객체기반 소프트웨어 프로세스 프로그래밍을 위한 SimFlex 언어의 구조

김 영 곤[†] · 이 명 준[††] · 강 병 도[†††]

요 약

소프트웨어 프로세스는 소프트웨어의 생산에 사용되는 행위, 규칙, 절차, 기법, 도구의 집합체로 정의할 수 있다. 소프트웨어 프로세스 모형은 실세계 소프트웨어 프로세스의 개념적 표현이며 프로세스 프로그래밍 언어로 기술될 수 있다. 본 논문에서는 객체기반 소프트웨어 프로세스 프로그래밍을 위해 설계된 SimFlex 언어구조를 제시한다. SimFlex의 설계는 객체의 개념에 기반을 두고 있으므로 문법적으로나 구문적으로 복잡한 소프트웨어 프로세스를 간결하게 모형화할 수 있다. SimFlex의 언어구조는 주요 프로세스 중심 소프트웨어 개발환경 및 연관된 프로세스 프로그래밍 언어의 분석으로부터 도출되었으므로 SimFlex는 바람직한 객체기반 프로세스 프로그래밍 언어를 위해 필요한 핵심적인 특성을 포함한다. 더욱이 SimFlex는 적절한 적합화를 통하여 특정 프로세스 중심 소프트웨어 개발환경에 포함될 수 있는 기반 프로세스 프로그래밍 언어로 동작할 수 있도록 설계되었다.

On the SimFlex Language Constructs for Object-Based Software Process Programming

Young-Gon Kim[†] · Myung-Joon Lee[††] · Byeong-Do Kang[†††]

ABSTRACT

The software process can be defined as the set of activities, rules, procedures, techniques and tools used within the production of software. A software process model is a conceptual representation of a real world software process and can be described by process programming languages. In this paper, we present the language constructs of SimFlex designed for object-based software process programming. The design of SimFlex is based on the object concept, so that it can model complex software processes concisely both in syntax and semantics. Since the language constructs of SimFlex are derived from the analysis of major PSEEs and their associated process programming languages, SimFlex includes the core characteristics required for a desirable object-based process programming language. In addition, SimFlex is designed to act as a template software process definition language which could be included in specific PSEEs through customization appropriate to those PSEEs.

※본 논문은 한국전자통신연구원에서 수행한 "정보통신 서비스 개발 프로세스 모형화 기술개발" 과제의 위탁 결과물입니다.

† 준 회 원:울산대학교 전자계산학과
†† 정 회 원:울산대학교 전자계산학과
††† 정 회 원:한국전자통신연구원 소프트웨어공학 연구실
논문접수:1997년 5월 6일, 심사완료:1997년 9월 11일

1. 서 론

지난 수년동안 잘 정리된 소프트웨어 개발 프로세스(software development process)의 적용이 소프트웨어 개발시간을 단축하고 따라서 소프트웨어 생산물의 질을 향상시킬 수 있다고 생각되어 왔으며 소프트웨어 개발 프로세스를 소프트웨어 생산을 향상시키는 하나의 도구로 간주하여 많은 관심을 기울여 왔다. 이 결과로 실제의 생산에는 크게 적용되지는 않았지만 상당수의 프로세스 중심 소프트웨어 개발환경(PSEE: Process-centered Software Engineering Environment)이 설계되고 구축되었다[1, 2, 3].

프로세스 중심 소프트웨어 개발환경에서는 다양한 단계의 추상화(abstraction)와 입자성(granularity), 그리고 프로세스의 여러 측면을 표현하기 위하여 하나 또는 그 이상의 프로세스 프로그래밍 언어로 기술되는 명시적인 프로세스 정의가 사용된다. 이러한 프로세스 모형은 일반적으로 프로세스 중심 소프트웨어 개발환경의 저장소에 저장되며, 관련된 사용자에게 적절한 지원을 제공하기 위하여 프로세스 중심 소프트웨어 개발환경내에서 수행될 수 있다[3, 4].

소프트웨어 프로세스(software process)를 명시적으로 기술하고 수행시키기 위해서는 프로세스 프로그래밍 언어(process programming language)가 필요하다. 현재 많은 프로세스 중심 소프트웨어 개발환경이 자신의 고유한 프로세스 프로그래밍 언어를 소개하고 있지만 이러한 언어들은 그들이 속한 프로세스 중심 소프트웨어 개발환경의 특성에 의존하여 기본적인 개념이나 기법, 언어 구조, 포함되어야 할 핵심적인 특성에 대해서는 거의 동의가 이루어지지 않고 있는 실정이다[4].

본 연구에서는 현재까지 소개된 주요 프로세스 프로그래밍 언어를 분석하여 프로세스 프로그래밍 언어가 제공하여야 하는 특성과 언어구조를 도출하고, 이를 바탕으로 문법과 의미가 간결하면서도 풍부한 표현을 제공하는 새로운 프로세스 프로그래밍 언어 *SimFlex*를 제안한다. 다른 프로세스 프로그래밍 언어와는 달리 SimFlex는 프로세스 모형(process model)의 재사용과 캡슐화를 위하여 프로세스 모형의 계층적 조합(hierarchical composition)을 제공하며, 관련된 저수준의 환경에 대한 적절한 적합화(customization)를 통하여 특정 프로세스 중심 소프트웨어 개발환경에 연관될 수 있는 기반 프로세스 프로그래밍 언어로 동작할 수 있도록 설계되었다.

본 논문의 구성은 다음과 같다. 2장에서는 프로세스 중심 소프트웨어 개발환경 및 프로세스 프로그래밍 언어의 특성에 대하여 살펴본다. 3장에서는 객체기반 프로세스 프로그래밍 언어가 가지고 있어야 할 특성을 객체기반 프로세스 프로그래밍 언어인 APPL/A[3], E^3[2, 5], MASP/DL[2], SPELL[2, 6] 언어에 비추어 차례대로 기술한다. 4장에서는 지금까지 살펴본 내용을 바탕으로 설계된 SimFlex 언어의 구조를 간략히 소개하고, 소프트웨어 프로세스에 대한 국제적인 표준 예제로서 받아들여지고 있는 *ISPW-6*[7, 8] 예제에 부분적으로 적용하여 SimFlex 언어의 타당성을 살펴본다. 마지막으로 5장에서는 결론 및 추후 연구방향에 대하여 설명한다.

2. 프로세스 중심 소프트웨어 개발환경 및 프로세스 프로그래밍 언어

이 장에서는 소프트웨어를 개발하고자 하는 개발자 및 관리자에게 여러가지 유용한 정보를 제공해 주기 위해 사용하는 컴퓨터 기반 시스템인 프로세스 중심 소프트웨어 개발환경에 대하여 기술하고 소프트웨어 프로세스를 명시적으로 기술하기 위한 프로세스 프로그래밍 언어 및 객체기반 프로세스 프로그래밍언어에 대하여 소개한다.

