Specification and Validation of Telephone Systems in LOTOS[1,2]

Rezki Boumezbeur and Luigi Logrippo

Protocols and Software Engineering Research Group
University of Ottawa,
Computer Science Department
Ottawa, Ontario, Canada K1N 9B4
E-mail: rezki@csi.uottawa.ca

[1] The research for this paper was carried out under the Design of Validation Environments for Protocol Engineering project as part of TRIO's Protocol and Software Engineering thrust.

[2] This paper was presented at the TRIO Retreat at Trent University (Peterborough, Ontario, Canada), May 8–10th, 1991.

Abstract

LOTOS (Language of Temporal Ordering Specification) is a Formal Description Technique that was developed by the ISO for the specification of OSI services and protocols. In this paper we present a design methodology for the description in LOTOS of telephone systems with modern telephone features. The description of a *Sample Telephone System* is formalized. The LOTOS specification was extensively debugged and tested using the facilities provided by the University of Ottawa interpreter.

1. Introduction and Motivation

LOTOS [ISO89][BB87] is a specification language developed by ISO over the last few years for the specification of OSI services and protocols. It has a formal basis and it has been shown to be appropriate for the specification of other types of distributed systems as well.

During the last few years, our research group has been working on different applications of the LOTOS language such as the formal specification of some OSI services and protocols, distributed algorithms, and basic telephone systems. The latter study was the subject of an early work leading to the specification of a simple telephone system (POTS: Plain Old Telephone System).

In this paper, we show that the LOTOS language is appropriate for formally specifying not only POTS systems, but also Telephone Systems where more sophisticated telephone features are provided to users. This includes features such as: Forward an incoming call, Ring Again, Transfer the current call/Three-Way Calling, Hold a call, and Conference call features. Our LOTOS specification of a *Sample Telephone System* allows for an arbitrary number of subscribers to use the Telephone Service simultaneously. It also allows the environment to update the system at any time during its functioning[3]. A particularity of Telephone Systems is that they are governed by the use of signals depending on the status of the requested lines. This has led to the design of a specification based on the use of constraints including the constraints imposed by the real world functionality of a specific entity. All these aspects are discussed in this paper at least in some detail.

We conclude our work with a discussion of debugging the specification using an interpreter. This will include a presentation of some interesting test sequences taken from the real world functioning of Telephone Systems.

2. Specification of Telephone Systems in LOTOS

Over the last few years, our research group has produced different types of specifications of telephone systems using the FDT language LOTOS [FLS90][Bou91]. In addition, the domain of formal description of telephone systems is receiving increasing interest from other researchers. Some related work may be found in [Zav85][Tvr89].

The idea of applying the FDTs to telephone systems comes from the fact that several concepts originally developed for the description of data communication applications are found to be very appropriate for the description of telephone systems. There are, however, some important differences between telephone system specifications and OSI service specifications; for example, the "data communication" phase for telephone systems cannot be specified precisely in LOTOS, unless *voice* is packetized. Also timing concepts are neglected in LOTOS at the current stage, whereas they are

[3] Updating the system refers to adding new lines to the system or removing some old lines from that system.

very important in telephone systems. However, some research has been done in this latter subject and some proposals exist [QF87][Bri88].

The most abstract view of a system is that of a black box. In this view, one is not trying to specify the internal structure of the system but, instead, all the focus is on the input/output of the interface functions between the system and its environment. For more details on this concept refer to [VSA+89][Bou91]. In the domain of telecommunications, we can distinguish three different types of structures of a system: the centralized structure, the decentralized structure, and the distributed centralized structure. Each of these structures is applied to a system depending on the range it is intended to cover [Tan89][Bou91].

3. Specification Design Principles

The first aim of a system design process is to derive an implementation that fulfils the user requirements. But due to the complexity of some distributed systems, the design process may become a more complicated task. In order to achieve a better - clear and readable - representation of a system, the design process may be carried out in steps, where each step represents a level of abstractness of the system. In LOTOS, the complete specification of the behavior of a (complex) system may be achieved in a step-wise fashion. First, the system is described in terms of a set of processes that represent distinct objects, then each resulting process is decomposed into sub-processes. The process of refinement of a system is repeated until we end up with simpler descriptions where no further decomposition is possible. Each stage of refinement represents a level of abstraction in the formal description of a system. This method of step wise refinement helps designers to get more precise descriptions of the behavior of each separate component, and therefore it gives a well structured and readable behavior of the overall system.

