1998 International Computer Symposium
Workshop on Software Engineering and Database Systems
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

AN INTERACTIVE LOTOS TOOLSET FOR USERS IN INDUSTRY

Soosun Cho, Kwangyong Lee, Youngbae Oh, and Heung-Nam Kim

Software Engineering Department, ETRI – Computer & Software Technology Laboratory,
161 Kajong-dong, Yusong-gu, Taejon, 305-350, KOREA
Email:scho@etri.re.kr, kylee@cub.etri.re.kr, {yboh, hnkim}@etri.re.kr

ABSTRACT

This paper reports on an interactive toolset for editing and analyzing LOTOS specifications. It has been developed for the industrial use in telecommunication and information systems, particularly for the small-to-medium sized enterprises. In consideration of the poor environment of formal methods in Korea, it was designed to satisfy usual requirements for novice formal developers - usefulness and convenience. For this aim, easy-to-use graphical interactivity was implemented as the most remarkable feature of the toolset.

The toolset consists of two individual tools: a syntax-directed editor and a visual simulator for LOTOS. The syntax-directed editor provides template-based input and dynamic syntax checking. It serves user-friendly editing methods. The visual simulator is a tool to analyze a specification through step-by-step graphical interactions and eventually check whether it behaves correctly.

1. INTRODUCTION

Formal methods have long been discussed in the software engineering community. In general, it is suggested that using formal methods will result in highly reliable system. However up to date, formal methods have not been sufficiently applied in industry. Only recently have some results of its application been reported [10][15][18]. In paper [4], authors pointed out that for ideal use of formal methods, the researchers should strive to make their notations and tools accessible to nonexperts. Also, they emphasized that education is vital to the success of the formal methods.

In this paper we present a toolset for editing and analyzing LOTOS specifications. This toolset has been developed as a part of the project "TISDM: Telecommunication and Information Systems Development Methodology", which aims to advance techniques for developing and maintaining telecommunication and information systems providing a wide range of services on the information superhighway [12]. The results of the project are expected to be following:

- To provide a standard developing method for telecommunication and information systems
- To transfer technology to small-to-medium sized enterprises

As a part of TISDM project, the LOTOS toolset also has the objective to support small-to-medium sized enterprises for software quality improvement and productivity enhancement. We took LOTOS as a target specification language for telecommunication systems, and developed a LOTOS toolset for syntax-directed editing and visual simulation.

This project is the first attempt in Korea to develop a methodology based on formal methods, and distribute it to the industry. Thus, the toolset was designed for providing convenience and usefulness, in consideration of the poor environment of formal methods in Korea. Specially, among the many available features of specification support tools, we chose the easy-to-use graphical interactivity first of all.

The developed toolset is composed of a syntax-directed editor and a simulator, which are called SyDL, a Syntax-Directed editor for LOTOS and ViSL, a Visual Simulator for LOTOS. Because editing LOTOS specifications is not easy work for beginners, a user-friendly, syntax-directed editor is needed. Also, the step by step simulation is helpful to start applying LOTOS to the practitioners' development environment. Simulation is a rather informal validation method compared to model checking or theorem proving. However, it provides rapid replies to an analyzer. Because simulation is very useful during the initial debugging, it would be the most accessible activity for the non-experts. Furthermore, simulation works on the LOTOS specification only, without any reference implementation or test cases, thus, being simply applied.

As the results of TISDM project, the toolset will be contributed to novice formal developers of small-to-medium sized enterprises to use formal methods in telecommunication system development. We expect that it is also helpful to educate in formal language courses at universities.

The layout of this paper is as follows. Section 2 and 3 are meant to introduce LOTOS language and surveys the existing LOTOS support tools. In section 4 and 5, we discuss the structures and functions of SyDL and ViSL. And their implementations are discussed in section 6. Concluding remarks are given in section 7.

2. LOTOS

LOTOS (Language Of Temporal Ordering Specification) was designed as a very high level specification language for the abstract specification of communication systems.

