Research Institute of Ontario. We would like to thank Mike Irwin and Eugene Zywicki of Gandalf for their support of this project. In addition, we are indebted to Mohammed Faci, Souheil Gallouzi and Bernard Stepien of the University of Ottawa for useful discussions and comments on the paper.

References

[BB] Bolognesi, B., and Brinksma, E. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems 14 (1987)* 25-59. Also reprinted in [VVD] 23-73.

[BC] Bolognesi, T., and Caneve, M. Equivalence Verification: Theory, Algorithms, and a Tool. In: [VVD] 303-326.

[BO] Boehm, P., deMeer, J., and Schoo, P. PERLON Persistency Checker for Data Type Definitions. In [VVD] 285-302.

[Br] Brinksma, E. A Theory for the Derivation of Tests. In [VVD], 235-247.

[Bro] Brown, R.L, Denning, P.J., and Tichy, W.F. Advanced Operating Systems. Computer, Vol. 17 (1984), No. 10, 173-190.

[B1] Brinksma, E. On the Design of Extended LOTOS. PhD Thesis, Twente University (NL), 1988.

[DH] DeNicola, R., and Hennessy, M.C.B. Testing Equivalences for Processes. *Theoretical Computer Science 34, (1984)*, 83-133.

[EM] Ehrig, B., Mahr, B. *Fundamentals of Algebraic Specifications.* Springer-Verlag, 1985.

[FLS] Faci, M., Logrippo, L., and Stepien, B. Formal Specification of Telephone Systems in LOTOS. To appear in: Brinksma, E., Scollo, G., and Vissers, C. (eds.) *Protocol Specification, Testing, and Verification IX*, North-Holland.

[GHL] Guillemot, R., Haj-Hussein, M., and Logrippo, L. Executing Large LOTOS Specifications. In: Aggarwal, S., and Sabnani, K. (eds.) *Protocol Specification, Testing, and Verification VII*, North-Holland, 1988, 399-410.

[GL] Gueraichi, L., and Logrippo, L. Derivation of Test Cases for LAP-B from a LOTOS Specification. To appear in the Proc. of the 2nd FORmal TEchniques Symposium (Vancouver, December 1989).

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

[ISO] International Organization for Standardization. Information Processing Systems. Open Systems Interconnection. Basic Reference Model for Open Systems Interconnection, 1984.

[ISO1] International Organization for Standardization. Information Processing Systems. Open Systems Interconnection. LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behavior (ISO International Standard 8807), 1988.

[LOBF] Logrippo, L., Obaid, A., Briand, J.P., and Fehri, M.C. An Interpreter for LOTOS, a Specification Language for Distributed Systems. *Software-Practice and Experience, 18 (1988)* 365-385.

[Led] Leduc, G.J. The Intertwining of Data Types and Processes in LOTOS. In: H.Rudin and C.H.West (eds.) *Protocol Specification, Testing, and Verification, VII.* North-Holland, 1987, 123-136.

[Mil] Milner, R. *A Calculus of Communicating Systems.* Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, 1980.

[MM] Manas, J.A., and de Miguel-More, T. From LOTOS to C. In: K.J.Turner (ed.) *Formal Description Techniques* North-Holland, 1989, 79-84.

[Par] Park, D. Concurrency and Automata on Infinite Sequences, Proc. 5th GI Conference, *Lecture Notes in Computer Science N. 104*, 167-183, 1981.

[QP] Quemada, J, Pavon, S., and Fernandez, A. Transforming LOTOS Specifications with LOLA. The Parameterised Expansion. In: K.J.Turner (ed.) *Formal Description Techniques* North-Holland, 1989, 45-54.

[Tur] Turner, K.J. An Architectural Semantics for LOTOS. In: H.Rudin and C.H.West (eds.) *Protocol Specification, Testing, and Verification, VII.* North-Holland, 1987, 15-28.

[Vis] Vissers, C.A., Scollo, G., Alderden, R.B., Schot, J., and Ferreira-Pires, L. The Architecture of Interaction Systems. Lecture Notes, Twente University of Technology (NL), 1989.

[VSV] Vissers, C., Scollo, G., and Van Sinderen, M. Architecture and Specification Style in Formal Descriptions of Distributed Systems. In Aggarwal, S., and Sabnani, K., Protocol Specification, Testing, and Verification, VIII, North-Holland, 1988, 189-204.

