SDL-92에서 객체지향 언어의 코드 생성을 위한 개념 변환

(Conceptual Transformation for Code Generation from SDL-92 to Object-oriented Languages)

이 시 영 † 이 동 길 †† 이 준 경 ††† 김 승 호 ††††

(Siyoung Lee) (DongGill Lee) (Joon-Kyung Lee) (Sung-Ho Kim)

요 약 시스템의 명세 및 기술 언어인 SDL-92는 객체지향 개념의 도입에서 기존의 시스템 명세 및 설계 문서들과 사용자들을 포용하기 위해 프로세스와 시그널에 기반한 통신 방법을 고수하였다. 이러한 객체지향 개념의 도입은 메소드와 객체 기반의 객체지향 언어 프로그램의 자동 생성에 있어 대응 개념의 부재라는 문제점뿐만 아니라 이에 따르는 가시성 및 통신 방법과 같은 부수적인 문제점들까지 유발하고 있다. 따라서 본 논문에서는 메소드와 객체에 기반한 일반적인 객체지향 언어 모델을 제시한 후, SDL-92에서 제시된 모델로의 변환에서 발생하는 문제점들을 고찰하고 이를 해결할 수 있는 개념 변환 방법들을 제시한다. 제시된 변환 방법은 목적 언어의 구문으로의 사상 방법을 제공함으로써 객체들에 내장된 병렬성을 활용할 수 있고 변환된 프로그램에 대해 컴파일러 수준의 이식성을 보장할 수 있다.

Abstract SDL-92, the language for specification and description of system, has held on to the communication method that based on processes and signals in the adoption of object-oriented concept to embrace the previous documents of system specification and description and users. It has caused problems, not only the absence of corresponding concepts in automatic generation to object-oriented language program based on method and object, but also some side effects accompanied by them like visibility and communication method. So, in this paper, we present a general object-oriented language model which based on method and object, make a study of problems in the transformation from SDL-92 to proposed model, and then propose conceptual transformation methods to solve them. The proposed transformation method can utilize the built-in parallelism in objects and guarantee the compiler level portability in translated program by providing translation into the syntax of target language.

1. 서 론

근래에 들어서 소프트웨어 개발 방법론의 중요한 연구 초점이 구현에서 명세와 설계 쪽으로 이동함에 따라 분석과 실행이 가능한 SDL(Specification and Description Language)과 같은 명세 및 설계 언어를 통해 직접적인 프로그램 코드의 작성 없이 소프트웨어를 개발하는 방법이 다양하게 연구되어 왔다[1][2][3][4][5]. 특히 통신용 소프트웨어는 대형이고, 긴 생명주기를 가지며, 대규모 개발 인원이 필요하다는 특성을 가지므로, 시스템의 효율적인 유지 보수를 지원할 수 있는 정형적인 명세와 검증 방법이 필요하다. 따라서 ITU-T(International Telecommunication Union - Telecommunication Standardization Sector)와 ISO (International Organization for Standardization) 같은 국제 표준화 기구에서는 SDL, LOTOS(Language Of

† 학생회원 : 경북대학교 컴퓨터공학과
sylee@bluecord.co.kr
†† 비 회 원 : 한국전자통신연구원 개발환경연구실 연구원
dklee@etri.re.kr
††† 정 회 원 : 한국전자통신연구원 개발환경연구실 연구원
leejk@etri.re.kr
†††† 종신회원 : 경북대학교 컴퓨터공학과 교수
shkim@bh.kyungpook.ac.kr
논문접수 : 1999년 3월 3일
심사완료 : 2000년 2월 3일

Temporal Ordering Specifications), 그리고 Estelle와 같은 명세 언어들을 표준화하여, 소프트웨어의 구현에 앞서 명세 언어들을 통한 시스템의 명세 및 기술을 권고하고 있다[6][7][8]. 그러나 LOTOS와 Estelle가 ISO의 OSI(Open Systems Interconnection) 프로토콜 기술에 사용이 제한되는 반면 SDL은 확장된 유한 상태 기계(EFSM, Extended Finite State Machine) 개념을 기반으로 하여 시그널을 이용한 통신용 소프트웨어 명세의 용이성, 그리고 SDL/GR(SDL Graphical Representation)과 같은 시각화된 도식 표현의 제공에 따른 이해의 편리성에 따라 시스템 공학에서 널리 사용되고 있다[9][10]. 그리고 SDL을 기반으로 하여 시스템 개발 주기의 단축, 소프트웨어 품질 향상, 개발비용 감소, 패턴과 코드의 재사용, 생산성의 향상, 그리고 사용자 만족과 같은 장점을 제공할 수 있는 구현 프로그램의 자동 생성에 관한 연구가 수행되어져 왔다[12][13][14][15][16].

객체지향 SDL인 SDL-92는 기존의 시스템 설계와 사용자들을 수용하기 위해 시그널에 의한 통신 방법을 유지한 채 아래와 같이 형을 이용한 개체의 상속과 전이의 재정의를 통한 행위의 상속으로 객체지향 특성을 도입하였다[10].

(a) 개체의 계승을 위해 기존의 시스템, 블록, 프로세스와 같은 개체에 대해 형을 지원
(b) 동작의 상속을 위해 전이 단위의 상속을 지원하는 가상 전이를 지원
(c) 시그널에 의한 통신 방법의 유지

그러나 이러한 혼성의 객체지향 특성은 구현 언어로 객체지향 언어를 사용할 때 클래스, 메소드와 같은 대응 개념의 부재라는 문제점과 이에 따르는 통신 방법, 가시성과 같은 부수적인 문제점들을 유발하였다.

기존의 C++로의 번역에 관한 연구들은 SDL-92의 프로세스와 시그널에 기반한 통신 방법을 클래스와 통신을 위한 라이브러리로 표현하는 변환 기법을 사용하였다[13][14][15][16]. 그러나 이러한 접근 방법은 각 프로세스들이 서로 다른 프로세서에 분산되어 있는 환경에는 적합하나, 동일 프로세서에 존재하는 프로세스들 간의 통신을 표현하는 데는 적합하지 않다. 또한 이러한 변환 방법은 목적 언어의 구문으로 지원할 수 있는 메소드에 기반한 통신 방법을 제외함으로써, 자동 번역된 프로그램의 이식성을 저하시키고, 지원 라이브러리와 운영 체제와 같은 동작 환경에 대한 종속성을 높이는 단점이 있다.

따라서 본 논문에서는 SDL-92의 구현 언어로 사용될 수 있는 객체지향 언어의 일반적인 모델을 가정한 후, 기존의 연구 방법과는 달리 제시된 모델의 객체에 내장된 병렬성과 메소드를 이용한 통신 개념을 사용할 수 있도록 SDL-92의 대응 개념들을 변환하고, 이를 객체지향 언어의 개념으로 사상하는 방법을 제시하고자 한다.

제시된 개념 변환 방법은 목적 언어의 구문으로 SDL-92의 개념들을 변환할 수 있도록 대응 개념들의 사상을 설계하고, 시그널에 기반한 행위 기술과 통신 방법을 메소드에 기반한 행위 기술과 통신 방법으로 변환하는 방법을 제시함으로써 번역된 프로그램에 대해 컴파일러 수준의 이식성을 보장할 수 있다. 또한 특정 객체지향 언어에 대해 종속적이지 않으므로, 객체와 메소드에 기반한 다른 객체지향 언어로의 번역에도 적용할 수 있는 장점이 있다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 기존의 SDL 변환 기법에 대해 고찰하고, 객체지향 언어로의 변환시에 문제점을 유발하는 SDL-92의 특성들과 이로 인해 발생하는 문제점들에 대해 논한 후, 구현 언어로 사용될 객체지향 언어 모델을 제시한다. 그리고 3장에서는 개념 변환을 위한 자료 구조들과 이를 이용한 개념 변환 방법을 제시한 후, 마지막으로 4장에서 결론을 내린다.

2. SDL-92에서 객체지향 언어로의 변환에 따른 문제점

2.1 기존 SDL-92의 변환 기법에 대한 고찰

현재에 들어 소프트웨어 개발 주기의 단축을 위해 CASE 도구들을 통한 SDL 시스템 기술의 자동 코드 생성을 소프트웨어 산업에 통합하는 것은 점차 필수적인 것이 되고 있다[3]. Telelogic의 SDT와 Verilog의 GEODE와 같은 상업적인 SDL 통합 도구들은 실제 수행이 가능한 C 코드 생성기를 제공하고 있으며[3][5], 1990년대에 들어 객체지향 패러다임의 성숙에 따라 객체지향 언어를 목표로 하는 코드 생성에 관한 연구도 활발히 이루어져 왔다[13][14][15][16].

Fisher 등은 SDL-92로 작성된 명세와 기술을 시뮬레이션의 목적으로 C++로 번역하는 방법을 제시하였다[13]. 제시된 방법에서는 프로세스와 채널, 타이머, 그리고 각각의 시그널에 대해 클래스를 선언하고 해당 클래스의 인스턴스들을 송수신할 수 있는 INPUT(), OUTPUT()과 같은 멤버 함수들을 외부 라이브러리의 지원으로 프로세스에 대응하는 프로세스 클래스에서 정의하는 방법을 사용하였다.

