ETRI CHILL-96 컴파일러의 설계와 구현
(Design and Implementation of the ETRI CHILL-96 Compiler)

김 상 은 † 이 준 경 †† 이 동 길 †
(SangEun Kim) (Joon-Kyung Lee) (DongGill Lee)

요 약 CHILL 언어는 전전자 교환기 및 통신 시스템용 소프트웨어 개발에 사용되어왔다. ETRI CHILL-96 언어는 객체지향성, 병행성, 포괄적 타입과 같은 특성을 가지고 CHILL 언어를 확장한 것이다. 본 논문에서는 ETRI CHILL-96 컴파일러의 설계 및 구현과 관련된 몇 가지 기술적 주제들에 대해 다룬다. 재명명 변환 규칙들과 함수 재구성 기술들은 이름 충돌의 방지와 풍부한 디버깅 정보의 생성을 위해 사용되었다. 이러한 새로운 확장된 기능들은 확장된 CHILL 중간 코드로 컴파일 및 변환되는 단계에 전처리 되어진다. 이러한 컴파일 기술들은 ETRI CHILL-96언어가 기존의 CHILL 언어에 의해 개발된 소프트웨어들과 호환성을 유지할 수 있게 해 준다.

Abstract CHILL language has been used for the software development of electronic switching and telecommunications system. ETRI CHILL-96 language is an extended CHILL language with the notions of object-orientation, concurrency, and generic type. In this paper, we discuss some design and implementation issues of ETRI CHILL-96 compiler. Renaming translation rules and function restructuring techniques are adapted for the purpose of preventing name conflict and producing debugging information. Those new extended features are preprocessed in the compilation and translated to extended CHILL intermediate codes. Such compilation technique enables ETRI CHILL-96 language to hold compatibility with software developed by CHILL language.

1. 서 론

1996년 ITU-T(구 CCITT)는 기존의 표준 언어인 CHILL을 확장하여 교환기 및 통신 시스템용 객체지향 소프트웨어 개발을 위한 새로운 국제 표준 구현 언어로 CHILL-96을 권고하였다[1]. 1980년대에 CCITT에 의해 처음으로 권고된 CHILL언어는 교환기 시스템 소프트웨어를 개발하기위한 표준 고급 구현 언어로 널리 사용되었다. 이 언어는 교환기와 같은 특수한 응용 분야에 적합한 다양한 언어적 특징들을 가지고 있는데 병행성(이벤트, 버퍼, 임계, 프로세스 등), 고신뢰 소프트웨어를 위한 강한 타입 점검 기능, 교환기 및 통신용 시스템 소프트웨어의 개발을 위한 실시간 기능 지원을 위한 시간 관리 기능 등과 같은 특징들을 지닌다. CHILL-96언어는 기존의 CHILL 언어에 객체지향성, 확장된 실시간 기능, 확장된 병행성 기능과 같은 특징들이 확장되었다.

최근에 와서 많은 교환기 회사들이 자신만의 독특한 객체지향적 방법론을 적용하여 교환기용 소프트웨어들을 개발하고 있다[2,3,4]. 이 방법론을 적용할 때 발생하는 장점과 성능 측정 및 분석 결과를 통해 교환기 및 통신 시스템용 소프트웨어 개발에 있어서 객체지향 방법론의 적용 가능성을 찾아볼 수 있고, 프로그램 구조의 단순화, 프로그램 논리의 이해도 향상, 대용량 소프트웨어에 대한 재사용성과 유지 보수의 용이성 등과 같은 장점들을 찾아볼 수 있다.

본 논문에서는 이러한 연구 결과들을 배경 삼아 새로이 권고된 국제 표준 구현 언어인 CHILL-96을 우리의 환경에 맞게 재구성하였다. 이것을 우리는 ECHILL-96(Etri CHILL-96)이라고 하는데 객체지향성(Object-orientation), 중복성(Overloading), 포괄성(Genericity)과 같은 언어적 특징들을 가지고 있다. 이

† 비 회 원 : 한국전자통신연구원 교환·전송기술연구소 연구원
graceckim@etri.re.kr
dglee@etri.re.kr
†† 정 회 원 : 한국전자통신연구원 교환·전송기술연구소 연구원
lcejk@etri.re.kr
논문접수 : 1998년 10월 13일
심사완료 : 2000년 2월 18일

들 특징들은 개발하고자 하는 교환기 및 통신용 시스템의 재사용성과 신뢰성을 향상시킨다.

본 논문의 구성은 다음과 같다. 2장에서 OO_CHILL, Object CHILL과 같은 관련 사례 연구들에 대해 언급하고, 3장에서 ECHILL-96에 포함된 객체지향성, 중복성, 포괄성 등과 같은 새로운 언어적 기능들에 대해 소개하고, 4장에서 ECHILL-96 언어를 위한 구현 환경인 ECHILL-96 컴파일러의 구조에 대해 언급하고, 5장에서 기존의 소프트웨어 블록에 대한 컴파일과 새로운 pilot 프로젝트 개발을 수행하여 컴파일 시간 및 실행 시간 비교를 통해 새로운 컴파일러의 적용성 여부에 대해 살펴보고, 마지막으로 6장에서 본 논문의 결론을 맺는다.

2. 사례 연구

교환기 및 통신 시스템용 소프트웨어의 개발에 있어서 객체지향 언어의 적용이나 객체지향 개발 방법론을 적용한 사례들이 널리 연구되고 있다. 교환기 및 통신 시스템과 같은 대용량의 소프트웨어 개발에 있어서 객체지향 개발 방법론을 적용한 사례로는 NTT의 IN(Intelligent Network) 서비스용 소프트웨어 개발[2], Hitachi의 ATM 교환기 소프트웨어 개발[3], INTRASOFT의 통신 시스템 서비스용 객체지향 개발[4] 등과 같은 연구들이 있다. 하지만 이들 연구들은 언어적 차원에서 새로운 언어를 정의하고, 실행 환경을 제공해 주는 것이 아니라 각자 환경에 맞는 독자적인 언어들(예를 들어, OKI의 Trioss-C)을 사용하고 있거나 프로그램의 구조적 측면에서 재사용성과 유지 보수성을 고려하여 프로그램 구조를 재구성한 것들이다. 한편 기존의 CHILL언어에 객체지향적 특성을 부여하는 접근 방식도 여러 곳에서 시도되었는데 가장 대표적인 것이 알카텔(Alcatel)에 의해 개발된 OO_CHILL[5]과 지멘스(Siemens)에 의해 개발된 Object CHILL[6]이다. 본 장에서는 기존의 CHILL 언어에 객체지향적 확장을 시도한 사례들에 대해 중점적으로 살펴보고자 한다.

