Review of Estelle and LOTOS with Respect to Critical Computer Applications

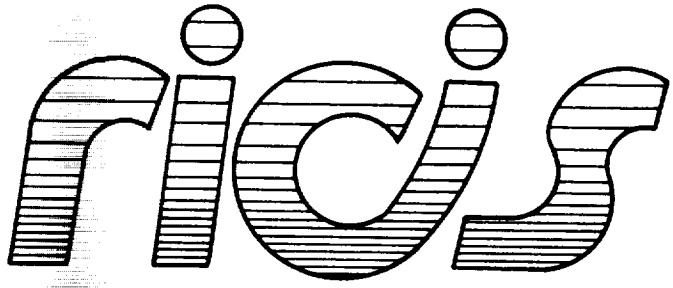
Rodney L. Bown

University of Houston-Clear Lake

May, 1991

Cooperative Agreement NCC 9-16
Research Activity No. SE.26

NASA Johnson Space Center
Engineering Directorate
Flight Data Systems Division



Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

Review of Estelle and LOTOS with Respect to Critical Computer Applications

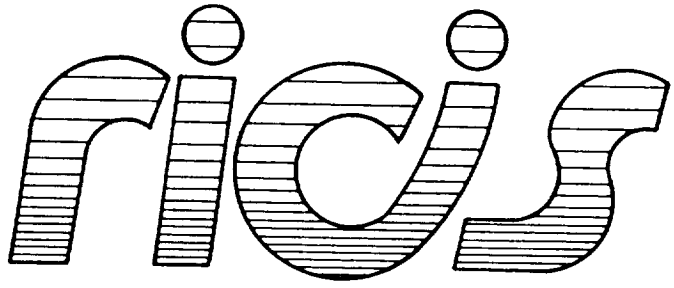
Rodney L. Bown

University of Houston-Clear Lake

May, 1991

Cooperative Agreement NCC 9-16
Research Activity No. SE.26

NASA Johnson Space Center
Engineering Directorate
Flight Data Systems Division



Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Dr. Rodney L. Bown, Associate Professor of Computer Systems Design at the University of Houston-Clear Lake. Dr. Bown also served as RICIS research coordinator.

Funding has been provided by the Engineering Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was William C. Young, of the Project Integration Office, Flight Data Systems Division, Engineering Directorate, NASA/JSC.

RICIS TECHNICAL REPORT

Review of Estelle and LOTOS with Respect to Critical Computer Applications

Principal Investigator

Rodney L. Bown

in partial fulfillment of

RICIS Task SE.26

May 1991

TABLE OF CONTENTS

Review of Estelle and LOTOS with Respect to Critical Computer Applications

1. Introduction

Man rated NASA space vehicles seem to represent a set of ultimate critical computer applications. These applications require a high degree of security, integrity, and safety. A variety of formal and/or precise modelling techniques are becoming available for the designer of critical systems. The design phase of the software engineering life cycle includes the modification of non-developmental components. This report provides a review of the Estelle and LOTOS formal description languages that were developed under the European Community ESPRIT program. The project was called SEDOS for "Software Environment for the Design of Open distributed Systems".

The project resulted in ISO standards for Estelle and LOTOS. Tutorial documents and example are starting to appear in the technical literature. The appendices to this report contain details of the languages and a set of references. The languages have been used to formally describe some of the Open System Interconnect (OSI) protocols.

2. Potential Space Applications

The Space Station Freedom and the space shuttle rely upon high integrity communications for their safe operations. Estelle and LOTOS are maturing to a level that will support the design or modification of communication systems. The set of reference material is now quite extensive. A first step would be several proof of principle projects that would provide training for the designer and demonstrate the potential of the languages.

3. Language Comparison

Table 1 compares Estelle and LOTOS [DIAZb89]. The most obvious difference between the languages is in their representation. Estelle is based on an extended Pascal syntax. LOTOS uses a more formal mathematical notation. This represents the concrete versus abstract approach of the two languages.

It is suggested that NASA use Estelle first on a well defined protocol due to its more concrete approach. This could be followed by reverse engineering (recapturing) the design in the more abstract LOTOS.

4. Tool Support and Activities

The seems to be more tool support for Estelle. This is primarily due to leveraging its extended Pascal syntax. The SEDOS project developed prototypes of a syntax-driven editor, a compiler, a simulator, and a verifier on Bull SPS7 or Sun-3 workstations. These tools are integrated into an Estelle workstation.

LOTOS is supported by an early collection of prototype tools called the LOTOS Implementation Workbench. This writer believes that LOTOS has more long term benefit due to its more abstract nature. Both languages are mature and supported by their respective ISO standards.

5. Structure of the Appendices

This report consists of this top level discussion which is supported by the appendices. The appendices provide review of the mathematical background, examples with syntax, and table of contents for the ISO standards.