2.1 프로세스 중심 소프트웨어 개발환경

지난 몇 년 동안 소프트웨어 프로세스는 소프트웨어 생산물의 질을 향상시키기 위한 도구로서 많은 사람들의 관심을 받아왔으며 여러 프로세스 중심 소프트웨어 개발환경이 설계되고 구축되어 왔다[1]. 소프트웨어 프로젝트의 크기와 복잡도가 증가함으로 인하여 모든 프로젝트 진행 과정을 적절히 조절하고 구성원 사이의 관계를 잘 조화시키는 작업이 점점 어려워짐에 따라 소프트웨어를 개발하는 사람에게 여러가지 유용한 정보를 제공하기 위하여 컴퓨터를 이용한 시스템이 고려되었는데 이를 프로세스 중심 소프트웨어 개발환경(Process-centered Software Engineering Environment)이라 한다. 프로세스 중심 소프트웨어

개발환경은 소프트웨어 생산물의 질을 향상시킬 뿐만 아니라 자동화된 개발 과정을 제공한다. 소프트웨어 프로세스의 목적은 높은 품질의 소프트웨어를 보다 빠르고 적은 비용으로 개발하는데 있으므로 이 목적을 달성하기 위하여 소프트웨어 프로세스는 소프트웨어로서 표현이 되어야 하며, 엄격한 소프트웨어 공학적인 환경으로부터 영향을 받는 것은 바람직하다고 하겠다.

이러한 프로세스 중심 소프트웨어 개발환경들 중 Cap의 Process Weaver[2], ICL의 Process Wise Integrator[8]등 몇몇 프로세스 중심 소프트웨어 개발환경은 상업화되어 발표되었다. 프로세스 중심 소프트웨어 개발환경의 프로세스는 행위(activity)와 이 행위에 관련되어 있는 생성물(product) 및 리소스(resource)들의 집합체이며 생성물의 생성을 위한 생산 프로세스(production process), 프로세스의 진보(evolution)를 위한 메타 프로세스(meta process), 그리고 프로세스 모델과 다양한 프로세스 도구로 구성되는 프로세스 지원 환경(process support system)으로 구성된다. 이 중 프로세스 지원 환경은 프로세스에 대한 명시적 정의인 프로세스 모형을 수정하고 수행시킬 수 있으며 프로세스 모형화 도구와 프로세스 모형을 수행시킬 수 있는 프로세스 엔진(process engine)으로 구성된다.

2.2 프로세스 프로그래밍언어

소프트웨어 프로세스를 형식화하고 구문적인 형태로 기술하기 위한 프로세스 프로그래밍 언어는 소프트웨어 프로세스의 각 단계에 대한 추상화를 제공하며, 소프트웨어 생산에 필요한 여러 요소들을 기술할 수 있도록 한다. 이 프로세스 프로그래밍 언어에 의해 기술된 프로세스 모형은 프로세스 프로그래밍 언어의 번역기(language interpreter)에 의하여 내부적으로 표현 가능한 프로세스로 번역이 되고, 이를 수행지원도구가 수행시킴으로써 최종 사용자는 자신이 원하는 작업을 프로세스 중심 소프트웨어 개발환경 하에서 기술된 프로세스에 적합하게 수행할 수 있게 된다.

현재까지 제안된 여러 프로세스 프로그래밍 언어들을 그 기법상으로 나누어 보면 〈표 1〉과 같다.

오늘날 소프트웨어의 개발 추세를 볼 때 프로세스 모형이 복잡하고 이해하기 어렵기 때문에 구조화된 기법의 지원이 필수적이라 볼 수 있으며, 객체지향 접근법이 소프트웨어 프로세스 모형 프로그래밍에 다소간 성공적으로 적용되고 있다[2, 5]. 이러한 추세와 아울러 여러 프로세스 프로그래밍 언어들의 분석을 바탕으로 새로운 프로세스 프로그래밍 언어를 설계하였다. 본 논문에서 제시하는 새로운 프로세스 프로그래밍 언어인 SimFlex는 객체기반언어로 설계되었으며, 이는 객체로 모든 기반 요소들을 표현하는 것이 실세계를 잘 반영해 줄 수 있으며 또한 보통의 사람들이 이해하기 쉽도록 간결한 형태로 소프트웨어 프로세스를 기술할 수 있다는 생각에 바탕을 둔 것이다.

〈표 1〉 프로세스 프로그래밍 언어
〈Table 1〉 Process Programming Languages

명령문 언어	대부분의 언어에 복합되어 사용되고 있음
규칙기반 언어	Marvel[11], MERLIN[12], SPELL[13], MSAP/DL[14]
그래프 기반 언어	Process Weaver, SPELL, FUNSOFT[15], SLANG[16]
객체기반 언어	E3, SPELL, MASP/DL, APPL/A
접근기반 언어	APPL/A, Adele[17], Marvel, MASP/DL

2.3 객체기반 프로세스 프로그래밍 언어

대표적인 객체기반 프로세스 프로그래밍 언어로는 E^3의 PML을 들 수 있으며 넓은 의미로 보면 EPOS의 SPELL, ALF의 MASP/DL, APPL/A 등도 객체 모형을 어느 정도 제공하고 있기 때문에 객체기반 프로세스 프로그래밍 언어에 포함시켜 함께 생각해 보기로 한다.

전형적인 객체기반언어에서 데이터와 프로세스 정보는 객체로 캡슐화된다. 각 객체는 어떠한 작업을 수행할 수 있는 행위의 집합과 그와 관계된 데이터의 집합으로 구성된다. 객체 계층 구조는 객체의 재사용을 용이하게 하여 기존의 객체에 새로운 상태정보의 첨가, 응답형태의 변경, 추가적인 행위의 첨가등으로 새로운 객체를 쉽게 생성할 수 있다.

따라서 객체기반언어는 소프트웨어 프로세스 내의 객체와 그에 관련된 행위를 모형화하는데 적합하다. 그러나 객체 자체만으로는 소프트웨어 프로세스를 자연스럽게 모형화하는데에는 한계가 있으므로 객체

기반 프로그래밍 언어는 객체로 모형화된 행위의 표현뿐만 아니라 객체사이의 관계와 행위사이의 관계 등을 표현할 수 있는 언어구조를 제공하는 것이 바람직하다.

3. 객체기반 프로세스 프로그래밍언어의 바람직한 구성요소

새로운 프로세스 프로그래밍 언어가 발전적으로 설계되기 위해서는 기존의 프로세스 프로그래밍 언어의 장단점을 분석하고 이를 바탕으로 새로운 언어가 가져야 할 유용한 특성을 도출하는 것이 바람직하다[20, 21]. 따라서 이 장에서는 객체기반 프로세스 프로그래밍 언어의 바람직한 구성요소로서 객체 및 이들사이의 관계, 그리고 프로세스의 수행제어, 선행조건 및 후행조건, 트리거를 제안하고, 이들을 기존의 객체기반 프로세스 프로그래밍 언어와 연관지어 살펴보고자 한다.

3.1 객체(object)의 지원

객체는 각각의 구성요소들을 이용하여 복잡한 시스템을 쉽게 구축할 수 있는 소프트웨어 모형과 개발원칙으로 설명될 수 있다. 따라서 소프트웨어 개발 프로세스의 각 구성요소들을 객체로 기술하는 것은 실세계를 잘 반영할 수 있으며 복잡한 소프트웨어 프로세스를 간략하고 자연스럽게 나타낼 수 있다.