2.1 OO_CHILL

OO_CHILL은 1990년대 초반에 기존의 CHILL언어에 객체지향성을 지원하기 위해 알카텔에 의해 개발되었다. OO_CHILL의 가장 중요한 요소로는 틀(Mould), 객체(Object), 상속, 메시지 전송(Message passing)과 같은 것들이 있다.

- 틀 : 속성과 메소드를 가지는데 속성들은 public 가시성을 가지고, 메소드들은 private 가시성을 가진다.
- 객체 : C++처럼 객체들을 직접적으로 명시하는 것이 아니라 포인터를 통해 간접적으로 명시된다. 이렇게 함으로서 비효율적이지만 공유 객체를 허용한다.
- 상속 : 재사용성을 제공해 주는 가장 중요한 기술로서 단일 상속을 지원한다. 또한 private, public과 같은 가시성 유형을 제공한다.
- 메시지 전송 : 객체들간의 통신을 하기 위한 방법으로서 기존의 프로시듀어 호출을 확장한 것이다. 프로시듀어 호출은 프로시듀어 본체에 의해 결정되는데 반해 임의의 객체에 메시지를 전송할 때는 해당 메시지의 이름과 메시지를 보낸 객체에 의해 결정된다.

이처럼 OO_CHILL은 기존의 CHILL 환경을 지원해 주기 위한 중립적 언어로서 추상 자료 형, 사용자 제어 객체 관리, 상속, 다형성, 가상 메소드의 동적 바인딩 지원 등과 같은 장점들을 지니고 있다. 하지만 OO_CHILL은 Metamould, 다중상속, 일반성 등의 미지원과 같은 문제점들을 내포하고 있다.

2.2 Object CHILL

1992년 지멘스에 의해 개발된 Object CHILL은 ISDN 프로젝트를 위한 교환기 시스템 소프트웨어 개발에 실제 적용된 언어로서 객체지향적 기본 요소들 즉, 클래스, 객체, 상속과 다형성들을 지원한다. 또한 병행성, 포괄적 클래스 등과 같은 기능들을 제공한다.

- 클래스와 상속 : Object CHILL의 클래스는 export 인터페이스, import 인터페이스, internal 인터페이스와 같은 잘 정의된 기본적인 내/외부 인터페이스를 제공한다.
- 병행성 : CONCURRENT 클래스에 의해서 동시에 여러 개의 객체들을 생성 및 수행할 수 있게 해준다.
- 포괄성 : 매개인자 다형성이라고도 불리는 포괄성 지원을 위해 Object CHILL은 ANY, ANY_ASSIGN, ANY_DISCRETE, ANY_INT와 같은 포괄적 모드들을 제공한다.

Object CHILL은 기존의 CHILL 언어에 객체지향적 확장을 하여 실제 프로젝트에 사용되었다는 측면에서 그 의의를 찾을 수 있지만 ITU-T에서 권고한 CHILL-96의 어의 및 구문을 완벽하게 지원하지는 못한다. 예를 들어, CHILL-96의 타스크 모드에 해당하는 Object CHILL 구문이 존재하지 않는다.

이처럼 기존의 CHILL 언어에 객체지향적 확장을 시도한 연구가 진행되었지만 ITU-T에서 권고한 CHILL-96의 구문 및 어의와 일치하는 구현 사례는 존재하지 않는다. 본 논문에서 구현 대상 언어로 선정한 CHILL-96은 ITU-T에서 권고한 표준 구현 언어로서 객체지향적 개념들이 잘 정의되어 있고, 언어적 차원에서의 병행

성 지원, 실시간성 지원등과 같은 특성들을 지니고 있다.

3. ECHILL-96의 새로운 언어적 특성

일반적으로 교환기 및 통신 시스템용 소프트웨어들은 대용량, 복잡성, 고신뢰성, 실시간성등과 같은 특징들을 지닌다. 이들 속성들을 지원해 주기 위해 ECHILL-96 언어는 다양한 언어적 특징들을 새로이 추가하였는데 객체 모델(Object model), 추상 자료 타입(Abstract data type, ADT), 정보 은닉(Information hiding), 상속성(Inheritance), 중복성, 포괄성이 그것이다. 본 장에서는 이들에 대해 상세히 살펴보고자 한다.

3.1 객체 모델

ECHILL-96은 객체지향성을 지원하기 위해 세 개의 객체 모델을 제공하고 있다. 이들 객체 모델을 통칭하여 모레타 모드(MoReTa mode)라고 하는데 모듈 모드(Module mode), 임계 모드(Region mode), 타스크 모드(Task mode)의 합성어이다. 객체 모델에 대한 개념적 구조는 그림 1과 같다.