In the design of our *Sample Telephone Specification* we used a mixture of the different specification styles discussed in [VSvS88]. The use of a specification style in the design of a system must respect some general principles to achieve a qualitative design. These are mainly Orthogonality, Generality, and Open-endedness. Orthogonality preserves locality aspects of (sub)systems, Generality suggests the use of general-purpose parameterized definitions, and Open-endedness supports flexibility of the design to ease the modification of the system functionality.

The (formal) description of a system may have as its prime concern the description of observable behavior, like a black box, or the description of the internal behavior of that system, like a white box. The former description is called "*extensional*", while the latter is called "*intensional*". For each type of description some specification styles have been defined to guide the design process. In the extensional description of a system only observable interactions are shown and their temporal ordering relationship is defined in terms of alternatives of sequences or parallel composition of constraints. With the intensional description, however, both observable and internal interactions of a system are presented.

4. Specification of the Sample Telephone System in LOTOS

In our *Sample Telephone Specification* and in addition to the standard *NaturalNumber* and *Boolean* types, some other data items are defined. First, as we are dealing with telephone systems, we need to define *telephone numbers* and some operations on them

We need also to define the sets *BusySet* (for busy numbers), *ServiceSet* (for numbers in service),

and *ForwardSet* (for numbers where a *Forward* feature is active) with the operations (*empty*, *insert*, *remove*, *isin*, *notin*, *is_empty*, *head*, and tail) to manage them. We defined a set *ForwardPairSet* that contains pairs of numbers of the form *ForPair (N,FN)* where *N* has forwarded a call to *FN*. Other data items needed by our specification are *keyboard keys* and *signals*. Note that the signals *RING*, *BUSY*, *DITO* (DIal TOne), *DISC* (DISConnection), *OOFS* (Out OF Service), *FORW* (FORWard), and *RANO* (Ring Again NOtification) are of special use. They differ from the others on two accounts. First of all, an event with such a signal occurs only if specific conditions are satisfied. Second, they may affect the set *BusySet*.

4.2. Informal Description of the Specification

The specification must be written in such a way as to allow for an arbitrary number of *Subscribers* to use the *Telephone Service* simultaneously. It must also describe the interactions of the system with the environment, for which we define a gate *user*, and of signal exchanges between different components, where we use gate *line*. Within an event, different types of data may be associated with the same gate. The first data item defines the line being used while the second one defines the action being handled on that line (a primitive). Possibly, there is an additional data item to determine the destination line.

Within a single connection, *One Connection*, the *Telephone System* works in the following manner: the *Origination Side* picks up the telephone handset, then it gets: an out of service signal[4] or a dial tone signal if its line is free. In the first situation, the *Origination Side* is forced to hang up its telephone, while in the second one it may start dialing the callee's number. The latter results in one of three situations: the *Origination Side* gets (1) a busy signal, (2) an out of service signal or (3) an audible ring if the callee's line is free. In (2), the *Origination Side* must hang up the telephone while in (1) the *Origination Side* may hang up the telephone and then terminate the call, or may invoke a *Ring Again* feature and then hang up the telephone. Thereafter, the *System Network* keeps looping on the *Destination Side*'s line until it gets free to notify (*RANO*) the *Origination Side*. The latter may then pick up the telephone handset, which causes the communication path to be established, or cancel the *Ring Again* feature without lifting the handset. It can also cancel this feature while waiting for a *RANO* signal. However, in situation (3) the *Origination Side* may wait for an answer or hang up the telephone. The callee's telephone then starts ringing or the *Destination Side* may have forwarded incoming calls to another line, and therefore the *End Destination* is now the party to whom this call was forwarded. If the *End Destination* answers the call before an *OnHook* from the *Origination Side* is detected, then its telephone stops ringing, the communication path between both subscribers is established, and the conversation may begin. Meanwhile, either subscriber may invoke any feature among *Hold*, *TWC*, *Transfer* and *Conference* features.

1. If a subscriber decides to invoke a *Hold* feature, it must first let the other party know then it presses the appropriate key. The other party is now on hold but it may leave the original call at any time. The process of calling a new party within a connection is similar to the one found in a simple connection, except that the *Origination Side*'s telephone is already *OffHook*, and that in order to terminate or abort a call, the *Hold* key again is pressed, which causes re-connection to the first party. However if the first party has left the connection, the *Origination Side* must hang up the telephone after it returns to the first call and the connection is released.