APPL/A에서는 객체의 추상화와 캡슐화, 모듈화 등을 Ada 언어 자체에서 제공하는 기능을 사용하고 있으므로 APPL/A에서는 프로세스 정의 언어 차원의 객체를 Ada의 일반적 record 형으로 정의하며 Ada 패키지 차원의 추상화를 제공한다. 그리고 미리 정의된 모형 객체는 지원하고 있지 않다.

E^3에서는 모든 객체는 기본 객체인 object로부터 상속을 받는다. E^3의 미리 정의된 객체인 task, role, data, tool은 주어진 행위, 역할, 생산물, 도구에 대한 모형 객체를 나타낸다.

MASP/DL에서 객체는 PCTE의 객체 시스템인 OMS (Object Management System)의 객체 모형이며 타입을 가지는 객체 관계 다이어그램이다. 객체는 서로다른 객체 타입으로 나누어질 수 있으며 공통적인 여러 속성을 가지고 있다. 그리고 subtype은 이미 정의된 객체를 재사용할 수 있도록 한다.

SPELL에서는 프로세스 모형을 나타내는 task와 생성물인 product를 하나의 객체 개념으로 표현하고 있다. 그리고 SPELL에서 제공하는 모형 객체로는 TaskEntity와 DataEntity가 있으며 PM_Entity는 프로세스 요소의 객체이고 모형 객체인 TaskEntity와 DataEntity의 상위 객체 역할을 한다.

객체기반 프로세스 프로그래밍 언어에서는 기본적으로 객체의 개념을 지원함과 동시에 프로세스 프로그래머가 짧은 시간내에 원하는 프로세스 프로그래밍이 가능하도록 다양한 모형객체를 지원하는 것이 바람직하다고 볼 수 있다.

3.2 관계(relation)의 지원

소프트웨어 프로세스를 객체로 모형화하였을 경우, 이들 객체사이의 관계를 표현할 수 있어야 한다. 이 관계에 대한 표현은 행위 사이의 관계, 생산물 사이의 관계, 행위와 생산물 사이의 관계, 생산물과 도구 사이의 관계 등 여러 관계로 구성될 수 있다. 이러한 관계는 프로세스의 정의 및 계획의 타당성에 영향을 미칠 수 있으며 이를 위한 일련의 행위(activity)와 재반응(reactivity response)이 필요하다. 그리고 프로세스의 분석과 일관성 유지를 위해 다양한 관계가 사용될 수 있다.

APPL/A의 Relation unit은 소프트웨어 객체와 생산물 사이의 관계를 기술하며 소프트웨어 생산물과 다른 소프트웨어 프로세스 데이터간에 나타나는 일반적이고 유연한 자료구조를 제공한다. APPL/A의 관계는 Ada의 타스크와 비슷한 수행 흐름을 가지고 있어서 병행성 및 추론, 동적인 메모리의 할당과 같은 목적을 위해 자동적으로 수행될 수 있다.

E^3에서 프로세스 모형 클래스 사이에 주어진 모든 관계는 클래스 단계에서 구축된다. E^3에서 제공하는 모형 관계로는 부분 타스크로의 분해를 나타내는 subtask, 데이타의 입출력을 나타내는 input과 output, 타스크의 수행순서를 결정하는 preorder, 다른 타스크의 재수행을 나타내는 feedback, 타스크의 책임을 나타내는 responsible, 주어진 데이터의 조작도구를 명시하는 use 등이 있다.

MASP/DL에서 관계 타입은 두 링크 타입의 쌍으로 정의한다. 이 관계 타입은 관계의 이름, 목적 객체

의 타입 집합, 주어진 객체가 관계내에서 링크되어야 하는 객체의 최대값과 최소값을 표시하는 카디널, 객체 역할, 관계를 기술하는 속성 등을 제공하여 나타낸다. 이 관계는 두 객체사이의 상호 의존성이나 연관성을 표현한다. 관계의 집합은 객체 집합 사이의 관계를 모형화하는 것을 허용한다.

SPELL은 객체사이의 이진 관계를 지원한다. SPELL의 관계형의 집합은 루트에 PM_Rel을 가진 계층으로 구성되어 있다. 모든 관계는 이진 관계이고 두 개의 연결된 개체형, 즉 목적(destination) 타입과 원시(source) 타입은 역할의 속성에 표현된다. 관계의 접근은 cre_rel, read_rel, del_rel 함수에 의해 객체에 연관되어 있는 관계를 수행시킨다.

객체기반 프로세스 프로그래밍 언어에서는 객체를 기본적으로 제공하므로 객체들 사이의 연관관계를 나타낼 수 있도록 하는 것이 바람직하며, 이러한 연관관계 또한 프로세스 프로그래머가 편리하게 이용할 수 있도록 미리 정의된 모형관계를 지원하는 것이 바람직하다.

3.3 프로세스의 수행제어

현재 프로세스 프로그래밍에서 실제의 동작은 타스크나 행위에서 발생한다. 기존에 있는 대부분의 프로세스 프로그래밍언어에서는 선행조건과 후행조건, 트리거 등을 지원하여 타스크간의 수행 흐름을 수행시에 동적으로 판단할 수 있도록 되어 있다. 그러나 실제의 프로세스 프로그래밍에서는 타스크간의 선, 후 관계를 명시적으로 표현할 필요성을 느끼게 된다. 이를 위해서는 정적인 타스크간의 수행 제어를 위해 명시적으로 타스크간의 선후 관계 및 병행성을 표현해 줄 수 있어야 한다.

APPL/A에서는 자신이 행위에 대한 명세를 가지고 수행되는 것이 아니고 Ada에서 제공하는 병행 객체인 타스크에 의해 수행된다. 따라서 APPL/A에서의 수행 제어는 Ada 언어에서 제공하는 타스크 단계의 수행과 동적인 수행 제어인 트리거, 그리고 선행조건, 후행조건에 의해 수행이 제어되게 된다.

E^3의 행위는 기본적으로 객체사이의 관계를 이용하여 수행제어가 발생하게 된다. 이 관계는 타스크의 앞, 뒤에 나타나는 데이터를 명시해 줌으로써 타스크의 동기화를 유도할 수 있다. 이와 함께 E^3에서는 타스크의 수행을 외부적으로 명확히 명시해주기 위해 preorder 관계를 두어 타스크간의 수행제어를 명확히 나타낼 수 있다.

MASP/DL로 객체에 대한 행위를 정의할 때, 각 행위에 대한 수행제어는 path expression에 나타난 행위들 사이의 수행 순서를 결정한다. path expression은 여러 행위에 대한 수행을 결정해주는 표현으로써 이 path expression에 나타난 행위들은 그 나타난 순서대로 수행을 하게 된다.

SPELL에서는 객체인 타스크가 수행중에 선행조건, 후행조건에 의해 수행제어가 일어나게 된다. 즉 생성된 데이터에 의한 조건에 의해 타스크의 수행 순서가 결정되도록 되어 있다.