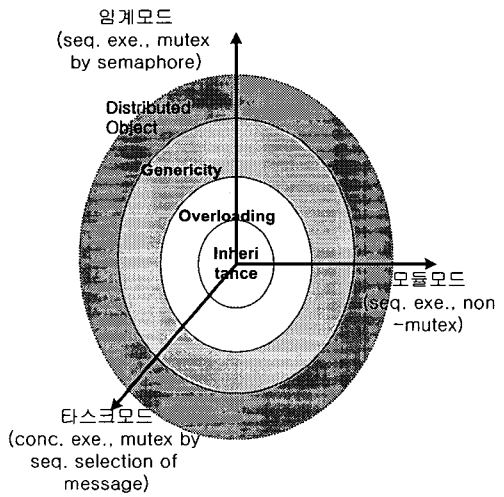

그림 1 객체 모델

이들 모델 각각에 대해 살펴 보면 첫째, 모듈 모드는 순차적인 수행을 하면서 자신의 원소들에 대한 병행 접근에 대해 상호 배제를 제공하지 않는 것이다. 모듈 모드는 가장 전형적인 객체의 형태로서 일반적인 객체지향 언어에서는 클래스라고 부른다. 둘째, 임계 모드는 순차적인 수행을 하면서 자신의 원소들에 대한 병행 접근에 대해 상호 배제 기능을 제공하는 것이다. 이러한 상호 배제 기능은 세마포 기능을 통해 이루어진다. 교환기 소프트웨어에서 트렁크 라인 구축과 같은 제한적인 자원을 여러 블록이 공유하여 사용하는 응용 프로그램의 개발 시 임계 모드를 이용할 수 있다. 셋째, 타스크 모드는 병행적 수행을 하면서 자신의 요소에 대한 병행 접근에 대해 상호 배제 기능을 제공하는 것이다. 타스크 모드는 순차적인 메시지 선택에 의해 상호 배제 기능을 제공한다. 교환기 및 통신 시스템 소프트웨어와 같이 병행성을 지닌 응용을 구축할 경우 가장 빈번히 이용할 수 있는 객체 모델이다. 한편 이들 객체 모델들은 전통적으로 자신의 원소 자료(Component data)와 원소 프로시듀어(Component procedure, 일반적으로 메소드라고 불림)로 이루어져 있다. 이들은 또한 명세부와 구현부로 크게 나누어져 있는데 명세부는 외부와 통신하기 위한 인터페이스이고, 구현부는 외부로부터 숨겨진 부분으로서 상세한 구현 내용이 포함된 부분이다. 이러한 명세부와 구현부의 분리를 통해서 개발된 소프트웨어의 독해성과 유지 보수성이 향상되고 재사용 라이브러리 시스템과 같은 새로운 응용으로의 확장이 용이하다.

3.2 ECHILL-96의 객체지향적 기능

ECHILL-96에서 지원하고 있는 추상 자료 타입(Abstract data type, ADT), 정보 은닉(Information hiding), 상속성, 중복성, 포괄성 등은 객체지향적 특징들을 지원하기 위한 중요한 기능들이다. 본 절에서는 이들 각각에 대해 상세히 살펴본다.

ADT는 자료와 연산으로 이루어진 일종의 틀(template)[5]이라고 할 수 있는데 사용자들에게 상세한 구현 내용들에 대해 정보 은닉을 하기 위해 사용된다. 일반적으로 이것은 외부 명세부와 내부 구현부로 이루어지는데 외부 명세부는 외부와의 통신 규약에 대해 기술하고, 실제 상세 구현 내용들은 내부 구현부에 존재한다.

상속성이란 상위의 클래스로부터 자료와 연산을 하위의 클래스로 전승시켜주는 것을 말한다. 즉, 상위의 클래스를 상속 받아 하위의 클래스에서 새로운 원소들을 첨가할 수 있는 기능을 제공하므로 개발될 소프트웨어의 확장성을 향상시킨다. 그림2는 이러한 상속에 대한 이해를 돕기 위해 상속성을 이용한 ECHILL-96 예제 프로그램이다. 그림 2에서 Telephone 타스크 모드는 일반적인 전화기의 기본적인 기능을 모델링한 것으로서 전화기를 초기화하는 루틴(InitTelephone), 전화기를 등록하는 루틴(PhRegister), 전화기에 신호를 보내는 루틴(SigToPhone)으로 구성된다. Telephone 타스크 모드는 외부 인터페이스에 해당하는 명세부(TASK SPEC,

2-1)와 내부 구현부에 해당하는 본체부(TASK BODY, 2-4)로 이루어져 있다. 그림2의 2-2는 전화기의 상태, 전화번호 등과 같은 것을 나타내는 원소 자료부이고, 2-3과 2-5는 전화기의 초기화, 전화기의 on-hook/off-hook 시그널 송신, 전화기 등록 등과 같은 것을 나타내는 원소 프로시듀어부를 구현한 것이다. Timer Telephone 타스크 모드는 일반적인 전화기에 알람 기능을 가진 전화기를 모델링한 것으로서 Telephone 타스크 모드를 상위 클래스로 가지고, 시간 기능을 지원하기 위해 TimerOn/Off와 같은 확장된 기능을 추가하였다. TimerTelephone 타스크 모드도 Telephone 타스크 모드와 같이 외부 인터페이스에 해당하는 명세부(2-6)와 내부 구현부에 해당하는 본체부(2-9)로 구성된다. 그림2의 2-7은 시간 기능 지원을 위한 원소 자료이고, 2-8과 2-10은 시간 기능 지원을 위한 원소 프로시듀어이다. InitTelephone, SigToPhone 등과 같은 원소 프로시듀어는 상위 클래스(Telephone)의 동일한 원소 프로시듀어를 재구현하여 시간 기능을 지원하도록 확장하였다.

```
SYNMODE Telephone = TASK SPEC  (2-1)
  ...
  NEWMODE PhoneStates = SET( phoneUnconnected,  (2-2)
                             phonePassive,
                             phoneActive       );
 ...
  DCL                        phoneState          PhoneStates;

  DCL  line        RefLine;
  DCL  phone       RefTelephone;
  DCL  phName      Message;
  DCL  phNumber    INT;
 ...
  InitTelephone : PROC( refTelephone RefTelephone ) END;
    (2-3)
  PhRegister    : PROC( no INT, name Message ) END;
  SigToPhone    : PROC( sig SigType ) END;
 ...
END Telephone;

SYNMODE Telephone = TASK BODY  (2-4)
 ...
  InitTelephone:  (2-5)
  PROC( refTelephone RefTelephone );
    ...
  phoneState := phoneUnconnected;
  phNumber   := 0;
  Print( "I'm Telephone\n" );
    ...
  END InitTelephone;
  PhRegister:
  PROC( no INT, name Message );
   ...
    DCL data IntTidPair;
  data.no  := no;
  phNumber := no;
    data.tid := phone;
  oam.PhRegister( data, name );
   ...
END PhRegister;

  SigToPhone:
  PROC( sig SigType );
   ...
    speaker.SigToSpeaker( sig );
   ...
  END SigToPhone;
 ...
END Telephone;

SYNMODE  TimerTelephone  =  TASK  SPEC  BASED_ON
Telephone  (2-6)
  ...
  NEWMODE sendYesNo = SET( yesTimer, noTimer );  (2-7)
  DCL timer       Timer;
  DCL date        Dates;
 ...
  InitTelephone: PROC( refTTelephone REF TimerTelephone )
    (2-8)
  REIMPLEMENT END;
  SigToPhone   : PROC( sig SigType )
  REIMPLEMENT END;
  TimerOff     : PROC() END;
  TimerOn      : PROC() END;
 ...
END TimerTelephone;

SYNMODE  TimerTelephone  =  TASK  BODY  BASED_ON
Telephone  (2-9)

  InitTelephone:  (2-10)
  PROC( refTTelephone REF TimerTelephone )
   REIMPLEMENT;
    ...
    phoneState := phoneUnconnected;
    phone      := refTTelephone;
    sended     := noTimer;
    microphone.InitMicrophone( fD, phone );
    speaker.InitSpeaker( fD );
    timer.InitTimer( phone, ->speaker );
    ...
  END InitTelephone;
  SigToPhone:
  PROC( sig SigType ) REIMPLEMENT;
    ...
     speaker.SigToSpeaker( sig );
    phoneState := phonePassive;
    ...
     line >.SigToLine( onHook );
  phoneState := phoneActive;
     ...
    sended     := noTimer;
    phoneState := phonePassive;
     ...
```

```
  END SigToPhone;
  TimerOff:
  PROC();
    ...
    speaker.TimerOff();
    ...
  END TimerOff;
  TimerOn:
  PROC();
    ...
     speaker.TimerOn();
    ...
  END TimerOn;
END TimerTelephone;
```