6. Conclusion and Recommendations

A short review of the literature and discussions with colleagues indicates that there is more activity in the LOTOS community. Recent reports from the Microelectronic Computer Corporation (MCC) cite activity with LOTOS but do not mention Estelle [GERH91]. MCC is conducting a review of Formal Methods. One of the sponsors is NASA/JSC code FT41. The reports are available to the sponsors of the study. There seems to be a growing European community of protocol designers using Estelle and LOTOS.

It is recommended that a set of computer system components be hand modeled using both Estelle and LOTOS. These models would complement current interface control documents.

It is recommended that prototype Estelle and LOTOS workstations be established at the Research Computer Development Facility (RCDF) at UHCL RICIS. These workstations would support the implementation and demonstration of the hand modeled components using Estelle and LOTOS. The recommended activities would be conducted is cooperation with other formal model activities at UHCL.

Table 1.

Comparing Estelle and LOTOS.

	Estelle	LOTOS
Design	Hierarchies of communicating modules	Hierarchies of communicating processes
Semantics	Extended state machines	Calculus-of-communicating systems agents
Communications	Infinite FIFO queues	Multiple rendezvous events
Designer's view of a module	Internal: Waits for inputs and sends outputs	External: Describes a temporal ordering of events
Data	Based on Pascal	Based on abstract data types
General approach	More concrete	More abstract

References and Partial Bibliography

[BOLO90]
Bolognesi, Tommaso. "On the Soundness of Graphical Representations of Interconnected Processes in LOTOS." Proceedings of the ACM SIGSOFT International Workshop on Formal Methods in Software Development. Napa, California 9-11 May 1990. Published as Software Engineering Notes 15.4 Sept 1990. 1-7.

[CHUN90]
Chung, Anthony and Deepinder Sidhu. "Experience with an Estelle Development System." Proceedings of the ACM SIGSOFT International Workshop on Formal Methods in Software Development. Napa, California 9-11 May 1990. Published as Software Engineering Notes 15.4 Sept 1990. 8-17.

[DIAZa89]
Diaz, Michel, et.al. The Formal Description Technique Estelle, Results of the ESPRIT/SEDOS Project. Amsterdam: Elsevier Science Publishers. 1989.

[DIAZb89]
Diaz, Michel and Chris Vissers. "SEDOS: Designing Open Distributed Systems." IEEE Software November 1989. 24-33.

[EHRI85]
Ehrig, H. and B. Mahr. Fundamentals of Algebraic Specification 1. Berlin: Springer-Verlag. 1985.

[ESTEL89]
International Standard ISO 9074. Information processing systems - Open Systems Interconnection - Estelle: A formal description technique based on an extended state transition model. 1989.

[GERH91]
Gerhart, Susan. "Formal Methods at MCC: Transition Study, Spec Tra & Future." NASA/JSC Formal Methods Briefing. April 3, 1991.

[LOGR90]
Logrippo, Luigi, Tim Melanchuk, and Robert J. Du Wors. "The Algebraic Specification Language LOTOS: An Industrial Experience." Proceedings of the ACM SIGSOFT International Workshop on Formal Methods in Software Development. Napa, California 9-11 May 1990. Published as Software Engineering Notes 15.4 Sept 1990. 59-66.

[LOTOS89]
International Standard ISO 8807. Information processing systems - Open Systems Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour. 1989.

[MILN80]
Milner, Robin. A Calculus of Communicating Systems. Berlin: Springer-Verlag Lecture Notes in Computer Sciences 92. 1980.

[VANE89]
Van Eijk, Peter H. J., Chris A. Vissers, and Michel Diaz. The Formal Description Technique LOTOS, Results of the ESPRIT/SEDOS Project. Amsterdam: Elsevier Science Publishers B. V. 1989.

[VUON88]
Voung, Son T., Allen C. Lau, and R. Issac Chan. "Semiautomatic Implementation of Protocols Using an Estelle-C Compiler." IEEE Transactions on Software Engineering 14.3 14.3 384-393, Mar 1988.

APPENDIX A

A Review of Calculus of Communication Systems

Reference:

Milner, Robin. A Calculus of Communicating Systems. Berlin: Springer-Verlag 1980.

This appendix provides a review of Milner's book in support of a UHCL RICIS Technical Report on LOTOS. This report was submitted to NASA Johnson Space Center in April 1991. Milner presents a calculus of concurrent systems within a 171 page 1980 book that is part of Springer-Verlag's Lecture Notes in Computer Science. The presentation is partly informal, and aimed at practice; which is unfolded through the medium of examples. These examples illustrate the expressive power of the calculus and the techniques for verifying properties of a system.

This author introduces an algebraic method the supports the formal semantics of LOTOS (Language for Temporal Ordering Specifications). LOTOS is an International Standard (ISO 8807) developed as part of the ESPRIT SEDOS (Software Environment for the Design of Open Systems). A companion to LOTOS is Estelle. Estelle and LOTOS have been developed to describe services and protocols for distributed architectures.