객체기반 프로세스 프로그래밍 언어에서는 프로세스의 수행을 제어하기 위하여 세밀한 부분은 동적으로 수행을 제어하고, 여러 부분 프로세스를 포함하고 있는 상위 프로세스는 자신의 하위 프로세스에 대하여 명시적으로 수행을 제어할 수 있도록 하는 것이 바람직하다.

3.4 선행조건 및 후행조건의 지원

선행조건은 관련된 타입의 오퍼레이션을 수행하기 전에 만족해야하는 제약조건이다. 만약 선행조건의 평가가 만족되지 않으면 그 오퍼레이션은 수행될 수 없으며 선행조건이 만족될때까지 대기하여야 한다. 후행조건은 오퍼레이션의 수행후에 만족하여야 하는 제약조건을 나타낸다. 이러한 선행조건과 후행조건을 제공함으로써 객체나 관계에 대한 연산을 수행할 때 필요한 조건을 검사할 수 있으며 이와 함께 타스크간의 동기화도 부분적으로 이룰 수 있다.

APPL/A에서 predicate 유닛은 관계상의 조건 명세를 허용하며 함수와 비슷하게 호출될 수 있으며 제약조건과 같이 강제성을 지닐 수도 있다. 이 predicate의 강제성은 APPL/A 내의 일관성을 정의하는데 사용되어 관계에 대해 일관된 상태를 정의하고 강제성을 부여하는데 사용될 수 있다. 따라서 APPL/A에서는 predicate를 이용하여 소프트웨어 프로세스와 생산물의 일관성을 명시하고 유지할 수 있다.

MASP/DL에서 operator는 자신의 이름, 선행조건, 후행조건, 인자 등으로 구성되어 있다. 선행조건은 관련된 타입의 연산을 수행하기 전에 만족해야 하는 제

약조건이다. 만약 선행조건의 평가가 만족되지 않았다면 그 연산은 수행되지 않는다. 후행 조건은 연산의 수행후에 만족해야 하는 조건이다.

SPELL에서 타스크의 수행제어는 실제적으로 수행관리자에 의해 제어받는다. 이 수행관리자는 주어진 타스크를 수행하고자 할 때의 조건을 명시하는 PRE_DYNAMIC, 주어진 타스크가 달성해야 할 작업을 나타내는 순차적인 프로그램인 CODE, 에러를 다루는 부분인 POST_DYNAMIC의 세부분을 이용하여 수행제어를 담당한다.

E^3는 선행조건이나 후행조건을 제공하고 있지 않다.

객체기반 프로세스 프로그래밍 언어에서는 어떠한 프로세스가 수행되기 위하여 만족해야 하는 선행조건과 수행된 후에 만족해야 하는 후행조건을 제공함으로써 프로세스가 수행해야 하는 조건을 검사할 수 있고 프로세스의 수행을 제어할 수 있도록 하는 것이 바람직하다.

3.5 트리거(trigger)의 지원

트리거는 객체나 관계에 연관된 오퍼레이션이 수행될 때 발생하는 이벤트(event)에 대한 재반응(reactive response)으로 정의된다. 트리거는 소프트웨어 프로세스의 병행조작을 지원할 수 있으며 동적인 프로세스의 수행을 제어할 수 있다. 뿐만 아니라 트리거는 예외상황(exception)에 대한 처리를 담당하여 다른 프로세스에게 수정에 대한 이벤트나 변화에 대한 지시, 계산의 수행 등을 전달할 수 있다. 이 트리거는 주로 객체 또는 관계에 연관된 오퍼레이션의 호출이 일어나기 전, 혹은 일어난 후에 특별한 명령을 수행한다.

APPL/A의 트리거 유닛은 관계에 대한 오퍼레이션을 호출함으로써 발생되는 이벤트를 제어하는 논리적인 수행 흐름이며 소프트웨어 프로세스내의 병행조작을 위해 사용될 수 있다.

SPELL에서의 트리거는 예외상황 처리나 변화의 전달기능을 담당하며 객체내에 정의된 오퍼레이션이 호출되기 전 또는 호출된 후에 특별한 명령을 수행하는 것이다.

MASP/DL에서는 if-then 구문을 이용하여 트리거를 제공한다. 즉 어떠한 오퍼레이션을 수행할 때의 상태에 따라 특정 작업을 수행할 수 있도록 한다.

E^3에서는 트리거를 제공하지 않는다.

객체기반 프로세스 프로그래밍 언어에서는 프로세스의 수행중에 발생하는 예외상황이나 프로세스의 수행이 종료되었을 때 발생할 수 있는 예외상황을 미리 선언할 수 있도록 하는 것이 바람직하다.

이상에서 살펴본 내용을 정리하면 〈표 2〉와 같다.

〈표 2〉 APPL/A, E^3, MASP/DL, SPELL 언어의 평가
〈Table 2〉 Evaluation of APPL/A, E^3, MASP/DL, SPELL language

O : 지원 △ : 부분적 지원 X : 지원안함

요건 / 프로세스 프로그래밍언어	객체의 지원	관계의 지원	수행제어	선후행 조건	트리거
APPL/A	△	O	O	O	O
E^3	O	O	O	X	X
MASP/DL	△	O	O	O	O
SPELL	O	△	O	O	O

이를 바탕으로 효과적이고 자연스러운 프로세스 프로그래밍을 지원하기 위해 프로세스 프로그래밍언어가 갖추어야 할 특성은 다음과 같이 기술할 수 있다.

먼저 행위, 생산물, 리소스, 역할 등과 같이 소프트웨어 프로세스를 구성하는 객체를 지원하여야 한다. 이와 아울러 가장 기본이 되고 사용자가 쉽게 상속받아서 이용할 수 있는 모형객체와 모형관계를 지원하는 것이 바람직하다. 또한 SPELL과 같이 수행제어가 내부 조건의 변화를 통한 동적인 수행제어만을 지원하기보다는 적절한 규모의 수행단위는 외부적인 수행제어를 명시하여 실제 소프트웨어 프로세스가 진행되는 과정을 전체적으로 파악할 수 있도록 지원하는 것이 바람직하다. 그리고 객체나 관계의 일관성 유지를 원천적으로 달성하기 위해서는 선행조건 및 후행조건을 지원하는 것이 필요할 것이다. 트리거는 예외상황이 발생했을 때 이를 알려주는 방법으로서 유용하게 사용이 되며, 아울러 큰 단위의 수행제어가 아니라 사건중심(event-driven) 수행제어를 제공하기 위한 동적수행제어를 위해 필요하다. 경우에 따라서는 예외상황과 동적수행제어를 언어 구문적으로 구별하여 나타내는 것도 바람직하다.

4. SimFlex 언어의 구조

본 장에서는 앞에서 분석, 제시된 객체기반 프로세스 프로그래밍 언어가 가져야 할 바람직한 구성요소에 입각하여 설계된 SimFlex 언어의 특성에 대하여 살펴보고 언어구조 및 실제 사용 예를 보인다.

