공학 연구논문집 제28권 제1호 pp. 299~316, 울산대학교.
Journal of Engineering Vol. 28, No. 1, pp. 299~316, 1997
University of Ulsan.

객체기반 소프트웨어 프로세스 프로그래밍 언어에 대한 고찰

김영곤 · 박양수 · 이명준
전자계산학과

<요 약>

소프트웨어 프로세스는 사용자의 요구사항을 수행되는 소프트웨어로 실현시키는데 필요한 소프트웨어의 생산 및 유지보수와 같은 복잡한 소프트웨어공학 활동이다. 지난 몇 년동안, 소프트웨어 프로세스 기법에 대하여 많은 연구가 이루어졌다. 특히, 소프트웨어 프로세스를 언어의 형태로 기술하는 프로세스 프로그래밍 언어에 대부분 상당한 관심을 기울여 왔다.

본 논문에서는 객체기반 프로세스 프로그래밍 언어의 주요한 특성에 대하여 조사하고 분석한다. 또한 이러한 프로세스 프로그래밍 언어를 서로 비교하고, 프로세스 프로그래밍에 필요한 언어 구조의 방향에 대해 제안한다.

On Object-Based Software Process Programming Languages

Young-Gon Kim · Yang-Su Park · Myung-Joon Lee
Department of Computer Science

<Abstract>

A software process is complex software engineering activities such as software production and maintenance, needed to realize a user's requirements into functioning

software.. Over the last several years, there has been a lot of research on software process technology. In particular, much of the research concerns process programming languages, wihch describe software processes in linguistic form.

In this paper, we examine and analyze the characteristics of major object-based process programming lanugages. Also, we compare those process programming languages, suggesting a direction to desirable language constructs for process programming.

1. 서 론

소프트웨어 시스템을 구축하는 것은 많은 행위(activity), 생산물(artifact), 사람(personnel), 리소스(resource)들의 복잡한 상호작용을 포함한다[22]. 소프트웨어 프로젝트의 크기와 복잡도가 증가함에 따라 모든 프로젝트 진행 과정을 적절히 조절하고 구성원 사이의 관계를 잘 조화시키는 작업이 어렵게 되어 있다. 이에 따라 소프트웨어를 개발하는 사람에게 여러 가지 유용한 정보를 제공하기 위하여 컴퓨터를 이용한 시스템이 개발되었는데 이를 프로세스 중심 소프트웨어 개발환경(PSEE : Process-centered Software Engineering Environment)라 한다. PSEE는 소프트웨어 생산물의 질을 향상시킬 뿐만 아니라 자동화된 개발 과정을 제공한다. 이 PSEE는 소프트웨어 개발 프로세스에 대한 명시적인 정의인 프로세스 모형(Process Model)을 가지며 이 모형은 일반적으로 프로세스 프로그래밍 언어(Process Programming Language)에 의해 기술된다. 프로세스 프로그래밍 언어는 소프트웨어 프로세스의 각 단계에 대한 추상화를 제공하며, 소프트웨어 생산에 필요한 여러 요소들을 기술할 수 있도록 한다. 현재 다양한 프로세스 프로그래밍 언어가 나와있지만 이들 언어들이 가지고 있는 특성들이 다양하고 소프트웨어 프로세스를 표현하는 방법도 다양하게 지원을 하고 있으므로 이를 종합적으로 살펴보고 그 중심기능을 평가해 보는 작업이 필요하다[18]. 이들 프로세스 프로그래밍 언어는 그 표현방법에 따라 명령문에 기반을 둔 명령문 언어(imperative language), 일련의 규칙에 의한 규칙기반 언어(rule-based language), 그래프로 표현한 그래프 기반 언어(graph-based language), 객체지향 언어(object-oriented language), 트리거에 의한 접근기반 언어(access-oriented language)로 나누어 볼 수 있다[22]. 그러나 현재의 추세를 볼 때, 프로세스 프로그래밍 언어는 이들 언어들의 복합된 형태로 나타나고 있다.

