A GRAPHICAL PARALLEL COMPOSITION OPERATOR FOR PROCESS ALGEBRAS

Hubert Garavel and Mihaela Sighireanu
INRIA Rhône-Alpes / VASY team
655, avenue de l'Europe
38330 Montbonnot St Martin
France
hubert.garavel@inria.fr, mihaela.sighireanu@inria.fr, http://www.inrialpes.fr/vasy

Abstract Process algebras are suitable for describing networks of communicating processes. In most process algebras, the description of such networks is achieved using parallel composition operators. Noticing that the parallel composition operators commonly found in process algebras are often limited in expressiveness and/or difficult for novice users, we propose a new parallel operator that allows networks of communicating processes to be described easily, in a simple and well-structured manner. We illustrate on various examples (token-ring network and client-server protocol) the theoretical and practical merits of this operator.

Keywords: Concurrency, E-LOTOS, Formal Description Technique, LOTOS, Process Algebra, Protocol.

1. INTRODUCTION

Process algebras have been designed as a theoretical framework for the study of concurrency. Classical examples of process algebras are: ACP [BK84], CCS [Mil80; Mil89], CSP [Hoa85], MEIJE [dS85], etc. There also exist specification languages combining process algebraic concepts with features borrowed from functional or imperative programming languages, e.g., the OCCAM [Cam89] language based on CSP, the μCRL [GP91] language based on ACP, and the LOTOS [ISO88] language which combines the best features of CSP and CCS.

Process algebras have undeniable advantages: expressiveness, compositionality, formal semantics given in terms of Labelled Transition Systems (LTS) [Par81] using structural operational semantics [Plo81; GV92], verification algorithms based on behavioural equivalences and preorders, refinement methods, etc. Process algebras have been used successfully many times to model the behaviour of real systems. In addition, simulators, model-checkers, and theorem-provers are available for analyzing process algebraic descriptions, e.g., [DG95; CMS95; FGK$^+$96].

In spite of these advantages, the usual process algebras suffer from limitations in terms of usability (because of their steep learning curve, they often require a substantial training effort), readability (process algebraic descriptions are sometimes difficult to understand), and coverage (important aspects of system description, such as timing, probabilistic aspects, and priorities, are not addressed, although various extensions have been discussed in the literature).

Fortunately, work is going on to extend and improve the mainstream process algebras. In particular, the International Standardization Organization (ISO) has been working since 1992 on the definition of a revised version of the LOTOS language. This revised version, named E-LOTOS and currently at the stage of Final Committee Draft [Que98], includes new features suitable for increasing both the expressiveness and user-friendliness of the language. The work on E-LOTOS has generated many proposals for enhancing both the data type part and the behaviour part of LOTOS (see, e.g., [GS98] for an overview and a discussion on these issues).

In this paper, we focus our attention on the improvement of the parallel composition operators of LOTOS. Although we assume some basic knowledge of LOTOS, our proposals could certainly be applied to other process algebras.

The paper is structured as follows. Section 2. introduces basic definitions and notations. Section 3. suggests to replace the binary parallel composition operators found in most process algebras with a new n-ary operator, better suitable for an easy description of networks of communicating processes. Section 4. proposes further enhancements to this operator, by relaxing the *maximal cooperation* paradigm used in process algebras such as CSP or LOTOS. Section 5. illustrates the usefulness of this parallel operator on a concrete application, the ODP[1] trading function [ISO95a]. Finally, Section 6. gives some concluding remarks.

2. BASIC DEFINITIONS AND NOTATIONS

In the sequel, we use the following notations borrowed from the value-passing process algebras (especially, LOTOS) terminology.

We denote with $B_1, B_2, ...$ the algebraic terms constructed using the standard behavioural operators (inaction, action prefix, choice, etc.); these terms are called *behaviours* or *processes*[2]. For our purpose, an exact syntactic definition of behaviours is not required. We denote with "$B_1 = B_2$" the syntactic identity of terms B_1 and B_2.

We denote with $G_1, G_2, ...$ the identifiers corresponding to communication points; these identifiers are called *gates*. We define two particular gates: τ, which denotes a non-observable event, and δ, which is used to express the synchronized termination of concurrent behaviours. We denote with $\widehat{G_1}, \widehat{G_2}, ...$ the sets of gates.

We denote with $L_1, L_2, ...$ the tuples of the form $\langle G, v_1, ..., v_n \rangle$, where G is a gate and $v_1, ..., v_n$ a (possibly empty) list of typed values; these tuples are called *actions* or *labels*. We denote with $gate(L)$ the gate corresponding to the first element of the tuple L.