[VVD] van Eijk, P., Vissers, C.A., and Diaz, M. *The Formal Description Technique LOTOS.* Nort-Holland, 1989.

These processes were subsequently decomposed demonstrating that LOTOS specifications can be developed top down when the tools and application domain are known. Thus, LOTOS more directly (and, we think, more intuitively) describes protocol behaviors than FSM based methodologies, which do not allow this type of decomposition.

3.3. Learning and Using LOTOS

The most difficult aspect to learning LOTOS was making the conceptual shift away from languages commonly used in the Gandalf environment such as C or PLM. Another problem was the current scarcity of introductory materials on the language. These problems can be expected to slow initially the acceptance of LOTOS in industrial environments. However, a four-week course was sufficient to put the two industrial participants in a position to start using the language and the interpreter. After an additional three months of independent work, they were able to produce complete and useful LOTOS specifications.

In our experience, the difficulty of learning LOTOS can be roughly compared to the difficulty of learning other "unconventional" functional or logic-based languages, such as LISP or Prolog. Of course, learning was facilitated by the fact that, unlike other specification languages, LOTOS is (at least partially) executable.

4. CONCLUSIONS

This project explored the applicability of LOTOS for describing the architecture of a data switch "in the large" (macro system design), and also to specify fine grain interaction "in the small" (micro system design). The Starmaster exhibited many of the characteristics typically found in most modern digital systems, including most distributed systems, e.g. multiple processors and multiple internal communications pathways.

The investigation showed LOTOS to be a powerful and expressive language for specifying protocols and distributed systems. LOTOS specifications are precise and unambiguous due to the formal semantics of the language, although of course establishing the correspondence between LOTOS events and "real-life" events can be a problem. As well, LOTOS promotes precise reasoning about a specification, and can help identify inconsistencies and incompleteness in the requirements. Finally, LOTOS allows specifications to be modular and well structured, and favors top-down development and separation of concerns. However, no language, as well-structured as it might be, can be a panacea. Experience with LOTOS demonstrates that stylistic concerns are as important as for traditional programming languages. It is as possible in LOTOS as in any other language to write obstreperous and perplexing specifications or programs.

Beyond the scope of this project, our results indicate that this language, which to date has been confined largely to the specification of OSI protocols and services, could play a significant role in the design of systems which involve complex hardware/software interactions. For example, it might well find new uses in the specification of other architectural problems, such as systems involving multiple busses inside a computer system or even inside a single VLSI chip.

There remain some areas of concern. Although abstract specifications are beneficial in hiding implementation concerns, the latter must be dealt with at some point during product development. The methodology for using LOTOS specifications in the implementation phase is still a subject of research. Therefore, a risk remains in going from a specification to an implementation. LOTOS specifications have the advantage of being more abstract with respect to FSM and EFSM ones. Thus it is probably easier to introduce translation errors when implementing from a LOTOS specification.

An important factor towards a wider acceptance of LOTOS appears to be the creation of Abstract Data Type libraries much larger than the one provided with the standard version of the language. In addition, ADT "shorthands" for such common concepts as record structures, arrays, etc., appear to be highly desirable. Finally, the current LOTOS standard does not allow the creation of libraries of LOTOS modules. Again, this is a shortcoming that will have to be overcome.

The result of this project within Gandalf has been a decision that a follow-up project be selected from currently planned new product development, that has a requirement for specifying control interactions. LOTOS will be used to specify the interactions, and to develop an implementation from the specification. Software design methodologies required for this purpose will be evaluated and determined. As well, a designer with expertise in the product's requirements will be introduced to LOTOS, to further evaluate the ease of acquiring skills in the language. Now that some experience has been gained in LOTOS, the potential benefits of it identified by this project can begin to be quantified. Furthermore, during this second more extensive project, it should be possible to explore some other of the possibilities presented by the language (such as formal derivation of test cases from specifications, formal derivation of implementations from specifications, formal verification, graphic LOTOS, etc.).

Acknowledgment. Work reported here was funded in part by the University Research Incentive Fund of the Province of Ontario, and by the Telecommunications

but at the same time one or more processes actually may be signalling its displeasure to the others. Often the failure represents an "address out-of-range or inactive" condition, i.e. a poll-and-failed-response combination. This means that the addressed entity currently does not exist, although it may have existed in the past or it might exist in the future. In order for this scheme to work, there must be an active process which will act as a "ghost" process. This ghost process takes part in every related rendezvous in order to say that the address is non-existent. Of course, the ghost behavior must be de-activated when the address is considered existent, which may be accomplished in a rich number of ways in LOTOS.