Originally it has been developed specially for ISO-OSI (Open Systems Interconnection), for specifications of OSI protocols and services. But it is applicable to distributed, concurrent information processing systems in general [2][11]. LOTOS models reality by means of atomic events that are carefully ordered in time. A specification states the allowed sequences of events, without necessarily stating how these sequences are achieved.

In LOTOS, a system is seen as a set of processes that interact and exchange data with each other and with their environment [2]. The inter-process communication occurs by means of a "rendezvous" mechanism, called "interaction" or "synchronization". Process participates in a "rendezvous" if and only if they all offer an event at the same interaction point, called "gate".

One of the most important features of LOTOS is the fact that the dynamic semantics of a specification can be brought out by execution of it. Because LOTOS semantics are defined operationally, it is possible to implement these semantics in an interpreter, which for a behavior expression can enumerate the set of possible next actions and the behavior expressions resulting by the execution of each one of them [16]. In this paper, the execution of LOTOS behavior expressions, particularly, the user-friendly interaction for the execution is one of the main topics.

LOTOS has two main components: a "data" component that deals with the description of data structures and value expressions based on the formal theory of Abstract Data types ACT ONE [7], and a "control" component that describes the externally observable behavior of the system. This component is based on Milner's CCS (Calculus of Communicating Systems) [17]. If the data types are not defined, so can not be used on the process synchronization, it is called basic LOTOS. On the other hand, if the process synchronization involves the exchange of data values, it is called full LOTOS. Target of our toolset is full LOTOS. Therefore, the concerned process synchronization occurs with an action which is formed of three components: a gate, a list of event, and a optional predicate

3. EXISTING LOTOS SUPPORT TOOLS

Many LOTOS support tools are developed in LOTOSphere [13] to compose the design environment, LITE (Lotosphere Integrated Tool Environment). One of the LITE individual tools, CRIE [1] is a structured editor that incorporates the syntax and static semantics of the complete LOTOS language. The tool gives immediate feedback when a syntax or static semantics error is typed by the user. But currently, this tool is not being distributed, any more.

As a simulator, SMILE [6] allows a user to evaluate the LOTOS behavior symbolically. It contains functions for the analysis of the abstract data type part of the specification, execution and debugging the LOTOS specification, transformation of the specification into strong bisimulation equivalent EFSM (Extended Finite State Machine).

Another validation tool in LITE, LOLA [14] is a transformational and state exploration tool with application in testing, simulation, debugging. The transformation functionality allows the generation of the equivalent EFSM. The execution functionality can be used to simulate LOTOS specifications step by step.

Many other LOTOS tools also have been developed outside of the LOTOSphere project. The University of Ottawa LOTOS Toolkit [8] is an environment to analyze LOTOS specifications, based on the integration of several tools. ISLA [9] is a step by step executor. It also provides the service of checking syntax and static semantics of a LOTOS specification.

Most of the preceding tools were developed from researches at laboratories of universities, thus the user-friendly interactions or useful services were not considered. Because the human mind is strongly visually oriented and acquires information at a higher rate by relying on graphical relationships rather than by reading stream of text [20], it might be the most important feature of a tool to provide easy-to-use graphical interaction mechanism for LOTOS validations.

There have been some trials for visual approaches using LOTOS. One of them is G-LOTOS [3] a graphical syntax for the LOTOS, which is an ISO international standard also. G-LOTOS is intended to provide a better readability and more intuitive understanding of formal specifications than textual LOTOS. The individual tool of LITE, Glotos supports the editing of G-LOTOS. However the aim of our research is to provide the convenient editing method and easy-to-use graphical interaction mechanism for production of correct standard LOTOS, so no attempt was made for the graphical representation itself.

Another work for visual approach using LOTOS is SOLVE (Specification using an Object-based, LOTOS-defined, Visual language) [21]. The aim of SOLVE is to support effective specification and visual animation. For this aim, the SOLVE language and a set of tools that allow direct visual animation of systems specified in this language have been developed. It was designed to be used by people who are not familiar with formal languages.