4.1 SimFlex 언어의 특성 및 구성요소

SimFlex는 기존의 프로세스 프로그래밍 언어가 가지고 있는 구조의 복잡성과 특정 프로세스 중심 소프트웨어 개발환경에 한정된 언어로서의 단점을 보완하는데 그 기본 목적을 두고 설계되었다. 즉 언어의 문법과 의미를 간결하게 표현할 수 있으며 소프트웨어 개발 프로세스 모형의 계층적 조합을 제공하여 복잡한 프로세스도 계층적으로 조명이 가능하도록 한다. 뿐만 아니라 적절한 적합화를 통하여 특정 프로세스 중심 소프트웨어 개발환경에 포함될 수 있는 기반 프로세스 프로그래밍 언어로 사용될 수 있도록 설계되어 있다.

3장에서 제시된 객체기반 소프트웨어 프로세스가 가지고 있어야 하는 바람직한 특성에 따라 설계된 SimFlex의 구성요소는 〈표 3〉과 같다.

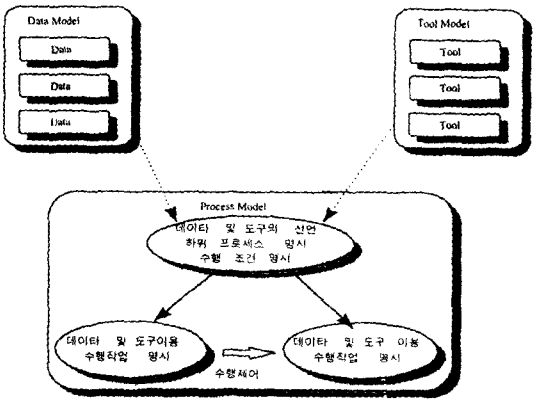
〈표 3〉 SimFlex 언어의 구성요소
〈Table 3〉 Components of SimFlex Language

프로세스	각 소프트웨어 프로세스 모형을 정의한다.
전역생성물 및 이벤트의 선언	모든 프로세스가 이용하는 생성물 및 이벤트를 선언한다. 특히 전역생성물은 SimFlex에서 객체로 정의되며 다른 생성물로부터 상속받을 수 있다.
프로세스의 사용형태 선언	instantiation 도구에 의해 호출될 프로세스의 사용 형태를 선언하며 이 프로세스가 사용하는 생성물을 선언한다.
내부생성물의 선언	자신 및 하위 프로세스에서 사용할 생성물을 선언한다. 전역생성물과 마찬가지로 객체로 정의되며 다른 생성물로부터 상속받아서 선언될 수 있다.
선행/후행조건의 선언	각 프로세스의 오퍼레이션이 만족해야 하는 조건을 명시한다.
예외상황의 선언	프로세스가 수행할 時나 수행한 후에 만족해야 하는 생성물의 상태를 나타내며 생성물의 상태는 AND, OR 형태로 조합될 수 있다.
관계의 선언	해당 오퍼레이션을 수행할 때 발생할 수 있는 예외상황을 선언한다.
하위프로세스의 선언	프로세스 모형내에 기술된 객체들의 연관관계를 선언한다. 이 관계는 두 생성물사이에 어떠한 연관관계가 성립되어야 하는지를 나타낸다.
타스크의 선언	한 프로세스를 구성하는 하위 프로세스들을 선언한다. 하위프로세스들은 자신을 선언한 상위프로세스의 수행제어를 받으며 상위프로세스의 생성물을 공유할 수 있다.
	실제 프로세스가 수행해야하는 작업을 명시한다. 프로세스가 수행해야 하는 작업을 순차수행(→), 병행수행(‖), 조건부수행(if), 반복수행(while)등의 구문으로 표현할 수 있다.

〈표 3〉에서 보는 바와 마찬가지로 SimFlex 언어는 소프트웨어 프로세스 모형을 기술하기 위한 프로세스를 정의할 수 있으며, 또한 해당 프로세스의 수행을 제어할 수 있는 선행조건 및 후행조건을 명시할 수 있도록 되어 있다. 예외상황이 발생했을 때는 이를 알려줄 수 있는 예외상황을 발생시킬 수 있으며 프로세스의 생성물간의 연관관계를 표현할 수 있도록 한다. 프로세스의 계층적인 조합으로 하위 프로세스를 선언할 수 있으며 이들 하위 프로세스간의 수행제어를 위하여 타스크 부분에서 하위 프로세스 간의 수행순서를 결정할 수 있도록 한다. 이러한 SimFlex 언어의 특성을 도식적으로 나타내면 (그림 1)과 같다.

(그림 1)에서 보는 바와 같이 SimFlex는 프로세스 모형, 데이터 모형, 도구 모형을 객체로서 표현할 수 있다. 데이터 모형에서는 소프트웨어 프로세스내에서 사용하는 생성물들을 표현하도록 하며 도구 모형에서는 이들 생성물들을 유도해내기 위해 필요한 도구들을 명시한다. 프로세스 모형에서는 이들 생성물과 도구를 이용하여 실제로 작업을 수행하게 된다. 프로세스 모형 내에서는 자신이 사용하는 데이터나

도구들을 선언할 수 있으며 자신이 선언한 데이터나 도구는 하위 프로세스에서도 사용할 수 있다. 그리고 상위 프로세스는 하위 프로세스들의 수행을 선행조건과 후행조건, 명시적인 수행제어를 통하여 제어하게 되며 단말 노드의 프로세스는 실제 프로세스가 담당하고 있는 작업을 수행하게 된다.



(그림 1) SimFlex의 구조
(Fig. 1) SimFlex Structure

4.2 SimFlex 언어의 구조

SimFlex 프로세스 프로그래밍 언어의 구조는 소프트웨어 프로세스가 기본적으로 가지고 있어야 하는 속성을 파악하고, 이를 바탕으로 기존의 프로세스 프로그래밍 언어에서 제시하고 있는 특성을 분석한 결과를 바탕으로 설계하였다. 구문과 소프트웨어 프로세스의 예제에 대한 좀더 상세한 기술은 [19]에 잘 나타나 있다.

프로세스

프로세스는 각 소프트웨어 프로세스 모형을 정의한다. 프로세스를 정의하기 위한 구문은 다음과 같다.

⟨*process-declaration*⟩ ::= Process [resuable] ⟨process-name⟩ is
⟨*usage-declaration*⟩
⟨*product-declaration*⟩
[⟨*precondition-declaration*⟩]
[⟨*postcondition-declaration*⟩]
[⟨*exception-declaration*⟩]
[⟨*relation-declaration*⟩]
[⟨*use-declaration*⟩]
[⟨*task-declaration*⟩]
end ⟨*process-name*⟩;

SimFlex로 기술된 프로세스 모형에서는 소프트웨어 프로세스를 기술하데 필요한 구성요소를 표현하도록 한다. 특히 *use-declaration* 절을 이용하여 한 프로세스를 여러 하위 프로세스로 나눌 수 있다. 이러한 프로세스의 계층적인 조합은 한 프로세스를 기술할 때 프로세스를 계층적으로 조합하여 프로세스를 선언할 수 있음을 보여준다. 이와 함께 이미 작성된 프로세스 모형의 재사용을 위하여 reusable 속성을 부여할 수도 있다. 이러한 프로세스 모형의 재사용성은 상위 프로세스의 입력과 출력을 제외한 모든 생성물이 내부적으로 선언되어 있어서 상위 프로세스와 하위 프로세스가 다른 프로세스 모형의 하위 프로세스로 사용될 수 있음을 나타낸다.

