A New Integrated Software Development Environment Based on SDL, MSC, and CHILL for Large-scale Switching Systems

DongGill Lee, JoonKyung Lee, Wan Choi, Byung Sun Lee, and Chimoon Han

CONTENTS

I. INTRODUCTION

II. ENVIRONMENT CONCEPT IN A DEVELOPMENT METHODOLOGY FOR LARGE-SCALE SWITCHING SYSTEMS

III. ARCHITECTURE AND CHARACTERISTICS

IV. EXPERIENCES FROM THE USE OF ISDE IN TWO PROJECTS

V. CONCLUSIONS

REFERENCES

ABSTRACT

This paper presents a new software development environment that supports an integrated methodology for covering all phases of software development and gives integrated methods with tools for ITU-T (Telecommunication Standardization Section of the International Telecommunication Union) languages. The design of the environment to improve software productivity and quality is based on five main concepts: 1) formal specifications based on SDL (Specification and Description Language) and MSC (Message Sequence Charts) in the design phase, 2) verification and validation of those designs by tools, 3) automatic code generation and a safe separate compilation scheme based on CHILL (CCITT High-Level Language) to facilitate programming-in-the-many and programming-in-the-large. 4) debugging of distributed real-time concurrent CHILL programs, and 5) simulation of application software for integrated testing on the host machine based on CHILL. The application results of the environment compared with other approaches show that the productivity is increased by 19 % because of decreasing implementation and testing cost, and the quality is increased by 83 % because of the formal specifications with its static and dynamic checking facilities.

I. INTRODUCTION

A software development environment should be highly flexible, extensible, and very well-integrated. Software development environments are intended to provide a cohesive and integrated set of tools to support the process of software development. There is much current research into environment design focusing on maximizing the degree to which these tools can be integrated [1]-[3].

CHILL (CCITT High-Level Language) [4], [5] is recommended by ITU-T (Telecommunication Standardization Section of the International Telecommunication Union, formerly CCITT) as a standard programming language. CHILL is characterized by strong type checking, information hiding, piecewise programming, concurrent programming, and exception handling. SDL (Specification and Description Language) [6] is well-suited for all systems whose behavior can be modeled by extended finite-state machines in which the focus is to be placed especially on aspects of interaction. MSC (Message Sequence Charts) [7] specify scenarios of message flows between a system and its environment at the system level with scenario hierarchy to use case modeling during the analysis phase and test suits during the test phase.

In order to improve productivity and quality of telecommunication software, several SDL environments [8]-[11] and several CHILL environments [12]-[15] have been developed in many countries and used to develop telecommunication systems. Most of these environments are dependent on their own computing environment, switching system, and organization. Some of the environments such as SDT [9], GEODE [10], and CHIPSY [14] are designed to be portable and flexible in their application domains. SDT and GEODE focus on SDL and MSC environments for the design phase, while CHIPSY focuses on a CHILL environment for the implementation phase.

With emphasis on integrated methods and methodologies for covering all phases, we have developed the *Integrated Software Development Environment* (ISDE) with a view to achieving transportability by using the ITU-T languages: CHILL, SDL and MSC. Based on an integrated paradigm, ISDE supports various activities occurring in system analysis, system specification and design, implementation, debugging and simulation for telecommunication software development with standard languages and methodologies. Also, due to compliance with standards and a high-degree of portability, ISDE can be used to develop a wide range of telecommunication application software. ISDE was successfully applied to development of TDX-10 ISDN switching system [16]. The results show that ISDE can decrease the total cost considering productivity and quality of the development of a system because the reduced cost in the implementation and testing phases is much higher than the increased cost in the design phase caused by using formal languages recommended by ITU-T.

This paper consists of five sections. In the

subsequent section, we present the concepts of ISDE based on ITU-T languages. We explain the architecture and the characteristics of our environment in section 3. In section 4, we describe our experience of using ISDE in the development of switching system and compare it with related works. Finally, in section 5, we summarize our approaches and contributions.

II. ENVIRONMENT CONCEPT IN A DEVELOPMENT METHODOLOGY FOR LARGE-SCALE SWITCHING SYSTEMS

ISDE is a language-centered environment. Figure 1 shows the software development paradigm supported by ISDE. In terms of the activities in software development, the system consists of four parts: *Integrated SDL Environment for Telecommunication systems* (ISET) [17], *CHILL Programming Environment* (CPE) [18], *CHILL Debugging environment* (CDE) [19] and *CHILL Simulation Environment* (CSE)) [20]. ISET and CPE are integrated by translating SDL to CHILL and by means of uniform environment user interfaces.

Our large-scale switching systems such as TDX-10 and ATM switching system are local/tandem/toll switching systems that support 100,000 subscribers. In terms of software development, they should be thought of as real-time, fault-tolerant, and distributed systems based on Motorola 68020 or SUN SPARC processors, and they require methodologies and environments for very large software programming projects. Figure 2 shows the structure of our development process [21] for the analysis and design phases.

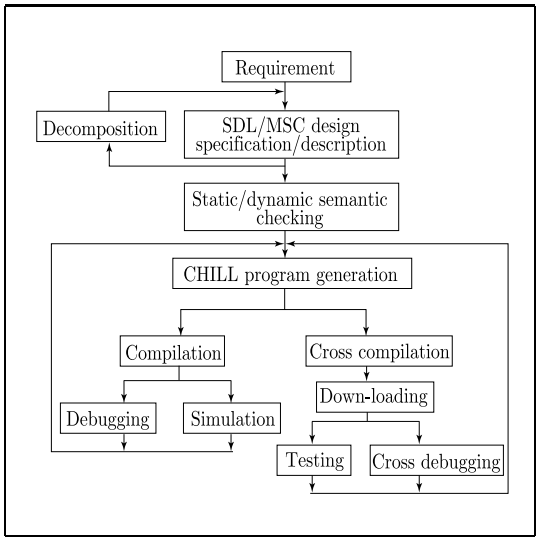

Fig. 1. Development process with ISDE.