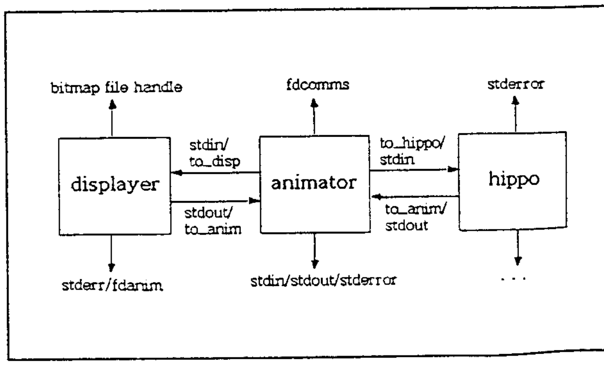


Fig. 1 Interfaces between SOLVE Animation Tools

The SOLVE tools run under the X window environment for animating specifications written in the SOLVE language. The interfaces of SOLVE animation tools are as Fig. 1. In the SOLVE animation tool, an early LOTOS simulator *hippo*, the predecessor of SMILE is used as simulation engine. When *animator* is invoked with a LOTOS specification file, it spawns *displayer* and *hippo* as child processes. *Animator* communicates via Unix pipes to/from the standard input/output of *displayer* and *hippo*.

The SOLVE tools inspired and shaped our work. Our LOTOS toolset also was developed for effective specification and simulation. It was designed to provide user-friendly, graphical interfaces for editing and simulating. Unlike SOLVE, our primary concern is to deal with LOTOS specification itself. For the effective specification, the template-based editing is proposed, and for the enhanced interface of simulation, the graphical views of execution of LOTOS behaviors are presented. Because any well designed graphical interface confirms comprehensibility and usability of a tool, we consider this aspect as the most important feature of the toolset.

4. SyDL

4.1 Overview of SyDL

SyDL is designed for users who are not familiar with structure and syntax of LOTOS language. In order to support production of correct specifications, SyDL allows the syntax-directed editing which is realized by direct input of syntax templates and dynamic syntax checking. Most of users not familiar with LOTOS usually make mistakes very often, because the greater part of the LOTOS-based work is editing of specification. Template-based input might be helpful to overcome such a problem. SyDL functions are as follows:

- Normal editing
- Direct input of templates by syntax trees
- Dynamic syntax checking for the partial or whole specification
- Key words coloring

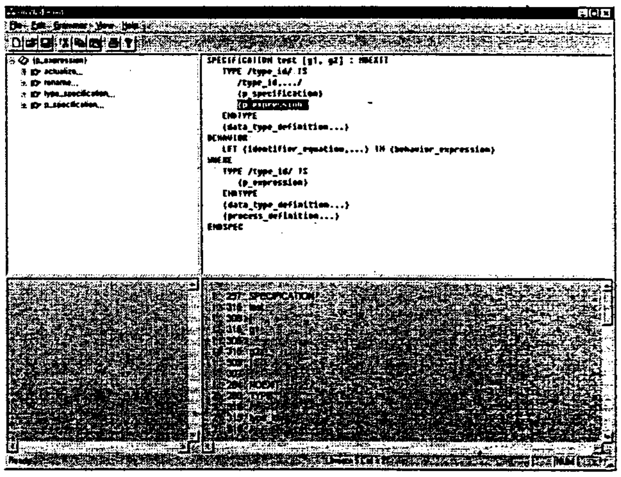
4.2 Functions of SyDL

The syntax-directed editor SyDL is composed of three parts, the edit window, the syntax tree window, and the log message window. Fig. 2 is showing the display of SyDL.

On the edit window placed on the right-up side, a user can type any LOTOS syntax like use a normal ASCII editor. He/she can also input directly a LOTOS syntax template via double-clicking a node on the syntax tree at the left-up window. In the displayed specification, words being blue-colored represent the reserved LOTOS symbols. The right-bottom log message window shows the results of lexical analysis and parsing a given specification.

