RAiSE : 다양한 의미론과 사용의 용이성을 제공하는 그래픽 프로세스 모델링 언어

이 형 원†

요 약

프로세스 모델링 언어 설계에 있어서의 핵심적인 요구사항은 풍부한 기술적 의미론의 제공과 사용의 용이성간의 적절한 균형이다. 기존 프로세스 모델링 언어들은 프로세스 언어가 갖추어야 할 이러한 두 가지 측면을 동시에 만족시키지 못함으로써 그 동안의 다양한 연구에도 불구하고 실제 소프트웨어 산업계에서 널리 사용되지 못하고 있다.

본 논문에서는 이러한 단점을 제거한 프로세스 모델링 언어 RAiSE를 정의하고 이를 잘 알려진 벤치마크 프로세스인 ISPW-6 소프트웨어 프로세스에 적용한 결과를 설명한다. RAiSE는 쉽게 이해할 수 있는 그래픽 표기법을 기반으로 하여 다양한 모델링 패러다임들의 필수적인 요소들을 결합시킴으로써 이해하기 쉬우면서도 엄격하고 풍부한 의미론을 제공한다. RAiSE에 의해 작성된 프로세스 모델은 규칙 기반 전문가 시스템 도구인 CLiPS로 구현한 프로세스 엔진에 의해 해석되고 실행된다.

키워드 : 소프트웨어 프로세스, 프로세스 모델링, 프로세스 모델링 언어

RAiSE : A Graphical Process Modeling Language Providing Semantic Richness and Ease of Use

Hyungwon Lee†

ABSTRACT

A key issue for process language design is balancing the need for semantic richness with the need for ease of use. Most process modeling languages fail to satisfy above two conflicting aspects, which is an impediment to the widespread adoption of process modeling languages in the software industry despite of a variety of software process language studies.

This paper describes a process modeling language RAiSE attempting to resolve such problem and presents the result of applying RAiSE to a well-known benchmark process, ISPW-6 software process example. RAiSE provides rigorous, yet clear semantics through combing essential features in various modeling paradigms and defining them in a well-structured graphical notation. Process models represented in RAiSE are interpreted and enacted by process engine implemented using CLiPS, a rule based expert system tool.

Key Words : Software Process, Process Modeling, Process Modeling Language

1. 서 론

소프트웨어 공학 분야에 있어서의 핵심 목표는 소프트웨어 개발 과정의 비용 절감과 생성된 소프트웨어의 품질 향상에 있으며 소프트웨어 프로세스는 이 두 가지 목표에 중요한 역할을 담당한다. 프로세스 모델링 언어는 소프트웨어 프로세스 기술에 관한 관심이 모아지기 시작한 초기부터 지금까지 중요하게 강조되어온 연구 분야이지만 기존 프로세스 모델링 언어들은 그 동안의 다양한 연구에도 불구하고 실제 소프트웨어 산업계에서 널리 사용되지 못하고 있다. 그 이유는 프로세스 모델링 언어의 설계에 있어 크게 두 가지 상호 충돌되는 접근 방법이 균형을 이루지 못해왔기 때문이다[1, 2]. 먼저, 프로세스는 조직, 행위, 산출물, 자원, 사건, 작업자, 예외 등의 다양한 의미론을 이용하여 기술될 수 있으며 또한, 그렇게 해야 한다는 주장에 입각해서 개발된 프로세스 모델링 언어들이 있다. 이러한 언어들은 강력하고 의미론적으로 매우 풍부하지만 텍스트 기반의 이해하기 어려운 형태로 표현해야 하며 매우 복잡하기 때문에 특히, 프로그래머가 아닌 사람들은 사용하기 어렵다. 반대로, 단순화에 초점을 맞춰 개발된 프로세스 모델링 언어들이 있는데 대개의 경우 의미론적 깊이와 너비를 제한하게 되며 그래픽 형태의 표현 방법을 사용한다. 이 언어들은 언어적 간결함을 강조하는 대신 실질적인 사용에는 제한을 받을 수밖에 없다. 즉, 대다수의 PSEE(Process-centered Software Engi-

† 정 회 원 : 강릉대학교 정보전자공학부 컴퓨터공학전공 교수
논문접수 : 2005년 9월 14일, 심사완료 : 2005년 11월 17일

neering Environment)에서 제공하는 프로세스 모델링 언어들은 엄격한 의미론과 사용의 용이성이라는 프로세스 언어가 갖추어야 할 중요한 두 가지 측면을 동시에 만족시키지 못한다. 효율적인 프로세스 실행과 분석을 위해서는 풍부하면서도 엄격한 의미론을 가져야 하며, 프로세스 모델링을 담당하는 대부분의 사람들이 프로그래밍 전문가가 아니며 프로세스 모델을 사용하는 기본 목적이 프로세스 이해라는 점에서는 쉽게 정의하고 이해할 수 있는 프로세스 모델링 언어가 제공되어야 한다.

본 논문에서는 상호 보완 관계에 있는 다양한 프로세스 모델링 패러다임을 하나의 모델 내에서 비전문가라도 쉽게 이해하고 정의할 수 있도록 표현해주는 그래픽 프로세스 모델링 언어인 RAiSE에 대해 기술한다. RAiSE는 상태 기반의 행위 중심 모델링 패러다임을 근간으로 하여 주요 프로세스 모델링 패러다임의 핵심 개념들을 반영하는 그래픽 표기법을 제공한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 프로세스 모델링 언어에 관한 연구 결과들을 기술하고 3장에서는 RAiSE의 표기법과 의미론을 정의한다. 4장에서는 RAiSE를 대표적인 벤치마크 프로세스인 ISPW-6 프로세스[3]에 적용한 결과를 제시하고. 5장에서는 RAiSE의 실행 환경인 PRAiSE에 대해 기술한다. 6장에서는 다른 프로세스 모델링 언어와의 차별성을 비교 분석하고 마지막으로 7장에서 결론 및 향후 연구 과제에 대해 논한다.

2. 관련 연구

소프트웨어 프로세스를 표현하고 실행시키기 위해서 프로세스 모델링 언어가 갖추어야 할 핵심 요소들은 행위, 산출물, 자원, 개체 간 관계성 등이다[4, 5]. 행위(activity)는 프로세스를 구성하는 스텝들을 정의한 것으로 어떤 프로세스 모델이라도 행위가 그 중심에 놓여지게 되며 대부분의 프로세스 모델링 언어들은 행위 또는 프로세스 자체가 분할될 수 있도록 계층적 분해를 지원한다. 산출물(artifact)은 행위의 입력과 출력을 정의한 것으로 산출물 모델링은 다양한 크기와 복합적인 구성 등의 이유로 전형적인 소프트웨어 개발에서 요구하는 데이터 모델링보다 더 복잡하다. 자원(resource)은 행위를 수행하는데 필요한 인적 물적 자원을 정의한 것으로 소모성 자원이나 공유성 자원까지 포함한다. 개체 간 관계는 행위, 산출물, 자원 간의 다양한 의미론적 상관관계를 나타내기 위한 것으로 예를 들어 행위 간의 선행 관계, 산출물 간의 생성 관계, 행위/자원/산출물간의 연관 관계 등을 표현한다.

본 장에서는 기존 프로세스 모델링 언어를 패러다임과 표현 형태에 기반을 두어 분석한 결과를 설명하고 최근 동향에 대해서 기술한다.

2.1 프로세스 모델링 패러다임

프로세스 모델링 언어는 프로세스의 다양한 측면을 표현하는데 유용한 구조들을 얼마나 지원하는가에 따라 평가될 수 있다. 소프트웨어를 명세하고 프로그래밍하는 데 사용되는 언어들 역시 프로그램이 수행되는 프로세스를 표현하고 실행시키는 수단이라는 점에 착안하여 대부분의 프로세스를 연구하는 사람들은 소프트웨어 프로세스를 모델링하는 데 있어 이러한 언어들에 기초하여 프로세스 모델링 언어들을 제안하였다.