프로세스를 구성하는 각 요소를 살펴보면 다음과 같다.

프로세스의 사용형태 선언

프로세스의 사용형태 선언은 프로세스가 instantiation 도구에 의하여 호출될 형태를 선언한다. 이 부분에는 해당 프로세스가 외부로부터 입력받거나 출력으로 내보내야 할 생성물을 인자로 지정한다. 즉 프로세스 사용형태의 선언에서 인자는 프로세스의 수행동안에 이용하는 소프트웨어 생성물을 나타낸다. 프로세스 사용형태를 정의하기 위한 구문은 다음과 같다.

⟨*usage-declaration*⟩ ::= usage ⟨*process-name*⟩ (⟨*argument-list*⟩);
⟨*argument-list*⟩ ::= ⟨*argument-param*⟩ [, ⟨*argument-list*⟩]
⟨*argument-param*⟩ ::= in ⟨*product-name*⟩ | out ⟨*product-name*⟩
| inout ⟨*product-name*⟩

위에서 process-name은 항상 선언된 프로세스의 이름과 동일해야 하며 각 생성물이 입력으로 주어지는지, 출력으로 주어지는지를 인자로 가지게 된다.

생성물의 선언

SimFlex에서는 프로세스 자신이 사용하는 생성물뿐만 아니라 자신의 하위 프로세스에서 사용하는 생

성물을 현재 프로세스에서 선언하도록 한다. 이는 프로세스의 계층구조로 살펴볼 때 바로 상위의 프로세스가 하위 프로세스에 대한 정보를 모두 선언하도록 함으로서 상위 프로세스가 하위 프로세스에 대한 정보를 쉽게 참조할 수 있도록 한다. 이 생성물을 정의하기 위한 구문은 다음과 같다.

⟨*product-declaration*⟩ ::= **product** ⟨*product-list*⟩
⟨*product-list*⟩ ::= ⟨*product-name*⟩ is {⟨*product-status*⟩;
[⟨*product-list*⟩]

각 생성물에 대하여 생성물이 가질 수 있는 상태를 선언한다. 이 상태는 프로세스가 진행되는 동안 생성물이 어떠한 상태로 변화되는 지를 나타내는데 사용된다.

선행조건의 선언

선행조건은 관련된 프로세스가 수행되기 전에 만족해야 하는 조건이다. 선행조건에 나타난 생성물의 상태를 검사하여 그 상태가 만족된다면 해당하는 프로세스가 수행될 수 있다. 이 선행조건을 정의하기 위한 구문은 다음과 같다.

⟨*precondition-declaration*⟩ ::= **precondition** ⟨*condition-list*⟩
⟨*condition-list*⟩ ::= ⟨*product-name*⟩ is ⟨*product-status*⟩;
[**AND** | **OR**] [⟨*condition-list*⟩]

선행조건은 AND나 OR 조건으로 조합되어 프로세스의 수행 여부를 파악할 수 있다.

후행조건의 선언

후행조건은 관련된 프로세스가 종료될 때 만족해야 하는 조건이다. 후행조건에서는 현재의 생성물의 상태를 비교하여 그 상태가 만족된다면 현 프로세스가 제대로 수행되었음을 파악할 수 있고 이는 다른 프로세스가 수행되기 위한 선행조건으로 사용된다. 후행조건을 위한 구문은 다음과 같다.

⟨*postcondition-declaration*⟩ ::= postcondition ⟨*condition-list*⟩
⟨*condition-list*⟩ ::= ⟨*product-name*⟩ is ⟨*product-status*⟩
;[**AND**|**OR**] [⟨*condition-list*⟩]

선행조건과 마찬가지로 후행조건도 AND나 OR 조건으로 조합될 수 있다.

예외상황의 선언

예외상황은 객체나 관계에 연관된 오퍼레이션을 수행할 때 발생할 수 있는 이벤트들을 나타낸다. SimFlex에서 예외상황은 정상적인 오퍼레이션보다는 비정상적인 오퍼레이션이 수행될 때 해당 이벤트가 발생하도록 한다. 따라서 예외상황은 주로 후행조건이 만족되지 않았을 때나 시스템으로 부터 특정한 이벤트가 발생했을 때 전달되도록 한다.

⟨*exception-declaration*⟩ ::= **exception** ⟨*exception-list*⟩
⟨*exception-list*⟩ ::= ⟨*event-name*⟩ **when** ⟨*product-name*⟩
is [**not**] ⟨*product-status*⟩;
[⟨*exception-list*⟩]

만약 어떠한 생성물에 대하여 어떠한 상태가 만족되지 않거나 또는 어떠한 상태가 되었을 때 해당하는 이벤트를 발생시키도록 한다.

관계의 선언

SimFlex는 생성물사이의 관계를 지원하고 있다. 이 생성물 사이의 관계는 두 생성물 사이의 연관관계를 부여하여 생성물간의 선후관계, 의존관계등을 명시하게 된다. 또한 한 프로세스 모형내에서 선언된 관계는 자신의 하위 프로세스 모형에서도 이용할 수 있도록 한다. 이 관계를 정의하기 위한 구문은 다음과 같다.

⟨*relation-declaration*⟩ ::= relatioin ⟨*relation-list*⟩
⟨*relation-list*⟩ ::= ⟨*relation-name*⟩ is source : ⟨*product-name*⟩;
destination : ⟨**product-name**⟩;
end ⟨*relation-name*⟩
[⟨*relation-list*⟩]

관계의 선언은 source와 destination에 관련된 생성물의 이름을 지정함으로써 설정된다.

하위 프로세스의 선언

한 프로세스가 다시 여러 하위 프로세스로 이루어

에는 하위 프로세스를 선언하고, 해당 하위 프로세스들에 대한 수행제어를 할 필요가 있다. 하위 프로세스는 상위 프로세스의 자원을 공유하며 상위 프로세스의 작업을 더욱 세부적으로 나누는 역할을 한다. 이 하위 프로세스의 선언에 대한 구문은 다음과 같다.

⟨*use-declaration*⟩ ::= use ⟨*process-list*⟩
⟨*process-list*⟩ ::= ⟨*process-name*⟩ (⟨*argument-list*⟩);[⟨*process-list*⟩]

타스크의 선언

프로세스의 타스크 선언에는 실제로 해당 프로세스가 수행해야 할 작업을 명시해 준다. 가장 하위 프로세스가 아닌 경우라면 타스크에서 하위 프로세스에 대한 수행제어를 담당하고, 하위 프로세스들이 모두 종료되었을 때 모든 작업이 제대로 진행되었는가를 검사해 볼 수 있도록 한다. 하위 프로세스의 수행제어를 위해서 순차적인 수행, 병행 수행, 반복수행을 지원한다. 만약 가장 하위 프로세스라면 실제의 사용자와 연관되어 작업에 대한 지원을 해주게 된다.

