High-level Message Sequence Charts

S. Mauw and M.A. Reniers

Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven,
The Netherlands. E-mail: {sjouke,michelr}@win.tue.nl

We study High-level Message Sequence Charts – a concept incorporated into *MSC*96 for composing *MSC*s explicitly. A formal semantics is given which extends the accepted process algebra semantics of *MSC*92. We assess the language by studying a simple example, which leads us to consider the extension of *HMSC* with gates.

1. INTRODUCTION

The standardization of Message Sequence Charts (*MSC*s) [8] in 1992 by the CCITT has increased the interest in and use of *MSC*s considerably. Due to the variety of applications, many extensions have been proposed since 1992 for increasing the use of *MSC* in specific application domains or in general. Several of these proposed new language constructs were selected when extending the *MSC* language to *MSC*96 [7].

The composition of *MSC*s has been a main issue for the upgrade of the recommendation. In *MSC*92 composition of *MSC*s was hardly covered, while in *MSC*96 there are several new language features for constructing *MSC*s from simpler *MSC*s. In this paper we will focus on one of these composition techniques, namely High-level Message Sequence Charts (*HMSC*s).

An *HMSC* is a graphical overview of the relation between the *MSC*s contained. It helps in keeping track of the control-flow. In an *HMSC* alternative, sequential and parallel composition as well as recursion are captured in an attractive graphical layout: references to *MSC*s are related by means of arrows connecting them. One can look at *HMSC* as the synthesis of the roadmap approach [16,18] and the operator approach [6].

One of the current aims is to also extend the semantical definitions for *MSC*92 [11,9] to the *MSC*96 language. Because *MSC*96 has become quite a large language, we propose to study the new constructs first in isolation and get a full understanding of these features before combining them into one semantics definition.

In this paper, we will give a definition of the semantics of the sub-language *HMSC* of *MSC*96, based upon the recommended process algebra semantics of *MSC*92. Further, we discuss the use of *HMSC* by studying the well-known Alternating Bit Protocol (*ABP*) from different views. This case study motivates to extend *MSC*96 with gates on *HMSC* nodes.

This paper is structured as follows. First, we give an introduction to High-level Message Sequence Charts (*HMSC*s). As a basis we take Basic Message Sequence Charts (*BMSC*s). Then, in Section 3, we present a denotational and indirectly an operational semantics for *HMSC*. In Section 4, we focus on a layered description of the well-known *ABP*. In Section 5 we argue in

favour of an extension of *HMSC*s based on gates. We conclude with some remarks and topics for further research.

Acknowledgements

We would like to thank Anders Ek, Loe Feijs, Jens Grabowski, Øystein Haugen, and Ekkart Rudolph for their participation in the discussions on our ideas.

2. HIGH-LEVEL MESSAGE SEQUENCE CHARTS

2.1. Basic Message Sequence Charts

A Basic Message Sequence Chart (*BMSC*) contains a description of the asynchronous communication between instances. Additionally local actions can be specified on instances. An instance is an abstract entity of which one can observe (part of) the interaction with other instances or with the environment. The *BMSC P* in Figure 1 defines the communication behaviour between instances i, j, k, l and the environment. An instance is denoted by a vertical axis. The time along each axis is running from top to bottom.

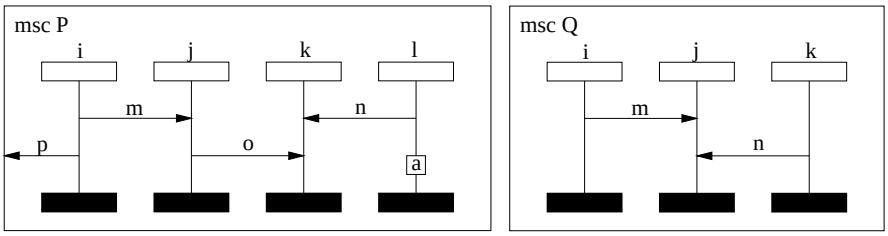


Figure 1. Example Basic Message Sequence Charts.

A communication between two instances is represented by an arrow which starts at the sending instance and ends at the receiving instance. In *BMSC P* from Figure 1 we consider the messages m, n, o, p. Message p is sent to the environment. The behaviour of the environment is not specified. For instance l a local action a is defined.

Although the activities along one single instance axis are completely ordered, we will not assume a notion of global time. The only dependencies between the timing of the instances come from the restriction that a message must have been sent before it is received. For *BMSC P* this implies for example that message o is received by k only after it has been sent, and consequently, after the reception of m by j. For the sending and receiving of m and n no order is specified. Since we have asynchronous communication it is even possible to first send m, then send and receive n, and finally receive m.

2.2. High-level Message Sequence Charts: *HMSC*

The most simple *HMSC* is a *BMSC*, as in Figure 1. The purpose of the compound *HMSC*s is to describe the relations between the *MSC*s contained in a graphically attractive way. A compound *HMSC* consists of a collection of components, enclosed by a frame. The components are thought of as complex *MSC*s that operate in parallel. Every component consists of a number of nodes and

a number of arrows that imply an order on the nodes. We make a distinction between three kinds of nodes. Every component has exactly one start node, indicated by an upside-down triangle (∇). Further, it may contain a number of end nodes, indicated by a triangle ($\triangle$), and several *HMSC* references. An *HMSC* reference consists of a frame with rounded corners enclosing the name of the referenced *HMSC*. We require that within a component every node (including the end nodes) is reachable from the start node. In Figure 2 an *HMSC* is shown. For simplicity we do not draw the abundant frame from *MSC*96 to denote parallelism.

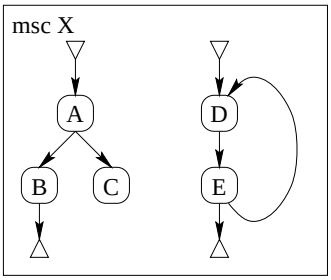


Figure 2. Example *HMSC*.

An arrow between two *HMSC* references implies that they are composed vertically. Splitting of an arrow denotes that the successors are alternatives. A cycle connecting a number of *HMSC* references expresses a repetition. In this way infinitary behaviour can be described.