[4] In real world this is not a signal, however the subscriber can recognize it.

2. When a call comes in to its telephone, the *Destination Side* may press the *Transfer* key then it dials the number of the party to whom this call is to be transferred. When the new party answers the call, the *Destination Side* leaves for good the conversation and this new party will be connected to the *Origination Side* in a normal call.

3. If after a transfer of a call, the *Destination Side* doesn't leave the call, all three parties are connected together in a conference like conversation. The feature is then called a *Three-Way-Calling* (*TWC*).

4. While in conversation, a participating subscriber may press the *Conference* key, consult (privately) with a third party then press the *Conference* key again to bring this new party to the conference. Note that an arbitrary number of subscribers may be involved in a conference call and may leave it at any time.

4.3. Architecture of the Specification

As mentioned above, the LOTOS constraint-oriented style is the one which we used the most. In this style, the overall behavior of a system is specified in terms of a parallel composition of the behavior of the component subsystems. The main operators in this style are || (parallel composition with synchronization on all gates) and **|[L]|** (parallel composition with synchronization on all gates in the list **L**). Of course, the "choice" operator **[]** is also used. However, other styles were also used. As a result of using the constraint-oriented style, three types of constraints were identified. The constraints introduced in our Sample Telephone Specification [Bel88] are of different uses. A constraint may be applied within the behavior of a process (*Local*), between two processes (*End-to-End*), or on the overall system behavior (*Global*).

4.3.1. Initial Architecture

The top level of our specification is the parallel composition of two processes, *Multi_Connections* and *System_Network*. The former process creates a single connection, *One_Connection*, and gives the environment the possibility of invoking new connections. To each connection, a local controller of the behavior of subscribers is associated. All the connections must synchronize on internal signal exchanges (gate *line*) with the process *System_Network*. We explicitly hide the gate *line* because in real world the first stage of processing a signal exchange is an internal event to the *Telephone System*. To control the lines and provide subscribers with appropriate signals, the process *System_Network* uses the sets of numbers, *BusySet*, *ServiceSet*, *ForwardSet* and *ForwardPairSet*. Initially, all these sets are empty. To update the set *ServiceSet*, the environment must choose the appropriate action and then enter a number (telephone number). The top level of the specification is presented in Code 1.

```
specification Telephone_System[user] :noexit
  (* ...Abstract Data Type Definitions... *)
  behavior
   hide line in
     Multi_Connections [user,line]
     |[line]|
     System_Network[user,line](empty,empty,empty,nopair)
     where
      (*...Process Multi_Connections Definition...*)
      (*...Process System_Network Definition...*)
endspec (* Telephone_System *)
```

Code 1 Top Level of the Specification

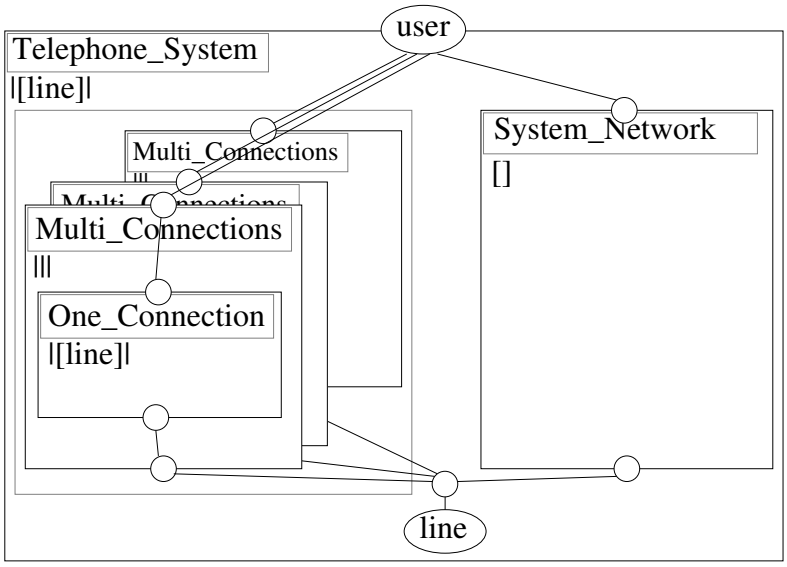

Figure 1 Top Level's Graphical Representation of the Specification

4.3.2. Creating Concurrent Connections

Processes that describe the behavior of different connections are composed in parallel. The LOTOS Interleaving operator "|||" is used because in real world an arbitrary number of connections may be handled in parallel and each connection is processed separately, see Code 2.

```
process Multi_Connections [user,line] :noexit :=
  One_Connection [user,line]
  |||
  i ;
  Multi_Connections [user,line]
endproc
```