Template-based editing

In editing, if a node of syntax tree is selected, then the corresponding syntax template is added to the cursor position at the edit window. Because the added syntax template usually includes non-terminal symbols enclosed with braces, a user may select one of these non-terminal symbols and the corresponding syntax tree is displayed again as shown in Fig. 2. The syntax tree has nodes that represent all possible extensions of the non-terminal symbol. It is waiting for another node selection.



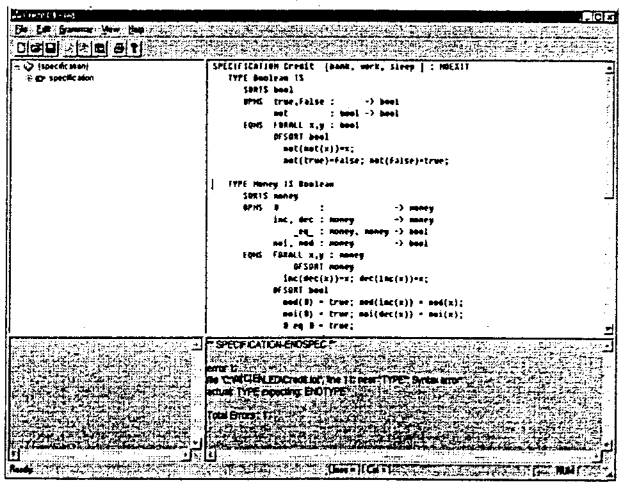
```
SPECIFICATION test [g1, g2] : NOEXIT
   TYPE /type_id/ IS
      /type_id..../
      {p_specification}
      {p_expression}
   ENDTYPE
   {data_type_definition...}
BEHAVIOR
   LET {identifier_equation,...} IN {behavior_expression}
WHERE
   TYPE /type_id/ IS
      {p_expression}
   ENDTYPE
   {data_type_definition...}
   {process_definition...}
ENDSPEC
```

Fig. 2 Display of Syntax-Directed Editor

By repeating of syntax template input, all of the non-terminal symbols can be removed. Then, the specification is completed with immediate typing on the slash-enclosed terminal symbols. The syntax templates represent the LOTOS syntax rules. It is generated from LOTOS syntax diagram [11, annex D].

Dynamic syntax checking

As soon as the data type definition parts, the library parts or the process parts are completed, the corresponding parsers work automatically to perform the partial syntax checking. And the results of syntax checking are revealed at log message window. Unlike static parsing of other tools, SyDL checks the syntax dynamically when all of the non-terminal symbols are removed in the interesting parts. The log message window of Fig. 3 shows the results of syntax checking a specification. For example, when the reserved symbol ENDTYPE is missed in the specification, error message is occurred as shown in Fig. 3.



```
SPECIFICATION Credit  [bank, work, sleep ] : NOEXIT
   TYPE Boolean IS
      SORTS bool
      OPNS  true,False :       -> bool
            not        : bool -> bool
      EQNS  FORALL x,y : bool
            OFSORT bool
              not(not(x))=x;
              not(true)=False; not(False)=true;

   TYPE Money IS Boolean
      SORTS money
      OPNS  0         :               -> money
            inc, dec : money         -> money
                _eq_ : money, money -> bool
            noi, nod : money         -> bool
      EQNS  FORALL x,y : money
                OFSORT money
              inc(dec(x))=x; dec(inc(x))=x;
            OFSORT bool
              nod(0) = true; nod(inc(x)) = nod(x);
              noi(0) = true; noi(dec(x)) = noi(x);
              0 eq 0 = true;
```

```
SPECIFICATION-ENDSPEC
error 1:
file "C:\HCHEN\LED\Credit.lot", line 11: near "TYPE": Syntax error
actual: TYPE expecting: ENDTYPE

Total Errors: 1
```

Fig. 3 Result of Dynamic Syntax Check

5. ViSL

5.1 Overview of ViSL

ViSL is a LOTOS simulator that allows step by step evaluation of specifications. Because the semantics of LOTOS specifications is defined by means of labeled transition systems, a simulator amounts to the generation of transition system from a given specification. A state in such a labeled transition system corresponds to a LOTOS behavior expression, and each transition is labeled an action that corresponds to at least one action denotation in the LOTOS expression.