Diagrams with many nodes and arrows can easily become unreadable for the human eye. By introducing *connectors* we can improve a lot on this problem. A connector is indicated by a circle ($\circ$). Every combination of an incoming and an outgoing edge of a connector represents an arrow between the source of the incoming arrow and the destination of the outgoing arrow. A transformation of *HMSC*s with connectors to *HMSC*s without connectors can, in the presence of a formal definition of *HMSC* diagrams, easily be given. An example of an *HMSC* with connectors is given in Figure 7 (see Section 4).

3. SEMANTICS

This section is devoted to the semantics of *HMSC*. With respect to *MSC*92 formal semantics were defined based on Petri nets [5], Büchi automata [10], process algebra [11] and, more recently, partial order methods [1]. Since the process algebra approach was selected for standardization [9], we will use this approach as a starting point. First, we recapitulate the semantics of *BMSC*. Next, we define the operators needed for relating the *MSC*s contained in an *HMSC*. Finally, we define the semantics of *HMSC* based on an abstract syntax.

3.1. Basic Message Sequence Charts

In [11] a semantics for *BMSC* is presented. We will give a short explanation of this semantics. To each *BMSC* a closed process expression is associated. With every event specified in a *BMSC* an atomic action is associated as follows: The sending of a message *m* by instance *i*

to instance j (or the environment) is represented by $out(i, j, m)$ (or $out(i, env, m)$), the reception of a message m by instance j from instance i (or the environment) is denoted $in(i, j, m)$ (or $in(env, j, m)$), and a local action a on instance i is denoted $action(i, a)$. Together with the constants ε and δ which denote successful termination and inaction (or deadlock) respectively, the atomic actions mentioned above constitute the constants of the term algebra used for the semantics. Furthermore, the term algebra consists of binary operators $+$ and $\cdot$ for non-deterministic choice and strong sequential composition, respectively. The process expression that can be associated to a *BMSC* defines the order in which events may be executed by means of an operational semantics.

Next, we present a structured operational semantics for closed terms in the style of Plotkin [14] (see Table 1). Such an operational semantics consists of a number of inference rules of the following form:

$$\frac{p_1, \cdots, p_n}{c}.$$

This inference rule means that for every instantiation of variables in the *premises* $p_1, \cdots, p_n$ and the *conclusion* c we can conclude c from $p_1, \cdots, p_n$. If no premises are present, i.e., $n = 0$, then c is a tautology (often called an axiom). Premises and conclusions are constructed from the predicates $\downarrow$ and $\xrightarrow{a}$. The intuition of the unary predicate $\downarrow$ is as follows: $p\downarrow$ indicates that p has an option to terminate successfully. The intuitive idea of the predicate $\xrightarrow{a}$ (for every $a \in A$) is as follows: $p \xrightarrow{a} q$ denotes that process p can execute action a and after the execution thereof the resulting process is q.

With this operational semantics, we define the behaviour of a *BMSC*. By defining the usual notion of strong bisimilarity [13], we can also reason about the equality of *BMSC*s.

Table 1
Structured operational semantics for the constants and operators ($a \in A$).

$\dfrac{}{\varepsilon\downarrow}$	$\dfrac{x\downarrow}{x+y\downarrow}$	$\dfrac{y\downarrow}{x+y\downarrow}$	$\dfrac{x\downarrow, y\downarrow}{x\cdot y\downarrow}$	
$\dfrac{}{a \xrightarrow{a} \varepsilon}$	$\dfrac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'}$	$\dfrac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}$	$\dfrac{x \xrightarrow{a} x'}{x\cdot y \xrightarrow{a} x'\cdot y}$	$\dfrac{x\downarrow, y \xrightarrow{a} y'}{x\cdot y \xrightarrow{a} y'}$

Based on this structured operational semantics, to every closed process expression (and thus to every *BMSC*) a labeled transition system can be associated as follows: the initial node of the labeled transition system is given by the term under consideration. A state s has an outgoing edge labeled with a to a state s' iff $s \xrightarrow{a} s'$ is derivable from the inference rules and tautologies. Also, s is labeled by a termination arrow iff $s\downarrow$ is derivable.

Example 3.1 Consider the *BMSC* Q from Figure 1. For our convenience we denote this *BMSC*

with its name Q. The semantics of this *BMSC* is given by

$$\begin{aligned} S_{BMSC}(Q) = \; & out(i,j,m) \cdot (out(k,j,n) \cdot in(i,j,m) \cdot in(k,j,n) \\ & \qquad + in(i,j,m) \cdot out(k,j,n) \cdot in(k,j,n) \\ & \qquad) \\ + \; & out(k,j,n) \cdot out(i,j,m) \cdot in(i,j,m) \cdot in(k,j,n). \end{aligned}$$

The labeled transition system that is associated to this *BMSC* is given in Figure 3.

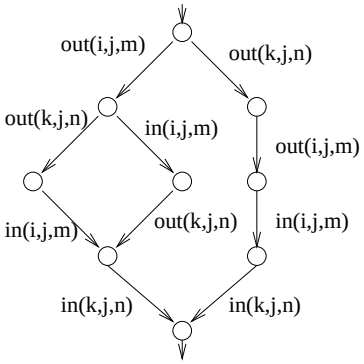


Figure 3. Labeled transition system.

3.2. The composition operators

The relations between the composing *MSC*s of an *HMSC* are graphically defined by arrows and semantically by using three operators.

The delayed choice operator ($\mp$) was introduced by Baeten and Mauw [2]. It acts as a deterministic choice in the context of strong bisimulation. The delayed choice between processes x and y, is the process obtained by joining the common initial parts of x and y and continuing with a non-deterministic choice ($+$) between the remaining parts.

The weak sequencing operator ($\circ$) is based on the interworking sequencing operator [12]. A generalization of this operator was studied in [15]. The weak sequencing of the processes x and y denotes their parallel execution with the restriction that an action from y can only be executed if that is permitted by x. In Figure 4 we give a typical example of vertical composition by means of the weak sequencing operator.

The free merge operator ($\|$) denotes the interleaved execution of its arguments without synchronization. It is well-known in concurrency theory.