Code 2 Process Multi_Connections Definition

Process *Multi_Connections* generates a new connection and then loops on itself to generate other connections in parallel[5]. The graphical representation corresponding to Code 1 and 2 is given by Figure 1, where parallel processes are represented next to each other while interleaved ones are drawn on top of each other. The gate over which processes synchronize is shown in the upper left corner of the box that includes those processes. In this figure, gate *user* is drawn on the border line of the outer box to be the access point for the environment. Gate *line*, however, is drawn inside that box to represent the fact that it handles only internal events.

Process *One_Connection* consists of the processes *Subscribers* and *Connection_Handler* composed in parallel and synchronize with each other on gate *line*. The corresponding graphical representation is shown in Figure 2

[5] The internal action "**i**" following the interleaving operator is used to prevent the LOTOS interpreter from looping endlessly, by attempting to provide an infinite number of connections.

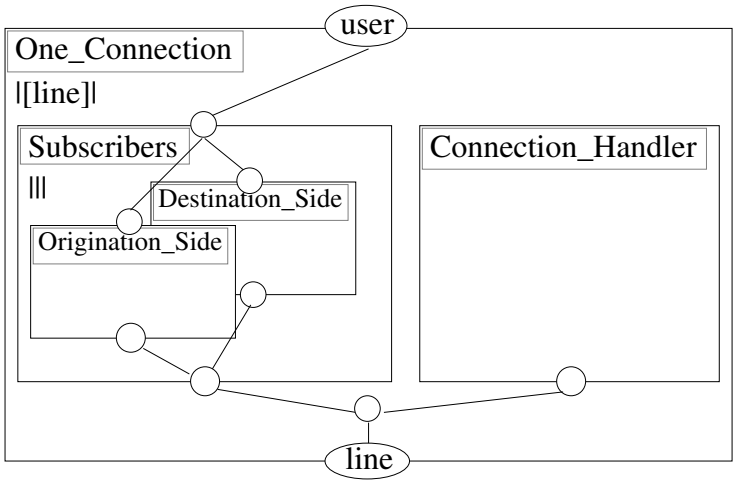

Figure 2 Graphical Representation of a One_Connection

4.3.3. Processing Disconnection of Telephones

Process *Connection_Handler* plays the role of a controller within a single connection. It is responsible for handling the connection of the lines, and also their disconnection, once the associated call has been terminated.

