USE OF THE CCITT SDL

TO ENGINEER COMMUNICATION SOFTWARE

Timo Noko

Helsinki University of Technology
Telecommunication Switching Laboratory
Otakaari 5A
02150 Espoo 15
Finland

Abstract

The pictorial CCITT Specification and Description Language (SDL) is becoming a well-established tool in telecommunication programming. The program-like representation of SDL, known as SDL/PR, will certainly add the attraction of the language, because it allows machine-processing of the specification. In this paper the construction of an experimental SDL-design tool is introduced. SDL is aimed for specification purposes, but when the implemented software is based on state machine models, the correspondence between specification and implementation is obvious.

1. INTRODUCTION

Communication equipments are composed of a large number of units which are usually distributed or geographically separated. The interaction and cooperation of the units is achieved by the interchange of signals according to the agreed set of protocols. The only relative ordering of operations in the concurrent processing of the subsystems is achieved by means of the message exchanges. The use of state diagrams to describe computer logic circuits is well established among hardware designers. This paper describes our advancement in developing state diagrams as software tools for building multimicroprocessor-based communication equipment. The use of state diagrams is not restricted to system design, it is also used in program design, simulation, testing and documentation.

The CCITT Specification and Description Language (SDL) is a pictorial language which is similar to conventional flow diagrams but has special components to describe program logic in terms of state machine elements.

2. THE SDL-DESIGN CONCEPTS

The basic concept in using SDL-language is to describe the processing logic in terms of states. A state can be defined as a point of equilibrium where processing remains dormant until an event occurs. When an event occurs, the processing logic is said to be in transition. A state transition terminates when the processing logic enters the target state. Processing will remain dormant once again until another event occurs. During a transition, processing actions that are performed fall into one of three categories:

(1) read, write and process data,

(2) generate an external event for another state machine,

(3) generate an internal event e.g. set a time-out.

The use of monolithic model to describe the communication system as a single state machine is acceptable at system level, as long as the number of states remains small, otherwise a large system becomes difficult to manage in a single level. Despite this constraint, our goal is to document the program design in more detail

than at the system specification level. In order to specify all the micro steps of the processing logic the monolithic model could not be used. It therefore became necessary to structure our software as a number of state machines, with each state machine performing specific function. Such an approach alleviated the problem of maintaining an unmanageable number of states.

3. THE CCITT SPECIFICATION LANGUAGE

The CCITT Specification and Description Language (SDL) became a CCITT recommendation in 1976, and a refined version was published by CCITT in 1980. SDL has been developed in parallel with the other two CCITT languages; the high-level programming language CHILL and the man-machine language MML. During the development of the languages the desirability of a harmonized language family has been considered and recognized, i.e., the use of common concepts and terminology. This correlation between the languages has been obtained; for example, both CHILL and SDL contain a number of common concepts. The graphical representation is considered to be the version most easily understandable and usable for humans. However, there is also need for a "program-like" representation form, especially for computer storage and manipulations. The CCITT study group IX has recently issued a report of standardization of the SDL/PR-language (7), which is program-like representation form of the graphical SDL/GR-presentation.

The object in SDL/PR that is actually performing the logical function is called a PROCESS. Each process is regarded as a finite-state machine that can be represented by a closed directed graph. A BLOCK is defined to contain one or more processes.

The communication between processes is defined to be performed only via sending and receiving signals. An SDL-process is defined as an object that is either a state awaiting an INPUT (the reception of a signal) or is in transition. A STATE is defined as a condition in which the actions of a process are suspended and awaiting an input. A transition is defined to be a sequence of actions which occurs when a process changes from one state to another in response to an input.

In a transition there are three types of actions allowed:

* OUTPUT which is an action generating a signal which in turn will act as an INPUT elsewhere;

* DECISION which is an action asking a question, usually concerning the local database, to which the answer can be obtained at that instant and which chooses one of several paths to continue the transition;

* TASK which is an action within a transition which is neither a decision nor an output, and often results a change in the database.

