DEPARTMENT OF COMPUTER SCIENCE

Resilient Process Theory (RsPT)

P.N. Taylor

Technical Report No. 282

April 1997

Resilient Process Theory (RsPT)

P. N. Taylor.

Department of Computer Science, Faculty of Information Sciences,

University of Hertfordshire, College Lane, Hatfield, Herts. AL10 9AB. U.K.

Tel: 01707 284763, Fax: 01707 284303, Email: p.n.taylor@herts.ac.uk

April 8, 1997

Abstract

This short paper is intended to highlight the focus and direction of my research work to date. Its purpose is to illustrate the course of future studies that I intend to undertake during my last year of research, leading up to the submission of my Ph.D thesis in September 1997.

My research has concentrated upon the modelling of object-oriented communicating processes using process algebras. Behavioural reuse and the modification of process behaviour within an environment have been central to my research so far.

I have identified certain shortfall in the ability of current process algebras to model a system based on objects. Modelling inheritance between processes and maintaining the stability of communications between those processes has proved difficult given the facilities of existing process algebras. Process substitution and modification (via inheritance) can introduce deadlock into a previously stable system.

I propose a variant of process algebras that use an asynchronous communications model (known as Resilient Process Theory–RsPT). My theory permits the modelling of a stable object-oriented communicating system which can allow the behaviour of processes (viewed as objects) to be reused and extended. Processes in my proposed formal model are less likely to fail if synchronising communications do not occur. A resilient process can continue to execute provided that inputs to the process are still available; outputs do not effect a process's ability to execute as these are placed in a buffer of infinite size.

Introduction

My research work has concentrated upon the specification of communicating systems which have an underlying object-oriented model. I am specifically interested in the behavioural reuse and modification of processes in a communicating system. Particularly with the exchange of a process for a new version of itself (via some form of inheritance) whilst maintaining the stability of both the process and the system being modified.

So far, my research has shown that existing mainstream process algebras, such as Milner's CCS [9] and Hoare's CSP [4], do not contain the declarative power to enable them to model objects and inheritance. For example, a class definition in CSP is not abstract, it is visible and can be treated as any other process. The ability to access an abstract definition differs from that of a pure object-oriented view of a class where access is restricted to only instances of a class. Consequently, child processes do not contain a copy of their parent's behaviour. Instead, that behaviour is referenced via inter-process communication. To attempt to capture the pure object-oriented view of inheritance this communication between parents and their siblings must be restricted from the view of the environment. This restriction is required to stop any influences that the environment might have over such inter-process communications.

Modelling inheritance in existing process algebras has, so far, only been done by process reference rather than by an implied knowledge of the behaviour of the parent process, as passed down to the child. Therefore, the parent and child are treated as separate entities. In an example with multiple child processes each child will explicitly reference the behaviour of its parent, causing delays in the responsiveness to every other child as the parent is bound to each child until it terminates its current sequence of actions. Object-oriented design and implementation does not operate in such a way as each child contains a copy of the information of the parent. From my research, it is clear that existing process algebra's attempts to model inheritance do not offer a clear representation of either objects or inheritance.

During my research I have identified three different types of inheritance that I would like to model in a process algebra.

1. Strict Inheritance. The straight forward copying of behaviour and the extension of choice within an existing process definition. Process $P \stackrel{\text{def}}{=} a.b.c.P + d.e.\emptyset$ and process $Q \stackrel{\text{def}}{=} P + j.k.l.Q$. The rules governing a process algebra's attempts to model strict inheritance are given by Rudkin in his theory of objects and inheritance in LOTOS ([11, p.413]), a

formal language for specifying network protocols that is derived from both CCS and CSP [5].

Note that Q will need to modify the recursive process references at the end of each action sequence for it to offer those actions again. To solve the recursion problem Rudkin also introduces *self* as a generic variable that is instantiated with the name of the process's calling function [11, p.416]. Without *self* Q would perform the actions of P and then be trapped in the recursion defined in the original specification of P, being unable to offer any further actions of Q. The revised process definitions for P and Q are therefore: $P \stackrel{\text{def}}{=} a.b.c.self + d.e.\emptyset$ and process $Q \stackrel{\text{def}}{=} P + j.k.l.self$

2. Casual Inheritance. The ability to insert new behaviour into an existing sequence of actions defined as a branch of behaviour in a parent process. Process $P \stackrel{\text{def}}{=} a.b.c.self + d.e.\emptyset$ and process $Q \stackrel{\text{def}}{=} P[\emptyset/f.g.self]$, such that the full definition of Q is now $Q \stackrel{\text{def}}{=} a.b.c.self^* + d.e.f.g.self^*$. Note*: *self* is again used to solve the problem of recursion in behavioural inheritance. Action renaming is used in this example to show that $\emptyset$ is replaced by $f.g.self$. However, at this time no formal definition of casual inheritance exists. Renaming is used in this example as a brief guide to the requirements of casual inheritance without expressing exactly *how* it is to be achieved.