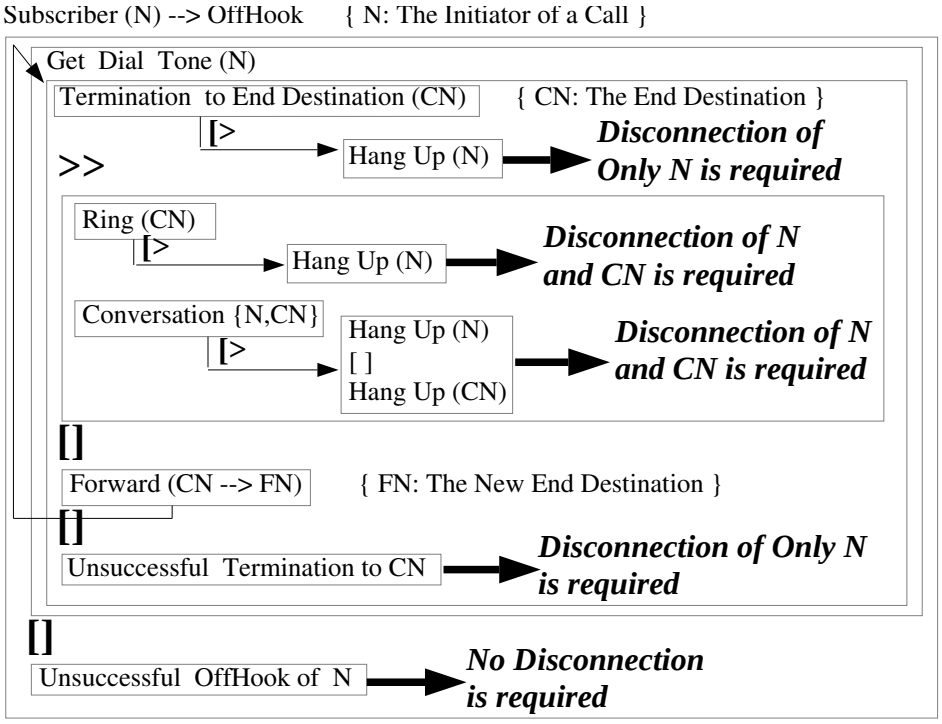

Figure 3 Different Types of a Disconnection

A call may be aborted by the *Origination Side*, may not terminate successfully to the end destination, or may be terminated by a subscriber after its successful termination. To abort or terminate a call we used the LOTOS disable operator "**[>**" that kills the current process by transferring the control to

the process that follows that operator. Instead, the LOTOS enable operator ">>" is used to sequence two processes. When the first process terminates successfully, the control is then transferred to the process following that operator, see Figure 3.

4.4. Design Steps of the Specification

Within a single connection the temporal ordering of the actions performed by different subscribers is handled by a local controller, called *Connection Handler*. The behavior of the participating subscribers is defined in process *Subscribers*.

4.4.1. The Subscribers

Any subscriber, Code 3, may be the originator of a call and its actions are handled in parallel with the ones performed by any specified receiver of that call. However, subscribers don't have to synchronize on any action, and therefore we use the LOTOS *interleaving* operator "|||" for their parallel composition.

```
process Subscribers [user,line] :noexit:=
  Origination_Side [user,line]
  |||
  Destination_Side [user,line]
endproc
```

Code 3 Process Subscriber Definition

After an *OffHook*, the *Origination Side* may get a *DITO* or an *OOFS* signal. This depends on the current status of its line and it is defined in LOTOS by use of the choice operator "**[]**". A sequence of actions is performed by the *Origination Side* before establishment of a connection and the call may be aborted at any time, see Section 4.3.3. The control is passed from one phase of a call processing to another by use of the LOTOS enabling operator ">>". After a successful termination of the first phase, the next phase starts by ringing at the *Destination Side*, by a busy signal, or by an out of service signal. After establishment of a connection, the conversation may begin. The destination line is identified by the dialed number "*CN*", and its behavior is described in process *Destination_Side*. The *Destination Side*'s telephone may start ringing or the call has been forwarded to another telephone, and therefore the *Destination Side*'s telephone must be kept busy. Once it rings, disconnection of the *Destination Side*'s line is required if the *Origination Side* aborts the call and also if the *Destination Side* answers the call. However, if a call is aborted during its first phase, disconnection of the *Destination Side*'s line is not required, see Figure 3.

4.4.2. The Connection Handler

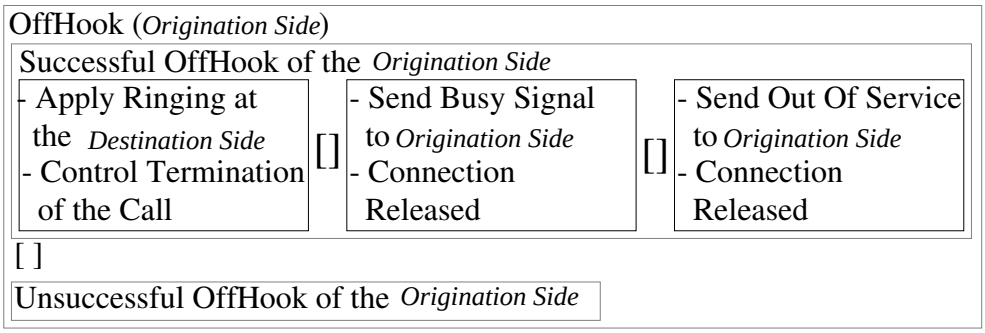

Figure 4 Connection_Handler Refinement

Within process *Connection_Handler*, the signals associated with both sides are ordered according to their origins. Each line is identified by the associated number. The top level description of the process *Connection_Handler* is summarized in Figure 4.

4.4.3. Telephone Features

For modularity of the specification and to ease its modification, each feature is specified in a different process. In this section we describe some of the features that are provided by our *Sample Telephone System*. *Ring Again* and *Call Forward* features are possible only before establishment of a connection. However, the remaining ones, *Hold*, *TWC/Transfer*, and *Conference* features are possible only after the establishment of a connection.