In addition to this, there is one more concept defined within a process, that is, the SAVE. This concept has been defined in order to clarify the semantics of inter-process communication. It saves an incoming signal (without receiver), otherwise the signal is discarded.

One or more transition flow lines can converge. This is accomplished by the use of the keyword JOIN and the point of convergence is denoted by a label. A transition is otherwise terminated with a target-state symbol NEXTSTATE.

The keyword INTERNAL distinguishes internal inputs/outputs from external inputs/outputs (i.e. external being the default option).

An SDL graph consists of a number of graphical symbols, each representing one of the subconcepts of a process, connected by directed flowlines. The symbols used are shown in figure 1.

Figure 2 shows an example of SDL/PR-language. The corresponding SDL/GR-graph is shown in figure 3.

4. MACHINE REALIZATION OF STATE MACHINES

A state machine can be realized in a computing machine as a graph-driven parser which has the following concepts:

(1) a state transition graph representing transitions of a state machine,

(2) state execution routines performing actions after events causing state transitions.

Each node in the graph is a state. At each node there are a number of data items which represent the pre-defined events, routine entry arguments and the entry address of the action routine. The parser emulates the state machine by matching an input event with those at the resident node. If a match is found, the state execution routine (transition string) is entered with an approriate argument representing the event (i.e. external or internal input). After the execution of

the transition string, a value is returned to indicate which target state should be reached. If no match is found, the input event is discarded in case there is no active SAVE-operation concerning the event.

There are many advantages of this technique. Firstly, the program constructed in this manner is highly modular because the software is constructed in two levels. The overall logic flow is readily displayed in the SDL-graph. Detailed program logic can be followed by reading the program code of the transition strings. The other advantage is that any change in the design can be easily obtained by altering the graph. However the most subtle one is the reduction of software complexity, since the large number of event matchings are performed interpretively in a table-driven manner.

To support multiple existences of a state machine, a suitable indexing schema has to be availaible, a natural ordering of such existences can be achived e.g. from the hardware adresses (port or line numbers).

5. DESIGN AND DOCUMENTATION FACILITIES

The use of SDL/PR-language provides automatic generation of graphical SDL-documentation that is up-to-date, accurate and eliminates the need of separate documentation phases.

The diagrams produced can be used both as an aid during the test phase as a part of the final system design documentation. Figures 2 and 3 are examples of the actual transition description and its documentation.

The SDL/PR-translator in itself is programmable, to suit the specific needs and documentation standards of the user. The translation process is driven by description of the source language (the grammar of the language in Backus-Naur notation) and the description of the corresponding graphical items and layout of documentation. Different output devices are available. The use of lineprinter e.g. requires printing in line-by-line basis and use of the standard ASCII-character set.

At system level a simplified and less detailed documentation is often required. We have developed five levels of the simplification algorithm:

(1) Remove TASKs (i.e. modification of data),

(2) Remove data-driven branches (DECISIONs) with a common traget state and common output,

(3) Remove transitions with a common target state and common output,

(4) Omit OUTPUTs,

(5) Preserve states only.

Consistency checks assure the designer that the product has certain properties which the implemented system must also possess. In the design phase the product is only partially finished, therefore the consistency checks are made according to programmable rules:

(1) Check the context-free SDL/PR syntax only,

(2) Check that the target state and a target label for JOIN exists,

(3) Check that the received signal exists (is eventually available),

(4) Check that the receiver exist (signal is eventually received).

Preliminary studies of an interactive design tool whose input is in the form of drawings (with the aid of a ligth-pen) has been made. The graphical display represents a tiny "window" which can be scrolled to view or modify hidden parts of the net. The main difficulties in the design are the editing facilities and the amount of processing time when a portion of the large net is purged and the display has to be reconstructed.

6. SIMULATION AND VALIDATION FACILITIES

SDL/PR-language does not facilitate precise implementation of the data domain. But as mentioned above, our system supports a set of simplification facilities which effectively cancel the data by adding a certain amount nondeterminism. Any approach to system analysis must provide "multilevel support". The concept of "level" refers to the amount of detail provided in a system description. A common example is communication protocol layers, where one layer provides a service to a higher layer and uses the services of a lower layer; only the specification of the service, not the precise implementation details, are required by the layer above. The state-exploration method of validation is both time and space consuming. However, by introducing sufficient abstraction and restriction methods the overhead can be easily avoided.