Because the simulation of LOTOS specification can be accomplished by using simulation tools, the interactions between tools and users are one of the most important features. In this research, we propose graphic simulation trees and the user interfaces using these trees. ViSL functions are as follows:

- Moving to interesting behaviors in a specification
- Transition by selecting an action on the simulation tree
- Undoing the last transition
- Observing the trace of actions

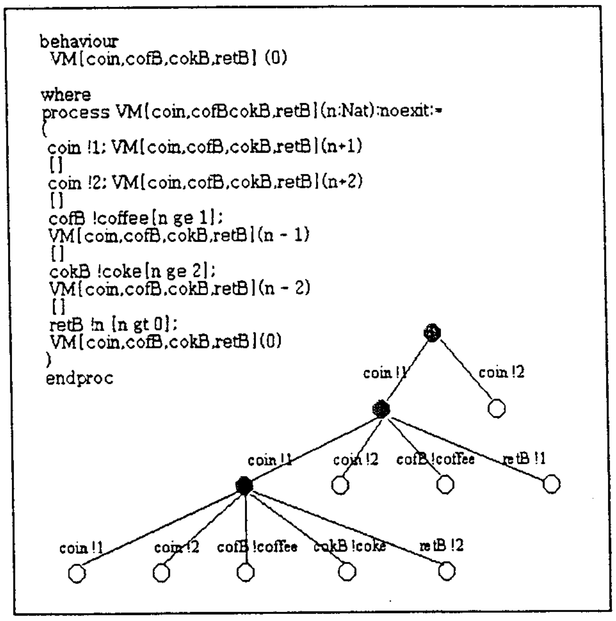
5.2 Simulation trees

One of the basic roles of a simulator is to calculate all possible actions at the current state. After an action is taken from possible ones by environment including a user, a transit from the current state to the next state occurs. The reached state is subsequently evaluated, and so the process continues. This results in a sequence of actions, or a trace. Usually, a simulator also allows a user to backtrack, and select another action for execution. In this way, a complete tree can be generated. This tree is usually referred to as a simulation tree [5, Chap.2].

The simulation tree is composed of nodes corresponding to states and edges corresponding to transitions. Fig. 4 represents LOTOS behavior specification and its simplified simulation tree. The specification models a vending machine that interacts with environments through gates *coin, cofB, cokB*, and *retB*. The data type definition part is omitted.

At the beginning simulation, because the value of *n* is zero, a user able to select one of two actions, "*coin !1*" or "*coin !2*". Like Fig. 4, if the selected action is "*coin !1*" then the number of capable actions are increased, "*coin !1*", "*coin !2*", "*cofB !coffee*" or "*retB !1*". The simulation tree represented at Fig. 4 is generated by selection of actions "*coin!1*" and "*coin!1*" again.

In this way, a user is able to analyze the dynamic semantics of specified behavior whether it satisfies the specifier's intention. Simulation is very rapid and intuitive validation method for users. And it allows a user to drive which parts of the LOTOS specification have contributed to a certain series of actions that are generated.



```
behaviour
 VM[coin,cofB,cokB,retB] (0)

where
process VM[coin,cofBcokB,retB](n:Nat):noexit:=
(
 coin !1; VM[coin,cofB,cokB,retB](n+1)
 []
 coin !2; VM[coin,cofB,cokB,retB](n+2)
 []
 cofB !coffee[n ge 1];
 VM[coin,cofB,cokB,retB](n - 1)
 []
 cokB !coke[n ge 2];
 VM[coin,cofB,cokB,retB](n - 2)
 []
 retB !n [n gt 0];
 VM[coin,cofB,cokB,retB](0)
)
endproc
```