1. The *Ring Again* feature may be invoked by the *Origination Side* when attempting to call a subscriber whose line is currently busy. This feature is described by a process *Ring_Again_Feature* that is instantiated by process *Origination_Side* provided that the requested number (*RN*) is different from its number (*N*). However, if on the detection of a busy signal the *Origination Side* decides to hang up its telephone, the process *Ring_Again_Feature* is not instantiated and therefore the connection is released by a process *Hang_Up*. The process *Ring_Again_Feature* is instantiated as follows:

- - -> ***Ring_Again_Feature[user,line](N,RN)***

2. The *Forward* feature may be invoked only by the *Destination Side* (*CN*). This is to forward all incoming calls to its telephone, to another telephone with a different number. Whenever a call comes in to the first telephone, it is automatically forwarded to the other telephone. This feature remains in effect until it is cancelled. A sample instantiation of such a feature is:

- - -> ***Call_Forwarded [user,line] (CN)***

3. After a connection between two subscribers has been established, either subscriber may invoke any feature among the remaining ones. When such a feature is invoked, the conversation with the other side is disabled and the invoker of the feature may proceed to call a third side. The conversation is also disabled but for good if a subscriber goes *OnHook*. A sample instantiation of such a feature is:

- - -> ***Feature [user,line] (N,FeatureKey)***

In the call of a process *Feature*, *FeatureKey* stands for the key of the feature being invoked. The feature keys are *HoldK*, *TransK*, and *ConfK*.

4.4.4. The System Network

The process *System_Network* synchronizes with process *Multi_Connections* on gate *line*. An event with a signal is possible only if the associated predicate is evaluated to true. A sample configuration of the *System Network* is presented in Figure 5.

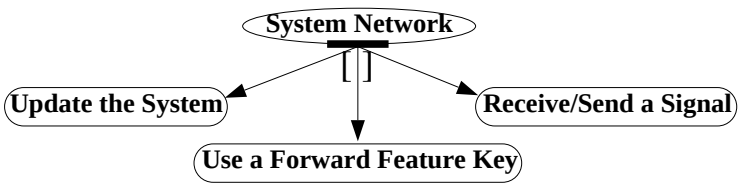

Figure 5 Structure of the System_Network

Note that the *System Network* allows the environment to update the set *ServiceSet* at any time. This requires certain predicates to be satisfied. For instance, a new number can be added to the set *ServiceSet* only if it was out of service (*notin(new_num,ServiceSet)*).

5. Debugging the Sample Telephone Specification

Using the LOTOS interpreter of the University of Ottawa [HH88], we were able to extensively debug our specification of the *Sample Telephone System*. In Step-by-Step execution, the debugging focuses on specific stages of processing a call. It may also be used to check the consistency of a particular feature by validating at each step the sequence of events. Symbolic execution [GL89], instead, is used to generate a tree of symbolic sequences of events involved in a call. This gives all the possible paths allowed by the specification for that particular phase. Note that due to the size of our specification, the generation of the whole symbolic execution tree is not practical. However, symbolic execution trees for some specific processes of the specification were generated using a small depth of execution.

6. A Sample Testing of the Sample Telephone Specification

The testing process is an appropriate way to detect errors in a specification. Using the facilities provided by the interpreter ISLA, some test sequences have been run on our *Specification*. This was done by composing in parallel a test sequence, as a process, with the whole specification. The execution of the composition will result in an execution tree, from which a test sequence may reach its last action. If it does, we say that the test sequence is accepted by the specification, otherwise we say that it is refused. Note that a test sequence must be deterministic in order to be able to judge its acceptance. However, LOTOS concepts allow specification of *non-deterministic* systems [BSS87]. Note also that when composing in parallel the test process with the specification, the internal action '**i**', see Code 2, is replaced by gate '*line*'. This is to avoid the occurrence of multiple **i**'s in the execution tree and also to allow synchronization of a test process with the specification when it involves multiple calls in parallel.

7. Conclusions

We have presented a design methodology to specify Telephone Systems in LOTOS. We have also shown how modern telephone features can be specified and included in the initial specification of a system.

A method to debug LOTOS specifications was presented and test sequences were validated using an interpreter. By precisely specifying the features of the system at the design stage, the development team can achieve a better view of the system's functionality. The specification can then be validated by various debugging, testing, and verification methods, and therefore costly design errors can be detected at the design stage. Finally, an implementation can be obtained from the formal specification, and test cases for the implementation can be derived from the specification. The latter topic is not covered in this paper, however the related methodology is also subject of research [BSS87].