Structural operational semantics defines how a behaviour is translated into a (possibly infinite) *labelled transition system* [Par81], which represents all the possible evolutions of the behaviour. The labelled transition system is defined by a transition relation noted "$B_1 \xrightarrow{L} B_2$", which expresses that B_1 can perform an action L and mute to B_2 afterwards.

3. FROM BINARY TO N-ARY PARALLEL COMPOSITION OPERATORS

In LOTOS and most process algebras, parallel composition operators play a central role in the description of concurrent systems. Basically, there are two main uses of parallel composition:

- Parallel composition is the natural mean to describe a set of distributed components that execute concurrently and communicate with each other by message passing: in such an approach, the operands of a parallel composition operator correspond to physically distributed entities. In the taxonomy proposed by [VSS88], this use of parallel composition is called the *resource-oriented specification style.*

- Parallel composition can also be used for refinement purpose. Typically, a given (possibly sequential) component can be divided into a set of sub-components, each of which expresses temporal constraints on the occurrences of certain events. These sub-components are combined together using parallel composition, which acts as the logical conjunction of the corresponding temporal constraints, thus leading to a constrained behaviour. In such case, parallel composition expresses neither physical distribution nor concurrency, but rather a logical modularization of a complex component. In the taxonomy of [VSS88], this use of parallel composition is called the *constraint-oriented specification style.*

Because of this double use of parallel composition, we believe that a suitable parallel composition operator must support *multiway synchronization*, i.e., rendezvous synchronization involving more than two behaviours:

- As far as resource-oriented style is concerned, multiway synchronization is not really necessary: two-way synchronization is sufficient to describe the communication between an emitter and a receiver. Most process algebras allow such *handshaking* synchronization, with some differences with respect to the form of value exchanges that take place during the synchronization.

- But, as far as constraint-oriented style is concerned, multiway synchronization is mandatory. For instance, a controller for a robot with n degrees of freedom can be expressed as the parallel composition of n sub-processes, each sub-process controlling the motion of the robot with respect to a given degree of freedom; to perform a given mission (e.g., moving the robot from one location to another one), all sub-processes have to synchronize.

With the notable exception of CCS, most process algebras (ACP, CSP, MEIJE, LOTOS...) support multiway synchronization, which is clearly a desirable feature. Yet, many process algebras rely on (associative) binary parallel composition operators to express multiway synchronization between n processes. This can be explained by the fact that the original motivation behind the design of process algebras was the search for a "minimal" model of concurrency.

For instance, LOTOS has three parallel operators, noted "B_1 |[$\widehat{G}$]| B_2", "B_1 ||| B_2", and "B_1 || B_2", respectively. The first operator is the most general: it expresses that B_1 and B_2 execute concurrently and synchronize only on the gates of $\widehat{G} \cup \{\delta\}$. The second and third operators are particular cases of the former: "|||" corresponds to the case where $\widehat{G}$ is empty (fully asynchronous execution) and "||" to the case where $\widehat{G}$ is the set of all visible gates (fully synchronous execution). From our experience in describing complex, industrial systems using LOTOS, we believe that expressing parallel composition using binary operators has major drawbacks:

- For a given network of concurrent processes, there are usually several different algebraic terms representing this network. For instance, the network shown on Figure 1.1 can be described using two (equivalent) LOTOS terms, e.g.,

$$((B_1 \;|[G_1]|\; B_2) \;|[G_1, G_3, G_4]|\; B_3) \;|[G_2, G_3, G_4]|\; (B_4 \;|[G_2]|\; B_5)$$

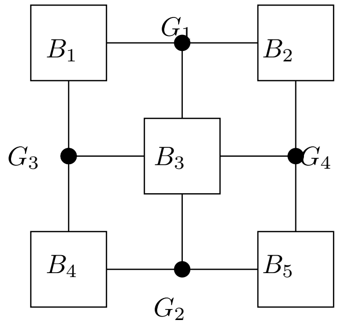

Figure 1.1

or

$$((B_1 \;\texttt{|[}G_1\texttt{]|}\; B_2) \;\texttt{|||}\; (B_4 \;\texttt{|[}G_2\texttt{]|}\; B_5)) \;\texttt{|[}G_1, G_2, G_3, G_4\texttt{]|}\; B_3$$

The absence of a canonical form is practically unfortunate, as an algebraic description will strongly depend on the style adopted by its author, thus leading to a lack of uniformity. Moreover, the problem of determining whether two terms are equivalent (for all possible sub-terms B_1, B_2, etc.) is decidable, but not immediate in the general case, as it implies to solve a system of boolean equations [Kar94; Kar97].