3. Transitive Inheritance. Process $P \stackrel{\text{def}}{=} a.b.c.self + d.e.\emptyset$ and process $Q \stackrel{\text{def}}{=} P + j.k.l.self$. Therefore, process $R \stackrel{\text{def}}{=} Q$ also inherits the behaviour of P by the definition of transitivity over process behavioural inheritance. The ROOA method [10] develops an object-oriented model for LOTOS specifications to adopt. However, due to the restriction in LOTOS that parent (superclass) processes must provide exit functionality and child processes must not there is no allowance for a LOTOS process in ROOA to straddle both a parent and child class definition, being the parent of one process and child of another (as Q is defined to be above). A LOTOS process cannot be defined to have both `exit` and `noexit` functionality. Therefore the ROOA method breaks down if transitive inheritance is attempted.

My research interests, w.r.t the reuse of process behaviour by capturing inheritance, have highlighted further problems aside from the unsatisfactory modelling of inheritance.

The stability of a system which substitutes parent for child processes is another facet of my research which has grown out of the original problems associated with process behavioural inheritance (as described above).

Behavioural reuse by substituting P for Q can lead to an unstable system as a result of the substitution. Process Q may fail to synchronise with some other process R that the previous parent process P did not originally synchronise with. In effect, Q extends the behaviour of P by requesting a synchronisation with R and at the same time also weakens the system. Let us assume that R is busy or has failed. Consequently, at some time later in the execution of the system Q must wait for R; a situation that did not occur in the original system. Process Q suddenly has the power to crash the entire system by influencing processes that are not party to its own failed communications.

According to the definition of conformance, which Rudkin states is the criteria processes must meet in order to be deemed as suitable replacements ([11, p.412]), process Q is a valid replacement for P. Conformance states that one process may be substituted for another, in a system where the first process was expected, if (and only if) the replacement process fails to offer certain actions that the original process also failed to offer. Formalisation of the conformance law is now presented [1, p.11]:

Definition 1 Conformance

Let Q and P be processes.

(Q conf P) iff

$\forall s \in Traces(P). \forall A \subseteq L(P)$	If Q fails to offer an action a (after sequence
if $\exists Q' \bullet \forall a \in A \bullet Q \xrightarrow{s} Q' \not\xrightarrow{a}$	s) then P must also fail to offer the same
then $\exists P' \bullet \forall a \in A \bullet P \xrightarrow{s} P' \not\xrightarrow{a}$	action a (after the same sequence s).

From the previous argument, Q has brought instability to the system even though it was a valid substitution for P. It is not possible to predict the effect that modifying or reusing a process may have on the system therefore a solution to maintaining system stability must be found.

My research is aimed at finding a solution to the problem of modelling reuse and inheritance whilst still maintaining the stability of the system within which the modified processes reside. I argue that it should be possible to replace processes with new versions of themselves with (possibly) more behaviour without altering the stability of the system if extensions to the original behaviour of the system should fail. Within certain criteria the original behaviour of the system should still prevail regardless of the fragility of any newly inserted behaviour.

1 Process Algebra with Synchronous Communications

The standard process algebra model of communication, as used by CCS and CSP, is synchronous communication carried out between processes brought together under parallel composition over common channels shared between those processes. The synchronisation of channels in a system which attempts to capture inheritance forces the failure of that system if the extended behaviour of a new substituting process fails. The failure being due to a failed synchronisation. Any process that is not ready to synchronise on a common channel in the parallel composition of the communicating processes will cause any other process in the composition to wait. When **all** of the processes are in the same state (i.e: ready to communicate on a common channel) then they will **all** communicate together; hence synchronisation.

In the system $(P \parallel Q \parallel R)$, if process R is busy then P and Q will both wait for R. As you can see, R can effectively hold up both P and Q. If R has failed then, consequently, P and Q will wait indefinitely.