Acknowledgments This work was supported in part by the Algerian Government, the Telecommunications Research Institute of Ontario, and by Bell-Northern Research. We are grateful to J. Sincennes, M. Faci, M. Haj-Hussein, and other colleagues of the Protocols Research Group of the University of Ottawa for valuable suggestions and technical assistance.

Appendix Technical Abbreviations

i. Signals

The following is a list of the signals sent/received by the *System Network* to/from the telephones. They represent internal events within the *Telephone System*.

BUSY : Busy Signal
DISC : Disconnection Signal
DITO : Dial Tone Signal
FORW : Forward Signal
OOFS : Out Of Service Signal
RANO : Ring Again Notification Signal
RING : Ring Signal

ii. Other Abbreviations

In this section, we list some of the keys available on the *Sample Keyboard DMS-100* [Bel88], followed by some common abbreviations used in this paper.

In addition to the *Call Transfer* feature key, any other feature may be activated by first pressing the *Call Transfer* feature key and then dialing the appropriate feature code.

ConfK : Conference Feature Key
HoldK : Hold Feature Key
OffHook : Pick up Handset
TransK : Transfer Feature Key
FDT : Formal Description Technique
ISLA : An Interactive System for LOTOS Applications
ISO : International Organization for Standardization
LOTOS : Language Of Temporal Ordering Specifications
OSI : Open Systems Interconnection
POTS : Plain Old Telephone Systems

References

[BB87] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[Bel88] Bell Canada, Inc. *The DMS-100 Business Set User Manual*, 1988.

[Bou91] Rezki Boumezbeur. Design, Specification, and Validation of Telecommunication Systems in LOTOS. Master's thesis, University of Ottawa, Ottawa, Ontario, Canada, (Forthcoming) 1991.

[Bri88] E. Brinksma. On the Design of Extended LOTOS. A Specification Language for Distributed Systems, 1988. Doctoral Dissertation.

[BSS87] E. Brinksma, G. Scollo, and C. Steenbergen. LOTOS Specifications, their Implementations and their Tests. In G. von Bochmann and B. Sarikaya, editors, *Protocol Specification, Testing, and Verification VI*, pages 349–360. North-Holland, 1987.

[FLS90] M. Faci, L. Logrippo, and B. Stepien. Formal Specifications of Telephone Systems in LOTOS. In E. Brinksma, G. Scollo, and C. Vissers, editors, *Protocol Specification, Testing, and Verification IX*. North-Holland, 1990.

[GL89] H. Guillemot and L. Logrippo. Derivation of Useful Execution Trees from LOTOS by using an Interpreter. In K. J. Turner, editor, *Formal Description Techniques*, pages 311–325. North-Holland, 1989.

[HH88] Mazen Haj-Hussein. An Interactive System for LOTOS Applications (ISLA). Master's thesis, University of Ottawa, Ottawa, Ontario, Canada, November 1988.

[ISO89] ISO, IS 8807. *Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique based on the Temporal Ordering of Observational Behaviour*, May 1989.

[QF87] J. Quemada and A. Fernandez. Introduction of Quantitative Relative Time in LOTOS. In H. Rudin and C.H. West, editors, *Protocol Specification, Testing, and Verification VII*, pages 105–121. North-Holland, 1987.

[Tan89] Andrew S. Tanenbaum. *Computer Networks. Second Edition*. Prentice-Hall, Inc., 1989.

[Tvr89] I. Tvrdy. Formal Modelling of Telematics Services using LOTOS. *Microprocessing and Microprogramming*, 25(1-5):313–317, 1989.

[VSA+89] C.A. Vissers, G. Scollo, R.B. Alderden, J. Schot, and L. Ferreira Pires. *The Architecture of Interaction Systems. The Structuring of Distributed Systems*. Lecture Notes. C.A. Vissers, Enschede, The Netherlands, February 1989.

[VSvS88] C. A. Vissers, G. Scollo, and M. van Sinderen. Architecture and Specification Style in Formal Descriptions of Distributed Systems. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing, and Verification VIII*, pages 189–204. North-Holland, 1988.

[Zav85] P. Zave. The Distributed Alternative to Finite-State-Machine Specifications. *ACM Trans. On Prog. Lang. and Systems*, 7(1):10–36, January 1985.