By multilevel support we mean that such a specification must, in turn, be usable in the implementation of a higher level of system description in order to prove higher level properties about a larger system. The behaviour of the system is defined by the concurrent execution of the atomic actions of the state machines of the system. A particular execution path of such a system is a sequence of actions where that sequence is one possible interleaving of the actions. In order to verify concurrent system we must be able to show that all possible interleavings of the actions satisfy some correctness criteria.

Addition of timing information may reduce the amount of possible interleaving exhibited by the system, thus reducing the number of execution paths and the number of possible combinations of states produced by the system. Therefore, a system that cannot be verified in the absence of timing information may, in fact, be correct with the addition of timing information. Timing information is usually represented by a set of programmable timeouts.

The analysis of the otherwise error-free SDL/PR-language description is initiated with the construction of the abstract state machine models. In construction there are number of choices available in order to support the multi-level concept. The goal of the model are that it supports both explicit timing specifications and most importantly, that it supports hierarchical, multilevel verification. Hierarchical verification refers to the process of decomposing a proof into smaller subproofs and, conversely, composing a set of proved specifications into a proof of a larger system. The following key ideas can be identified with respect to the support of this capability.

* Ability to abstract a portion of a system -- Often it is most natural or efficient to prove things about a "portion" of the system being verified.

* Composition of abstractions -- Once it is proved that a subsystem implements some abstraction, that abstraction must be composable with other parts of the system so that further proofs can be performed on the composition.

7. THE DESIGN SYSTEM IMPLEMENTATION

The SDL-design system has been implemented in LISP-language. LISP-programming has many attractive features in constructing experimental and/or programmable systems. The development process is interactive and eases the production of well-defined and tested software modules. The uniformity of the program/data-domain (while possessing tempting features, such as the easy implementation self-altering programs) allows various tracing features to be implemented.

The structure of SDL/PR-compiler is divided into five phases:

(1) Translation of the SDL/PR-language into LISP-expression i.e. into a semantic net according to the programmed syntax rules.

(2) Simplification of the net as required.

(3) Restructuring the net until the most efficient topology has been found.

(4) Validation of the net as required.

(5) Translation of the net into corresponding graphical symbols.

(see Figure 4)

8. EPILOGUE

Our goal is not to produce a fully automated system but to create a useful tool to facilitate the production of communication equipment. This tool is used in the design and documentation phase of the multimicroprocessor data exchange (5). In the production of larger SPC-exchanges there is specific needs (e.g. company-oriented standards of documentation) which our system is designed to solve. We believe that most of the problems found in the SPC-programming is the "abstractness" of the design object. When there is a "real" object such as a diagram immediately available, the system is more comprehensible and the problem is easier to pinpoint.

ACKNOWLEDGEMENTS

Author wishes to thank prof. Kauko Rahko and his staff, from the Telecommunication Switching Laboratory, for their support and practical discussions.

The theoretical background is to be credited to a set of lectures by prof. Leo Ojala, from the Digital Systems Laboratory (5).

REFERENCES

1. Recommendations Z 101-104, CCITT Yellow Book, Int. Telecommun. Union, Geneva, Switzerland, 1982

2. "Functional Programming, Application and Implementation", by Peter Henderson, Prentice-Hall 1980.

3. "Functional Languages for The Data Processing Networks", by Timo Noko, Report 5/81 Helsinki University of Technology, Telecommunication Switching Laboratory.

4. "Protocol Representation with Finite-State Models", by Danthine A. A.S., IEEE Transactions on Communications, Vol 28, Num 4.

5. "Specification and Analysis of Communication in Distributed Data Processing Systems", by Jukka Oranen, Electricity in Finland 55, 1982.