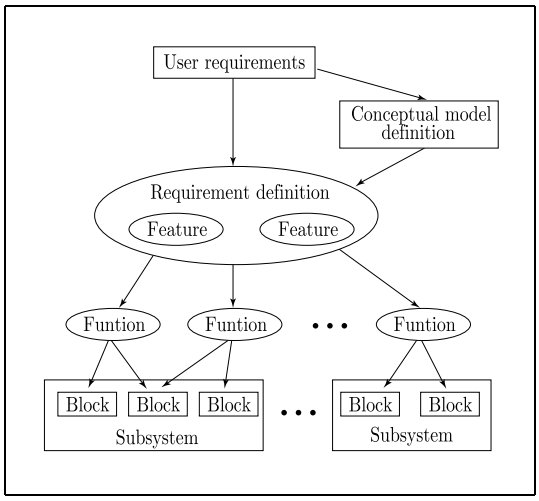

Fig. 2. Development approach.

Concrete requirements are defined to be used between the customer and the developer by an inspection of informal user requirements. When defining requirements, a conceptual model is defined to achieve conceptual integrity among various developers. The requirements definition specifies everything that the end-users see, including features (functions from the user's point of view) and external interfaces in the system development.

Functions are formally defined from requirements with what the system is to do rather than how the system is to be implemented. Each function is specified in the form of SDL system and process diagrams, as if the system had only the process definition for the function. The system diagram defines processes, external interfaces in the environment, and signals to/from processes. The process diagrams mainly show how the system level states are changed and what responses are made according to external stimuli.

A function is subject to being transformed into one or more blocks which are the basic development units in design and implementation. SDL is used to specify the behavior of functions and to design the internal structure and behavior of blocks. MSCs show the normal sequence of signals interchanged between one or more blocks and their environment per function. The interaction of messages and actions, interfaces between components, and identification of data can be defined from the analysis of MSCs. MSCs are used as the starting point for drawing SDL process diagrams. Subsystems are formed by grouping blocks. When blocks are defined, they are designed in the form of SDL block diagrams and control flow diagrams in reference to MSCs and message surveys. In our development approach, using MSCs is not regulation in detail design for the components of a block. Each block is mapped to an executable CHILL module.

ISET provides a paradigm for formal specifications and descriptions by supporting the usage of formal languages that are SDL and MSC, for verification and validation of system specifications by static semantic checking based on an entity-relationship model of SDL and by dynamic semantic checking based on a numerical Petri-net model of SDL, for stepwise refinement, for documentation, and for translation into CHILL programs. After semantic checking of SDL descriptions, the SDL descriptions are translated into CHILL programs and executed interactively on a simulation environment for validation.

Basically, CPE provides an environment for editing, compiling, and running CHILL programs in the host environment. In order to prepare programs to be executed on the target under the host environment, it provides the facilities to cross-compile source programs, and to down-load them onto distributed multiprocessor target systems.

For debugging and testing of CHILL programs CDE provides CHILL source-level debugging facilities on the host environment and target machines. Typically, programmers examine and trace the logic of programs by us-

ing a source-level debugger first, then simulate programs in a CHILL Simulation Environment for integration testing with some other execution modules. CSE provides a target software environment by simulating distributed multiprocessing, concurrency within a program, interactions of signals in terms of CHILL, real-time data base management, and time management. After the integration test on a host, programmers debug real-time facilities of an execution module running on the target systems at source-level under the control of the host environment. Finally, the entire software in the real environment is tested. Since our switching system supports multi execution modules within a processor, it needs to run all of the multi execution modules together to monitor and trace message passing and real-time facilities among them by using a tracer such as the *IPC monitor*.

III. ARCHITECTURE AND CHARACTERISTICS

1. Integrated SDL Environment

A layered architecture with a unified representation of both Graphical Representation of SDL (SDL/GR) and textual Phrase Representation of SDL (SDL/PR) not only facilitates a flexible and extensible environment, but also makes it easy to integrate tools and languages. Figure 3 shows the SDL environment architecture, ISET.

Tools of ISET may be classified in terms of functional roles as follows: the graphical environment with *SDL graphic editor* and *hard-copy generator*; the analysis with *SDL syntax checker, SDL static semantic analyzer* and *SDL dynamic semantic analyzer*; translation with *SDL/GR to PR translator, SDL/PR to GR translator, SDL/PR compiler, SDL to CHILL translator*, and *SDL to DB translator.*

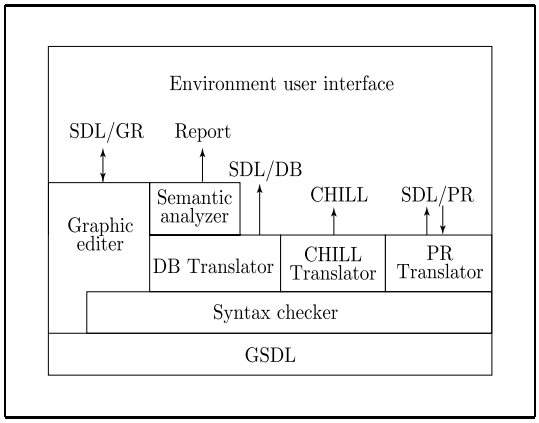

Fig. 3. ISET architecture.

A. Internal Integration

Since the conceptual model of SDL is based on extended finite-state machines, its abstract syntax is thought of as graphs consisting of nodes and transition arcs. Moreover, two-dimensional languages with graphical representation have a natural representation that is a general directed graph rather than a simple tree. The graph representation of SDL (GSDL) is an internal language that preserves all the semantics of SDL with graphical information to represent SDL/GR diagrams also. Basically, it has predefined node types and arc types.

The node types define semantics of SDL symbols and their graphical shapes. However, the arc types define only graphical shapes (solid, dashed, or thick lines, and lines with or without arrow heads), therefore they are interpreted in the context of the connectivity of node types. Internal integration is achieved by adapting GSDL as a core language for all tools to share GSDL. SDL descriptions saved in the form of GSDL may be followed by translation into other languages: SDL/GR, SDL/PR, CHILL, and SDL/DB. The GSDL not only facilitates integration of environment tools, but also enables them to share and utilize internal data structures such as parse trees, symbol tables, and abstract syntax graphs.

