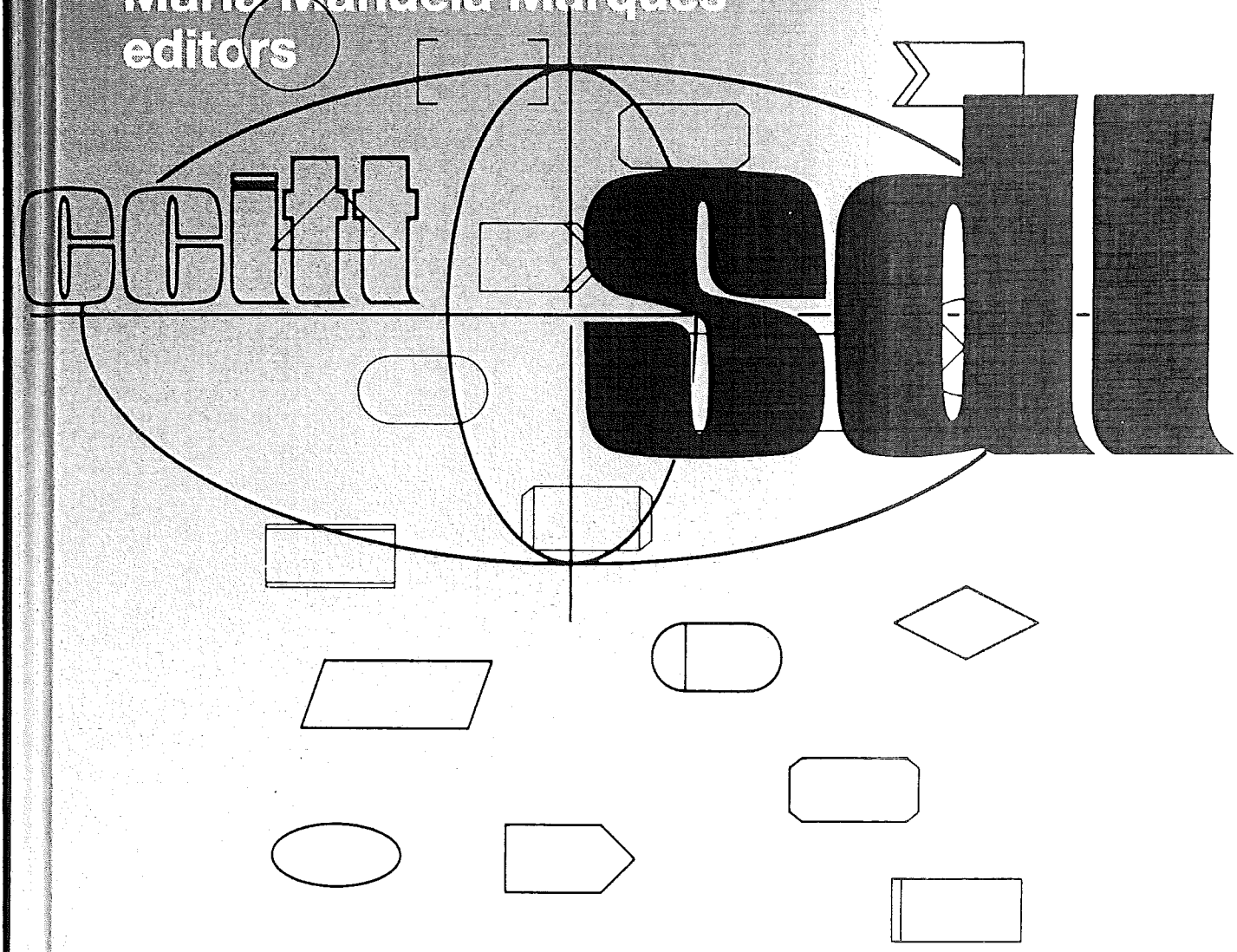
SDL '89
The Language at Work

Ove Færgemand
Maria Manuela Marques
editors



North-Holland

SDL '89: The Language at Work
O. Færgemand and M.M. Marques (Editors)

USING SDL IN SWITCHING SYSTEM DEVELOPMENT

Chan J. Chung, Jin P. Hong, Wan Choi, Han K. Kim, and Young K. Lee

Electronics and Telecommunications Research Institute,
P.O. Box 8, Dae Dog Danji
Chungnam, Republic of Korea

This paper presents a way and experiences using SDL(CCITT Specification and Description Language) in developing TDX-10 switching system. We exploit SDL to specify a functional behaviour of the system from the external point of view, to design software structure and behaviour, and to derive implementations. Some guidelines are necessary to use the language intellectually. It needs to be backed up with sound development tools for ease application[2]. Our experience has been showing that the use of SDL in the context of large switching system development has made marvelous contribution to both improvement of specification quality and enhancement of software productivity.