The author's goal is provide an underlying theory whose basis is a small well-knit collection of ideas and which justifies the manipulations of the calculus. The calculus is founded on two central ideas. The first is observation. The aim is to describe a concurrent system fully enough to determine exactly what behavior will be seen or experienced by an external observer. Two systems are indistinguishable if one cannot tell them apart without pulling them apart. The author provides a formal definition of observation equivalence in Chapter 7 and investigates its properties.

Every interesting concurrent system is built from independent agents which communicate. The second central idea of the calculus is synchronized communication. Communication between two component agents is regarded as an indivisible action of the composite system, and the heart of the algebra of systems is concurrent composition, a binary operation which composes two independent agents, allowing them to communicate.

In 1980 related theories of concurrency include work by C. A. Petri; Net Theory by Genrich, Lautenbach, and Thiagarajan; MODULA by N. Wirth; Distributed Processing by P. Brinch Hansen; and Communicating Sequential Processing by C. A. R. Hoare.

In Chapter 1 Milner provides a discussion of Experimenting on non-deterministic machines. The term Experimenting upon

acceptors is introduced. An acceptor is a black box, whose behavior you want to investigate by asking it to accept symbols one at a time. These are called atomic experiments. The concepts of observable and unobservable (hidden) events is introduced. The foundation equivalence based on observable events is being developed at this point of the presentation material.

The author unfolds the state transition graph to represent Behavior as a tree. The following definitions are introduced at this point with the author's notation.

Definition: A label is a member of a given (fixed) set $\mathbf{A}$

Definition: A sort is a subset of $\mathbf{A}$

Definition: A Rigid Synchronization Tree (RST) of sort L is a rooted unordered, finitely branching tree of whose arcs is labelled by a member of L.

The symbol τ is used to represent unobservable actions.

Definition: A Synchronization Tree (ST) of sort L is a rooted, unordered, finitely branching tree each of whose arcs is labelled by a member of $\mathbf{L} \cup \{\tau\}$.

Chapter 2 presents a discussion of Synchronization. The term Mutual experimentation refers to the question of how should two machines interact? Binary semaphores are used as a simple example.

Chapter 3 presents a case study in synchronization, and proof techniques. The example is a scheduling problem. The author admits that these exercises are playing to some extent, but they may have some significance for building asynchronous hardware from components.

A section on Observation Equivalence provides a mathematical definition for equivalent agents. Observation equivalence can be described in general terms as follows. An s experiment means "p can produce p' under s" The meaning of equivalent agents can be stated in words as follows:
p and q are equivalent iff for every s
(i) For every result p' of an s-experiment on p, there is an equivalent result q' of an s-experiment on q.
(ii) For every result q' of an s-experiment on q, there is an equivalent result p' of a s-experiment on p.
The motivation for the definition is this: we imagine switching p on, performing an experiment, and switching it off again. For q to be equivalent, it must be possible to switch q on, do the

same experiment, and switch it off in a state in which p was switched off (and the same, interchanging p and q).

An interesting Exercise is cited related to Deadlock.

Prove the if p is equivalent q then the following is true of both or of neither, for a given set of experiments $\lambda_1, \ldots \lambda_n$, λ_{n+1}

"It is possible to do a $\lambda_1 .. \lambda_n$ experiment and reach a state where a λ_{n+1} experiment is impossible."

SPECIAL NOTE: This note is provided by Milner and elaborated by this reviewer. One property of agents is not respected by equivalence. It is possible for p and q to be equivalent even though p possesses an infinite series of silent computations such that p diverges while q does not. There is a note on page 99 in section 7.1 This is also discussed in section 7.3. A software engineer should observe that p could be the coded implementation of the q specification. The proof of p's equivalence to q does not prevent the divergence of p due to internal computations. This is a property of unobservable malicious code (Trojan Horse, viruses, etc) within the computer security domain.

Chapter 4 provides some Case studies in value-communication. The behaviors (Synchronization Trees) may be built using six kinds of operations, together with the all-important use of recursion.

The operations fall into two classes:

(1) Dynamic operations (Chapter 1)

Inaction	NIL
Summation	+
Action	$\mu \in \Lambda \cup \{\tau\}$

The dynamic operations build non-deterministic sequential behaviors.

(2) Static operations (Chapter 2)

Composition	|
Restriction	$\backslash\alpha$ $(\alpha \in \Delta)$
Relabelling	[S]

The static operations establish a fixed linkage structure among concurrently active behaviors.

The examples given were static combinations of sequential behaviors, yielding systems with fixed linkage structure. But dynamically -evolving structures can be gained by defining recursive behaviors composition.

The previous calculus is pure synchronization. The calculus is extended to pass values: accepting input pulses, and giving output pulses.

Chapter 5 provides the syntax and semantics of CS$. CCS and atomic actions are defined precisely. This chapter starts the development of the central notion of observation equivalence. Observation equivalence is developed in Chapter 7. From this a stronger notion of observational congruence is developed.