B. Verification and Validation

In ISET, the formal specifications is defined as the use of formal languages that are SDL and MSC and the verification of the specifications is defined as the semantic checking in terms of SDL semantic rules. Semantics of SDL can be divided into static semantics and dynamic semantics. Static semantic analysis is primarily concerned with checks for consistency and completeness and with correctness and compliance with SDL semantic rules. On the other hand, a major concern of dynamic semantic analysis is incorrectness of dynamic behavior such as the possibility of deadlock and unreachability. The entity-relationship model was used for representing static semantics while our new numerical Petri-net model for SDL [22] was employed for representing dynamic semantics. For verification and validation, ISET provides an SDL static semantic analyzer, an SDL dynamic semantic analyzer, and an SDL to CHILL translator for testing.

From the entity-relationship model of SDL [23], a normalized relational model is derived by an SDL to DB translator. The analyzer reads the SDL/DB form from the SDL database and binds all or parts of the system descriptions to configure structures and to resolve interfaces and interactions among components. It then performs static semantic checking for semantic checks such as visibility and uniqueness of names, consistency of signals defined in blocks, processes, and services as well as consistency and completeness between incoming signals and outgoing signals, and channel connectivity and consistency in interaction diagrams. To analyze the dynamic behavior of a system described in SDL, the dynamic semantic analyzer can effectively construct a symbol table and a state transition table from transformation rules based on SDL syntax and semantics, generate a reachability graph, and perform dynamic behavior checking and generate diagnosis messages for deadlocks, unreachability, detection of cycles and no receiver.

C. Generation of CHILL Programs

From SDL/GR and SDL/PR that are represented by GSDL, a CHILL program can be generated. The generated CHILL code cannot

be guaranteed to be completely executable unless the SDL user provides additional CHILL code inside the SDL symbols as detailed as possible, as specified by the programming language, because of the degree of details and semantic gap between specification (or design) and implementation. One of the most difficult problems is data type conversion. The generated code can be executed on the CHILL Simulation Environment.

D. Prototyping Based on the CHILL Simulation Environment

The automatic generation of CHILL programs enables users to use the prototype executable code to refine SDL specifications and descriptions by the CHILL Simulation Environment. In the early design phase, all kinds of execution modules could be constructed and tested according to the system hierarchy in SDL. After a basic test is performed for each block, the (part of) system is integrated by adding the individually tested blocks and verified/validated whether a certain function meets its specification.

2. CHILL Programming Environment

The CHILL programming environment (CPE) consists of the *CHILL compilation system* (CCS) and the *CHILL programming support environment* (CPSE). CCS consists of : the *native compilers* for the host machines VAX/UNIX, SUN-3/UNIX, MIPS, ALPHA, and SUN/Solaris; the *cross-compilers* for target switching systems; the *separate compilation system* which supports piecewise programming. The CPSE includes the following tools: a *syntax-directed editor*; a set of language-sensitive tools for assisting program development in CHILL such as an *interface analyzer*, a *static analyzer*, a *pretty printer*, a *cross-reference generator*; a set of cross-development tools for building target programs, including *cross-assembler/linker, program builder*, and *down-loader*.

A. High Portability

CPE was designed paying much attention to retargetability of the compilers and rehostability of the tools in order to promote flexibility of the CPE. We separated the language-dependent part from the machine-dependent part in building the compilers. Each compiler is mainly composed of a front end and a back end. The front end, which emits intermediate code, EM [24], is shared by the compilers, while several back ends exist for the corresponding host or target machines. The back end employs a pattern matching table-driven code generation technique to facilitate retargeting. It enabled us to make a code generator for a new machine just by preparing a machine description table, even if it required some hard work to find better code patterns. We also developed an RTL translator for translating the EM code to the RTL code that is an intermediate code of GNU compilers. The RTL translator makes it possible for the CHILL compil-

ers to use many new machine back ends and the good optimization facility of the GNU compilers. The separation and reduction of machine-dependent part enabled us to develop a native compiler and a cross-compiler simultaneously without spending lots of time. It also allows the environment to meet processor improvement of target systems.

B. Safe Separate Compilation

Since the characteristics of large-scale switching systems software are programming-in-the-large and programming-in-the-many, ensuring the consistency between implementation modules and their specifications in whole system is an important problem. The issue of separate compilation for very large-scale software is that we have to make the implementation modules and their specifications automatically consistent in every update for guarantee of safety. Our approach is that the specifications of implementation modules may be manually or automatically derived from the modules in order to ensure their consistency. The separate compilation is done in three steps: automatic specification generation, change analysis [25], and compilation or recompilation. The automatic specification generation approach eliminates the writing of specification modules which are required to provide interface information. The modification of implementation modules causes the compiler to generate their specifications. After the compiler performs visibility and semantic analysis, the compiler generates the information for the granted names with derived mode information and the seized names for specifications. A change in the specification may directly or indirectly affect associated modules. This causes change analysis to check the consistency of the specifications for all modules. If some changes occurred, it produces the context files corresponding to the source files affected just by the changes. Updating a context file causes recompilation of the corresponding implementation modules. In this case, the compiler includes the context file and compiles the source program again. The sequence of actions can be represented in time dependency among files containing implementation, specification modules, contexts, and so on. The configuration manager determines such time dependencies for the given configuration description. Actual control of the separate compilation procedure is realized by means of the *make* facility in UNIX.

C. Multi-language : CHILL and UNIX/C

For the enhancement of system portability, UNIX machines were decided to be the host machines for CPE and every tool. In our experience, porting CPE from VAX/UNIX to SUN workstation took a few weeks. Furthermore, by keeping a compatible interface between CHILL and C procedures we can take advantage of various UNIX capabilities such as I/O routines and a variety of libraries on the

host as well as on the target system. To achieve a compatible environment of CHILL and C, some features were considered or added, such as call frames like in C, compatible data representation for common data types, string literals, and so on.

3. CHILL Debugging Environment

The internal layer architecture and tools of the CHILL Debugging Environment (CDE) are shown in Figure 4.