그림 2 상속성

중복성이란 동일한 연산이 서로 다른 동작을 가지게 허용하는 개념이다. 즉, 일종의 다형성 개념을 제공하는 것으로 하나의 연산 이름이 서로 다른 요소를 지칭할 수 있는 것을 허용하는 것이다. 하지만 이러한 중복성은 언어의 복잡도를 높이는 부정적인 측면이 있기 때문에 ECHILL-96에서는 단지 프로시듀어 중복성만을 허용한다. 즉, 연산자 중복성과 같은 기능은 제공하지 않는다.

포괄성이란 매개적 다형성(Parametric polymorphism)이라고도 불리는데 프로그램 원소들의 포괄적 매개 인자화 개념이다[6,7]. 포괄성으로 가장 잘 알려진 예제로는 스택 연산 예제를 들 수 있다. 기본적인 스택에 정수 연산을 수행하는 스택, 실수 연산을 수행하는 스택 등과 같이 다양한 종류의 새로운 스택을 정의할 수 있다. 이처럼 서로 다른 연산을 수행하는 스택에 대해 동일한 연산을 이용하여 pop, push등과 같은 기능을 수행할 수 있다. 복잡한 구조를 가진 하나의 요소는 이들이 생성될 때에 전달되는 매개인자의 값이나 타입에 따라 수행되어야 할 연산을 선택하여 실행한다. 이러한 특징들은 개발되는 소프트웨어의 재사용성과 유연성을 증대 시키는 효과가 있다.

4. ECHILL-96 컴파일러의 구조적 모델

3장에서 언급한 ECHILL-96 언어의 새로운 언어적 기능들을 지원하기 위한 ECHILL-96 컴파일러를 구현하였다. ECHILL-96 컴파일러는 기존의 CHILL 언어로 개발된 대용량의 소프트웨어들을 위한 개발 환경과 100% 호환성을 제공하고 있다[7]. 이것은 소프트웨어 유지보수 비용을 최소화하는데 중요한 접근 방법이다. 이러한 목적을 달성하기 위해 본 논문에서는 직접 변환 규칙 기법과 확장된 CHILL 중간 코드 생성과 같은 두 단계의 접근 방법을 사용하였다. 본 장에서는 이들에 대한 상세 설명과 ECHILL-96 컴파일러의 구조에 대해 기술한다.

4.1 직접 변환 기법

기존의 개발된 대용량 소프트웨어들과의 상호 호환성은 소프트웨어 개발 및 유지 보수에 있어서 중요한 요소이다. 이러한 점을 고려하여 새로이 권고된 ECHILL-96과 CHILL간의 직접 변환 기법을 이용한 코드 생성 방법은 미세한 언어적 차이를 가진 두 언어에 대해 재명명과 재구조화 모듈을 수행하여 코드를 생성한다[8]. 재명명 모듈은 새로운 언어적 개념인 모레타 모드들에 대해 풍부한 디버깅 정보의 제공과 이름 충돌을 방지하

표 1 재명명 변환 규칙

	변환 규칙	예제
변수 이름	<variablename> =>__<depth>_<block name>__<variable name>	__3_n1_n2_n3__xxx
모레타 모드 이름	<moreta mode name> => <m|r|t><length of moreta mode><moreta mode name>	m5stack
프로시듀어 이름	<procedure name> => <procedure name>__<moreta name> F<parameter> <return><parameter>=> {bo|ch|in|re|ra#_#|sb#-#|bs#|cs#|vs#|ss#_#|se#_X_|po#_X_|bu#Y|ev#|it}|Tm_n <return> => E|__y	fff__mmmFinN4strtT1_2__y
포괄적 형이름	<generic name> => G<moreta name><pn><parameter> <pn> => # of parameter <parameter> => {0|1|#|#_#|_#|_str_|#_X_}	Gm1X2Zinin10

기 위한 것으로 아래의 표1과 같은 재명명 변환 규칙을 적용하였다.