Gries는 SDL++이라는 도구 집합을 개발하여 SDL-92 프로세스 다이어그램의 변형인 SDL++ 클래스들을 C++ 클래스들로 자동 번역하는 방법을 제시하였다[14]. 시스템 개발자가 추가한 원시 프로그램들과 함께 해석된 SDL++ 클래스들의 프로세스 다이어그램들은 기존의 SI-GRAPH-SET-SDL-ED 구성 요소와 함께 편집되어 C++ 프로그램으로 번역되었다. 그러나 제시된 SDL++ 클래스의 프로그램 생성은 프로그램 수준에서 개발자에 의한 확장된 문서 표현들의 수정을 필요로 하여 설계 수준에서 시스템 구현을 이룰 수는 없었다.

Inochencio 등은 ISDN의 터미널 신호 처리를 위해 SDL-92로 작성된 시스템 기술을 C++로 번역하는 방법을 제시하였다[15]. 제안된 방법은 SDL 프로세스들을 C++ 클래스로 변환하고, SDL에서 사용하는 통신 방법을 외부 라이브러리를 이용하여 일대일 사상을 통해 변환함으로써 실시간 환경에서 수행이 가능한 C++ 코드를 생성하였다. 생성된 C++ 코드는 클래스로 대응되는 프로세스에서 시그널을 위한 큐를 유지하고, 해당 시그널들을 위한 입출력 메소드를 제공함으로써 시그널에 의한 통신 방법을 그대로 유지하였다.

RASTA(Russian Academy of Science Translator) 프로젝트에서는 SDL-92의 ADT(Abstract Data Type)를 C++ 클래스로 자동 번역하는 방법을 제시하였다[16]. 제안된 컴파일러는 TRS(Term-Rewriting System) 항등식을 사용하여 ADT내에서 연산자를 정의하는 공리들을 C++ 클래스의 멤버 함수로 해석하는 방법을 사용하였다. 그러나 RASTA 프로젝트에서는 SDL-92의 데이타 기술 방법인 ADT 만을 지원하므로 SDL-92의 행위 기술을 변환하는 데는 한계점이 있었다.

앞에서 고찰한 바와 같이 SDL-92에서 객체지향 언어로의 연구는 기존의 C로의 연구를 바탕으로 C++에만 한정되어 있었다. 따라서 제시되었던 변환 방법은 SDL-92의 시그널과 전이에 기반한 행위 기술과 통신 방법을 C++ 내의 클래스와 멤버 함수로 사상하는 방법이 아니라, SDL-92 프로세스의 기술 구조를 그대로 유지한 채 외부 라이브러리의 지원으로 사상하는 방법을 사용하였다. 그러므로 기존의 연구에서는 SDL-92에서 객체지향 언어로의 변환에 있어 객체와 메소드를 기반으로 하는 변환 방법을 제시하지는 못하였다. 그리고 기존의 변환 방법은 SDL-92의 행위를 기술하는 특성들을 외부 라이브러리로 처리함으로써, 시그널과 대응하는 메소드로의 변환에서 발생하는 중첩된 시그널 수신, 시그널 저장의 처리, 그리고 프로세스 인식자(Pid, Process Identifier)의 사상과 같은 문제점에 대한 해결책을 제시할 수 없었다.

2.2 객체지향 언어 모델

구현 언어로 사용될 수 있는 객체지향 언어 모델은 현존하는 다양한 객체지향들이 공통적으로 가지고 있는 특성을 나타낼 수 있어야 한다. 그러나 이러한 모델을 정립하는 것은 매우 어려운 문제이므로 본 논문에서 가정하는 객체지향 언어 모델은 기본적으로 UML(Unified Modeling Language)의 클래스 다이어그램에서 사용하는 클래스, 오퍼레이션, 그리고 에트리뷰트로 이루어진 개념들을 사용하고, 그 특성을 따른다[17]. 그러나 개념 전달이 용이하도록 UML의 오퍼레이션은 메소드로, 에트리뷰트는 변수로 지칭하도록 한다.

그 외 SDL-92의 적용 분야인 분산 및 실시간 환경을 고려한 객체지향 언어 모델의 특성들은 아래와 같다.

(a) 객체들 간의 메소드 호출은 비동기 통신으로 이루어진다.

(b) 객체의 생성은 정적인 선언뿐만이 아니라 동적인 할당으로도 이루어질 수 있다.

(c) 시스템의 중첩된 구조를 허용하기 위해 가시성을 제어하는 모듈을 가정한다. 모듈의 특성은 CHILL-96과 Ada-95에서 사용하는 모듈 개념과 동일하다[18][19].

중첩된 구조 개념은 C++와 같은 객체지향 언어에서는 존재하지 않는 개념이지만, CHILL-96과 Ada-95와 같은 객체지향 언어에서는 모듈성과 판독성의 향상을 위해 사용하고 있다. 중첩된 구조를 중첩되지 않는 구조로 변경하는 것은 정의 및 선언에 사용된 이름의 확장으로 이루어질 수 있으므로 본 논문에서 사용하는 객체지향 언어 모델에서는 모듈과 클래스로 이루어진 중첩된 구조를 가정한다.

2.3 SDL-92의 객체지향 특성과 문제점

SDL은 시스템의 요구 사항인 명세 및 설계와 구현을 위한 기술을 위해 ITU-T에서 권고한 국제 표준의 형식 명세 언어로서 아래와 같은 장점을 가지고 있다. 여기서 명세는 시스템에서 요구되는 사항을 의미하며, 기술은 설계 및 구현된 구조를 의미한다.

(a) 언어적인 측면

- 정형적 기반을 가지고 있으므로 개발 과정에서 오류의 발생을 없앨 수 있다.
- 구조화되어 있으므로 정보 은닉을 제공한다.
- 그래픽 표현을 가지므로 읽기가 쉽고, 유한 자동기계의 기반에서 쉽게 이해된다.
- 다른 응용을 위한 그래픽 표현에 상응하는 문자 표현을 가지고 있다.

(b) 응용적인 측면
- ITU-T와 ETSI에서 권고하는 국제 표준 언어이다.
- 자동화된 여러 CASE 도구들의 지원으로 시스템의 개발이 용이하다.
- CIF(Common Interchange Format)을 통해 정보의 교환이 용이하다.

SDL-92는 1992년 Z.100에 의해 권고되었으며, 개체의 상속, 행위의 상속, 그리고 다형성으로 이루어지는 객체지향 개념을 도입하였다. 그러나 SDL-92에서 도입된 객체지향 개념은 기존의 SDL로 개발된 시스템의 명세 및 설계들을 포용하기 위해 기존의 개념에서 객체지향 개념으로의 발전이 아닌, 기존의 개념에 객체지향 개념을 추가하는 혼성적인 형태로 이루어졌다.

이러한 형태의 시그널과 프로세스를 기반으로 하는 객체지향 개념의 도입은 구현 언어로 메소드와 객체를 기반으로 하는 객체지향 언어를 사용할 때 정합하는 대응 개념들의 부재에 의해 아래와 같은 문제점들이 발생한다 .

가) 통신 방법의 차이

SDL-92는 시그널을 이용하여 통신을 수행하고, 프로세스 내부에서는 상태와 시그널의 수신으로 인해 행위들이 결정된다. 그러나 객체지향 언어의 경우 통신을 위해 메소드를 사용하고, 객체는 메소드의 호출에 의해 행위를 결정한다. 이러한 통신 방법의 차이는 아래와 같은 개념의 차이를 가지고 있다.

(가) 시그널 수신의 중첩성과 메소드 호출의 비중첩성

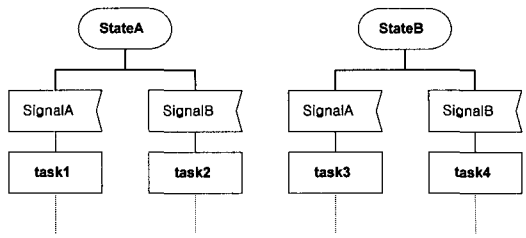
SDL-92에서는 내부에 상태를 가지는 프로시쥬어를 호출할 경우 프로세스에서 전이 중에 또 다른 시그널의 수신을 대기할 수 있다. 그러나 객체에서는 하나의 메소드가 수행되는 과정에서 또 다른 메소드의 호출을 대기할 수 없다.

(나) 상태 중심의 시그널 구성과 메소드 중심의 상태 구성

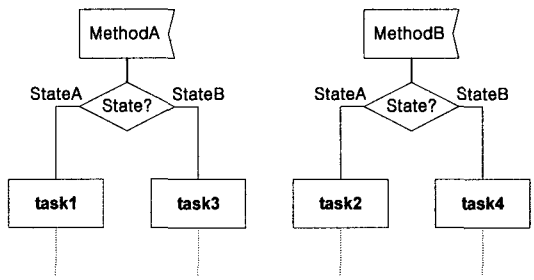
SDL-92의 프로세스는 그림 1의 (a)와 같이 하나의 상태와 그 상태에서 수신하는 시그널에 의해 행위를 결정한다. 이는 프로세스의 행위를 결정하는 첫번째 결정자가 상태가 됨을 의미하므로 SDL-92는 상태 중심의 시그널 구성을 가진다. 그러나 객체는 (b)와 같이 메소드가 호출된 후 객체의 내부 상태에 따라 수행될 행위를 결정한다. 따라서 첫번째가 결정자가 메소드가 되므로 메소드 중심의 상태 구성을 가진다.