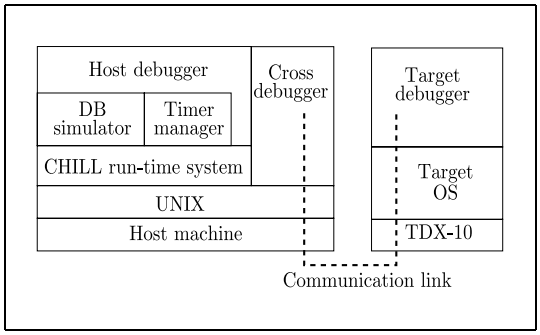

Fig. 4. CDE architecture.

A. Debugging of Concurrent Programs on the Host

The *host debugger* [26] can debug CHILL programs on the host at source level. Since concurrent processes are created and maintained by the CHILL Run-time System (CRS) under a UNIX process on the host, retrieval of the status of CHILL processes and synchronization primitives can be done by accessing data structures of the run-time system, and full control of individual threads is performed. For tracing a particular process instance or setting breakpoints at them, the debugger performs selection of a process only when the control reaches a predefined process name or process instance. This capability will lead users to set conditional breakpoints in process instances. The debugger can also communicate interactively by using CHILL signals to CHILL programs.

B. Cross Debugging

The *cross-debugger* [27] is a tool running on the host to trace and debug programs at source level being executed on the target, communicating with a *target debugger* [27] that is a primitive debugger residing on the target. The cross-debugger accesses all the resources such as source code and symbolic information on the host, and controls debugging processes by minimal support of the target. The cross debugger has a similar functionality as the host debugger except for the tracer part. When starting a debugging session, the cross debugger separates the actual executable code from the symbolic information, then downloads the actual code on the target. The main concept of cross debugging is implemented in a cross tracer that is the entry point to the target system. The cross tracer can control all the cross debugging processes with a small set of the low-level operations. All operations and results to/from the target debugger are handled through the cross tracer via the communication link. The host debugger receives the execution results in the form of raw data. It displays this data at the CHILL-level. The target debug-

ger has low-level operations such as reading or writing contents of memory and registers, setting breakpoints at instruction level, executing a piece of code, and providing concurrent processing information via a request to the target operating system.

C. Regression Testing of Real-Time Programs

Debugging of a concurrent program is difficult due to its nondeterministic behavior and timing constraints imposed on them. These two factors rule out the reproduction of an error. The deterministic replay mechanism records only events that cause nondeterministic behavior of concurrent processes in the event history and replays the concurrent processes using a scheduling method which executes concurrent processes following the order of the recorded events. These facilities are performed by the *event monitoring system* which records dependent events which affect the runtime behavior of concurrent programs and by the deterministic replay which controls the reproduction of the program behavior using a recorded event history and input data.

4. CHILL Simulation Environment

Our large-scale switching systems have distinguished characteristics that are real-time, embedded, interactive, and distributed. These systems contain very large and complex software. Therefore, in the development of switching software, simulation is believed to be one of the most important processes in producing expensive,high-quality software.

The internal layer architecture and the simulation environment tools are shown in Figure 5.

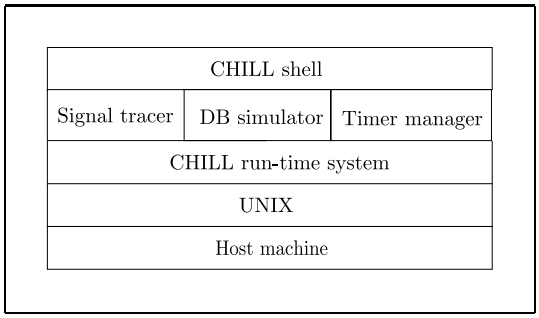

Fig. 5. CSE architecture.

A. Simulation of Distributed Concurrent Processing

CHILL provides a variety of concurrent processing features: process, mutual exclusion, event, buffer and signal. The CHILL Run-time System (CRS) provides simulation of all the concurrent processing features. CRS consists of a kernel and CHILL primitive handlers. The kernel provides the functions, which are regarded as the most machine and operating system dependent part, such as context switching, scheduling, allocation of runtime stacks for processes, and synchronization routines invoked by primitive handlers. The interface between application programs and CRS is accomplished via the primitive handlers. Since they are realized by procedures invoked by processes, they execute their functions on the process contexts of the invoking

processes. When a program starts, the control goes first to the kernel of the CRS in the context of the UNIX process. Then the kernel initializes the status and it creates the outermost process that is the parent process of all CHILL processes to be created later, and it allocates run-time stack for the process. At this time the application program is executed in the context of the outermost process. Since each action statement for concurrent processing is translated by the compiler into one or more primitive handler calls, a handler is invoked in the same way of procedure call mechanism in the context of the calling process. The difference to ordinary procedures is that the primitive handlers are allowed to access the kernel data structures. A handler may return to the calling process, completing the request, or it may be blocked. When it is blocked, it causes context switching to the kernel process. Then the scheduler selects a process from the ready queue, and resumes that process.

For simulation of distributed processing which is performed by the CHILL signals in switching systems, communication between processes of other modules are achieved via common shared memory mechanism in UNIX. On the other hand, communication between processes within a module is achieved via local shared memory which is dedicated to the module. The two types of shared memory are allocated by the CHILL Shell, and used and managed by the CHILL Run-time System. The process control block of processes and information on signals is stored in the common shared memory. Information on event, buffer, and region are stored in the local shared memory. Semaphores for the two types of shared memory are respectively allocated to support mutual exclusion.

B. Event Monitoring

The switching system is composed of many processes, and the synchronization and communications between the processes are achieved mainly by signals. Correctness of the system depends on the contents and sequences of signals transmitted between the processes. Thus, our approach to simulate a switching system is based on the event monitoring technique [28], which traces signals and messages transmitted between processes with a post-mortem view of the execution of the program. The *signal tracer* records the events occurring while a module is executed into a signal database, which will be used for tracing signals. The *signal analyzer* analyzes the information on signal transmission history between processes in the database which were generated by the signal tracer, and displays them on a graphic window.

C. Application Library for Simulation

To simulate the switching system software on the host machine, a library for timer management is provided. The *timer manager* [29] manipulates time constraints for the real-time system based on the UNIX timer. It can control activation of a special process, receiving a registered signal and stopping of a process on the CRS. Since the data in the TDX-10 is

managed by the Data Base Management System (DBMS) which is the relational data model with real-time responses, the DB simulator provides the simulation facilities of the DBMS for the switching system on the CRS.

D. Interactive Facility for Simulation

For the interactive facility for simulation, the *CHILL Shell* interprets and executes user commands interactively, and shows states of processes as running or being delayed. It can also send messages to the running processes with a data form in CHILL. Moreover, one or more CHILL programs on CHILL Shell can be executed. The CHILL Shell allocates the shared memory and semaphore, required by the module before the execution, and deallocates them after their execution.

5. Comparisons with Related Works

SDT [9] and GEODE [10] focus on SDL and MSC environments for the design phase, while CHIPSY [14] focuses on a CHILL implementation. However, ISDE covers all phases from the design phase to testing based on integrated methodologies and methods. SDT and GEODE translate SDL into C but ISDE translates SDL into CHILL. Since this scheme can reduce the semantic gap between the specification and the implementation, the design phase and the implementation phase could be consistently integrated by SDL to CHILL translation.

ISDE is more flexible than CHIPSY in retargeting because ISDE adopts a pattern matching table driven code generation scheme [24] and GNU back ends with code optimization of high quality. Moreover, our experience shows the EM code that was suggested by Tanenbaum [24] does not have a sufficient instruction set for RISC machine and CHILL. Thus, we have extended the EM instruction set by the addition of functions for supporting efficiently parameter passing, bit manipulation, and function pointer manipulation.

In comparison with state graph [10] in GEODE, our numerical Petri-Net [22] for SDL was restrictive to use for analysis of a large system because the number of states of numerical Petri-Net can increase rapidly due to incremental integration of SDL specification. However, it is helpful in the early time of the design phase for checking interactions among blocks.

Although Tichy's separate compilation scheme [25] minimizes recompilations by incremental change analysis, it causes the difficult problem to determine the sequence of the recompilations for resolving semantic and its cyclic dependency. Our separate compilation scheme also minimizes recompilation. Moreover, our separate compilation scheme does not depend on the sequence of recompilation, because it performs 3 steps, that are specification generation, change analysis, and context generation, in the context of an entire program at once. In an early stage of large software development, this scheme was very useful in practice, it was inefficient in the last stage of the development because whenever

a change occurred, checking of all system interface had to be performed. To solve this disadvantage, our scheme was modified to perform 2 stages of separate compilation: one was change analysis in system interfaces and the other was change analysis within an executable module.

IV. EXPERIENCES FROM THE USE OF ISDE IN TWO PROJECTS

1. Comparison of two Experimental Projects

An experimental project for developing a demo system was performed by university students. The object system was a small subset PABX (Private Automatic Branch eXchange) demo system that has simple basic functions for call processing and maintenance. Its architecture was designed as a distributed concurrent system based on message passing as in the TDX series. The project was composed of several sub-projects according to what methodology was applied. Among them, in this paper, the two approaches in Table 1 were observed. The methodology and tools of project A were STD (State Transition Diagram) and MSC for design. The C language and its programming environment in UNIX were used as implementation tools. This approach is very similar to the previous approach in our organization.

For the verification of design specifications, the STD static analyzer and MSC analyzer were used. From the experimental project, some data are shown in Table 1. Considering the indications of Albrecht [30] that 1 line of source (LOC) of CHILL approximates 1.29 times the functionality of 1 LOC of C, the number of C source code lines are compensated by multiplying the factor for the functionality. The effect of other major factors [31] for evaluating productivity and quality are neglected because the important factors including people, problem, product, and resources are approximately the same. After the development of the two demo systems, the defects shown in Table1 were reported until the com-

Table 1. Comparison of the two approaches.

Project name		A	B
Development environment		STD, MSC, C	ISDE
# of source line (LOC)		3420	3400
Man power (MN)	Design	1.9	2.8
	Implementation	2.2	1.1
	Testing	3.2	2.1
	Total	7.3	6.0
Defect	Design	3	1
	Implementation	8	5
	Total	11	6
Productivity (LOC/DAY)		16	19
Quality (LOC/DEFECT)		310	567

pletion of the projects, that was about 1 month. In this paper, we use the definition of the productivity and quality which is defined by Pressman [32] as follows:

Productivity = LOC/person-day, where LOC is line of code
Quality = LOC/Defect, or
= Byte/Defect, where Byte is memory size of code

From the Table 1, we can observe the following aspects:

- The results of this project show that the productivity and quality of our approach is 18.7 % and 82.9 % higher than of the ordinary approach.

- The portion of implementation and testing cost is remarkably reduced. On the other hand, that of design cost is increased. However, it was possible to decrease the total cost of the development of a system. Figure 6 shows comparisons of the time spent in each phase, using the ISDE approach and the ordinary approach.

- By using the formal specification and concurrent languages with powerful verification and validation tools, ISDE remarkably enhances the reliability of software which is one of the most important characteristics of switching system software.

- SDL/CHILL are more suitable languages than STD/C to develop real-time concurrent systems based on distributed architecture because SDL and CHILL provide formal specifications and languages features for concurrent processing and distributed processing.

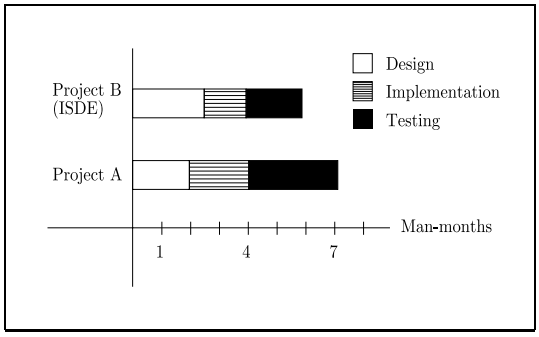

Fig. 6. Effort spent in the two approaches.