A structured operational semantics for the operators $\mp$, $\circ$, and $\|$ is provided in Table 2. Auxiliary predicates $\cdots \xrightarrow{a}$ (for $a \in A$) are introduced. The predicate $x \cdots \xrightarrow{a} x'$ means that x allows y to execute event a in a context $x \circ y$. Thus, this predicate is used to restrict the collection of possible events that can be executed by y. The process x' that results from permitting a to be executed in the permission relation $x \cdots \xrightarrow{a} x'$ is obtained by omitting from x all alternatives that do not allow the execution of a by y. The basis for this so-called *permission relation* is provided

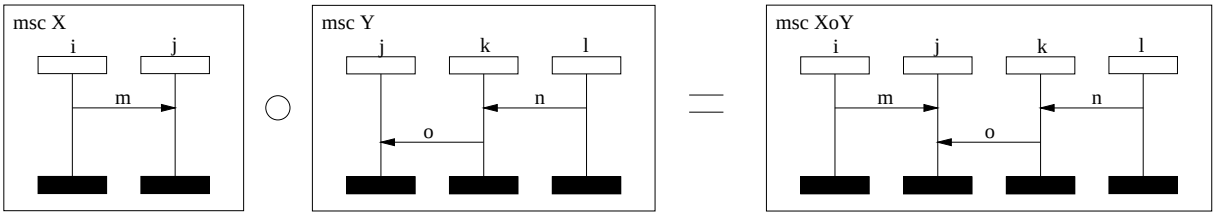


Figure 4. Vertical composition.

by the relation $I \subseteq A \times A$, called the *independence relation*. Two atomic actions are called independent, aIb, if they are defined on different instances. Formally, the independence relation I is defined as follows: aIb iff $inst(a) \neq inst(b)$. The function *inst*, which associates to an event the instance on which it is specified, is defined by $inst(out(i, j, m)) = inst(out(i, env, m)) = inst(action(i, a)) = i$ and $inst(in(i, j, m)) = inst(in(env, j, m)) = j$.

The inference rules also contain *negative premises* $x \overset{a}{\not\rightarrow}$ and $x \not\cdot\overset{a}{\cdot\!\rightarrow}$. The expression $x \overset{a}{\not\rightarrow}$ means that process x cannot execute an action a. Similarly, $x \not\cdot\overset{a}{\cdot\!\rightarrow}$ indicates that process x does not permit the execution of atomic action a. The operational semantics of delayed choice is taken from [2], the operational semantics of the weak sequencing is based on [15], and the structured operational semantics of the free merge is taken from [3].

Table 2
Structured operational semantics for $\mp$, $\circ$, and $\|$ ($a, b \in A$).

$$\frac{aIb}{b \overset{a}{\cdots\!\rightarrow} b} \qquad \frac{}{\varepsilon \overset{a}{\cdots\!\rightarrow} \varepsilon} \qquad \frac{}{\delta \overset{a}{\cdots\!\rightarrow} \delta} \qquad \frac{x \overset{a}{\cdots\!\rightarrow} x',\ y \overset{a}{\cdots\!\rightarrow} y'}{x \cdot y \overset{a}{\cdots\!\rightarrow} x' \cdot y'}$$

$$\frac{x \overset{a}{\cdots\!\rightarrow} x',\ y \not\cdot\overset{a}{\cdot\!\rightarrow}}{x + y \overset{a}{\cdots\!\rightarrow} x'} \qquad \frac{x \not\cdot\overset{a}{\cdot\!\rightarrow},\ y \overset{a}{\cdots\!\rightarrow} y'}{x + y \overset{a}{\cdots\!\rightarrow} y'} \qquad \frac{x \overset{a}{\cdots\!\rightarrow} x',\ y \overset{a}{\cdots\!\rightarrow} y'}{x + y \overset{a}{\cdots\!\rightarrow} x' + y'}$$

$$\frac{x\!\downarrow}{x \mp y\!\downarrow} \qquad \frac{y\!\downarrow}{x \mp y\!\downarrow} \qquad \frac{x \overset{a}{\rightarrow} x',\ y \overset{a}{\rightarrow} y'}{x \mp y \overset{a}{\rightarrow} x' \mp y'} \qquad \frac{x \overset{a}{\rightarrow} x',\ y \overset{a}{\not\rightarrow}}{x \mp y \overset{a}{\rightarrow} x'} \qquad \frac{x \overset{a}{\not\rightarrow},\ y \overset{a}{\rightarrow} y'}{x \mp y \overset{a}{\rightarrow} y'}$$

$$\frac{x \overset{a}{\cdots\!\rightarrow} x',\ y \not\cdot\overset{a}{\cdot\!\rightarrow}}{x \mp y \overset{a}{\cdots\!\rightarrow} x'} \qquad \frac{x \not\cdot\overset{a}{\cdot\!\rightarrow},\ y \overset{a}{\cdots\!\rightarrow} y'}{x \mp y \overset{a}{\cdots\!\rightarrow} y'} \qquad \frac{x \overset{a}{\cdots\!\rightarrow} x',\ y \overset{a}{\cdots\!\rightarrow} y'}{x \mp y \overset{a}{\cdots\!\rightarrow} x' \mp y'}$$

$$\frac{x\!\downarrow,\ y\!\downarrow}{x \circ y\!\downarrow} \qquad \frac{x \overset{a}{\cdots\!\rightarrow} x',\ y \overset{a}{\cdots\!\rightarrow} y'}{x \circ y \overset{a}{\cdots\!\rightarrow} x' \circ y'} \qquad \frac{x \overset{a}{\rightarrow} x'}{x \circ y \overset{a}{\rightarrow} x' \circ y} \qquad \frac{x \overset{a}{\cdots\!\rightarrow} x',\ y \overset{a}{\rightarrow} y'}{x \circ y \overset{a}{\rightarrow} x' \circ y'}$$

$$\frac{x\!\downarrow,\ y\!\downarrow}{x \parallel y\!\downarrow} \qquad \frac{x \overset{a}{\cdots\!\rightarrow} x',\ y \overset{a}{\cdots\!\rightarrow} y'}{x \parallel y \overset{a}{\cdots\!\rightarrow} x' \parallel y'} \qquad \frac{x \overset{a}{\rightarrow} x'}{x \parallel y \overset{a}{\rightarrow} x' \parallel y} \qquad \frac{y \overset{a}{\rightarrow} y'}{x \parallel y \overset{a}{\rightarrow} x \parallel y'}$$