- There are some process networks that can not be expressed as algebraic terms. For instance, the network on Figure 1.2 can not be expressed using LOTOS parallel composition, because it involves two-by-two synchronization on the same gate G, whereas LOTOS would force all three processes to synchronize on G using a three-way rendezvous (this is called the *maximal cooperation* paradigm). Sufficient conditions for a process network to be translated into a LOTOS behaviour expression are studied in [Bol90]. Although the example of Figure 1.2 may seem somehow artificial, Section 5. will show that networks involving two-by-two synchronization are useful for describing client-server architectures.

Besides these theoretical issues, there are also pragmatic considerations against binary parallel composition operators. The main argument is that binary operators create a discrepancy between the graphical representation of process networks (always present in the designer's mind)

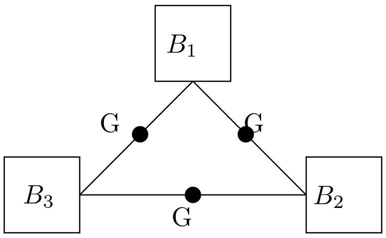

Figure 1.2

and the textual representation as an algebraic term. On the one hand, it is not easy for novice users to write an algebraic term corresponding to a given network of concurrent processes. On the other hand, given an algebraic term, it is not always immediate to infer the corresponding network.

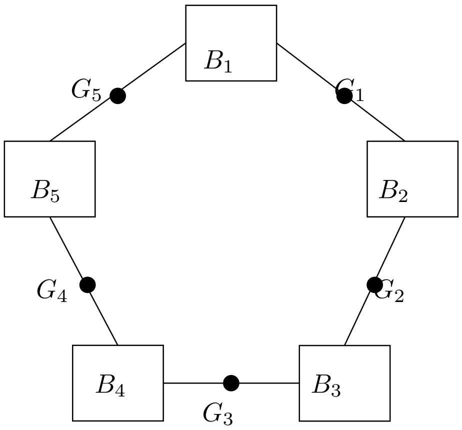

Figure 1.3

Some network topologies are particularly tricky to express using binary operators. For instance, the simplest algebraic term for representing the token-ring network shown on Figure 1.3 is:

$$(B_1 \;|[G_1]|\; B_2 \;|[G_2]|\; B_3 \;|[G_3]|\; B_4) \;|[G_4, G_5]|\; B_5$$

which is particularly non-intuitive because the circular symmetry of the network cannot be preserved during the translation to an algebraic term. In this respect, the difficulties inherent to the process algebraic approach have to be compared with graphical formalisms such as SDL [IT92] or Statecharts [Har87], in which the user simply has to draw the desired network.

For these reasons, we suggested to introduce in E-LOTOS a new n-ary parallel composition operator that would replace the binary operators of LOTOS. Based on an early suggestion by [Bri88], we made several iterative proposals [Gar95; SG96], before our proposal was accepted for being included in E-LOTOS. The basic syntax of the n-ary parallel composition operator is:

$$
\begin{array}{ll}
\textbf{par} & \\
 & \widehat{G_1} \rightarrow B_1 \\
|| & \widehat{G_2} \rightarrow B_2 \\
|| & ... \\
|| & \widehat{G_n} \rightarrow B_n \\
\textbf{endpar} &
\end{array}
$$

This operator describes a network of $n \geq 1$ concurrent behaviours $B_1, ..., B_n$. We define I to be the set $\{1, ..., n\}$. To each behaviour B_i is associated an *interface* consisting of a set of gates $\widehat{G_i}$ on which B_i must synchronize. Each $\widehat{G_i}$ can be empty; in such case the arrow before B_i can be omitted.

We later discovered that such an extended parallel composition operator had been already proposed in, at least, two occasions: [Bol90] suggests that process networks could be described using an n-ary operator (noted *MaxCoop*) and gives rules for translating process networks into strongly equivalent LOTOS behaviour expressions ; [DS92; DS95] include in CSP a similar operator, the semantics of which is expressed in terms of traces.

Instead, we define the n-ary parallel composition operator by means of two rules of structured operational semantics. The first rule expresses that any behaviour B_i can execute asynchronously any action L whose gate G does not belong to the interface $\widehat{G_i}$ and is different from the termination gate δ (this encompasses the case where $L = \tau$), while the other behaviours B_j with $j \neq i$ do not evolve:

$$
\frac{(\exists L) \ \ (\exists i \in I) \ \ B_i \xrightarrow{L} B_i' \ \wedge \ \mathit{gate}(L) \notin \widehat{G_i} \cup \{\delta\} \ \wedge \ (\forall j \in I \backslash \{i\}) \ \ B_j' = B_j}{\textbf{par} \ \widehat{G_1} \rightarrow B_1 ... \widehat{G_n} \rightarrow B_n \ \textbf{endpar} \xrightarrow{L} \textbf{par} \ \widehat{G_1} \rightarrow B_1' ... \widehat{G_n} \rightarrow B_n' \ \textbf{endpar}}
$$