이러한 여러 프로세스 프로그래밍 언어를 살펴볼 때, 객체지향 기술의 발달과 그 응용 경험을 잘 살릴 수 있고 실세계를 잘 반영해 줄 수 있으며 가장 간결한 형태로 소프트웨어 프로세스를 기술할 수 있는 객체기반 언어(object-based language)로 프로세스 프로그래밍을 하는 것이 바람직한 형태라 볼 수 있다. 객체기반 언어는 객체의 개념을 어느 정도 지원할 뿐만 아니라 복잡하고 어려운 프로세스 프로그래밍을 위해 구조화된 기법을 제공해 주고 프로세스 모형을 시뮬레이션하기 쉬운 장점을 가지고 있다. 객체 기반 언어로는 EPOS의 SPELL 언어, ALF의 MASP/DL 언어, Arcadia 프로젝트의 APPL/A 언어, 그리고 E3의 PML 언어등이 있다. 본 논문에서는 이들 기존의 객체기반 프로세스 프로그래밍 언어가 가지고 있는 언어의 구조를 비교 평가하고 바람직한 객체기반 프로세스 프로그래

밍 언어의 방향에 대하여 살펴보고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 프로세스 중심 소프트웨어 개발 환경에 및 프로세스 프로그래밍 언어에 대하여 살펴본다. 프로세스 중심 소프트웨어 개발 환경이 나타나게 된 배경과 소프트웨어 프로세스를 기술하기 위한 프로세스 프로그래밍의 특성, 그리고 객체기반 프로세스 프로그래밍 언어에 대하여 차례대로 살펴본다. 3장에서는 객체기반 프로세스 프로그래밍 언어에 대한 기존의 연구를 대략 살펴보며, 아울러 객체기반 프로세스 프로그래밍 언어가 가지고 있어야할 여러 가지 특성을 APPL/A, E3, MASP/DL, SPELL 프로세스 프로그래밍 언어에 비추어 차례대로 기술한다. 마지막 4장에서는 3장에서 살펴본 네가지 프로세스 프로그래밍 언어들을 평가하여 그 장단점을 살펴보며 기능적으로 균형잡힌 프로세스 프로그래밍 언어가 필요로 하는 요소에 대해 살펴본다.

2. 프로세스 중심 소프트웨어 개발 환경 및 프로세스 프로그래밍 언어

프로세스 중심 소프트웨어 개발 환경(PSEE : Process-centred Software Engineering Environment)은 소프트웨어를 개발하는 사람들에게 유용한 여러 정보를 제공해 주기 위한 컴퓨터 기반 시스템이다. 프로세스 프로그래밍 언어는 소프트웨어 프로세스를 명시적으로 기술하여 그 수행으로 부터 도출되는 생산물들의 질을 보다 향상시키기 위한 도구로서, 현재 객체지향 기법을 이용한 객체지향 프로세스 프로그래밍 언어가 많은 각광을 받고 있다. 본 장에서는 프로세스 중심 소프트웨어 개발 환경에 대한 기술과 아울러 프로세스 프로그래 언어 및 객체기반 프로세스 프로그래밍 언어를 소개한다.

2.1 프로세스 중심 소프트웨어 개발 환경(PSEE)

지난 몇 년 동안 소프트웨어 프로세스는 소프트웨어 생산물의 질을 향상시키기 위한 도구로서 많은 사람들의 관심을 받아왔으며, 아직까지는 산업체에서 많은 테스트를 거치지 않았지만 많은 프로세스 중심 소프트웨어 개발 환경(PSEE)이 설계되고 구축되어 왔다. 이중 Cap의 Process Weaver, ICL의 Process Wise Integrator등 몇몇 PSEE는 상업화되어 발표되었다. 현재까지 PSEE에 대한 기본적인 개념, 기법, 언어구조, 도구의 구조 등에 대하여 합의된 표준은 거의 없으나 기본적인 몇가지 사항에 대한 정의를 살펴보면 다음과 같다.