2. Development of the TDX-10 ISDN Switching System

The development project of a large-scale switching system for ISDN (Integrated Services Digital Network), that is TDX-10 ISDN [16], was performed for 5 years since 1990 in ETRI. In 1992, the system was tested and produced good results. For 3 years from 1993 on, the system has been commercialized, and in 1995 the first cut-over and operation began. Table 2 describes the status of the defects of that period. From 1990 to 1992, we spent 12 months for formal specification and design, 8 months for CHILL programming, and 16 months for debugging and integration testing.

Some parts of the software matrix of TDX-10 ISDN is shown in Table 2.

Table 2. TDX-10 ISDN Software.

# of source files		2,120
# of source lines (executable code)		985,186
Object size (KBYTE)		19,737
# of procedure definitions		8,202
# of process definitions		748
# of signal definitions		4,085
Total man power (MM)		171
Defect	# of defects	1,203
	System design (%)	2.7
	Implementation (%)	58.0
	Operational data (%)	3.1
	Interface & integration (%)	22.3
Productivity (LOC/DAY)		26.3
Quality (LOC/DEFECT)		1,602
Quality (KBYTE/DEFECT)		16.3

The result data from the application of TDX-10 ISDN switching system is that productivity is 26.3 LOC/Day and quality is 1602 LOC/Defect. If memory size is used as a measure of software size in stead of LOC, the quality is 16.3K Byte/Defect. This result is higher than some other studies [33,34]. The comparisons with some other studies are as following:

- F. Myatty [33] gives statistical data for productivity and quality based on a size-oriented software matrix. For a medium-sized software system, 10 to 20 lines of executable source code are typically produced per day per person during the entire period of development of the system. The approximate number of errors found in every 1000 lines of executable source code in a delivered software system is less than 4 errors.
- Data of a successful project for the development of a robot controller called RC2000 in General Electric Company is published by J. L. Lawrence [34]. He explains that the project is a very successful project in its area. The project used the DARTS (Design Approach for Real-Time Systems) method [35] that is a data flow approach for design and a programming language that is very similar to Ada. The DARTS design method can be thought of as extending the Structure Analysis/Structure Design method by providing an approach for structuring the system into tasks with synchronization as well as a mechanism for defining the interfaces between tasks. J. L. Lawrence estimated that software productivity of the project was 20 LOC/Day and software quality based on memory size was 4.2K Byte/Defect. Almost half the members of the development team

worked on the previous project.

Productivity and quality in the TDX-10 ISDN project is 32 % and 290 % higher than in the RS2000 project respectively. The two systems have similar system requirements including real-time, concurrent processing, and embedded system. In the case of Myatty's study [33], the productivity of our approach is 32 % higher and the quality is up to 500 % higher. From the above comparisons, we can conclude that ISDE contributes to enhance the productivity and quality of switching systems. Especially, it was very helpful to enhance the reliability of the switching system by formal specifications with verification and validation. Although the software was a very large-scale and complex concurrent parallel processing system, ISDE was successfully applied and used in developing the TDX-10 ISDN.

3. Benefits from Using ISDE

Our experiences of ISDE applied to the development of switching system software shows that this environment has made a great contribution to productivity and quality enhancement of complex distributed concurrent processing systems in the following aspects:

- Precise design specification using SDL and MSC.
- The quality of design specification could be much improved by the checking of static and dynamic semantics for correctness and execution of design specification in the context of the target environment.
- Owing to the simulation of design specifications by automatic CHILL code generation, the target system could be prototyped rapidly in the early phase and almost all of the design errors could be detected and removed. Moreover it allowed us to define and test interfaces well. This activity contributed to decreasing the total cost of development and maintenance.
- Automatic generation of CHILL code skeletons
- The smooth transition with the correctness checking from the design phase to the implementation phase through the automatic program generation allowed us to decrease implementation costs of large-scale software.
- High portability of CPE was an important factor when deciding the target processors and computer environment for users. It took about 1 year to port CPE for a new target process including new development of the optimization, and about 6 months for a new host workstation. The GNU back ends in CPE were used for experiments to estimate the performance of switching system software in a new processor within a short time. These allow us to adopt easily the new

processor for target systems and host computers.

- Separate compilation based on automatic context generation, change analysis, and strong safety checking were very efficient in programming-in-the-many and in-the-large by supplying valid interfaces before integration. However, in the latter half of development, the overhead of the safe compilation increased because the system interfaces were fixed and changes of the code occurred in local scope.

- During integration testing, simulation activities on the host machine were very helpful for developing very large-scale distributed software because all kinds of execution modules defined in the system hierarchy in SDL could be constructed easily and they could be simulated on the host environment to test their dynamic interactions by tracing all messages among the execution modules.

- In the host simulation, special situations which may occur in operation could be generated by additional software for test case generation. These kinds of testing were very useful to improve the reliability of target software.

- Since several companies were involved in the development of TDX-10 ISDN, our environment played a key role in providing a standard working environment for cooperations with other companies. As a result, it facilitated technology transfer to cooperating companies based on common methodologies and methods.

The experiences of SDL and CHILL usage are as follows:

- SDL is a useful language for people to abstract, project, and decompose large and complex problems. Moreover, the SDL descriptions are very useful as a communication mechanism because no serious problems arose in conveying design ideas represented in SDL from the design organizations to the implementation organizations.

- Standard SDL usage rules were defined to improve understandability and to set up SDL subset rules. The followings are SDL features that were not used: service, signal refinement, block substructure, channel substructure, view and reveal. For efficient communication, extensions of SDL to support Event and Buffer defined in CHILL were required. The concurrent features in CHILL with exception handling were estimated very well due to efficiency and semantic consistency of SDL.

- For efficient communications, signal features for set-up of initial communication channels in CHILL were

introduced. We defined an initial signal and a normal signal. The normal signal is used when both the process instance of sender and receiver are identified before communicating. The initial signal is used when the process instance of the receiver is not identified. The sender should send the initial signal to the destination processor. The extended statement for the send action is denoted as "SEND initial-signal IN destination-processor;". The receiver should inform a list of initial signals to be received to the demon process operated in its processor. Then the demon process can know which process wants to receive what initial signals. So, when an initial signal has arrived, the demon process assigns a receiver and passes the signal to the destination process. Then an initial communication channel is established. The mechanism of initial signal reduced much communication overhead for broadcasting or multicasting to a process definition.