Since the operators $\|$ and $\mp$ are symmetric and associative and ε is a unit for $\|$ and δ for $\mp$ ([3,2]), we can generalize these as follows for a finite set I:

$$\mathop{\mp}_{i\in I} P_i = \begin{cases} \delta & \text{if } I = \varnothing \\ P_j \mp \left(\mathop{\mp}_{i\in I\setminus\{j\}} P_i \right) & \text{if } j \in I \end{cases}$$

and

$$\mathop{\|}_{i\in I} P_i = \begin{cases} \varepsilon & \text{if } I = \varnothing \\ P_j \parallel \left(\mathop{\|}_{i\in I\setminus\{j\}} P_i \right) & \text{if } j \in I \end{cases}$$

The language *HMSC* can be used to describe infinitary behaviour. Therefore, we will extend our semantic domain with recursive specifications. Let Σ be an arbitrary signature, and let V be a set of recursion variables. A recursive specification $E(V)$ is a set of equations

$$\{X = s_X(V) \mid X \in V\}$$

where each $s_X(V)$ is a term over the signature Σ and the set of variables V. The set of terms constructed like this is denoted by $Rec(\Sigma)$.

Next, we will present an operational semantics for recursion (see Table 3) which generates exactly one solution for every recursive specification. Let E be a recursive specification in which X occurs as a recursion variable. Then $\langle X \mid E\rangle$ denotes the solution for X with respect to the recursive specification E. The process $\langle X \mid E\rangle$ can terminate or execute an action if its defining equation (in the context of E) can do so. For t a term possibly containing recursion variables, the process $\langle t \mid E\rangle$ denotes the process t with all occurrences of recursion variables r replaced by their solution $\langle r \mid E\rangle$. The function eqs is defined by $eqs(\langle r \mid E\rangle) = E$.

Table 3
Structured operational semantics for recursion ($X = s_X \in E$, $a \in A$).

$\dfrac{\langle s_X \mid E\rangle\downarrow}{\langle X \mid E\rangle\downarrow}$	$\dfrac{\langle s_X \mid E\rangle \xrightarrow{a} y}{\langle X \mid E\rangle \xrightarrow{a} y}$

More difficult is the definition of the permission relation for recursive specifications. The reason for this is that the recursive specification itself must be adapted.

Suppose that the recursive specification E is given by $\left\{ X^i = t^i \circ \left(\mathop{\mp}_{j\in J} X^j \right) \,\middle|\, j \in I, J \subseteq I \right\}$, where t^i is a closed term and I and J are index sets. All recursive specifications that are required for describing the semantics of High-level Message Sequence Charts are of this form.

The set of equations that result from transforming E due to the permission of action a is denoted by E_a and consists of a fresh recursion variable X^i_a for each variable X^i from E. For the equation $X^i = t^i \circ \left(\mathop{\mp}_{j\in J} X^j \right) \in E$ we introduce the equation:

- $X_a^i = t_a^i \circ \left(\mathop{\mp}_{j \in J} X_a^j \right)$ if $t^i \stackrel{a}{\cdots\!\rightarrow} t_a^i$ and $i \in J$, or
- $X_a^i = t_a^i \circ \left(\mathop{\mp}_{j \in J} X_a^j \right)$ if $t^i \stackrel{a}{\cdots\!\rightarrow} t_a^i$, $i \notin J$ and there is at least one equation for the variables X_a^j that is not of the form $X_a^j = \delta$, or
- $X_a^i = \delta$, otherwise.

Then $\langle X \mid E \rangle \stackrel{a}{\cdots\!\rightarrow} \langle X_a \mid E_a \rangle$ provided that the equation for X_a is not of the form $X_a = \delta$. We give a simple example to illustrate this. Suppose that we have the recursive specification $E = \{X = a \circ (Y \mp Z),\ Y = b \circ X,\ Z = c \circ X\}$, where a, b, and c are pairwise independent actions. Suppose that we are interested in the process $\langle X \mid E \rangle \circ b'$ where b' and b are dependent. The action b' can only be executed if the process $\langle X \mid E \rangle$ permits the execution of b'. Then we must construct the recursive specification $E_{b'}$: $E_{b'} = \{X_{b'} = a \circ (Y_{b'} \mp Z_{b'}),\ Y_{b'} = \delta,\ Z_{b'} = c \circ X_{b'}\}$. Hence $\langle X \mid E \rangle \stackrel{b'}{\cdots\!\rightarrow} \langle X_{b'} \mid E_{b'} \rangle$. Thus the process $\langle X \mid E \rangle \circ b'$ is capable of performing the action b' and thereby evolves into the process $\langle X_{b'} \mid E_{b'} \rangle$. This example shows that by permitting action b' the choice for executing the b actions is resolved.

The following theorems express the soundness of the definitions so far. They are proven using standard techniques.

Theorem 3.2.1 *The term deduction system that consists of the deduction rules introduced so far uniquely defines a transition relation.*

Theorem 3.2.2 *Strong bisimulation is a congruence with respect to the operators* $\mp$, $\circ$, *and* $\|$.

3.3. Abstract syntax of *HMSC*

A *hierarchical graph* is a mathematical structure that represents the information contents of an *HMSC*. The set *HGid* represents the set of all *HMSC* names. Obviously, this includes the names of *BMSC*s. Since we did not provide a formal graphical syntax for *HMSC* we cannot provide a formal mapping from *HMSC* to hierarchical graphs. However, the intuition is clear. A node in an *HMSC* contains a reference to another *HMSC* via its name.

Definition 3.3.1 (Hierarchical graphs) A hierarchical graph is either a *BMSC* or a tuple $\langle id, Nodes, Starts, Ends, Edges, l \rangle$, where

- $id \in HGid$ is the name of the hierarchical graph;
- *Nodes*, *Starts*, and *Ends* are pairwise disjoint sets of *HMSC* reference nodes, start nodes and end nodes respectively with $Starts \neq \emptyset$;
- $Edges \subseteq (Nodes \cup Starts) \times (Nodes \cup Ends)$ is a set of edges. An edge (n, n') is denoted by $n \rightarrow n'$;
- $l : Nodes \rightarrow HGid$ is a labeling function which associates to a node a reference to an *HMSC* by means of an *HMSC* name;

such that every node and end node is reachable from exactly one start node, and an *HMSC* is not referenced from one of its own nodes (recursively). The set of all hierarchical graphs is denoted by *HG*.