(다) 시그널의 독립성과 메소드의 종속성

시그널의 독립성은 시그널이 특정 프로세스에 종속되지 않음을 말한다. 그러나 메소드들은 정의된 객체에 대해 종속적이다. 따라서 시그널의 송수신에 의한 통신을 메소드의 호출에 의한 통신으로 변환하기 위해서는 프로세스에 독립적인 시그널을 프로세스에 종속시켜야 한다.



(a) 상태 중심의 시그널 구성



(b) 메소드 중심의 상태 구성

그림 1 SDL-92와 객체지향 언어의 내부 구성의 비교

나) 프로세스 인식자 변수와 객체 참조형 변수의 차이

SDL-92에서 시그널 통신을 위해 사용하는 프로세스 인식자 변수는 어떠한 프로세스 인스턴스 값이라도 모두 할당 받을 수 있다. 반면에 대응되는 객체 참조형 변수는 동일 또는 계승 구조상 상위의 클래스로부터 실례화된 인스턴스의 값만을 할당받을 수 있다. 따라서 SDL-92의 하나의 프로세스 집합을 객체지향 언어의 하나의 클래스로 대응시키는 것은 1:N의 사상 관계를 가지게 된다.

다) 가시성의 차이

명세의 중첩과 정의의 지역화는 SDL-92의 기본적인 범위와 가시성 규칙이다. 따라서 명세 계층에서 상위의 선언과 정의들은 하위의 개체에서 모두 볼 수가 있다. 반면에 모듈과 객체는 가시성을 제어하는 단위가 되므로 동일한 가시성을 보장할 수 있는 방법이 필요하다.

라) 공유 변수의 개념 차이

SDL-92의 원격 변수는 값의 수출입 시점에 따라 동

기화 기능을 포함하고 있다. 따라서 이러한 원격 변수를 객체의 전역 변수로 사상하는 것은 실행시에 값의 차이를 유발할 수도 있다.

2.4 SDL-92 기술의 제한 사항

SDL-92로 이루어진 시스템의 명세 및 기술을 구현 언어로 번역함에 있어서, 서로 대응하는 개념이 존재하지 않을 경우, SDL-92에서는 존재하나 목적 언어에 존재하지 않을 경우, 이의 사용을 금지하여야 한다. 본 논문에서 가정한 객체지향 언어 모델에 의해 제한되는 SDL-92의 특성은 아래와 같다.

정의 1. 객체지향 언어를 위한 SDL-92 기술의 제한 사항

(a) 연속 시그널이 없어야 한다.
(b) 가능 조건이 없어야 한다.
(c) 임의 전이가 없어야 한다.
(d) 수출된 프로시쥬어(exported procedure)가 없어야 한다.
(e) 시그널의 저장이 있을 경우 시그널 저장의 제약 조건을 만족해야 한다.

객체지향 언어는 외부 메소드의 호출에 의해 자신의 동작을 결정한다. 따라서 프로세스 스스로 자신의 행위를 결정하는 연속 시그널과 가능 조건은 사용을 금지해야 한다. SDL의 수출된 프로시쥬어는 분산 환경에서 동기화된 원격 프로시쥬어 호출 기능을 수행하므로, 비동기 통신의 개념을 가지는 객체 간의 메소드 호출로 변환할 수 없다. 이를 위해서는 수출된 프로시쥬어의 정의 및 호출은 시그널 송수신을 사용하여 표현해야 한다 [11].

정의 2. 다음과 같은 제한 조건을 시그널 저장의 제약 조건이라고 한다.

(a) 시그널 저장의 유효성 : 한 상태에서 저장되는 시그널은 그 상태에서 직접적으로 전이가 가능한 최소한 하나 이상의 상태에서 입력 또는 시그널 저장에 사용되어야 한다.

(b) 시그널 저장의 유한성 : 저장된 시그널을 소모하는 순서에는 순환이 포함되지 않아야한다.

저장의 유효성은 시그널 저장이 사용될 시에 목시적으로 가해지는 제한 조건이다. 저장의 유한성은 본 논문에서 제시하는 시그널 저장의 제거 알고리즘의 종료를 보장하는 제한 조건이다. 만일 저장된 시그널의 소모 순서에 순환이 포함되면 무한개의 상태 추가가 이루어져야 하므로 유한성이 보장되어야 한다.

3. SDL-92에서 객체지향 구현 언어로의 개념 변환

3.1 개념 변환을 위한 자료 구조

SDL은 시스템의 명세와 프로세스의 기술 두 부분으로 이루어진다. 따라서 객체지향 언어로의 개념 변환을 위해서는 이 두 가지 측면에서 SDL을 표현할 수 있는 자료 구조가 필요하다.

가) 구조 트리

SDL-92의 시스템 명세 구조는 각 개체의 내부에 명세된 개체에 대해 이름만 기술한 후 외부에서 다시 명세하는 원격 구조나 내부에 개체의 명세를 포함하는 중첩 구조로 표현된다. 이러한 SDL-92의 원격 및 중첩된 명세 구조는 객체지향 언어의 중첩된 구조로 변환함에 있어 구조의 사상과 가시성의 변환을 어렵게 하므로 중첩된 구조로 일관되게 표현해야 한다.

구조 트리는 원격 및 중첩된 SDL-92의 시스템, 블록, 프로세스 명세 구조들을 계층 구조로 변환한 것이다.

정의 3. 구조 트리는 다음의 성질을 가지는 트리이다.

(a) 구조 트리는 하나의 루트 노드와 하나 이상의 루트가 아닌 내부 노드, 그리고 하나 이상의 단말 노드를 가진다.
(b) 루트 노드는 SDL-92의 시스템에 대응하는 정보를 가지며 시스템 노드라 부른다.
(c) 루트가 아닌 내부 노드는 SDL-92의 블록에 대응하는 정보를 가지며, 블록 노드라 부른다.

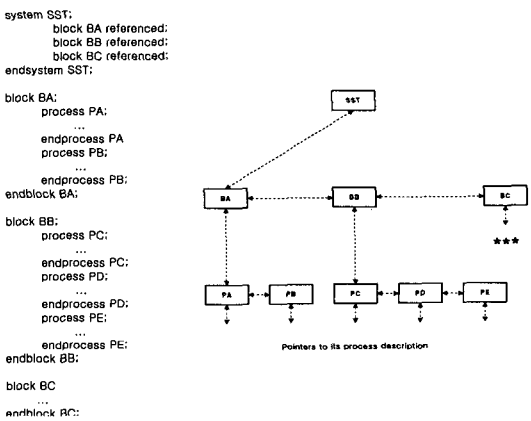


그림 2 구조 트리의 예

(d) 단말 노드는 SDL-92의 프로세스에 대응하는 정보를 가지며, 프로세스 노드라고 부른다. 프로세스 노드를 자식으로 가지는 노드를 단말 블록 노드라고 한다.

그림 2는 SDL/PR의 명세와 이에 대응하는 구조 트리의 예이다. 구조 트리는 좌장자-우형제 트리로 구성되며, 단말 노드인 프로세스 노드에서는 해당 프로세스의 전이를 기록하고 있는 CEFSM-C(Communicating EFSM with Condition node) 모델을 가리키는 포인터를 가진다.

나) CEFSM-C 모델

SDL-92의 프로세스는 시그널 수신에 의해 전이가 촉발되고, 전이 도중 내부 변수에 접근할 수 있는 CEFSM에 기반을 두고 있다[9][10][20]. 따라서 상태와 시그널의 수신 만으로 다음 상태를 결정할 수 있는 CFSM과는 달리, 전이의 수행 과정에서 동일한 상태와 수신 시그널을 가지는 상황에서도 접근된 변수의 값에 따라 서로 다른 상태로 전이가 가능하다[9][21]. 그러므로 시그널과 전이에 기반한 프로세스의 기술을 재구성하기 위해서는 상태와 상태 사이의 전이를 유일하게 결정하는 제어 조건을 나타낼 수 있는 SDL-92 시멘틱 모델이 필요하다.

CEFSM-C 모델은 프로세스 기술에서 상태 뿐만이 아니라 전이의 분할이 발생하는 결정문도 조건 노드(condition node)라는 상태로 표현하고, 결정문에서 전이의 분할을 결정하는 조건들을 연속 시그널의 형태로 표현함으로써, 프로세스에서 시그널 수신과 결정문에 의해 상태에서 상태로 전이하는 것을 유일하게 결정한다.

CEFSM-C 모델은 아래와 같이 정의된다.

정의 4. **CEFSM-C 모델**은 아래와 같이 입력 큐를 가진 9-튜플로 이루어진다.

$M = (I, V, S, s_0, F, Save, A, T, t_0)$

I : 입력 시그널들의 유한 집합

V : 변수들의 유한 집합

S : 상태들의 유한 집합

$s0$: 유일한 초기 상태, $s_0 \in S$.

F : 프로세스의 종료와 프로시쥬어의 종료 또는 복귀에 해당하는 종료 상태들의 유한 집합.

$Save$: $S \rightarrow$ powerset(I)로 표현되는 각 상태에서 유일한 시그널 저장 함수.

$Save(s_0) = Save(F) = \phi$.

A : 전이 중에 수행되는 행위들의 일련된 순서인 행위 블록의 유한 집합.

T : $T(E) \rightarrow S \times A$인 전이 함수.

$E = \{ < S, Event > \mid Event \in I \cup Condition(V) \cup I \times Condition(V) \}$이고, $Condition$ 은 V로 이루어진 논리식의 집합.

t_0 : 유일한 시작 전이.