The main problem with existing process algebras is that their reliance upon synchronous communications does not lend itself to modelling a system where modification and extension is possible. In an object-oriented system the possibility of change is high as new child processes are introduced to supplant their parents. Clearly, a model of communications that is less susceptible to failed synchronisations between processes is required.

2 Asynchronous Process Communications

To solve the problem of synchronous communications reducing the stability of object-oriented communicating systems I have researched the applicability of existing asynchronous process algebras. To date, two theories for asynchronous communications have been reviewed; both theories being derived from the language of CSP. These theories are entitled "The Theory of Asynchronous Processes" (APT) [8] and "Receptive Process Theory" (RPT) [7]. Incidentally, RPT evolved from the work by Dill on speed independent circuits [2]. A process in both APT and RPT is modelled with an unbounded (infinite) buffer on both its input and output channels.

Due to their common heritage both APT and RPT can be converted into CSP processes by simply removing their unbounded buffers, enabling the rich facilities of the CSP language to be applied to those processes.

A process algebra based on an asynchronous communications model offers the ability to

communicate between processes and the environment without requiring the target of those communications being (necessarily) available.

By using the concept of unbounded buffers on their input and output channels APT and RPT effectively *surround* each process with a buffer. Each process then synchronises with this buffer rather than synchronising explicitly with other processes. Being of infinite size the buffer of an asynchronous process is always ready to accept a communication (hence the name given to receptive processes as they are always capable of receiving input; they are always receptive).

APT and RPT processes can talk or listen on their I/O channels without requiring other processes in the parallel composition being ready to receive the communications. Therefore, both APT and RPT processes are not affected by a failed synchronisation between communicating processes.

A problem with the existence of unbounded buffers on both input and output channels is that an explicit synchronisation between processes is no longer possible. There is no direct contact between processes if buffering is used on both input and output channels; all communications have to go through the buffers. Therefore, neither APT nor RPT can provide an explicit synchronisation between processes (unless it is through the ordering of their buffer's action traces).

A trace of a process is an ordered sequence of visible communications between the process and its environment. The ordering of communications of a process is found in the trace of that process.

The reordering of these traces may be necessary as synchronisation may be required between two processes (via their buffers). Note that explicit synchronisation between processes does not take place in either APT or RPT as the buffers form a shield around each process. Reordering the trace elements of a process shifts inputs left and outputs right and states that two trace elements denoting the same channel must remain ordered (to enforce channel ordering rather that trace element ordering) [8, p.6]. For example, $\langle b.w, a.v\rangle \sqsubseteq \langle a.v, b.w\rangle$ denotes that message w on channel b, followed by message v on channel a is reordered by the new trace $\langle a.v, b.w\rangle$. The reader should be aware that the ordering of a trace has some part to play in the (implicit) synchronisation of APT and RPT processes.

3 Resilient Process Theory

The main thrust of my research will be aimed at providing a mathematical model of asynchronous communications that attempts to offer more overall flexibility than those currently in use. My proposed communications model only buffers the output channels of a process and is called "Resilient Process Theory" (RsPT).

I propose to modify the theories of APT and RPT in order to produce RsPT. One topic for further discussion is the nature of the buffers that I intend to introduce into RsPT. At present the use of multi-sets seems to allow more flexibility than traces (i.e: sequences) as the ordering and reordering of the trace elements will not be necessary. The cost of using a multi-set over a trace is that CSP trace theory, that both APT and RPT are based upon, will cease to be applicable. I must therefore decide which model to adopt and what consequences this design decision will have on RsPT's design. Much of the work on CSP failures sets (i.e: traces and refusal sets), which characterise an asynchronous process in APT, will be void if a multi-set is used.

Using a multi-set will avoid the problems of reordering traces, with its influence on synchronisation between processes and their buffers. However, what I might gain on one hand with multi-sets as buffers I may easily lose should the underlying CSP theory fall apart once traces are removed. Clearly the decision to use multi-sets will have many repercussions for the remaining mathematical model.

3.1 Buffered Process Communication

There are three ways of representing buffers in a process, dependent upon the location of the buffers themselves. The notation used to represent specific buffers is the open/close square bracket symbol ([]). Use of this symbol allows process actions to be included within the buffer if necessary, to show any contents prior to synchronisation (e.g: $(P[a] \parallel a.Q[b])$, where P and Q will synchronise on channel a and evolve to $(P[] \parallel Q[b]))$. Reference to a general buffer for a process uses the following notation (as seen in section 5 of this paper): Pb_P or Qb_Q. Operations on the contents of a buffer can then be performed on b_P, where $_P$ represents any process.