The second rule expresses that a behaviour B_i wanting to execute an action L labelled by a gate $G \in \widehat{G_i} \cup \{\delta\}$ must synchronize with all the other behaviours B_j such that $G \in \widehat{G_j} \cup \{\delta\}$:

$$
\frac{(\exists L) \ \ (\forall i \in I) \ \ (\mathit{if} \ \mathit{gate}(L) \in \widehat{G_i} \cup \{\delta\} \ \mathit{then} \ B_i \xrightarrow{L} B_i' \ \mathit{else} \ B_i' = B_i)}{\textbf{par} \ \widehat{G_1} \rightarrow B_1 ... \widehat{G_n} \rightarrow B_n \ \textbf{endpar} \xrightarrow{L} \textbf{par} \ \widehat{G_1} \rightarrow B_1' ... \widehat{G_n} \rightarrow B_n' \ \textbf{endpar}}
$$

This operator solves the aforementioned problems of binary operators by establishing a direct mapping between process networks and their textual representation, thus paving the way for tools that automatically perform the translation from graphical networks to algebraic terms and vice versa. For instance, the networks of Figures 1.1 and 1.3 can be expressed as follows:

$$\begin{array}{ll}
\textbf{par} & \\
 & G_1, G_3 \rightarrow B_1 \\
 \mid\mid & G_1, G_4 \rightarrow B_2 \\
 \mid\mid & G_1, G_2, G_3, G_4 \rightarrow B_3 \\
 \mid\mid & G_2, G_3 \rightarrow B_4 \\
 \mid\mid & G_2, G_4 \rightarrow B_5 \\
\textbf{endpar} &
\end{array}$$

and:

$$\begin{array}{ll}
\textbf{par} & \\
 & G_1, G_5 \rightarrow B_1 \\
 \mid\mid & G_1, G_2 \rightarrow B_2 \\
 \mid\mid & G_2, G_3 \rightarrow B_3 \\
 \mid\mid & G_3, G_4 \rightarrow B_4 \\
 \mid\mid & G_4, G_5 \rightarrow B_5 \\
\textbf{endpar} &
\end{array}$$

respectively.

As far as expressiveness is concerned, it is obvious that the general parallel operator "$\mid[\widehat{G}]\mid$" of LOTOS can be obtained as a particular case of the n-ary operator:

$$B_1 \;\mid[\widehat{G}]\mid\; B_2 \;=\; \textbf{par}\ \widehat{G} \rightarrow B_1 \;\mid\mid\; \widehat{G} \rightarrow B_2\ \textbf{endpar}$$

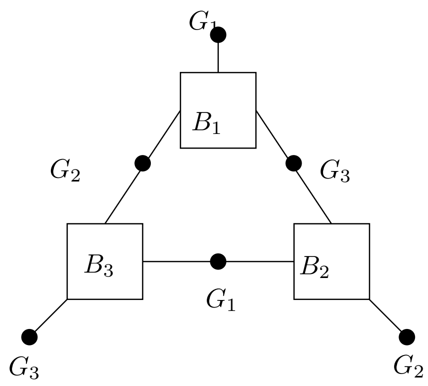

Figure 1.4

Reciprocally, the n-ary operator is strictly more expressive than the "$|[\widehat{G}]|$" operator of LOTOS. We prove this proposition using the process network shown on Figure 1.4 (which does not satisfy the sufficient conditions of [Bol90] because, for instance, the three processes can access gate G_1 but do not synchronize altogether on this gate). This process can easily be described using the n-ary operator:

$$
\begin{array}{l}
\textbf{par} \\
\qquad G_2, G_3 \rightarrow B_1 \\
\quad || \; G_1, G_3 \rightarrow B_2 \\
\quad || \; G_1, G_2 \rightarrow B_3 \\
\textbf{endpar}
\end{array}
$$

but cannot be expressed using the LOTOS binary parallel operators: to describe this network, one must first synchronize two processes together, then synchronize the result with the third process; assuming that B_1 and B_2 are to be synchronized first (which can be done without loss of generality because the network is symmetric), it is mandatory to synchronize them on gate G_3; then, the resulting term "$B_1 \; |[G_3]| \; B_2$" has to be synchronized with B_3 on gates G_1 and G_2. But the term obtained does not correspond to the network of Figure 1.4 where process B_1 can perform actions on gate G_1 independently from process B_3.