CEFSM-C 모델은 수신되는 시그널의 저장을 위해 묵시적으로 입력 큐를 유지한다. 그리고 상태 집합 S에서 프로세스의 상태에 해당하는 상태들은 명시적 상태라고 하고, 결정문에 해당하는 상태를 조건 노드라고 할 때, 조건 노드에서 수신된 시그널은 다음 명시적 상태에서 처리할 수 있도록 입력 큐에 순차적으로 저장된다. 명시적 상태에서 수신된 시그널에 대해 해당 상태에서 입력 시그널로 명시되지 않았거나 $Save$함수에 의해 저장이 지정되지 않았다면 그 시그널은 단순히 소모된다. 이러한 전이를 묵시적 전이라고 하고, 그 외의 전이들을 명시적 전이라고 한다. 조건 노드에서는 시그널 수신이 아닌 연속 시그널에 의해 반드시 하나의 전이가 발생하므로 이러한 전이를 조건 전이라고 한다.

정의 5. CEFSM-C 모델에서 시그널로 촉발된 전이 순서(STTS, Signal-triggered transition sequence)란 한 명시적 상태에서 시작하여 다음 상태가 명시적 상태가 될 때까지 하나의 명시적 전이와 0개 이상의 조건 전이가 연결된 전이 순서를 말한다.

정의에 따라 시그널로 촉발된 전이 순서 $<s_1, event_1><s_2, event_2>\cdots<s_n, event_n>$에서 s_1은 명시적 상태를 나타내고 그 외의 상태는 조건 노드를 나타낸다. 그리고 s_i:$event_i$/$actions_i \rightarrow s_{i+1}$, $1 \le i \le n$이다. 만일 시그널로 촉발된 전이 순서에 포함된 조건 전이가 0번 이상 반복될 경우는 해당 전이를 "["와 "]"로 둘러싸고, 1번 이상 반복될 경우에는 해당 전이를 "{"와 "}"로 둘러싼다. 여러 개의 조건 전이중 하나가 가능함을 표현할 때는 "("와 ")"로 둘러싸고, "|" 로 해당 조건 전이들을 연결한다.

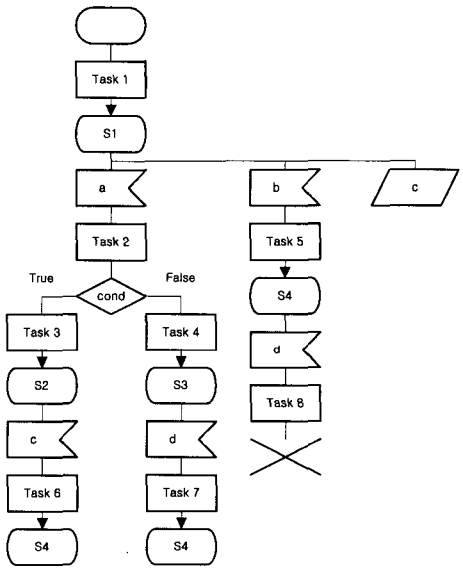
정의 6. SDL 전이 그래프(STG, SDL Transition Graph) G는 CEFSM-C 모델 M에 대응하는 아래와 같은 레이블을 가진 방향성 그래프이다.

(a) G의 노드 v와 M의 상태 s 사이에는 일대일 대응 관계가 존재한다.

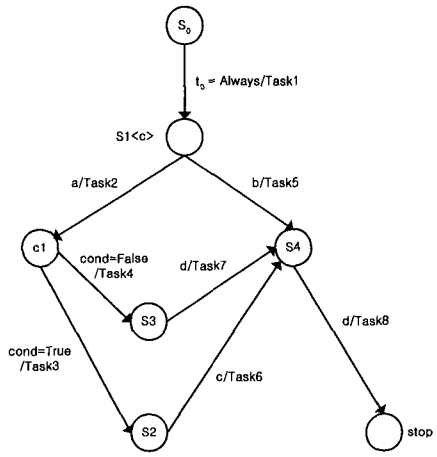
(b) G의 가지 e와 M의 전이 t 사이에는 일대일 대응 관계가 존재한다.

M의 상태 s에 해당하는 G의 노드 v는 $s<Save(s)>$

를 레이블로 가진다. 만일 $Save(s) = \phi$이면 $s<\phi>$는 간단하게 s로 표기된다. M의 전이 s:*event/actions* → t에 해당하는 G의 노드 $s<Save(s)>$에서 $t<Save(t)>$로의 방향성을 가지는 *event*와 이에 의해 수행되는 *actions*를 나타내는 *event/actions*를 레이블로 가진다. 만일 *event*에 의해 수행되는 *actions*가 없다면 *event*/λ는 간단하게 *event*로 표기된다. 프로세스 다이어그램과



(a) 프로세스다이어그램



(b) 대응 SDL 전이 그래프

그림 3 프로세스 다이어그램과 대응 SDL 전이 그래프의 예

정의에 따른 대응하는 SDL 전이 그래프는 그림 3과 같다.

3.2 SDL-92에서 객체지향 언어로의 개념 변환

SDL-92로 이루어진 시스템의 명세 및 기술을 구현 언어로 변환함에 있어서, 서로 상응하는 개념의 변환은 동일한 의미를 가질 수 있도록 대응 구문 사상의 형태로 이루어질 수 있다. 만일, 서로 대응하는 개념이 존재하질 않을 경우는, SDL-92에는 존재하나 목적 객체지향 언어에 존재하지 않는 경우, 시스템의 기술 시에 이러한 개념들의 사용을 금지하여야 한다. 그러나 프로세스들 간의 시그널을 이용한 통신과 같은 개념들은 SDL을 이용한 시스템의 기술에 필수적인 요소가 되므로, 이러한 개념들의 사용의 금지는 시스템의 기술을 불가능하게 한다. 따라서 이와 같이 시스템 기술에 필수적인 개념들은 다음에 기술한 바와 같이 개념 변환을 통해 목적 객체지향 언어의 구성 요소로 표현할 수 있도록 한다.

가) 시그널에서 메소드로의 개념 변환

SDL-92에서 프로세스 간의 행위 유발과 정보 교환은 시그널에 의해 일어나며, 시그널의 송신과 수신은 프로세스에 의해 일어난다. 객체지향 언어에서 이에 대응하는 요소들은 메소드와 객체가 되므로 그림 4와 같이 변환되어야 한다. 따라서 시그널에 기반한 통신과 메소드에 기반한 통신은 표 1과 같은 대응 관계를 가진다 [21][22][23].

프로세스의 전이를 객체의 메소드 정의로 변환하는 것은 첫번째 결정자가 상태가 되는 전이를 첫번째 결정자가 수신 시그널이 되도록 재구성함으로써 이루어진다 [25][26]. 그러나 EFSM 모델에서는 이러한 재구성이 수신 시그널을 중심으로 전이를 재구성하는 것으로 이루어질 수 있지만, SDL-92의 프로세스는 CEFSM-C 모델 기반으로 중첩된 시그널의 수신과 상태에서 수신 시그널의 저장과 같은 제어 흐름을 표현하는 특성을 가지고 있으므로 전이의 재구성 전에 이들을 제거하여야 한다.

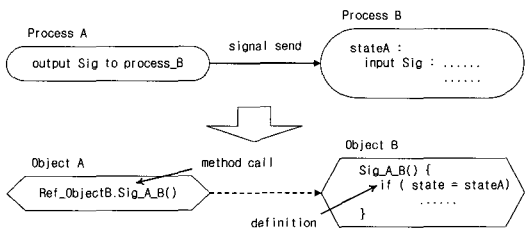


그림 4 시그널 기반 통신과 메소드 기반 통신의 대응 관계

표 1 SDL-92와 객체지향 언어의 대응 개념

SDL-92	객체지향 언어
프로세스	객체
상태	상태
시그널 송신	메소드 호출
시그널 수신	메소드 호출 수신
전이	메소드의 정의

(가) 시그널 수신의 중첩성 제거

시그널 수신의 중첩이란 시그널의 수신 후 전이가 수행되는 동안 또 다른 시그널의 수신을 대기하는 것을 말한다. SDL-92의 프로세스에서는 전이 과정에서 상태를 포함한 프로시쥬어를 호출할 경우에 중첩된 시그널의 수신이 발생한다. 이러한 전이는 메소드의 수행 중 다른 메소드의 호출은 가능하지만 다른 메소드의 호출을 대기할 수는 없기 때문에 메소드로 재구성할 수가 없다. 따라서 SDL-92의 프로시쥬어는 내부 상태의 유무에 따라 표 2와 같이 처리된다.

표 2 프로시쥬어의 처리

SDL-92	객체지향 언어
내부에 상태를 포함하지 않는 프로시쥬어	객체의 Private 메소드
내부에 상태를 포함하는 프로시쥬어	프로시쥬어의 분해 후 상위 수준 프로세스로 확장

중첩된 시그널의 수신을 없애기 위한 프로시쥬어의 분해는 해당 프로시쥬어 내의 상태들을 상위 수준 프로세스의 상태들로 확장하고, 해당 프로시쥬어에 대한 호출을 확장된 상태로의 전이로 변경함으로써 프로시쥬어의 호출과 동일한 전이가 발생하되 중첩된 시그널 수신은 발생하지 않도록 한다. 프로시쥬어의 분해를 위해 호출이 있는 프로세스의 전이와 프로시쥬어의 영역을 그림 5와 같이 구분한다.