The three separate views of buffered processes can be defined as follows:

1. Buffered input and output channels on a process - as defined in both APT and RPT [7, p.1] and written formally as $[]P[]$. Using the parallel composition operator $\parallel$, an expression of the form $([]P[] \parallel [\, C \,] \parallel []Q[])$ states that P and Q will engage in undirected communications

on common channels found in both their alphabet and the set C. Expressed formally in the CSP text as follows [4, p.72]:

$$traces(P \parallel Q) = \{t \mid (t \upharpoonright \alpha P) \in traces(P) \wedge (t \upharpoonright \alpha Q) \in traces(Q) \wedge t \in (\alpha P \cup \alpha Q)^*\}$$

The buffers of each process synchronise with the process itself and consequently have the same alphabets. Therefore, $\alpha_i P \equiv \alpha P \equiv \alpha_o P$, where ${}_iP$ denotes the input buffer for process P and ${}_oP$ denotes the output buffer of P.

2. The second type of process buffering, as yet unused by any of the reviewed asynchronous process algebras, is buffered input channels and unbuffered output channels. Represented formally as $[]P$ and written using the parallel composition operator as $([]P \parallel []Q)$. Only the input channels of an input buffered process are receptive. Buffered input entails that a process can continuously listen to the environment but must explicitly synchronise with other processes on output communications.

   The main disadvantage of this type of buffering is that although a process may listen indefinitely it must still wait if another process is not ready to receive any of its output communications. Were we to use this particular type of buffering then the problem of an unavailable process further down the line holding up the current process would still prevail. Therefore, we do not consider this type of buffer configuration further.

3. Finally, the buffer configuration for RsPT is defined. RsPT uses unbuffered input channels and buffered output channels on each process. We can show this buffer organisation formally as $P[]$. Using parallel composition we write $(P[] \parallel Q[])$. Again, common channels define synchronise points between processes, brought together under parallel composition. A process with the form $P[]$ may continually send messages to its unbounded output buffer. If P requests synchronisation on input then the sending process must be available for the communication to take place. P must also be ready to receive the communication. Therefore, we can enforce synchronisation with other processes by demanding that they communicate directly with the process on input, rather than with its buffer (as is the case with APT and RPT).

The advantage of RsPT over both APT and RPT is that my communications model allows a process to continue to execute if the target of the communication fails to respond. It seems logical

to model communications between processes so that they must wait if there is no response from the sender and are not concerned if the receiver of a communication fails to respond.

4 Summary

Existing process algebras do not model faithfully either objects or inheritance. From my research it it clear that any attempt to modify an existing system by replacing processes with their inherited counterparts has the potential to render that system more unstable than it was originally. Consequently, the modified system is more likely to fail.

One key issue that my research has identified has been the ability of a new process to influence the stability of an entire system, simply by substituting one process for its inherited child.

Part of the problems associated with modelling object-oriented communicating systems with existing process algebras is their use of synchronous communications to effect inter-process communication. A partial alleviation of this communications problem can be addressed by using process algebras with an asynchronous communications model. However, because existing asynchronous process algebras, like APT [8, 6] and RPT [7], use unbounded buffers on both input and output channels it is not possible to guarantee explicit synchronisation between processes, should they be required. Asynchronous processes synchronise with their buffers in a buffered input/output model, not each other. Inter-process communication in this case is implicit rather than explicit.

My proposed solution to the problem of a totally buffered system of processes is to introduce a refinement of APT and RPT; known as RsPT for "Resilient Process Theory". An RsPT process buffers its output channels only which gives it the flexibility to send messages to a process that may be unavailable to communicate without having to wait itself. Also, messages requested from other processes (taken as input to the RsPT process) will be synchronised directly with the buffer of the sending process. Should the sender be unavailable then the RsPT process will wait. The organisation of buffered output channels lends itself to a more resilient model for process communications, hence the name of the proposed theory.

Finally, rather than adopt a traces model (i.e: ordered sequences) for the visible communications of a process I am researching the use of a multi-set to replace the trace as the internal structure for the buffers. The multi-set will yield more flexibility as the reordering of trace elements will no longer be necessary, as it is in APT [8, p.4]. The contents of the multi-set can then

be interrogated by using standard set operations.

