FDT Tools for Protocol Development

by
A.A.F. Loureiro, S.T. Chanson,
and S.T. Vuong

Technical Report 91-5

Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z2 Canada

FDT Tools For Protocol Development

A.A.F. Loureiro S.T. Chanson S.T. Vuong

Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada V6T 1Z2

Abstract

FDT tools support protocol development by making certain activities feasible, easier to be performed, more reliable, and faster. This paper discusses the desirable properties of FDT tools and classifies them according to the different stages of the protocol development cycle. An assessment of the tools available so far and projections (or suggestions) of the tools to come are given. A list of the tools that have appeared since the mid 1980's is also included.

1 Introduction

Communication protocols are rules and procedures that govern interactions among communicating entities, and as such are crucial to the functioning of computer networks and distributed systems. The increasing use of these systems and the widespread acceptance of the OSI reference model and its standardized protocols have led to a great deal of research activities in the area of protocol engineering in the past decade. It was understood early on that protocol engineering should be based on formal methods that require, in particular, a formal way to specify protocols. The term Formal Description Technique (FDT) was coined in the late 70's to refer to techniques for the exact specification of protocols and services.

Concurrent with academic research activities on FDTs, efforts have been made within ISO and CCITT since the late 70's to develop standardized FDTs. After approximately a decade of hard work, three complementary FDTs prevail in their full status of International Standards (IS) or Recommendations: Estelle (IS9074) and LOTOS (IS8807) defined by ISO, and SDL (Z100) by CCITT. They are designed to have the expressive power and abstraction capabilities to allow for the complete architectural specification of OSI protocols and services at appropriate levels of abstraction.

As the FDTs are becoming mature and accepted in industries, many formal methods and FDT support tools have been developed for use in the different stages of the protocol development cycle. The real success of FDTs will largely depend on the effectiveness of these methods and tools in handling realistic, large scale protocols. The paper by von Bochmann [125] provides an excellent survey on tools developed up to the mid 80's when the FDTs were still in the process of being standardized. Thus, it is not surprising that most of the tools reported were for nonstandardized FDTs. Since then, there has been a proliferation of FDT support tools. This paper is dedicated to examining FDT tools that have been developed since the mid 80's. We shall restrict our attention to the three standardized FDTs, i.e., Estelle, LOTOS and SDL. Sections 2 and 3 discuss the roles of FDTs and FDT tools respectively in the protocol development process. Section 4 classifies the FDT tools based on the different phases of the protocol development cycle. Section 5 contrasts the

tools developed before and after the mid 80's, and section 6 observes the current trends and makes projections for FDT support tools in the coming decade. A summary of the FDT tools since the mid 80's is given in the appendix.

2 The Role of FDTs

FDTs are essential in every phase of the protocol engineering process. In general, FDTs provide a basis for: (i) the development of unambiguous, clear and concise specifications, (ii) the verification of specifications, (iii) the functional analysis of specifications, (iv) the development of implementations from a specification, and (v) the determination that an implementation conforms to its specification (i.e., conformance testing).

To support (i), (iv) and (v), an FDT should satisfy the two basic requirements of expressive power and abstraction level. Expressive power refers to the ability of an FDT to allow its users to easily compose, survey, understand, modify and extend a specification. This includes the requirements for conciseness and structuring facilities for specifications, e.g., process composition, abstraction and instantiation, and recursion. The abstraction level requirement refers to the ability of an FDT to allow the specification of the concepts of the application area, i.e., the OSI architectural concepts and constructions, at the appropriate level of abstraction. In this respect, OSI concepts (e.g., service access points, connection endpoints, service primitives, protocol data units and constraints) should be expressible in a completely implementation independent way without introducing a morass of arbitrariness and irrelevant details in the specification.

To support (ii) and (iii), an FDT should have a strong mathematical basis that makes it easier to analyze and prove desirable syntactic and semantic properties of protocols.

We note the standardized FDTs have been primarily developed and used for the support of the specification phase of the OSI standards. Estelle has a complete formal definition based on two paradigms: extended finite state machine (EFSM) and the procedural language Pascal. LOTOS also has a complete formal definition based on two paradigms: a transition system for expressing the dynamic behavior in the process part, and an abstract data type (ADT) definition based on ACT ONE in the data part. Both SDL syntaxes (PR–textual and GR–graphical) are formally defined and can be mixed at the discretion of the user. SDL also has two paradigms: finite state machine (FSM) and an ADT based on ACT ONE. [29, 38] present some useful comparisons and guidelines for using Estelle, LOTOS, and SDL.

The success of FDTs will depend on their ability to support not only the specification phase, but the complete protocol development process. This implies the FDTs should be effectively integrated with the design methods. FDTs are not goals in themselves, but they provide means to achieve goals, which can be determined by the degree in which the integrated methods and tools improve the quality of designs and reduce the time and the resources needed in the design process. The report by Vissers [123] provides an excellent perspective of FDTs. In the next section, we discuss the role and desirable characteristics of FDT support tools.

3 The Role of FDT Tools

The application of FDTs to the development of large, complex protocols and distributed systems depend very much on the availability of appropriate computer tools. The complexities of the theories behind the methods and the time required to process them often prohibit the use of formal methods without the supporting tools. The FDT provides the formalism and the FDT tools allow the application of the formal methods in a mechanized manner. More and better tools will popularize the use of FDTs that should produce the following advantages:

- better quality—the tool can partially or fully automate a process hard to do manually, avoiding long development time and human errors. Users can use formal methods and optimization techniques without having to fully understand the theories behind them.
- faster results—automation results in higher productivity.

Desirable characteristics of FDT Tools In addition to allowing formal methods to be used in the protocol development cycle, FDT tools are also software engineering tools. Thus some desirable characteristics of software engineering tools also apply to FDT tools. We present below some considerations relevant to FDTs from the perspective of protocol development. Further details are given in section 4.

Ease of use It should include facilities to support:
- User interface—a nice user interface improves productivity. A tool that supports concurrent multiple graphics and text windows would be useful. Of the three FDTs, only SDL supports both graphical and textual specifications. A graphical syntax for LOTOS called G-LOTOS [82] is under study. Currently, there are tools that support Estelle/LOTOS specifications in a graphical form.
- Customization—a tool will be used more effectively if it is possible to tailor various aspects of the interface to suit the user needs (including I/O formats), and if the user can define new commands.
- Helpfulness—a tool should help and guide the user perform his task. An interactive tool that prompts the user for command parameters, for example, would be desirable. Another useful facility is a tutor system to guide beginners.
- Error handling—a tool should check for and correct user errors whenever possible.

Power The power of a tool can be analyzed at two levels: internal and external. The internal level is related to the functions it is capable of performing and how fast they can be done. Certainly a tool that supports all phases of protocol development is more powerful than those that only support syntax editing, for instance. The external level is related to the possibility of interfacing with other tools and the environment in which the tool is embedded.

Robustness A robust tool includes facilities to support:
- Consistency—it would be desirable if a tool allows concurrent access to the specification by members of the project team and maintains data integrity under simultaneous update attempts. This is important since formal specification is used in all stages of protocol development and different people may be working on different aspects of the specification.
- Evolution—tools evolve over time to accommodate changes in the environment and/or requirements. They should be designed to handle the changes well and to retain compatibility between versions.
- Version control/management—specifications may evolve over time to accommodate modifications, new improvements or features, and may be presented in different specification styles to facilitate different aspects of protocol development. Therefore it would be desirable to maintain, compare, and control multiple versions of the same specification.

- Fault tolerance—a tool should be able to recover from environmental failures during its execution. If a failure occurs in the middle of a long computation such as reachability analysis, it would be desirable to continue the analysis from the last checkpoint performed and not start the process again.
- Self-instrumented—the FDT tool is a piece of software that may contain errors. Therefore, the tool should have some kind of self-instrumentation to help locate the source of an error.

4 Classification of FDT Tools

Several classifications of FDT tools have been proposed. Bruijning [29] uses a classification based on the functions provided by the tools:

- Syntax checking—parses an FDT specification, generates error messages if present, and can create an internal representation if the specification is syntactically correct (e.g., tree-form representations for Estelle, and abstract syntax trees for LOTOS).
- Static semantic checking—verifies at "compilation time" or before "simulation" if the semantics conforms to the standard (e.g., module interfaces). It is natural to integrate the static semantic checking with syntax checking.
- Execution tools—can be divided into compilers or translators that translate a specification into a high-level programming language or into the machine code of a target machine, and simulators or interpreters that simulate/interpret a specification on a computer.
- Dynamic state tracking and transition stepping—helps in the debugging phase by tracing the activities during execution. These tools improve productivity and help to find errors in addition to those locatable in syntax and static semantic checks.
- Deadlock and starvation detection—deadlock and starvation are special dynamic behavior properties that can be defined and detected by more general flow analysis techniques. For example, deadlock can be defined by a temporal logic formula and proven by simulation and model checking. Deadlock and starvation are two important properties that should be detected by a tool.
- Test generation—it would be desirable to automatically generate test sequences for a given specification.
- Refinement process verification—this question is related to equivalence between refinements of the same specification. It would be very useful to have a top-down specification process. Unfortunately, none of the specification languages support this and therefore tools are required.

Turner's [115] classification of software tools for formal methods has a stronger software engineering flavour than that of Bruijning's but is not as fine-grained:

- Bookkeeping tools—allow specifications to be created, maintained, and printed. In case of existence of multiple versions, an automatic version control mechanism would be very useful.
- Front-end tools—manipulate directly the specification text in a computer readable form. They include: lexical analyzer, syntax checker, static semantics analyzer (to check type compatibility, etc.), parser, formatter (to print a specification), and cross referencer.

- Verification tools—analyze properties that can be extracted from formal specifications. These include: algebraic simplifier (to reduce expressions to a simpler or standard form), verifier or theorem prover, proof checker/editor/assistant, state space analyzer or reachability analyzer, equivalence/congruence checker, and symbolic execution. Other analysis tools are language-specific, for example to check for sufficient completeness of axioms (for ADTs).
- Back-end tools—are concerned with the transformation and implementation of a specification.

Based on the applications of the development methods and tools, von Bochmann [125] classifies the tools into three broad categories:

- tools for specification development,
- tools for creation of implementation code,
- tools for testing.

We shall use a hierarchical classification scheme based on the phases in the communication protocol development cycle as we feel the main reason for the tools is to reduce the effort and time required for protocol development. Thus, there are four classes of tools corresponding to the phases of specification, validation and verification, implementation, and testing. Each class is subdivided as follow (using Turner's terminology whenever possible):

- **Specification**
  - creating, editing, maintaining and printing of the specification (bookkeeping tools)
  - syntax and static semantic checking of the specification (front-end tools)
- **Validation and Verification**
  - checking of syntactic properties of the protocol (validation)
  - checking of semantic properties of the protocol (verification)
- **Implementation (back-end tools)**
  - translative tools such as compilers and translators, including tools that translate specifications to object code or some other forms such as programming languages, intermediate forms and other FDTs.
  - symbolic tools such as interpreters and simulators
- **Testing**
  - active testing:
    * test case generation
    * test case management, including editing, maintaining, storage and retrieval of test cases
    * test driver (straightly speaking, the test driver is FDT-independent)
  - passive testing (monitoring communication activities and analyzing them for errors against the specification - also known as result analysis).

Note that some tools provide services that cross category boundaries. For example, compilers will also check for syntax errors in the specification, and some tools for verification can be easily enhanced to perform result analysis as well. Independent of the classification used, tools are needed that automate or help to reduce the human effort in each of the phases of the protocol development cycle mentioned above.

5 Past and Current FDT Tools

In 1987, von Bochmann [125] presented a survey of protocol development tools. That survey was derived from a more extensive study [60]. Of the 58 tools reported, only 17 (less than a third) supported Estelle, LOTOS or SDL. Estelle accounted for eight tools, LOTOS two, and SDL seven. Until 1986, the use of FDTs was quite limited; both Estelle and LOTOS were in the draft proposal state. However, Estelle caught on much faster than LOTOS because it is procedure (and therefore implementation) oriented and is more similar to the classical programming languages. LOTOS, on the other hand is property oriented, more abstract and difficult to learn. It is therefore understandable that Estelle tools out-numbered LOTOS tools by a factor of four in the survey. However, it is interesting to note the presence of only seven SDL tools, one less than Estelle. SDL was intended for applications in telecommunications and was available before Estelle or LOTOS.