알고리즘 1. 내부에 상태를 가지는 프로시쥬어의 분해

입력 : 프로세스와 분해할 프로시쥬어의 CEFSM-C 모델

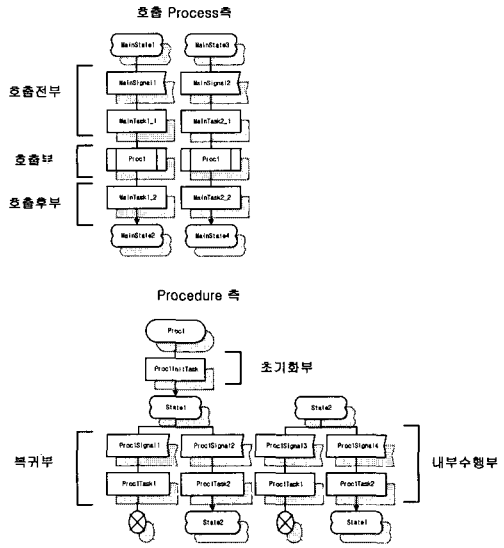


그림 5 프로시쥬어 분해를 위한 영역 구분

P_p = (I_p, V_p, S_p, s_{p0}, F_p, $Save_p$, A_p, T_p, t_{p0})

P_{pro} = (I_{pro}, V_{pro}, S_{pro}, s_{pro0}, F_{pro}, $Save_{pro}$, A_{pro}, T_{pro}, t_{pro0})

출력 : 중첩된 시그널 수신이 제거된 프로세스의 확장된 CEFSM-C 모델

P_E = (I_E, V_E, S_E, s_{E0}, F_E, $Save_E$, A_E, T_E, t_{E0})

제 1단계 : P_E의 초기화와 분해할 프로시쥬어의 검색

1.1 $P_E \leftarrow P_p$;

1.2 내부에 상태를 가진 프로시쥬어를 검색;

제 2단계 : 분해할 프로시쥬어 P_{pro}의 호출을 가진 전이들에 대해

2.1 현재 전이 정보 <*state*, *signal*>와 전이의 호출후부 → P_{pro}에 등록;

2.2 P_{pro} 대한 호출 → 매개변수 바인딩, 전이 정보의 저장, 그리고 t_{pro0}의 수행;

2.3 전이의 다음 상태 → P_{pro}의 첫번째 상태;

2.4 결과를 P_E에 반영;

제 3단계 : P_{pro}에 대해

3.1 t_{pro0}를 삭제하고 변수와 상태를 P_E에 추가;

3.2 복귀 전이를 제외한 모든 전이를 P_E에 추가;

3.3 제 2단계에서 저장된 전이 정보와 전이의 호출후부를 참조하여 복귀 전이를 재구성하고 이를 P_E에 추가;

정리 1. 프로시쥬어 분해로 확장된 프로세스의 타스

크의 수행은 분해 전의 프로세스와 프로시쥬어의 타스크 수행과 동일하다.

증명 주어진 프로세스와 프로시쥬어에 대해 CEFSM-C 모델 Pp와 Ppro을 아래와 같이 정의한다.

P_p = (I_p, V_p, S_p, s_{p0}, F_p, $Save_p$, A_p, T_p, t_{p0}).

P_{pro} = (I_{pro}, V_{pro}, S_{pro}, s_{pro0}, F_{pro}, $Save_{pro}$, A_{pro}, T_{pro}, t_{pro0}).

여기서 P_p의 전이 집합 T_p는 아래와 같이 이루어진다.

$T_p = T_{p\_call} \cup T_{p\_non\_call}$, T_{p_call}은 전이 중에 P_{pro}로의 호출이 존재하는 전이의 집합이고, $T_{p_non_call}$은 전이 중에 P_{pro}로의 호출이 존재하지 않는 전이의 집합이다.

전이 $t_i \in T_{p_call}$과 그의 행위 블록 $a_i \in A_P$는 아래와 같이 이루어진다.

$t_i = s_i \times event \rightarrow a_i \times s_j$, s_i and $s_j \in S_p$이고, $a_i = a_{i_befroe_call} \cdot call_action \cdot a_{i_after_call}$이다. 여기서 $a_{i_befroe_call}$은 프로시쥬어 T_{pro}에 대한 호출이 발생하기 전까지의 타스크의 일련된 수행을 나타내고, $a_{i_after_call}$은 프로시쥬어 T_{pro}에 대한 호출 후의 타스크의 일련된 수행을 나타낸다. *call_action* 은 T_p에 대한 호출을 나타낸다. 중위 연산자 "·"는 타스크의 연속된 수행을 나타내는 연산자이다.

그리고 P_{pro}의 전이 집합 T_{pro}는 아래와 같이 이루어진다.

$T_p = \{ t_{pro0} \} \cup T_{p\_non\_return} \cup T_{p\_return}$, 여기서 $T_{p_non_return}$은 복귀가 포함되지 않은 전이들의 집합이고, T_{p_return}은 복귀가 포함된 복귀 전이들의 집합이다.

전이 $t_j \in T_{p_return}$와 그의 행위 블록 $a_k \in A_{pro}$는 아래와 같이 이루어진다.

$t_j = s_k \times event \rightarrow a_{k_before_return} \times f_{pro}$이고, $a_k = a_{k\_before\_return}$, 여기서 $a_{k_before_return}$은 복귀 전이에서 복귀 명령어 전까지의 타스크의 일련의 수행을 나타낸다. a_k에 복귀 명령어가 포함되지 않는 이유는 T_{pro}에 대한 호출에서의 복귀가 T_{pro}의 종료 상태인 F_{pro}에서 자동적으로 일어나기 때문이다.

주어진 P_p와 P_{pro}에 대해 확장된 프로세스의 CEFSM-C 모델인 P_E는 아래와 같이 정의된다.

P_E = (I_E, V_E, S_E, s_{E0}, F_E, $Save_E$, A_E, T_E, t_{E0})

$I_E = I_p \cup I_{pro}$, $S_E = S_p \cup S_{pro}$, $s_{E0} = s_{p0}$, $F_E = F_p$, 그리고 $t_{E0} = t_{p0}$ 이다.

$V_E = V_p \cup V_{pro} \cup \{return\_state\_var\}$, 여기서 *return_state_var*은 P_{pro}의 수행 후 복귀하여야 하는 전이를 저장하는 상태 변수이다.

$Save_E$ = NULL, 프로시쥬어 의 분해 이전에 Save 심벌의 제거가 이루어지므로 이 단계에서 CEFSM-C 모델의 *Save* 함수들은 모두 NULL이 된다.

$T_E = T_{p_non_call} \cup T_{pro_non_return} \cup T_{E_new}$가 된다. T_{E_new}는 아래와 같이 새로이 정의된 전이들로 이루어진 집합이다.

$t_i \in T_{p_call}$와 $t_{pro0} \in T_{pro}$, 그리고 $t_{ret} \in T_{p_return}$에 대해 각각의 전이를 아래와 같이 나타낼 때,

$t_i = s_i \times event \rightarrow a_{i\_before\_call} \cdot call \cdot a_{i\_after\_call} \times s_j$

$t_{pro0} = s_{p0} \times \text{True} \rightarrow a_{pro0} \times s_{pro\_first}$

$t_{ret} = s_k \times event \rightarrow a_{k\_before\_return} \times f_{pro}$

$t_{E_new} \in T_{E_new}$는 다음과 같이 생성된다.

$t_{E\_new} = \{ s_i \times event \rightarrow a_{i\_before\_call} \cdot parameter\_binding \cdot a_{pro0} \times s_{pro\_first} \} \cup \{ s_k \times event \rightarrow a_{k\_before\_return} \times f_{pro} \} \cup \{ f_{pro} \times \langle return\_state\_var \rangle \rightarrow return\_assignment \cdot a_{i\_after\_call} \times s_j \}$

A_E는 T_E의 각 행위 블록들로 구성된다.

새로이 생성된 P_E가 P_p 그리고 P_{pro}와 동일한 타스크의 수행을 가짐은 아래와 같이 증명된다.

- P_p의 전이 중에서 P_{pro}에 대한 호출이 없는 경우 T_E는 $T_{p_non_call}$을 포함하므로 P_p에서의 타스크의 수행 순서와 P_E에서의 타스크의 수행 순서는 동일하다.
- P_p의 전이 중에서 P_{pro}에 대한 호출이 있는 경우 P_p와 P_{pro}에 의해 발생할 수 있는 타스크의 수행 순서는 아래와 같다.

$\{(t_{i\_1\_before\_call} \mid t_{i\_2\_before\_call} \mid \cdots \mid t_{i\_l\_before\_call})\} \cdot t_{i\_call} \cdot t_{pro0} \cdot [(t_{j\_1\_non\_return} \mid t_{j\_2\_non\_return} \mid \cdots \mid t_{j\_m\_non\_return})] \cdot \{(t_{k\_1\_return} \mid t_{k\_2\_return} \mid \cdots \mid t_{k\_n\_return})\} \cdot \{(t_{i\_1\_after\_call} \mid t_{i\_2\_after\_call} \mid \cdots \mid t_{i\_l\_after\_call})\}$

정의에 따르면 P_E에 의해 발생할 수 있는 타스크의 수행 순서는 아래와 같다.