6. "A Multimicroprocessor Controlled Data Exchange - A Gateway to Public Data Networks", by Kauko Rahko, Reijo Juvonen, Tarmo Hiltunen, Presented at 6th International Conference on Computer Communication, London, 7-10 September 1982.

7. CCITT Study Group XI - Contribution No. 133, March 1982

BIOGRAPHY

Mr. Timo Noko was born in 1952 and graduated in 1978 from the Helsinki University of Technology, where he is currently employed. He is presently collecting material for his licentiate thesis. His research interests include communication protocols, system verification, and applications of lazily evaluated, purely functional languages.

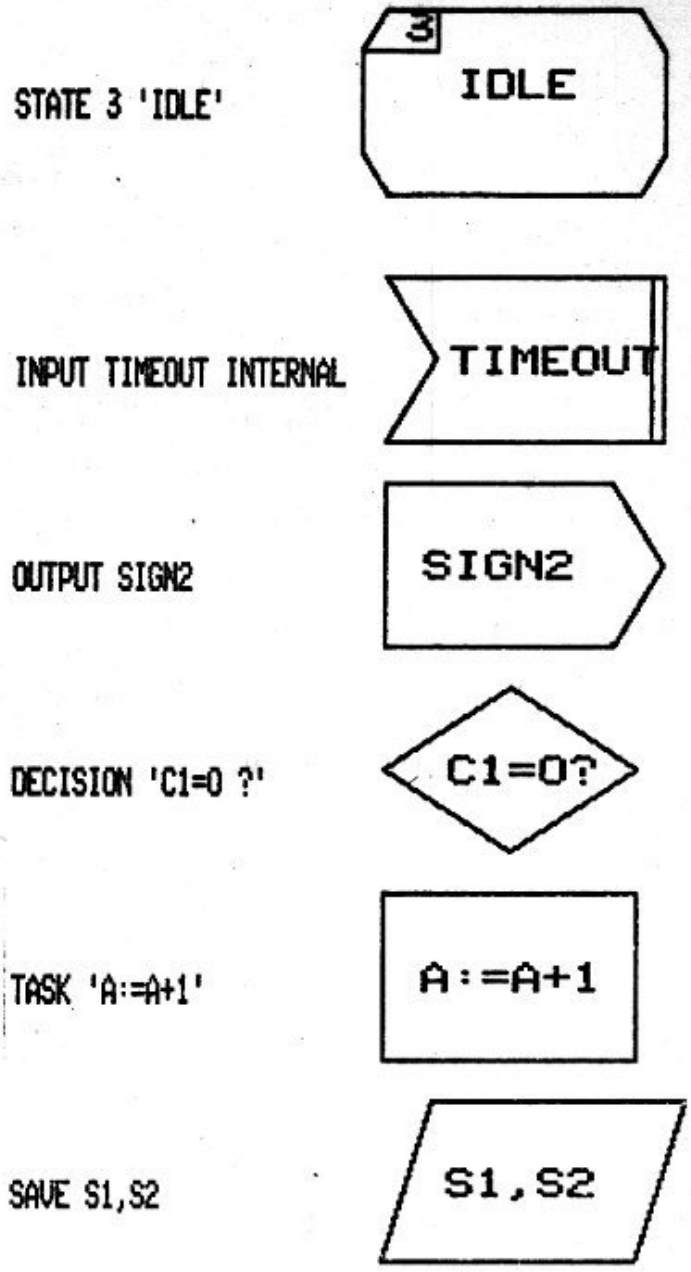

Fig. 1 SDL/PR and SDL/GR items

```
BLOCK MOD1;
 PROCESS TERMINAL_HANDLER;

  STATE 2 'WAIT/FOR/CHAR';

   INPUT 'CHAR' FROM USER;
    DECISION 'CHAR IS/CR,DEL/?'
         COMMENT 'ASCII/SET/ASSUMED';
      (CR): OUTPUT 'ECHO/CR';
            OUTPUT 'ECHO/LF';
            TASK 'END/THE/LINE';
            OUTPUT 'LINE/READY' TO LINE_HAND;
            OUTPUT 'SET/TIMEOUT/#2' INTERNAL;
            NEXTSTATE 3 'WAIT/FOR/ACK';
     (DEL):
            TASK 'LP:=LP-1' COMMENT 'REMOVE/CHAR';
            OUTPUT 'ECHO/BS';
            OUTPUT 'ECHO/SP';
            OUTPUT 'ECHO/BS';
            JOIN SETTO;
    (OTHERS):
            OUTPUT 'ECHO/CHAR';
            TASK 'CHAR/TO/LBUF';
            TASK 'LP:=LP+1';
         SETTO:
            OUTPUT 'SET/TIMEOUT/#1' INTERNAL;
            NEXTSTATE 2 'WAIT/FOR/CHAR';
     ENDDECISION;

   INPUT 'TIMEOUT/#1' INTERNAL;
         OUTPUT ' "LAZY/FINGERS" ' TO USER;
         OUTPUT 'DISCON' TO USER;
         OUTPUT 'BREAK' TO LINE_HAND;
         NEXTSTATE 1 'IDLE';

   INPUT 'BREAK' FROM LINE_HAND;
         OUTPUT 'DISCON' TO USER;
         NEXTSTATE 1 'IDLE';

   ENDSTATE 2;

  ENDPROCESS;
ENDBLOCK;
```