프로세스(process)는 목적지향, 부분적인 순서화, 상호교환적인 행위(activity)와 이 행위들에 관계된 생산물(artifact)과 리소스(resource)들의 집합으로 이해할 수 있다. 행위, 생산물, 도구등은 프로세스 요소(process element)로 불린다. 프로세스는 생산 프로세스(production process), 메타 프로세스(meta process), 그리고 프로세스 모델과 다양한 프로세스 도구로 구성되는 프로세스 지원 환경으로 구성된다. 생산 프로세스는 부분적으로 컴퓨터 외부적 활동이며 컴퓨터화된 생성 도구에 의해 할당된 사람을 통하여 수행된다. 프로세스 지원 환경은 일반적으로 컴퓨터 내부적인 활동이지만 수동적인 절차에 의해 제공된다. 프로세스 지원 환경은 프로세스에 대한 명시적 정의인 프로세스 모형을 가지며 이 모

형은 일반적으로 프로세스 프로그래밍 언어(PML:process modeling language)에 의해 기술된다. 이때 이 프로세스 프로그래밍 언어는 소프트웨어 프로세스에 대한 각 단계에 대한 추상화를 나타낼 수 있으며 서로 다른 여러 프로세스들를 기술할 수 있다. 이 소프트웨어 프로세스는 지도(guidance), 조절(control), 자동화(automation) 등 사용자에게 유용한 특성을 제공하기 위해 자동적으로 수행될 수 있다.

2.2 프로세스 프로그래밍 언어

지난 몇 년 동안에 걸쳐 소프트웨어 시스템은 소프트웨어 개발 프로세스의 순차적인 수행으로 부터 도출된 결과물로 보아야 한다는 생각이 널리 퍼져왔다. 그러나 소프트웨어 프로젝트의 크기가 매우 비대해 지고 있으며 또한 그 구조가 점점 복잡하여지기 때문에 모든 프로젝트를 관리하고 팀 구성원들의 역할을 잘 조정시키는 작업이 갈수록 어려워지고 있다. 이러한 이유로 인해 소프트웨어 프로세스를 형식화하고 구문적인 형태로 기술하기 위해 프로세스 정의 언어의 필요성이 대두되었다.

프로세스 프로그래밍 언어는 기본적으로 소프트웨어의 생산에 필요한 행위(activities), 생산물(artifacts), 생성도구(production support tool), 대리인(agents) 등을 기술할 수 있어야 한다. 현재까지 제안된 여러 프로세스 프로그래밍 언어들을 그 기법상으로 나누어 보면, 명령 언어(imperative language), 규칙기반 언어(rule-based language), 그래프기반 언어(graph-based language), 객체기반 언어(object-oriented language), 접근기반 언어(access-oriented language)등으로 나눌 수 있다[22]. 규칙기반 언어로는 Marvel[13], MERLIN[11], EPOS[7], Articulator[17], MASP/DL[15] 등의 언어가 있으며, 그래프기반 언어로는 Process Weaver[10], DesignNet[14], EPOS[7], FUNSOFT[9], SLANG[4], VPL[21] 등의 언어가 있다. 객체기반 언어로는 E3[2], PML[5]등이 있고 접근기반 언어로는 APPL/A[19], Adele 2[3], E-L[8], Marvel[13], MASP/DL[15] 등의 언어가 있다.

이러한 언어들의 범주를 살펴볼 때, 객체기반 언어를 이용하여 소프트웨어 프로세스를 모형화하는 것이 바람직한 형태라 볼 수 있다. 이는 객체로 모든 기반 요소들을 표현하는 것이 실세계를 가장 잘 반영해 줄 수 있으며 또한 가장 간결한 형태로 소프트웨어 프로세스를 기술할 수 있기 때문이다. 즉 프로세스 모형이 복잡하고 어렵기 때문에 구조화된 기법의 지원이 필수적이라 볼 수 있으며, 또한 객체기반 모형 설계는 프로세스 모형을 증명하거나 시뮬레이션하거나 구현하기 위해 특정 언어를 사용해야 하는 제약을 가지고 있지 않다. 이러한 객체 기반 접근법은 소프트웨어 프로세스 모형 프로그래밍에 다소간 성공적으로 적용되고 있다[2, 5].