프로세스 모델을 표현하기 위한 수단인 프로세스 모델링 언어는 프로세스 모델링을 수행하는 패러다임을 반영한다. 즉, 해당 프로세스 모델 및 실행 환경의 기본적인 개념과 원리를 가장 잘 표현하기 위한 방법을 이용하게 된다. <표 1>에는 프로세스 모델링 언어를 프로세스 모델링 패러다임에 따라 분류한 결과를 요약하였는데[4, 6-9] 각각의 패러다임은 나름대로의 장단점을 갖고 있기 때문에 한 가지 패러다임만을 이용하는 것은 일관성 측면에서는 장점이 되지만 프로세스 모델링에 필요한 모든 요구 사항을 만족시킬 수 없다.

〈표 1〉 프로세스 모델링 패러다임

패러다임	프로세스 모델링 언어/PSEE
비실행	ETVX, IDEF0, BPMN
상태 기반	Statemate, SLANG, Process Weaver
규칙 기반	MARVEL, GRAPPLE, Adele, Merlin, OIKOS, EPOS, ELF
프로그래밍	APPL/A, Little-JIL, PML, BPML
정형적 언어	Hakinowa, HFSP
구조적 기법	Statemate, IDEF0
데이터 모델링	AD model, Statemate
객체 지향	EPOS, MARVEL, MVP, OBJ, SPEM, UML 확장 언어들
정량적 모델링	System Dynamics

2.2 표현 형태

프로세스 모델링 언어는 개발 공정을 융통성 있고 쉽게 이해할 수 있으면서도 완전하게 표현할 수 있는 구조를 제공해야 한다. 기존 프로세스 모델링 언어는 그 형태에 따라 크게 네 가지로 분류할 수 있다(<표 2>). 프로세스 스크립트는 구조화된 언어 또는 의사 코드를, 프로세스 프로그램은 문법과 의미론을 갖는 정형적 기술 방법을, 양식은 지식기반 언어의 프레임이나 주요 항목을 갖는 표를, 다이어그램은 가시적인 정형성을 각각 이용한다.

많은 프로세스 언어들은 정형성에 초점을 맞추고자 텍스트 형태로 프로세스를 기술하도록 요구하는 반면 그래픽 프로세스 언어들에 대한 연구가 점점 활발해지고 있다. 대개의 경우, 예를 들어 SLANG, Melmac, ProcessWeaver, Teamware 등은 네트워크 형태의 모델을 사용하는데 이들은 한 프로세스 스텝 내에서의 수평적 정보 흐름과 제어 흐름을 강조한다. STATEMATE는 상태 모델에 기초한 세 가지 그래픽 관점을 제공하며, Oikos는 프로세스 개체들 간의 구조적 관계성을 표현하는 다이어그램을 사용하고, APEL은 개체들 간의 제어 흐름도와 자료 흐름도 및 개체 내에 상태도

〈표 2〉 프로세스 모델링 언어의 표현 형태

표현 형태	프로세스 모델링 언어/PSEE
프로세스 스크립트	ISTAR, STATEMATE, BPML,
프로세스 프로그램	AP5, APPL/A, ESP, EPOS, GRAPPLE, HFSP, LOTOS, MARVEL, MVP, OBJ, SLANG, PML
양식(form)	PBS, STATEMATE
다이어그램	AD model, DesignNet, EPM, STATEMATE, Little-JIL, BPMN, UML 확장 언어들

를 제공한다. Little-JIL은 네트워크 형태가 아닌 프로세스의 계층적 분할을 강조하는 다이어그램을 이용하여 다양한 제어 흐름을 표현하도록 지원한다.

2.3 최근 동향

프로세스 모델링 언어의 최근 동향 중 주목해야 할 것은 다음과 같이 요약될 수 있다.

2.3.1 UML을 확장한 프로세스 모델링 언어

대표적인 모델링 언어인 UML의 대중성, 정형성, 가시성을 활용하여 프로세스 모델을 표현하려는 연구가 지속되고 있다[9-11]. 예를 들어 Chou[9]는 작업, 작업의 순서, 작업 동기화, 예외처리를 모델링하는 P-활동 다이어그램과 산출물, 역할, 도구, 일정, 예산 등의 관계를 모델링하는 P-클래스 다이어그램을 제공하는데 이는 UML의 활동 다이어그램 및 클래스 다이어그램을 확장한 것이다. 이러한 UML 확장 언어는 최근 OMG(Object Management Group)에서 SPEM (Software Process Engineering Metamodel)[12]에 대한 명세서를 발표하면서 더욱 활발히 연구될 것으로 보인다. SPEM은 소프트웨어 프로세스 모델과 구성 요소를 정의하기 위한 메타 모델로 UML 표준 의미론에 스테레오타입, 태그, 제약조건이 더해진 UML 프로파일로 구성되어 있으며 완전한 MOF(Meta Object Facility) 기반 메타 모델을 정의한다. SPEM의 기본 아이디어는 실체적 개체(Work Products)에 대한 연산(Activities)을 수행하는 추상적 개체(Process Roles) 간의 협동이 바로 소프트웨어 개발 프로세스라는 것이다. SPEM 표기법은 메타 모델에서 자주 사용되는 클래스들에 대한 가시적인 아이콘들을 제공한다.

2.3.2 특정 개발 프로세스에 적합한 소프트웨어 프로세스 모델링 언어

전통적인 소프트웨어 개발 프로세스와는 다른 방식을 사용하는 도메인을 대상으로 하여 그 특성에 맞는 소프트웨어 프로세스 모델링이 가능하도록 기존 프로세스 모델링 언어를 확장하여 적용하려는 연구가 활발히 진행되고 있다. 개방형 소프트웨어 개발 프로젝트에 초점을 맞춘 연구가 그 예로서 Jensen과 Scacchi의 연구[13]에서는 PML[8,14]의 제어 흐름 문법을 프로세스 메타 모델에 확장하는 방법으로 정형적인 프로세스 표현 언어를 제공하며 Lonchamp의 연구[15]는 SPEM을 개방형 소프트웨어 개발에 맞도록 확장한 다중 레벨 모델링 방식을 정의한다. 이러한 연구들은 전통적인 폐쇄형 소프트웨어 개발과 하이브리드된 형태로 발전할 것으로 예측된다.

2.3.3 비즈니스 프로세스 모델링 언어

소프트웨어 개발 프로세스 분야와는 별도로 비즈니스 프로세스 모델링 언어에 대한 연구가 활발히 이루어지고 있는데 1970년대 IDEF를 시초로 하여 워크플로 기반 시스템을 거쳐 최근 XML을 기반으로 하는 표준안들이 출현하고 있다. BPMI(Business Process Management Initiative)에서 제정한 BPML(Business Process Modeling Language)[16], BEA systems, IBM, Microsoft가 중심이 되어서 IBM의 WSFL과 Microsoft의 XLANG을 통합하고 확장한 BPEL4WS(Business Process Execution Language for Web Services)[17], WfMC(Workflow Management Coalition)에서 제안한 XPDL과 Wf-XML 등 여러 기관과 단체에서 다양한 표준안들을 발표하고 있다. 이러한 XML 기반 언어들은 실행과 프로세스 교환에 초점을 맞추는데 반하여 BPMI가 제안한 BPMN (Business Process Modeling Notation)[18]은 BPD(Business Process Diagram)에 기반한 시각화 표기법을 지원하고 BPEL4WS에 대한 정형적 변환을 제공함으로써 비즈니스 프로세스 모델링 자체의 효율성을 높이기 위한 표준이다. 이들 표준안들은 전체적인 통합 프레임워크 하에서 서로 연계하는 방향으로 발전되고 있다.