재구조화 모듈은 객체 모델, 상속성, 중복성 등과 같은 객체지향 개념이 포함된 ECHILL-96 언어 기능을 객체지향 개념이 없는 CHILL로 변환하는 것으로서 모듈 속에 정의된 모레타 모드 및 이의 구성 원소들이 내장 모듈 및 확장된 이름을 갖는 원소들로 재배치되고 내장 모듈들 간에 상속을 위해 모듈 이름을 Grant/Seize 하여 프로그램 구조를 재구조화 시킨다. 이러한 재명명과 재구조화 변환 기법을 이용하게 됨으로서 생성되는 중간코드 및 목적 코드의 크기가 대응되는 CHILL 코드에 비해 증가하는 단점이 발생한다.

4.2 확장된 CHILL 중간 코드 생성

4.1절에서 언급한 변환 규칙에 의해 구축된 심볼 테이블을 중심으로 하여 추상 구문 트리(Abstract syntax tree, AST) 의 순회를 통해 확장된 CHILL 중간 코드를 생성한다. AST는 각 구문 요소들이 트리의 형태로 재구성된 것이다. 이러한 접근 방법은 새로이 개발될 소프트웨어와 기존에 개발된 소프트웨어 간의 호환성 문제를 해결해 주지만 이것은 과도기적 코드 생성 방법에 불과하고, 궁극적으로는 효율적이고 최적화된 코드 생성 알고리즘의 도입이 요구된다. 현재 ECHILL-96 컴파일러는 모레타 모드, 모레타 모드 동적 타입 점검, 상속성, 모레타 원소 접근 제어, 모레타 원소 프로시듀어 재구현, 다형적 포인터 및 원소 프로시듀어의 동적 결합 등과 같은 기능들을 위한 코드 생성 기법이 구현되어 있다. 이들 확장된 CHILL 중간 코드를 생성하기 위한 알고리즘을 살펴보면 다음과 같다.

<코드 생성 알고리즘>

1. 지역 변수 선언 및 초기화
2. 초기 디버깅 라인 정보 코드 생성
3. 선언부 코드 생성 함수 호출
   3.1 GRANT/SEIZE 정보 코드 생성
   3.2 CHILL 프로그램 구조(MODULE / REGION, SPECMODULE / SPECREGION, PROC, PROCESS,…) 코드 생성
      3.2.1 프로그램 구조내의 선언부 코드 생성 함수 호출
      3.2.2 실행부가 존재할 경우 실행부 코드 생성 알고리즘 수행
   3.3 ECHILL-96 모레타 모드 구조(MODULE/REGION/TASK SPEC/BODY) 코드 생성
      3.3.1 명세부 코드 생성 함수 호출
      3.3.2 구현부 코드 생성 함수 호출
      3.3.3 실행부가 존재할 경우 실행부 코드 생성 알고리즘 수행
4. 종료

<실행부 코드 생성 알고리즘>

1. 지역 변수 선언 및 초기화
2. AST의 각 노드들 종류(NASSIGNACT, NCALLACT, NLOCATION,…)에 따라 적당한 코드 생성
   2.1 장소부에 대한 코드 생성
   2.2 형제 노드가 존재할 경우 실행부 코드 생성 알고리즘을 재귀적으로 호출
3. 종료

4.3 ECHILL-96 컴파일러의 기능적 내부 구조

ECHILL-96 컴파일러를 기능적으로 구분해 보면 크게 두 부분으로 나누어지는데 입력되는 원시 파일들에 대해 어휘 분석, 구문 분석, 가시성 분석 및 어의 분석과 같은 기능을 수행하는 전위 처리부와 GRANT/SEIZE 정보간의 이름 결합, 타입 점검 및 중간 코드 생성과 같은 기능을 수행하는 후위 처리부로 이루어져 있다. 그림 3은 ECHILL-96 컴파일러의 기능적 내부 구조를 보여주고 있는데 각 부분은 7개의 주요 세부 기능들로 이루어져 있다.

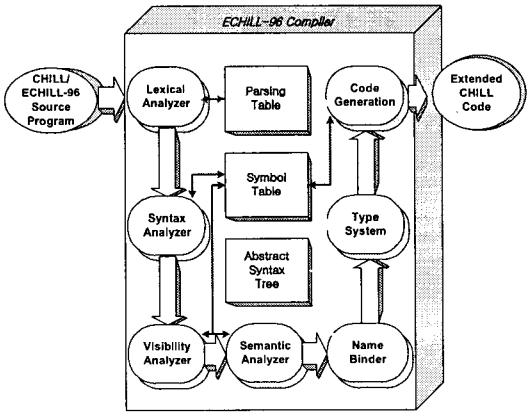

그림 3 ECHILL-96 컴파일러 구조도

어휘 분석(Lexical analyzer) 모듈은 입력되는

ECHILL-96 원시 파일들을 분석하여 의미 있는 최소의 단위인 토큰(token)으로 분할하는 것이다. ECHILL-96 컴파일러는 lex를 이용하여 이러한 어휘 분석을 수행한다[9]. ECHILL-96컴파일러는 ABSTRACT, BASED_ON, MODULE, REGION, TASK, ASSIGNABLE / NOT_ASSIGNABLE, CONSTR, DESTR, SELF등과 같이 새로이 추가된 예약어들에 대해 토큰 생성을 수행하는 부분이 추가되었다.

구문 분석(Syntax analyzer) 모듈은 입력된 ECHILL-96 원시 파일들에 대해 새로이 추가된 구문적 특징들에 대한 구문 오류를 검출하는 기능을 수행한다. 예를 들어, 객체의 정의/선언/생성, 객체 장소에 대한 접근 제어 및 컴파일러와 디버그를 위한 기본적인 심볼 테이블 및 AST의 구축등과 같은 기능들을 수행한다.

ECHILL-96 컴파일러는 이러한 구문 분석을 위하여 yacc을 이용하였다[10]. 구문 분석기는 내부적으로 다음과 같은 절차를 수행한다. 먼저 ECHILL-96 원시 프로그램내에 있는 단위 프로그램들간의 포함 관계 구조를 분석하여 심볼 테이블에 저장하고, 선언부에 나타난 명칭들에 관련된 각종 정보들(모드 정보, 크기 정보 등)을 수집하여 심볼 테이블에 저장하고, 마지막으로 각 단위 프로그램들에 대해 실제 실행을 나타내는 실행부가 있으면 이들에 대해 AST를 구축한다. 예를 들어, ECHILL-96 프로그램의 한 문장에 대해 AST를 구축한 결과는 그림4와 같은데 AST는 단말 노드(terminal node)와 비단말 노드(non-terminal node)로 이루어져 있다. ECHILL-96 컴파일러는 코드 생성 모듈 단계에서 중간 코드 형태인 확장된 CHILL 코드를 생성하기 위해 실질적으로 이들 노드들에 포함된 정보들(심볼 정보, 모드 정보, 등)을 이용하게 된다.