Fig. 4 A LOTOS specification and its Simulation Tree

5.3 Functions of ViSL

The Fig. 5 is showing the display of ViSL. The left-back window includes the LOTOS specification that will be simulated, and the right-up window includes the current state represented LOTOS specification. The left-center window is the simulation dialog on which one can choose an action from the menu, undo transition, or observe the trace that represents history of transition. The right-bottom window displays the simulation tree in a graphical form. On this tree, one can do same things, choose an action, undo, and observe the trace. The results on the dialog coincide with the results on the simulation tree.

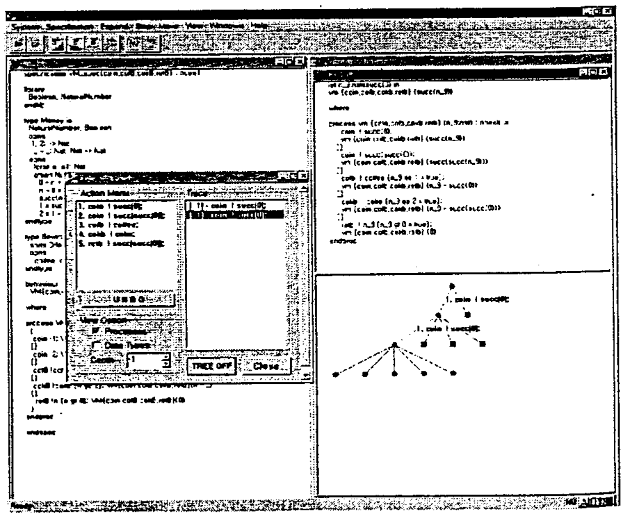


Fig. 5 Display of ViSL Simulator

Interactions using simulation trees

ViSL provides the visualized interactions for user interface. The graphical form of a simulation tree allows a user to analyze the LOTOS specification more intuitively and comfortably. The Fig. 6 represents a selection of an action and undoing it..When the mouse cursor is placed on a selectable node, the node's color is changed and the corresponding action name is displayed, then a user able to double-click to choose the node. Also, when the cursor is placed on a node which was already selected for the last transition, a user can choose it for undoing the transition. The second picture of Fig. 6 shows such situation. By using the simulation trees of ViSL, users are able to grasp the simulating situation at a glance, and know the current state very clearly.

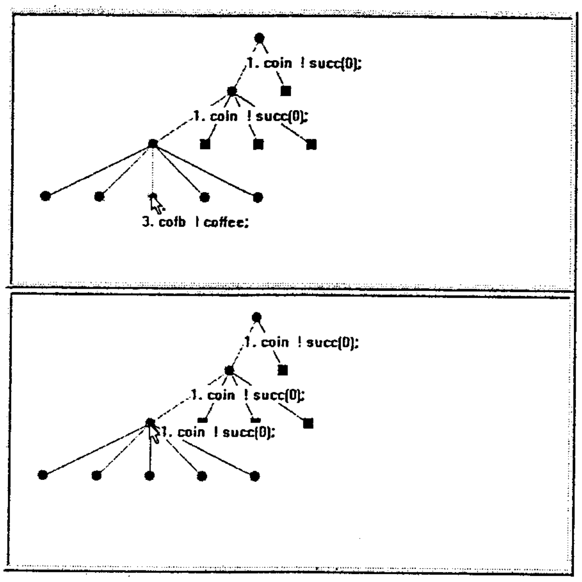


Fig. 6 Interactions using colored nodes of simulation trees

The nodes of simulation tree have one of three colors, red, green or blue. The red nodes are selected ones, which represent the trace linked by red lines. The action names on the red lines make the trace more readable. The blue color of leaf nodes means that they are selectable. When the cursor located on one of them, the blue color changes to green.