5 Future Work

With the identification of a suitable communications model my task becomes clear. A firm mathematical foundation for the theory behind RsPT must be developed in order to provide a formal language that allows object-oriented communicating systems to be modelled faithfully. I hope to be able to derive much of what is required from the existing models used in APT, RPT and CSP. It would be useful to maintain much of the existing underlying theory as the CSP foundations from which the theory is derived is consistent, complete, robust and well known. It is not my intention to have to reinvent much of the what will be required in order to capture formally RsPT.

An operational and denotational semantics for RsPT must be drawn up to give me a firm mathematical foundation for the language. Further research is necessary in order to determine the depth to which the theoretical foundation of RsPT needs to be captured. Using computation semantics, Hennessey's name for operational semantics ([3, p.30]), reduction rules for RsPT, derived from the following first draft, will be required. Note that references to general buffers for processes P and Q are used (i.e: Pb_P and Q_Q, instead of $P[]$ and $Q[]$).

The following mathematical structure is introduced as a model for the structure of the buffers to be used in RsPT. Using the alphabet of a process (αP) as the domain of a relation, a partial function between a channel from αP and a sequence of values is defined.

$$\begin{aligned} \text{For example: } \alpha P &= \{a, b, c, d\} \\ b_P &= \{a \mapsto \langle v, w \rangle, b \mapsto \langle x, y \rangle, c \mapsto \langle z \rangle, d \mapsto \langle\rangle\} \end{aligned}$$

The invariant for the contents of b_P is given as:

$$\text{dom } b_p = \alpha P$$

Note that the initial sequence of values for a channel can be empty, as it is in the example for $b_P(d)$. In RsPT it is possible to explicitly synchronise on a channel without passing a value along that channel (see rule 5 below). The signature of b_P is given as:

$$b_P : \mathit{Channel} \nrightarrow \text{seq}\, \mathit{Value}, \text{ where } \mathit{Channel} = \alpha P$$

Value is informally defined to be any value (i.e: message) sent along a channel.

Standard set operations are defined upon the elements of b_P. Standard sequence operations are defined upon ran b_P.

Two macro operations on sequences are used to extract the correct sequence elements mapped to by a specific channel: *first* (i.e: oldest item in a sequence) and *last* (i.e: newest item in a sequence). FIFO queuing of sequences is assumed, therefore the oldest item is the far right item in the sequence and the newest item is far left. Function axioms for these two new operations on sequences are defined as follows:

$$\begin{array}{|l} \mathit{first} : \mathrm{seq}_1 X \rightarrow X \\ \hline \forall s : \mathrm{seq}_1 X \bullet \mathit{first}(s) = s(\#s) \end{array} \qquad \begin{array}{|l} \mathit{last} : \mathrm{seq}_1 X \rightarrow X \\ \hline \forall s : \mathrm{seq}_1 X \bullet \mathit{last}(s) = \mathit{head}(s) \end{array}$$

A labelled transition system for RsPT can now be defined. The following rules govern the communication of channel/value pairs from within a process to its output buffer and between processes.

1. Internal communication from a process's action sequence to its output buffer ($P \rightarrow Pb_P$) is defined as follows:

$$\overline{c!v.Pb_P \xrightarrow{c.v} Pb_P \oplus \{c \mapsto \langle v \rangle \frown b_P(c)\}}$$

2. Inter-process communication (IPC) between the output buffer of P and Q ($Pb_P \rightarrow Q$) is defined as follows:

$$\overline{Pb_P \parallel c?v.Qb_Q \xrightarrow{c.v} Pb_P \oplus \{c \mapsto (b_P(c) \setminus \mathit{first}(b_P(c)))\} \parallel Qb_Q}$$

   If $c \in \mathrm{dom}\, b_P \wedge \mathit{first}(b_P(c)) = v$

   - Incidentally, the expression $\{c \mapsto (b_P(c) \setminus \mathit{first}(b_P(c)))\}$ can be written in extension as $\{c \mapsto b_P(c) \setminus \{\#b_P(c) \mapsto b_P(\#b_P(c))\}\}$. For reasons of brevity the extended version of the sequence replacement expression is not used.

