process control, etc. It is necessary, however, to include the exchange of parameters into the considerations. Also, subsystem failures which are not handled by the algorithm should be taken into account.

Acknowledgements. The idea of deriving protocol specifications from service specifications arose in discussions with H. Ichikawa and M. Itoh during a visit of G. v. Bochmann at the NTT Mosashino ECL in Tokyo. We thank H. Ichikawa and T. Murakami for their detailed study and helpful comments on an earlier version of the algorithm presented here.

REFERENCES

[Boc76] Bochmann,G.v.: Semantic Evaluation from Left to Right, Communications of the ACM, Feb.1976, Vol.19, No.2, pp.55-62

[BoSu80] Bochmann,G.v., Sunshine,C.A.: Formal Methods in Communication Protocol Design, IEEE Transactions on Communications, Vol. COM-28, No.4, April 1980, pp.624-631

[Bri85] Brinksma,E.: A Tutorial on LOTOS, 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. 171-194

[Chu84] Chung,R.: A Methodology for Protocol Design and Specification based on an Extended State Transition Model, Proc. ACM SIGCOMM Symposium, June 1984, Montreal, Computer Communication Review, Vol.14, No.2, pp.34-41

[Gou84] Gouda,M., Yu,Y.: Synthesis of Communicating Finite-State Machines with guaranteed Progress, IEEE Transactions on Communications, COM-32, No.7, July 1984, pp.779-788

[Hoa78] Hoare,C.A.R.: Communicating Sequential Processes, CACM Vol.21, No.8,1978

[Lav79] Laventhal,M.S.: A Constructive Approach to Reliable Synchronization Code, Proc. 4th Int. Conf. on Software Engineering, 1979, pp.194-202

[Mac83] Mackert,L.: Modellierung, Spezifikation und korrekte Realisierung von asynchronen Systemen, Arbeitsberichte des IMMD Bd.16, Nr.7, Universitaet Erlangen-Nuernberg, Erlangen 1983, 266p.

[Mer83] Merlin,P., 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

[Mil80] Milner,R.: A Calculus of Communicating Systems, Lecture Notes in Computer Science 92, Springer-Verlag, Berlin 1980, 171p.

[ViLo85] Vissers,C.A., 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

[Zaf80] Zafiropulo,P., West,C.H., Rudin,H., Cowan,D.D., Brand,D.: Towards Analyzing and Synthesizing Protocols, IEEE Transactions on Communications, Vol. COM-28, No.4, April 1980, pp.651-661