3. 프로세스 모델링 언어 RAiSE

3.1 RAiSE 설계 시 고려 사항

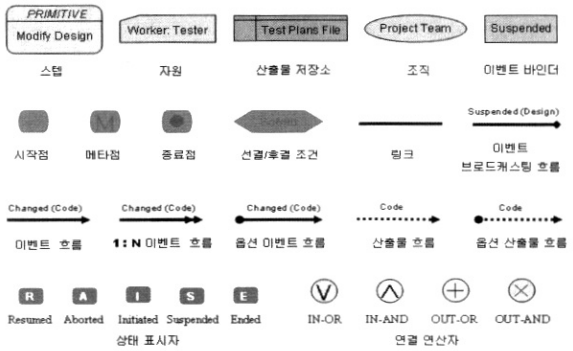
RAiSE의 설계 철학은 프로세스 모델은 프로세스의 각 스텝이 시작되고 종료될 때까지 어떠한 상태 변화를 일으키는지 그리고 상태 변화의 원인이 무엇인지를 명시적으로 표현해주어야 한다는 점에 바탕을 두고 있다. 이러한 상태 기반의 행위 모델링 패러다임을 기초로, 객체 간 정보 전달을 나타내기 위하여 구조적 기법의 자료 흐름도 원리를, 스텝 간 다양한 제어 흐름을 표현하기 위하여 프로세스 프로그래밍 개념을, 개체 상속과 프로세스 스텝의 캡슐화를 지원하기 위하여 객체 지향 개념을 도입하였다. 기타 RAiSE의 설계 시 중요하게 고려했던 사항은 다음과 같다.

- *사용의 용이성*: 프로세스 언어의 복잡성 및 이에 따른 사용하는데 드는 노력은 프로세스 기술이 널리 사용되는데 가장 큰 걸림돌이다. 따라서 소프트웨어 엔지니어라면 누구라도 쉽게 이해하고 사용할 수 있는 문법과 의미론을 제공해야 한다.
- *다양한 프로세스 모델링 관점의 지원*: 프로세스 모델링의 네 가지 관점인 기능 관점, 행위 관점, 조직 관점, 정보 관점을 효율적으로 지원하는 프로세스 언어가 제공되어야 한다.

- *프로세스의 계층적 분할*: 복잡한 프로세스는 반드시 계층적으로 분할하여 표현하고 관리하여야 한다. 특히, 각 레벨별로 중복해서 나타나는 정보를 최소화하여 각 추상화 수준에 맞는 정보만 표현되도록 해야 한다.
- *프로세스 개체간의 관계*: 프로세스 모델의 가장 중심이 되는 개체인 스텝이 다른 개체들과 맺게 되는 다양한 관련성을 모호하지 않게 자연스럽게 표현해주어야 한다.
- *순응적/반응적 제어 흐름*: 일반적인 경우에 있어서 스텝들의 실행 순서를 나타내주는 순응적(proactive) 제어 흐름뿐만 아니라 오류나 사건에 의해 스텝들의 상태나 실행 순서가 변경되는 모습을 나타내는 반응적(reactive) 제어 흐름도 같이 표현되어야 한다.
- *하나의 표현 프레임 워크*: 다양한 관점과 정보를 표현하기 위해 여러 개의 명세화 문서를 작성해야 하는 언어도 있다. 예를 들어, Statemate의 경우 행위도, 상태도, 모듈도 등을 작성해야 하며, UML의 경우 열 가지 가까운 다이어그램을 작성해야 한다. 이러한 접근 방법은 비전문가의 사용을 매우 어렵게 한다.

3.2 RAiSE의 표기법과 의미론

프로세스 모델링 언어 RAiSE[19]는 상호 보완 관계에 있는 다양한 프로세스 모델링 패러다임을 하나의 모델 내에서 비전문가라도 쉽게 이해하고 정의할 수 있도록 표현해주는 그래픽 언어이다. RAiSE는 상태 기반의 행위 중심 모델링 패러다임을 근간으로 하여 주요 프로세스 모델링 패러다임의 핵심 개념들을 반영하는 그래픽 표기법(그림 1)을 제공한다.



(그림 1) RAiSE 표기법

(1) 기본 개체

*스텝*은 누군가에게 할당된 작업을 말하며 다른 개체들은 모두 스텝을 중심으로 연결된다. 스텝은 스텝 타입과 이름을 가지며 계층적 분할에 의해 또 다른 RAiSE 모델로 표현될 수 있다. RAiSE 모델로 확장된 스텝, 즉 대응되는 RAiSE 모델이 존재하는 스텝을 *복합 스텝*이라 하며 더 이상 확장되지 않는 스텝을 *단위 스텝*이라 한다.

*자원*은 한 스텝의 수행에 필요로 하는 개체를 말하며 각 스텝이 실행되기 전 획득되어야 하는 개체들을 표현하는데 사용된다. 자원은 "자원타입 : 자원 인스탄스 타입"의 형식으로 선언되는데 미리 정의된 자원타입에는 스텝을 수행하는 작업자인 Worker, 해당 스텝에 대한 관리 권한을 갖는 소유자인 Owner, 해당 스텝에 필요한 소프트웨어인 SW, 해당 스텝에 필요한 컴퓨터나 장치를 나타내는 HW가 있다. 하나의 RAiSE 모델에 소속된 모든 스텝에 대해 동일한 자원이 요구되는 경우 모델의 오른쪽 상단에 그 자원을 독립적으로 표현한다. 만약 공유 자원과 같은 타입의 자원을 특정 스텝이 별도로 갖는다면 해당 스텝에 대해서는 공유 자원의 효력이 상실된다. 한 스텝의 자원은 하위 스텝에 자동으로 상속되며 복합 스텝은 Owner 타입의 자원만을 갖는다.

*산출물 저장소*는 산출물을 저장하는 물리적 장소(파일, 디렉토리, 데이터베이스 등)를 나타내고, *조직*은 프로젝트에 관여하는 개인이나 팀을 말하며 스텝으로부터 특정 조직이나 개인에게 보내지는 메시지를 표현할 때 사용된다. *이벤트 바인더*는 이벤트를 처리할 때 사용한다. 모든 프로세스 모델은 *시작점* 심볼에서 실행이 시작해서 *종료점* 심볼에서 종료된다. *메타점*은 메타 스텝이 실행되는 시점으로 모델 소유자가 이후 도달 가능한 스텝들에 대한 스케줄링 작업과 작업자 선정 작업을 수행할 수 있다. 시작점은 스텝의 실행 전에 반드시 만족해야 할 *선행 조건*을, 종료점은 스텝 종료 후 만족해야 할 *후행 조건*을 가질 수 있다.

(2) 개체 간 연결

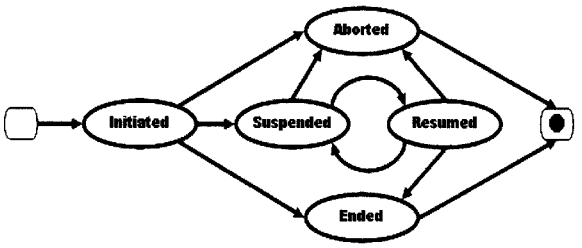
두 개체를 연결하는 방법에는 크게 *산출물 흐름*과 *이벤트 흐름*으로 나눌 수 있다. 산출물 흐름은 산출물 저장소의 산출물을 스텝에 전달하거나 스텝이 산출물을 산출물 저장소에 저장할 경우 사용된다.

*이벤트 흐름*은 발생한 이벤트를 메시지의 형태로 전달하고자 할 때 사용된다. 이 메시지를 *이벤트 메시지*라 하며 특히, 스텝 간 연결을 포함한 대부분의 객체 간 연결은 이벤트 흐름을 이용하여 표현한다. 이벤트 흐름에서 이벤트를 표현하는 형식은 "이벤트 메시지 이름(첨부 산출물 리스트)"로 여러 산출물이 동시에 전달된다면 산출물 사이에 '+'를 넣어 표현한다.