An *HMSC*-document contains a number of *HMSC*s. It is required that all *HMSC*s in an *HMSC*-document have different names. Then, an *HMSC*-document can, in the abstract syntax, be represented by a (partial) mapping $H : HGid \to HG$. For technical reasons we require that the nodes (including start and end nodes) of any two hierarchical graphs are disjoint.

3.4. Denotational semantics for *HMSC*

We will associate a recursive specification to every hierarchical graph by means of a mapping S. Since the nodes of a hierarchical graph may contain references to other hierarchical graphs, the semantic mapping S is labeled with the mapping $H : HGid \to HG$ which represents an *HMSC*-document.

A recursive specification for a hierarchical graph (say with name id) is obtained by introducing a recursion variable $\overline{id}$ and a recursion variable $\overline{n}$ for every node n in the hierarchical graph (this includes start and end nodes). The relation between these nodes is formalized by defining one recursive equation for every recursion variable introduced as follows.

- The overall behaviour of the hierarchical graph is obtained by the parallel execution of the behaviours associated to the start nodes: $\overline{id} = \Vert_{s \in Starts} \ \overline{s}$.
- For every start node s of the hierarchical graph a recursive equation $\overline{s} = \overline{n_1} \mp \cdots \mp \overline{n_m}$ is introduced, where $n_1, \cdots n_m$ are the successor nodes of start node s.
- For every *HMSC* reference node n which refers to an *HMSC* with name i (i.e., $l(n) = i$) a recursive equation $\overline{n} = \overline{i} \circ (\overline{n_1} \mp \cdots \mp \overline{n_m})$ is introduced, where $n_1, \cdots, n_m$ are the successor nodes of node n.
- For every end node e the equation $\overline{e} = \varepsilon$ is introduced.

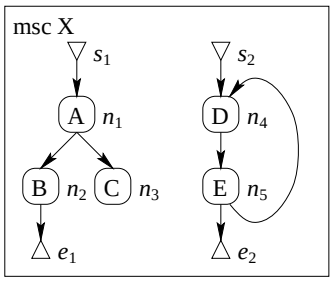
Furthermore, we also have to add equations describing the behaviour of the referenced *HMSC*s. The formal definition is given below.

Definition 3.4.1 Let $H : HGid \to HG$ be a function that represents a set of hierarchical graphs. The function $S_H : HG \to Rec(\Sigma)$ is defined as follows. If X is a *BMSC* with name id, then $S_H(X) = \langle \overline{id} \mid \{\overline{id} = S_{BMSC}(X)\} \rangle$ and if $X = \langle id, Nodes, Starts, Ends, Edges, l \rangle$ then $S_H(X) = \langle \overline{id} \mid E \rangle$ where

$$E = \left\{ \begin{array}{l} \overline{id} = \Vert_{s \in Starts} \ \overline{s}, \quad \overline{s} = \mathop{\mp}_{s \to n \in Edges} \overline{n}, \\ \overline{n} = \overline{l(n)} \circ \left(\mathop{\mp}_{n \to n' \in Edges} \overline{n'} \right), \\ \overline{e} = \varepsilon \end{array} \middle| \begin{array}{l} s \in Starts, \\ n \in Nodes, \\ e \in Ends \end{array} \right\} \cup \bigcup_{n \in Nodes} eqs(S_H(H(l(n)))).$$

From now on, if no confusion can arise, we will denote a recursion variable by n instead of $\overline{n}$.

Example 3.4.2 Consider the *HMSC* shown in Figure 2. In Figure 5 the same *HMSC* is shown with the names of the nodes in the abstract syntax as annotation. Suppose that the semantics of the *HMSC*s A, B, C, D, E are given by $\langle r_A \mid E_A \rangle, \cdots, \langle r_E \mid E_E \rangle$ respectively. Then the semantics of *HMSC* X is given by $\langle X \mid E \rangle$ where E consists of the equations of $E_A, \cdots, E_E$, and additionally the equations shown in Figure 5.



$$
\begin{aligned}
X &= s_1 \parallel s_2 \\
s_1 &= n_1 \\
s_2 &= n_4 \\
n_1 &= r_A \circ (n_2 \mp n_3) \\
n_2 &= r_B \circ e_1 \\
n_3 &= r_C \circ \delta \\
n_4 &= r_D \circ n_5 \\
n_5 &= r_E \circ (n_4 \mp e_2) \\
e_1 &= \varepsilon \\
e_2 &= \varepsilon
\end{aligned}
$$

Figure 5. *HMSC* annotated with the recursion variables and the equations for those variables.

4. EXAMPLE: THE ALTERNATING BIT PROTOCOL

In this section, we will give a description of the behaviour of the Alternating Bit Protocol (*ABP*) in *HMSC*. The *ABP* is developed for the transmission of data from one entity (the sender S) to another entity (the receiver R) by means of an unreliable communication medium; channel K from S to R for messages and channel L from R to S for acknowledgements (see Figure 6).

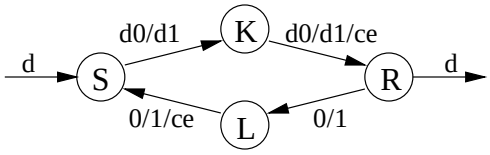


Figure 6. The architecture of the *ABP*.

It is assumed that media errors can be detected, e.g., by means of a checksum error (*ce*). Other faults, such as message loss will not occur. In Figure 6 we have added which messages may be transferred between the entities. In communication with the environment, only plain data items d play a role. From S to R frames are transmitted which consist of a data item and a bit value (e.g. $d0$). Furthermore channel K may send checksum errors (*ce*) to the receiver. Channel L is used to transmit acknowledgment bits 0 and 1 and it may produce a checksum error.

The specification will be presented in a top-down fashion. From the overall description in Figure 7 (msc *ABP*), we learn that it operates in two alternating phases: a *0-phase* and a *1-phase*.

The two phases are similar, except that the bits 0 and 1 are swapped. In Figure 7 we will give the 0-phase only. It shows that there is one main scenario, namely successful transmission. This trace starts with an input of a datum from the environment (*S-in*), followed by successful transmission of this datum in phase 0 (*tr-ok-0*). Next, the datum is sent to the environment (*R-out*) and the transmission of the acknowledgement in phase 0 succeeds (*ack-ok-0*).