2.3 객체기반 프로세스 프로그래밍 언어

2.1절에서 살펴본 대로 객체기반 프로세스 프로그래밍 언어로는 E3, PML 등을 들 수 있으며 넓은 의미로 보면 EPOS의 SPELL, ALF의 MASP/DL, APPL/A등도 객체 모형을 어느 정도 제공하고 있기 때문에 객체기반 프로세스 프로그래밍 언어에 포함을 시켜 함께 생각해 보기로 한다.

전형적인 객체기반 언어에서 데이터와 프로세스 정보는 객체로 캡슐화된다. 각 객체는 어떠한 작업을 수행할 수 있는 행위(activity)의 집합과 그와 관계된 데이터 값의 집합으로

구성된다. 클래스 계층 구조는 상속을 지원하며 때때로 클래스 사이의 다중 상속도 지원한다. 이는 클래스의 재사용을 용이하게 하여 처음부터 클래스를 새로이 설계하는 대신, 기존의 클래스에 새로운 상태정보의 첨가, 응답형태의 변경, 추가적인 행위의 첨가등으로 새로운 클래스를 생성할 수 있다.

이러한 여러 가지 이유로 인하여 객체기반 언어는 소프트웨어 프로세스 내의 객체를 모형화하는데 적합하다. 상속은 사용자가 최소한의 노력으로 최적화된 클래스를 구축할 수 있도록 도와준다. 그러나 객체 자체만으로는 소프트웨어 프로세스를 자연스럽게 모형화하는데에는 한계가 있다. 소프트웨어 프로젝트 관리를 위한 많은 정보가 행위의 발생에 치중되어 있으며 따라서 프로젝트 관리를 위한 객체기반 프로그래밍 언어는 객체로 모형화된 행위의 표현뿐만 아니라 행위사이의 관계도 표현할 수 있는 언어구조를 제공하는 것이 바람직하다.

3. 객체기반 프로세스 프로그래밍 언어의 구성요소

소프트웨어 프로세스를 객체에 대한 관점에서 볼 때, 이 소프트웨어 프로세스를 기술하고 수행시키기 위해서는 객체기반 프로세스 프로그래밍 언어에 대한 기본적인 특성을 파악하여야 한다. 현재 여러 연구자가 이를 위해 프로세스 프로그래밍 언어가 가져야 하는 다양한 특성을 제안하였지만 아직 공통적으로 받아들여지는 내용은 없으며 객체에 대한 개념으로 이를 분류한 것도 아니다. 그러나 이들 연구자가 분류한 프로세스 프로그래밍 언어가 포함하여야 하는 기본적인 요구사항의 특성으로 Conradi와 Liu[6]는 행위(activity), 생산물(artifact), 역할(role), 사용자(user), 그룹(group), 생성 도구(production tool), 진보(evolution) 등을 제안하였으며 Junkermann[12]등은 행위, 역할, 생산물, 리소스(resource) 등을 외부적으로 명시하고 생산물 사이의 관계를 내부적으로 포함하도록 제안하였다. Lonchamp[16]는 행위, 생산물, 대리인(agent), 역할, 도구, 제약(constraint)이 전통적인 프로세스 개념에 부합한다고 설명하였다.

Sutton과 Tarr[18]는 소프트웨어 프로세스를 기술하기 위해 프로세스 프로그래밍 언어가 가져야할 요소로 행위, 생산물, 리소스, 관계(relationship), 일관성(consistency)을 포함하여야 한다고 제안하였다. 여기서 행위는 프로세스의 일부분으로 발생하는 각 단계를 정의하는 것이며, 생산물은 구현된 코드, 작성된 생성물 등과 같이 소프트웨어 시스템을 구성하는데 필요한 모든 요소를 정의하는 것이다. 리소스는 행위에서 사용되는 사람과 컴퓨터 자원을 명시하는 것이며, 관계는 행위, 생산물, 리소스 사이에 존재하는 논리적인 상호 연결을 지시하는 것이다. 마지막으로 일관성은 상호 연결된 행위, 생산물, 리소스 사이에서 요구되는 조건에 대한 만족을 보장하는 것이다.