A code fragment illustrating a poll operation to a live process follows:

```
   (bus ?x:adr ?sp:service_primitive ?status:bool
            [(x eq my_adr) implies status];
   . . .
    [x eq my_adr] -> ...
   )
|[bus]|
   (bus !child_adr !poll ?status:bool;
    [status] -> ...
   . . .
   )
```

In the above example, synchronization occurs on gate *bus* causing an exchange of three data values: an entity address value, a service primitive value, and a boolean status value. The first contributor simply insists that if the address *x* is equal to the value of the variable *my_adr* then the status value must be *true*, otherwise (*x* not equal to my_adr) the status value is unconstrained (because this rendezvous wasn't really for it). The second member of the rendezvous (there might be more) insists upon the address being the value of the variable *child_adr*, and the service primitive to be a *poll*, while leaving the status unconstrained. After the rendezvous, the first contributor examines the address to see if the synchronization was really for it and the second one examines the value of variable *status* as set in the rendezvous to see if the poll succeeded or not.

3.2.2. Call Management Specification

The call management specification used one process in partial synchronization with a medium to encapsulate all call behaviors. This process corresponds to the constraints imposed by the lowest level modules as described above. Call management has both master and slave components. Although the behavior describes interactions between the NCL and ISIMs, the presence of the ISIM was the significant constraint and all behavior is encapsulated within that process.

The top level process was composed of interleaved master and slave processes that specified NCL and ISIM call behavior respectively. A medium was used to coordinate interactions between these components. In this way, we represent the fact that these components interact indirectly, and therefore transmitted messages are not immediately received. Call collision handling can then be specified and executed.

Both the master and slave processes follow similar structures. This is a choice of behaving as a call initiator, call receiver, or being configured. The only difference in their structure is that the latter use guards to insure that their allowable behavior is consistent with their configuration, i.e. that they can originate or accept calls or both.

LOTOS operators were found to be well suited to describing call management interactions. For example, a call scenario can intuitively be thought of as:

1) A connection setup phase, possibly interrupted and aborted for some reason.
2) A connection phase, eventually disabled by a disconnect indication.
3) A disconnection phase that verifies that new connections may be initiated.

This scenario translated almost directly into top level LOTOS processes. For example, the [simplified] LOTOS for an incoming call originated by an ISIM (slave) is shown below.

```
process SlaveCallIn[ma,sl]: noexit:=
(
    ma!C_Request;
    (
        SlaveConnecting[ma,sl]
        >>  (
                SlaveConnection[ma,sl]
                [>   SlaveNormalDisc[ma,sl]
            )
    )
)
```

A connect request action is offered followed sequentially by process SlaveConnecting. When this process completes, it enables (>> operator) the SlaveConnection process which in turn is disabled ([>) by process SlaveNormalDisc.

3.2. The Starmaster Protocol.

Gandalf's Starmaster is a distributed data PBX. Intelligent Subscriber Interface Modules (ISIMs) for both local and wide area network interfaces and direct subscriber interfaces reside in Subscriber Logic Shelves (SLSs). They connect in a star topology to the Node Logic Shelf (NLS) which contains the Node Control Logic (NCL). The NCL and ISIMs have a master/slave relationship. Intelligent modules, referred to as Local Assemblies (LAs), link the NLS and SLSs. Together, these modules define a hierarchical architecture from the NCL at the top to ISIMs at the bottom. A master/slave relationship exists between each adjacent pair of modules.

One document currently specifies both the system architecture and the protocols used by the different modules to communicate. It is complex and reflects the evolution of the product over a number of years. Also, it is influenced by specific implementation choices. Elements of the system architecture, the call management protocols used between NCL and ISIMs, and the link protocols employed by LAs, were chosen to be re-specified in LOTOS. The ability of LOTOS to precisely document these elements was to be considered as a significant indication of the suitability of LOTOS for use within Gandalf. As well, these elements provided a well defined problem within the scope of typical Gandalf applications and allowed the project to center on the evaluation of LOTOS rather than on defining the protocol requirements.

Two specifications were completed as part of this project: a top level system architecture specification and a call management specification. Time did not permit completion of the link protocol but we do not feel that this affected our ability to perform the evaluation. Although initially both specifications had been intended to be parts of the same specification, they stand well on their own. Each specification is briefly explained in the following subsections.