As a side remark, it is worth noticing that the network of Figure 1.4 can only be expressed by combining the "$|[\widehat{G}]|$" operator of LOTOS together with process instantiation. Technically, this can be done by defining an auxiliary process P with auxiliary gates G_1', G_2', G_3', and by a clever instantiation of this process so as to rename the auxiliary gates into G_1, G_2, G_3 respectively:

$$
\begin{array}{l}
P[G_1, G_2, G_3, G_1, G_2, G_3] \\
\textbf{where} \\
\textbf{process} \; P[G_1, G_2, G_3, G_1', G_2', G_3'] := \\
(B_1[G_1', G_2, G_3] \; |[G_3]| \; B_2[G_1, G_2', G_3]) \; |[G_1, G_2]| \; B_3[G_1, G_2, G_3'] \\
\textbf{endproc}
\end{array}
$$

The same result could be achieved using the relabelling operator existing in other process algebras, such as ACP or CSP (in LOTOS, the process instantiation performs relabelling implicitly). However, this solution is probably too tricky for most users; in any case, the n-ary parallel operator is simpler and more intuitive.

Finally, we slightly extend the n-ary operator by allowing to specify a set $\widehat{G_0}$ of synchronization gates common to all processes B_i (assuming that $\widehat{G_0} \cap \widehat{G_i} = \emptyset$ for each $i \in I$). This extension is practically helpful for avoiding redundant lists of gates; it is simply defined as a syntactic shorthand (where "$\uplus$" denotes disjoint union of sets):

$$\left(\begin{array}{l}\textbf{par}\ \widehat{G_0}\ \textbf{in}\\ \quad\ \ \widehat{G_1} \rightarrow B_1\\ \ |\,|\ \widehat{G_2} \rightarrow B_2\\ \ |\,|\ ...\\ \ |\,|\ \widehat{G_n} \rightarrow B_n\\ \textbf{endpar}\end{array}\right) =_{def} \left(\begin{array}{l}\textbf{par}\\ \quad\ \ \widehat{G_0} \uplus \widehat{G_1} \rightarrow B_1\\ \ |\,|\ \widehat{G_0} \uplus \widehat{G_2} \rightarrow B_2\\ \ |\,|\ ...\\ \ |\,|\ \widehat{G_0} \uplus \widehat{G_n} \rightarrow B_n\\ \textbf{endpar}\end{array}\right)$$

4. FROM MAXIMAL TO "M AMONG N" COOPERATION

Although superior to the LOTOS parallel composition operator, the n-ary operator described in Section 3. is not expressive enough to represent certain process networks, such as the one of Figure 1.2. This limitation is unfortunate, because it precludes several networks of practical interest from being modelled, especially the case where a pool of n processes synchronize two by two on the same gate. Although CCS permits such "2 among n" synchronization, other process algebras, such as CSP or LOTOS, do not allow it, because they rely on the maximal cooperation paradigm.

Based on our practical experience, we suggest to extend the n-ary operator in order to allow "m among n" synchronization, i.e., when a set of n processes synchronize m by m on the same gate (with $m \leq n$). Our extended operator is based on our previous proposals [Gar95; SG96] submitted to the E-LOTOS standardization Committee. This operator has the following syntax:

$$\begin{array}{l}\textbf{par}\ g_1\texttt{\#}m_1, g_2\texttt{\#}m_2, ..., g_p\texttt{\#}m_p\ \textbf{in}\\ \quad\ \ \widehat{G_1} \rightarrow B_1\\ \ |\,|\ \widehat{G_2} \rightarrow B_2\\ \ |\,|\ ...\\ \ |\,|\ \widehat{G_n} \rightarrow B_n\\ \textbf{endpar}\end{array}$$

where $g_1, ..., g_p$ is a (possibly empty) list of gates and where $m_1, ..., m_p$ are natural numbers in the range *1, ..., n* associated to these gates. Each clause "$\texttt{\#}m_j$" is optional: if omitted, m_j has the default value n. We define $\widehat{G_0}$ to be the set of gates $\{g_1, ..., g_p\}$ and we require that $\widehat{G_0} \cap \widehat{G_i} = \emptyset$ for $i \in I$. Notice that we do not require the gates $g_1, ..., g_p$ to be pairwise distinct.

Informally, the semantics of this operator is the following. As regards the gates of $\widehat{G_1} \cup ... \cup \widehat{G_n} \cup \{\delta\}$, this operator behaves exactly as the one described in Section 3.. As regards the gates of $\widehat{G_0}$, this operator specifies that processes $B_1, ..., B_n$ can perform m_j among n synchronization on each gate g_j. Two special cases are of interest: if $m_j = 1$, each process B_i can execute asynchronously an action on gate g_j; if $m_j = n$, all processes B_i have to synchronize on gate g_j.