6. IMPLEMENTATION

The toolset executes on Windows95 or WindowsNT. Because the main objective of this toolset is to contribute to telecommunication system development area widely, it takes the most popular operating systems. Most of LOTOS toolsets introduced in section 3 are executed on UNIX environment. This fact makes it more difficult to distribute LOTOS-based development method to industry, particularly, to small-to-medium sized enterprises. We expect that the execution environment of our toolset might benefit such companies.

The toolset is developed on Windows95 with MS-Visual C++ 5.0 compiler. In the implementation of SyDL, the Abraxas PCYACC / PCLEX are used. For dynamic syntax checking, 3 partial parsers and a whole parser are developed with PCYACC and PCLEX.

6.1 Implementation of SyDL

In the implementation of SyDL, the syntax trees are represented by a fixed format and saved as a header file, rules.hpp. Fig. 7 shows the format and the examples. At the example of Fig. 7, the tree name is *{data_type_definition...}* and the two nodes' names are *type...* and *library....* The symbol '#' and '@\' mean that there is an indentation and a new line at the edit window, respectively. Because the syntax tree information is saved a header file, it is possible to change the contents of syntax trees very easily.

Format	Example
"non-terminal name@\ tree node name@\ code #1@\ code #2@\ ... ";	"{data_type_definition...}@\ type...@\ TYPE /type_id/ IS@\ #{p_expression}@\ ENDTYPE@\ {data_type_definition...}@\ "; "{data_type definition...}@\ library...@\ LIBRARY@\ #{type_id list}@\ ENDLIB@\ {data_type_definition...}@\ ";

Fig. 7 The format of a syntax tree

6.2 Implementation of ViSL

ViSL has three subsystems, *engine, manager*, and *GUI. GUI* has the role that interfaces to a user with easy-to-use graphical interaction mechanism, and *manager* interconnects *GUI* and *engine. Engine* provides some basic functions, which execute state transitions and yield a list of possible actions at each state transition. *Manager* calls *engine* functions with arguments, the user's input. And it returns the value from *engine* to *GUI.* With the returned values from *manager, GUI* generates simulation trees in a graphical form.

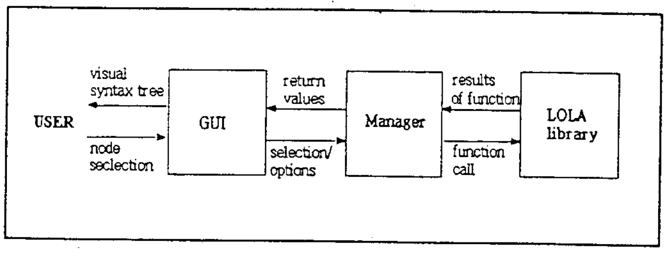


Fig. 8 The structure of ViSL

In the implementation of *engine*, a part of LOLA source code is reused. It is constructed to LOLA library. LOLA is a transformation tool, which provides LOTOS to LOTOS transformations. Also, it supports data evaluation, testing, and step by step simulation. In this research we select the part of step by step simulation and some substantial functions for recompiling LOLA library.

7. CONCLUSIONS

In this paper we introduced the tools, SyDL and ViSL. The aim of our research was to develop a practical LOTOS toolset and distribute it to industry as a part of an integrated methodology. For this aim, the basic functions of LOTOS-based toolset have been investigated and the syntax-directed editing and visual simulating were taken as our supporting functions.

Formal specification editors and simulators are the most fundamental tools that allow constructing a correct specification and analyzing dynamic semantics. Specially, for a beginner of formal languages, these tools must work very conveniently. SyDL, the template-based editor allows the user-friendly editing interfaces. And, ViSL provides the simulation tree in a graphical form, which makes enhanced interactions possible for the step-by-step simulation.

We developed the toolset to introduce the formal methods in industry for the first time, in Korea. The project is expected to last until October 1998 with technology transfer to small-to-medium sized enterprises. The participating companies in technology transfer will use the toolset as a part of the TISDM methodology. Also, we plan to research on using the LOTOS toolset for the development of telecommunication systems. It will be done as joint research. Our research team will cooperate with the practitioners in the participating companies.