- Concurrent features in CHILL were also extended for supporting target operating system dependent facilities efficiently. Since CHILL defines that all of the signals should be saved, which is the opposite to the semantics of SDL, we added a nonpersistent property to the receive case action for signals. The extended receive case statement is denoted "RECEIVE CASE [NONPERSISTENT EXCEPT signal-names];". NONPERSISTENT means that the signals are not saved. EXCEPT describes the signals which are saved. The default action for signal is persistent for the signals. Extended mechanism for signals could reduce overheads of the operating system for saving unnecessary signals, and also remove the semantic gap between CHILL and SDL.

- The start action in CHILL is extended by directives for efficient resource allocation, time supervision and agent facility. Some directives are as follows:

  - STACKSIZE directive describes the stack size of process. The default size of the imaginary outmost process is 4K byte and of internal processes is 1k byte.
  - SYSTEM directive means the started process is system process. In comparison with a user process, a system process has more authority to use resources of the system and a higher priority.
  - PROXY directive generates an agent process that has the same process instance value as the starting process or a specified instance value in the directive. When an agent process is started, then the parent process is terminated.

Since the agent process and its parent process can have the same instance value, the agent process can perform the roles of the parent process and new missions of the agent process.

- Compatibility of CHILL and C made it possible to develop the target operating system and ISDE in parallel and to use utilities of the UNIX environment as well as the input/output facility of UNIX/C.

- Some features of CHILL were rarely used. Typical examples were nested modules, begin end blocks and definition of buffer or event within a process. The number of unused keywords in the CHILL definition was about 20. Some of them are addr, based, begin, bin, card, entry, forbid, getstack, max, pred, ptr, row, result, upper, xor. This fact shows that it is necessary to adapt CHILL to each organization by pruning some alternative and complex language features for better understanding of CHILL and simplifying the development of compilers and related tools.

V. CONCLUSIONS

We have developed an integrated software development environment for switching systems based on ITU-T languages to enhance the productivity and quality of software. The environment supports activities to specify and describe the behavior of desired switching systems in SDL with MSC, to generate and compile CHILL programs automatically, and to debug and simulate distributed concurrent real-time CHILL programs. For the integrated tool environment to support the given methodology, the environment has been built to have high transportability and flexibility based on an integrated layered architecture.

Our experience in applying the environment to software development for switching systems shows the following:

- The environment increases software productivity by 19 %. The effective factors are integrated methods to reduce the cost of development and maintenance such as debugging and maintenance of software at design level, error detection in early phases, easy rapid prototyping, debugging for distributed real-time concurrent property, and integration test on a host machine by simulation.

- The environment also enhances software quality by 83 % by using formal methods with CASE (Computer-Aided Software Engineering) tools such as formal specification with verification and validation, simulation and testing of specifications, automatic CHILL code generation, concurrent programming, safe separate compilation, distributed concurrent debugging, and sim-

ulation of entire application software of the target system.

By development of the environment to assist a consistent methodology and to provide well-integrated methods with tools based on ITU-T languages, we conclude that this environment is a good solution to improve software productivity and quality of large-scale switching system and it can also be utilized in other fields of telecommunications system development because of their transportability and flexibility with compliance to ITU-T standard languages. Currently, we are extending ISDE to support object-oriented design and implementation based on ITU-T languages and object-oriented testing and maintenance based on Kung's approach [36].

REFERENCES

[1] Michal Young, Richard N. Taylor, and Dennis B. Troup, "Software environment architectures and user interface facilities," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, 1988, pp. 697-708.

[2] A. N. Habermann and D. Notkin, "Gandalf: Software development environments," *IEEE transactions on software engineering*, vol. SE-12, no. 12, Dec. 1986, pp. 1117-1127.

[3] R. Kadia, "Issues encountered in building a flexible software development environment," *Fifth ACM SIFSOFT Symposium on Software Development Environments*, Virginia, USA, Dec. 1992, pp. 169-180.

[4] CCITT, "CCITT high level language (CHILL)," Recommendation Z.200, ISO/IEC 9496, 1995.

[5] K. Rekdal, "CHILL - The standard language for programming SPC system," *IEEE Transactions on Communications*, vol. 30, no. 6, 1982, pp. 24-28.

[6] CCITT, "CCITT specification and description language (SDL)," Recommendation Z.100, COM X-R 17-E, May 1992.

[7] CCITT, Message Sequence Charts (MSC), Recommendation Z.120, Geneva, 1992.

[8] J. H. Heilesen, B. Michael, B. Renard, "Conformance testing of SDL support tools," *SDL '93 Using Objects*, O. Faergemand and Amardeo Sarma, Ed., North-Holland, 1993, pp. 411-426.

[9] Per Blysa, "SDT: The SDL design yool," *SDL '93 Using Objects*, O. Faergemand and Amardeo Sarma, Ed., North-Holland, 1993, pp. 513-514.

[10] Vincent Encontiere, "GEODE," *SDL '93 Using Objects*, O. Faergemand and Amardeo Sarma, Ed., North-Holland, 1993, pp. 501-503.

[11] G. Amsjo, A. Nyeng, "SDL-based software development in siemens A/G - Experience of introducing rigorous use of SDL and MSC," *7th SDL Forum*, Oslo, September 1995, pp. 339-348.

[12] Kristen Rekdal, "CHILL - The international standard languages for telecommunications programming," *Telektronikk*, vol. 89, no. 2/3, 1993, pp. 5-10.

[13] N. Sato, K.Ohmori, "Construction of CHILL environment using generally available tools," *5th CHILL Conference*, Rio de Janeiro, Brazil, Mar. 1990, pp. 145-152.

[14] CHIPSY Reference Manual, Version 15.020, Kvatro Telecom AS, 1994.

[15] J. F. H. Winkler *et al.*, "Object CHILL - An object-oriented language for telecom applications," *XIV International Switching Symposium*, Yokohama, Japan, Oct. 1992, vol. 2, pp. 204-208.