3.2.1. System Architecture

As mentioned above, the Starmaster was a hierarchical machine with all its major components connected in a strict tree structure: each entity could communicate with only one parent (except the root) but with multiple children (except the leaves). The entities on each layer were of the same type distinctive to that layer, and most entities were physical cards, but some were (or could be) software processes or address-identifiers of software processes. The ability to map these entities on to the LOTOS specification proved to be the major large-scale system design problem.

Fortunately, the properties of a hierarchical machine can be mapped into those of a "protocol stack" similar to the one defined by the ISO Reference Model [ISO]. The critical feature is that direct communication occurs only between adjacent layers (parent-to-child and child-to-parent), and thus adjacent layers must be used to reach indirectly the more distant layers. This allows the adoption of many protocol-related constructs which are relatively well-developed in LOTOS, such as service primitives, service access points, upper-lower layer synchronization, etc. [Tur][Vis]. It should be noted that a protocol stack is even more constrained than the similar "layers of abstraction" notion which is popular in operating systems design because the latter permits downward communication to be initiated across more than one level [Bro]. The main lessons learned from this exercise were the degree to which LOTOS forced the meticulous resolution of all ambiguities in the specification and how simulated execution on the interpreter led to a much greater appreciation of the total interdependencies between components. In a protocol stack, seemingly independent actions can ripple ultimately outwards to all other layers, as well as be limited by converging constraints from other layers.

An interesting LOTOS issue reflected the inherent ability in LOTOS to trade off computational responsibility between the *data type* and the *control* part of the language, already mentioned above. It was found that a global memory process could be used, which kept track of aggregate data structures, such as the list of children belonging to a given parent, or multiple LOTOS processes could be used which would monitor each other's actions and thus behave like a distributed memory system. The LOTOS synchronization primitives are sufficient to support either approach (Bernard Stepien, another member of the LOTOS group at the University of Ottawa, independendently worked towards a similar conclusion while considering the problem of formally specifying a telephone switch).

In order to support a logical negation in the specification of distributed behavior, it was found possible to simulate a "failed" rendez-vous, such as a poll action with no successful response - even though LOTOS synchronization semantics would at first appear to strictly prohibit such a thing! In LOTOS, every process which possesses the synchronization tag (the "gate") must fully agree to the rendezvous, including the data which must be contributed to all parties to the rendezvous. Any failure to agree causes the rendezvous to fail. However, one of those agreed data values may be interpreted by convention to be a status value which represents overall "success" or "failure" of the synchronization. Thus, synchronization happens because all processes agreed to exchange data -

```
01  specification Max3[in1,in2,in3,out] : noexit

02  type integer is
03    sorts int
04    opns
05      zero   :     -> int
06      succ   :   int -> int
07      largest  :  int, int -> int
08    eqns forall X,Y: int ofsort int
09      largest ( zero , X ) = X;
10      largest ( X , zero ) = X;
11      largest ( succ(X) , succ(Y) ) = succ ( largest(X,Y) );
12  endtype

13  behavior
14    hide mid in
15     (
16       Max2 [in1,in2,mid]
17      |[mid]|
18       Max2 [mid,in3,out]
19     )

20   where
21     process Max2 [val1,val2,max] : noexit :=
22        ( val1?X:int; exit(X, any int)
23          |||
24          val2?Y:int; exit(any int, Y)
25         )
26            >> accept V: int, W: int in
27                 max!largest(V, W); stop
28     endproc
29  endspec
```

The specification is to be read as follows:

Lines 2 to 12 define the type *integer* with its associated operation *largest*. This is done according to the semantics of [EM]. Of course, the standard LOTOS library contains all these definitions, so normally the user will include them by invoking the library.

Lines 14 to 19 describe the top structure of the specification, which consists of two instantiations of process *Max2*. The latter is capable of finding the largest of two numbers, read in any order from gates *val1* and *val2*, and outputting it on gate *max*. As the two copies of *Max2* are instantiated, their gates are relabelled respectively *in1*, *in2*, *mid*, and *mid*, *in3*, *out*, resulting in the fact that the output value computed by one copy is fed to the other over gate *mid*. Note that *mid* is *hidden*, because it is meant for internal communication between the two instances of *Max2* only.