During the last five years, activities in the development of FDT tools have increased enormously. More than 60 tools are included in the appendix. LOTOS has become the favorite FDT perhaps because users have come to appreciate the strong mathematical foundation of LOTOS and its power in describing complex system behavior concisely. Of course, LOTOS has shortcomings too. For instance, some LOTOS specifications can be very hard or even impossible to be implemented [120]. Most of the LOTOS tools reported support only a subset of LOTOS. This means that a lot more work is needed to make the tools truly useful.

Tables 1, 2, and 3 summarize the Estelle, LOTOS, and SDL tools described in the appendix. By and large, the functions of these tools are limited to the following (using our classification scheme):

- Specification tools—syntax oriented editors with some support to check static semantics.
- Validation and Verification tools—checking of syntactic properties of the protocol only, very few tools check for semantic properties.
- Implementation tools—most are translators that generate C code to run under UNIX.
- Testing tools—most of them are restricted test case generation tools.

Thus, the current state-of-the-art of FDT tools has a long way to go to cover the full spectrum of activities in the protocol development cycle.

Legend for Tables 1, 2, 3, and 4:

Specification:	**B**ookkeeping, **F**ront-end
Implementation:	**C**ompiler, **S**imulator
Validation&Verification:	**Sy**ntactic, **Se**mantic
Testing:	**A**ctive, **P**assive

Tool	Spec		V&V		Impl		Test	
	B	F	Sy	Se	C	S	A	P
Contest-Estl					⋆		⋆	
EC compiler					⋆			
Echidna Project					⋆			
EDB						⋆		
EDS	⋆		⋆		⋆			⋆
ESTIM						⋆		
EVA			⋆					
EWS	⋆	⋆			⋆	⋆		
GROPE						⋆		
NIST Tool	⋆	⋆			⋆	⋆		
PEDS	⋆				⋆	⋆		
PEW	⋆				⋆	⋆		
PIPN			⋆		⋆	⋆		
TVEDA							⋆	
UBC Compiler					⋆	⋆		
VEDA		⋆			⋆			
Xesar				⋆				

Table 1: Estelle tools.

Tool	Spec		V&V		Impl		Test	
	B	F	Sy	Se	C	S	A	P
Aldébaran				⋆				
ASDE	⋆				⋆			
Auto				⋆				
Cæsar					⋆			
CENTAUR	⋆	⋆			⋆	⋆		
COOPER							⋆	
Data Type Compiler					⋆			
KDD Lotos-C Compiler					⋆			
LCRIS	⋆	⋆				⋆		
LISP-based LOTOS Env.						⋆		
LOLA					⋆			
LOTEST							⋆	
LOTO Parlog Transl.					⋆			
LOTOS Simulator in OBJ						⋆		
LOTTE		⋆				⋆		
PIL		⋆			⋆			
SEDOS Tools	⋆	⋆			⋆	⋆		
Smile						⋆		
SPIDER						⋆		
Squiggles				⋆				
TETRA								⋆
Timed LOTOS Interp.						⋆		
Topo					⋆			
UO-GLOTOS	⋆				⋆	⋆		⋆
UO-LOTOS Toolkit						⋆		

Table 2: LOTOS tools.

Tool	Spec		V&V		Impl		Test	
	B	F	Sy	Se	C	S	A	P
AT&T SDL Tools	⋆		⋆		⋆	⋆		
Danish SDL-Tool	⋆	⋆			⋆			
DASON	⋆	⋆			⋆	⋆		
ELVIS	⋆	⋆		⋆			⋆	
ESCORT	⋆		⋆					
FOREST							⋆	
GEODE	⋆	⋆			⋆	⋆		
ISET	⋆	⋆			⋆			
MELBA+	⋆	⋆			⋆			
SDL-Ada Translator					⋆			
SDL Based SW Devel.	⋆	⋆			⋆	⋆		
SDL-Chill Translator					⋆			
SDL Tools from LSU	⋆				⋆			
SDL Tower	⋆	⋆						
SDT-2	⋆	⋆				⋆		
SSI		⋆			⋆	⋆		
TA-2			⋆					
TSDL-Tool			⋆					
TESDL							⋆	
YAST	⋆	⋆			⋆			

Table 3: SDL tools.

It is not fair to compare the total number of tools reported for each FDT because some of them perform more than one function and span category boundaries according to our classification. A more reasonable way is to count the number of different functions performed by the tools according to the classification of section 4. This is presented in Table 4.

FDT	Spec		V&V		Impl		Test	
	B	F	Sy	Se	C	S	A	P
Estelle	5	3	3	1	11	9	2	1
LOTOS	5	5	0	3	11	11	2	2
SDL	13	11	4	1	12	6	3	0
Total (124)	23	19	7	5	34	26	7	3

Table 4: Number of tools that perform each function.

As can be seen, most of the tools support the implementation and specification phases of protocol development and only a few tools are in the areas of verification and testing. This is consistent with the historical development of the tools. The first tools to appear were specification and implementation tools while the users were content with ad-hoc verification and testing. Essentially, if a protocol implementation worked in practice, it was considered verified and tested. Table 4 shows that there is a need to develop more formal tools for verification and testing.

As the field matures, there is a tendency to develop new integrated tools that support more than one stage of the protocol development cycle, and to integrate these tools with those developed earlier. Table 5 shows the "integration level" of the tools according to the classification presented in section 4. Each row shows the total number of tools that supports one, two, three or all four stages of the protocol development cycle. In the three appendices there are 62 tools that perform

FDT	# of Functions			
	1	2	3	4
Estelle (17)	9	7	0	1
LOTOS (25)	18	6	1	0
SDL (20)	7	11	2	0
Total	34	24	3	1

Table 5: Integration level.

124 functions, i.e., 2 different functions per tool. Presently, most of the tools perform only one function. SDL has more tools supporting two stages of protocol development because it is a more mature FDT.

Finally, we observe that many tools were not developed from scratch. They embedded and/or used other tools such as YACC, LEX, CSG (Cornell Synthesizer Generator) [117], and X Windows. At least two FDT tools are used in other tools: the HIPPO simulator (LOTOS) is embedded in LOTTE (LOTOS) and Xesar (Estelle) is used in Elvis (SDL) to perform verification. In the latter case, an SDL specification is translated into Estelle/R. This allows the FDT tools to be built faster and more systematically.

6 FDT tools in the 1990s

Based on the current activities and trends in FDT research, we can try to predict the type of FDT tools that will be available in the next several years. Over the long term, we would like to see tools that satisfy the desirable properties described in section 3. Generally speaking, we expect to see more and more truly integrated toolsets; tools that support a complete environment for protocol development. The tools in a toolset share internal data representations and/or are input/output compatible. Since integrated tools must address issues related to the development cycle in addition to isolated functions, we expect the tools will have a more heavily software engineering orientation. Thus the object model will be used more and more especially because of its properties of encapsulation and inheritance. Tools that transform a specification written in one style to another while preserving the semantic properties should become available since both verification and testing could be made easier if specifications follow some specific structures. Also, we expect tools designed to run on multiprocessor and other parallel computers will begin to appear as parallel processing become popular. As well, people will be more and more concerned with the performance of protocols. Thus we expect to see tools that estimate protocol performance based on timing information defined in the protocol specifications (already, there are LOTOS and SDL enhancements which allow time to be specified, and so does Estelle's delay clause). Of course, there will be tools that measure and tune the performance of protocol implementations, but these tools are likely to be FDT-independent.

All of the tools will have better and more user-friendly interfaces, many of which will be graphical to make them easier to use. Artificial intelligence techniques will likely be used to assist the user perform his tasks and to make the tools run more efficiently.

Specific comments on tools in each of the categories in our classification are given below:

- Tools supporting specification analysis and design
  As mentioned above, tools for transformation from one style of FDT to another style to enhance testability and to facilitate verification are needed. Also, more and more tools will translate specifications written in an FDT to other forms such as another FDT, a high level programming language and even some intermediate form. Research on intermediate form representations should continue as some forms are more suitable for test case generation, others for verification or result analysis, and yet others for implementation. The use of intermediate form for checking equivalence is an interesting subject and we may see some tools in the near future serving this purpose.

- Tools supporting verification and validation
  As can be seen in Table 4, more tools are needed in this category. We define validation to be the process of checking the syntactic properties of a protocol (such as deadlock, unspecified receptions, livelock etc.). Verification is the process of checking the semantic properties of a protocol. Most of the research in the past has been on validation because it is a simpler process. We believe that work in verification will receive more attention and will result in more tools. Initially they are likely to be driven by the results of program and hardware verifications.

- Tools supporting protocol implementation
  This is related to tools for specification to some extent. The idea is to translate an FDT specification into some form suitable for execution or interpretation. Tools that take an FDT specification as input and generate a program in a high-level language as output could be called source-to-source translators. A methodology that can be applied in this process is to

derive from algorithm $\mathcal{A}$ in FDT $\mathcal{F}$ an algorithm $\mathcal{A}'$ in high-level language $\mathcal{L}$. Note that each distinct part of algorithm $\mathcal{A}'$ must be semantically equivalent to the correspondent part in algorithm $\mathcal{A}$. Depending on the formalism used in the FDT, a construct in $\mathcal{F}$ can be translated into one or more constructs in $\mathcal{L}$. Once this mapping process has been done it is fairly easy to develop such translators. The high level language can then be compiled into the object code of the target machine. Estelle, LOTOS, and SDL are based on formalisms that have little relationship to the traditional computer architecture, i.e., the von Neumann machine. This makes it hard to directly compile FDT specifications into object code.

- Tool supporting FDT-based testing
  Again, there is a shortage of testing tools. Currently, most test case generation methods start with a finite state specification of the protocol and generate test cases based on control flow only. More work will consider test case generation from FDT specifications and will include data flow as well. We will also see tools supporting the complete testing environment, including tools that manage the test cases generated, possibly using some relational database systems. These tools will also support dynamic selection of test cases based on the result of testing and may provide some help in locating the errors when a test case fails. Possibly we will need knowledge-based tools to exploit the full potentialities of FDTs.

7 Conclusions

The importance of FDTs and FDT tools in the protocol development cycle are beginning to be understood. This is evidenced by the dramatic increase in research activities in this area in the last several years and more FDT tools are appearing each year. The application of FDTs to real problems will become more popular once better, more complete, and more efficient FDT tools are available. In this paper, we have outlined the desirable properties of FDT tools and classified the tools using a hierarchical scheme based on the four phases of the protocol development cycle. This is followed by an assessment of the tools available so far and predictions (or suggestions) of the tools to come. A list of the Estelle, LOTOS and SDL tools that have appeared since the mid 80's is included in the appendix. It is hoped that this paper will increase the awareness of FDT tools and the roles they play in the protocol development cycle.

Appendix

Sections A, B and C of the appendix contain brief descriptions of tools based on Estelle, LOTOS, and SDL respectively. The list of tools has been compiled from the open literature since 1987 and it is by no means exhaustive. We believe the tools reported here are representative of the state-of-the-art tools in the last five years but there will be others we have missed. [125] contains a survey of tools before 1987.

The general format of each item in the appendix is: tool_name, category according to section 4, description of the tool, language used in the implementation and platforms supported, and applications where the tool has been used. In the case where a tool supports more than one FDT, the FDTs are listed under the heading FDT supported.

A Estelle based tools

A.1 CONTEST-ESTL

Category Implementation/Compiler; Testing/Active

Origin Department of Electrical and Computer Engineering, Concordia University, Montréal, Québec, Canada H3G 1M8

General Information

■ *Description:* Contest-Est [48] takes modular specifications in Estelle as input and executes the following steps:

- Compilation—performs the lexical, syntactic and semantic analysis, and generates a parse tree and a symbol table, but no code is generated.
- Normalization—transforms the original specification into another one where all transitions have single paths. This step also generates the following intermediate forms that are used in the next stage: control flow graph in a tree form, control flow graph in an FSM form, and data flow graph.
- Display the structure, control and data flow graphs.
- Generate test sequences to cover both the control and data flow of the specification.

■ *Language and Platforms:* The tool is implemented in C on a SUN workstation.

■ *Applications:* [48] shows the test sequences generated for a simplified transport protocol described in [124].

A.2 EC COMPILER

Category Implementation/Compiler

Origin Bull S.A., France

General Information Bull S.A., DRCG-ARS, 68, Route de Versailles, 78430 Louveciennes, France

■ *Description:* EC [20] has two components: a translator and a code generator. The translator takes as input the Estelle source text, detects syntax errors, and checks the static semantics (i.e., integration of the syntactic and static semantic checking). The output generated is an intermediate code composed of three parts: a symbol table, a transition table, and a tree-form representation of the transition blocks.