(3) 스텝의 상태

RAiSE는 상태 전이 패러다임에 기반을 두고 있는데, 기본적인 아이디어는 한 스텝은 어떤 산출물을 출력함과 동시에 상태가 변화되며, 이는 또 다른 스텝의 상태에 영향을 미친다는 것이다. 스텝이 인스탄스화 되면, 다섯 상태(Resumed, Aborted, Initiated, Suspended, Ended)[1] 중 하나에 놓여지게 된다. 스텝 상태는 스텝의 외부 또는 내부 요인에 의해 전이되며 가능한 상태 전이는 모두 여덟 가지이다(그림 2). 각 상태에 해당하는 *상태 표시자*는 스텝에 부착되어 산출물 흐름이나 이벤트 흐름과 연결된다. 모든 스텝에는 시작과 종료가 있으므로 I 상태 표시자와 E와 A 중 적어도

1) 앞 글자를 따면 본 프로세스 모델링 언어의 이름인 RAiSE가 된다.



(그림 2) 스텝의 상태 전이

하나의 상태 표시자를 갖는다. 입력 흐름이 상태 표시자로 향한다면 스텝의 외부 요인에 의해 상태 전이가 일어난다는 것을 의미하며, 상태 표시자가 출력 흐름을 갖는다면 스텝 내부에서 상태 전이가 일어난다는 것을 의미한다.

(4) 제어 구조

실제의 소프트웨어 프로세스는 반복, 선택, 병행, 결합 등 다양하고 복잡한 실행 순서를 내포하고 있으며, RAiSE에서는 그러한 실행 순서를 다음 네 가지 연결 연산자를 이용하여 이해하기 쉬우면서도 효과적으로 표현할 수 있다.

- IN-OR : 입력 흐름 중 하나라도 만족된다면 계속 진행한다.
- IN-AND : 입력 흐름 모두가 만족된다면 계속 진행한다.
- OUT-OR : 출력 흐름 중 일치되는 흐름으로 분기한다.
- OUT-AND : 모든 출력 흐름으로 동시에 분기한다.

(5) 이벤트의 처리

이벤트란 스텝 실행에 영향을 주는 사건을 말하며, RAiSE에서는 발생시키는 주체에 따라 모델에 표현된 이벤트 메시지에 의해 발생되는 메시지 이벤트와 시스템 클럭에 의해 발생되는 시간 이벤트로 구분한다.

이벤트를 표현하기 위해서는 이벤트의 발생과 처리 방법을 정의해야 한다. 메시지 이벤트의 발생을 정의하는 방법에는 크게 두 가지가 있다. 첫째, 이벤트 흐름을 이용하여 직접 두 스텝 사이를 연결하는 것인데 메시지를 통해 이벤트를 발생시키는 가장 간단한 방법으로 이벤트의 발생과 처리가 한 모델 내에서 이루어질 때 사용한다. 둘째, 이벤트 브로드캐스팅 흐름을 사용하는 것으로 메시지를 통해 이벤트를 발생시킨다는 점은 이벤트 흐름과 같으나 이벤트 처리가 다른 RAiSE 모델에서 이루어져야 할 때 사용한다. 이벤트 흐름에 의해 발생된 이벤트는 앞에서 본 바와 같이 동일한 모델 내에 있는 스텝의 상태를 전이시키게 되며 이벤트 브로드캐스팅 흐름에 의해 발생된 이벤트는 이벤트 바인더를 통해 처리한다. 이벤트 브로드캐스팅 흐름에 나타나는 이벤트의 이름과 이에 대응되는 이벤트 바인더의 이름은 동일해야 하며, 이벤트와 더불어 전달되는 정보는 이벤트 바인더에 연결된 산출물 흐름의 이름에 바인딩된다. 즉, 이벤트 바인더에 연결된 산출물 흐름의 이름은 프로그래밍 언어의 형식 인자의 역할을 하기 때문에 동일한 이벤트에 대한 처리를 한 곳에서 수행할 수 있다.

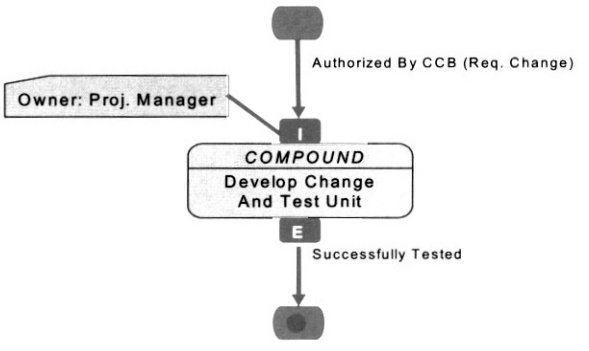
프로세스 엔진에 의해 발생되는 시간 이벤트 역시 이벤트 바인더를 이용하여 처리하며 시간 이벤트는 절대 시작 시간, 절대 종료 시간, 상대 종료 시간을 나타낼 수 있다.

(6) 비결정성(nondeterminism)

프로세스 실행 시간에 의미가 결정되는 비결정성이 가능하도록 RAiSE는 자원의 비결정성 및 스텝 실행의 비결정성을 지원한다. 자원의 비결정성은 스텝 실행에 필요한 자원을 실행 시간에 할당하기 위한 용도로 사용된다. 작업자 자원의 경우, "Worker : 작업자 인스탄스 타입[할당 정책 모드]"로 정의되는데 할당 정책 모드가 R이면 프로세스 엔진은 작업자 인스탄스들 중 임의로(random) 결정하고, W이면 작업자 인스탄스들의 부하량(workload)에 따라 결정되며[2], ?이면 부모 스텝이 시작할 때 또는 해당 단위 스텝이 시작할 때 부모 스텝의 소유자가 결정한다. 예를 들어, 스텝의 작업자가 "Worker: QA Engineer[W]"로 정의되면 프로세스 실행 시 QA Engineer 중 할당된 작업이 가장 적은 엔지니어를 작업자로 선정하게 한다. 스텝 실행의 비결정성은 연결 연산자 중 IN-OR와 OUT-OR에 의해 지원된다. IN-OR는 입력 흐름 중 어느 하나라도 먼저 생성되면 스텝의 실행을 시작시키는 역할을 하며 OUT-OR는 스텝 실행 후 실제로 출력한 결과에 따라 어느 스텝이 실행될지를 결정한다.

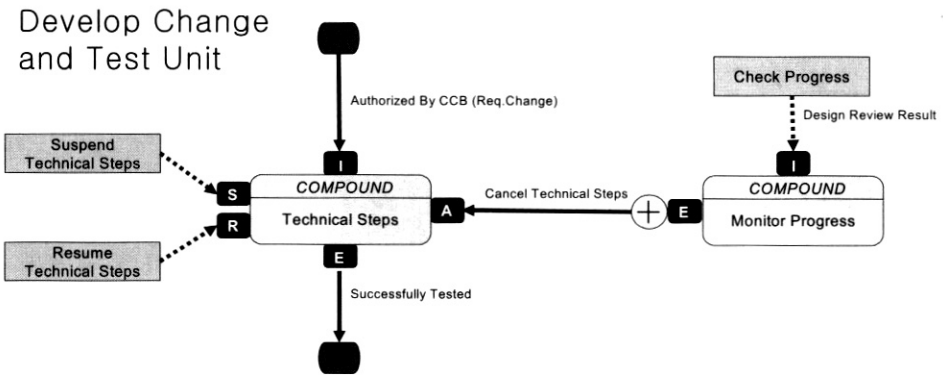
4. 적용 예

RAiSE의 차별성 및 적용 가능성을 분석하기 위해 제 6차 International Software Process Workshop에서 제안된 벤치마크 프로세스인 ISPW-6 소프트웨어 프로세스 예제[3]를 모델링하였다. 이 프로세스는 실세계에서 나타나는 다양한 형태의 프로세스 이슈들을 포함하고 있으며 해결책에 대한 일관성있는 기초를 제공하기 때문에 프로세스 모델링 언어를 충분히 시험하고 장단점을 파악하는데 도움을 준다. 이 예제는 사용자 요구 사항의 변경이 있을 경우 설계, 코딩, 단위 테스트의 관리 문제에 초점을 맞추고 있다.