Fig. 2 SDL/PR-example

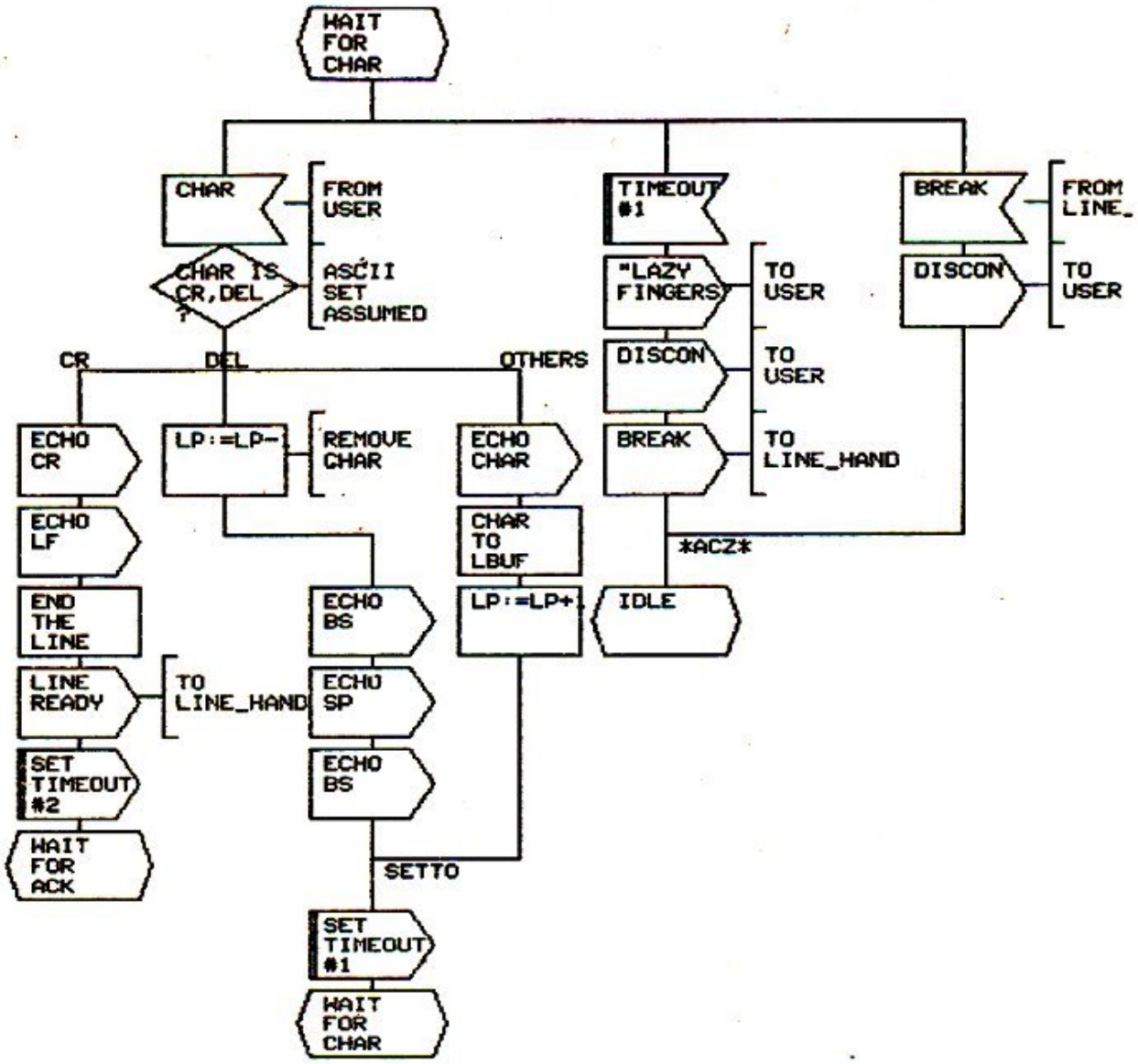

Fig. 3 SDL/GR-form of the previous figure

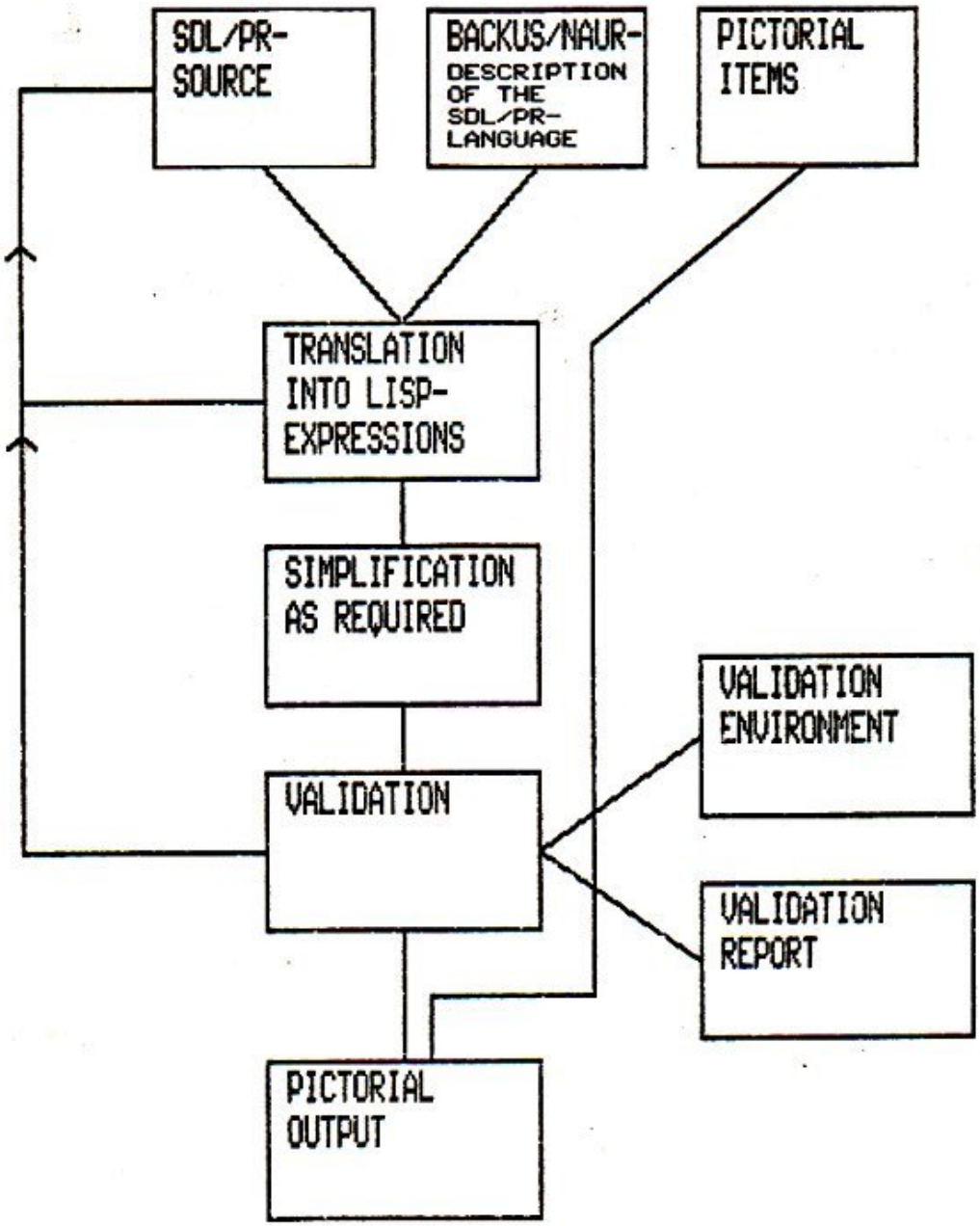

Fig. 4 Components of the SDL-tool

Sponsored By

UNIVERSITY OF HAWAII

UNIVERSITY OF SOUTHWESTERN LOUISIANA

1982

HICSS - 17

HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES

In Cooperation With

ASSOCIATION FOR COMPUTING MACHINERY

IEEE COMPUTER SOCIETY

Institute for the Advancement of Decision Support Systems

CONFERENCE CO-CHAIRMEN
RALPH H. SPRAGUE, JR.
Department of Decision Sciences
University of Hawaii
2404 Maile Way
Honolulu, HI 96822
(808) 948-7430

BRUCE D. SHRIVER
Computer Science Department
University of
Southwestern Louisiana
P. O. Box 44330
Lafayette, LA 70504
(318) 231-6606

TRACK CHAIRMAN
DECISION SUPPORT SYSTEMS
RALPH H. SPRAGUE, JR.

MEDICAL INFORMATION PROCESSING
THOMAS R. COUSINS
BRUCE D. SHRIVER
TERRY M. WALKER