The code generator of the Estelle Compiler takes as input the intermediate code generated by the translator and produces code in C and ML (a metalanguage developed by INRIA). It is possible to take this intermediate code and translate it to other programming languages or as input to a simulator/interpreter.

■ *Language and Platforms:* EC is written in Pascal (version of the SYNTAX tool developed at INRIA) and runs on computers from Telmat and Bull under UNIX.

■ *Applications:* [56] discusses the experience of describing the OSI-TP protocol (Distributed Transaction Processing) in Estelle. From that specification, C code is generated using the EC compiler. At INRIA, a preprocessor to EC was built to allow conditional compilation.

A.3 ECHIDNA PROJECT

Category Implementation/Compiler

Origin IRISA Campus de Beaulieu, F-35042 Rennes Cedex, France

General Information

■ *Description:* The Echidna project creates an environment to prototype distributed algorithms on parallel machines. It provides:

- Estelle compiler [64, 65] for multiprocessor machines—translates an Estelle specification into C code to be compiled and linked with a system dependent kernel. The target system can be mono or multiprocessor.
- Software tools [63] to observe the behavior of the distributed algorithm under experimentation

■ *Language and Platforms:* The compiler is written in Pascal and runs on Intel iPSC/1 and iPSC/2 hypercubes, networks of SUN workstations using TCP/IP, Transputer based FPS-T40 hypercube, Gould and PCs.

■ *Applications:* The Echidna tool has been used in some synchronizers study performed at IRISA, and in the MAC/TR atomic fault resistant diffusion protocol validation (DELTA 4 Esprit Project) [64].

A.4 EDB

Category Implementation/Simulator

Origin Bull S.A., DRCG-ARS, 68, Route de Versailles, 78430 Louveciennes, France

General Information

■ *Description:* EDB [19] has two components: a simulator and a debugger. The simulator can execute an Estelle specification in two ways: step-by-step or in an uninterrupted mode.

The debugger can provide a report with error and warning messages; detect logical design errors (such as deadlocks, and undesirable sequences of transitions); and insert observers. Observers are commands (trace, break-points) in EDB query language that are performed after firing a transition.

An Estelle Debugger specification is composed of three parts:

- Debugger scheme—describes the algorithms of the debugger (based on the inductive operational definition of Estelle semantics) and the handling of time parameters.
- Command interpreter—the commands of the Estelle debugger help the user to navigate in the tree of module instances.
- Error processing—using the commands of the debugger it is possible to display variables and modify those that are permissible. If desired, the user may also make choices to resolve nondeterminism.

■ *Applications:* [56] discusses the experience of describing the OSI-TP protocol (Distributed Transaction Processing) in Estelle. That specification is simulated and debugged using EDB.

A.5 EDS

Category Specification/Bookkeeping; Validation; Implementation/Compiler; Testing/Passive

Origin Pheonix Technologies Ltd., 675 Massachusetts Ave., Cambridge, MA 02139, USA

General Information

■ *Description:* EDS (Estelle Development System) [27] is a protocol development system to support the specification, implementation, verification, and testing of protocols. It has the following components:

- Estelle compiler—translates an Estelle specification into C code.
- Document generator—prints an Estelle specification using a text formatter.
- Cross-Reference generator—generates extensive tables and is responsible for answering queries about definitions, typing and usage.
- FSM analyzer—analyzes the protocol state machines to verify some properties such as deadlock-freeness, boundedness, termination, and completeness.
- Test driver—simulates the network and monitors the behavior of an implementation during tests.

■ *Applications:* [27] describes the analysis of the Key Distribution protocol.

A.6 ESTIM

Category Implementation/Simulator

Origin LAAS, 7 Avenue du Colonel Roche, 31077 Toulosse Cedex, France

General Information

■ *Description:* ESTIM (Estelle SimulaTor based on an Interpretative Machine) [35, 36, 37] has two components:

- Interpreter—takes as input the ML code generated by the Estelle Compiler and interprets it. Its underlying semantics are based on an extended Colored and Timed Petri Net model. It consists in interpreting the Estelle operations and the firing of transitions. The selection of transitions can be made by the user or randomly by ESTIM.
- Simulator kernel—intended mainly to debug an Estelle specification before implementing it. It implements the Estelle semantics defined in [31]. Using Estelle* it is possible to analyze the reachability graph associated with the specification. ESTIM also provides an interactive access to display elements of the global state and modify them when appropriate.

■ *FDT supported:* Estelle* [30, 32]—variant of Estelle where communication is modelled by rendez-vous.

■ *Language and Platforms:* ESTIM is written in ML and runs on the SUN 3 workstation under UNIX and BULL SPS7/SPS9 machines under SPIX.

■ *Applications:* [36] describes several ISO and non ISO protocols and services that have been simulated using ESTIM.

A.7 EVA

Category Validation

Origin CREI, Politecnico di Milano, Plazza Leonardo da Vinci 32, 20133 Milano, Italy

General Information

■ *Description:* EVA (Extended Validation Algebra) [28] is based on algebraic method that allows to discover any deadlock or livelock present in a system that can be modeled as a

hierarchical structure of modules. It is possible to represent and simulate sequential and parallel communication.

■ *Language and Platforms:* EVA is written in C and runs under UNIX.

A.8 EWS

Category Specification/Bookkeeping, Front-End; Implementation/Compiler, Simulator

Origin VERILOG, Toulose, France

General Information

■ *Description:* EWS (Estelle WorkStation) [6] is an integrated environment that supports the following tools:

- Syntax oriented editor—performs the syntax checking of an Estelle specification and generates a syntax-free Estelle source file in ASCII.
- Translator—translates an Estelle specification into an intermediate form making a static semantics check.
- C code generator—produces C code from the intermediate form that can be compiled with the Estelle primitives library.
- Implementation kernel—run-time system. It has an Estelle run-time primitives library and specific machine-dependent routines.
- Debugging oriented simulator—interactive environment to debug Estelle specifications. It is possible to access simultaneously the source file of the Estelle specification, the intermediate form, and the C code.

■ *Language and Platforms:* EWS runs on Appolo and SUN workstations under UNIX.

■ *Applications:* [6] cites some protocols in different areas that have been tested under EWS. In particular, the protocols ISO Transport class 4, ISO Session, ISO ACSE and ROSE, and ISO FTAM.

A.9 GROPE

Category Implementation/Simulator

Origin (`grope@udel.edu`); University of Delaware, Newark, DE 19711, USA

General Information

■ *Description:* GROPE (Graphical Representation Of Protocols in Estelle) [85, 86] provides a dynamic (graphical) representation of the execution of an Estelle specification. It is a window-based system that displays the architecture of a protocol, the extended finite state machine of a module and the changing of states, and animates transitions firing and the exchange of interactions between modules. It is possible to select transitions to resolve nondeterminism, examine the details of objects represented graphically, and to control the interpretation of a specification over time.

It is also possible to interface other Estelle tools that follow the international standard, such as the NIST WISE simulation environment.

■ *Language and Platforms:* GROPE is implemented in Smalltalk-80 and runs on SUN worksta-

tions.

■ *Applications:* [86] describes an experience of using GROPE in the simulation of the Virtual Terminal specification in Estelle (according to the ISO international standard) and the Fact Exchange protocol (to support reliable exchange of independent "chunks" of data over very low bandwidth communication channels).

A.10 NIST Integrated Tool Set

Category Specification/Bookkeeping, Front-end; Implementation/Compiler, Simulator

Origin U.S. Department of Commerce, National Institute of Standards and Technology, National Computer Systems Laboratory, Systems and network Architecture Division, Gaithersburg, MD 20899, USA

General Information

■ *Description:* Presently, the NIST integrated tool set is comprised of the following tools:

- NIST Estelle Compiler [108, 109]—code generator. This Estelle to C compiler checks the syntax and static semantics, and translates Estelle constructs into tables and structures in C code. The C code generated must be linked with a run time library to produce the executable code.
- WISE [103, 104]—simulation environment. An object-oriented model for Estelle was defined [105] that can be applied to distributed implementations of Estelle specifications. That model was incorporated into the WISE tool which has a syntax-directed editor, static semantics checking, and code generation.
- WISARD [107]—editor and translator. Similar to WISE, WISARD is a syntax-directed editor and translator based on the Cornell Synthesizer Generator from Cornell University [112].
- NIST Portable Estelle Translator (PET) [106]—checks the correctness of an Estelle specification (static syntax and semantics checking). The output is an object file containing the external representation of the specification in the static model.
- DINGO [106]—code generator. Based on the object file produced by PET, it generates distributed implementations of Estelle specifications in C++.

■ *Language and Platforms:* The WISE tool is implemented in Smalltalk. PET and DINGO are implemented in C++.

A.11 PEDS

Category Specification/Bookkeeping; Implementation/Compiler, Simulator

Origin U.S. Department of Commerce, National Institute of Standards and Technology, National Computer Systems Laboratory, Systems and network Architecture Division, Gaithersburg, MD 20899, USA

General Information

■ *Description:* PEDS (Prototyping in Estelle for Distributed Systems) [78] is an environment integrated by a language preprocessor, support libraries, automatic editing facility, and a compilation control facility which automates the use of the tools. The tool is based on the NBS prototype Estelle compiler to translate an Estelle specification into C language. After this ini-

tial step, the code can be compiled and linked with support libraries to produce an executable version of the system. This version can be used for simulation or distributed execution.

■ *Language and Platforms:* PEDS runs on UNIX

A.12 PEW

Category Specification/Bookkeeping; Implementation/Compiler, Simulator

Origin Data Network Architectures Laboratory, Department of Computer Science, University of Cape Town, Private Bag, RONDEBOSCH, 7700 South Africa

General Information

■ *Description:* PEW (Protocol Engineering Workstation) [74] is an integrated environment that contains the following components:

- Editor—full-screen editor with keyboard assignable macros, context-sensitive help, and cut-and-paste operations. It uses an internal representation to store Estelle specifications shared by all components. The editor is independent of Estelle except in two features: the language help and the template expansion facilities.
- Compiler—produces a symbol table and a pseudo-code called E-code that is an extension of Pascal p-code.
- Interpreter Environment—contains the following modules: E-code interpreter (interprets the E-code generated by the compiler), Process Manager (collects and outputs statistics regarding processes), Scheduler (high-level control of execution), Queue Manager (similar to the Process Manager but for message queues), Memory Manager (handles dynamic memory allocation for all subsystems), and a Symbolic Debugger (provides a high-level interface to the interpreter).
- Performance Analyzer—uses the statistics collected by the interpreter environment to predict the protocol performance.

The tool is integrated in the sense that the editor can be invoked during compilation or simulation.

■ *Language and Platforms:* The PEW tool is implemented in C on an IBM-PC running DOS.

■ *Applications:* The alternating bit protocol is used in [74] to exemplify different modules of the tool.

A.13 PIPN

Category Validation; Specification/Compiler, Simulator

Origin LAAS, CNRS, 7 Avenue Colonel Roche, F-31077 Toulouse Cedex, France

General Information

■ *Description:* PIPN (Prolog Interpreted Petri Nets) [7] is a prototype for a Petri net based verifier. It uses an underlying model for Estelle based on Predicate Petri nets. The properties of the system are expressed using Temporal Logic formulas which are checked and evaluated on the reachability graph of the Predicate net model. The tool has three components:

- Compiler—performs the composition of modules and derives a global Predicate Net.
- Simulator—allows concurrent execution of events.
- Verifier—performs deadlock and livelock detection, temporal logic assertions, and abstract automaton derivation.

■ *Language and Platforms:* PIPN is written in Prolog.

A.14 TVEDA

Category Testing/Active

Origin CNET LAA/SLC/EVP, BP-40, F-22301 LANNION Cedex, France

General Information

■ *Description:* TVEDA [93, 94] implements algorithms to derive test cases from an Estelle specification. The tool uses a pragmatic approach in the sense that the algorithms reflect the methodology used by experts in Conformance Testing Centers to generate test cases manually.

■ *Language and Platforms:* TVEDA is based on VEDA and is implemented in Prolog.

■ *Applications:* [93] describes the experience of generating test cases in TTCN for the ISDN LAPD protocol specification in Estelle.