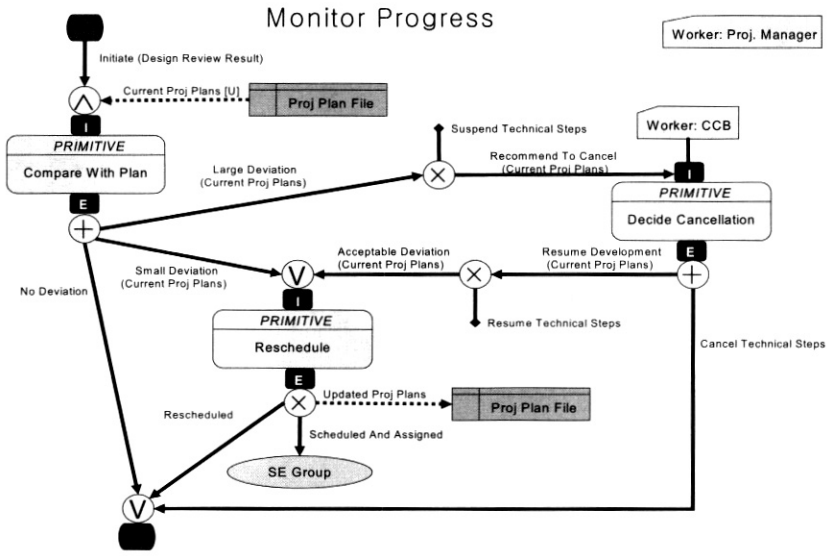


(그림 3) 최상위 RAiSE 모델

2) 디폴트는 W이다.



(그림 4) 스텝 Develop Change And Test Unit



(그림 5) 스텝 Monitor Progress

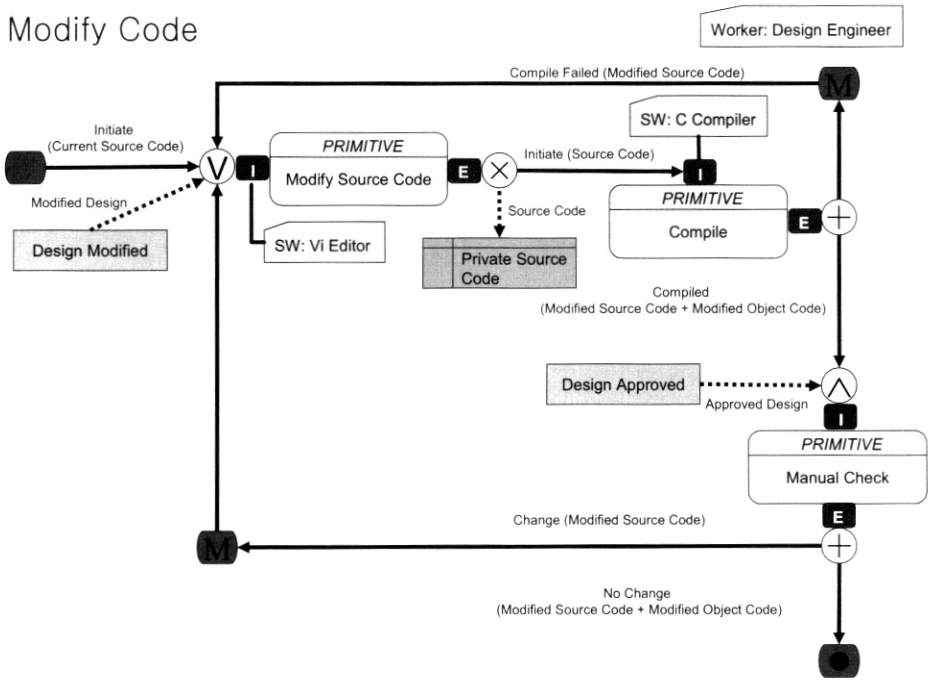
(그림 3)의 최상위 RAiSE 모델은 하나의 스텝 Develop Change And Test Unit으로 이루어져 있는데 이 스텝은 산출물 Req. Change가 첨부된 이벤트 Authorized By CCB에 의해 시작되고, 성공적으로 변경 과정을 마쳤음을 나타내는 이벤트 Successfully Tested를 발생시키면서 종료한다.

(그림 4)는 복합 스텝 Development Change And Test Unit을 확장한 RAiSE 모델로, 변경된 요구 사항에 따라 실제적인 작업을 수행하는 스텝인 Technical Steps와 변경 과정을 모니터링하는 스텝인 Monitor Progress로 구성되어 있다. 스텝 Technical Steps는 세 가지 외부 이벤트[3]에 의해 상태가 변화될 수 있는데 첫째, 이벤트 Suspend Technical Steps가 발생하면 Suspended되고 둘째, 이벤트 Resume Technical Steps가 발생하면 Resumed되며 마지막으로 스텝 Monitor Progress가 종료 시 생성한 이벤트가 Cancel Technical Steps이면 Aborted된다. 스텝 Monitor Progress는 이벤트 Check Progress[4]가 발생하면 시작된다.

(그림 5)는 스텝 Monitor Progress에 대한 RAiSE 모델이다. 이 모델의 스텝 Decide Cancellation을 제외한 두 스텝의 작업자는 Proj. Manager이며, 모든 스텝은 단위 스텝으로서 더 이상 확장되지 않는다. 프로젝트 계획대로 진행되었는지를 조사하는 스텝 Compare With Plan을 시작하려면 산출물 저장소 Proj Plan File에 있는 산출물 Current Proj Plans가 필요하다. 이 스텝이 성공적으로 종료하면 세 가지 메시지 이벤트 중 하나를 발생시키는데 이벤트 No Deviation은 계획과 차이가 없음을 의미하므로 성공적으로 스텝 Monitor Progress를 종료시킨다. 이벤트 Small Deviation의 경우 프로젝트 계획을 수정하기 위한 스텝 Reschedule이 시작되고 이 스텝이 종료하면 산출물 Updated Proj Plans를 산출물 저장소 Proj Plan File에 저장하고 이와 동시에 SE Group에게 수정된 정보를 전달한다. 이벤트 Large Deviation의 경우 이렇게 큰 계획과의 편차를 감안하고서라도 계속 작업을 진행할지, 또는 작업을 중지할지를 판단하기 위해서 일단, 현재 진행 중에 있는 스텝 Technical Steps를 중지

3) 모두 스텝 Monitor Progress 수행 중에 발생 가능한 이벤트이다. (그림 5 참조)

4) 본 논문에는 나타내지 않았지만 스텝 Technical Steps의 각 하위 스텝이 종료될 때마다 발생된다.



(그림 6) 스텝 Modify Code

시키고 동시에 CCB에 의해 수행될 스텝 Decide Cancellation을 시작하게 한다. 이 스텝이 종료되고 나면 계속 진행해도 좋음을 나타내는 이벤트 Resume Development를 발생시키거나 더 이상 변경 작업을 수행할 필요가 없음을 나타내는 이벤트 Cancel Technical Steps를 발생시키게 되는데 전자의 경우는 중지되었던 스텝 Technical Steps의 실행을 재개하는 이벤트 Resume technical Steps를 생성한 후 스텝 Reschedule의 시작으로 이어지며 후자의 경우는 앞에서 설명한 바와 같이 스텝 Technical Steps를 중지시키게 된다.