Lines 21 to 28 describe process *Max2*. It allows interleaving between the input actions on gates *val1* and *val2*. Both values input are then forwarded to the action on line 27, which calculates the largest of them and inputs it.

Lines 14 to 19 of this specification are an example of what has been called *resource-oriented* specification style. Lines 21 to 28 are an example of *constraint-oriented* style. Lines 22 to 27 could also be written, in *monolithic* style, as follows:

```
    val1?X:int; val2?Y; max!largest(X,Y); stop
  []
    val2?Y:int; val1?X; max!largest(X,Y); stop
```

and the equivalence between the two specifications could be proved easily by using the simplest rules of bisimulation.

3. THE GANDALF PROJECT

3.1. Background

Gandalf is a supplier of communications equipment. Many of the products developed by Gandalf have architectures with one or more embedded microprocessors. Standardized and proprietary protocols to coordinate and facilitate communication, both internal and linking to other vendor's equipment, are an integral part of Gandalf's products. These protocols are used for a broad spectrum of applications including link data transfer, routing, call control, network management, system coordination. The communication media ranges from shared memory and processor busses to various links and networks.

Informal or semi-formal methods, such as various mixtures of English text, finite state machines, message diagrams and program code fragments have previously been used to specify the protocols Gandalf has developed. These methods have resulted in errors, omissions and ambiguities being discovered during implementation and have increased the cost of integrating new products and features into existing product lines. The errors appeared to be due at least in part to the imprecise semantics of the specification techniques. For example, a finite state machine may be sufficiently precise for many purposes, but it can be unclear what it means to compose two machines in parallel. The lack of abstraction features has also been a problem, having resulted in specifications difficult to understand and maintain. The goal of this project was to evaluate the suitability of LOTOS and the University of Ottawa Toolkit within Gandalf, with respect to their abilities to overcome these past problems and provide long-term strategies for the future.

LOTOS specifications of OSI protocols produced within ISO use a constraint-oriented style because it emphasizes modularity and abstractness and it is well suited to describing open, implementation independent, protocols. Other styles may be more appropriate for different purposes. For example, a resource oriented style is useful for incorporating known implementation issues into a specification. Effective decomposition of the specification though, is more than a function of the chosen style. Knowledge of the applications requirements and the skills of the specifiers are equally important.

2.4. Executability of LOTOS Specifications and LOTOS Tools

Because of the fact that LOTOS is (partially) executable, a specification is effectively a "fast prototype" of the entity specified, thus it is possible to exercise a specification of a complex system at the design stage. This means that design errors can be found much earlier in the software development cycle than with other techniques.

The two LOTOS interpreters in existence today are described in [LOBF][GHL][VVD]. The interpreter discussed in the first two references is the one that has been used in this project. Other tools existing in various prototypical form are: symbolic expanders [QP], elementary theorem provers [BC], Abstract Data Type persistency checkers [BO], partial translators into C [MM]. An aim of this international tool development effort is to progress towards a full LOTOS-based CASE. Unlike most current CASE methodologies, a LOTOS-based CASE would be firmly grounded on the unifying concepts of the formal semantics of the language.

2.5. Formal Verification in LOTOS

It is possible to carry out in LOTOS proofs such as the ones found in [Mil][Hoa], and the proof methods are similar to those found in these references. The best developed proof techniques involve the concept of "bisimulation" [Par][B1]. Proof methods based on the concepts of "traces" and "refusal sets" [Hoa] are also being considered. Unfortunately however, because of the presence of internal actions, some of the proof methods developed for CSP, such as fixpoint induction methods, do not seem to be immediately applicable to LOTOS.

An important open problem is to find a unified verification framework for both the *control* and the *data* part.

Of course, the challenging aspect is to be able to prove properties of systems of realistic size. To this end, computer-assisted verification tools are being envisioned.

2.6. A Theory of Implementation and Testing

A rich formal theory of implementation and testing is being developed around LOTOS [BB][Br]. This means that the relation "I is an implementation of S" is formally defined for two expressions I and S. This formalization is given by the *reduction* relation, where I *reduces* S if: i) I can only execute actions that S can execute and: ii) I can only refuse actions that can be refused by S. Intuitively, I can be more deterministic than S, and can contain fewer options. In other words, in LOTOS the abstraction of a specification with respect to the implementation is represented by a higher level of nondeterminism.