1. INTRODUCTION

Today's telecommunication systems of which distinguished characteristics are real-time, embedded, interactive, and distributed one are said to contain inherently some of the largest and most complex software ever constructed. The key to successfully develop such a system is to clearly specify "problems" and to transform them into a viable implementation. At the heart of this lies the choice of an appropriate formal language[2] to ensure clear specification of desired systems and to facilitate unambiguous communication among people involved in a project. The advantages of using a formal language are[13]:

- the human being is forced by the representation rules to fill in all the significant details required by the interpretation model. Shortly, it forces you to think
- the representation allows one and only one interpretation, independent of the reader
- the specification can be processed by computer tools to provide information on consistency, completeness, deadlock analysis, and simulation etc.
- the specification can be used as a prototype(can be executed)
- the specification can be automatically transformed into an equivalent program code.

TDX-10 is a large-scale digital switching system which has been being developed by ETRI(Electronics and Telecommunications Research Institute) since 1987 funded by KTA(Korea Telecommunication Authority) in cooperation with four domestic manufacturers[10]. Before initiating the project, we decided to use SDL and CHILL(CCITT High Level Language) in accordance with CCITT recommendations, and tried to set up the SDL/CHILL based development methodologies on the basis of experience with the previous TDX-1 switching system development project in which software engineering approaches are not applied adequately.

Adaptation of SDL in the TDX-10 development methodologies are intended to fix logical problems by formalizing the system's external behaviour prior to the beginning of system implementation, to describe physical model for realization, and to derive implementations.

Also, it is our another important policy to use SDL as a standardized vehicle to communicate among developers, because more than 500 system engineers and software engineers from six independent organizations in different places have been involved in the project.

2. TDX-10 DEVELOPMENT METHODOLOGIES

In general, the development of digital switching systems requires well defined methodologies in order to coordinate works of thousands of people due to the nature of their complexity and massive scale. The methodologies define the methods which are set of rules streamlining system development process and may be supported by various set of tools[8][12]. Work methodology is critically important for ensuring the successful development of any large and complex system, where typically continuous work is required to fix deficiencies in the initial design, to add new functions, to adapt for the new market fields, to modernize architecture of system in accordance with the technology advancement, or to upgrade user functions.

Figure 1 shows our development approaches from *user's requirements* to implementational *blocks*, and depicts their relationships[10].

In terms of TDX-10 development methodologies, concrete *requirements* are defined to be used as a written contract between customer(user) and developer, by investigation and inspection of *user requirements* which are usually ambiguous, incomplete, and inconsistent. During defining *requirements*, *conceptual model* that shows philosophy of basic structure and behaviour is defined to achieve conceptual integrity among various developers[12]. *Requirements* definition document specifies everything that the end-users do see, including *features* (functions from the user's point of view), external-interfaces in the system environment, input/output operational messages from/to external-interfaces, and non-functional constraints such as time limits.

Functions are formally redescribed from *requirements* in accordance with what the system to do rather than how they are to be implemented. A formalized operational *function* specifications can show the external behaviour of required system and allow us to predict how the system will behave, and to verify whether the system behaviour will be acceptable in early stage of system development[12].

A *functions* is subject to transformed into one or more *blocks* which are the basic development unit in design and realization. The concept of the transformation is from "What(problem domain-oriented)" into "How(implementation domain-oriented)". A *block* is realized purely with software only, hardware only, or both software and hardware. The interfaces between *blocks* are defined as *messages*. *Subsystems* are formed by grouping *blocks*. After software *blocks* are designed and implemented, a basic test is performed on the simulated and/or real environments for each *block*.

Having been incrementally integrated by adding the individually tested *blocks* the (part of) system is verified on the real environments whether a certain *function* meets its specification. After all *functions* are tested, the entire system is verified and validated if the operation of the system meets the *requirements* for the field trial and commercial operation.

3. THE USE OF SDL IN TDX-10 DEVELOPMENT

Currently, SDL is used to specify the behavior of *functions* and to design the internal structure and behavior of *blocks*. *Block* descriptions can also be used to derive implementations.