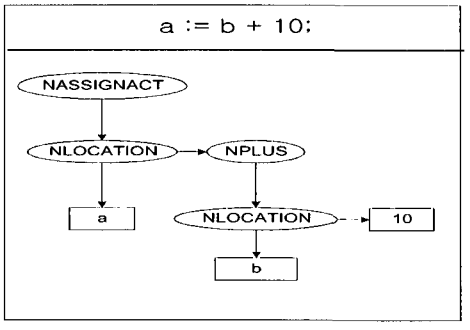

그림 4 AST 구축 예

가시성 분석(Visibility Analyzer) 모듈은 GRANT / SEIZE 선언문의 처리를 수행하는 모듈로서 ECHILL-96의 모듈리언은 각 이름들의 경계 역할을 하기 때문에 모듈리언의 범위를 넘어서 참조될 수 없다. 하지만 이러한 경계를 넘기 위해서 GRANT와 SEIZE 선언문을 이용한다. 즉, 현재의 가시성 범위에 있는 정보를 다른 가시성 범위로 확장하기위해서 GRANT문을 이용하고, 다른 가시성 범위에 있는 정보를 현재의 가시성 범위로 끌어들이기 위해서는 SEIZE 문을 이용한다. 가시성 분석 모듈에서는 구문 분석 모듈에서 생성된 심볼 테이블을 참조하여 가시성 제어를 위해 심볼 테이블을 재구성한다. 그림5는 가시성 제어를 하기 전과 후의 심볼 테이블의 변화를 나타내고 있다.

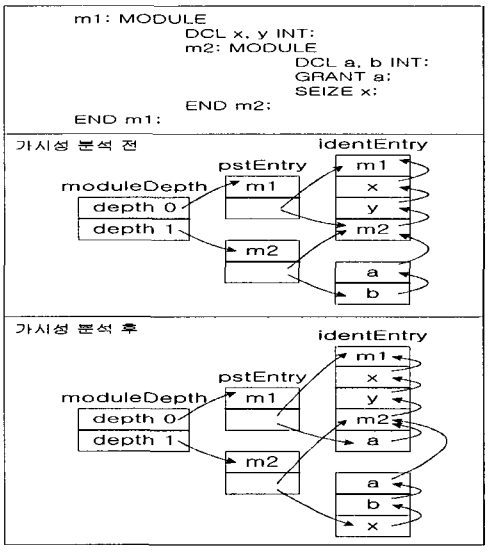

그림 5 가시성 분석 전.후의 심볼 테이블 변화

어의 분석(Semantic analyzer) 모듈은 구문 분석기에 의해 생성된 심볼 테이블내의 각종 이름들 사이에 존재하는 어의 점검(예를 들어, 모레타 명세부와 구현부 간의 원소 정의 및 선언의 일치성 점검 및 상속 계층의 점검, 모드 점검 등)을 수행하는 모듈이다. 어의 분석 모듈은 크게 선언부(DCL, SYN, SYNMODE, NEWMODE, SIGNAL등) 어의 분석과 실행부(조건문, 반복문, 분기문, 치환문, 호출문 등) 어의 분석으로 이루어져 있다. 이처럼 어의 분석 모듈이 분리되어 있으므로 인해 이들에 대한 동시 개발이 이루어져 개발 순기의 단축, 오류 수정의 용이성 등과 같은 효과를 나타내었다.

이름 결합기(Name binder)와 타입 시스템(Type system) 모듈은 실행문에서 사용된 객체 장소에 대한 접근 제어, 객체 장소간의 연산 제어, 매개인자 전달 시 타입 호환성 점검, 치환문의 타입 호환성 점검 등과 같은 기능들을 수행하는 모듈이다.

코드 생성(Code generation) 모듈은 AST의 순회와 심볼 테이블의 참조를 통해 확장된 CHILL 중간 코드를 생성한다. 코드 생성 모듈은 크게 프로그램 구조에 대한 코드 생성, 객체 정의들에 대한 코드 생성 및 실행문에 대한 코드 생성으로 이루어져 있다. 이러한 코드 생성 기법은 기존에 개발된 대용량의 소프트웨어들과 100% 호환성을 제공해 준다.

이처럼 ECHILL-96 컴파일러는 기능적으로 모듈화를 이루고 있어 동시에 여러 개발자가 독립적으로 각 모듈을 개발할 수 있었고, 각 단계별 오류 검출 및 수정이 용이하였다.

5. 성능 비교

컴파일러의 성능을 비교하기 위해서는 다양한 방법들이 있을 수 있다. 예를 들어, 컴파일 시간의 비교, 실행 시간의 비교, 메모리 점유율 비교, 목적 코드 크기 비교, 메시지 송/수신 처리율 비교 등과 같은 많은 요소들이 있다. 하지만 본 논문에서는 ECHILL-96 컴파일러의 성능 및 적용 가능성을 측정하기 위해 다음과 같은 두 가지 방법을 시도하여 성능 비교를 수행하였다. 첫째, CHILL 컴파일러와 ECHILL-96 컴파일러의 컴파일 시간을 비교하였고, 둘째, 이들 두 컴파일러간의 실행 시간을 비교하였다.