A.15 UBC Estelle-C Compiler

Category Implementation/Compiler, Simulator

Origin Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada V6T 1Z2

General Information

■ *Description:* The UBC Estelle-C Compiler [128] accepts an Estelle specification as input and generates a source file in the C language as output.

■ *Language and Platforms:* The UBC Estelle-C Compiler is written in C and runs on SUN workstations, micro VAX II, and VAX/750 under UNIX.

A.16 VEDA

Category Specification/Front-end; Implementation/Compiler

Origin France

General Information

■ *Description:* VEDA [62] is a verification-oriented simulator. It uses the classical technique of random simulation runs. It has two components:

- analyser—parses an Estelle specification, checks the static semantics, and generates an internal representation of the specification.
- generator—inputs the internal representation and generates Pascal code to be used in the simulation.

■ *Language and Platforms:* VEDA is written in Prolog.

A.17 Xesar

Category Verification

Origin France

General Information

- *Description:* Xesar [100] verifies protocols written in Estelle/R. The specifications are expressed as properties given by temporal logic formulas.
- *FDT supported:* Estelle/R [95]—variant of Estelle where communication is modelled by rendez-vous.
- *Language and Platforms:* Xesar is written in C and runs on SM90 and SUN workstations under UNIX.
- *Applications:* [8] discusses the experience of describing an atomic multicast protocol (AMp) in Estelle/R and validating it using the Xesar tool. [99] contains a verification of the sliding window protocol.

B LOTOS based tools

B.1 ALDÉBARAN

Category Verification

Origin IMAG-LGI, BP53X, 38041 Grenoble Cedex, France

General Information

- *Description:* Aldébaran [44, 45] allows to compare and reduce labeled transition systems with respect to several equivalence relations (strong bisimulation, observational equivalence, acceptance model equivalence, and safety equivalence).
- *Applications:* [45] describes the verification results of LOTOS specifications for Milner's scheduler, Datalink protocol, and rel/REL_{fifo} protocol.

B.2 ASDE

Category Specification/Bookkeeping; Implementation/Translator

Origin Department of Telematics Engineering, Madrid University of Technology, ETSI Telecomunicación, Ciudad Universitaria, E-28040 Madrid, Spain

General Information

- *Description:* ASDE (Advanced System Design Environment) [75] supports two approaches to develop LOTOS specifications:
  1. Traditional—based on structured edition followed by a subsequent analysis.
  2. Transformational—based on formal development by applying correctness-preserving transformation rules. Using those rules, a specification can be refined in such a way that predefined external conditions are fulfilled. The transformation of LOTOS terms and the description of the entire process is performed by the `TransLotos` transformational language.

The environment has the following tools: two structured editors (one for LOTOS and the other one for `TransLotos`), a transformational kernel (to define, select and apply rules), catalogues of transformation rules, an expander of behavior expressions, a translator to Petri

Nets, and a development history controller. All of them are based on the Cornell Synthesizer Generator [112].

- *Language and Platforms:* ASDE is based on the Cornell Synthesizer Generator [112] and uses the SSL functional language. The development history tool is developed on top of SCCS, the standard UNIX history management tool.
- *Applications:* [75] describes a few simple examples using ASDE.

B.3 AUTO

Category Verification

Origin INRIA, Route des Lucioles, Sophia Antipolis, 06565 Valbonne Cedex, France

General Information

- *Description:* Auto [77] is a verification system for parallel and communication processes. It builds a network of communicating finite automata from the input program representing its behavior. The automaton can then be compared with others, reduced by some abstraction, or explored step by step. It uses a tool called AutoGraph to display its results.
- *FDT supported:* LOTOS. It also accepts programs written in CCS [79], SCCS [58], and Esterel [11].
- *Applications:* [15] describes a verification of a point-to-point sliding-window communication protocol with non-acknowledged messages.

B.4 CÆSAR

Category Implementation/Compiler

Origin VERILOG, France

General Information

- *Description:* CÆSAR [50, 51] accepts a subset of LOTOS process part as input, transforms it into extended Petri Nets, and then into state graphs. In the run-time phase, the state graphs are generated from the Petri Nets using reachability analysis. The state graphs can be verified by using either temporal logics or automata equivalences but verification is not supported in CÆSAR.

  CÆSAR.ADT [50] allows the LOTOS data part be statically compiled into data structures and functions in C. The code produced is deterministic in the sense that the application of rewrite rules is determined at compilation time. Unification and backtracking are not performed during execution.

  It is claimed that CÆSAR can be easily extended to deal with simulation, test generation and sequential code generation.
- *Language and Platforms:* CÆSAR is written in C and runs on SUN workstations.
- *Applications:* [50] describes the problem of data representation through concrete applications.

B.5 CENTAUR

Category Specification/Bookkeeping, Front-end; Implementation/Compiler, Simulator

Origin PTT Research Tele-Informatics, P.O. Box 15.000, 9700 CD Groningen, The Netherlands

General Information

■ *Description:* CENTAUR [14, 33] is a programming environment generator that has five components. All of them use a common representation, based on abstract syntax trees. The components are:

- Syntax-directed editors—derived from the syntax definition of LOTOS. CENTAUR has two editors: ctedit (CENTAUR editor) and GSE (Generic Syntax-directed Editor).
- Pretty-printer—prints an abstract syntax tree in text form according to layout rules.
- Type-checker—verifies the static semantics. The checking is performed by building an environment according to the definitions and then checking the specification with respect to the environment. Process, gate, and variable environments are defined similarly to an environment in a block structured language. Type environments are more complicated because of overloading and import mechanism in Act One. There are two formalisms to define semantics: TYPOL (logic based formalism for the manipulation of abstract syntax trees) and ASF (Algebraic Specification Formalism).
- Transformation tool [69]—transforms a LOTOS specification in plain text to another LOTOS specification (also in plain text) following some rules and keeping the original meaning. It can be used to transform the style in which the specification is written.
- Simulator—both semantics supported by the tool (TYPOL and ASF) can be used in the simulation of the specification.

B.6 COOPER

Category Testing/Active

Origin Tele-Informatics Group, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

General Information

■ *Description:* The COOPER tool [3] generates the canonical tester for a basic LOTOS specification, based on the CO-OP method [129], and using the Cornell Synthesizer Generator [112].

■ *Language and Platforms:* An implementation that supports canonical tester is described in [39]. COOPER is based on the Cornell Synthesizer Generator.

B.7 Data Type Compiler

Category Implementation/Compiler

Origin Technische Universität Berlin, Fachbereich Informatik (20), Institut für Software und Theoretische Informatik, Franklinstraße 28/29, FR 6-1, D-1000 Berlin 10, Germany

General Information

■ *Description:* The Data Type compiler [130] accepts the data part of a LOTOS specification as input and generates assembly code for a target machine. This process is achieved by performing the following steps: transformation of the data type specification into a canonical term rewriting system; generation of function definitions with case expressions for pattern matching, generation of a LATERM instruction sequence for each function definition; finally LATERM instructions are translated into assembly code (e.g., SUN or VAX workstations).

LATERM (LAzy TErm Rewriting Machine) is an abstract machine that allows efficient term rewriting and is based on implementation methods for functional languages.

- *Language and Platforms:* LATERM is written in C-Prolog.
- *Applications:* A very simple example is described in [130].

B.8 KDD LOTOS-C COMPILER/COMPILER

Category Implementation/Compiler

Origin Telecommunications Software Laboratory, KDD Kamifujuoka R&D Laboratories, 2-1-15 Ohara Kamijukuoka-shi, Saitama 356, Japan

General Information

- *Description:* The compiler [88] accepts a LOTOS specification as input and generates C code as output along with a process scheduler. The process scheduler executes a LOTOS process as a pseudo process. Each pseudo process has its own stack area, hardware registers, and a small part of the stack area, called segments. The compiler uses a structure to represent the parent-children process relationship to perform efficient multiway synchronization.
- *Language and Platforms:* The compiler is written in C and runs on VAX under VMS and UNIX.

B.9 LCRIS

Category Specification/Bookkeeping, Front-end; Implementation/Simulator

Origin CPqD Telebrás, Caixa Postal 1579, 13085 Campinas, SP, Brazil

General Information

- *Description:* LCRIS [76] has three components:
  - Editor—similar to a standard editor with a context sensitive help customized to LOTOS.
  - Syntax and static semantics checker—verifies the syntax and analysis of static semantics.
  - Simulator—interactive single step simulator. At each step, it is possible to choose an event to be executed, and to undo to previous behavior expressions.
- *Language and Platforms:* The tool has been implemented using Yacc and C on an IBM-PC.

B.10 LISP-BASED LOTOS ENVIRONMENT

Category Implementation/Simulator

Origin IBM Research Center, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

General Information

- *Description:* The kernel of the LISP-based LOTOS environment [70] uses the language *LL* (to represent internally LOTOS in the form of LISP S-expressions). The dynamic part of LOTOS can be represented directly in *LL* and the data part is defined in LISP.
- *Language and Platforms:* The kernel of the tool is based on the *LL* language.

B.11 LOLA

Category Implementation/Transformation

Origin Department of Telematics Engineering, Madrid University of Technology, ETSI Telecomunicacion, UPM, E-28040 Madrid, Spain

General Information

- *Description:* LOLA (LOTOS Laboratory) [92, 97] is a transformational tool used in validation and in design by stepwise refinement.
- *Applications:* [97] shows the transformations of a LOTOS specification of an alternating bit protocol. [92] describes the testing functionalities of LOLA.

B.12 LOTEST

Category Testing/Active

Origin University of Montréal, Canada

General Information

- *Description:* LOTEST (LOTOS test case generation tool) [2] accepts a specification in full LOTOS as input, transforms it to an equivalent extended finite state machine called *chart*, and then generates test cases. It contains the following three interactive tools:
  - cgtool—to view the *chart* as an FSM.
  - dfgtool—to view and identify the data flow part in the *chart*.
  - edittc—to view and edit the test cases generated.
- *Language and Platforms:* LOTEST runs on SUN workstations and is based on SunView.

B.13 LOTOS-PARLOG Translator

Category Implementation/Translator

Origin PARLOG Group, Imperial College, London SW7 2BZ, U.K.

General Information

- *Description:* The tool [52] translates a subset of LOTOS into the parallel logic programming language Parlog. The translator accepts full LOTOS syntax but only the dynamic part is translated from the parse tree to the PARLOG language [111].
- *Language and Platforms:* The translator is written in Parlog and runs under UNIX and requires the PARLOG SPM system [49].

B.14 LOTOS Simulator in OBJ

Category Implementation/Simulator

Origin Electrotechnical Laboratory, 1-1-4 Umezono, Tsukuba, Ibaraki 305, Japan

General Information

- *Description:* The simulator [90, 89] is an implementation of dynamic semantics of LOTOS.

Static semantics is obtained defining hierarchical module structures in OBJ.

- *Language and Platforms:* The tool is implemented in OBJ.
- *Applications:* [90] describes a few small examples.

B.15 LOTTE

Category Specification/Front-end; Implementation/Simulator

Origin Department of Applied Computer Science, PTT–Dr. Neher Laboratories, Leidschendam, The Netherlands

General Information

- *Description:* LOTTE (LOTOS Tool Environment) [121] provides a prototype environment for the integration of LOTOS tools. It contains a syntax and static semantics checker, an interactive report generator for gate sort lists analysis, and an interactive single step simulator (HIPPO from SEDOS Tools).
- *Language and Platforms:* LOTTE runs under UNIX.
- *Applications:* [114] describes the experience with LOTTE on an ISDN protocol.

B.16 PIL

Category Specification/Front-end; Implementation/Translator

Origin INRIA Rocquencourt, BP 105 Domaine de Voluceau, 78153 Le Chesnay Cedex, France

General Information

- *Description:* PIL (Pre-Implementation of LOTOS) [96] is based on a language called PRIMOL [83] that can be viewed as a hierarchy of cooperating extended finite state machines. LOTOS specifications are transformed into PRIMOL using PIL. The correspondent PRIMOL specification is an intermediate representation between the LOTOS specification and a real implementation. PIL has the following components:
  - LOTOS Analyzer—checks syntax and static semantics of the specification and generates an abstract tree corresponding to the specification. PIL analyzes mainly the process definition rather than the data part.
  - URL (Unique Reference LOTOS) translator—takes the abstract tree as input and generates a flattened abstract tree as output (all nesting of both processes and data type definitions are removed resulting in an one-level representation of the specification).
  - PRIMOL translator—translates a flattened abstract tree into PRIMOL. It is the last part of the pre-compilation.
  - PRIMOL-PRIMOL transformer—transforms a PRIMOL specification into another one.
- *Language and Platforms:* The tool is implemented in C on a SUN workstation.
- *Applications:* A few small examples are presented in [96].