$\{(t_{i\_1\_before\_call} \mid t_{i\_2\_before\_call} \mid \cdots \mid t_{i\_l\_before\_call})\} \cdot parameter\_binding \cdot t_{pro0} \cdot [(t_{j\_1\_non\_return} \mid t_{j\_2\_non\_return} \mid \cdots \mid t_{j\_m\_non\_return})] \cdot \{(t_{k\_1\_before\_return} \mid t_{k\_2\_before\_return} \mid \cdots \mid t_{k\_n\_before\_return})\} \cdot return\_assignment \cdot \{(t_{i\_1\_after\_call} \mid t_{i\_2\_after\_call} \mid \cdots \mid t_{i\_l\_after\_call})\}$

여기서 t_{i_call}은 *parameter_binding*과 동일하고, $t_{k_n_return}$은 $t_{k_n_before_return} \cdot return_assignment$와 동일하므로 P_p와 P_{pro}에서의 타스크의 수행 순서와 P_E에서의 타스크의 수행순서는 동일하다. 따라서 P_E는 P_p와

P_{pro}에서와 동일한 타스크의 수행 순서를 가진다.

제시한 알고리즘에 따라 그림 5를 분해한 예는 그림 6과 같다. 그림과 같이 Proc1에 대한 호출은 Proc1의 초기화부의 수행 후 확장된 상태 Proc1Sate로의 전이로 변환되고, Proc1의 복귀부는 전이의 남은 부분인 호출 후부들을 추가하여 재구성된다.

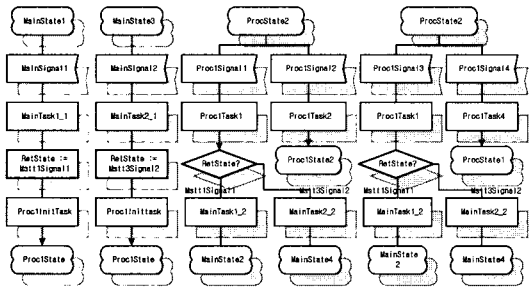


그림 6 그림 5의 프로시쥬어 분해 결과

(나) Save 심벌의 제거

SDL-92의 Save 심벌은 다른 명세 및 프로그래밍 언어와 SDL-92를 구분 짓는 중요한 특징 중의 하나로서 현 상태에서 수신된 시그널을 다음 상태에서 처리할 수 있도록 하여 준다[9]. 그러나 상태에서 특정한 시그널의 저장은 시그널 중심 상태 구성으로의 변환을 어렵게 하므로 메소드로의 변환 전에 제거되어야 한다.

기존의 Luo가 제시한 상태 확장을 통한 시그널 저장의 제거 알고리즘은 제한된 FSM 기반에서 동작하므로 언어 번역에 적용하기에는 제약이 많다. 따라서 본 논문에서는 CEFSM-C 모델 기반에서 시그널 저장의 제약 조건을 만족할 때 수행이 가능한 알고리즘을 제시한다.

알고리즘 2. CEFSM-C 모델을 이용한 SAVE 심벌의 제거

입력 : 명시적 상태에서 Save 함수를 가진 CEFSM-C 모델 M.

출력 : 명시적 상태에서 Save 함수가 제거된 CEFSM-C 모델 M.

제 1 단계 : M에 대응하는 SDL 전이 그래프 G를 작성.

제 2 단계 : M에서 저장된 시그널을 제거할 시그널로 촉발된 전이 순서를 검색.

FOR (M의 명시적 상태 s에 대응하는 G의 노드 v에 대해)

IF (v의 레이블에 저장되는 시그널의 이름이 포함)

FOR (s에서 각각의 시그널로 촉발된 전이 순서에 대해) BEGIN

u ← 시그널로 촉발된 전이 순서의 결과 상태에 해당하는 노드;

IF (u의 입력 시그널에 v에서 저장된 시그널이 존재)

각각의 저장된 시그널에 대해 제 3 단계를 호출;

ENDIF;

ENDBEGIN;

v의 노드 레이블에서 저장되는 시그널들을 삭제;

s의 Save 함수 ← NULL;

ENDIF;

제 3 단계 : G에서 상태의 확장으로 SAVE 심벌을 제거.

3.1 *save* ← v에서 저장된 u의 입력 시그널;

3.2 IF (v&*save*<Save(v)> 노드가 존재하지 않음) THEN

3.2.1 G에 노드 v&*save*<Save(v)>를 추가;

3.2.2 G에 가지 v:*save*/$\lambda \rightarrow$ v&*save*<Save(v)>를 추가;

3.2.3 노드 v&*save*<Save(v)>에 노드 v의 모든 시그널로 촉발된 전이 순서를 추가;

3.3FI;

3.4 v&*save*<Save(v)> 노드의 시그널로 촉발된 전이 순서 중에서 결과 노드가 u인 시그널로 촉발된 전이 순서에 u에서 *save*에 의해 발생되는 모든 시그널로 촉발된 전이 순서를 연결;

3.5 G의 변화를 M에 반영;

정리 2. 시그널 저장의 제약 조건을 만족하는 프로세스 다이어그램에서 SAVE 심벌의 사용은 상태 확장을 통해 제거할 수 있다.

증명 변환 전의 CEFSM-C 모델을 M, 변환 후의 CEFSM-C 모델을 M′ 라고 하자.

상태 확장을 통한 SAVE 심벌의 제거가 정당함을 보이기 위해 두 가지 경우로 나누어 생각한다. 표기는 알고리즘 2의 표기를 이용한다.

(a) 시그널의 저장과 관련이 없는 상태와 전이

알고리즘 2에서 시그널 저장의 제거는 3.2.1과 3.2.2에 의해 유한개의 새로운 상태와 전이의 추가만이 일어난다. 그리고 추가된 명시적 상태에서 3.2.3에 의해 가능한 모든 시그널로 촉발된 전이 순서들을 가지고 있다. 따라서 M′ ⊇ M 이므로 시그널의 저장과 관련이 없는

상태와 전이는 M과 M′ 가 동일하다.

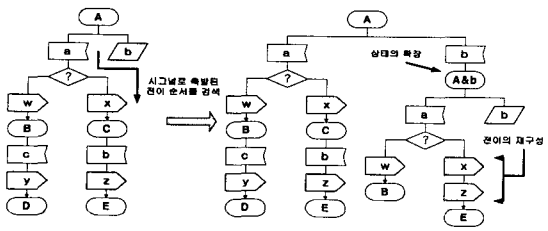
(b) 시그널의 저장과 관련된 상태와 전이

M의 v에서의 *save*의 저장과 입력 시그널 *event*의 수신 후 발생할 수 있는 전이는 다음과 같이 나타낼 수 있다. 아래에서 "·" 기호는 전이에 포함된 타스크문들의 연속된 수행을 나타낸다.

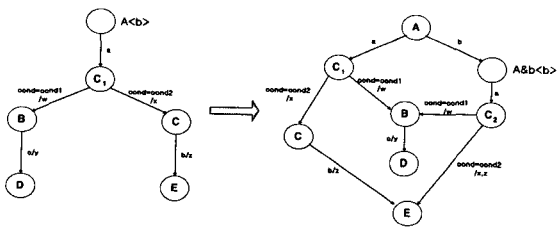
(v에서 u로의 시그널로 촉발된 전이 순서)·(u에서 *save*로 가능한 시그널로 촉발된 전이 순서 중 하나)

M′ 의 v에서 *save*의 수신으로 발생하는 전이는 3.2.2에 의해 v&*save*<Save(v)>로 행위가 λ 인 전이가 발생한다. 그리고 v&*save*<Save(v)>에서 입력 시그널에 의해 발생하는 전이는 3.2.3과 3.4에 의해 (v에서 u로의 시그널로 촉발된 전이 순서)·(U에서 *save*로 가능한 시그널로 촉발된 전이 순서 중의 하나)가 된다. 따라서 가능한 전이는 λ·(v에서 u로의 시그널로 촉발된 전이 순서)·(u에서 *save*로 가능한 시그널로 촉발된 전이 순서 중의 하나)가 되므로 M에서의 전이와 동일하다.

따라서 M과 M의 전이의 수행은 동일하다. 그러므로 시그널 저장의 제약 조건을 만족하는 M은 상태 확장을 통해 시그널의 저장이 제거된 M로 변환될 수 있다.



(a) 프로세스 다이어그램



(b) SDL 전이 그래프

그림 7 제시된 알고리즘에 따른 SAVE 심벌의 제거 예

제시된 알고리즘에 의해 SAVE 심벌의 제거를 수행한 예는 그림 7과 같다.

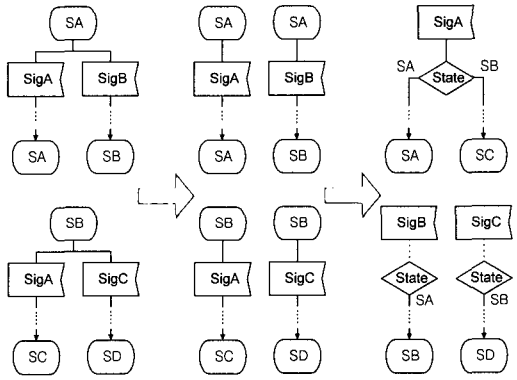


그림 8 전이의 재구성

(다) 시그널 중심 상태 구성으로의 재구성과 메소드로의 사상

시그널의 중첩 수신과 Save 심벌이 제거된 CEFSM-C 모델은 시그널로 촉발된 전이 순서의 결정 요인이 상태와 시그널로만 이루어진다. 시그널에서 메소드로의 개념 변환의 마지막 단계는 이러한 상태 중심의 시그널 구성을 그림 8과 같이 시그널 중심의 상태 구성을 가지는 시그널 중심 표기(SON, Signal-oriented Notation)로 재구성하는 것이다[25].