5.1 컴파일 시간 비교

대용량 소프트웨어(140만 라인이상)를 컴파일하여 두 컴파일러간의 컴파일 시간 비교에 의한 접근 방법은 ECHILL-96 컴파일러의 신뢰성을 검증할 수 있는 척도가 된다. 그림 6은 컴파일을 수행한 전체 ATM 교환기 시스템용 블록 중에서 성능에 영향을 끼치는 주요 블록을 선별하여 보여주고 있다. 이들 주요 블록은 전체 91,079 라인으로 이루어져 있는데 기능별로 과금 및 통계를 담당하는 블록(Charging/Statistics block, 41,361 라인), 호 처리 블록(Call control block, 33,482 라인), 운용 보전 블록(Configuration block, 16,236 라인)과 같은 세 부류로 나누어진다. 이들 블록들을 현재 ATM 교환기 시스템 소프트웨어 개발용 호스트 환경인 Cray CS6400 시스템하에서 컴파일을 수행하였고, 비교 대상이 되는 시간은 해당 블록 당 CPU의 수행 시간을 의미한다. 그림6에서 제시한 시간 축은 동일한 블록을 100회씩 수행한 결과의 평균 수행 시간 값을 나타낸다.

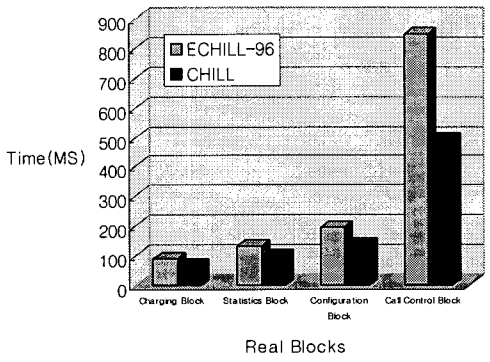

그림 6 CHILL과 ECHILL-96 컴파일러의 컴파일 시간 비교

그림 6에서 ECHILL-96 컴파일러의 컴파일 시간은 CHILL 컴파일러보다 5% ~ 30% 성능 감소를 나타내고 있다. 이와 같은 성능 감소의 가장 큰 요인으로는 코드 생성부에 있다. 즉, ECHILL-96 컴파일러는 추상 구문 트리라는 트리 구조를 순회하면서 확장된 CHILL 코드를 생성하므로 이러한 트리를 순회하는데 많은 시간이 소요된다. 차후에 추상 구문 트리 순회를 보다 효과적이고 빠르게 할 수 있는 알고리즘에 대한 연구가 이루어져야 한다.

5.2 실행 시간 비교

실행 시간 성능을 비교하기 위해 두개의 사설 구내 교환기(Private Automatic Branch eXchange, PABX) 시뮬레이션 시스템을 개발하였다. 이 시스템은 OAM (Operation And Management), PABX 및 PHONE 블록과 같은 세 개의 주요 기능 블록으로 이루어져 있는데 이들에 대한 시스템 구조를 설계 명세 언어인 SDL로 나타내면 그림7과 같다.

그림7에서 OAM블록은 새로운 가입자의 등록, 접속 가능한 가입자 정보의 관리등과 같은 기능을 담당하고, operator, control, monitor 및 dbmanager 상세 블록과 같은 네 개의 상세 블록들로 이루어져 있다. Operator 상세 블록은 외부 블록과의 통신을 담당하고, control 상세 블록은 사용자의 입력(음성이나 문자)에 대해 처리하고, monitor 상세 블록은 OAM 블록의 오류 메시지를 관리하고 입력되는 명령어를 처리하고, dbmanager 상세 블록은 예약된 가입자에 대한 정보와 대화 결과에 대한 보고들을 데이터베이스화하여 관리하는 기능을 담당하고 있다. PABX 블록은 접속 가능한 가입자의 접속

을 수행한다. 이 블록은 필요한 정보를 시그널의 형태로 OAM 블록과 통신하는데, server, line, trunkalloc 및 trunk 상세 블록과 같은 네 개의 상세 블록으로 이루어져 있다. Server 상세 블록은 OAM의 입력 명령 결과를 시그널의 형태로 OAM 블록에게 보내는 기능을 담당하고, line 상세 블록은 subscribers와 PABX 블록간의 대화를 수행하고, trunkalloc 상세 블록은 트렁크 라인의 배타적인 할당을 담당하고, trunk 상세 블록은 실제적인 대화를 수행한다. PHONE 블록은 전화기간의 자료 교환을 담당하는데 내부적으로 phone, microphone 및 speaker 상세 블록과 같은 세 개의 상세 블록으로 이루어져 있다. Phone 상세 블록은 PABX와의 대화를 수행하고, microphone 상세 블록은 사용자의 입력과 텍스트 형태의 음성 입력을 처리하고, speaker 상세 블록은 음성 메시지를 출력하고, 이 메시지를 다른 가입자 단말에 출력하는 기능을 담당하고 있다.

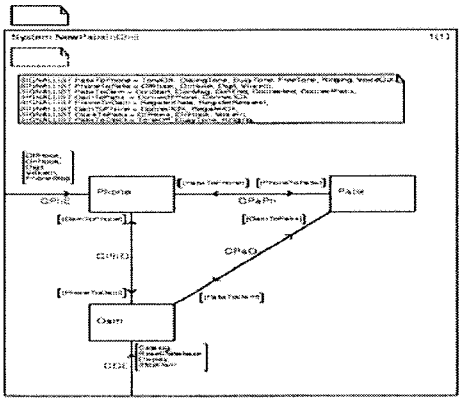

그림 7 시뮬레이션 시스템 구조

이와 같은 구조를 가진 시뮬레이션 시스템의 실행 시간 성능 측정을 위해 목적 시스템으로 ATM 교환기용 시험 시스템을 선정하여 운용 시험을 하였다. ATM 교환기용 시험 시스템은 1개의 SPARC CPU를 가진 VMEBus가 장착된 시스템이다. 실행 시간 측정의 대상은 해당 블록 당 CPU의 수행 시간을 의미하는데 실행 시간 측정을 위해서 트렁크의 할당, 통화 등과 같은 일련의 작업들을 실행하는 시간을 측정하였다. 그림 8에 나타난 시간 축은 동일한 시뮬레이션 시스템을 100회씩 수행한 결과에 대한 평균 수행 시간 값이다.