There are two places where a deviation of the main scenario may occur (viz. the two loops in the msc *0-phase*). The first problem that is anticipated at, is an erroneous transmission (the upper loop in msc *0-phase*). If this happens, a phase 1 acknowledgement is issued and, regardless whether this acknowledgement arrives correctly, the transmission is repeated. In the same way an error in the acknowledgement channel is solved (the lower loop).

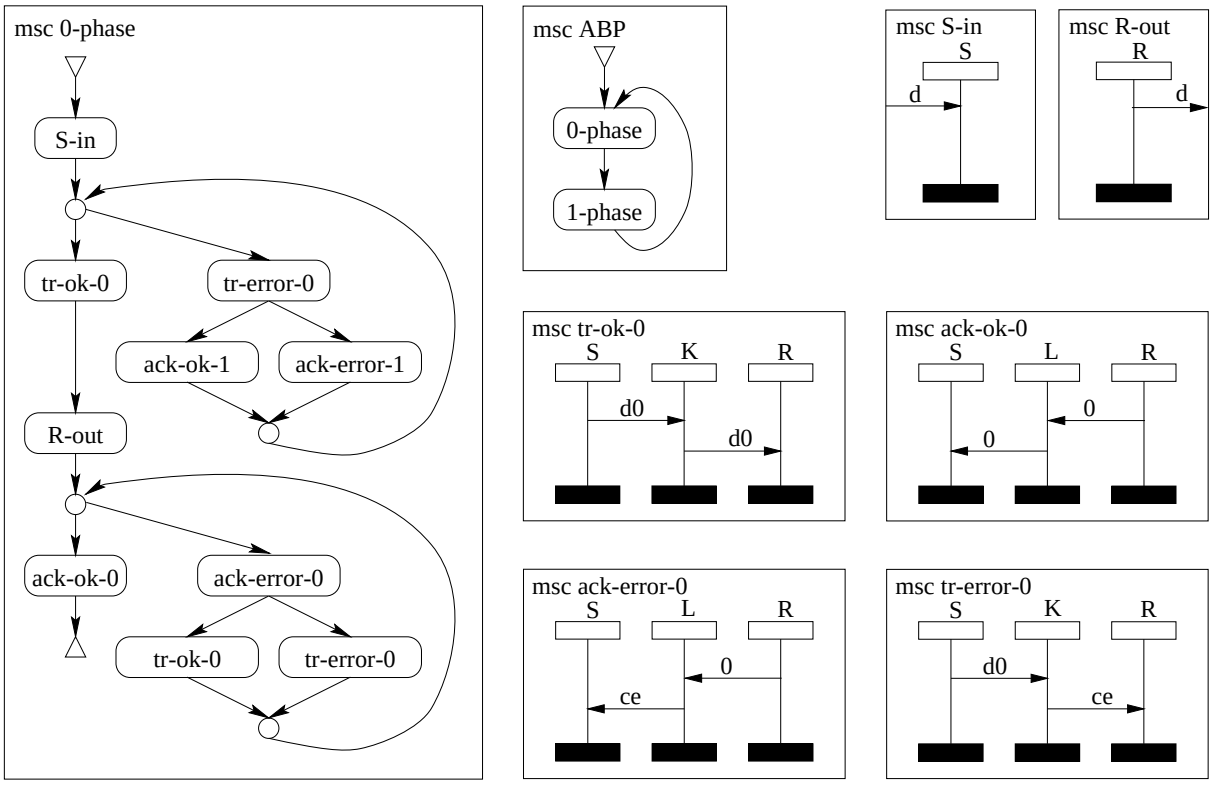


Figure 7. Specification of the *ABP*.

Finally, we give the lowest level definitions of the *MSC*s that occur in the phase descriptions (see Figure 7). For instance, msc *tr-ok-0* contains the description of correct transmission of a datum *d* in phase 0 from *S* to *R*. Notice that the datum is attributed with the phase bit. The *MSC*s *tr-ok-1*, *tr-error-1*, *ack-ok-1* and *ack-error-1* are not displayed. They can be easily derived from their 0-phase counterparts.

We can draw several conclusions from this example. First, the decomposition elaborated here is just one of many possible descriptions of the same protocol (see e.g. [4]). It has the virtue that it helps in understanding the overall operation of the protocol. If one is interested in under-

standing the behaviour of the entities in separation, a horizontal decomposition technique can be used. This way, we obtain the parallel composition of the four instances involved. Since there is communication between these components, we need a notion of *gates* for defining the interface between the components (see [6]). We elaborate on this issue in Section 5.

Next, observe that we have defined several *MSC*s that differ only in the phase bit. A shorter specification can be easily obtained by introducing parameterized specifications. This is closely related with extending the *MSC* language with data types. In the current description, it is not stated that the sender can receive any datum d from the environment. It rather says that the same input message d is repeated every time an input occurs. This can only be viewed as an abstraction.

5. GATES IN *HMSC*

5.1. Why gates?

As shown in the *ABP* example from Section 4, *HMSC* proves to be very useful for most forms of (de)composition. Vertical composition is denoted by linking nodes with arrows (weak sequential composition), alternatives are denoted by allowing more than one successor node (delayed choice) and horizontal composition is denoted by juxtaposition of simpler constructs (free merge).

However, we consider the free merge as too weak. It can only be used for horizontal decomposition if the components have no communication interaction. The free merge works only for *free components*. Several alternative merge operators have been suggested, such as the Interworking merge [12] and the environmental merge [17,18].

We propose to extend *HMSC* as defined in *MSC*96 with gates and to adapt the merge operator in order to handle the proper linkage of gates (see also [6] for a discussion on gates).

Gates are already part of *MSC*96, but not at the level of *HMSC*. A gate is a point at the frame of a simple *MSC* construction at which a message starts or ends. A gate may be identified by the name of the message, or by an explicit gate name. When combining simple *MSC*s that contain gates into more complex *MSC*s, a gate is either bound to some other gate or it is inherited by the compound. In the same way we can construct gates in *HMSC*, which is not allowed in *MSC*96.

When allowing gates in *HMSC*, two obvious ways for binding come up. The first option is to require explicit binding of gates by means of a message symbol. Although this is very much in the spirit of *MSC*, examples show that an *HMSC* becomes crowded with crossing lines and looses its purpose for overview specification.