마지막으로 (그림 6)은 스텝 Technical Steps의 하위 스텝 중 하나인 Modify Code에 대한 모델이다. 스텝 Modify Code가 시작되면, 먼저 Current Source Code를 입력으로 받아 스텝 Modify Source Code가 실행되고 종료 후 Source Code를 저장한 후 스텝 Compile이 실행된다. 컴파일이 실패하면 다시 Modify Source Code 스텝이 실행되며, 성공하면 이벤트 Design Approved가 발생할 때까지 기다렸다가 스텝 Manual Check이 실행된다. Manual Check 결과 코드와 설계 문서가 일치하면 성공적으로 종료되지만 일치하지 않으면 다시 스텝 Modify Source Code가 시작되어야 하며 이때 메타점에서 스케줄링 등의 작업이 모델 소유자에 의해 이루어진다. 이 모델에 포함된 모든 스텝의 작업자는 Design Engineer이다.

5. RAiSE 모델의 실행 환경 PRAiSE

RAiSE로 작성된 모델을 실행시키기 위한 환경인 PRAiSE (PSEE for RAiSE)는 규칙 기반 PSEE로 규칙 기반 패러다임은 정형적으로 프로세스를 정의하고 자동화시킬 수 있으며 소프트웨어 프로세스의 특징인 병렬성을 효율적으로 지원할 수 있다는 장점을 갖기 때문에 여러 PSEE에서 채택되어 왔다[4].

PRAiSE는 프로세스 모델을 작성하고 시뮬레이션하기 위한 프로세스 모델 편집기, 프로세스 수행자와 PRAiSE 간의 통신을 위한 사용자 인터페이스인 작업자 관리기, 작성된 프로세스 모델을 해석하여 실행시키는 프로세스 엔진으로 구성된다[20].

프로세스 모델 편집기는 기본적으로 RAiSE를 이용하여 프로세스 모델을 작성하도록 해주며 추가로 실제 프로세스 실행 전에 작성된 프로세스 모델을 검사하기 위한 시뮬레이션 기능을 제공한다. 해당 모델은 프로세스 엔진이 해석할 수 있도록 자동적으로 사실(fact)의 형태로 변환되어 저장된다.

작업자 관리기는 PRAiSE와 각 작업자간의 메시지 전달을 처리해주는 인터페이스 역할을 담당하며 웹을 통해 어디에서든지 접근 가능하다. 임의의 스텝을 담당할 작업자가 결정되면 해당 작업자의 작업자 관리기로 스텝을 시작하라는 정보가 보내지고 해당 작업자는 스텝을 수행하게 된다. 해당 작업자가 스텝을 수행하면서 해당 스텝의 상태 정보, 예를 들어 스텝의 시작, 종료, 실패 등을 작업자 관리기에 기록한다. 또한 작업자 관리기는 스텝 수행 시 요구되는 입력 조건과 출력 조건을 전달받아서 필요한 정보를 요청하는 역할도 담당한다.

프로세스 엔진은 PRAiSE의 핵심 구성 요소이다. 작성된 프로세스 모델은 규칙 기반 전문가 시스템 도구인 CLiPS [21]로 구현한 프로세스 엔진에 의해 해석되고 실행된다. 프로세스 엔진의 역할은 크게 두 가지이다. 첫째, 프로세스 모델을 해석하는 것으로 CLiPS 추론 엔진과 미리 정의된 실

행 규칙(rule)들로 구성된 해석 모듈에 의해 이루어진다. 사실의 형태로 저장된 프로세스 모델을 실행하기 위해서는 각 사실들이 의미하는 바가 무엇인지를 해석하는 과정이 필요하며 이는 RAiSE 언어의 의미론에 맞도록 프로세스 엔진에 미리 정의해 놓은 규칙(rule)들을 CLiPS 추론 엔진이 해석하여 그 결과를 프로세스 엔진에 전달하는 방식으로 처리된다. 둘째, 해석 결과를 다른 구성 요소에게 알리고 다른 구성 요소들로부터의 정보를 수신하여 CLiPS에게 알려주는 것으로 비주얼 베이직으로 작성된 인터페이스 모듈이 담당한다. 즉, 프로세스 엔진은 CLiPS의 생성 규칙에 의해 만들어지는 새로운 정보들 중 프로세스 모델의 수행, 스텝의 시작 및 종료, 자원 요청 등과 관련된 정보들을 감지해내고 동시에 CLiPS의 작업 메모리에 새로운 정보를 추가함으로써 계속적인 모델 해석 작업을 수행한다. PRAiSE에서는 이러한 프로세스 엔진과 CLiPS사이에 지속적인 정보 교류를 처리하기 위해 CLiPS ActiveX Control을 사용한 통신 메커니즘을 사용한다.

6. 분 석

6.1 가시성

일반적으로 그래픽 기반의 표현 방식은 복잡한 프로세스를 표현하기에 어려움이 많기 때문에 프로세스 실행을 위한 프로세스 모델링 언어들은 대부분 텍스트 기반이다. 그러나 소프트웨어 프로세스의 복잡성을 제어하기 위해서는 다이어그램을 이용한 프로세스 모델의 표현이 필수적이다. 프로세스 가시화의 장점으로는 1) 프로세스에 대한 직관력을 향상시키고 2) 프로세스의 정적 및 동적 특성을 이해하는데 도움을 주며 3) 프로젝트 팀 구성원들 간의 의사소통을 증진시키며 4) 프로세스 실행 중의 변경을 용이하게 한다[5].

프로세스 가시화는 지금까지 두 부류의 접근 방식이 있어 왔는데 IDEF0, Process Weaver, Teamware, Little-JIL 등은 하나의 다이어그램을 이용하는 언어를 제공하며 Statemate, Oikos, SPEM 등은 서로 다른 관점을 지원하는 다중 다이어그램을 제공한다. 전자의 경우 다이어그램이 매우 복잡해질 가능성이 있으며 후자의 경우 다양한 관점들 간의 일관성을 제어하기가 매우 어렵다는 단점을 갖는다. RAiSE는 하나의 다이어그램을 이용하지만 다양한 관점들이 충분히 명료하게 표현됨과 동시에 다이어그램의 단순성을 극대화한다. 다른 프로세스 모델링 언어들을 이용하여 ISPW-6 프로세스를 모델링한 결과를 비교해 보면 RAiSE는 높은 이해도와 다양하고 풍부한 의미론이 제공하는 프로세스 언어임을 알 수 있다. 예를 들어, 대표적인 그래픽 프로세스 언어인 Little-JIL의 경우 70개 이상의 스텝이 표현된 반면[22], RAiSE의 경우 20개미만의 스텝이 표현됨으로써 상대적으로 훨씬 간결하게 프로세스를 표현할 수 있다. 이는 Little-JIL의 경우 제어 추상화와 프로세스 추상화라는 두 가지 추상화에 의해 프로세스가 계층적으로 분할되기 때문에 비단말 스텝이 많아지는 반면, RAiSE의 경우는 프로세스 추상화에 의해 계층적 분할이 이루어지며 제어 추상화는 스텝들 간의 제어 흐름으로 나타나기 때문이다.

6.2 제어 구조

프로세스 모델링 언어가 제공하는 제어 구조는 크게 순방향 제어 및 반응 제어로 구분할 수 있다. 순방향 제어(proactive control)는 프로세스를 최종 목표로 진행될 수 있게 유도하는 것이고 반응 제어(reactive control)는 어떻게 프로세스가 우발적인 사건에 대응하는가를 표현하는 것이다. 이 두 가지 제어는 프로세스를 표현하고 실행시키는데 필수적인 요소이지만 기존의 프로세스 모델링 언어들은 한 가지 방식에만 초점을 맞추고 있다. 예를 들어, Marvel이나 Merlin같은 규칙 기반 시스템들은 반응 제어는 잘 지원하는 반면 순방향 제어는 모델 작성자가 정의한 선행 조건 또는 후행 조건에 의해 시뮬레이션할 수 있을 뿐이다. Teamware, Process Weaver, SLANG 등은 반대로 순방향 제어는 제공하지만 반응 제어는 제대로 지원하지 못한다[5]. SPEM의 경우 순방향 제어는 활동도를 이용하고 반응 제어는 상태전이도를 이용하여 표현할 수 있지만 서로 다른 다이어그램을 사용하기 때문에 프로세스를 이해하거나 일관성을 유지하기 매우 어렵다. RAiSE에서는 순방향 제어는 이벤트 흐름을 이용하여, 반응 제어는 이벤트 브로드캐스팅 흐름을 이용하여 명시적으로 표현할 수 있다.