정의 7. 프로세스에서 시그널에 의한 전이 S는 다음과 같이 4-튜플로 이루어진다.

$$S = <s_1, Sig, action, s_2>$$

s_1는 전이 전의 상태, *Sig*는 수신되는 시그널, *action*은 수행되는 행위, 그리고 $s2$는 전이 후의 상태이다. S의 의미는 s_1 상태에서 *Sig* 의 수신 시에 *action*을 수행한 후 s_2 상태로 전이한다는 것이다.

상태 중심의 시그널 구성을 시그널 중심의 상태 구성으로 변환한 시그널 중심 표기에서 전이는 아래와 같이 정의된다.

정의 8. 시그널 중심 표기에서 전이 T는 다음과 같이 4-튜플로 이루어진다.

$$T = \ll Sig, s_1, action, s_2 \gg$$

Sig 는 수신되는 시그널, s_1은 전이 전의 상태, *action*은 수행되는 행위, 그리고 s_2는 전이 후의 상태이

다. T의 의미는 *Sig*를 수신한 후 현재 상태가 s_1이면 *action*을 수행한 후 s_2 상태로 전이한다는 것이다.

정리 3. 전이 S의 집합은 의미의 변화 없이 전이 T의 집합으로 변환된다.

Set(S) = Set(T)

증명 SDL-92에서 전이는 그림 8에서 보인 것 같이 (a)의 상태 중심과 (b)의 시그널 중심으로 나타낼 수 있다[25]. 여기서 (b)에서 공통의 입력 시그널을 가진 전이를 모으면 (c)와 같이 시그널 중심 표기로 변환된다. SDL-92에서 시그널 수신에 의한 전이의 발생은 아래와 같이 명시적 전이와 묵시적 전이 두 가지 상태만을 가진다.

(a) 명시적 전이 : 상태 중심의 구성이나 시그널 중심의 구성에서 명시적 전이가 발생한다는 것은, 현재 상태에서 수신 가능한 시그널 중의 하나가 수신되어 다른 상태로 전이한다는 것을 의미한다. 이것은 시그널 중심 표기에서 수신된 시그널에 의한 전이가 현재 상태를 포함하고 있으면 해당 전이가 발생하는 것과 동일하다.

(b) 묵시적 전이 : 상태 중심의 구성이나 시그널 중심의 구성에서 묵시적 전이가 발생한다는 것은, 현재 상태에서 수신이 가능하지 않는 시그널 중의 하나가 수신되어 시그널의 소모만 발생한다는 것을 의미한다. 이것은 시그널 중심 표기에서 수신된 시그널에 의한 전이가 현재 상태를 포함하지 않으면 아무런 전이가 발생하지 않는 것과 동일하다.

따라서 시그널 중심 표기에서의 전이는 SDL-92에서 시그널에 의한 전이와 동일하다.

나) 프로세스 인식자 변수의 사상

프로세스들 간의 시그널 송수신은 Pid 형의 변수를 사용하여 수행된다. 이러한 Pid 형은 그림 4에서 보인 바와 같이 객체의 참조형에 대응된다. 그러나 Pid 변수가 모든 프로세스의 Pid 값을 다 할당 받을 수 있는 반면에 객체의 참조형 변수는 클래스 계층 구조상 동일 또는 하위의 클래스에서 실례화된 객체의 참조값 만을 할당 받을 수 있다. 따라서 Pid 변수와 객체의 참조형 변수의 대응 관계는 1:N이 된다.

그러나 이러한 형태의 사상은 개념상으로는 정당하나 실시간에 발생할 수 있는 SENDER변수의 동적 할당과 같은 부분을 코드 생성시에 정적으로 결정해야 된다는 문제점을 가지고 있다. 이를 해결하기 위해서는 프로세스 내의 하나의 전이를 동일한 내용을 가지지만 서로 다른 객체 참조형 변수를 사용하는 전이들로 세분화하여야 하고, 이는 중복된 코드의 생성으로 이어진다[26].

따라서 본 논문에서는 이러한 중복된 코드의 생성을 피하기 위해 객체의 계승에 따른 동적 바인딩에 착안한 프로세스 인식자 변수의 사상 방법을 제안한다. 즉, 그림 9와 같이 추상 클래스 *AbstracProcessClass*를 정의한 후, 프로세스 집합에 대응하는 모든 클래스들을 *AbstractProcessClass*로부터 계승 받아 정의하도록 한다. 그리고 SDL-92의 모든 프로세스 인식자 변수와 내부 변수들은 *AbstractProcessClass*의 참조형으로 사상된다.

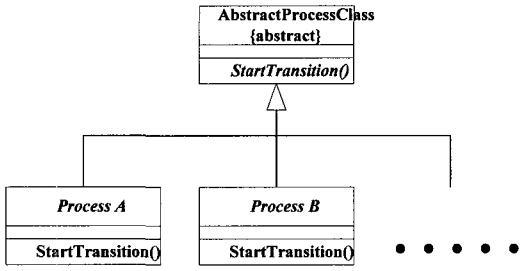


그림 9 프로세스 인식자 변수의 사상을 위한 클래스의 계승 구조

다) 가시성의 변환

SDL-92의 가시성 규칙은 명세의 중첩과 정의의 지역화에 의해 결정된다. 즉, 하위의 단위들은 자신들이 속한 상위의 단위의 선언 및 정의에 대해 가시성을 가지고, 중첩 구조에서 동일한 이름이 사용될 때는 하위 단위의 이름이 우선적으로 적용된다. 변수의 선언은 프로세스 이하 수준에서만 가능하다.

이에 비해 제시된 객체지향 모델에서는 모듈과 클래스에 의한 가시성을 제어하는 중첩 구조를 가지고 있으므로 동일한 가시성을 보장하기 위해서는 하위 단위에서 상위 단위의 정의 및 선언들을 수입하는 과정이 필요하다. 이러한 과정은 구조 트리에서 깊이 우선 탐색으로 아래와 같은 절차를 거쳐 이루어진다.

알고리즘 3 가시성의 변환을 위한 Seize 집합의 결정

입력 : 가시성 제어가 되지 않은 구조 트리;

출력 : 가시성 제어가 이루어진 구조 트리와 각 노드의 가시성 획득 집합 *Seize*;

제 1단계 : 구조 트리를 깊이 우선 탐색하면서 각 노드에서 제 2단계를 호출

제 2단계 : 각 노드에서의 가시성 제어

DO FOR 부모 노드의 *Seize* 집합에 존재하는 모든 항목에 대해;

IF (현재 노드의 선언 및 정의 항목과 충돌이 발생하지 않음)
현 노드의 *Seize* 집합 ← 현재 항목을 추가;
ENDIF;
ENDDO;

Seize 집합에 포함된 항목에 대한 처리는 아래와 같이 목적 객체지향 언어의 특성에 따라 달라진다.

(a) 가시성을 제어하는 모듈 구조를 가지는 경우 : 제어 단위 간의 가시성 제어 명령으로 사상

(b) 가시성을 제어하는 모듈 구조를 가지지 않는 경우 : 블록 노드들에 존재하는 선언과 정의들을 시스템 노드로 이름의 확장을 통해 이동시키고, 각 프로세스 노드에 있는 *Seize* 집합 내의 각 항목이 정의되어 있는 블록 노드에 적용된 이름의 확장 방법을 해당 항목에 적용

*AbstractProcessClass*에 대한 정의는 구조 트리의 루트 노드인 시스템 노드에서 이루어지고, 각 단말 노드인 프로세스 노드까지 모든 노드에서 *Seize* 집합에 추가된다. 이는 명시적인 프로세스 인식자 변수에 대한 선언이 없더라도 내부 변수의 활용에 사용되기 때문이다.

라) 공유 변수의 사상

동일 블록 내의 프로세스들 간의 변수 공유는 REVEAL/VIEW로 이루어진다. 이러한 변수 공유는 클래스 내의 PUBLIC 변수로 사상한다.

공통의 상위 블록을 가지는 프로세스들 간의 변수 공유는 REMOTE 선언과 EXPORT/IMPORT의 수행을 통해 이루어진다. 그러나 원격 변수는 값의 공유 외에도 EXPORT/IMPORT의 수행 시점에 따른 값의 동기화 기능을 가지므로 REVEAL/VIEW와 같이 사상할 때는 실행시에 미묘한 값의 차이를 유발할 수도 있다. 따라서 아래와 같이 사상함으로써 값의 공유 외에도 값의 동기화를 지원할 수 있도록 한다.