Chapter 6 provides a presentation on Communication Trees (CTs) as a model of CCS. This chapter is not essential to the technical development. Its purpose is to assist understanding by giving the natural generalization of STs to admit value passing.

Chapter 7 provides the development of Observation equivalence and its properties. Equivalence is not congruence. On Page 99 the following comment is made: Thus, whenever we have proved B is equivalent to C (e.g. B may be a program and C its specification) we cannot deduce the B has no infinite unseen action, even if C has none. In one sense we can argue for our definition, since infinite unseen action is - by our rules - unobservable! But the problem is deeper; it is related to so-called fairness, which we discuss briefly in section 11.3. In any case, there is a more refined notion of equivalence which respects the presence of infinite unseen action, with properties close to those we mention for the present one.

This discussion relates to the common software engineering problem of insuring consistency between specifications and implementations.

Equivalence is not a congruence. A congruence relation is stronger than equivalence. It is desired to know that if B and C are equivalent, then in whatever context we replace B by C the result of the replacement will be equivalent to the original - which is only true for an equivalence relation which is a congruence.

Milner then defines Observation congruence. Observation congruence is the weakest congruence stronger than (smaller than) equivalence. The author provides some theorems and a definition of stability.

Definition: B is stable iff B cannot reach B' by an infinite set of unobservable actions.

Thus a stable behavior is one which cannot 'move' unless you observe it. Stability is important in practice; one of the reasons why the author's scheduler example worked, is that it will always reach a stable state if it is deprived of external communication for long enough.

The author introduced a guard g which is observable. Then one can deduce from B is equivalent to C (for any B,C) that g.B is

equivalent to g.C and hence g.B is observation equivalent to g.C since both are stable.

The Laws of Observation Congruence are provided in the author's notation. Law (1) may be explained by saying that, under the guard g, internal action on B rejects no other capabilities and therefore has no effect. Laws (2) and (3) absorb the effects of internal actions.

Chapter 8 provides some proofs about familiar data structures as well as algorithms, which find natural expression in CCS. In addition the author illustrates how the properties of observation equivalence and congruence allow us to prove that systems work properly. The topics are Registers and memories, chaining operations, pushdowns, and queues.

Chapter 9 provides a translation of CCS for a rather simple language. The syntax of commands is:

assignment
sequential composition
conditional
iteration
declaration
parallel composition
input
output
no action.

The parallel composition is the major new construct. For example can the 'concurrent' assignments overlap in time? The author discusses the semaphore and Hoare's "Toward a theory of parallel programming." Hoare's idea is to allow the programmer to declare arbitrary abstract resources. For example, the programmer may associate a particular resource R with the output device, and adopt the discipline that every OUTPUT command occurs within a "WITH R ..." context. He can thus protect a sequence of OUTPUT commands from interference. There is a possibility of deadly embrace or deadlock, but a compile time check can prevent this. The program must be such that any nesting of "WITH R ..." commands with distinct R's must agree with the declaration nesting of the R's. For our translation: "WITH R DO C" must not contain "WITH R ..." for the same R.

Chapter 10 provides a precise a notion of Determinacy, and a related concept Confluence. Strong cfluency is defined as:

The behavior program A is strongly confluent iff

When A is transformed to B under b (1)
and
A is transformed to C under c (2)
implies either b = c and B is equivalent to C

or
when B is transformed to D under c
and C is transformed to E under b
then D is equivalent to E

The case "B equivalent to C' represents intuitively that (1) and (2) are essentially the "same action". The definition of determinacy demands this for observable actions.

A is strongly determinate iff
A is transformed to B by observable action b
and
A is transformed to C by observable action b
implies B is equivalent to C

Chapter 11 provides a set of conclusions. The author has shown how the calculus is based on a few and simple ideas and that it allows us to describe succinctly and to manipulate a wide variety of computing agents, that it offers rich and various proof techniques, that it offers some concurrent programming concepts, and that it allows the precise formulation of questions which remain to be answered.

The question of fairness is presented and defined as:
If an agent persistently offers an experiment, and if an observer persistently attempts it, then it will eventually succeed. A model which reflects this is sometimes called fair.
Is CCS fair? The author leaves the question open.

In summary the author's work has been concerned with the notion of expressing behavior. Behavior is regarded as a congruence by considering which expressions can be distinguished by observation. The author summarizes his approach as an attempt to calculate with behaviors without knowing what they are explicitly; the calculations are justified by operational meaning, and may help towards a better understanding - even an explicit formulation - of a domain of behaviors.

APPENDIX B

Estelle overview

An Estelle description is a hierarchy of communicating modules. A module is defined by a couple: a header and a body. The header defines a module's external visibility, including where information will be received and sent. The body defines a module's internal composition, which could include its behavior (defined by a transition set), its structure (defined by a set of parent and child modules), or both (defined by both transitions and child modules).

Behavior. The basic behavior of a module is defined as an extended sate machine that has machine states (called major states), variables, and transitions between major states. A major state and the values of its related variables define an extended state.