[16] H. G. Park, "Narrowband and broadband ISDN technology implementation by TDX switching system," *International Symposium on Telecommunications*, Slovenia, Oct. 1992, pp. 119-122.

[17] J. P. Hong *et al*., "Integrated SDL environment," in *SDL '89: The Language at Work*, O. Faergemand and M. M. Marques, Ed., North-Holland, 1989, pp. 117-126.

[18] J. P. Hong, DongGill Lee, K. S. Park, and H. G. Bahk, "A CHILL programming environment," *5th CHILL Conference*, Rio de Janeiro, Brazil, Mar. 1990, pp. 166-174.

[19] E. H. Paik *et al*., "Debugging and simulation environment for a Switching System," *Pacific Telecommunications Conference*, Hawaii, Jan. 1994, pp. 642-646.

[20] K. S. Park, "Simulation environment for telecommunication systems," *Conference on communication technology*, China, June 1994, pp. 1387-1391.

[21] Chan J. Chung *et al.*, "Using SDL in switching system development," in *SDL '89: The Language at Work*. O. Faergemand and M. M. Marques, Ed., North-Holland, 1989, pp. 377-386.

[22] Hwan C. Kimvol *et al*., "The automated verification of SDL specifications using numerical petrinets," *SDL '91: The Language at Work*. O. Faergemand *et al*., Ed., North-Holland, 1991, pp. 83-94.

[23] Hong-Keun Kim *et al.*, "Data modeling and implementation of repository for SDL '92 environments," *Changha International CASE Symposium '95*, China, Oct. 1995, pp. 122-127.

[24] Andrew S. Tanenbaum *et al.*, "A practical tool kit for making portable compilers," *Communications of the ACM*, vol. 23, no. 9, 1983, pp. 654-660.

[25] Walter F. Tichy, "Smart recompilation," *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 3, 1986, pp. 273-291.

[26] S. H. Kim *et al.*, "Source-Level debugging CHILL programs," *5th CHILL Conference*, Rio de Janeiro, Brazil, Mar. 1990, pp. 227-236.

[27] Jun-Cheol Park *et al.*, "A debugger for CHILL programs in a cross-development environment," *5th CHILL Conference*, Rio de Janeiro, Brazil, Mar. 1990, pp. 211-215.

[28] C. McDowell and D. Helmbold, "Debugging concurrent programs," *ACM Computing Surveys*, vol. 21, no. 4, Dec. 1989, pp. 593-622.

[29] Eun-Hyang Lee *et al.*, "An implementation of time manager for CHILL run-time system," *'92 The Korea Information Science Society Conference*, vol. 19, no. 2, Korea, Oct. 1992, pp. 531-534.

[30] A. J. Albrecht, and J. E. Gaffney, "Software function, source lines of code and development effort prediction," *IEEE Transactions on Software Engineering*, vol. SE-9, no. 6, Nov. 1983, pp. 639-648.

[31] Walston, C, and C. Feiix, "A method for programming measurement and estimation," *IBM Systems Journal*, vol. 19, no. 1, 1977, pp. 54-73.

[32] Roger S. Pressman, *Software Engineering - A Practitioner's Approach*. McGraw-Hill, 1992.

[33] F. Mynatt, *Software Engineering with Student Project Guide*. Prentice-Hall, 1990.

[34] J. L. Lawrence, "The RC2000: A software success story," *ACM Sigsoft Software Engineering Notes*, vol. 10, no. 1, Jan. 1985, pp. 31-42.

[35] H. Gomaa, "A software design method for real-time systems," *Communications of the ACM*, vol. 27, no. 9, Sep. 1984, pp. 938-949.

[36] David Kung, Jerry Gao *et al*., "Developing an object-oriented software testing and maintenance environment," *Communications of the ACM*, vol. 38, no. 10, Oct. 1995, pp. 75-86.

DongGill Lee received the B.S. degree in Electronic Engineering from Kyoungbuk National University, Korea in 1983 and the M.S. degree and Ph.D degree in Computer Science from Korea Advanced Institute of Science and Technology in 1985 and 1993, respectively. He joined ETRI in 1985 and engaged in developing Software Development Environment for telecommunications systems from 1985 to 1993. Currently he is working toward development of an Object-oriented Software Development Environment for telecommunications systems. His interesting research area includes programming language, compiler construction, and software engineering.

JoonKyung Lee graduated from Sogang University with a B.S. degree in physics 1985 and from Soongsil University with M.S. and Ph D. degrees in computer engineering 1987 and 1997. Since 1987 he has worked for the development of CHILL programming environments at ETRI. His research interests include programming languages, compilers and object-oriented system software, especially in the field of telecommunicational switching system.

Wan Choi received the B.S. degree in electronics engineering from Kyungpook National University, Korea, in 1981, and the M.S. degree in computer science from Korea Advanced Institute of Science and Technology, Korea, in 1983. He has obtained the professional engineer in 1988. He has been engaged in CHILL/SDL Software Development Environment Project as the project leader. His research interests include software engineering, compiler design, and computer aided software engineering.

Byung Sun Lee received the B.S. degree in Mathematics from Sungkyunkwan University in 1980 and the M.S. degree in Computer Science from Dongguk University in 1982. He is currently working toward the Ph.D. degree at Korea Advanced Institute of Science and Technology, Taejon, Korea. He joined ETRI in 1982, and is currently a Head of Software Environments Section. His interests include formal methods, software development environments, software reliability and software fault tolerance for real-time systems, especially for telecommunication networks and switching systems.

Chimoon Han received the B.E. and M.E. degrees in electronics engineering from Kyungpook National University in 1977 and Yonsei University in 1983, respectively. He received the Ph.D. degree in electronics engineering from the University of Tokyo, Japan, in 1990. Since he joined Korea Institute of Science and Technology in 1977, he was involved in developing optical fiber communication and radio mobile communication systems until 1982. Since 1983, He has been involved in developing ISDN user-network interface system, LAN system, ATM switching system and wireless ATM PCS system in ETRI. He is currently the Director of Systems Technology Department in Switching Technologies Division, in ETRI. His research interests include the system architecture and its performance evaluation, system engineering, system design, and implementation for ATM switching systems and wireless ATM PCS systems.