University of
Southwestern Louisiana

HARDWARE
[illegible]
University of Michigan

SOFTWARE
LARRY WEISSMAN
SofTech, Inc.

OFFICE SYSTEMS AND TECHNOLOGY
RAYMOND R. PANKO
University of Hawaii

CONTRIBUTING SPONSOR COORDINATOR
RALPH R. GRAMS
University of Florida
Gainesville

PROGRAM ADVISORY BOARD
ERIC D. CARLSON
Convergent Technologies
WILLIAM E. REMUS
University of Hawaii
WILLIAM E. HOWDEN
University of California,
San Diego
HELMUT K. BERG
Honeywell Corporate
Computer Science Center

CONFERENCE COORDINATOR
EMILY M. YANO JORGENSEN
Center for Executive Development
University of Hawaii
2404 Maile Way C-202
Honolulu, HI 96822
(808) 948-7096
Cable: UNIHAW

Dr. Larry Weissman
SofTech, Inc.
460 Totten Pond Road
Waltham, MA 02154

Mr. Timo Noko
Helsinki University of Technology
Telecommunication Switching Laboratory
Otakaari 5A
02150 ESPOO
Finland

Dear Mr. Noko:

We are pleased to inform you that your paper, Use of the CCITT SDL to Engineer Communication Software, has been accepted for presentation at the Seventeenth Annual Hawaii International Conference on System Sciences. In the case of multiple authored papers, we are corresponding with one author only; will you please send a copy of this letter to any other co-author(s) of your paper.

Your paper will be presented in a 90-minute session with two other papers. Unless you receive other instructions from your session chairman, you should plan to limit your formal remarks to twenty minutes, allowing ten minutes for comments from a discussant and questions from the audience.

If specific comments on the paper were provided to me by the referees, they are enclosed. In some cases, referees comments will be sent to you by your session chairman.

The enclosed materials should be submitted in preparation for the conference: