SDL과 CHILL 개발 환경의 통합 방법

(An Integration Method of SDL and CHILL Developing Environment)

최 원 혁 † 이 동 길 †† 이 시 영 ††† 김 승 호 ††††

(Wonhyuk Choi) (DongGill Lee) (Si-Young Lee) (Sung-Ho Kim)

요 약 본 논문에서는 교환기와 같은 실시간 분산 시스템의 개발을 위한 SDL과 CHILL에 기반한 통합 소프트웨어 개발 방법이 제시된다. 이미 다양한 분야에서 SDL과 CHILL의 도구들이 각각 개발되어 왔지만, 본 논문에서는 이들을 통합할 수 있도록 SDL에서 CHILL로의 자동 코드 생성과 심벌 디버깅 환경에 중점을 둔다. 이는 완벽한 수행과 심벌 추적 정보의 삽입을 위한 자동 코드 생성 단계와 SDL 심벌 디버거와 CHILL 시험 환경을 이용한 실시간 심벌 디버깅 단계로 이루어지며, 첫째 SDL에 기반한 정형 명세, 둘째 설계의 검증과 확인, 셋째 자동 코드 생성, 넷째 분산된 실시간 병행 CHILL 프로그램의 디버깅, 그리고 마지막으로 응용 프로그램의 CHILL에 기반한 호스트 통합 시험을 제공한다.

Abstract In this paper, we present an integrated software developing method for real-time distributed system like switching system based on SDL and CHILL. As many SDL and CHILL tools are already developed in various application areas, we concentrate on automatic code generation from SDL to CHILL and symbolic debugging environment to integrate them : automatic code generation step for complete execution and insertion of symbolic tracing information, and run-time symbolic debugging step using SDL symbol debugger and testing environment for CHILL. Our approach can provide 1) formal specification based on SDL, 2) verification and validation of the design, 2) automatic code generation, 4) debugging of distributed real-time concurrent CHILL program, 5) simulation of application software for integrated testing on the host machine based on CHILL.

1. 서 론

통신 시스템과 같은 대형의 실시간 분산 시스템들의 개발은 긴 생명주기(life cycle)와 대규모 개발 인력을 필요로 한다. 이로 인해 시스템 개발 후의 유지보수에 들어가는 비용은 시스템의 초기 개발 비용보다 상대적으로 더 많은 투자를 필요로 한다.

따라서, ITU-T(International Telecommunication Union Telecommunication Standardization Sector)에서는 대형 시스템의 개발 시간과 비용을 줄이고 시스템의 질적 향상을 위해 생명주기의 각 단계에 따라 적용할 수 있는 언어로, 시스템의 명세(specification)와 기술(description)을 위해 SDL (Specification and Description Language)[1]을, 구현을 위해 CHILL (CCITT High Level Language)[2]을 권고하였다[3].

70년대 중반부터 쓰이기 시작한 SDL은 통신 산업 뿐만이 아니라 교환기 시스템과 같은 실시간 수행을 필요로 하는 모든 영역에서 널리 쓰이고 있고, 그 유용성을 입증 받았다. SDL을 통한 시스템의 명세는 명세 및 기술 문서로부터 소프트웨어의 정형적인 검증과 확인을 가능하게 하고, 이를 바탕으로 구현 프로그램의 자동 생성을 가능하게 한다. 또한, 시스템의 행위가 확장된 유한 상태 기계 형태로 기술되므로 프로세스 간의 상호작용이 중점이 되는 시스템을 설계하는데 적합한 언어

† 정 회 원 : 한국전자통신연구원 개방형플랫폼팀 연구원
whchoi@etri.re.kr
†† 비 회 원 : 한국전자통신연구원 개발환경연구실 팀장
dglee@etri.re.kr
††† 비 회 원 : 블루코드테크놀로지 인터넷기술팀 과장
sylee@bluecord.co.kr
†††† 종신회원 : 경북대학교 컴퓨터공학과 교수
shkim@bh.knu.ac.kr
논문접수 : 2000년 7월 27일
심사완료 : 2001년 3월 7일

이다. 이러한 이유로 실제적인 산업 분야에서는 SDL을 지원하는 도구들이 많이 개발되어 사용되고 있다.[4,5]

CHILL은 병렬 수행, 모듈과 구조적 프로그래밍, 분할 컴파일, 강력한 형의 검사, 자료의 추상화 능력을 제공하기 때문에 교환기와 같은 대형의 실시간 소프트웨어를 개발하는데 적합한 구현 언어이다. 많은 상업적인 개발자들은 각자의 CHILL 환경을 개발 했는데, 이러한 환경들을 통하여 CHILL은 이미 ATM 교환기 또는 ISDN 교환기와 같은 시스템을 개발하는데 사용되고 있다. [6,7]

SDL은 정형적인 명세와 그 명세의 검증과 확인에 강점을 가지고, CHILL은 실시간, 분산, 병렬성을 가지는 소프트웨어를 구현하는데 적합한 언어이므로, 두 환경의 통합은 교환기와 같은 시스템을 개발하는데 좋은 환경을 제공한다[8]. Telelogic과 Verilog 같은 상업적인 CASE 도구 개발 업체에서는 통신 및 실시간 분산 시스템의 개발을 위하여 SDL을 기반으로 하는 시스템 명세 및 기술의 생성에서부터 SDL 수준의 디버깅과 SDL에서 C나 C++로 수행코드의 생성까지 일관성을 유지할 수 있는 다양한 CASE 도구들을 제공하고 있다[9,10]. 그러나, 아직도 SDL과 CHILL을 통합할 수 있는 도구는 지원되지 않고 있다.

그러므로, 본 논문에서는 코드의 자동 생성과 심벌 디버깅을 통하여 Tau와 같은 SDL 환경과 TECH (Testing Environment for CHILL)와 같은 CHILL 환경을 통합할 수 있는 개발 방법을 제안한다. 제안된 개발 방법을 지원하기 위해서 새로운 CASE 도구로 SDL에서 CHILL로의 번역기와 SDL 심벌 디버거를 설계하고 구현한다.

ATM 교환기 시스템의 개발 시에 제안된 통합 개발 방법을 부분적으로 적용함으로써, 제안된 방법이 다음과 같은 장점을 가짐을 알 수 있었다. 제안된 통합 개발 방법은 첫째로 전체 개발 시간과 비용을 줄일 수 있고, 둘째로 개발자가 설계 수준에서 프로그램을 검증할 수 있을 뿐만 아니라, 셋째로 설계와 구현의 개념차이에서 오는 오동작을 줄일 수 있었다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 본 논문에서 제안한 SDL과 CHILL 환경의 통합방법의 전제가 되는 설계 중심의 개발 방법에 대해 기술한 후, 이를 위하여 제안한 SDL 심벌 디버깅 환경에 대해 설명한다. 3장과 4장에서는 SDL 심벌 디버깅 환경을 지원하기 위한 인터페이스 도구인 SDL에서 CHILL로의 번역기와 SDL 심벌 디버거에 대해서 기술한다. 그리고, 5장에서는 이러한 도구들의 실제적인 구현과 사용자 인터페이스, SDL 심벌 디버깅 환경의 동작에 대해서 설명하고, 마지막으로 6장에서 결론을 맺는다.

2. SDL과 CHILL 환경의 통합

본 장에서는 설계와 분리된 구현으로 인해 발생되는 문제점들로 인해 대두된 설계 중심의 개발 환경에 대해 알아보고, 이를 위하여 제안된 SDL과 CHILL 환경을 하나의 개발 환경으로 통합하는 SDL 심벌 디버깅 환경에 대해 설명한다.

2.1 설계 중심의 개발 환경

구현 중심의 소프트웨어 개발 방법은 명세 및 설계 단계와 일관성 없이 소프트웨어를 개발하기 때문에 사용자의 요구사항 변화에 대하여 민감하게 대처하기가 어려우므로 유지보수에 많은 비용이 들어간다. 이와 같은 이유로, 현재의 소프트웨어 개발은 구현 중심에서 설계 중심의 개발 방법으로 변하고 있다.

설계 중심의 개발 방법은 분석과 실행이 가능한 명세와 설계를 통한 구현 프로그램의 자동생성을 통하여 소프트웨어를 개발한다. 설계 수준에서 정형화된 명세를 검증함으로써 설계와 구현 사이의 불일치를 없애고, 사용자의 요구 사항에 민감하게 대처할 수 있으므로 개발 이후의 유지보수 비용을 줄일 수 있다.

이와 같은 설계 중심의 개발 방법을 위한 개발 환경은 설계와 구현 환경을 연결할 수 있는 새로운 인터페이스 도구들을 추가함으로써 이루어진다. 구현된 프로그램을 설계 수준에서 검증할 수 있는 디버깅 환경과 설계 문서에서 구현 언어로의 자동 번역기가 추가되는 인터페이스 도구들이다[11,12].

2.2 SDL 심벌 디버깅 환경

본 논문에서는 설계 중심의 개발 방법으로써 인터페이스를 이용하여 SDL과 CHILL의 개발 환경을 통합하는 방법을 제안한다. 본 논문에서 제안한 방법에서는 가장 널리 쓰이고 있는 상업적인 SDL 도구인 Telelogic Tau를 SDL 개발 환경으로 채택하고, CHILL 컴파일러와 TECH(Testing Environment for CHILL)[6]를 CHILL을 위한 개발 환경으로 채택한다. 두 개발 환경을 연결하는 인터페이스로 SDL에서 CHILL로의 번역기[9,12]와 SDL 심벌 디버거를 설계하고 구현한다. 이러한 통합된 개발 환경은 SDL 심벌 디버깅 환경(SDL symbol debugging environment)이라고 한다[8].

SDL의 개발 환경으로 Tau를 선택한 이유는 실시간 분산 시스템 소프트웨어의 시각화, 설계, 구현 및 검사 등을 통하여 시스템 개발의 생명주기 전체를 지원하는

환경이기 때문이다. 또한, Tau는 UML Suite, SDL Suite, TTCN Suite와 같이 프로젝트의 모든 단계를 위한 도구들을 제공한다. 더욱이 이러한 도구들은 Post Master라 불리는 통신 메커니즘을 통하여 통합되는데, 이를 통하여 SDL 편집기와 같은 Tau의 도구들에 쉽게 접근할 수 있다[13].

CHILL의 개발 환경으로는 실시간, 병렬, 분산 수행의 특징을 가지는 ATM 교환기 소프트웨어와 같은 통신 시스템을 개발하는데 용이한 CHILL 컴파일러와 TECH를 사용한다. TECH는 다중 프로세스의 수행 추적, 소스 코드의 나열, 중지점 설정, 프로그램 위치의 설정과 시험, 단일 단계 추적과 같은 전통적인 디버깅 기능을 제공한다. 또한, TECH는 교환기 소프트웨어를 위한 실시간 분산 병렬 디버깅, 강력한 명령어 체계, 시그널 송신 능력 그리고 시각적인 사용자 인터페이스를 지원한다 [14].

SDL과 CHILL, 두 환경 사이를 중계하기 위해서 SDL에서 CHILL로의 번역기, SDL 심벌 디버거와 같은 새로운 도구들이 개발된다. SDL에서 CHILL로의 번역기는 개념 변환과 직접 사상을 통하여 SDL-92의 시스템 명세를 CHILL 프로그램으로 변환한다. 번역기는 코드의 생성 뿐만이 아니라 SDL 편집기에 의해서 주어지는 SDL/GR의 주석 정보를 번역하여 CHILL 프로그램에서 함수의 형태로 삽입되는 심벌 정합 정보로 해석한다. 이는 SDL 심벌과 CHILL 프로그램의 수행을 정합하는 역할을 한다. SDL 심벌 디버거는 SDL 편집기에서 번역된 CHILL 프로그램의 수행을 SDL 심벌로 보여주도록 SDL 편집기를 제어하고, SDL 심벌 수준에서 수행되는 여러 가지 디버깅 작업들을 수행한다.

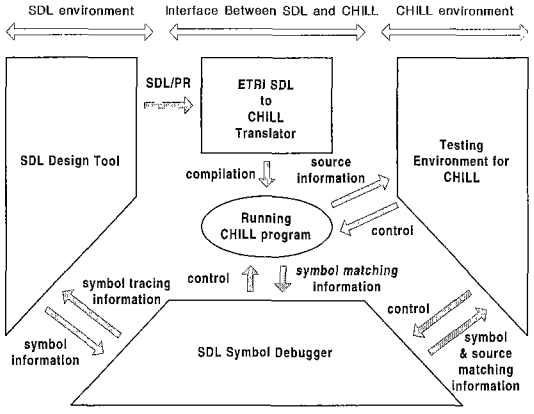


그림 1 SDL과 CHILL 환경의 통합을 위한 SDL 심벌 디버깅 환경

그림 1은 본 논문에서 제안한 SDL과 CHILL 개발 환경을 통합하는 SDL 심벌 디버깅 환경이다. 일반적으로 ATM 교환기 시스템의 개발 시에 시스템의 명세와 주요한 행위는 SDL에 의해서 기술되고, 세부적인 동작은 CHILL로 작성된다. 심벌 디버거를 위한 디버깅 정보는 SDL에서 CHILL로의 번역 시에 삽입되고, TECH를 위한 정보는 수행 프로그램의 번역 시간에 삽입된다. 그러므로, CHILL 프로그램의 수행은 심벌 디버거와 TECH에 의해 동시에 제어된다. 이러한, 심벌 디버거와 TECH의 동시 수행은 SDL 심벌 수준의 디버깅과 CHILL 소스 수준의 디버깅을 모두 지원할 수 있다.

3. SDL에서 CHILL로의 번역기

본 장에서는 2장에서 제안한 통합된 개발 환경인 SDL 심벌 디버깅 환경을 위한 두 가지의 인터페이스 도구 중의 하나인 SDL에서 CHILL로의 번역기의 구성과 번역의 기본 개념에 대해 살펴본다.

3.1 SDL에서 CHILL로의 번역기의 구성

본 논문에서 제안된 SDL에서 CHILL로의 번역기는 그림 2와 같이 전처리기(Preprocessor), 구문 분석기(*Syntax analyzer*), 구조 변환기(Structure transformer), 코드 생성기(Code generator)로 구성된다[9]. 각 모듈의 기능을 살펴보면, 전처리기는 다음 단계에서 그 변환을 용이하게 하기 위하여 매크로 확장, join 연산을 통하여 SDL/PR을 확장한다. 이 때에 SDL/PR의 주석이 해석되는데, 이 정보는 정합 인터페이스의 형태로 삽입된다. 구문 분석기는 확장된 SDL/PR의 구문을 분석해서 구조 트리(structure tree)와 전이 트리(transition tree)를 생성해 내고, 구조 변환기는 구조 트리와 전이 트리를 이용하여 CHILL 코드를 생성하기 위한 정보를 수집하고 코드 생성의 중간 형태인 정보를 축적한 구조 트리와 전이 트리로 변환한다. 마지막 단계인 코드 생성

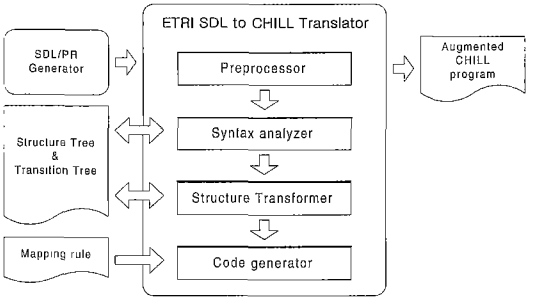


그림 2 SDL에서 CHILL로의 번역기의 시스템 명세

기는 구조 변환기에서 만들어진 정보로 정합 인터페이스를 포함하는 CHILL 수행 코드를 생성한다.

3.2 SDL에서 CHILL로의 번역의 방법

SDL에서 CHILL 코드로의 번역은 두 가지의 기본적인 개념인 개념 변환(conceptual transformation)과 직접 사상(direct mapping)을 통해 수행된다. 그리고 이러한 개념을 적용시키기 위해 기본 자료 구조인 구조 트리와 전이 트리를 생성하여 사용한다. 구조 트리는 원격 및 중첩된 SDL의 시스템, 블록, 프로세스 명세 구조들을 계층 구조로 변환한 것인데, CHILL로의 구조, 가시성 그리고 공유변수의 변환에 사용된다[9]. 전이 트리는 SDL의 프로세스의 상태 중심의 시그널 구조에서 전이에 해당하는 부분들을 트리 구조로 표현한 자료 구조이다. 이 때 전이는 <state, input signal> 형태로 표현된다[9]. 그림 3은 SDL/PR에 해당하는 구조 트리와 전이 트리의 예를 보여준다.

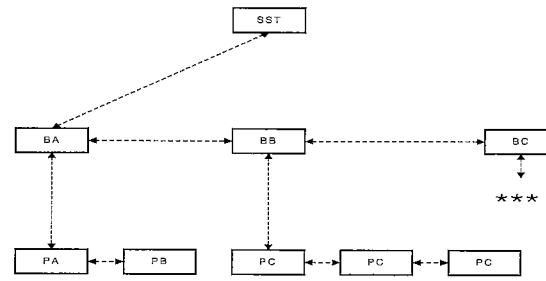
개념 변환은 SDL과 CHILL의 대응 요소의 의미적인 차이를 없애기 위해, 즉,상이한 대응 구조, 통신 방법, 내부 행위 기술, 가시성과 같은 개념들을 변환할 수 있도록 SDL의 구성요소의 구조 변환과 정보의 획득으로 이루어진다[9].

개념 변환이 수행된 후에 얻어지는 구조 트리와 전이 트리에 직접 사상 규칙을 적용시킴으로써 CHILL 프로그램이 생성된다[9]. 이러한 직접 사상 규칙은 구조, 형, 구문의 직접 사상으로 나누어진다.

구조의 사상(mapping of structure)은 SDL의 구조를 CHILL의 대응하는 구문 구조로 사상하는 것이다. 구조의 사상은 표 1에서 보는 바와 같이 기술된 대응 구조에서 가시성과 존재성과 같은 의미 보존에 중점을 둔다.

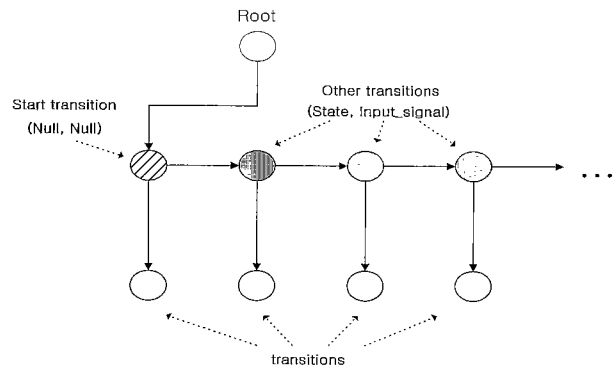
CHILL-96과 달리 CHILL은 구조적 언어이므로 SDL에서의 Block type이나 Process type과 같은 객체 지향 개념은 실례화(instantiation)를 통하여 처리된다.

```
system SST;
        block BA referenced;
        block BB referenced;
        block BC referenced;
endsystem SST.

block BA.
        process PA;
                ...
        endprocess PA
        process PB.
                ...
        endprocess PB;
endblock BA.

block BB.
        process PC;
                ...
        endprocess PC;
        process PD.
                ...
        endprocess PD.
        process PE;
                ...
        endprocess PE;
endblock BB.

block BC
        ...
endblock BC;
```



(가) 구조 트리

```
start ;
        Start transition  ;
state  WaitAllocated ;
  input AllAllocated ;
        transitions  ;
  input UnAllocated ;
        transitions  ;
state  Idle ;
  input DisplayOk ;
        transitions  ;
  input Display ;
        transitions  ;
        ......
```



(나) 전이 트리

그림 3 구조 트리와 전이 트리의 예

표 1 구조의 사상

SDL	CHILL	REMARK
System	Main module	
Block	Module	
Block type	Module	Instantiation
Process	Process	
Process type	Process	Instantiation
Service	Process	Merge to Process
Procedure	Procedure	
Signal	Signal	
Package	Instantiation	
Channel and Route	Grant/Seize	

형의 사상(mapping of type)은 개념상 대응하는 형들의 직접적인 사상을 중심으로 한다. 그러나, 문자열이나 자연수와 같은 길이나 크기에 제한을 가지지 않는 형들의 사상은 CHILL과 같은 구현 언어에서는 불가능하다. 이러한 형은 표 2에서 보듯이 최대값을 가지는 제한된 형으로 사상한다.

또한, CHILL 컴파일러가 실수에 대응하는 개념을 제공하지 않으므로 실수에 대한 사상은 제외된다. 언어적인 측면에서 SDL의 구문 구조는 CHILL의 부분 집합이 되므로 구문 구조의 사상(mapping of syntax)은 복잡한 변환이나 사상 규칙을 필요로 하지 않고 사상할 수 있다.

표 3은 주요한 구문의 사상 규칙의 예를 보여준다.

표 2 형의 사상

SDL	CHILL	REMARK
Boolean	BOOL	
Character	CHAR	
Charstring	ARRAY(1:MaxChar) CHAR	MaxChar = 255
Integer	INT	
Natural	RANGE(1:MaxInt)	MaxInt = 2311
Pid	INSTANCE	
Duration	RANGE(1:MaxInt)	MaxInt = 2311
Time	SROS API	
Syntype	SYNMODE	
Newtype	NEWMODE	

표 3 구문의 사상

Syntax	SDL	CHILL
SYSTEM	SYSTEM My_system; … … …	main : MODULE … … …; END main;
PROCESS	ENDSYSTEM; PROCESS My_P(2,); …… ENDPROCESS;	My_P: PROCESS(); … … … ; END My_P; … … …; START My_P(·); START My_P(·);
Declaration	DCL my_num INTEGER;	DCL my_num INT;
Creation	CREATE My_P(·);	START My_P(·);
Decision	DECISION My_var 1; (p) : …; (q) : …; … … … ENDDECISION;	IF ((My_var-1)=p) THEN …; ELSIF((My_var-1)=q) THEN …; … … …; FI;
Task	TASK my_val:=10 * 5;	my_val:=10 * 5;

4. SDL 심벌 디버거

본 장에서는 3장에서와 마찬가지로 인터페이스 도구들 중의 하나인 SDL 심벌 디버거의 구성과 기능 그리고, 심벌 디버거와 통합된 개발 환경에서 중요한 기능을 가지는 인터페이스의 설계와 구현에 대하여 설명한다.

4.1 SDL 심벌 디버거의 구성과 기능

SDL 심벌 디버거는 자동 생성된 CHILL 프로그램의 수행을 TECH와 사용자의 인터페이스간의 통신을 거쳐 SDL 편집기를 통하여 시각적으로 보여준다. SDL 심벌 디버거의 구성은 그림 4과 같이 네 개의 프로세스로 구성된다.

수행 중인 CHILL 프로그램 내의 하나의 프로세스와 SDL 심벌 디버거의 실시간 통신기 프로세스 하나를 연결하는 기능을 수행하는 연결 관리기(Connection Manager), 실행중인 CHILL 프로그램의 프로세스로부터 정합 인터페이스 정보를 수신하고, 이를 해석하여 추적 인터페이스로 변환한 후 외부 인터페이스로 전송을 담당하는 실시간 통신기(Runtime Communicator), 실시간 통신기로부터 수신된 추적 인터페이스를 PostMaster로 전송함으로서 심벌 추적을 제어하는 외부 인터페이스(External Interface)와 사용자의 입력을 받아 제어 인터페이스를 통하여 SDL 심벌 디버거의 각 기능들을 제어하는 인터페이스 관리기(Interface Manager)로 구성된다.

SDL 심벌 디버거의 각 기능은 첫째로 가장 기본적인

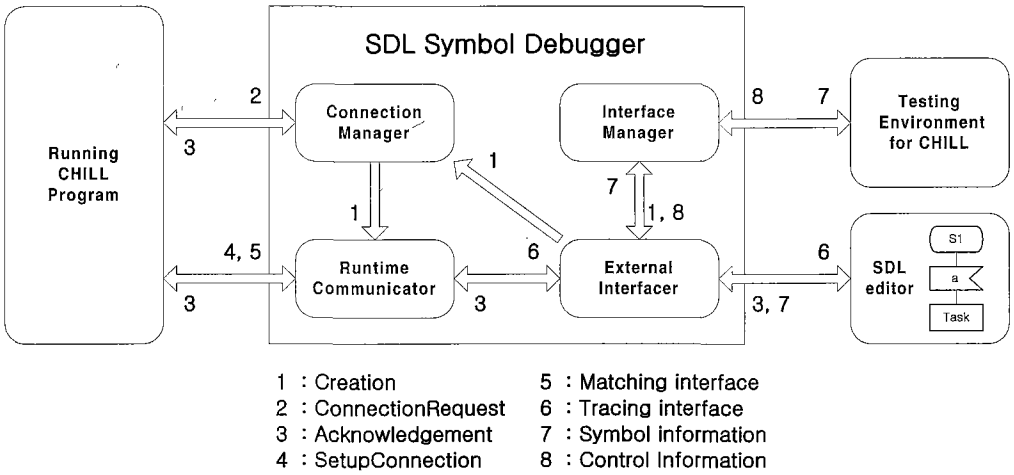


그림 4 SDL 심벌 디버거의 구성

기능인 생성된 CHILL 프로그램의 수행을 SDL 심벌의 형태로 시각화 해주는 SDL 심벌 수준 실행 추적(Symbol level execution trace), 둘째로 심벌의 단계적인 수행을 가능케 하는 SDL 심벌 수준 단일 스텝 추적(Symbol level single step trace), 셋째로 중지점 설정을 통해 부분 디버깅을 가능하게 하는 SDL 심벌 단위 중지점 설정(Symbol level break point establishment and resume) , 넷째로 TECH와의 통신을 통하여 생성된 코드 수준에서 디버깅을 할 수 있게 하는 SDL 심벌에서 CHILL 소스 코드로의 참조(Symbol to source reference), 마지막으로 CHILL 소스 코드에서 SDL 심벌로의 참조(Source to symbol reference)로 이루어진다.

4.2 인터페이스의 설계

SDL 심벌 디버거의 구현에서 가장 중요한 개념은 세 가지 인터페이스의 설계와 적용이다. 이들 인터페이스는 심벌 디버거 내의 정보의 흐름 뿐만이 아니라, SDL 심벌 디버깅 환경을 하나로 통합하게 해주는 역할을 한다. 세 가지의 인터페이스 중 첫번 째는 SDL에서 CHILL로의 번역시 변환된 CHILL 코드 블록에 정합하는 심벌 정보를 나타내는 정합 인터페이스이고, 두번 째는 수행중인 CHILL 프로그램이 전송한 정합 인터페이스를 해석하여 실시간으로 SDL 심벌 추적을 위해 사용되는 추적 인터페이스이다. 마지막으로, 사용자의 입력을 받아 심벌 추적의 기능을 제어함으로써 심벌 디버거의 다양한 기능을 제공하게 하는 제어 인터페이스이다.

4.2.1 정합 인터페이스

정합 인터페이스는 CHILL 프로그램의 생성시에 생성된 코드 블록과 정합하는 SDL 심벌의 이름과 위치 정보를 기록한 것이다. 이런 정합 인터페이스는 번역된 CHILL 프로그램의 수행시에 심벌 추적을 위해 삽입된 정합 인터페이스 함수에 의해 시그널 형태로 심벌 디버거로 전달된다. 심벌 디버거로부터의 반응 대기는 정합 인터페이스 함수의 호출 후, 코드 블록의 수행 전에 이루어진다.

정합 인터페이스의 생성은 그림 2에서 보듯이 SDL에서 CHILL로의 번역시 전처리기에 의해 생성되며, SDL 편집기에서 SDL/PR의 생성시에 SDL/GR정보의 유지를 위해 주석으로 삽입한 SDT Reference[13]를 해석하여 이루어진다. 생성되는 정합 인터페이스는 SDL 기술 중에서 프로세스에 포함된 부분에만 한정되며 시스템과 블록의 기술에 관련된 부분은 심벌 추적에 관여하지 않으므로 포함하지 않는다.

정합 인터페이스는 표 4와 같이 세 부분으로 나누어진다.

• 프로세스의 시작

프로세스의 시작은 SDL 프로세스 기술에 해당하는 새로운 인스턴스가 생성됨을 말한다. 같은 프로세스 기술로부터 생성된 인스턴스들이라도 서로 다른 수행 흐름을 가지므로, 각각의 인스턴스마다 심벌 디버거에게 자신의 시작을 알리고 자신의 수행 흐름에 대한 정보를 전송해야 한다.

• 심벌의 수행

심벌 수행의 정보는 SymbolInformation 시그널에 의

해 전송된다. 정합 인터페이스의 대부분을 차지한다.

• 프로세스의 종류

STOP에 의해 수행을 종료하는 프로세스는 자신의 종료를 해당 실시간 통신기에 통보한다.

표 4 정합 인터페이스

Matching interface		Function
Start of Process	ConnectionRequest	Request to Connection Manager for connection
	SetupConnection	Transmit connection information to Runtime Communicator
Execution of symbol	SymbolInformation	Transmit execution information of symbol to Runtime Communicator: Object ID, Row position, and Column position of symbol
Termination of process	TerminateInstance	Notification of process termination to Runtime Communicator

4.2.2 추적 인터페이스

추적 인터페이스는 그림 5와 같이 심벌 디버거가 실행 프로그램으로부터 전송된 정합 인터페이스를 SDL 심벌 추적 정보로 변환한 것이다. 변환되어진 추적 인터페이스는 SDL 편집기의 PostMaster를 거쳐 SDL 편집기로 전송됨으로써 현재 실행 중인 코드에 해당하는 심벌을 보여 주게 된다.

그림 4에서 보듯이 수행 중인 CHILL 프로그램의 SymbolTraceStart 함수는 ConnectionRequest와 Setup Connection을 CHILL 프로그램과 심벌 디버거 사이의 통신을 초기화하기 위해 심벌 디버거로 보낸다. 이러한 정합 인터페이스는 LoadDiagram과 LoadFile이라는 추적 인터페이스로 변환된다. 표 5는 정합 인터페이스에서 추적 인터페이스로의 변환을 보여준다. 표에서도 알 수 있듯이 SDL 편집기의 초기화는 어떠한 정합 인터페이스도 요구하지 않으므로 CHILL 프로그램의 수행과는 상관 없이 심벌 디버거를 통하여 이루어진다.