3.1. I

As sh
specifi
cess d
the en
system
Usuall
which
are sp

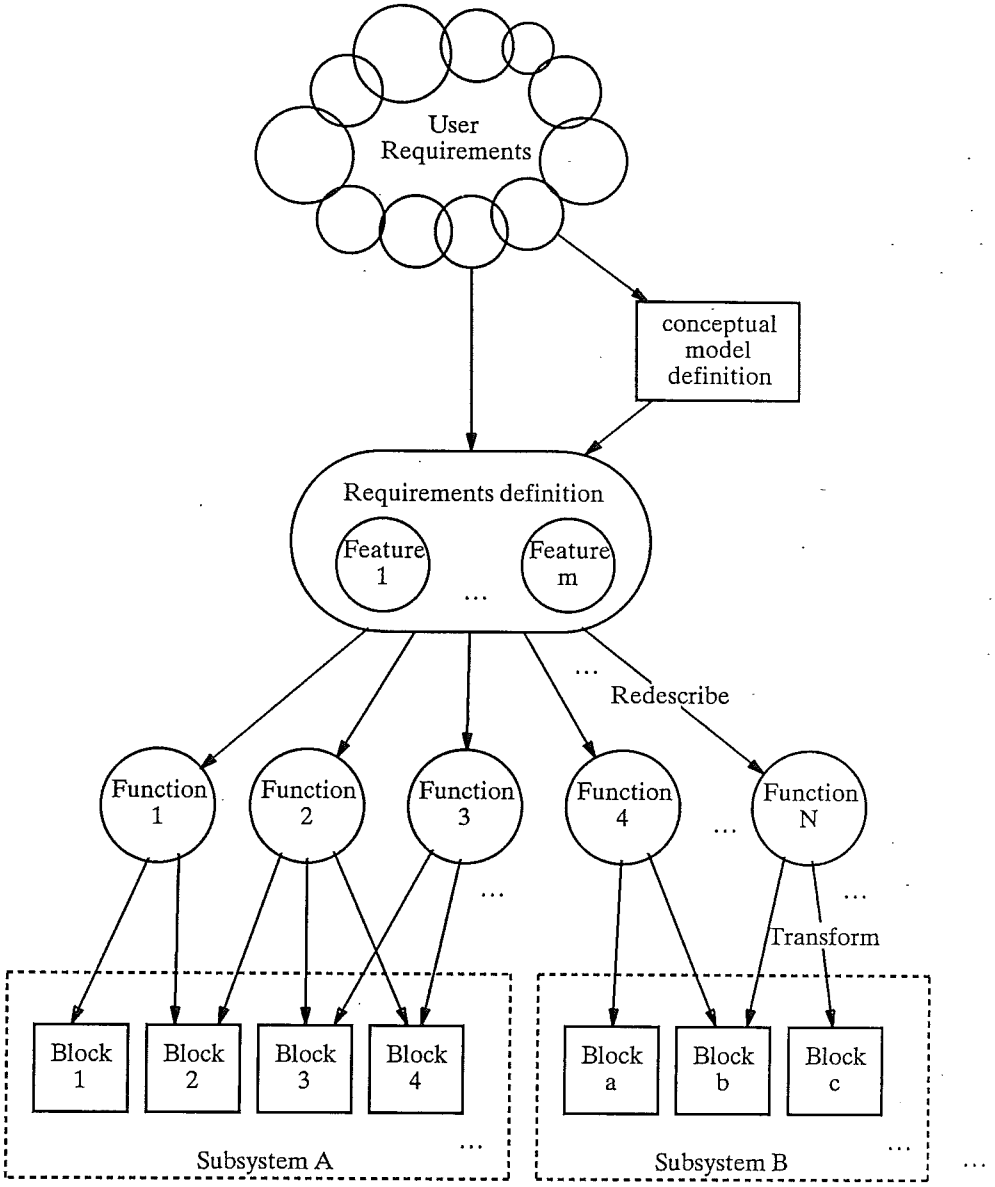


Figure 1. The Development Approach

3.1. Functional Specification

As shown in Figure 2 each *function* modeled as an EFSM(Extended Finite State Machine) is specified in the form of SDL system and process diagram, as if the system had only the process definitions for the *function*. The system diagram defines processes, external-interfaces in the environment, and signals to/from processes. The process diagram mainly shows how the system-level states are changed and what responses are made, according to external stimuli. Usually, the *function* concerning multi-user can be specified by two process definitions of which the number of process instances is (1, 1) and (0, n). No interaction between *functions* are specified.

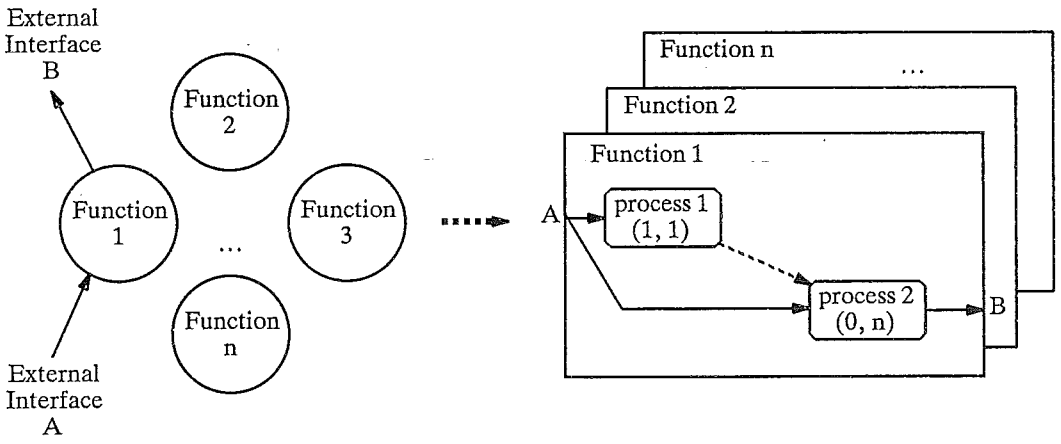


Figure 2. The concept of *Function* specification

3.2. Software Design

Functions are transformed into *blocks* by maximizing the cohesion as well as by minimizing the coupling between them, and by considering the manageable size and traceability. Derived *blocks* from *functions* are equivalent to SDL leaf-blocks implemented by a set of EFSM working autonomously and concurrently with other EFSM[18]. We use the following approach to define *blocks*.

(1) Define candidate software *blocks* from *function* as functional as possible, considering the physical processor boundary, minimization of interfaces, and balanced size, etc.

(2) Define *messages* between *blocks* and DB relations(tuples) accessed by *blocks*. *Message* are classified into software (CHILL) signal, CHILL buffer, CHILL timer signal, or hardware-software interwork implemented by memory mapped I/O.

(3) Analyze *function* to *block* transformation according to the following rules. This analysis can be done using automated tools that will be discussed later:

- no more than 9 *message* channels for each *block* are allowed
  (Research on human perception and recall shows that more than 9 elements will overload most readers[17])
- virgin-birth(only incoming channel) *block* not allowed
- black-hole(only outgoing channel) *block* not allowed
- no common data location except CHILL buffer between *blocks* are allowed
- uniform *block* size is recommended.
- etc.

If not satisfactory, go to (1) to refine *block* transformation.

(4) Draw sequence charts showing normal sequence of signals interchanged between one or more *blocks* and their environment per *function*. Sequence charts are usually used as a starting point for the drawing of the SDL process diagram[5].
Define *subsystem* by grouping *blocks*.

(5) The following aspects of message surveys are made:

- system - *subsystems* - *blocks*
- system - processors - *blocks*
- *functions* - *blocks*

After *blocks* are defined, they are designed in the form of SDL block diagram and control flow diagram by referencing sequence charts and *message* surveys. This process diagram can be used to output a CHILL program skeleton, using the SDL to CHILL transformer (*sdl2ch*)[3][4].

4. SDL USAGE RULES

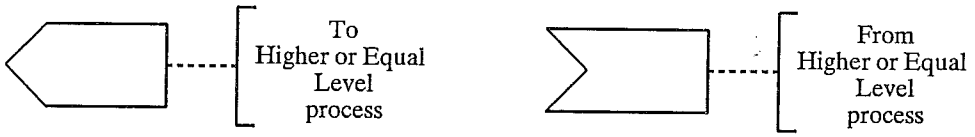
We have tried to set up SDL subset rules, based on the SDL'88 recommendations [15][18]. SDL/GR and common textual grammar are used. State oriented representation and pictorial elements are not used. The followings are not (yet) employed SDL features:

- Service
- Signal Refinement
- Block Substructure
- Channel Substructure
- Import/Export
- View/Reveal
- etc.

4.1. Additional SDL/GR Drawing Rules

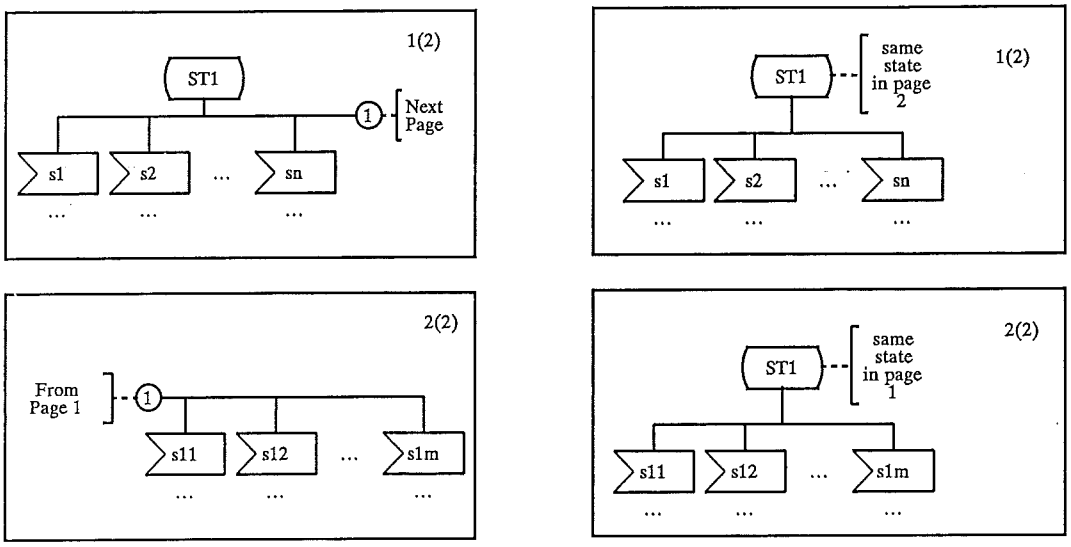
To improve the understandability of SDL/GR the following rules are laid down:

- signals to/from processes of higher or equal hierarchy have to be drawn as followings[1][5]



- signals to/from Processes of lower hierarchy have to be drawn as followings[1][5]

- when there are too many signal input symbols under a state to draw on a single page, it is recommended to follow the drawing rules as shown in Figure 3
- a same state symbol on different pages should have page references (see Figure 3)
- every process, procedure, and macro definition begins on a new page for easier reference[1]
- keep the mean flow straight[1]
- place the asterisk state on the first page of the process diagram
- every in/out connector from/to another page must have a page reference
- place the symbols under a state in the order of signal input, save, continuous signal.



(a) Not Recommended Style

(b) Recommended Style

Figure 3. The way of drawing many signal inputs under a state

4.2. Additional SDL Rules for Software Implementation

The following additional rules are laid down, for easy transformation from SDL to ETRI/CHILL under TDX-10 operating system, CROS(Concurrent Realtime Operating System)[6][7][9][11]:

- all data (definitions) are made in CHILL[1] except predefined SDL data such as SENDER, PARENT, OFFSPRING, SELF
- for the CHILL buffer concept[5][10], we define a SDL keyword *BUF_SIGNAL* and macros *sendB, receiveB* to send and receive the buffer, respectively
- instead of the implicit addressing mechanism, CROS provides the facility called "Initial Signal" to send a signal to a process in a physical processor(s). For the type of signal, we use "SEND *signal_name* IN *physical_processor_ID*". This form is also used to represent a broadcasting signal[11][16].

5. SDL TO CHILL TRANSFORMATION

Figure 4 shows how a typical *block* is implemented in CHILL, together with the basic transformation examples of the create and stop process instances, signal send/receive, implicit null transition, and dash nextstate.

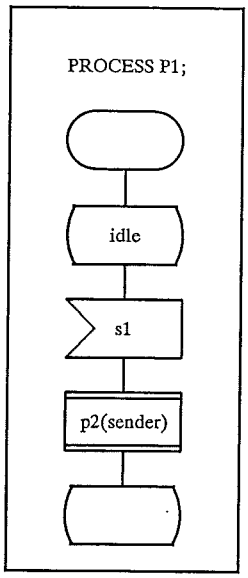
Also shown in Figure 5 are the SDL to CHILL transformation examples of enabling condition, continuous signal, save, initial signal, asterisk state, timer set, and timer reset.
Option is implemented by using conditional compilation[3].

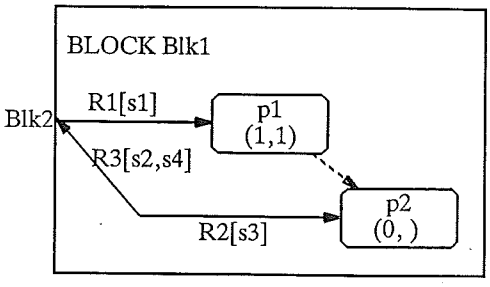
6. SDL USAGE EXPERIENCES

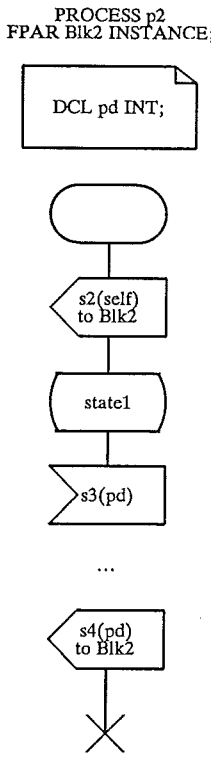
At present(June 1989), 75 *functions* and 98 *blocks* are defined for call processing, basic administration and maintenance functions without CCS(Common Channel Signalling) and

ISDN
define
chann

Thoug
size, i







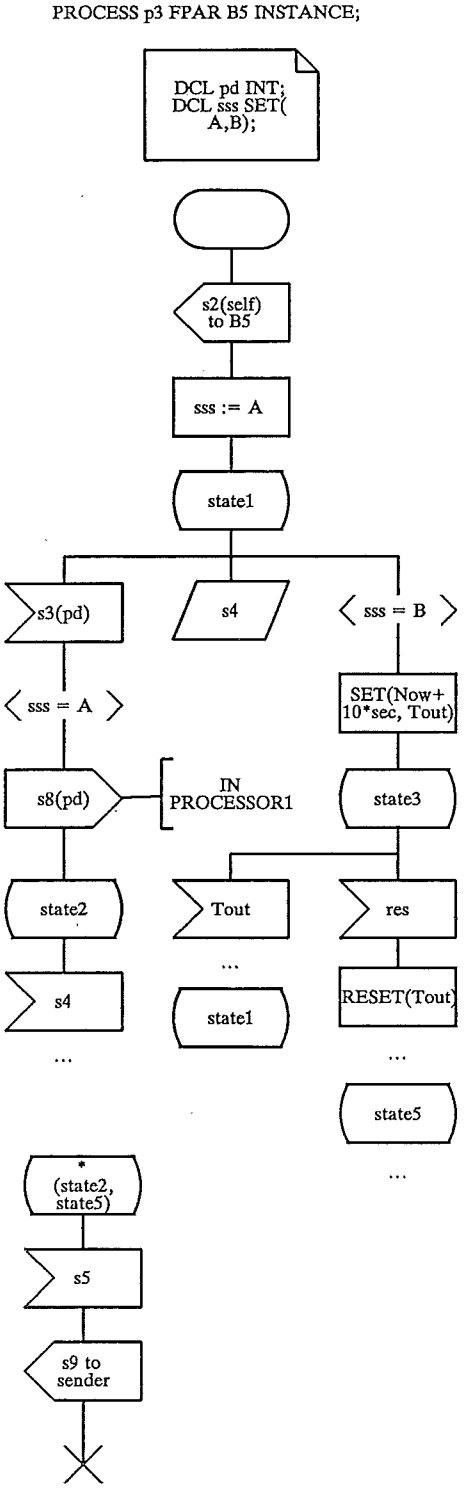

```
/* Block: Blk1 */
SPEC MODULE REMOTE cros;
 ...
main: MODULE
 SEIZE s1, s2, s3, s4;
  ...
 p1: PROCESS();
   DCL sender INSTANCE;
   DCL nextstate SET(idle);
   nextstate := idle;
   DO FOR EVER;
     state_loop: CASE nextstate OF
       (idle): RECEIVE CASE NONPERSISTENT
             SET sender;
         (s1): START p2(sender);
       ESAC;
     ESAC;
   OD;
 END p1;

 p2: PROCESS(Blk2 INSTANCE);
   DCL self INSTANCE;
   DCL pd INT;
   DCL nextstate SET(state1);
   self := THIS;
   SEND s2(self) TO Blk2;
   nextstate := state1;
   DO FOR EVER;
     state_loop: CASE nextstate OF
       (state1): RECEIVE CASE NONPERSISTENT
         (s3 IN pd):
           ...
           SEND s4(pd) TO Blk2;
           STOP;
       ESAC;
       ...
     ESAC state_loop;
   OD;
 END p2;
 start p1();
END main;
```

Figure 4. *Block* Implementation

ISDN(Integrated Service Digital Network) functions. Average 4 to 5 Process definitions are defined per *block* (leaf-block). The number of Signal definitions are about 3,000. The average channel number for each *block* is about 4.9.

Though the SDL descriptions are not formal enough yet, the amount of SDL document, in A4 size, is about 500 pages for *functions* and about 2500 pages for *blocks*,




```
SPEC MODULE REMOTE cros;
...
main: MODULE
 SEIZE s2, s3, s4, s5, s8, s9, res, Tout;
  ...

 p3: PROCESS(B5 INSTANCE);
   DCL sender, self INSTANCE;
   DCL sss SET(A, B);
   DCL Tid, pd INT;
   DCL nextstate
      SET(state1, state2, state3, state5);
   self := THIS;
   SEND s2(self) TO B5;
   sss := A;
   nextstate := state1;
   DO FOR EVER;
     state_loop: CASE nextstate OF
       (state1): RECEIVE CASE NONPERSISTENT
             EXCEPT s4 SET sender;
         (s3 IN pd):
            IF sss = A THEN
              SEND s8(pd) IN PROCESSOR1;
              nextstate := state2;
            ELSE
              SEND s3(pd) TO self;
            FI;
         (s5): SEND s9 TO sender;
            STOP;
         ELSE
            IF sss = B THEN
              Tid :=
              set_timesig(10, SEC, Tout);
              nextstate := state3;
            FI;
       ESAC;
      ESAC;
      (state2): RECEIVE CASE NONPERSISTENT
        (s4): ...
      ESAC;
      (state3): RECEIVE CASE NONPERSISTENT
        (res): cancel(Tid);
             nextstate := state5;
        (Tout): ...
             nextstate := state1;
        (s5): ...
      ESAC;
        ...
     ESAC state_loop;
   OD;
 END p3;
 ...
END main;
```

Figure 5. Sample SDL to CHILL transformation

Since no serious problems have been arisen on conveying design ideas represented in SDL from design organization, that is ETRI, to the implementation organizations, we are sure that the SDL descriptions are very useful as a standardized communication mechanism.

Note that SDL system diagram and block substructure diagram has not been described, since *blocks*(leaf-blocks) are transformed directly from *functions* as mentioned before. To support the diversified transformation, we use a ERA(Entity Relational Attribute) based Meta tool for generating methodology-environments called GMAD/MAD (automatic Generator for System Description and Analysis Manager / system Description and Analysis Manager)[14]. We describe the instances of *function, block, subsystem, message, process, processor, relation,* and their relationships in MAD/DL (Description Language). After transforming MAD/DL descriptions into MAD database, MAD generates various documents, for example, a message survey which is equivalent to that of the SDL system and block substructure diagram. MAD also provides interfaces to verify some of the transformation rules mentioned in 4.2. It is now being considered that the generation of SDL/PR describing system and block substructure, sequence chart, and signal definitions in CHILL from MAD.

A graphic editor(*sge*), syntax checker(*checksdl*), SDL to CHILL transformer(*sdl2ch*), and hard-copy generator(*psdl*) is also proved to be very useful[6][7]. Especially, *sdl2ch* has played a great role to improve software productivity. But we are facing inconsistency problems between code and SDL descriptions as time passes by. For the better maintenance, we are constructing a SDL-CHILL consistency checking environment using *GMAD/MAD*.

We also found that SDL can be used to represent the development process itself. For example, a configuration management process is described in SDL to guide clearly how to do and what to do for the developer and manager. These descriptions also will be used to generate interfaces such as menu panels of methodological tools.

7. CONCLUSIONS

We have experienced that SDL, especially SDL/GR, is a useful language for the "people" to abstract, project, decompose large and complex problems rigorously, unambiguously, completely, and consistently. But we also recognized that knowing SDL language syntax is one thing, and using it effectively is another. The followings are what we would like to say about exploiting SDL from a practical and pragmatic user perspective:

- informal SDL description is OK to start with[4], but it should become a formal one to achieve unambiguity and machine-processability etc.
- education and experience is required to utilize the full power of the language. Especially, the writing experience is really necessary. So, a workshop style education pays
- tailoring is necessary to their organization and methodologies. Adaptation is inevitable especially in the physical design phase, considering the target system implementation language, operation system, and environments
- tool environments must be integrated to ensure the consistency, for instance, between SDL descriptions and source code[5]. It is also important to lay down practical guidelines to use the SDL tool environments.
- since specification language is closely related to human being, Hangul(Korean Language) SDL environment is necessary in our country.