B.17 SEDOS Tools

Category Specification/Bookkeeping, Front-end; Implementation/Compiler, Simulator

Origin ESPRIT/SEDOS Project, Europe

General Information

■ *Description:* The SPRIT/SEDOS project[1] developed the following tools forming an integrated environment [116]:

- MELO (MENTOR LOTOS Editor)—editor with a built-in LOTOS syntax, designed using the INRIA MENTOR system.
- SEAL (Structure Editor Adapted to LOTOS)—editor for LOTOS specifications with some advantages over MELO such as ease of tailoring and a more friendly user interface.
- SCLOTOS (Syntax Checker for LOTOS)—checks the LOTOS syntax. It is based on LEX and YACC.
- LASTB (LOTOS Abstract Syntax Tree Builder)—produces the abstract syntax tree of a specification from the output of SCLOTOS.
- LISA (LOTOS Integrated Static Analyzer)—checks the static semantics of a LOTOS specification using the output from LASTB.
- PPLOTOS (Pretty Printer for LOTOS)—displays abstract syntax trees. The output generated is a LOTOS text that may be displayed or filed.
- LXREF (LOTOS Cross Reference Generator)—produces a cross reference of identifiers in a LOTOS specification using the output of LASTB.
- HIPPO—takes the output of LISA and simulates the behavior of the specification, including calculating value expressions. HIPPO supports both the ACT ONE abstract data type rewriting and the simulation of the process part of LOTOS specifications. Both BELASI (BEhavioral LAnguage Simulator) and STAR+ (Adapted CSP Simulator) were superseded by HIPPO.
- LIW (LOTOS Implementation Workbench)—is an environment that supports the following functions:
  - an interactive computer aided source to source transformer which performs the translation of a high level specification to a more machine oriented one;
  - a simplifier that performs source to source transformation to simplify the input to the actual compiler;
  - an ADT compiler that generates a rewrite system to produce normal forms and an equality test;
  - a temporal ordering compiler that handles processes which are translated into co-routine like pieces of C-code that interact with a kernel, or run time support system, to implement the multiway synchronization semantics of LOTOS.
- LOTOSTOOL—window management system to supervise the execution of separate toolset components, reducing the complexity caused by the fragmented tool functionalities and interdependencies. LOTOSTOOL is based on SunView.

■ *Language and Platforms:* All tools described above run under UNIX.

■ *Applications:* Several applications using the tools developed in the SEDOS Project are described in [116].

[1]The SEDOS project involved the cooperation of eleven organizations in six countries: LAAS du CNRS (France), University of Twente (The Netherlands), ICL/STC (England), BULL (France), ADI, University of Catania (Italy), Politecnico of Milano (Italy), INRIA (France), HMI (Germany), Politecnico of Madrid (Spain), and Technical University of Berlin (Germany).

B.18 SMILE

Category Implementation/Simulator

Origin University of Twente, Faculty of Informatics, 7500 AE Enschede, The Netherlands

General Information

- *Description:* Smile [119] is a symbolic simulator in the sense that any event can be analyzed without instantiating the variables associated to that event. The input to the simulator is a common representation provided by a synthesizer generated editor [112].
- *Language and Platforms:* Most of the Smile internal structure and the common representation use the Termprocessor Kimwitu [118].
- *Applications:* A couple of small examples are described in [119].

B.19 SPIDER

Category Implementation/Simulator

Origin Hewlett-Packard Laboratories, Filton Road, Stoke gifford, Bristol BS12 6QZ, U.K.

General Information

- *Description:* SPIDER (Service and Protocol Interactive Development EnviRonment) [68] is a graphical simulator for LOTOS supporting both textual and graphical LOTOS.

B.20 SQUIGGLES

Category Verification

Origin CNUCE, CNR, Via S. Maria 36, 56100 Pisa, Italy

General Information

- *Description:* Squiggles [13, 12] verifies strong, weak, and testing equivalences between basic LOTOS specifications or finite labelled transition systems and generates the strongly connected components of an FTS.
- *Language and Platforms:* Squiggles is implemented in Prolog and C and runs on SUN workstation under UNIX.
- *Applications:* [13] describes some applications and performance of Squiggles.

B.21 TETRA

Category Testing/Passive

Origin Université de Montréal, Québec, Canada

General Information

- *Description:* TETRA (TEst and TRAce analysis tool) [127, 126] can automatically compare the specified verdicts of a conformance test case with the protocol specification, and analyse the results obtained from a test run according to the reference specification.
- *Applications:* [127] describes a validation of an X.25 TTCN test suite and for the testing of an ACSE implementation.

B.22 TIMED LOTOS INTERPRETER

Category Implementation/Interpreter

Origin Key Centre for Software Technology, University of Queensland, St. Lucia, 4067, Australia

General Information

- *Description:* The interpreter [46, 47] incorporates the notion of time-consuming actions according to the Real Time Logic formalism. (Actions are modelled as time-consuming, with distinct start and end points.) The purpose of the tool is to simulate the timing behavior of real-time systems.

  A LOTOS specification is entered as a LISP expression, converted to standard LOTOS concrete syntax, and then "executed." The interpreter is not a "trace generator" but can exercise all possible traces defined by the specification.

- *Language and Platforms:* The tool is implemented in Common LISP on a SUN 3 workstation.

- *Applications:* [47] discusses some small illustrative examples and a stop-and-wait protocol.

B.23 TOPO

Category Implementation/Compiler

Origin Department of Telematics Engineering, Madrid University of Technology, ETSI Telecomunicacion, UPM, E-28040 Madrid, Spain

General Information

- *Description:* Topo [34, 61] takes a LOTOS specification as input and generates an equivalent C program as output.

- *Language and Platforms:* Topo is implemented in C on a SUN workstation under UNIX.

- *Applications:* [34] relates the experience of specifying the SRTS (Simplified Reliable Transfer Service) into LOTOS and then using the topo compiler to generate C code.

B.24 UO-GLOTOS

Category Specification/Bookkeeping; Implementation/Translator, Simulator; Testing/Passive

Origin University of Ottawa, Department of Computer Science, Ottawa, Ontario, K1N 9B4 Canada

General Information

- *Description:* The graphical LOTOS project (UO-GLOTOS) is a system based on a set of simple and uniform graphical primitives (to represent LOTOS constructs) and has a hierarchical structure to represent the control flow of a LOTOS specification. It includes five components:
  - Editor [26]—interactive menu-driven editor.
  - Two translators [26] (from textual to graphical LOTOS and vice-versa)—interactive menu-driven translators.
  - Test sequence generator [25].
  - Executor [24]—is a semi-automatic graphical tool for dynamically tracing the logical flow of a specification. It also allows to generate and/or execute test sequences for a given specification.

During execution, it provides a global view of the control flow of the system and its current status. The Executor has four modes of operation: (a) single step; (b) multi-steps; (c) interruption by the environment or an indeterminable situation; and (d) interruption by the environment only.

The syntax supported by the UO-GLOTOS project is different from the ISO proposal [82].

■ *Language and Platforms:* The UO-GLOTOS system is implemented in C on a SUN 3/60 workstation under the SunView window system.

■ *Applications:* A formal specification in LOTOS of a telephone system (several hundred lines) [43] has been translated into two screen windows using the translator. The specification of Class 0 Transport Protocol has been translated to graphical LOTOS [26].

B.25 UO-LOTOS Toolkit

Category Implementation/Interpreter

Origin Department of Computer Science, Protocols Research Group, University of Ottawa, Ottawa, Ontario, K1N 6N5, Canada

General Information

■ *Description:* The toolkit [54, 55] receives a LOTOS specification as input, checks its static semantics, and executes the specification to produce the next possible actions as output. It has two operation modes:

1. Step-by-step (interpretation)—the user chooses the next action to be performed from the set of actions presented at that step.
2. Symbolic tree generation—a tree of all possible actions is generated. Variable values may be represented in terms of other variables.

■ *Language and Platforms:* The toolkit runs on UNIX 4.0 and later versions under SUN 3 and 4 workstations.

■ *Applications:* [53] describes a semi-formal methodology for deriving test cases from LOTOS specifications using the execution trees generated by the interpreter.

C SDL based tools

C.1 AT&T SDL Tools

Category Specification/Bookkeeping; Validation; Implementation/Translator, Simulator

Origin AT&T Bell Laboratories, 200 Park Plaza, P.O. Box 3050, Naperville, IL 60566, USA

General Information

■ *Description:* AT&T SDL development system [1] has a graphical editor, three translators (SDL-GR/PR, SDL-PR/GR, and SDL-PR/C), a report generator, a simulator for either a graphical or textual specification, and a protocol verifier which can detect race conditions and deadlock.

■ *Language and Platforms:* The tools run under UNIX. The simulator and graphical editor require an AT&T 630 or 5620 graphics terminal.

C.2 DANISH SDL-TOOL

Category Specification/Bookkeeping, Front-end; Implementation/Translator

Origin Telecom Research Laboratory, Bórups Allé, DK-2200 Copenhagen, Denmark

General Information

■ *Description:* The Danish SDL-Tool [66] consists of the following modules:

- Syntax checker—checks the syntax of the specification in text form and if the description is correct it builds an internal abstract syntax model (AS_0).
- Static semantic analyzer—transforms AS_0 into the recommended abstract syntax of SDL (AS_1) and reports semantic errors detected.
- Pretty-printer—transforms AS_0 representation into SDL/PR.
- Report generator—presents a specification in different forms (e.g., state overview diagrams, block trees, and cross indices) using the AS_0 representation.
- Graphical editor—creates an SDL/GR specification.
- Translators between both syntaxes.

■ *Language and Platforms:* The tool runs on PRIME or VAX computers and needs a graphical terminal (Tektronix or IBM/AT) with emulator.

■ *Applications:* [66] describes the experience of using this tool in some specifications.

C.3 DASON

Category Specification/Bookkeeping, Front-end; Implementation/Translator, Simulator

Origin A.S. Computas, P.O. Box 3765, Granaaslia, N-7001 Trondheim, Norway

General Information

■ *Description:* DASON [67, 122] supports the following functions: editing of different diagrams (e.g., sequence charts, block-interaction, and process diagrams) using a graphical syntax editor, project organization, document generation and treatment, model and document archives, syntax checking, version management, and program generation (translation to C code). Once a specification is created and checked using a syntax analyzer it is stored in a specification database.

■ *Language and Platforms:* DASON is based on the object paradigm and runs on all types of VAX computers from DEC.

C.4 ELVIS

Category Implementation/Bookkeeping, Front-end; Verification; Testing/Active

Origin CNET, Centre PAA, Division CLC, 38/40 Rue du General Leclerc, 92131 Issy Les Moulineaux, France

General Information

■ *Description:* ELVIS (french acronym for Edition and Interactive Validation of SDL Systems) [16, 18] has an editor (graphical and textual syntaxes), static semantic checker, a dynamic

semantic checker based on computation and analysis of a global state-transition graph (based on Xesar, an Estelle tool), and test case generation.

■ *Language and Plataforms:* ELVIS is built upon CONCERTO–a general purpose software engineering environment developed at CNET. ELVIS runs on SUN workstations under UNIX.

■ *Applications:* [16] shows an example of verification and test generation in ELVIS.

C.5 ESCORT

Category Specification/Bookkeeping; Validation

Origin KDD R&D Laboratories, 2-1-15 Ohara, Kamifukuoka-Shi, Saitama, Japan 356

General Information

■ *Description:* ESCORT (Environment for Specifying Communications Software Requirements) [1] is a development environment based on SDL. It has a graphical editor to sequence charts, a translator from sequence charts to SDL language, and a module to check syntactic properties.

■ *Language and Plataforms:* ESCORT runs on SUN workstations with NeWS window system.

C.6 FOREST

Category Testing/Active

Origin Information Systems and Electronics Development Laboratory, Mitsubishi Electric Corporation, 5-1-1 Ofuna Kamakura-Shi Kanagawa 247, Japan

General Information

■ *Description:* FOREST (FORmal Environment for Systematic Testing) [73] generates test cases in TTCN format from a protocol specification described in SDL and also generates test data from the protocol data definition described in ASN.1. Actually, FOREST is comprised of three tools:

- TENT [81, 102]—generates an internal representation of the SDL specification in table form, and test sequences.
- APRICOT [84]—generates test data.
- TESPEC—defines the user interface to describe test specification and test report.