표 5 프로세스들에 의해 번역된 추적 인터페이스

Tracing interface	Runtime Communicator	External interfacer
Initialization		SetupPostMaster StartSDLEditor
Start of Process	LoadDiagram	LoadDiagram ShowDiagram
Execution Of symbol	ShowSymbol	ShowDiagram SelectObject
Termination of Process	TerminateInstance	UnloadDiagram EndPosterMaster

외부 인터페이서에 의해 변환된 추적 인터페이스는 SDL 편집기와의 통신을 위해서 Telelogic AB에서 제공하는 SPSendToTool과 SPRead라는 두 가지의 라이브러리 함수를 포함 한다[13]. SPSendToTool은 SDL 편집기로 각각의 추적 인터페이스에 해당하는 요청을 전달하고, SPRead는 그에 대한 응답정보를 반환한다. 또한, 이 정보는 통신 중의 에러 복구에도 사용된다.

4.2.3 제어 인터페이스

제어 인터페이스는 인터페이스 관리기가 사용자 인터페이스로부터 사용자의 입력을 받아 심벌 추적의 기능을 바탕으로 심벌 디버거의 다양한 기능을 제공하는 역

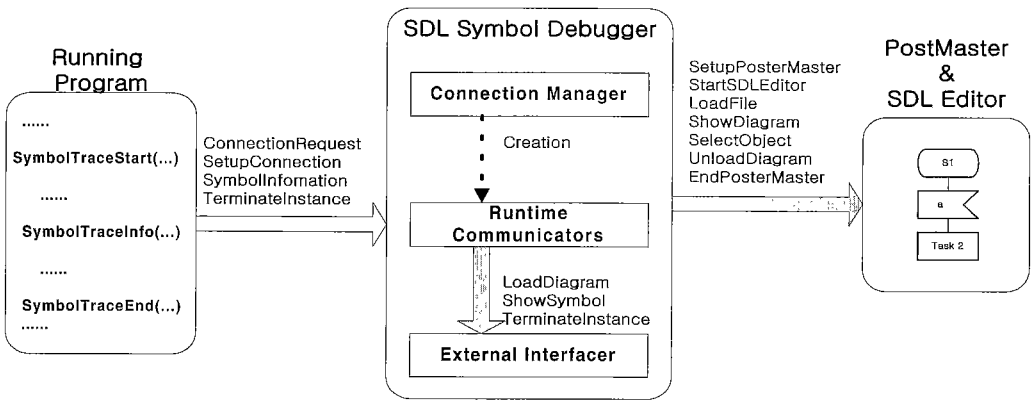


그림 5 정합과 추적 인터페이스의 번역

할을 한다. 이러한 제어 인터페이스는 외부 인터페이서가 PostMaster로부터 수신한 응답 정보를 실시간 통신기로 반환하는 것을 실시간에 제어함으로써 SDL 편집기에서 심벌의 단일 스텝 추적, 중지점 설정 등의 다양한 디버깅 기능을 제공한다.

4.2.4 인터페이스 접근 방법의 구성과 장점

그림 6은 SDL 심벌 디버거의 라이브러리 구성을 보여준다. 이러한 라이브러리는 CHILL 언어로 프로그래밍 되어 있으므로, Telelogic AB에서 제공하는 C 언어로 작성된 함수와 중계할 수 있는 인터페이스의 명세 파일을 필요로 한다. 하지만, 이렇게 CHILL 언어로 라이브러리를 작성한 이유는 수행 중인 CHILL 프로그램들의 통신을 용이하게 하기 위해서이다. 만약 인터페이스들이 C 언어로 작성되어졌다면 번역된 CHILL 프로그램안에 C 언어를 위한 중계 루틴이 필요 할 것이다. 이는 통신상에서 엄청난 과부하로 작용할 것이다.

본 논문에서 제시한 것처럼 정합과 추적의 인터페이스를 이용한 통합 방법이 가지는 장점은 인터페이스들이 SDL과 CHILL 환경에 독립적으로 설계되었으므로, SDL에서 CHILL로의 번역기 이외에 다른 예를 들어, C나 C++, 또는 JAVA 언어로의 번역기를 통한 개발 환경의 통합시에 본 논문에서 설계한 인터페이스를 직접적으로 적용할 수가 있다.

5. 사용자 인터페이스와 운용

SDL 심벌 디버거는 단지 CHILL 프로그램의 수행에 의하여 동작하므로, 연계 수행되는 TECH와 독립적으로 동작할 수 있다. 하지만, CHILL 프로그램의 흐름을 제어할 수 있는 TECH는 심벌 추적의 흐름에 영향을 미칠 수 있다. 그래서, TECH의 디버깅 과정에서 발생하는 모든 행위는 심벌 추적의 행위에 반영된다. 하지만, 그 반대의 경우 심벌 디버거의 행위가 TECH에게 는 성립되지 않는다. 심벌 추적이 심벌 디버거에 의해 제어될 때, 심벌에 해당하는 소스 코드 수준의 제어는 가능하지 않다. 이는 SDL 심벌에서 CHILL 소스코드로의 참조의 기능이 심벌 디버거와 TECH사이의 통신 방법의 문제로 아직 구현되지 않았기 때문이다.

그림 7은 SDL에서 CHILL로의 번역과 사용자 인터페이스를 보여준다. 그림의 좌상단은 번역 되어야 할 SDL/PR을 보여준다. 우상단은 번역된 CHILL 코드를 보여주는데, 이는 파일로 저장된다. 그리고, 하단 영역은 번역과정의 상태 정보를 출력한다. SDL에서 CHILL로의 번역기의 실제 모듈은 C언어로 작성되었으며 사용자 인터페이스는 Tcl/Tk를 이용하여 구성하였다. 이런 C 모듈과 Tcl/Tk로 작성된 사용자 인터페이스 사이의 연결은 C언어로 미리 작성된 통신 모듈을 통해 해결 하였다.