3. Communication between the output buffer of Q and the input buffer of P ($Qb_Q \rightarrow P$) is defined as:

$$\overline{c?v.Pb_P \parallel Qb_Q \xrightarrow{c.v} Pb_P \parallel Qb_Q \oplus \{c \mapsto (b_Q(c) \setminus \mathit{first}(b_Q(c)))\}}$$

   If $c \in \mathrm{dom}\, b_Q \wedge \mathit{first}(b_Q(c)) = v$

4. Communications between both of the output buffers of communicating processes P and Q ($Pb_P \leftrightarrow Qb_Q$) is defined as:

$$\frac{}{c?v.Pb_P \parallel d?w.Qb_Q \xrightarrow{c.v,d.w} Pb_P \oplus \{d \mapsto (b_P(d) \setminus first(b_P(d)))\} \parallel Qb_Q \oplus \{c \mapsto (b_Q(c) \setminus first(b_Q(c)))\}}$$

If $(d \in \text{dom}\, b_P \wedge first(b_P(d)) = w) \wedge (c \in \text{dom}\, b_Q \wedge first(b_Q(c)) = v)$

5. Unparameterised synchronisation between processes (i.e: no value passing) can also be defined in RsPT simply by passing the channel name and no value (represented by $\emptyset$) between communicating processes:

$$\frac{}{Pb_P \parallel c.Qb_Q \xrightarrow{c.\emptyset} Pb_P \parallel Qb_Q}$$

If $c \in \text{dom}\, b_P \wedge b_P(c) = \langle\rangle$

By convention, the symbol $\emptyset$ can be dropped in the previous expression, yielding $\xrightarrow{c}$ in future.

A simple (first draft) set of operational semantics of RsPT has now been presented. Clearly, more work is required to complete the mathematical theory which, it is hoped, will provide a suitably flexible communication model to help solve the inheritance issues that underlie this research.

The choice of amendments to the existing theories of APT and RPT, in view of the traces model, will need to be considered. One main difference between APT and RsPT is the idea of modelling a buffer as a multi-set, rather than a trace. The failures model of CSP, as used by APT, will need to be modified (if possible) to incorporate the multi-set view. As yet it is not clear whether a multi-set will decrease the stability of the existing traces model. So far a set-based model which maps channels to sequences of actions has been proposed. The ordering of the elements within these sequences can imply strict synchronisation.

As soon as the mathematical theory behind RsPT has been developed I expect to apply it to specific case studies taken directly from the work done with BAe on the EMBLEM project (Empirical Man-in-the-loop Battalion-level Effectiveness Model). Results from the application of RsPT to the complexities of example EMBLEM engagements will be evaluated to show the benefits of the theory of RsPT and will form part of the completed research thesis.

References

[1] E. Brinksma and G. Scollo. Formal notations of implementation and conformance in LOTOS. Memorandum inf-86-13, Department of Informatics, University of Twente, The Netherlands, 1986.

[2] L.D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed Independent Circuits.* ACM Distinguished Dissertation. MIT Press, Cambridge: MA, 1988.

[3] M.C.B. Hennessy. *The Semantics of Programming Languages.* Wiley, Chichester, 1990.

[4] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, London, 1985.

[5] Information Processing System Open Systems Interconnection. LOTOS–a formal description technique based on the temporal ordering of observational behaviour. Technical Report DIS 8807, International Standardization Organisation, 1987.

[6] H. Jifeng, M.B. Josephs, and C.A.R. Hoare. A theory of synchrony and asynchrony. In M. Broy and C.B. Jones, editors, *Programming Concepts and Methods*, IFIP, pages 459–478. Elsevier Science Publishers B.V: North Holland, 1990.

[7] M.B. Josephs. Receptive process theory. *Acta Informatica*, 29:17–31, 1992.

[8] M.B. Josephs, C.A.R. Hoare, and H. Jifeng. A theory of asynchronous processes. Technical Report PRG-TR-6-89, Programming Research Group, University of Oxford, Oxford University Computing Laboratory, 11 Keble Road, Oxford. OX1 3QD, 1989.

[9] R. Milner. *Communication and Concurrency.* Prentice-Hall, London, 1989.

[10] A.M.D. Moreira and R.G. Clark. ROOA: Rigorous Object-Oriented Analysis Method. Technical Report CSM-109, Department of Computing Science and Mathematics, University of Stirling, 1993.

[11] S. Rudkin. Inheritance in LOTOS. In K.R. Parker and G.A. Rose, editors, *Formal Description Techniques*, volume VI, pages 409–423. Elsevier Science Publishers B.V: North Holland, 1992.