■ *FDT supported:* SDL. In the paper [73] it was said that Estelle would be available with the same language model, and in the future, LOTOS with a different language model.

■ *Applications:* The CCR (Commitment, Concurrency and Recovery) protocol (application layer) has being specified in SDL and applied to the FOREST tool.

C.7 GEODE

Category Specification/Bookkeeping, Front-end; Implementation/Translator, Simulator

Origin VERILOG, 150, rue Nicolas Vauquelin, 31081 Toulouse Cedex, France

General Information

■ *Description:* GEODE [42] is an integrated environment that supports syntactic and graphic

edition, semantic verification, symbolic simulation, and translation to the C language.

- *Language and Plataforms:* GEODE runs under UNIX and VMS.

C.8 ISET

Category Specification/Bookkeeping, Front-end; Implementation/Translator

Origin Electronics and Telecommunications Research Institute, P.O. Box 8, Dae Dog Dan Ji, Chungnam, Korea

General Information

- *Description:* ISET (Integrated SDL Environment) [59] supports a graphical environment, static syntax checking, and translators (between the two syntaxes, to the CHILL language, to PIC–a language to typesetting graphics, and to a relational database).
- *Language and Plataforms:* ISET is based on window facilities.
- *Applications:* ISET has been used to design the TDX-10 switching system [59].

C.9 MELBA+

Category Specification/Bookkeeping, Front-end; Implementation/Translator

Origin Centre for Advanced Technology in Telecommunications, Royal Melbourne Institute of Technology, Melbourne, Australia

General Information

- *Description:* MELBA+ [21, 22, 23] supports a graphical syntax editor and translators (between the two syntaxes, and to either the CHILL or C languages). Both the syntax of the SDL specification and the implementations concepts are defined in extended BNF notation. The relationship between the specification syntax and the implementation language syntax is defined as a set of mappings.
- *Language and Plataforms:* MELBA+ is based on X windows.

C.10 SDL-Ada Translator

Category Implementation/Translator

Origin Telelogic AB, Baltzarsgatan 20, S-221 36 Malmö, Sweden

General Information

- *Description:* The tool [72] translates an SDL specification into Ada.

C.11 SDL based software development

Category Specification/Bookkeeping, Front-end; Implementation/Translator, Simulator

Origin OKI Electric Industry Co., Ltd., 11-22, Shibaura 4-chome, Minatoku-ku, Tokyo 108, Japan

General Information

- *Description:* The SDL-based software development [80] includes:

- SDL file manager—controls files (GR data, error files, common data, and sequence charts) used in the system.
- SDL/GR editor—supports the edition of the following diagrams in graphical form: system, block, substructure, process, service, procedure, and macro.
- SDL syntax checker—checks the syntax of an SDL/GR specification. It creates a common representation of the specification that is used by the other tools.
- SDL simulator—interprets the specification. It is possible to debug the specification by setting signals, data, and breakpoints.
- SDL code synthesizer—translates an SDL specification into C++. It has three parts:
  1. SDL/OS: set of classes for implementing SDL functions using OS primitives. This part is implemented in the TRON OS [101].
  2. Generated application: part that can be translated directly from the SDL specification.
  3. User provided: part dependent on hardware and other specific resources that must be provided by the user.

■ *Language and Plataforms:* The tool is implemented in LISP on a SUN workstation.

C.12 SDL-CHILL TRANSLATOR

Category Implementation/Translator

Origin Norwegian Telecommunications Administration, Research Department, N-2007 Kjeller, Norway

General Information

■ *Description:* The tool [57] accepts an SDL specification as input and generates a program in the Chill language.

■ *Language and Plataforms:* The translator is implemented in REFINE [98] which is based on the object paradigm.

■ *Applications:* A small example is shown in [57].

C.13 SDL TOOLS FROM LATVIAN STATE UNIVERSITY

Category Specification/Bookkeeping; Implementation/Compiler

Origin Computing Center of the Latvian State University, Blvd. Rainis 29, Riga 226250, Latvia (USSR)

General Information

■ *Description:* The following tools have been developed at the Latvian State University to support SDL specifications [9]: a graphical editor, a compiler (translates an SDL specification into a intermediate code called S and then to the Pascal language), and a debugger.

■ *Language and Plataforms:* The tool is implemented in RIGAL (a language developed at the Latvian State University) and runs on VAX under MVS.

C.14 SDL TOWER

Category Specification/Bookkeeping, Front-end

Origin TFL, LyngsøAlle 2, DK-2970 Hørsholm, Denmark

General Information

■ *Description:* The SDL tower [91] is part of the SPECS architecture an has four tools: editor (both graphical and textual syntaxes), an on-line help, a static analyzer (checks a specification for syntatic and semantic errors), and a database handler.

C.15 SDT-2

Category Specification/Bookkeeping, Front-end; Implementation/Simulator

Origin Telelogic AB, Baltzarsgatan 20, S-21136, Malmö, Sweden

General Information

■ *Description:* SDT [4, 5, 87, 113] is a CASE tool to develop real-time systems. It contains five tools (most of them can be used independently from each other). The tools are:

- Graphical editor—window based editor to create and maintain SDL specifications. It has an online syntax checker based on SDL syntax rules.
- Analyser and converter—performs (full) syntatic and (almost full) semantic analysis of a specification. It is also possible to analyze individual units. The converter transforms a specification in graphical form (SDL/GR) to plain text (SDL/PR).
- Simulator—has a simulator-generator that generates a Pascal program from the specification, and a kernel and monitor system that is the run-time kernel and user interface for the simulator. The user interacts with the simulator invoking commands of the monitor system.
- Report generator—generates reports from a specification according to the SDL hierarchy.
- Maintenance Functionality—supports facilities to define and delete "databases" (files created by the graphical editor), and to manipulate information inside and between "databases."

■ *Language and Plataforms:* SDT/2 runs/will run on workstations (SUN, VAX, and Apollo) using X windows.

C.16 SSI

Category Specification/Front-end; Implementation/Translator, Simulator

Origin Telelogic AB, Box 4148, S-203 12 Malmö, Sweden

General Information

■ *Description:* SSI [71] is a simulator tool that allows to examine the internal state of the system and the signal interface between the system and the environment. An "executable simulator" is obtained feeding the specification into the SDL analyzer and simulator generator module. A Pascal program is generated if the specification is syntactically and semantically correct. In that case the program is compiled and linked with some precompiled Pascal units (run-time kernel and simulator's user interface) to create an "executable simulator."

■ *Language and Plataforms:*

■ *Applications:* [71] shows an example of a simulation session.

C.17 TA-2

Category Validation

Origin Telesoft Europe AB, P.O. Box 4148, S-203 12 Malmö, Sweden

General Information

- *Description:* TA-2 [41, 40] validates SDL specifications by analysing the dynamic behavior. Some of the properties that are checked are deadlock, output with no receiver, and unbound queue overflow. The tool is based on reachability analysis through symbolic execution of a formalized description of the system. The future version of TA-2 is intended to be a complement of SDT [113].
- *Language and Plataforms:* TA-2 is implemented in Prolog and runs on a Macintosh.

C.18 TSDL-Tool

Category Performance evaluation and Validation

Origin Informatik IV, Universität Dortmund; Postfach 50 05 00, Germany 4600 Dortmund 50

General Information

- *Description:* TSDL (Timed SDL) [10] is an extension of SDL that can be used to describe timing aspects that are important to validation and performance evaluation of communication protocols. The tool accepts as inputs a textual description in TSDL (similar to SDL/PR) and a control file (contains inital values of global variables, and other parameters).

  Basically, the tool has five modules: Parser and State Generator (generates the internal representation of an equivalent FSM); three analyser modules (Probabilistic Validation and Performance Evaluation, Non-Exhaustive Validation, and Non-Exhaustive Performance Evaluation), and a Data Management Module.
- *Language and Plataforms:* The tool is implemented in C on a Sun 3/60 workstation. There is another program that runs under the Sun Tools to create the TSDL model and the control file. It can be used either interactively or in batch mode.
- *Applications:* [10] models the Positive Acknowledgement/Retransmission (PAR) protocol [110] using TSDL.

C.19 TESDL

Category Testing/Active

Origin University of Hamburg, Computer Science Department, Rothenbaumchaussee 67, D-2000, Hamburg 13, Germany

General Information

- *Description:* TESDL [17] accepts an SDL specification as input and generates automatically test cases.
- *Language and Plataforms:* TESDL is implemented in Modula-2 on an IBM-AT.
- *Applications:* [17] describes a few small examples.

C.20 YAST

Category Specification/Bookkeeping, Front-end; Implementation/Compiler

Origin Faculty of Electrical Engineering, Telecommunications Department, 41 000 Zagreb, Unska 3

General Information

- *Description:* YAST (Yet Another SDL Tool) [131] has the following parts:
  - Graphical syntax editors—support editing of process, service, procedure and macro diagrams, and description of system structures (system decomposition, block diagrams, service and channel decomposition).
  - Translators between both syntaxes.
  - Code generator for the PL163 and C languages. PL163 has real time extensions appropriate for parallel execution of processes.
  - On-line SDL tutorial based on HyperCard.
- *Language and Plataforms:* YAST runs on Macintosh under MultiFinder.

References

[1] Demonstration presented at SDL '89: The Language at Work. Proc. of the Fourth SDL Forum, 1989.

[2] Lotest: A LOTOS test case generation tool. Tool demonstration at the Third Int'l Workshop on Protocol Test Systems, 30 October–1 November 1990.

[3] Rudie Alderden. Cooper: The compositional construction of a canonical tester. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 13–18, Vancouver, B.C., Canada, 5–8 December 1989.

[4] Michael Atlevi. SDT "The SDL Design Tool". In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 55–59, Amsterdam, The Netherlands, 6–9 September 1988. North-Holland.

[5] Michael Atlevi. SDT: A real-time CASE tool for the CCITT specification language SDL. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 49–53, Vancouver, B.C., Canada, 5–8 December 1989.

[6] Jean Michel Ayache, Jean Dufau, Michel Huybrechts, and Eric Mattera. EWS: An integrated workstation for the design and the automatic generation of distributed software. In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 85–89, Amsterdam, The Netherlands, 6–9 September 1988. North-Holland.

[7] Pierre Azema et al. Estelle validation and Prolog interpreted Petri Nets. In M. Diaz et al., editors, *The Formal Description Technique Estelle–Results of the ESPRIT/SEDOS Project*, pages 273–302. North-Holland, Amsterdam, The Netherlands, 1989.

[8] M. Baptista, L. Rodrigues, P. Veríssimo, S. Graf, J.L. Richier, J. Voiron, and C. Rodriguez. Formal specification and verification of a network independent atomic multicast protocol. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 425–434, Madrid, Spain, 5–8 November 1990.

[9] J.M. Barzdin et al. SDL tools for rapid prototyping and testing. In O. Færgemand and M.M. Marques, editors, *SDL '89: The Language at Work–Proc. of the Fourth SDL Forum*, pages 127–133, Amsterdam, The Netherlands, 1989. North-Holland.

[10] Falko Bause and Peter Buchholz. Protocol analysis using a timed version of SDL. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 269–285, Madrid, Spain, 5–8 November 1990.

[11] G Berry, P. Couronné, and G. Gonthier. Synchronous programming of reactive systems: An introduction to Esterel. Rapport RR-647, INRIA, 1987.

[12] Tommaso Bolognesi and Maurizio Caneve. Squiggles: A tool for the analysis of LOTOS specifications. In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 201–216, Amsterdam, The Netherlands, 6–9 September 1988. North-Holland.

[13] Tommaso Bolognesi and Maurizio Caneve. Equivalence verification: Theory, algorithms and a tool. In P.H.J. van Eijk et al., editors, *The Formal Description Technique LOTOS–Results of the ESPRIT/SEDOS Project*, pages 303–326. North-Holland, Amsterdam, The Netherlands, 1989.

[14] P. Borras et al. CENTAUR: The system. In Ed Brinksma et al., editors, *Protocol Specification, Testing, and Verification, IX. Tutorial Notes*, Amsterdam, The Netherlands, 6–9 June 1989.

[15] G. Boudol, R. de Simone, and D. Vergamini. Experiment with Auto and AutoGraph on a single case of sliding window protocol. Rapport RR870, INRIA, 1988.

[16] Anne Bourguet-Rouger and Pierre Combes. Exhaustive validation and test generation in ELVIS. In O. Færgemand and M.M. Marques, editors, *SDL '89: The Language at Work–Proc. of the Fourth SDL Forum*, pages 231–245, Amsterdam, The Netherlands, 1989. North-Holland.

[17] Lars Brömstrup and Dieter Hogrefe. TESDL: Experience with generating test cases from SDL specifications. In O. Færgemand and M.M. Marques, editors, *SDL '89: The Language at Work–Proc. of the Fourth SDL Forum*, pages 267–279, Amsterdam, The Netherlands, 1989. North-Holland.

[18] Juan Camacho et al. ELVIS: An integrated SDL environment. In O. Færgemand and M.M. Marques, editors, *SDL '89: The Language at Work–Proc. of the Fourth SDL Forum*, pages 165–175, Amsterdam, The Netherlands, 1989. North-Holland.

[19] V. Chari, J.-F. Lenotre, L. Lumbroso, and E. Mariani. An Estelle simulator/debugger tool (EDB). In M. Diaz et al., editors, *The Formal Description Technique Estelle–Results of the ESPRIT/SEDOS Project*, pages 381–396. North-Holland, Amsterdam, The Netherlands, 1989.

[20] V. Chari, J.-F. Lenotre, and E. Mariani. The estelle translator. In M. Diaz et al., editors, *The Formal Description Technique Estelle–Results of the ESPRIT/SEDOS Project*, pages 325–351. North-Holland, Amsterdam, The Netherlands, 1989.

[21] Kong E. Cheng. An extensible approach to automatic program generation in telecommunications applications. Master's thesis, Department of Computer Science, Royal Melbourne Institute of Technology, Melbourne, Australia, January 1987.

[22] Kong E. Cheng and Lindsay N. Jackson. SDL translation report 1. RMIT Research and Development Memorandum 112064M, Department of Computer Science, Royal Melbourne Institute of Technology, Melbourne, Australia, August 1987.

[23] Kong E. Cheng and Lindsay N. Jackson. Automatic translation of SDL specifications to implementation based on syntatic transformation. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 542–561, Vancouver, B.C., Canada, 5–8 December 1989.

[24] To-Yat Cheung and Yucheng Ye. An executor for graphical LOTOS. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 667–670, Madrid, Spain, 5–8 November 1990.

[25] T.Y. Cheung, Y. Wu, and X. Ye. Generating test sequences and their degrees of indeterminism for distributed systems (with application to LOTOS). Technical Report TR-90-40, Department of Computer Science, University of Ottawa, Ottawa, Ontario, Canada, 1990.

[26] T.Y. Cheung, Y.C. Ye, X. Ye, and G.Q. Wang. UO-GLOTOS: A syntax/system for representing, editing and translating graphical LOTOS. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 33–49, Vancouver, B.C., Canada, 5–8 December 1989.

[27] A. Chung, D.P. Sidhu, and T.P. Blumer. Automated validation of protocols using EDS. In S. Aggarwal and K. Sannani, editors, *Protocol Specification, Testing, and Verification, VIII*, pages 351–360, Amsterdam, The Netherlands, 1988. North-Holland.

[28] A. Coen, A. Lombardo, and S. Palazzo. The EVA tool: An approach to verify structuring in Estelle specifications. In M. Diaz et al., editors, *The Formal Description Technique Estelle–Results of the ESPRIT/SEDOS Project*, pages 303–321. North-Holland, Amsterdam, The Netherlands, 1989.

[29] The SPECS Consortium and Jeroen Bruijning. Evaluation and integration of specification languages. *Computer Networks and ISDN*, 13(2):75–89, 1987.

[30] J.-P. Courtiat. Estelle*: A powerful dialect of Estelle for OSI protocol description. In S. Aggarwal and K. Sannani, editors, *Protocol Specification, Testing, and Verification, VIII*, pages 171–186, Amsterdam, The Netherlands, 7–10 June 1988. North-Holland.

[31] J.-P. Courtiat. A Petri Net based semantics for Estelle. In M. Diaz et al., editors, *The Formal Description Technique Estelle–Results of the ESPRIT/SEDOS Project*, pages 135–174. North-Holland, 1989.

[32] Jean-Pierre Courtiat. Introducing a rendez-vous mechanism in Estelle: Estelle*. In M. Diaz et al., editors, *The Formal Description Technique Estelle–Results of the ESPRIT/SEDOS Project*, pages 175–203. North-Holland, Amsterdam, The Netherlands, 1989.

[33] Paul de Jager, Willem Jonker, Albert Wammes, and Johan Wester. An interactive programming environment for LOTOS. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 713–716, Madrid, Spain, 5–8 November 1990.

[34] T. de Miguel, T. Robles, J. Salvachúa, and A. Azcorra. The SRTS experience: Using topo for LOTOS design and realization. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 487–498, Madrid, Spain, 5–8 November 1990.

[35] P. de Saqui-Sannes and J.-P. Courtiat. ESTIM: The Estelle simulator prototype of the ESPRIT-SEDOS project. In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 15–29, Amsterdam, The Netherlands, 5–8 December 1988. North-Holland.

[36] P. de Saqui-Sannes and J.-P. Courtiat. From the simulation to the verification of estelle* specifications. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 524–541, Vancouver, B.C., Canada, 5–8 December 1989.

[37] P. de Saqui-Sannes and J.-P. Courtiat. Rapid prototyping of an Estelle simulator: ESTIM. In M. Diaz et al., editors, *The Formal Description Technique Estelle–Results of the ESPRIT/SEDOS Project*, pages 353–379. North-Holland, Amsterdam, The Netherlands, 1989.

[38] ISO Technical Report DTR-10167. Guidelines for the use of Formal Description Techniques for OSI specifications, 1989.

[39] E.H. Eertink. The implementation of a test derivation algorithm. Master's thesis, University of Twente, Enschede, The Netherlands, 1987. Memorandum INF-87-36.

[40] Anders Ek and Jan Ellsberger. A prototype analysing dynamic properties of SDL. In *Proceedings 2nd Nordic Workshop on Program Correctness*, Aalborg, Denmark, 1990.

[41] Anders Ek and Jan Ellsberger. TA-2: A prototype analysing dynamic SDL properties. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 621–624, Madrid, Spain, 5–8 November 1990.

[42] Vincent Encontre. GEODE: An industrial environment for designing real time distributed systems in SDL. In O. Færgemand and M.M. Marques, editors, *SDL '89: The Language at Work–Proc. of the Fourth SDL Forum*, pages 105–115, Amsterdam, The Netherlands, 1989. North-Holland.

[43] M. Faci and L. Logrippo. Formal specifications of telephone systems in LOTOS. Technical Report TR-89-07, Department of Computer Science, Ottawa, Ontario, Canada, 1989.

[44] Jean-Claude Fernandez. Aldébaran: A tool for verification of communication processes. Technical Report SPEÇTRE C14, LGI-IMAG, Grenoble, France, 1989.

[45] Jean-Claude Fernandez and Laurent Mounier. Verifying bisimulations on the fly. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 91–105, Madrid, Spain, 5–8 November 1990.

[46] C.J. Fidge. A basic LOTOS interpreter. Technical Report 140, Key Centre for Software Technology, University of Queensland, St. Lucia, Australia, 1989.

[47] C.J. Fidge. A LOTOS interpreter for simulating real-time behaviour. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 625–638, Madrid, Spain, 5–8 November 1990.

[48] Behdad Forghani, Srinivas Eswara, Vassilios Koukoulidis, and Behçet Sarikaya. An Estelle based test generation tool for modular specifications. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 6–12, Vancouver, B.C., Canada, 5–8 December 1989.

[49] Ian Foster, Steve Gregory, Graem Ringwood, and Ken Satoh. A sequential implementation of Parlog. In *3rd Int'l Conf. on Logic Programming*, London, UK, March 1986. Dept. of Computing, Imperial College.

[50] Hubert Garavel. Compilation of LOTOS abstract data types. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 195–214, Vancouver, B.C., Canada, 5–8 December 1989.

[51] Hubert Garavel and Joseph Sifakis. Compilation and verification of LOTOS specifications. In L. Logrippo et al., editors, *Protocol Specification, Testing, and Verification, X*, pages 359–376, Ottawa, Canada, 12–15 June 1990.

[52] David Gilbert. A LOTOS to Parlog translator. In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 31–44, Amsterdam, The Netherlands, 6–9 September 1988. North-Holland.

[53] Djaffar Gueraichi and Luigi Logrippo. Derivation of test cases for LAPB from a LOTOS specification. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 489–508, Vancouver, B.C., Canada, 5–8 December 1989.

[54] R. Guillemot, M. Haj-Hussein, and L. Logrippo. Executing large LOTOS specifications. In S. Aggarwal and K. Sannani, editors, *Protocol Specification, Testing, and Verification, VIII*, pages 399–410, Amsterdam, The Netherlands, 7–10 June 1988. North-Holland.

[55] R. Guillemot and L. Logrippo. Derivation of useful execution trees from LOTOS by using an interpreter. In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 311–325, Stirling, Scotland, 6–9 September 1988. North-Holland.

[56] Martine Guilmet, Phillippe Thomas, and Bruno Traverson. Design, implementation and validation of a multi-peer protocol using Estelle. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 583–604, Madrid, Spain, 5–8 November 1990.

[57] Svein O. Hallsteinsen and Arne Venstad. Transformational program development: An approach for translating SDL to CHILL. In O. Færgemand and M.M. Marques, editors, *SDL '89: The Language at Work–Proc. of the Fourth SDL Forum*, pages 283–292, Amsterdam, The Netherlands, 1989. North-Holland.

[58] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, Cambridge, MA, USA, 1988.

[59] Jin P. Hong et al. Integrated SDL environment. In O. Færgemand and M.M. Marques, editors, *SDL '89: The Language at Work–Proc. of the Fourth SDL Forum*, pages 117–126, Amsterdam, The Netherlands, 1989. North-Holland.

[60] CERBO Informatique Inc. and Protocoles Standards de Communication Inc. Methods and tools for the design and validation of OSI protocol specifications and implementations, July 1986. Prepared for the Dept. of Communications, Canada.

[61] J.A. Mañas and T. de Miguel. From LOTOS to C. In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 79–84, Amsterdam, The Netherlands, 6–9 September 1988. North-Holland.

[62] C. Jard, R. Groz, and J.F. Monin. Development of VEDA: A prototyping tool for distributed algorithms. *IEEE Transactions on Software Engineering*, SE-14(3):339–352, March 1988.

[63] Claude Jard and Jean-Marc Jezequel. Outils pour l'expeérimentation d'algorithmes distribués sur machines paralléles. In *Actes du Colloque C^3 d'Angoulème.* GRECO C^3/CNRS, December 1988.

[64] Claude Jard and Jean-Marc Jezequel. A multi-processor Estelle-to-C compiler to prototype distributed algorithms on parallel machines. In Ed Brinksma et al., editors, *Protocol Specification, Testing, and Verification, IX*, Enschede, The Netherlands, 6–9 June 1989.

[65] Claude Jard and Jean-Marc Jezequel. Un compilateur Estelle multi-processeurs pour l'expérimentation d'algorithms distribués sur machines parallèles. Technical Report 453, IRISA University of Rennes, January 1989.

[66] Allan Jensen. The Danish SDL-tool. In R. Saracco and P.A.J. Tilanus, editors, *SDL '87: State of the Art and Future Trends–Proc. of the Third SDL Forum*, pages 177–186, Amsterdam, The Netherlands, 1987. North-Holland.

[67] Ulrik Johansen et al. Automatic program generation of SDL specifications: Principles and solutions. In R. Saracco and P.A.J. Tilanus, editors, *SDL '87: State of the Art and Future Trends–Proc. of the Third SDL Forum*, pages 349–359, Amsterdam, The Netherlands, 1987. North-Holland.

[68] Stuart G. Johnston. SPIDER: Service and Protocol Interactive Development Environment. In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 67–71, Amsterdam, The Netherlands, 6–9 September 1988. North-Holland.

[69] J.M.M. Joosten. A transformation tool for basic LOTOS developed with the interactive programming environment generator CENTAUR. Master's thesis, University of Twente, The Netherlands, 1990.

[70] Günter Karjoth. A LISP-based LOTOS environment. In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 73–77, Amsterdam, The Netherlands, 6–9 September 1988. North-Holland.

[71] Jan Karlsson and Anders Ek. SSI: An SDL simulation tool. In O. Færgemand and M.M. Marques, editors, *SDL '89: The Language at Work–Proc. of the Fourth SDL Forum*, pages 211–218, Amsterdam, The Netherlands, 1989. North-Holland.

[72] Jan Karlsson and Lennart Månsson. Using SDL as specification and design language and Ada as implementation language. In R. Saracco and P.A.J. Tilanus, editors, *SDL '87: State of the Art and Future Trends–Proc. of the Third SDL Forum*, pages 339–348, Amsterdam, The Netherlands, 1987. North-Holland.

[73] Kotaro Katsuyama, Tetsuo Nakakawaji, Fumiaki Sato, and Tadanori Mizuno. OSI testing environment based on standardized formalisms. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 339–348, Madrid, Spain, 5–8 November 1990.

[74] Pieter S. Kritzinger and Graham Wheeler. A protocol engineering workstation. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 63–76, Vancouver, B.C., Canada, 5–8 December 1989.

[75] G. León et al. ASDE: Design of a transformational environment for LOTOS. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 643–657, Vancouver, B.C., Canada, 5–8 December 1989.

[76] Humberto M. Lima and Gualberto Rabay. LCRIS: A LOTOS PC-based integrated environment. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 681–684, Madrid, Spain, 5–8 November 1990.

[77] Eric Madelaine and Didier Vergamini. AUTO: A verification tool for distributed systems using reduction of finite automata networks. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 77–84, Vancouver, B.C., Canada, 5–8 December 1989.

[78] William Majurski. Developing implementations of Estelle specifications using the PEDS toolkit. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 717–720, Madrid, Spain, 5–8 November 1990.

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

[80] K. Miyake, Y. Shigeta, W. Tanaka, and H. Hasegawa. Automatic code generation from SDL to C++ for an integrated software development support system. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 677–680, Madrid, Spain, 5–8 November 1990.

[81] T. Mizuno, J. Munemori, F. Sato, T. Nakakawaji, and K. Katsuyama. COTTAGE: Systematic method for the development of communication software. In S. Aggarwal and K. Sannani, editors, *Protocol Specification, Testing, and Verification, VIII*, pages 269–280, Amsterdam, The Netherlands, 7–10 June 1988. North-Holland.

[82] ISO/IEC JTC1/SC21 N3253. G-LOTOS: A graphical syntax for LOTOS, 1989.

[83] E. Najm, J. Queiroz, and A. Serhrouchni. The pre-implementation and verification of LOTOS. In *Proceedings of IFIP TC6 Int'l Conf. on Computer Networking*, Budapest, Hungary, May 1990.

[84] T. Nakakawaji, K. Katsuyama, N. Miyauchi, and T. Mizuno. Development and evaluation of APRICOT (tools for abstract notation one). In *Proceedings 2nd Int'l Symposium on Interoperable Information Systems*, pages 55–62, 1988.

[85] Darren New and Paul Amer. Protocol visualization of Estelle specifications. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 671–675, Madrid, Spain, 5–8 November 1990.

[86] Darren New and Paul D. Amer. Adding graphics and animation to Estelle. In Ed Brinksma et al., editors, *Protocol Specification, Testing, and Verification, IX*, Enschede, The Netherlands, 6–9 June 1989.

[87] Gert Nilson et al. SDL toolbox to support different SDL environments. In O. Færgemand and M.M. Marques, editors, *SDL '89: The Language at Work–Proc. of the Fourth SDL Forum*, pages 87–93, Amsterdam, The Netherlands, 1989. North-Holland.

[88] Shingo Nomura, Toru Hasegawa, and Takashi Takizuka. A LOTOS compiler and process synchronization manager. In L. Logrippo et al., editors, *Protocol Specification, Testing, and Verification, X*, pages 165–184, Ottawa, Canada, 12–15 June 1990.

[89] Kazuhito Ohmaki, Kokichi Futatsugi, and Koichi Takahashi. A basic LOTOS simulator in OBJ. In *Proc. of Int'l Conf. of Info Japan '90*, pages 497–504, October 1990.

[90] Kazuhito Ohmaki, Koichi Takahashi, and Kokichi Futatsugi. A LOTOS simulator in OBJ. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 653–656, Madrid, Spain, 5–8 November 1990.

[91] Anders Olsen. The SPECS SDL tower tools. In O. Færgemand and M.M. Marques, editors, *SDL '89: The Language at Work–Proc. of the Fourth SDL Forum*, pages 155–164, Amsterdam, The Netherlands, 1989. North-Holland.

[92] Santiago Pavón and Martin Llamas. The testing functionalities of LOLA. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 685–688, Madrid, Spain, 5–8 November 1990.

[93] Marc Phalippou and Roland Groz. Evaluation of an empirical approach for computer-aided test case generation. In *3rd Int'l Workshop on Protocol Test Systems*, McLean, VA, USA, 30 October–1 November, 1990.

[94] Marc Phalippou and Roland Groz. From Estelle specifications to industrial test suites, using an empirical approach. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 179–196, Madrid, Spain, 5–8 November 1990.

[95] RT 5 Cesar Project. *Xesar User's Manual.* LGI-IMAG, Grenoble, France, May 1987.

[96] J. Queiroz, A. Serhrouchni, P. Cunha, and E. Najm. PIL: A tool for pre-implementation of LOTOS. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 693–700, Madrid, Spain, 5–8 November 1990.

[97] Juan Quemada, Santiago Pavón, and Angel Fernández. Transforming LOTOS specifications with LOLA: The parameterized expansion. In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 45–54, Amsterdam, The Netherlands, 6–9 September 1988. North-Holland.

[98] Reasoning Systems Inc., Palo Alto, CA, USA. *REFINE User's Guide.*

[99] J.L. Richier et al. Verification in Xesar of the sliding window protocol. In H. Rudin and C.H. West, editors, *Protocol Specification, Testing, and Verification, VII*, pages 235–248, Amsterdam, The Netherlands, 1987. North-Holland.

[100] J.L. Richier et al. *XESAR: A Tool for Protocol Validation – User Manual.* Laboratoire de Génie Informatique, Grenoble, France, 1.2 edition, September 1987.

[101] K. Sakamura. *TRON Project 1987.* Springer-Verlag, 1987.

[102] F. Sato, K. Katsuyama, and T. Mizuno. TENT: Test sequence generation tool for communication systems. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 1–5, Vancouver, B.C., Canada, 5–8 December 1989.

[103] R. Sijelmassi. An object-oriented model for Estelle and its Smalltalk implementation. Technical Report NCSL/SNA 89/7, National Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, USA, February 1989.

[104] R. Sijelmassi. User guide for WISE: A simulation environment for Estelle. Technical Report NCSL/SNA 89/6, National Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, USA, February 1989.

[105] R. Sijelmassi and P. Gaudette. An object-oriented model for Estelle. In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 91–105, Stirling, Scotland, 6–9 September 1988. North-Holland.

[106] R. Sijelmassi and B. Strausser. NIST integrated tool set for Estelle. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 661–665, Madrid, Spain, 5–8 November 1990.

[107] B. Strausser. User guide for WIZARD: A syntax-directed editor and translator for Estelle. Technical Report NCSL/SNA 89/5, National Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, USA, February 1989.

[108] B. Strausser et al. Internals guide for the NBS prototype compiler for Estelle. Technical Report ICST/SNA 87/4, National Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, USA, September 1987.

[109] B. Strausser et al. User guide for the NBS prototype compiler for Estelle. Technical Report ICST/SNA 87/3, National Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, USA, October 1987.

[110] Andrew Tanenbaum. *Computer Networks.* Prentice-Hall, Englewood Cliffs, NJ, USA, second edition, 1988.

[111] Stephen Taylor. *Parallel Logic Programming Techniques.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1989.

[112] T. Teitelbaum and T.W. Reps. *The Synthesizer Generator: A System for Constructing Language-Based Editors.* Springer-Verlag, New York, NY, USA, 1989.

[113] TeleLOGIC. *SDT User's Manual.* TeleLOGIC AB.

[114] Paul A.J. Tilanus and Yan Yang. Experience with LOTOS and environment LOTTE on an ISDN protocol. In C. Rattray, editor, *Specification and Verification of Concurrent Systems*, pages 486–499. Springer-Verlag, 1988.

[115] Kenneth J. Turner. A LOTOS-based development strategy. In S.T. Vuong, editor, *FORTE '89, 2nd Int'l Conf. on Formal Description Techniques*, pages 157–174, Vancouver, B.C., Canada, 5–8 December 1989.

[116] P. van Eijk et al., editors. *The Formal Description Technique LOTOS–Results of the ESPRIT/SEDOS Project.* North-Holland, Amsterdam, The Netherlands, 1989.

[117] Peter van Eijk. LOTOS tools based on the Cornell Synthesizer Generator. In Ed Brinksma et al., editors, *Protocol Specification, Testing, and Verification, IX*, Enschede, The Netherlands, 6–9 June 1989.

[118] Peter van Eijk and Axel Belinfante. The Termprocessor Kimwitu. Technical Report INF-90-45, University of Twente, Enschede, The Netherlands, 1990.

[119] Peter van Eijk and Henk Eertink. Design of the LOTOSPHERE symbolic LOTOS simulator. In J. Quemada et al., editors, *FORTE '90, 3rd Int'l Conf. on Formal Description Techniques*, pages 709–712, Madrid, Spain, 5–8 November 1990.

[120] Peter van Eijk, Harro Kremer, and Marten van Sinderen. On the use of specification styles for automated protocol implementation form LOTOS to C. In L. Logrippo et al., editors, *Protocol Specification, Testing, and Verification, X*, pages 153–164, Ottawa, Canada, 12–15 June 1990.

[121] Wilfried H.P. van Hulzen. LOTTE: A LOTOS Tool Environment. In K.J. Turner, editor, *FORTE '88, 1st Int'l Conf. on Formal Description Techniques*, pages 61–65, Amsterdam, The Netherlands, 6–9 September 1988. North-Holland.

[122] Eirik Vefsnmo. DASOM: A SDL tool. In R. Saracco and P.A.J. Tilanus, editors, *SDL '87: State of the Art and Future Trends–Proc. of the Third SDL Forum*, pages 35–42, Amsterdam, The Netherlands, 1987. North-Holland.

[123] Chris A. Vissers. FDTs for open distributed systems: A restrospective and a prospective view. In edpstv90, editor, *Protocol Specification, Testing, and Verification, X*, Ottawa, Ontario, Canada, 1990. Invited Paper.

[124] Gregor von Bochmann. Specifications of a simplified transport protocol using different formal description techniques. Technical report, University of Montréal, 1987.

[125] Gregor von Bochmann. Usage of protocol development tools: The results of a survey. In Harry Rudin and Colin H. West, editors, *Protocol Specification, Testing, and Verification, VII*, pages 139–161, Ruschlikon, Switzerland, 5–8 May 1987. IFIP, North-Holland.

[126] Gregor von Bochmann and Omar B. Bellal. Test result analysis with respect to formal specifications. In J. de Meer et al., editors, *Second Int'l Workshop on Protocol Test Systems*, pages 103–117, Amsterdam, The Netherlands, 1989. North-Holland.

[127] Gregor von Bochmann et al. Test result analysis and validation of test verdicts. In *Third Int'l Workshop on Protocol Test Systems*, McLean, VA, USA, 30 October–1 November 1990.

[128] S.T. Vuong et al. UBC Estelle-C compiler (version 2.3): User's manual and internal guide. Technical Report TR 90-2, University of British Columbia, CICSR, Vancouver, B.C., Canada, April 1990.

[129] C.D. Wezeman. The CO-OP method for compositional derivation of canonical testers. In Ed Brinksma et al., editors, *Protocol Specification, Testing, and Verification, IX*, Amsterdam, The Netherlands, 6–9 June 1989. North-Holland.

[130] Dietmar Wolz and Paul Boehm. Compilation of LOTOS data type specifications. In Ed Brinksma et al., editors, *Protocol Specification, Testing, and Verification, IX*, Enschede, The Netherlands, 6–9 June 1989.

[131] Milan Zorić et al. Tool set development and the use of SDL. In O. Færgemand and M.M. Marques, editors, *SDL '89: The Language at Work–Proc. of the Fourth SDL Forum*, pages 77–86, Amsterdam, The Netherlands, 1989. North-Holland.