6.3 자원 및 자원 할당 모델링

프로세스 표현과 실행에 있어 자원은 매우 중요한 요소임에도 불구하고 대부분의 언어들은 이에 대한 정의가 없거나 미흡하다. AP5, APPL/A, HFSP, Marvel, SLANG 등은 자원 정의에 대한 미리 정의된 메커니즘을 제공하지 않는다. SPEM 메타 모델에서도 도구 자원이 일급 클래스(first class)가 아니며[15] Teamware나 EPOS 정도가 인적 자원 및 도구에 대한 기술을 제공하는 수준이다. 그러나 자원의 특성에 따른 융통성있는 할당 방식에 대한 표현과 의미론은 지원되지 못한다. 3장에서 기술한바와 같이 RAiSE는 자원의 비결정성을 지원함으로써 효과적인 자원의 정의 및 유연성있는 자원 할당 정책의 사용이 가능하다.

6.4 메타 프로세스

메타 프로세스(meta-process)는 소프트웨어 제품의 개발과 유지보수를 담당하는 산출 프로세스(production process)의 실행과 동적 진화(dynamic evolution)에 영향을 주는 프로세스이기 때문에 산출 프로세스에 필수적으로 수반되어야 한다[5]. 지금까지는 산출 프로세스와 메타 프로세스의 지원이 결합된 프로세스 모델링 언어를 제공하는 접근 방식이 대부분이다. 예를 들어, EPOS, SLANG, GRAPPLE 등은 산출 프로세스를 데이터의 형태로 표현하고 이를 변경하는 방식으로 동적 진화를 처리한다. 그러나 이러한 방식은 산출 프로세스의 표현과 메타 프로세스의 표현이 혼재되어 있기 때문에 개발자들의 직접적인 관심 대상인 산출 프로세스의

표현을 매우 복잡하게 만들며 이해하기 어렵게 된다. 따라서 단순성을 극대화하기 위해서는 산출 프로세스와 메타 프로세스의 분리가 필수적이다.

RAiSE에서는 메타 프로세스와 관련된 표현을 최소화하기 위해 실제 프로세스 실행 시에 적용되는 별도의 메타 프로세스 관련 의미론을 정의함으로써 이 문제를 해결한다. 대표적인 예로 첫째, 메타점에서는 그 모델의 소유자가 1) 이후 실행될 복합 스텝들에 대한 실제 소유자 선정, 2) 단위 스텝의 작업자 할당 정책이 "?"이면 직접 작업자를 선정(3장 참조), 3) 스텝들에 대한 스케줄링, 4) 모델 자체를 수정 또는 대치할 수 있는 권한을 갖게 함으로써 동적 프로세스 진화가 자연스럽게 이루어진다. 둘째, 복합 스텝에서의 상태 전이는 상태 전파(state propagation)의 개념을 이용함으로써 스텝 실행에 대한 메타 프로세스 및 메타 데이터 표현을 최소화할 수 있다. 예를 들어 복합 스텝이 내부 요인에 의해 Aborted되려면, 하위 모델의 상태가 위(복합 스텝)로 전파되어야 한다. 즉, 하위 모델의 스텝 중 적어도 하나가 Aborted된 상태이고 동시에 하위 모델의 어떤 스텝도 더 이상 수행되지 못한다면 해당 복합 스텝은 Aborted 상태가 된다. 반대로, 복합 스텝이 외부 요인에 의해 Aborted되면 아래로 스텝 상태가 전파되어 하위 모델의 실행중인 스텝들이 모두 Aborted 상태가 된다. 마지막으로 허용되지 않는 상태 전이를 에러로 처리하지 않음으로써 예외 처리를 위한 메타 프로세스의 표현을 산출 프로세스와 분리시키며 동시에 실행의 안정성을 높인다. RAiSE에서 스텝의 상태 전이가 가능한 경우는 모두 여덟 가지이다(그림 2). 나머지 상태 전이는 모두 허용되지 않는다. 예를 들어, Aborted된 스텝은 Resumed될 수 없다. 하지만, 스텝의 상태는 실행 시에 동적으로 변하기 때문에 매우 예측하기가 힘들며, 따라서, 허용되지 않은 상태 전이가 발생할 수 있다. 이 경우 프로세스 엔진은 에러를 발생시키지 않고 무시해 버린다. 이렇게 처리하는 이유는 모델러가 미처 고려하지 못한 상황일 수도 있지만 의도적으로 표현했을 수도 있기 때문이다.

6.5 비즈니스 모델링 언어와의 비교

BPML, BPEL4WS, XPDL, Wf-XML 등의 XML 기반 비즈니스 프로세스 모델링 언어들은 충분한 정형성과 의미론을 가지고 있지만 프로세스 스크립트를 직접 작성해야 하기 때문에 사용하고 이해하기 매우 어렵다는 단점을 가지고 있다[5]. 전자메일 투표 프로세스는 20개 정도의 BPMN 객체가 포함된 1장의 BPD로 표현 가능하지만 이에 대응되는 BPEL4WS 코드는 400줄 정도로 매우 길다[18]. 따라서 XML 기반 비즈니스 프로세스 모델링 언어는 프로세스 실행의 표준으로, 다이어그램 기반 표기법인 BPMN은 비즈니스 프로세스 표현의 표준으로 각각 자리를 잡을 것으로 판단된다.

RAiSE와 BPMN의 BPD는 적용 분야는 다르지만 표현법과 의미론에 있어 유사한 점을 많이 지니고 있다. 기본적으로 작업이 중심이 되는 계층적 네트워크 형태의 프로세스 모델이 만들어지며 이벤트, 예외처리, 다중 인스탄스, 타이머 등을 양쪽 모두 지원한다. 또한 RAiSE의 시작점/종료점, 연결 연산자, 이벤트 흐름, 이벤트 브로드캐스팅 흐름, IN-OR/IN-AND/OUT-OR/OUT-AND는 BPD의 Start/End, Gateway, Sequence Flow, Message Flow, Merging/Join/Inclusive/Fork와 각각 대응된다. 그러나, BPD는 비즈니스 프로세스에 맞도록 다양한 형태의 Type Dimension과 Transaction을, RAiSE는 스텝의 상태를 명시적으로 표현하는 상태 표시자와 다른 스텝의 실행에 영향을 주는 이벤트 브로드캐스팅이라는 차별화된 표기법과 의미론을 각각 제공한다.

일반적으로 비즈니스 프로세스 모델링 언어들은 소프트웨어 프로세스 모델링 언어보다 표기법과 의미론이 상세하고 복잡한데 이는 비즈니스 프로세스는 실행 전에 미리 세밀한 부분까지 정의되어야 하며 엄격한 실행이 요구되고 자동화된 에이전트에 의해 실행될 수 있어야 한다는 점에 기인한다. 반면 소프트웨어 프로세스는 비즈니스 프로세스보다 상대적으로 정의하기 어렵고 소프트웨어 개발자가 이해하고 실행할 정도의 높은 추상화 수준으로 정의되어야 한다. 그러나 소프트웨어 프로세스와 비즈니스 프로세스는 상당한 유사성을 갖고 있기 때문에 각각의 장점을 서로 수용하는 형태로 발전할 것으로 예측된다.