An extended state machine reacts to external events that correspond to information sent to and received by the module it represents. A module receives information through interactions processed at interaction points. Each interaction point has an infinite FIFO queue associated with it.

Transitions are of two types: receiving and spontaneous. A receiving transition is of the form:

```
from state_1 to state_2
priority non_negative_expression
when IP-name.interaction_name
provided predicate
begin     ... set of Pascal Estelle statements
          including when needed the emission of
          interactions by the statements
          output IP_name.interaction_name[parameters] ...
end
```

In this example, the transition is enabled when the machine is in state_1, the predicate is true, and interaction_name is at the head of the IP_name's FIFO queue. If this transition is in the set of those enabled and has the highest priority, then it is selected to be in the set of the ready-to-fire transitions. One of these transitions is non-deterministically chosen to be fired. When it is fired, the begin-end block is first atomically executed and the control state is passed to state_2. Interactions may be sent successively to different interaction points during the execution of the begin-end block.

A spontaneous transition is of the form:

```
from state_1 to state_2
priority non_negative_expression
delay (min_value, max_value)
provided predicate
begin ...same block as before ...
end
```

Spontaneous transitions include a delay clause. To be enabled, the machine must first be in the major state state_1 and the predicate must be true. When this happens, a virtual clock is started and the transition is enabled between the times min_value and max_value, if the predicate remains true and if the machine remains in the same state. Selection and firing are done similar to a receiving transition.

After describing a module, you must give it an initial extended state. You do this with the initialize clause, which lets you assign parameter values and the initial major state. You also declare a type and instantiate the module. The actual instantiation is written as:

```
init module_var with header_name [list of actual parameters].
```

Of course a module can be instantiated and released dynamically, by executing the appropriate instructions in a begin-end transition block.

After instantiation, a module potentially can fire a transition. The first transition will be the initialize transition. If more than one transition can be fired, a transition is selected non-deterministically.

Structure. A module definition can contain transitions and/or can be divided into other modules. These parent and children modules are interconnected with one of two primitives: connect and attach. How this is done depends on whether the modules belong to the same hierarchy level.

Case 1. In the first case, if two modules, child_module_var1 and child_module_var2, belong to the same hierarchy level, they are bound together by the father module with the statement

```
connect child_module_var1.child_IP1
     to child_module_var2.child_IP2.
```

which relates two interaction points, child_IP1 of the former module and child_IP2 of the latter. This means that an interaction sent to the child_module_var1's child_IP1 interaction point will be received in the FIFO queue associated with child_module_var2's interaction point, child_IP2.

For example, suppose an array of user modules (User[i]) each has an interaction point (U). You can interconnect these modules with a set of entities (E[i]), each of which has a high interaction point (H) and a low interaction point (L). These interconnection entities can themselves be interconnected with a service (Ser) that has an array of interaction points (N[i]), for all values of i. You would accomplish all this with the statement

```
connect User[i].U to E[i].H;
connect E[i].L    to Ser.N[i]
```

Case 2. In the second case, if two modules belong to adjacent hierarchy levels, they are interconnected by the father module with the statement

```
attach father_IP1 to child_module_var1.child_IP1.
```

This means that any interaction sent to the father's interaction point (father_IP1) will be forwarded to the child's interaction point (child_IP1). Once the modules are attached, the FIFO corresponding to father_IP1 will cease to exist and all interactions with it are received in the FIFO corresponding to child_IP1.

An example nontrivial Estelle transition (which includes the use of init and attach and uses some self-explanatory keywords: trans, forone, suchthat, and otherwise) is

```
trans
from init_state to state_after_CR
when User[k].connect_request
begin forone net:network suchthat net.NB_connection<=10
  do begin
       net.NB_connection:=NB_connection+1:
       attach User[k] to net.H[NB_connection];
       k:=k+1
  end
  otherwise begin
       init new_net with network.body;
       new_net.NB_connection:=1;
       attach User[k] to new_net.H[1];
       k:=k+1
  end
end
```

This transition fires when it receives a request for a new network connection. The transition block says that if one instantiated management module with less than the maximum multiplexed connection (10) exists, the requested connection will

be attached to one of these modules, selected non-deterministically. If no such modules exists, the requested connection is handled by a new network-connection-handling module that is instantiated and attached dynamically.

Of course, this simple example only illustrates Estelle's description style.

APPENDIX C

LOTOS Overview

Background

LOTOS is a formal description technique (FDT) that has been developed for the specification of distributed systems, and in particular to Open Systems Interconnection (OSI) standards. LOTOS and the companion FDT Estelle were developed under the European Community ESPRIT program. The project was called SEDOS for "Software Environment for the Design of Open distributed Systems".

In the SEDOS project the development of LOTOS was mainly focused on OSI Standards. In this line LOTOS has been used to make complete descriptions of :

HDLC
IEEE LAN service
ISO Connection-less Internetting Protocol
ISO Network Service
ISO Transport Protocol
ISO Transport Service
ISO Session Protocol
ISO Session Service
ISO Presentation Protocol and
ISO Transaction Processing Service.