The second possibility is to bind gates by name identification. That is, an input and an output gate are connected if they have the same gate name. This makes it harder to find out which gates are connected, but yields a much more quiet picture. In order to have this binding by gate name identification, a possibility must exist for renaming gates. Unfortunately, in *MSC*96 substitution of gates is not allowed. Since we cannot find any semantical or logical reasons to exclude gates from substitutions, we propose to extend *MSC*96 in this respect.

5.2. The Alternating Bit Protocol revisited

The purpose of this section is to give an *MSC* specification of the *ABP* from an instance oriented point of view. In the previous *ABP* specification the emphasis was on the overall behaviour of the complete system. This is a nice way to get an overview of the protocol itself, but it does not describe as clearly the behaviour of each component in isolation.

Here, we have a more implementation directed view, closer to the *SDL* way of specification. The components of the system are represented by separate *HMSC*s that operate in parallel. They communicate via gates. The behaviours of the components are clearly separated. This style of specification is very suitable as an intermediate stage between the *classical* use of *MSC* and *SDL*.

Before giving an explanation of the *ABP* in Figure 8, we will first point out some additional differences between our diagrams and *MSC*96. First, we allow complete Basic Message Sequence Charts to occur in *HMSC* nodes, rather than references to *BMSC*s. Experience showed that this makes the drawings much easier to understand. Second, we omitted the *instance start* and *instance end* symbol for making the drawing less crowded. The third difference is that we omit the abundant frame surrounding a parallel construct.

The *ABP* specification in Figure 8 can be understood as follows. It consists of four concurrently operating entities. The sender S and the receiver R and two channels. Channel K connects the sender to the receiver and channel L connects the receiver to the sender (see Figure 6 and see the names of the gates in Figure 8). Looking at the behaviour of the channels first, we see that channel K repeatedly receives some data frame via gate g and sends either the same frame to gate h, or some specific value called *ce* (for checksum error). Channel L exposes the same behaviour, be it that input is received from gate i and sent to gate j. This channel is only capable of transmitting bits.

Notice that we have sketched the behaviour of the channel using four scenarios. Two basic scenarios in which transmission succeeds and two faulty scenarios leading to a checksum error. The alternative composition of these scenarios does not imply that the choice between correct operation and faulty operation is resolved before the reception of the input. Due to the use of the delayed choice operator, the choice is rather made after reception of the frame.

The behaviour of the sender S is a repetition of the following. The sender starts in the so-called zero-phase. First some data d to transmit is received from the environment. This data is extended with the bit 0 and yields the frame $d0$ which is sent along gate g. Then the sender awaits an incoming message from gate j. In case this is a negative acknowledgement (1) or a checksum error (*ce*) generated by channel L, the transmission of frame $d0$ is repeated. In case of a positive acknowledgement (0), the zero-phase is concluded and the process starts the one-phase, which is the converse of the zero-phase.

The behaviour of the receiver R can be explained likewise. The receiver also starts in the zero-phase and awaits the correct transmission of a zero-frame via gate h. The first time this frame is received the data contained is copied to the environment. Then the receiver enters the one-phase.

6. CONCLUDING REMARKS

The expressive power of the *MSC* language has increased considerably since the recent introduction of new constructs. We have studied the sub-language of *HMSC*s in isolation, in order to assess this feature's use and semantics.

It is perfectly feasible to define a semantics of *HMSC* in the line of the already existing process algebra semantics of *MSC*92. It is based upon the known process algebra operators for weak sequencing, delayed choice and free merge. These operators are already incorporated into *MSC*96 on the level of *MSC* expressions. In order to capture the infinite behaviour of a system description in *HMSC*, we have used recursive equations. This yields a sound operational semantics.

By explaining the use of *HMSC* by means of the well-known toy example of the *ABP*, we