⟨*task-declaration*⟩ ::= task ⟨*task-statement-list*⟩
⟨*task-statement-list*⟩ ::= ⟨*simple-statement*⟩;
[⟨*task-statement-list*⟩]
|⟨*sequential-statement*⟩;[⟨*task-statement-list*⟩]
|⟨*parallel-statement*⟩;[⟨*task-statement-list*⟩]
|⟨*if-statement*⟩;[⟨*task-statement-list*⟩]
|⟨*iteration-statement*⟩;[⟨*task-statement-list*⟩]
⟨*simple-statement*⟩ ::= ⟨*process-name*⟩;
⟨*sequential-statement*⟩ ::= ⟨*simple-statement*⟩
[→⟨*sequential-statement*⟩]
⟨*parallel-statement*⟩ ::= ⟨*simplement-statement*⟩
[‖⟨*parallel-statement*⟩]
⟨*if-statement*⟩ ::= if ⟨*expression*⟩ then
⟨*task-statement-list*⟩ endif;
⟨*iteration-statement*⟩ ::= while ⟨*expression*⟩ do
⟨*task-statement-list*⟩ end;

4.3 ISPW-6 프로세스 모형의 기술

많은 연구자들의 흥미를 유발시키고 있는 소프트웨어 프로세스 모형화에 대한 접근법을 비교하고 대조시켜보고자 하는 목적으로 ISPW-6라고 일컬어지는 일반적인 문제가 정의되었다. 본 절에서는 설계된 SimFlex를 이용하여 ISPW-6 문제의 한 부분인 TEST 프로세스를 기술하고자 한다. ISPW-6에서의 TEST 프로세스는 수정된 코드를 위한 유닛 테스트 패키지의 응용 프로그램과 결과의 분석 과정을 포함한다. 이 단계는 설계자와 품질보증 공학자가 참여한다. 모든 테스트 패키지는 어떠한 분석이나 추가적인 행위가 일어나기 전에 수행되어야 한다. 만약 소스코드가 수정되어야 한다면 유닛 테스트가 추가적으로 수정되어야 한다. 지적된 수정이 완료되면 유닛 테스트가 재시작된다. 만약 유닛이 테스트를 통과하고 적당한 수준을 만족한다면 이 프로세스는 완료된다. 이때 프로젝트 관리자는 유닛 코드의 가장 최신 버전이 통합 테스트를 위해 사용될 수 있다는 메시지를 받게 된다. 이러한 과정을 SimFlex로 표현하면 다음 (그림 2)와 같다.

```
process test is

usage
  test(in object_code, in unit_test_pkg, out test_result, out fb_edit);

product
  test_plan is (created, modified);
  ut_package is (created, modified);
  fb_unitpack is (normal, feedback);

precondition
  object_code is compiled;
  unit_test_pkg is created;

postcondition
  test_result is created;
  fb_edit is (normal | feedback);

exception
  INVALID_RESULT when test_result is not created;

relation
  none;

use
  modify_testplan(in out test_plan);
  modify_testpack(in test_plan, in out test_ package, in fb_unitpack);
  test_unit(in ut_package, in object_code, in test_result, out family,
          out fb_edit, out fb_unitpack);

task
  modify_testplan -> modify_testpack -> test_unit;

  while (fb_unitpack == feedback) do
    modify_testpack -> test_unit;
  end;

end test;
```

(그림 2) TEST 프로세서의 기술
(Fig. 2) Description of TEST Process

이 예제는 ISPW-6 소프트웨어 프로세스중 테스트 프로세스를 SimFlex로 기술한 것이다. 테스트 프로세스는 object_code와 unit_test_pkg를 입력으로 받고 test_result와 fb_edit을 출력으로 내어 보낸다. 이때 fb edit은 테스트 과정상에서 문제가 발생했을 때, 이전 프로세스로의 피드백을 위한 생성물이다.

테스트 프로세스 내부에서 이용하는 생성물로는 test_plan, ut_package, fb_unitpack등이 있으며 이들 각각은 created와 modified, created와 modified, 그리고 normal, feedback 상태를 가진다. 테스트 프로세스가 수행하기 위한 선행조건으로는 object_code가 compiled 상태이어야 하며 unit_test_pkg가 created 상태가 되어 있어야 한다. 테스트 프로세스가 종료되었을 때는 test_result가 created 상태가 되어야 하며 fb_edit이 normal이나 feedback 상태가 되어야 한다. 만약 이 프로세스가 종료될 때 test_result가 created 상태가 아니라면 INVALID_RESULT라는 예외상황을 발생시키도록 한다.

이 테스트 프로세스는 modify_testplan, modify_testpack, test_unit이라는 세 개의 하위 프로세스로 구성이 되며 이들은 각각 차례대로 수행이 된다. 만약 test_unit까지 수행이 되었을 때 fb_unitpack이 feedback 상태가 되었다면 피드백된 사항을 처리하기 위하여 modify_testpack과 test_unit을 반복수행하도록 한다.

5. 결 론

소프트웨어 프로세스는 사용자의 요구사항을 수행할 수 있는 소프트웨어로 실현시키고자 할때 필요한 소프트웨어의 생산활동이나 유지보수등과 같은 복잡한 소프트웨어 개발과정이다. 본 논문에서는 소프트웨어 프로세스를 프로그래밍 언어의 형태로 기술할 수 있는 프로세스 프로그래밍 언어인 SimFlex 언어의 구조에 대하여 기술하였다. SimFlex 언어를 발전적으로 설계하기 위하여 기존에 나타나 있는 프로세스 프로그래밍 언어를 분석하여 기존 객체기반 프로세스 프로그래밍 언어의 장단점을 파악하였으며, 이를 바탕으로 SimFlex 언어가 가져야 하는 바람직한 특성을 다음과 같이 도출하였다.

첫째, 복잡한 소프트웨어 개발 프로세스를 자연스럽고 효율적으로 모형화하기 위하여 객체의 개념을 지원한다. 둘째, 프로세스 요소간의 연관관계를 유지하기 위하여 객체간의 관계를 지원한다. 셋째, 목적 소프트웨어 개발 프로세스의 주요한 수행제어를 사용자가 손쉽게 이해할 수 있도록 프로세스의 수행제어를 명시적으로 기술하도록 한다. 넷째, 소프트웨어 개발 프로세스의 수행이 신뢰성을 가지도록 선행조건과 후행조건을 제공한다. 다섯째, 예외상황을 다루기 위하여 트리거를 제공한다. 여섯째, 잘 정의된 프로세스 모형을 다른 프로세스 모형의 하위 프로세스로 재사용할 수 있도록 프로세스 모형의 계층적 조합을 제공한다.