LOTOS has also been applied to other field of technology, in particular to the development of Computer Integrated Manufacturing architectures.

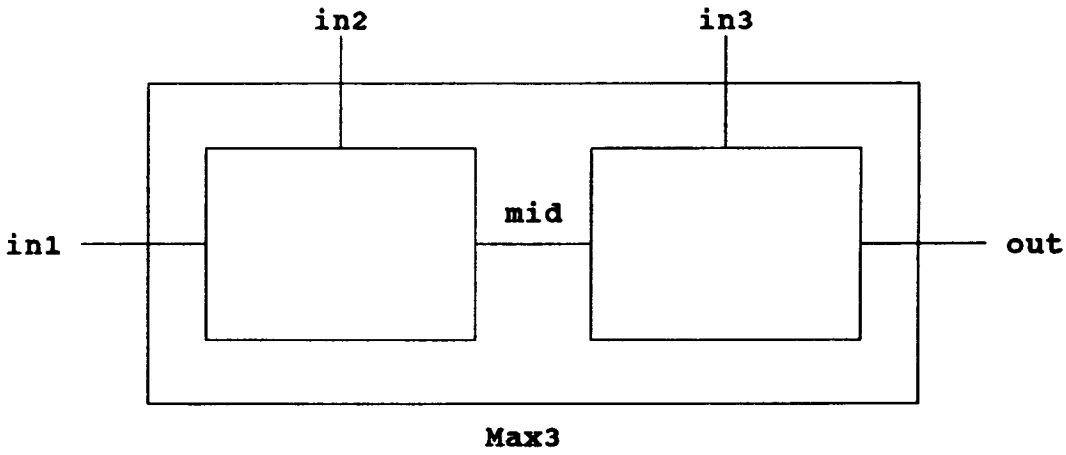
Introduction

The basic idea that LOTOS developed from was that systems can be specified by defining the temporal relation among the interactions that constitute the externally observable behavior of a system. The description technique is based on process algebraic methods. Such methods were first introduced by Milner's work on Calculus of Communicating Systems (CCS) [MILN80]. Milner's book is reviewed in Appendix A.

LOTOS also includes a second component, which deals with the description of data structures and value expressions. This part of LOTOS is based on the formal theory of abstract data types, and in particular the approach of equational specification of data types, with an initial algebra semantics. Most concepts in this component were inspired by the abstract data type technique ACT ONE although there are a number of differences [EHRI85].

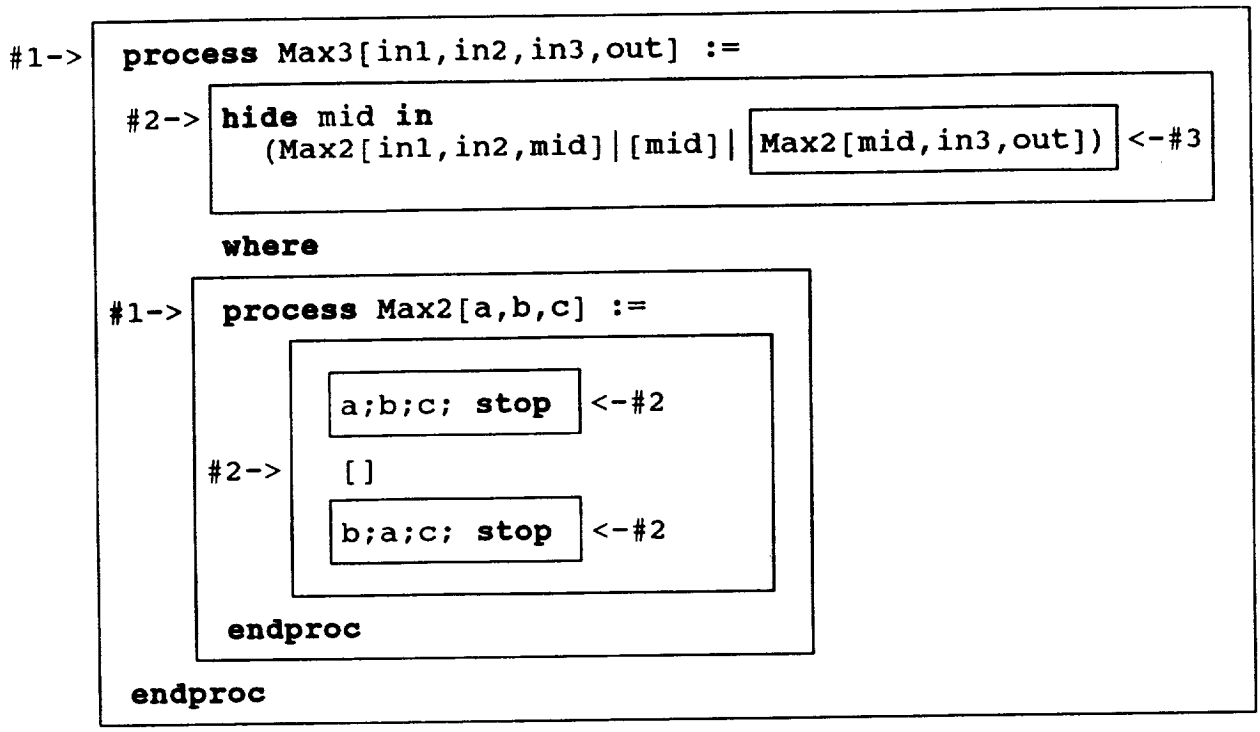
Example

In LOTOS a distributed concurrent system is seen as a process, possibly consisting of several sub-processes. A sub-process is a process itself, so that in general a LOTOS specification describes a system via a **hierarchy of process definitions.** A ***process*** is an entity able to perform ***internal, unobservable actions***, and to interact with other processes, which form its ***environment***. Complex interactions between processes are built up out of elementary units of synchronization which are called ***events***, or ***(atomic) interactions***, or simply ***actions***. The following representation is taken from [VANE89]



Spatial representation of process Max3

External observable actions are **in1**, **in2**, **in3**, and **out.** **mid** is an example of an internal unobservable action. The next page provides the LOTOS syntax for process Max3.



#1 <process definition>
#2 <behavior expression>
#3 <process instantiation>

Definition of process Max3

An Additional Example

The following example is taken from [DIAZ89] which uses a more textual presentation to represent interaction with the system's environment. Some of the previous discussion is repeated to keep the example in perspective. LOTOS considers both the system and the environment to be processes. An interaction, which is called an event in LOTOS, is defined as an activity common to two or more processes in which the values of types (called sorts in Act One) are established.

A LOTOS process is defined in terms of possible event-offer sequences. An event offer is represented by an indication of the place at which the event may occur, the event gate (G), and the value or values the process is willing to establish in the event (<Es> and <Ee>). Given this, the system and environment processes are described as follows:

```
process system [G]:=
        ...
        G<Es>:
        ...
        ...
endproc
```

```
process environment [G]:=
        ...
        G<Es>:
        ...
        ...
endproc
```

After an event, all the processes involved in it refer to the same value or values that were established. Because LOTOS is for the design and specification of open distributed systems, it must express these data values at a high abstraction level so they are implementation-independent. Hence LOTOS uses abstract data types.

Abstract Data Types

In LOTOS, abstract data types include the syntax that describe its values or its signature (which includes sorts and operations definitions) and a list of equations that describe its semantics:

```
type Extended_Natural_Numbers is
        sorts    nat
        opns     0:->nat
                 succ:nat->nat
                 _+_:nat,nat->nat
        eqns
                 forall x,y:nat
                 ofsort nat
                 X+0=X;
                 x+succ(y)=succ(x+y);
         endtype (*Extended_Natural_Numbers*)
```

Abstract data types are a powerful specification tool themselves, but they lack facilities to define interaction properly. Where interaction is not involved, LOTOS offers a choice in the specification style: you can either represent some property with a data structure, storing control information in the process's parameters, or represent this information directly in the process definition.

Events and Value Passing

In LOTOS, events are atomic instances of interaction: All processes involved in an event have the same view of it. This means either that an event has taken place for all processes defined to be involved (which also means that they all refer to the same data value or values established), or the event has not taken place at all. Events must be implemented reliably.

LOTOS abandons the traditional I/O concepts. Instead, it uses a much more powerful interaction concept. In an interaction, the processes involved can negotiate -- based on constraints -- about which data value or values to establish.

This not only provides a uniform way to express events, but, more importantly, it gives it provides an extremely powerful specification capability. By imposing different constraints, three kinds of interactions can be defined: value checking, value passing, and value generation.

In value checking, the example processes (system and environment) are synchronized. An example of value checking, using the notation from the examples above and the usual shorthand for natural numbers (0=0, succ(0)=1, succ(1)=2, and so on) is

```
process system[G]:=
        ...
        G!3
        ...
endproc
```

```
process environment[G]:=
        ...
        G!3
        ...
endproc
```

System and environment synchronize on the value 3 (of the type extended natural numbers).

In value passing, once the event has occurred the value is passed from one process to another, as in

```
process system[G]:=
        ...
        G!3
        ...
endproc
```

```
process environment[G]:=
        ...
        G?y:nat;
        ...
endproc
```

The value 3 has been passed. This models conventional I/O.

In value generation, the event has occurred and the x and y variables have the same value, which is chosen non-deterministically from the data domain in which the system and environment constraints overlap, as in

```
process system[G]:=
        ...
        G?x:nat[even(x)];
        ...
endproc
```

```
process environment[G]:=
        ...
        G?y:nat[y5];
        ...
endproc
```

Where x and y will have the same value (non-deterministically chosen between one of the values 0,2, and 4).

Architecture Support

After a description is provided for the event offers, they can be sequenced with temporal ordering operators to compose a process description. In LOTOS, an event ordering is defined with a behavior expression. How these behavior expressions are formed reveals LOTOS's algebraic nature: the many behavior expressions are combined into a new one with language operators that correspond to important architectural concepts. For example, if B1 and B2 are behavior expressions then B1[]B2 defines the choice between B1 and B2. Table A lists other important operators.

Rules are another important algebraic element in LOTOS. Rules let the designer to transform one behavior expression into another that expresses the same observable behavior according to an equivalence relationship. For example, B[]B can be replaced by B for any behavior expression B.

In LOTOS, process abstraction is comparable to facilities in imperative languages for defining functions and procedures. For example, if an adder process abstraction is defined with formal event gates input1, input2, and output, this expression

```
process   Adder[input1, input2, output]:noexit:=
      (input 1 ?x:nat|||input2?y:nat)
      ;output!(x+y)
      ;Adder[input1,input2,output]
endproc
```

defines the generic behavior of an adder that accepts two values of sort nat in any order and outputs their sum.

Finally, a LOTOS specification begins with

```
specification System[G1, G2,...](P1:sortP1,P2:sortP2,...):noexit
  type(*global data types definition*)endtype
  behavior(*definition of behavior expression*)
  endproc
```

Summary

Table A provides the definition of some of the LOTOS operators. As in structured programming, LOTOS's abstraction facilities and algebraic nature let you produce well-structured specifications. Its operators provide a modular structure, and its hierarchy of process abstractions lets you distribute complexity over several abstraction layers.

References

[DIAZ89]
Diaz, Michel and Chris Vissers. "SEDOS: Designing Open Distributed Systems." IEEE Software November 1989. 24-33.

[EHRI85]
Ehrig, H. and Mahr, B. Fundamentals of Algebraic Specification 1. Berlin: Springer-Verlag. 1985.

[LOTOS89]
International Standard ISO 8807 (LOTOS)

[MILN80]
R. Milner, "CCS, a Calculus of Communicating Systems," Lecture Notes in Computer Sciences 92, Springer-Verlag, Berlin, 1980.

[VANE89]
Van Eijk, Peter H. J.; Vissers, Chris A.; and Diaz, Michel. The Formal Description Technique LOTOS, Results of the ESPRIT/SEDOS Project. Amsterdam: Elsevier Science Publishers B. V. 1989.

Table A
Some LOTOS operators.

Name	Form	Meaning
inaction	**stop**	deadlock
action	a;B	event a precedes the process B
choice	B1[]B2	choice between process B1 and B2
parallel	B1|||B2	interleaving full synchronization
	B1||B2	
	B1|[A]|B2	synchronization on events in A(*)
sequence	B1>>B2	process B1 precedes process B2
disrupt	B1[>B2	process B2 disrupts process B1
hiding	**hide** A in B	hide the events in A(*) from the environment of B

(*) A is a set of gate identifiers.

APPENDIX D

International Standard ISO 9074 (Estelle)

Title Sheet

Table of Contents

INTERNATIONAL STANDARD

ISO 9074

First edition
1989-07-15

Information processing systems — Open Systems Interconnection — Estelle: A formal description technique based on an extended state transition model

Systèmes de traitement de l'information — Interconnexion de systèmes ouverts — Estelle: Technique de description formelle basée sur un modèle de transition d'état étendu

Reference number
ISO 9074 : 1989 (E)

25

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council. They are approved in accordance with ISO procedures requiring at least 75 % approval by the member bodies voting.

International Standard ISO 9074 was prepared by Technical Committee ISO/TC 97, *Information processing systems.*

Users should note that all International Standards undergo revision from time to time and that any reference made herein to any other International Standard implies its latest edition, unless otherwise stated.

International Organization for Standardization
Case postale 56 • CH-1211 Genève 20 • Switzerland
Printed in Switzerland

CONTENTS

APPENDIX E

International Standard ISO 8807 (LOTOS)

Title Sheet

Table of Contents

INTERNATIONAL STANDARD

ISO 880

First edi
1989-0

Information processing systems — Open Systems Interconnection — LOTOS — A formal description technique based on the temporal ordering of observational behaviour

Systèmes de traitement de l'information — Interconnexion de systèmes ouverts — LOTOS — Technique de description formelle basée sur l'organisation temporelle du comportement observationnel

Reference num
ISO 8807 : 1989

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council. They are approved in accordance with ISO procedures requiring at least 75 % approval by the member bodies voting.

International Standard ISO 8807 was prepared by Technical Committee ISO/TC 97, *Information processing systems.*

Users should note that all International Standards undergo revision from time to time and that any reference made herein to any other International Standard implies its latest edition, unless otherwise stated.

Annex A forms an integral part of this International Standard. Annexes B, C, D and E are for information only.

Printed in Switzerland

CONTENTS

ANNEXES

TABLES

FIGURES