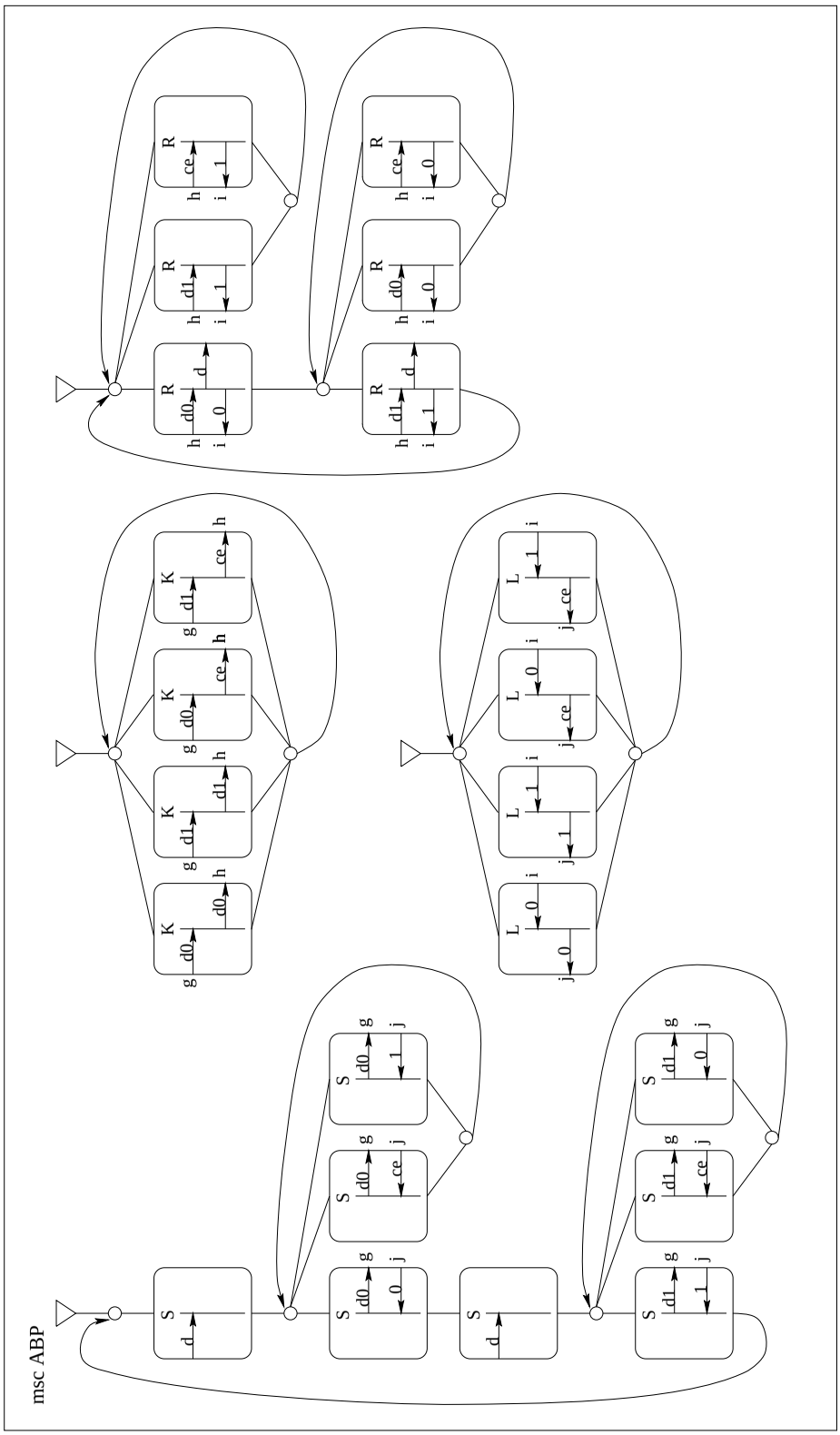


Figure 8. The *ABP*: process oriented specification.

have argued that *HMSC* is very suited for overview descriptions. *HMSC* is mainly used here for vertical decomposition and displaying alternative scenarios. However, a more process oriented view, in which the system is decomposed into communicating subsystems, is hard to achieve with *HMSC*. Therefore, we propose to add gates to the language. By using gates, we were able to give a specification of the components of the *ABP* in a more *SDL*-like style.

We have presented two views of the *ABP*: an overall description expressing the control-flow and an instance oriented description. In principle an instance oriented description can be obtained from the overall description as follows. For every instance we first make a copy of the *HMSC* and then delete all other instances from the *MSC*s. If in this process we break a complete communication into an output and an input event, we need to introduce a gate which is later used to connect the two parts of the communication. In general this will result in an instance oriented description which may not be optimal with respect to readability. We can transform this initial instance oriented description into a more tractable form. The result of this transformation on the *ABP* example is shown in Figure 8. Of course we need to show that these transformation are semantically correct. For this purpose a formal definition as as presented is necessary.

We think that the possibility to switch between different views within the same language fits very well with the variety of uses of the *MSC* language. A thorough study on the (formal) relation between these different views is apparently an important step towards the (semi-) automatic derivation of *SDL* code from *MSC* scenarios. It may be expected that the extension of *HMSC* with gates does not introduce semantical difficulties.

Our simple case study also revealed two more shortcomings of *MSC*96. First, since data is not incorporated in the language, we were not able to give a full specification of the *ABP*. We abstracted from the actual set of data to be transmitted by giving them the same name d. Second, our specification would benefit from being able to reuse *MSC*s that only differ in the value of the alternating bit. A parameterization mechanism will show helpful.

Finally, we mention two minor issues. The *HMSC* specification of the *ABP* benefits from the possibility to consider complete *MSC*s as a node in an *HMSC* specification, rather than a reference. Additionally, there are reasons for removing the restriction that a gate can not be substituted.

REFERENCES

1. R. Alur, G. J. Holzmann, and D. Peled. An analyzer for Message Sequence Charts. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 35–48, Passau, Germany, 1996. Springer-Verlag.
2. J. C. M. Baeten and S. Mauw. Delayed choice: an operator for joining Message Sequence Charts. In D. Hogrefe and S. Leue, editors, *Formal Description Techniques VII*, IFIP Transactions C, Proceedings Seventh International Conference on Formal Description Techniques, pages 340–354. Chapman & Hall, 1995.
3. J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.
4. L. M. G. Feijs. Synchronous sequence charts in action. Technical Report CSR 95-25, Eindhoven University of Technology, Department of Computing Science, 1995.
5. J. Grabowski, P. Graubmann, and E. Rudolph. Towards a Petri net based semantics defini-

tion for Message Sequence Charts. In O. Færgemand and A. Sarma, editors, *SDL'93 - Using Objects*, Proceedings of the Sixth SDL Forum, pages 179–190, Darmstadt, 1993. Amsterdam, North-Holland.

6. Ø. Haugen. MSC structural concepts. Technical Report TD 9006, ITU-T Experts Meeting SG 10, Turin, 1994.
7. ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva. Publication scheduled 1997.
8. ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, September 1993.
9. ITU-TS. *ITU-TS Recommendation Z.120 Annex B: Algebraic semantics of Message Sequence Charts*. ITU-TS, Geneva, April 1995.
10. P.B. Ladkin and S. Leue. Interpreting message flow graphs. *Formal Aspects of Computing*, 7(5):473–509, 1995.
11. S. Mauw and M. A. Reniers. An algebraic semantics of Basic Message Sequence Charts. *The Computer Journal*, 37(4):269–277, 1994.
12. S. Mauw, M. van Wijk, and T. Winter. A formal semantics of synchronous Interworkings. In O. Færgemand and A. Sarma, editors, *SDL'93 - Using Objects*, Proceedings of the Sixth SDL Forum, pages 167–178, Darmstadt, 1993. Amsterdam, North-Holland.
13. D. M. R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings 5th GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1980.
14. G. D. Plotkin. A structural approach to operational semantics. Technical Report DIAMI FN-19, Computer Science Department, Aarhus University, 1981.
15. A. Rensink and H. Wehrheim. Weak sequential composition in process algebras. In B. Jonsson and J. Parrow, editors, *CONCUR'94: Concurrency Theory*, volume 836 of *Lecture Notes in Computer Science*, pages 226–241, Uppsala, 1994. Springer-Verlag.
16. E. Rudolph. MSC roadmaps. Towards a synthesized solution. Technical Report TD 9017, Experts Meeting ITU-TS SG 10, 1995.
17. E. Rudolph, P. Graubmann, and J. Grabowski. Message Sequence Chart: composition techniques versus OO-techniques - 'tema con variazioni'. In R. Bræk and A. Sarma, editors, *Proceedings Seventh SDL Forum*, Oslo, 1995. Amsterdam, North-Holland.
18. E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, 1996. Special issue on SDL and MSC, guest editor Ø. Haugen.