The increased intellectual leverage justifies the effort(i.e., the cost). As Dijkstra says in A Discipline of Programming (Prentice-Hall, 1976), "The power of a formal notation should manifest itself in achievements we could never do without it !"[19].

REFERENCES

[1] Victor M. M. Reijs, "The use of SDL in an ISDN-terminal project," *SDL '87 : State of the Art and Future Trends,* ed. R.Saracco and P. A. J. Tilanus, North-Holland, 1987, pp. 3-9.

[2] R Tinker, R.A. Orr and M.T. Noris, "Supporting the SDL user," *SDL '87 : State of the Art and Future Trends,* ed. R.Saracco and P. A. J. Tilanus, North-Holland, 1987, pp. 57-64.

[3] J. H. A. FRANCO, J. HAIM, and H. M. LIMA, "Going from SDL to CHILL: The TROPICO approach," *SDL '87 : State of the Art and Future Trends,* ed. R.Saracco and P. A. J. Tilanus, North-Holland, 1987, pp. 329-338.

[4] Paolo BAGNOLI and Laura DRAGONI, "SDL USAGE WITHIN THE ITALIAN ADMINISTRATION," *SDL '87 : State of the Art and Future Trends,* ed. R.Saracco and P. A. J. Tilanus, North-Holland, 1987, pp. 11-19.

[5] Mike Regan, John Colton, and Rick Reed, "Experience Using CCITT SDL," *SDL '87 : State of the Art and Future Trends,* ed. R.Saracco and P. A. J. Tilanus, North-Holland, 1987, pp. 21-31.

[6] J. P. Hong, W. Choi, Y. J. Shin, and H. C. Kim, "Integrated Software Development Environment based on CCITT/SDL for Telecommunication Systems," *Proceedings of Second IEE/BCS Conference - Software Engineering '88,* Liverpool, July 1988, pp. 196-200.

[7] J. P. Hong, J. S. Lee, W. Choi, and Y. S. Shin, "SDL-Oriented Graphical Environment", *SDL '87 : State of the Art and Future Trends,* ed. R.Saracco and P. A. J. Tilanus, North-Holland, 1987, pp. 117-126.

[8] Bruno S. Vienna, "Software Development Methodology and Environment in the TROPICO System," *Proceedings of International Switching Symposium(ISS'87),* Phoenix, USA, Nov. 1987, B8.3.6.

[9] D. G. Lee and J. P. Hong, "CHILL Compiling System and its Environment," *Proceedings of IEEE Region 10 Conference and Exhibition,* Seoul, August 1987.

[10] Y. K. Lee, H. K. Kim, and H. H. Lee, "SYSTEM CONCEPTS AND IMPLEMENTATION OF TDX-10 DIGITAL SWITCHING SYSTEM," *Proceedings of Chinese-Korean Telecommunication Technology Symposium,* Taipei, Taiwan ROC, April 1989, pp. 24-31.

[11] Seok Youl Kang, "Realization of a Realtime Operating Systems for TDX-10," *Proceedings of Chinese-Korean Telecommunication Technology Symposium,* Taipei, Taiwan ROC, April 1989, pp. 13-18.

[12] Y.S. Chun, "A Life-Cycle Support System for Real Time Systems," *Ph. D. Dissertation,* Seoul National Univ., 1986

[13] CSELT, "COURSE ON SDL," 1987, Chapter 2, pp. 15.

[14] Y. K. Song, S. M. Han, H. Park, W. H. Chae, and J. P. Hong, "Automatic Generation of System Description and Analysis Tools for managing Software Development Process," *ETRI Technical Memo(in Korean),* Jan. 1989

[15] CCITT, *Draft Recommendation Z.100-Z.104,* SDL Newsletter, 1986.

[16] ETRI, "CHILL Reference Manual," *(in Korean),* Aug. 1988.

[17] Yourdon, "Structured Analysis and Design for Real-Time Systems," Yourdon, inc. Edition 3.0, January 1984

[18] Ference BELINA, Dieter HOGREFE, "The CCITT-Specification and Description Language SDL," *Computer Networks and ISDN Systems 16* North-Holland, 1988/1989, pp. 311-341.

[19] Paul C. Grabow, "A perspective on Specification Language," *Proceedings of ICCL 88* Maiami USA, 1988, pp. 164-18.