Similarly, the relation *A and B are testing equivalent* [DH] has been formally defined as: *A reduces B and B reduces A*. Roughly speaking, two specifications are testing equivalent if their externally observable behaviors are identical. This corresponds to the *failure equivalence* of Hoare [Hoa]. By using these concepts, it is possible to derive implementations and test cases in a formal way from a LOTOS specification.

It must be observed, however, that so far these concepts have been fully developed for restricted forms of the language only. A practically usable approach for the derivation of test cases from LOTOS specifications, which however unfortunately still lacks a formal basis, is described in [GL].

2.7. LOTOS in Practice

Specifications of real-life systems of thousands of lines have been written in LOTOS. Some of these are on their way towards becoming part of ISO International Standards. Some examples are: several OSI layers (Network, Transport, Session), specifications of telephone systems [FLS], etc. (in addition of course to all best known "textbook" examples such as the Alternating Bit Protocol, the Dining Philosopher’s problem, etc.). Several such examples are included in [VVD]. The language is starting to be used in industrial environments, especially in the UK where British Telecom and Hewlett-Packard have substantial LOTOS groups.

2.8. A Small LOTOS Example

The following example, adapted from [BB], is a LOTOS specification for an entity which is able to accept three natural numbers in any order and stops after printing the largest of them.

languages in use today. Its static semantics are defined by an attributed grammar, while its dynamic semantics are based on algebraic concepts. LOTOS is made up of two components: a *data type* component, which is based on the algebraic specification language ACT ONE [EM], and a *control* component, which is based on a mixture of Milner's CCS [Mil] and Hoare's CSP [Hoa]. Most of the theoretical framework of the control component, and especially the concept of *internal action* are based on Milner's work. In particular, non-determinism is modelled by internal actions as in [Mil] rather than by adding special operators as in [Hoa]. The rendez-vous semantics follow Hoare's "multi-way rendez-vous" concept, by which all processes that share a gate must participate in a rendez-vous on that gate. Actions, however, can be transformed into internal actions by *hiding* them. In this way, further participation in the action of processes outside the *hide* is prevented.

LOTOS dynamic semantics for the *control* component is expressed in operational terms by inference rules as in [Mil], and the operators were chosen in such a way that it has been possible to prove about them a rich set of algebraic properties, similar to those of [Mil]. Therefore, the language is at the same time "executable" (by virtue of the operational semantics), and amenable to proof techniques (by virtue of the algebraic properties).

The language is purely recursive in nature, without side effects (except those produced by process synchronization). It supports process parameterization, for both value and gate parameters. Following is a very brief and informal account of the language, hopefully sufficient to enable the reader to follow our discussion.

The basic element of the *control* part of a LOTOS specification is the *action offer*, where a process declares itself ready to synchronize with other processes and establish one or more values. For example, *g!3* states that the process is offering to synchronize on the specific value *3* with other processes, on gate *g*. *g?x:integer* states instead that the process is ready to synchronize on any integer value with other processes, on gate *g*. Thus, two processes containing these two complementary action offers may be able to synchronize, and if they do, the second process gets the value *3* for the variable *x*. "Multiple" and "bidirectional" action offers are also possible, such as *g !3 ?x:integer*, where the process declares itself ready to simultaneously offer a value, and receive one, still on gate *g*. *Selection predicates* can establish conditions for the executability of an action. For example, *g!x:integer [x gt 3]* means that the process is ready to accept only an *x* greater than *3*. Similarly, an action can be subject to a guard, e.g. *[y gt 4] -> g!y* means that *y* is offered only if greater than *4*.

Action offers can be combined by the use of several operators. Some of the most important are: [] (choice), |[A]| (parallel execution with synchronization via gates in set A), || (parallel execution with synchronization on *all* gates), ||| (parallel execution in interleave), **hide** (hiding of gates), >> (sequential composition of processes), and [> (*disable*, modelling a nondeterministic interruption).

The *data* part supports parameterized types, type renaming, and conditional rules.

A "graphic" version of the language is undergoing standardization within ISO and CCITT.

2.3. LOTOS Specification Styles.

Because of the fact that LOTOS is made up of what its designers viewed as the most valid parts of CCS and CSP, the language has considerable expressive power. It favors a highly structured specification style and top-down, as well as bottom-up, design.

In principle, a LOTOS specification has the goal of specifying in a precise way the externally observable behavior of the entity being specified. Like all languages, however, LOTOS can be used in several different ways. Reference [VSV] identifies four main different LOTOS styles, called monolithic, state-oriented, resource-oriented, and constraint-oriented. In *monolithic* style specifications, the main operator is the choice operator [], the parallel composition operators are not used, and the specification is written as a tree of alternatives. Milner's [Mil] Expansion Theorem asserts that there exists an algorithm for transforming an arbitrary LOTOS specification into a monolithic style specification, although the algorithm may not terminate in some cases. In *state-oriented* specifications, explicit state variables are used. In *resource-oriented* specifications, the specification modules are chosen in such a way as to identify implementation modules. In *constraint-oriented* specifications, processes identify families of constraints and the parallel composition of processes specifies the simultaneous satisfaction of all constraints. This turns out to be a powerful way to impose "separation of concerns". The concept of constraint-oriented specification was already present in [Hoa]. The example of Section 2.8 presents some of these styles. It will also show how the different styles can be mixed together in a specification.

Other LOTOS styles can be identified, of course. For example, both the *data type* and *control* part of LOTOS are each sufficiently powerful to describe complex systems [Led]. It is a decision of the specifier whether to use more of the one or of the other. The way this tradeoff is solved can greatly influence the ease of writing a specification, as well as its readability.

The Algebraic Specification Language LOTOS:

An Industrial Experience

Luigi Logrippo
University of Ottawa
Computer Science Department
Protocols Research Group
Ottawa, Ont. Canada K1N 9B4

Tim Melanchuk
Advanced Development Group
Gandalf Data Ltd.
130 Colonnade Rd. S.
Nepean, Ont. Canada K2E 7J5

Robert J. Du Wors
Connected Systems Group
61 Reaney Court
Kanata, Ont. Canada K2K 1W7

Abstract

The ISO specification language LOTOS is presented, together with the results of a project involving the evaluation of its usefulness in an industrial environment. LOTOS is a mixture of concepts taken from CCS and CSP, along with an algebraic abstract data type formalism. The language was used by Gandalf to develop a formal specification of an existing protocol, part of a distributed data PBX. The effort concentrated on the specification of two aspects of the protocol: the top level system architecture, and the call management phase. It is shown how the unique features of LOTOS were found to be useful for expressing these aspects. The results of the project were positive, and further use of LOTOS is planned within Gandalf.

1. INTRODUCTION AND MOTIVATION

The subject of specification languages for data communications protocols and services (often called Formal Description Techniques or FDTs) has been the focus of much recent research. The initial motivation for the FDTs was provided by the area of protocols and service standards. Because these are meant to be implemented in compatible ways across the world, and on many different implementation architectures, it was essential that they could be specified in a precise, implementation-independent language. The specification must capture those features of an implementation that are necessary for it to be able to communicate with other implementations. However, formal and exact specifications of protocols and services are useful in every phase of the protocol development life-cycle, even outside of the realm of standards. At the design stage, a formal specification of a protocol or service enables a number of checks to be carried out regarding the soundness of the design. Logical errors can be found well before they become buried in an implementation. At the implementation stage, a precise specification of the software product serves as a reference to the implementor and as a medium of communication between implementors. At the testing stage, test scenarios can be obtained from the formal specification.

In this paper, we report on the FDT LOTOS, a language developed within the International Organization for Standardization (ISO), and on an industrial experience of its use as a specification tool for an existing system, the Gandalf Starmaster Control Protocol. LOTOS being a relatively new language, few experiences exist on its industrial application, and (as far as we know) none of them is in North America.

Much additional information on LOTOS and its applications can be found in [VVD][ISO1], and in the annual series of proceedings *Protocol Specification, Testing and Verification*, and *Formal Description Techniques (FORTE)*, both published by North Holland.

2. THE FORMAL DESCRIPTION TECHNIQUE LOTOS

2.1. Background

The International Organization for Standardization (ISO) has been developing over the years a family of standardized data communications protocols and services, called OSI (Open Systems Interconnection). At the very beginning of this effort it was recognized that, in order for OSI to be a real standard, it was necessary to provide it with an appropriate standard FDT, in which its protocols and services could be specified. An international committee (of which Luigi Logrippo is a member) set out to produce such a FDT, and, some years later, the language LOTOS has now become an International Standard [ISO1]. Interestingly enough, the language is turning out to be very appropriate not only for OSI protocols and services, but also for a wide family of distributed systems.

2.2. LOTOS Principles

LOTOS, the Language of Temporal Ordering Specifications, is one of the most precisely defined