그림 8은 이러한 두개의 시뮬레이션 시스템의 실행 시간 비교 결과를 나타내고 있다. 실행 시간 비교를 위해 트렁크와 전화기의 수를 동적으로 변경하면서 수행하였다. 그림 8에서 ECHILL-96 컴파일러의 실행 시간 성능은 CHILL 컴파일러에 비해 5%~10%의 성능 감소를 초래하고 있다. 이러한 성능 감소의 주된 요인으로는 모레타 모드에 대한 동적 타입 점검과 원소 프로시듀어의 동적 결합과 같은 기능들이 영향을 끼치고 있다. 차후에 동적 원소 프로시듀어 테이블의 검색 기법을 최적화하는 연구가 수행되어야 한다. 한편 성능 감소와 같은 단점들 보다는 객체지향성으로 인한 소프트웨어의 확장성, 재사용성, 유지보수성 등과 같은 많은 장점들이 있으므로 교환기 및 통신 시스템과 같은 대용량의 실시간 소프트웨어 개발 시 유용하게 적용할 수 있다.

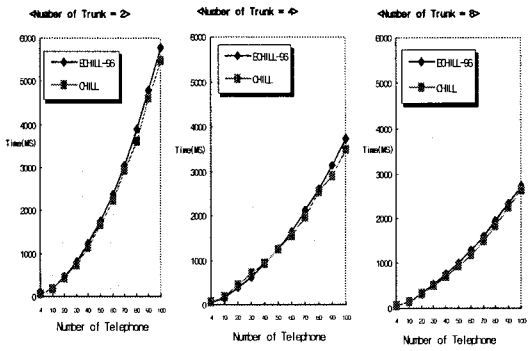

그림 8 CHILL과 ECHILL-96 컴파일러의 실행 시간 비교

6. 결 론

본 논문에서는 교환기 및 통신 시스템과 같은 대용량 소프트웨어 개발을 위해 새로이 권고된 국제 표준 구현 언어에 대해 소개하였고, 프로그래밍 환경으로 설계 및 구현된 ECHILL-96 컴파일러의 내부 구조, 기능, 특징 등에 대해 기술하였다. ECHILL-96 언어는 소프트웨어의 재사용성, 신뢰성, 유연성을 증대 시키기 위해 목적 시스템 환경에 맞게 재구성하였다. 또한 CHILL 및 ECHILL-96 컴파일러간의 컴파일 시간 및 실행 시간 성능 비교를 수행하여 두 컴파일러간의 성능을 비교하였다. 비록 컴파일 시간과 실행 시간이 기존의 CHILL 컴파일러보다 성능 감소를 가져오는 단점이 있지만 ECHILL-96 컴파일러는 새로 개발될 소프트웨어와 기존에 개발된 소프트웨어간의 하향 호환성을 제공해 주고 있어서 유지보수 비용이 최소화되는 효과를 기대할 수 있다. 앞으로의 연구 과제로는 성능 향상을 위한

ECHILL-96 컴파일러의 최적화 기법, 재사용성 증대를 위한 재사용 라이브러리 시스템의 개발, 분산 객체 처리 환경의 수용 등을 수행하여야 할 것이다.

참 고 문 헌

[1] International Telecommunication Union (ed.) CCITT High Level Language Recommendation Z.200, Geneva, 1996.

[2] Katsumi Maruyama, etc., "A Concurrent Object-Oriented Switching Program in CHILL," IEEE Communications Magazine, pp. 60-68, Jan. 1991.

[3] Nobuhiko Ido, etc., "Development of ATM Switching Software Based on Object-Oriented Hierarchical Structure," Proceedings of XIV International Switching Symposium, pp. 420-424, Oct. 1992.

[4] Panos Fitsilis, "Object-Oriented Development for Telecommunication Services," Information and Software Technology, vol. 37, no 1, pp. 15-22, 1995.

[5] Scortesse A., "OO_CHILL : Integrating the Object Paradigm into CHILL," Proceedings of the 5th CHILL Conference, pp. 111-117, Mar. 1990.

[6] Jurgen F. H. Winkler, Georg Diebl, "Object-CHILL - An Object Oriented Language for Telecom Applications," Proceedings of XIV International Switching Symposium, pp. 204-208, Oct. 1992.

[7] Cardelli L., Wegner P., "On Understanding Types, Data Abstraction, and Polymorphism," Computing Surveys, vol. 17, no 4, pp. 471-522, 1985.

[8] DongGill Lee, etc., "A New Integrated Software Development Environment Based on SDL, MSC, and CHILL for Large-scale Switching Systems," ETRI Journal, vol. 18, no 4, pp. 265-285, Jan. 1997.

[9] SangKug Kim, etc., "An Implementation of Translation Mechanism from CHILL96 to CHILL," Proceedings of KITE Fall Conference'96, vol. 19, no 2, pp. 286-289, Nov. 1996.

[10] Lesk, M. E. and E. Schmidt, "Lex ? A lexical analyzer generator," Computing Science Technical Report, no. 39, Bell Laboratories, Murray Hill, New Jersey, 1975.

[11] Johnson, Stephen C., "Yacc: Yet Another Compiler Compiler," Computing Science Technical Report, no. 32, Bell Laboratories, Murray Hill, New Jersey, 1975.

김 상 은

1990년 2월 계명대학교 전자계산학과 학사. 1992년 8월 계명대학교 전자계산학과 석사. 1992년 8월 ~ 2000년 현재 한국전자통신연구원/교환.전송기술연구소 선임연구원. 관심분야는 객체지향 시스템, 컴포넌트 기반 소프트웨어 개발, 분산 실시간 컴퓨팅 환경

이 준 경

1985년 2월 서강대학교 물리학과(부전공: 전산학과) 이학 학사 졸업. 1987년 2월 숭실대학교 전자계산학과 공학 석사 졸업. 1997년 2월 숭실대학교 전자계산학과 공학 박사 졸업. 1987년 2월 ~현재 한국전자통신연구원 선임연구원으로 재직중. 관심분야는 컴파일러 구성론, 프로그래밍 언어론 및 개발환경, 객체지향 시스템 환경, 소프트웨어 공학, 운영체제 등임.

이 동 길

정보과학회논문지 : 컴퓨팅의 실제
제 6 권 제 2 호 참조