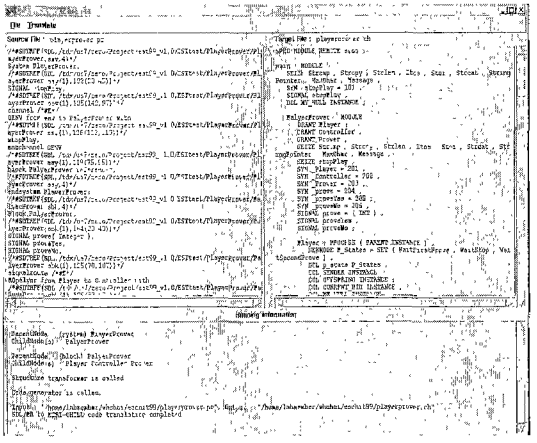


그림 7 SDL에서 CHILL로의 번역과 사용자 인터페이스

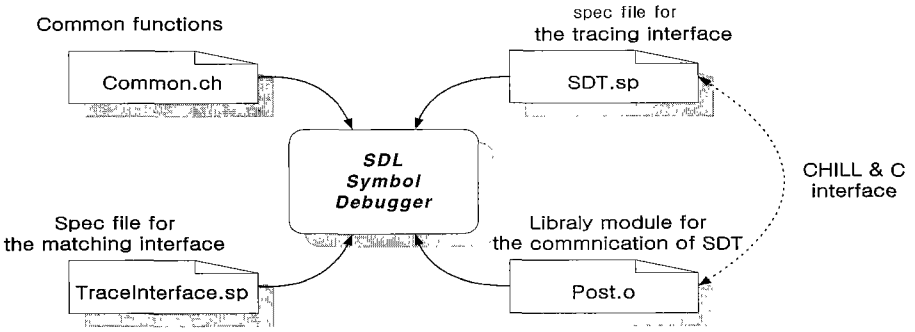


그림 6 SDL 심벌 디버거의 라이브러리 구성

그림 8은 SDL 심벌 디버거의 사용자 인터페이스를 보여준다. 그림의 좌상단 영역은 정합과 추적 인터페이스에 의한 내부 동작과 상태를 보여주고, top 패널과 우상단은 사용자의 인터페이스를 위한 패널이다. 이 패널을 통하여 사용자는 SDL 심벌 디버거의 각 기능의 수행을 제어할 수 있다. 하단은 SDL 편집기를 제어한 결과를 보여준다. 심벌 디버거의 실제적인 모듈은 CHILL 언어로 작성되었고 사용자 인터페이스는 SDL에서 CHILL로의 번역기와 마찬가지로 Tcl/Tk를 이용하였다. 마찬가지로 두 모듈의 통신은 C로 작성된 통신 모듈을 통해 해결하였다.

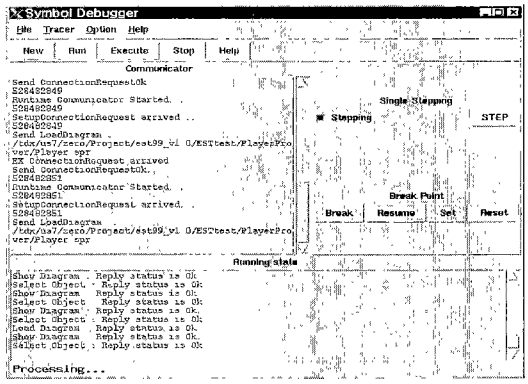


그림 8 SDL 심벌 디버거의 수행과 사용자 인터페이스

그림 9는 그림 8과 함께 SDL 편집기를 통하여 실제적으로 SDL 심벌 단일 추적의 수행 상황을 보여준다. Telelogic AB가 제공하는 SDL 편집기가 심벌 추적을 시각화한다 하더라도 SDL 편집기의 실제적인 제어는 심벌 디버거를 통하여 이루어진다.

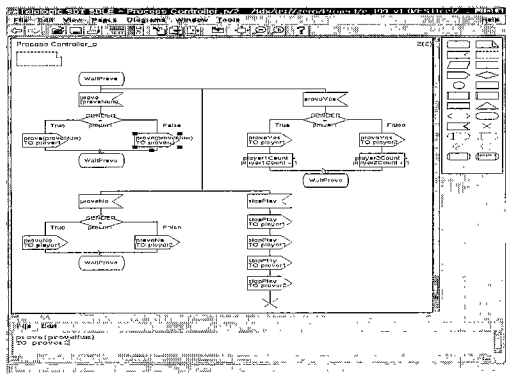


그림 9 SDL 편집기에서의 심벌 추적

6. 결 론

현재의 소프트웨어 개발은 구현 중심에서 설계 중심의 개발 방법으로 변하고 있다. 설계 중심의 개발 방법은 설계 문서에서 자동 생성된 구현 프로그램을 통하여, 설계 수준에서 정형화된 명세를 검증함으로써 설계와 구현 사이의 일관성을 가진다. 그러므로, 개발 이후의 유지보수가 효율적이다. 설계 중심의 개발 방법을 구축하기 위한 방법은 설계 환경과 구현 환경 사이를 연결하는 설계 수준에서 설계 문서를 검증할 수 있는 디버깅환경과 설계 문서에서 구현 언어로의 자동 번역기를 통해 이루어진다.

SDL과 CHILL은 교환기와 같은 통신 분야의 시스템 개발에서 쓰여지기 시작했고, 이제는 실시간 병렬 분산 시스템을 필요로 하는 다른 분야에서도 사용되고 있다. 이러한 상황에도 불구하고, 두 환경을 연계하는 방식은 단지 부분적인 코드의 자동 생성 수준에만 머물러 있다. 그래서 아직 두 환경이 통합되었을 때의 장점을 최대한으로 얻을 수 있는 강력한 통합 방법이 제시되고 있지 않다.

그래서, 본 논문에서는 교환기 시스템의 개발을 위한 SDL과 CHILL 환경의 통합 방법이 제안되었다. 이러한 통합은 정합 인터페이스와 추적 인터페이스를 통하여 두 환경을 중계하는 방식으로 이루어졌다. 두 환경의 중계를 위해서 SDL에서 CHILL로의 번역기와 SDL 심벌 디버거라는 새로운 도구를 설계하고 구현하였다.