새롭게 설계된 SimFlex는 문법과 의미가 간결하며 위의 바람직한 특성들을 자연스럽게 표현할 수 있도록 설계되었다. 그리고 적절한 적합화를 통하여 특정 프로세스 중심 소프트웨어 개발환경에 포함될 수 있는 기반 프로세스 정의언어로 사용될 수 있도록 설계되었다. 이러한 적합화는 SimFlex 프로세스 프로그래밍 언어가 프로세스 설계 도구나 모니터링 도구등과 같은 특정 환경과 독립적으로 설계되었으므로 외부적인 인터페이스만을 고려하면 쉽게 다른 환경으로 이식될 수 있다.

현재, Windows NT/95 운영체제하에서 C++ 언어를 이용하여 SimFlex 프로세스 프로그래밍 언어의 번역기와 번역된 프로세스 모형을 수행시킬 수 있는 프로세스 수행 관리자를 개발하고 있다. SimFlex 번역기와 프로세스 수행 관리자는 ETRI에서 개발중인 정보통신 응용 서비스 개발 플랫폼에서 프로세스 모형화 개발도구 및 소프트웨어 프로세스 모니터링 도구와 연계하여 동작될 예정이다.

참 고 문 헌

[1] Stanley M. Sutton, Jr., Peri L. Tarr, An Analysis of Process Languages, CMPSCI TR 95-78, August 1995.

[2] Anthony Finkelstein, Jeff Kramer and Bashar Nuseibeh, Software Process Modelling and Technology, Research Studies Press Ltd. 1994.

[3] Stanley M. Sutton, Jr., Dennis Heimbigner, and Leon J. Osterweil, "APPL/A:A Language for Software Process Programming", ACM Trans-

action on Software Engineering and Methodology, 4(3), pp. 221-286, 1995.

[4] Reidar Conradi and Chunnian Liu, Process Modelling Languages: One or Many?, In Wilhelm Schafer, editor, *Proceedings of the 4th European Workshop on Software Process Technology*, pp. 98-118, Springer-Verlag, 1995.

[5] Mario Baldi, Silvano Gai, Maria Letizia Jaccheri, and Patricia Lage, Obejct Oriented Software Process Model Design in E^3, November 1993.

[6] Reidar Conradi, Chunnian Liu, and Per H. Westby, EPOS PM: Planning and execuion, In *Proceedings of the 6th International Software Process Workshop*, 1990.

[7] Conradi, R., Liu C., and Jaccheri, M. L. Process modeling paradigms: An evaluation., In Proceedings *of the 7th International Software Process Workshop*(Yountville, Calif.), IEEE, New York, 1991.

[8] Ian Robertson, An Implementation of the ISPW-6 Process Example, in [7], pp. 187-206, 1994.

[9] P. Armenise, S. Bandinelli, C. Ghezzi, and A. Mozenti, Software Processes Representation languages: Survey and Assessment, In Proceedings of the Fourth *International Conference on Software Engineering and Knowledge Engineering, Capri(Italy)*, pp. 455-462, June 1992.

[10] Christer Fernstrom, Process Weaver: Adding process support to UNIX, In *Proceedings of the Second International Conference on the Software Process*, pp. 12-26, 1993.

[11] Gail E.Kaiser, Naser S.Barghouti, and Michael H.Sokolsky, Experience with Process Modeling in the MARVEL Software Development Environment Kernel, In *Bruce Shriver, editor, 23rd Annual Hawaii Internaltional Conference on System Sciences*, pp. 131-140, Kona HI, January 1990.

[12] B.Peuschel, W.Schaefer, and S.Wolf, A Knowledge-Based Software Deveopment Environment (on MERLIN), *International Journal of Software Engineering and Knowledge Engineering*, pp. 79-106, March 1992.

[13] Reidar Conradi, M. Letizia jaccheri, Cristna Mazzi, Minh Ngoc Nguyen, Amund Arsten, Design, Use and Implementation of SPELL, a language for Software Process Modeling and Evolution, *2nd European Workshop on Software Process Technology(EWSPT'92)*, Trondheim, Norway, 7-8 Sepetember 1992.

[14] Ph.Boveroux, G.Canals, C.Godart, Ph.Jamart and J.Lonchamp, Software Process Modelling in th ALF System: an example, In *Proceedings of the 1st European Workshop on Software Process Modelling*, Milan, May 1991.

[15] Volker Gruhn and Rudiger Jegelka, An Evalution of FUNSOFT Nets, In [18], pp. 196-214, 1992.

[16] Sergio Bandinelli and Alfonso Fuggetta, Computational Reflection in Software Process Modelling: the SLANG Approch, In *Proc. of 15th International IEEE-CS Conference on Software Engineering, Baltimore, MA*, pp. 144-154, IEEE-CS Press, May 1993.

[17] N. Belkhatir, J. Estublier, and W. L. Melo, Adele2: A Support to Large Software Development Process, In *Proc. 1st Conference on Software Proces(ICSPI), Redondo Beach, CA*, pp. 159-170, October 1991.

[18] Jean-Claude Derniame, editor, *Proc. Second European Workshop on Software Process Technology (EWSPT'92), Trondheim, Norway*, Springer Verlag LNCS 635, September 1992.

[19] 이명준, 소프트웨어 개발 프로세스 정의언어에 관한 연구, 최종보고서, 한국전자통신연구소, 1996.

[20] 김영곤, 정혜영, 이명준, 이권일, 인소란, 소프트웨어 개발 프로세스를 위한 프로그래밍 언어 구성요소, 한국정보과학회 추계학술발표논문집, 1996.

[21] 김영곤, 정혜영, 이명준, 이권일, 인소란, 객체기반 소프트웨어 프로세스 프로그래밍을 위한 SimFlex 언어의 구조, 한국정보과학회 충청지부

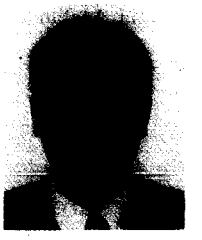
추계학술발표논문집, 1996.

김 영 곤

1994년 2월 울산대학교 전자계산학과 공학사
1996년 2월 울산대학교 전자계산학과 공학석사
1996년 3월~현재 울산대학교 전자계산학과 박사과정
관심분야: 프로그래밍 언어시스템, 병행 프로그래밍, 소프트웨어공학

강 병 도

1986년 2월 서울대학교 계산통계학과 졸업(전산과학 전공)
1988년 2월 서울대학교 대학원 계산통계학과 졸업(전산과학 전공, 이학석사)
1995년 2월 서울대학교 이학박사(소프트웨어공학전공)
1988년6월~현재 한국전자통신연구원 근무중
주요관심분야: 소프트웨어공학환경, 소프트웨어 개발방법론, 소프트웨어 프로세스 모델



이 명 준

1980년 서울대학교 수학과 졸업(학사)
1982년 2월 한국과학기술원 전산학과 졸업(석사)
1991년 8월 한국과학기술원 전산학과 졸업(박사)
1982년 3월~현재 울산대학교 전자계산학과 근무(현재 교수)
1993년 8월~1994년 7월 미국 버지니아대학 교환교수
관심분야: 프로그래밍언어, 분산객체 프로그래밍 시스템, 병행 실시간 컴퓨팅, 인터넷 프로그래밍시스템 등.