We also expect that the toolset could be used in formal language courses. In research [19], it was shown how visualization and interaction can be integrated into a formal language course, using JFLAP tool as an example. The research emphasized that the formal languages course becomes a more traditional computer science course by integrating visual and interactive tools into the course, allowing the student to experiment with the picture and receive immediate feedback.

At present, the two tools are independent of each other. SyDL produces syntactically correct LOTOS specifications. And ViSL works with the flattened LOTOS specification which is passed through static semantics checker linked to LOLA. As the integration of these two tools is in progress, the gab will be narrowed in the near future.

8. REFERENCES

[1] A. Belinfante, editor, 'The LOTOS Integrated Editor Crie User and Reference Manual', LOTOSPHERE Consortium, Feb. 1992.

[2] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language LOTOS", Computer Networks and ISDN Systems, Vol. 14, No. 1, pp.25-59, 1987.

[3] T. Bolognesi, E. Najm, and P.A.J. Tilanus, "G-LOTOS: a graphical language for concurrent systems", Computer Networks and ISDN Systems, Vol.26, No.9, pp.1101-1127, May 1994.

[4] E. M. Clarke and J. M. Wing et al, "Formal Methods: State of the Art and Future Directions", ACM Computing Surveys, Vol. 28, No. 4, Dec. 1996.

[5] E. H. Eertink, editor, 'Simulation Techniques for the Validation of LOTOS Specification', Proefschrift, 1994.

[6] E. H. Eertink, 'SMILE user manual (release 4.0)', Tele-Informatics and Open Systems Group, 1993.

[7] H. Ehrig and B. Mahr, 'Fundamentals of Algebraic Specification 1: Equations and Initial Semantics', EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1985.

[8] B. Ghribi and L. Logrippo, "A Validation Environment for LOTOS", Protocol, Specification, Testing and Verification, XIII, Elsevier Science publishers B. V., pp.93-108, 1993.

[9] M. Haj-Hussein, "An Interactive System for LOTOS Applications", Master's Thesis, University of Ottawa, 1988.

[10] A. Hall, "Using Formal Methods to Develop an ATC Information System", IEEE Software, pp. 66-76, March, 1996.

[11] ISO, 'International Standard ISO8807', 1st Ed., 1989.

[12] J. Jeon et al., "A Study on the Information and Communication System Development Methodology (II)", Report, SERI, Korea, Dec., 1997.

[13] J. van de Lagemaat, T. Bolognesi and C. Vissers, editors. 'LOTOSphere: Software Development with LOTOS', Kluwer Academic Publishers, Dordrecht, The Natherlands, 1995.

[14] D. Larrabeiti , S. Pavon and G. Rayvay, 'LOLA user manual (version 3R6)', Department De Ingenieria Telematica, 1995.

[15] P. G. Larsen, J. Fitzgerald, and T. Brookes, "Applying Formal Specification in Industry", IEEE Software, pp. 48-56, May, 1996.

[16] L. Logrippo, M. Faci and M. Haj-Hussein, "An introduction to LOTOS: learning by examples", Computer Networks and ISDN Systems, Vol. 23, No. 5, pp.325-342, 1992.

[17] R. Miller, 'Communication and Concurrency', International Series in Computer Science, Prentice-Hall, 1989.

[18] S. L. Pfleeger, "Investigating the Influence of Formal Methods", Computer, pp.33-43, Feb. 1997.

[19] M. Procopiuc, O. Procopiuc, and S. H. Rodger. "Visualization and Interaction in the Computer Science Formal Languages Course with JFLAP", in Proc. Frontiers in Education '96, Salt Lake City, USA, Nov. 6-9, 1996.

[20] G. Raeder, "A survey of current graphical programming techniques", IEEE Computer, pp11-25, August 1985.

[21] K. J. Turner, and A. McClenaghan, "Visual Animation of Formal Requirements", technical report, Department of Computing Science, University of Stirling, UK, April 1996.