본 논문에서 제안된 두 환경의 통합 방법이 ATM 교환기 시스템의 개발에 부분적으로 적용되고 구현되었을 때, 우리는 자동 번역과 심벌 및 소스 수준의 디버깅이 가지는 이득을 알 수 있었다.

첫째, SDL에서 CHILL로의 자동 번역을 제공하기 때문에, 설계와 구현 사이의 차이점을 최소화 할 수 있었다. 물론, 자동 번역된 프로그램의 코드가 개발자가 직접 작성한 코드보다 5 ~ 15% 정도 증가했지만, 이러한 코드의 증가는 심각한 문제를 유발시키지 않았다. 왜냐하면, 번역은 직접 사상에 의해 수행되기 때문에, 증가된 부분은 단지 SDL의 선언부와 사용되지 않는 내부 변수의 배정으로 국한되었다. 그러므로, 증가된 코드로 인한 수행 시간의 초과 정도는 무시될 수 있었다. 둘째, SDL 심벌 수준에서 CHILL 프로그램의 수행을 검사할 수 있으므로, 개발자가 설계의 관점에서 번역된 목적언어의 올바른 동작을 검증할 수 있었고, 설계와 구현의 개념차이에서 오는 오동작을 찾아 낼 수 있었다. 셋째, 심벌 디버거에서 제공되는 디버깅 방법은 TECH와 연

계되어 사용될 수 있으므로 심벌과 소스의 디버깅이 동시에 이루어질 수 있다. 넷째, 우리가 제안한 방법은 정보의 해석과 각 구성요소 사이의 통신을 통하여 수행되므로 주어진 SDL과 CHILL 환경의 기능을 수정 할 필요가 없었다. 마지막으로, 정합 인터페이스와 추적 인터페이스가 SDL과 CHILL에 독립적으로 구현되었으므로, 이러한 인터페이스들은 새로운 언어를 위한 번역기를 구현 할 때 쉽게 재사용될 수 있다.

참 고 문 헌

[1] ITU Recommendation Z.100, Specification and Description Language(SDL), 1996.

[2] ITU Recommendation Z.200, CCITT High Level Language(CHILL), 1996.

[3] 최완 ,박항구, 홍진표, 강석열, 송영기, 최고봉, 김환철, 신영승, 조준희, 이근구, 정찬진, SDL환경 : TDX-10 총서 제 6권, pp. 1-11, 한국전자통신연구소, 1994.

[4] P. Blysa, SDT: The SDL design tool, The sixth SDL Forum, SDL '93: Using Objects, pp.513-514, 1993.

[5] R. R. Singh, and J. Serviss, "Code generation Using GEODE: A CASE Study," The ninth SDL Forum, SDL'97: TIME FOR TESTING- SDL, MSC and Trends, pp.539-550, 1997.

[6] E. H. Paik, J. H. Chang, D. G. Lee, and K. P. Jun, A Remote Debugging System for Distributed CHILL Programs, Proceedings of SCI/ISAS'99, pp.129-132, 1999.

[7] N. Sato and K. Ohmori, Construction of CHILL environment using generally available tools, The fifth CHILL conference, pp.145-152, 1990.

[8] D. G. Lee, J. K. Lee, W. Choi, B. S. Lee, and C. M. Han, "A New Integrated Software Development Environment Based on SDL, MSC, and CHILL for Large-scale Switching Systems," ETRI journal, Vol.18, No.4, 1997.

[9] 이시영, 개념 변환과 직접 사상을 이용한 SDL-92에서 CHILL-96으로의 자동 번역, 경북대학교 박사학위논문, Dec., 1999

[10] 원준석, SDL 심벌 추적기의 설계 및 구현, 경북대학교 석사학위논문, Dec., 1999.

[11] 이병선, 이동길, 권용래, 설계 중심의 교환기 소프트웨어 개발방법, Communication Software98, pp. 127-130, 1998.

[12] S. Y. Lee, D. G. Lee, J. K. Lee, and S. H. Kim, "Conceptual Transformation from SDL- 92 to CHILL-96 using Signal Subordination," The sixth International Conference on Real- Time Computing Systems and Applications, pp. 484-487, 1999.

[13] Telelogic AB., Telelogic Tau 3.5 Manual, 1998.

[14] W.H. Choi, D.G. Lee, S.Y. Lee, S.H. Kim, "An Integration Method of SDL and CHILL Environment, "International Conference on Telecommunications (ICT2000), Vol.1, pp.448-452, 2000.

최 원 혁

1999년 경북대 컴퓨터공학과 졸업. 2001년 경북대 컴퓨터공학과 석사. 2001년 2월 ~ 현재 한국전자통신연구원. 개방형 플랫폼팀 연구원. 관심분야는 소프트웨어 공학, 미들웨어, Active 네트워크 등임

이 동 길

1983년 경북대학교 전자공학과 졸업. 1985년 한국과학기술원 전산학 석사. 1994년 한국과학기술원 전산학 박사. 1985년 ~ 현재 한국전자통신연구원 책임연구원으로 근무. CHILL 컴파일러 및 프로그래밍 환경 개발에 참여하였으며 현재는 객체지향 병행처리 소프트웨어 프로그래밍 환경 개발에 참여하고 있음. 관심분야는 컴파일러 구성론, 프로그래밍 언어론, 실시간 객체지향 병행처리용 소프트웨어 개발환경.

이 시 영

1995년 경북대 컴퓨터공학과 졸업. 1997년 경북대 컴퓨터공학과 석사. 2000년 경북대 컴퓨터공학과 박사. 1997년 ~ 2000년 2월 한국전자통신연구원 S/W 개발환경 위축연구원. 2000년 2월 ~ 현재 블루코드 테크놀로지(주) 인터넷 기술팀 과장. 관심분야는 통신 프로토콜, 소프트웨어 공학 등임

김 승 호

1981년 경북대학교 전자공학과 졸업(공학사). 1983년 한국과학기술원 전산학과 졸업(공학석사). 1994년 한국과학기술원 전산학과 졸업(공학박사). 1985년 ~ 현재 경북대학교 컴퓨터공학과 교수.