To provide a formal semantics, we introduce a predicate noted "$G \rhd J$", where $J \subseteq I$, that is true iff the processes in $\{B_i \mid i \in J\}$ can synchronize together on gate G. Obviously, for a given gate G, there may be several subsets J such that $G \rhd J$. This predicate is defined as follows:

- $\delta \rhd I$, meaning that all concurrent processes must synchronize on the termination gate δ, as in LOTOS;
- $(\forall j \in \{1, ..., p\})\ \ (\forall J \subseteq I \mid \mathit{card}(J) = m_j)\ \ g_j \rhd J$, meaning that each gate g_j achieves m_j among n synchronization;
- $(\forall G \in \widehat{G_1} \cup ... \cup \widehat{G_n})\ \ G \rhd \{i \in I \mid G \in \widehat{G_i}\}$, meaning that all processes having G in their interfaces must synchronize on G;
- $(\forall i \in I)\ \ (\forall G \notin \widehat{G_0} \cup \widehat{G_i} \cup \{\delta\})\ \ G \rhd \{i\}$, meaning that each process B_i can perform asynchronously any gate neither mentioned in $\widehat{G_0}$ nor in its interface $\widehat{G_i}$ (δ excepted and τ included).

Using this predicate, the operational semantics of our parallel operator can be defined with a single semantic rule:

$$\frac{(\exists L)\ \ (\exists J \subseteq I)\ \ (\mathit{gate}(L) \rhd J)\ \wedge\ ((\forall i \in J)\ \ B_i \xrightarrow{L} B_i')\ \wedge\ ((\forall i \in I \backslash J)\ \ B_i' = Bi)}{\textbf{par}\ g_j\#m_j ...\ \textbf{in}\ \widehat{G_i} \rightarrow B_i ...\ \textbf{endpar} \xrightarrow{L} \textbf{par}\ g_j\#m_j ...\ \textbf{in}\ \widehat{G_i} \rightarrow B_i' ...\ \textbf{endpar}}$$

Using this operator, the process network of Figure 1.2 can be specified using 2 among 3 synchronization:

par G**#2 in**
  B_1 || B_2 || B_3
endpar

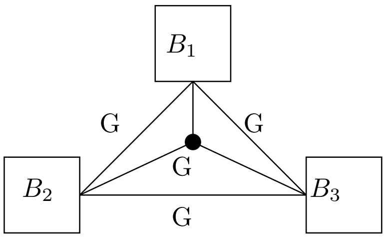

Figure 1.5

More complex process networks, such as the one on Figure 1.5, in which the same gate G has various degrees of synchronization, can also be described:

$$\begin{array}{l}\textbf{par } G\#2, G\#3 \textbf{ in}\\ \quad B_1 \;||\; B_2 \;||\; B_3\\ \textbf{endpar}\end{array}$$

5. APPLICATION

In this Section, we illustrate the application of our parallel operator to the description of the ODP trading function. ODP [ISO95b] is a standard framework for distributed applications. Within ISO, E-LOTOS has been developed in the same working group as ODP, with the intent of being the formal description technique for distributed applications. This explains that ODP-related problems have been a constant source of inspiration for E-LOTOS designers.

Our proposals for introducing "2 among n" synchronization in E-LOTOS [Gar95; SG96] was motivated by the highly dynamic nature of ODP systems: processes can be created and destroyed dynamically, and binary communications between processes can be established dynamically. Although it has been argued that such behaviours could only be described by means of mobile process calculi, such as the π-calculus [MPW92a; MPW92b], we believe that the most salient aspects of ODP systems can be captured in the framework of an usual process algebra, such as LOTOS, extended with our new parallel operator. A comparative study of both approaches can be found in [FNLL96].

The ODP trading function is a typical example of ODP systems: this function is defined informally in an ISO standard [ISO95a]. A formal description in E-LOTOS of the essential features of the trading function can be found as an appendix of the E-LOTOS definition document [Que98, Annex A.3]. In this paper, we focus on the architectural description of the trading function, so as to explain how our parallel operator can be used to describe dynamic communication patterns.

The ODP *trader* is a computer process that establishes a relationship between a pool of *objects* within an open and dynamically changing distributed system. For simplicity, we assume that there is an upper bound n on the number of objects in the system. Each object can act either as a *service provider* (or *server*), as a *client*, or even as both.

On the one hand, a server must inform the trader of the services it is ready to offer. Advertising a service offer is called *export*. The trader keeps in a database all the export requests sent by the servers. On the other hand, a client may ask the trader about available services.

Requesting knowledge about a particular service is called *import*. The trader matches the clients' service requests with its database of service offers and, if possible, selects an appropriate server. The identification of this server is sent back to the client, which can then contact directly the server without further interaction with the trader.