7. 결 론

프로세스 언어는 소프트웨어 프로세스 기술에 관한 관심이 모아지기 시작한 초기부터 가장 중요하게 강조되어온 연구 분야이고 그 동안 다양한 언어들이 개발되었음에도 불구하고 여전히 활발하게 논의되고 있으며 연구가 계속 수행되고 있다. 이러한 연구들의 궁극적 목표는 복잡할 수밖에 없는 소프트웨어 프로세스를 쉽게 정의하고 이해할 수 있으면서도 프로세스 실행의 정확성, 유연성, 안정성을 지원하는 프로세스 모델링 언어의 개발이다. 그러나 아직까지는 사용의 용이성과 엄격하고 풍부한 의미론을 모두 만족시키는 프로세스 모델링 언어가 제공되지 못하고 있다.

본 논문에서는 이러한 단점을 극복한 프로세스 모델링 언어인 RAiSE에 대해 설계 목표를 기술하고 문법과 의미론을 설명하였으며 ISPW-6 소프트웨어 프로세스 예제에 이를 적용한 결과를 기술하였다. RAiSE는 상태 기반 패러다임에 기반을 두고 다양한 패러다임의 장점을 접목한 그래픽 언어로 프로세스의 표현과 이해가 용이하다는 장점을 가지며 제어 구조, 자원 모델링, 메타 프로세스의 지원 등의 측면에서 기존 프로세스 모델링 언어의 단점을 극복할 수 있는 프로세스 모델링 언어이다.

향후 연구 과제로 첫째, 지속적인 실제 프로세스 모델링을 통해 RAiSE의 문법과 의미론을 계속 발전시켜 나가는 작업이 필요하며 프로세스 모델링 프로세스와 방법론을 제공하여 체계적인 프로세스 모델링이 이루어지도록 지원해야 할 것이다. 둘째, 여러 많은 소프트웨어 개발 프로세스를 모

5) Aleri 사의 회장 Alan Hambrook의 말을 인용하면 다음과 같다. "XML is fine, but there's only so much data you can push around in text."

델링하여 PRAiSE에 직접 적용해 봄으로써 프로세스 모델링 및 실행 시 발견되는 문제점을 파악하여 지속적으로 시스템을 향상시켜 나가는 작업도 이어져야 할 것이다. 셋째, 웹 컨텐츠 관리나 형상 관리와 같이 소프트웨어 프로세스가 시너지 효과가 있을 수 있는 분야로 RAiSE를 확장함으로써 범용 프로세스 모델링 언어로의 사용도 추진해야 할 것이다. 마지막으로 소프트웨어 프로세스 모델링뿐만 아니라 최근 각광받고 있는 비즈니스 프로세스 모델링 분야에도 사용될 수 있도록 RAiSE를 확장하는 연구가 필요할 것이다.

참 고 문 헌

[1] Cass, A. G., Lerner, B. S., McCall, E. K., Osterweil, L. J., Sutton, Jr., S. M. and Wise, A., "Little-JIL/Juliette: A Process Definition Language and Interpreter," Proceeding of the 22nd International Conference on Software Engineering, pp.754-757, June, 2000.

[2] Sutton, Jr., S. M. and Osterweil, L. J., "The Design of a Next-Generation Process Language," Proceeding of the Joint 6th European Software Engineering Conference and the 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp.142-158, 1997.

[3] Kellner, M. I., Feiler, P. H., Finkelstein, A., Katayama, T., Osterweil, L. J., Penedo, M. H. and Rombach, H. D., "ISPW-6 Software Process Example," Proceedings of the 1st International Conference on the Software Process, pp. 176-186, 1991.

[4] Fuggetta, A. and Wolf, A., Software Process, John Wiely & Sons Ltd., 1996.

[5] Sutton, Jr., S. M., Tarr, P. L. and Osterweil, L. J., "An Analysis of Process Languages," Technical Report 95-78, Department of Computer Science, University of Massachusetts at Amherst, November, 1995.

[6] Curtis, B., Kellner, M. I. and Over, J., "Process Modeling," Communications of the ACM, Vol.35, No.9, pp.75-90, Sep., 1992.

[7] Osterweil, L. J., "Modeling Processes to Effectively Reason About Their Properties," Proceeding of the PROCSIM'03 workshop, May, 2003.

[8] Atkinson, D. C., Weeks, D. C. and Noll, J., "The Design of Evolutionary Process Modeling Languages," Proceeding of APSEC-2004, Nov., 2004.

[9] Chou, S-C., "A Process Modeling Language Consisting of High Level UML-based Diagrams and Low Level Process Language," Journal of Object Technology, Vol.1, No.4, pp.137-163, Sep.-Oct., 2002.

[10] Jager, D., Schleicher, A. and Westfechtel, B., "Using UML for Software Process Modeling", ESEC/FSE'99, Sep., 1999.

[11] Eriksson, H. E. and Penker, M., Business Modeling with UML, Business Process at Work, John Wiley&Sons, 2000.

[12] OMG, Software Process Engineering Metamodel Specification(Version 1.1), Jan., 2005.

[13] Jensen, C. and Scacchi, W., "Experience in Discovering, Modeling, and Reenacting Open Source Software Development Processes," Proceeding of Software Process Workshop, May, 2005.

[14] Noll, J. and Scacchi, W., "Specifying Process Oriented Hypertext for Organizational Computing," J. Network and Computer Applications, pp.2439-61, 2001.

[15] Lonchamp, J., "Open Source Software Development Process Modeling," in Software Process Modeling, Acuna, S. T. and Juristo, N.(Eds), Springer, 2005.

[16] Arkin. A, Business Process Modeling Language, Nov. 2002.

[17] IBM, Business Process Execution Language for Web Services, May, 2003.

[18] BPMI.org, Business Process Modeling Notation, Working Draft(1.0), August, 2003.

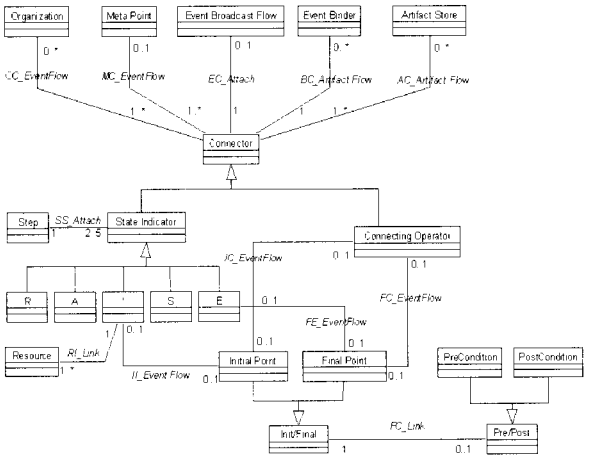
[19] RAiSE 1.0 Language Report, http://www.selab.kangnung.ac.kr

[20] 이형원, 이승진, "PRAiSE : 규칙 기반 프로세스 중심 소프트웨어 공학 환경," 정보과학회논문지 : 컴퓨팅의 실제, Vol.11, No.3, pp.246-256, June, 2005.

[21] Giarratano, J., CLiPS Basic Programming Guide, http://www.ghg.net/clips/, 1998.

[22] Lee, H., "Evaluation of Little-JIL 1.0 with ISPW-6 Software Process Example," Technical Report 99-33, University of Massachusetts, Computer Science Department, March, 1999.

부록 : RAiSE 메타모델



이 형 원

e-mail : lhw@kangnung.ac.kr

1987년 서울대학교 계산통계학과(학사)

1990년 서울대학교대학원 계산통계학과 전산과학전공(이학석사)

1995년 서울대학교대학원 계산통계학과 전산과학전공(이학박사)

1998년~1999년 University of Massachusetts at Amherst 방문연구원

1993년~현재 강릉대학교 정보전자공학부 컴퓨터공학전공 교수

관심분야 : 소프트웨어 프로세스, 형상 관리, 웹 컨텐츠 관리 등