(a) 원격 변수

수출 측의 변수 *RemoteVariable*로 사상

(b) 수입 프로세스 측

- 원격 변수의 값을 돌려주는 메소드 *Return-RemoteVariable*를 정의
- 수출 프로세스 측에 정의된 *RemoteVariable*에 대한 값을 요청하는 메소드 *RequestRemote-Variable*호출
- 자신에서 정의한 *ReturnRemoteVariable* 송신 메소드 호출 대기
- 매개변수로 전달된 *RemoteVariable*의 값 수신

(c) 수출 프로세스 측

- *RemoteVariable*에 대응하는 변수 *Variable*선언
- 수출의 수행시 *Variable*의 값을 *RemoteVariable*에 복사하는 메소드 *RequestRemoteVariable*정의
- *RequestRemoteVariable*의 호출시 *Remote-Variable*의 값을 *ReturnRemoteVariable*의 매개변수로 전달

4. 결 론

본 논문에서는 SDL-92로 이루어진 시스템의 명세 및 기술 문서로부터 타겟팅을 위한 객체지향 언어로의 번역에 따른 문제점들을 고찰하고, 이를 해결할 수 있는 변환 모델과 개념 변환 방법을 제시하였다. 첫째, 객체지향 언어 모델의 제시를 위해 UML의 클래스 다이어그램을 기반으로 하여, SDL-92가 적용되는 분산 실시간 환경의 특성을 포함하는 객체지향 언어 모델을 제시하였다. 그리고 이를 목적 언어로 할 때 문제점을 발생시키는 SDL-92의 객체지향 특성들을 고찰하였고, 사용이 제한되어야 하는 SDL-92의 제한 사항들을 제시하였다. 둘째, 개념 변환을 위해 SDL-92의 명세 측면과 기술 측면을 나타낼 수 있는 구조 트리와 CEFSM-C 모델을 제시하였다. 구조 트리는 SDL의 시스템, 블록, 그리고 프로세스로 이루어진 명세 구조를 중첩된 트리 구조로 나타내었고, CEFSM-C 모델은 상태, 시그널 수신, 그리고 전이로 이루어진 프로세스의 기술 구조를 나타내었다. 그리고 마지막으로 제시된 모델들을 기반으로 하여 중첩된 시그널 수신과 시그널 저장의 제거를 통한 시그널에서 메소드로의 변환, 프로세스 인식자 변수의 사상, 가시성의 변환, 그리고 공유 변수의 사상으로 이루어진 SDL-92에서 객체지향 언어로의 개념 변환 방법을 제시하고, 정당성을 증명하였다.

제시된 개념 변환 방법은 기존의 C++과 같은 외부 라이브러리 지원으로 이루어진 번역 방법과는 달리 객체지향 언어 내의 개념과 문법 구조를 사용하여 SDL-92의 통신 방법을 변환하는 방법을 제시함으로써 번역된 프로그램에 대해 컴파일러 수준의 이식성을 제공하고, 이에 따라 동작 환경에 대한 독립성을 보장할 수 있다. 그리고 제시된 개념 변환 방법이 특정 객체지향 언어의 구문을 기반으로 하지 않고, EFSM 기반으로 이루어지므로 전이의 재구성을 통해 다양한 객체지향 언어로의 변환에 쉽게 적용될 수 있는 장점이 있다. 또한 CEFSM-C 모델 기반에서 제시된 시그널 저장의 제거 알고리즘은 구현 언어로의 변환 뿐만이 아니라 SDL-92로 작성된 시스템 명세의 검사와 자동 검사 예

의 생성과 같은 분야에도 유용하게 사용할 수 있다.

향후 연구 과제는 본 논문에서 제시한 개념 변환과 사상에 대한 연구를 기반으로 하여 각 객체지향 언어의 구조와 구문을 활용할 수 있는 방법에 대한 연구이다. 본 논문의 연구 결과는 특정 객체지향 언어를 목적으로 하지 않으므로, 프로그래밍의 효율성을 위해 각 언어에서 제공하는 프로그램 구조와 특수한 구문을 최대한 활용할 수 있는 특화된 변환 방법과 사상에 대한 연구가 더 수행되어야 할 것이다.

참 고 문 헌

[1] K. E. Cheng and L. N. Jackson, MELBA+: AN SDL SOFTWARE ENGINEERING ENVIRONMENT, , *The fourth SDL Forum, SDL 89: The Language at Work*, pp.95-103, 1989.

[2] 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.

[3] R. R. Singh, and J. Serviss, Code generation Using GEODE: A CASE Study, *The ninth SDL Forum, SDL97: TIME FOR TESTING-SDL, MSC and Trends*, pp.539-550, 1997.

[4] SINTEF Telecom and Informatics, *TIMe 4.0 - The Integrated Method*, 1999.

[5] Telelogic, *Telelogic Tau 3.5 Manual*, 1998.

[6] A. Valenzano, R. Sisto, and L. Ciminiera, Derivation of Executable Code from Formal Protocol Specifications Written in LOTOS, *The tenth International Phoenix Conference on Computers and Communications, Phoenix*, pp.346-352, 1991.

[7] S. Budkowski and P. Dembinski, An Introduction to Estelle: A Specification Language for Distributed System, *Computer Network and ISDN System*, Vol.14, No.1, pp.2-23, 1987.

[8] ITU-T, *Specification and Description Language(SDL)*, ITU Recommendation Z.100, 1992.

[9] R. E. Miller and Y. Xue, Bridging Gap Between Formal Specification and Analysis of Communication System, *IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications*, pp.225-231, 1996.

[10] G. Luo, A. Das, G. V. Bochmann, Software Testing Based on SDL Specifications with Save, *IEEE Transaction on Software Engineering*, Vol.20, No.1, pp.72-87, 1994.

[11] A. Olsen, O. Færgemond, B. Møller-Pedersen, R. Reed and J. R. W. Smith, *Systems Engineering Using SDL-92*, ELSEVIER SCIENCE B.V., 1995.

[12] N. Durate, H. Galhardas, R. Carapuca, and M. M. Marques, Towards an Integrated SDL/CHILL Environment, *The fifth SDL Forum, SDL 91:Evolving Methods*, pp.379-392, 1991.

[13] J. Fisher, E. Holz, M. v. Lowis, and D. Witaszek, A Run Time Library for the Simulation of SDL92-Specification, *The sixth SDL Forum, SDL 93: Using Objects*, pp.105-118, 1993.

[14] G. Gries, SDL++ - A Toolset for the Object-Oriented Development of C++ Software, *The sixth SDL Forum, SDL 93: Using Objects*, pp.119-128, 1993.

[15] E. Inocencio and M. M. Fonseca, SDL to C++ Translator for ISDN Basic Rate Terminal Signaling, *The sixth SDL Forum, SDL 93: Using Objects*, pp.349-360, 1993.

[16] N. Mansurov, A. Kalinov, A. Ragozin, and A. Chernov, Design Issues of RASTA SDL-92 Translator, *The seventh SDL Forum, SDL 95: with MSC in CASE*, pp.165-174, 1995.

[17] G. Booch, J. Rumbaugh, and I. Jacobsen, *Unified Modeling Language User guide*, Addison Wesley Longman, 1998.

[18] ITU-T, CCITT HIGH LEVEL LANGUAGE(CHILL), ITU Recommendation Z.200, 1996.

[19] L. Guerby, *hypertext Ada95 Rationale*, http://lglwww.epfl.ch/Ada/rat95, 1996.

[20] T. Ramalingon, K. Thulasiraman, and A. Das, Context Independent Unique Sequences Generation for Protocol Testing, *INFOCOM 96*, Vol.3, pp.1141-1148, 1996.

[21] D. Y. Lee and J. Y. Lee, A Well-described Estelle Specification for the Automatic Test Generation, *IEEE Transaction on Computer*, Vol.40, No.4, pp.526-542, 1991.

[22] A. Scortesse, OO_CHILL : INTEGRATING THE OBJECT PARADIGM INTO CHILL, *Proceedings of the 5th CHILL conference*, Rio de Janeiro, Brazil, pp.111-117, March, 1990.

[23] G. Booch, *Object-Oriented Analysis and Design with Applications*, 2nd Ed., The Benjamin/Cummings Publishing Company, Inc., 1994.

[24] J. F. H. Winkler and G. Dießl, Object CHILL An Object Oriented Language for Telecom Applications, *International Switching Symposium 92*, Vol. 2, pp.204-208, 1992.

[25] 이시영, 김성재, 이동길, 이준경, 김승호, "SDL-92에서 CHILL-96으로 변환하기 위한 사상 규칙의 설계", *정보과학회 논문지*, Vol. 25, No. 8, pp.12931303, 1998.

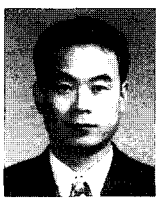
[26] S. Y. Lee, D. G. Lee, J. K. Lee, and S. H. Kim, Conceptual Transformation from SDL-92 to CHILL-96 using Signal Subordination, *The sixth International Conference on Real-Time Computing Systems and Applications(RTCSA'99)*, accepted, 1999.

[27] L. Cardelli and P. Wegner, On Understanding Types, Data, Abstraction, and Polymorphism, *ACM Computing Surveys*, Vol. 17, No. 4, pp.471-522, 1985.

이 시 영
1995년 경북대학교 컴퓨터공학과 졸업(학사). 1997년 경북대학교 대학원 컴퓨터공학과 졸업(공학석사). 1997년 ~ 현재 경북대학교 대학원 컴퓨터공학과 박사과정. 1996년 ~ 현재 한국전자통신연구원 S/W 개발환경 위촉연구원. 관심분야는 소프트웨어 공학, 컴파일러, 개발환경 및 영상처리 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 준 경
1985년 서강대학교 물리학과 졸업(학사, 부전공 전산학과). 1987년 숭실대학교 전자계산학과 졸업(공학석사). 1997년 숭실대학교 전자계산학과 졸업(공학박사). 1987년 ~ 현재 한국전자통신연구원 선임연구원. 관심분야는 컴파일러 구성론, 프로그래밍 언어론 및 개발환경, 객체지향 시스템 환경, 소프트웨어 공학, 운영체제 등임.



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