The interesting issue in this architecture is that a client can eventually communicate with a server the identity of which was unknown to him before asking the trader. In mobile process calculi, this situation can be described using a dynamic creation of mobile gate(s) and/or agent(s). However, alternative approaches are possible, which avoid the complexity of dynamic gate/agent creation. We can model the behaviour of the whole system by the following parallel composition. Let E (export), I (import), and W (work) be three gates used for server-trader, client-trader, and client-server communication respectively. Let "$O[E, I, W](i)$" and "$T[E, I]$" be two processes representing the i^{th} object and the trader, respectively. The whole architecture can be described by the following term:

```
par E, I in
  T[E, I] || par W#2 in
               O[E, I, W](1) || ... || O[E, I, W](n)
             endpar
endpar
```

It is worth noticing that the pool of n objects could be expressed in a more concise way using an extended parallel operator that iterates over a finite set of values (such an operator was proposed in [Gar95] and introduced in E-LOTOS).

The behaviour of the trader can be described with the following LOTOS process, where *request* and *reply* are two enumerated values indicating the direction of the messages exchanged on gates I and W (according to the syntax of LOTOS, "`[]`" denotes the choice operator, "`?` $x : s$" denotes the receipt of a value of type s to be stored in variable x, and "`!` v" denotes either the transmission of value v or the receipt of a value that has to be equal to v):

```
process T[E, I](d : DataBase) : noexit :=
  E ?j:Object ?s:Service;
    T[E, I](add_to_database(d, j, s))
  []
  I ?i:Object !request ?s:Service;
    I !i !reply !search_server_in_database(d, s)
      T[E, I](d)
endproc
```

Assuming that the i^{th} object is a client asking the trader for some service s provided by the j^{th} server, and then requesting this server directly, its behaviour can be described as follows (we assume that there is always a server available for the requested service):

```
process O[E, I, W](i : Object) : noexit :=
  I !i !request !s;
    I !i !reply ?j:Object;
      W !j !i !request !s...;
        W !j !i !reply !s...;
          ...
endproc
```

Similarly, assuming that the j^{th} object is a server advertising a given service s to the trader, then answering a client request, its behaviour can be described as follows:

```
process O[E, I, W](j : Object) : noexit :=
  E !j !s;
    W !j ?i:Object !request !s...;
      W !j !i !reply !s...;
        ...
endproc
```

6. CONCLUSION

Taking into account that the binary parallel composition operators found in usual process algebras (such as ACP, CCS, μCRL, LOTOS, and the early versions of CSP) are not fully appropriate for describing complex synchronization architectures, we suggest to extend the LOTOS parallel composition operator "$|[\widehat{G}]|$" in two directions:

- First, we propose to replace the binary operator with an n-ary operator that directly reflects the graphical structure of process networks. From the examples given, it is clear that the n-ary operator is simpler to use by novice users, easier to read (because the structure of process networks is preserved), strictly more expressive, and appropriate for an automatic translation from graphical networks to algebraic terms and vice-versa.

  Although similar proposals can be found elsewhere [Bol90; DS92; DS95], which is a sign of soundness, our approach is slightly different because our n-ary operator can describe process networks not tackled in [Bol90] (for instance, the one of Figure 1.4) and because we adopted structured operational semantics (as in [ISO88]), rather than traces (as in [DS92; DS95]).

- Second, we increased the expressiveness of this new operator by relaxing the maximal cooperation requirement of CSP and LOTOS, in order to support "*m* among *n*" synchronization. Taking the ODP trading function as an example, we show that the new operator is user-friendly, intuitive, and practically useful for the description of networks with mobility and dynamic reconfiguration capabilities.

Our research benefited from discussions with our colleagues in the framework of the E-LOTOS standardization Committee. The parallel operator presented in this paper is a refined version of a previous proposal, which we submitted to ISO [Gar95; SG96] and which has been integrated in the current version of E-LOTOS [Que97].

Notes

1. Open Distributed Processing
2. Here, we slightly deviate from the meaning of *process* in LOTOS

References

J. A. Bergstra and J. W. Klop. Process Algebra for Synchronous Communication. *Information and Computation*, 60:109–137, 1984.

T. Bolognesi. A Graphical Composition Theorem for Networks of Lotos Processes. In IEEE Computer Society, editor, *Proceedings of the 10th International Conference on Distributed Computing Systems, Washington, USA*, pages 88–95. IEEE, May 1990.

Ed Brinksma. *On the Design of Extended LOTOS, a Specification Language for Open Distributed Systems.* PhD thesis, University of Twente, November 1988.

J. Camillieri. An Operational Semantics for OCCAM. *International Journal of Parallel Programming*, 18(5):149–167, October 1989.

Rance Cleaveland, Eric Madelaine, and Steve Sims. A Front-End Generator for Verification Tools. In Uffe H. Engberg, Kim G. Larsen, and Arne Skou, editors, *Proceedings of TACAS'95 Tools and Algorithms for the Construction and Analysis of Systems (Aarhus, Denmark)*, May 1995. Also available as INRIA Research Report RR-2612.

D. Dams and J. F. Groote. Specification and Implementation of Components of a μCRL Toolbox. Technical Report Logic Group Preprint Series 152, Utrecht University, December 1995.

Robert de Simone. Higher-level synchronising devices in MEIJE–SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

J. W. Davies and S. A. Schneider. A Brief History of Timed CSP. Technical Monography PRG-96, Oxford University, 1992.

J. W. Davies and S. A. Schneider. A Brief History of Timed CS. *Theoretical Computer Science*, 138(2):243–271, February 1995.

Jean-Claude Fernandez, Hubert Garavel, Alain Kerbrat, Radu Mateescu, Laurent Mounier, and Mihaela Sighireanu. CADP (CÆSAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification (New Brunswick, New Jersey, USA)*, volume 1102 of *Lecture Notes in Computer Science*, pages 437–440. Springer Verlag, August 1996.

A. Février, E. Najm, G. Leduc, and L. Léonard. Compositional Specification of ODP Binding Objects. In *Proceedings of the 6th IFIP/ICCC Conference on Information Network and Data Communication, INDC'96, Trondheim, Norway*, June 1996.

Hubert Garavel. A Wish List for the Behaviour Part of E-LOTOS. Rapport SPECTRE 95-21, VERIMAG, Grenoble, December 1995. Input document [LG5] to the ISO/IEC JTC1/SC21/WG7 Meeting on Enhancements to LOTOS (1.21.20.2.3), Liège (Belgium), December, 18–21, 1995.

J. F. Groote and A. Ponse. Proof theory for μ-CRL. Technical Report CS-9138, CWI Amsterdam, 1991.

Hubert Garavel and Mihaela Sighireanu. Towards a Second Generation of Formal Description Techniques – Rationale for the Design of E-LOTOS. In Jan-Friso Groote, Bas Luttik, and Jos van Wamel, editors, *Proceedings of the 3rd International Workshop on Formal Methods for Industrial Critical Systems FMICS'98 (Amsterdam, The Netherlands)*, pages 187–230, Amsterdam, May 1998. CWI. Invited lecture.

J. F. Groote and F. W. Vaandrager. Structured Operational Semantics and Bisimulation as a Congruence. *Information and Computation*, 100(2):202–260, October 1992.

D. Harel. StateCharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, 1987.

C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1988.

ISO/IEC. ODP Trading Function. Draft International Standard 13235, ISO — Information Processing Systems, Genève, June 1995.

ISO/IEC. Open Distributed Processing — Reference Model. International Standard 10746, ISO — Information Processing Systems, Genève, 1995.

ITU-T. Specification and Description Language (SDL). ITU-T Recommendation Z.100, International Telecommunication Union, Genève, 1992.

Pim Kars. Representation of Process-Gate Nets in LOTOS and Verification of LOTOS Laws: the Boolean Algebra Approach. In Dieter Hogrefe and Stefan Leue, editors, *Proceedings of the 7th International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols FORTE'94 (Bern, Switzerland)*, October 1994.

P. Kars. *Process-Algebraic Transformations in Context*. PhD thesis, University of Twente, June 1997.

Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer Verlag, 1980.

Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes I. *Information and Computation*, 100(1):1–40, September 1992.

R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes II. *Information and Computation*, 100(1):41–77, September 1992.

David Park. Concurrency and Automata on Infinite Sequences. In Peter Deussen, editor, *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer Verlag, March 1981.

G. D. Plotkin. A structural approach to operational semantics. DAIMI FN-19 FN-19, Computer Science Department, Aarhus University, 1981.

Juan Quemada, editor. Committee Draft on Enhancements to LOTOS (E-LOTOS). ISO/IEC JTC1/SC21/WG7 Project 1.21.20.2.3, January 1997.

Juan Quemada, editor. Committee Draft on Enhancements to LOTOS (E-LOTOS). ISO/IEC FCD 15437, April 1998.

Mihaela Sighireanu and Hubert Garavel. E-LOTOS User Language. Rapport SPECTRE 96-06, VERIMAG, Grenoble, October 1996. In ISO/IEC JTC1/SC21 Third Working Draft on Enhancements to LOTOS (1.21.20.2.3). Output document of the edition meeting, Kansas City, Missouri, USA, May, 12–21, 1996.

C. Vissers, G. Scollo, and M. van Sinderen. Architecture and Specification Style in Formal Descriptions of Distributed Systems. In S. Aggarwal and K. Sabnani, editors, *Proceedings of the 8th International Workshop on Protocol Specification, Testing and Verification (Atlantic City, NJ, USA)*, pages 189–204. IFIP, North-Holland, 1988.