이러한 사항을 종합하여 볼 때, 프로세스 프로그래밍 언어가 가져야 하는 기본적인 특성에 대해 기존의 연구는 객체에 대한 근본적인 개념이 없이 진행되었다. 뿐만아니라 너무 외적인 측면에 부각이 되어 실제로 프로세스 프로그래밍 언어가 지니고 있어야 하는 근본적인 개념, 즉 객체에 대한 개념, 관계의 지원, 조건, 수행제어 등에 대한 고려가 부족한 편이다. 따라서 본 논문에서는 이들 객체와 객체사이의 관계, 프로세스의 수행제어, 선행조건 및 후행조건, 트리거, 프로세스의 계층적인 조합에 대해 차례대로 살펴보고자 한다.

3.1 객체(object)의 지원

객체는 각각의 구성요소들을 이용하여 복잡한 시스템을 쉽게 구축할 수 있는 소프트웨어 모형과 개발 원칙으로 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. 이러한 소프트웨어 개발 프로세스의 구성요소로는 행위, 생산물, 도구, 역할등이 있으며 이들은 소프트웨어 프로세스의 핵심 요소가 된다. 프로세스 프로그래밍 언어에서 각 객체(행위, 생산물, 도구, 역할)들은 추상화, 캡슐화, 모형객체, 객체 메소드 등을 가지고 있어야 한다.

추상화된 자료형은 같은 표현을 가지는 객체 집합으로 기술된다. 이때 각 자료형에 관계된 많은 오퍼레이션이 있다. 추상화된 자료형은 자료형에 관계된 사용자 정의 오퍼레이션의 구현을 숨김으로써 자료형을 나타낼 수 있다. 따라서 프로세스 프로그래밍 언어가 추상화된 자료형을 제공하면 사용자가 직접 필요한 객체를 위한 자료구조를 정의하고 자료구조를 이용할 수 있는 오퍼레이션을 작성할 수 있다. 이러한 추상화는 객체내의 데이터가 그 객체내에 정의된 오퍼레이션에 의해서만 조작되도록 하는 캡슐화를 제공하게 된다. 이 캡슐화는 소프트웨어 개발 프로젝트의 진행시 각 프로세스의 내부 상태는 잘 정의된 내부 오퍼레이션에 의해 변경되도록 하고 외부 프로세스에 의해서는 영향을 받지 않도록 하여 일관된 프로세스 상태를 유지하도록 한다.

모형 객체는 프로세스 프로그래밍 언어에서 기본적으로 제공해 주는 기반 객체 모형이다. 이 기반객체는 소프트웨어 프로세스를 모형화할 때 사용되는 객체의 기반 모형을 제공해 주거나 특수한 용도로 사용되는 객체를 나타낸다. 기반 모형으로 사용되는 모형 객체는 모든 객체의 최상위에 위치하는 Object, 그리고 이 Object로부터 상속받아 사용하는 activity, artifact, tool, role등이 있으며 이들은 각각 모든 행위, 생산물, 도구, 역할의 상위 객체 역할을 담당한다. 특수한 용도로 사용되는 객체는 프로세스의 감시, 프로세스의 상태 저장등 프로젝트 수행동안 나타나는 전반적인 사항을 조절하고, 관계된 행위의 기록을 남기는 역할을 담당하는 객체이다. 이들 모형 객체를 제공함으로써 사용자는 모형 객체를 이용하여 보다 쉽게 원하는 객체를 생성하여 이용할 수 있다.

각 프로세스 객체는 자신의 메소드를 정의하여 객체에 관계된 오퍼레이션을 수행할 수 있어야 한다. 객체에 관계된 오퍼레이션은 객체뿐만 아니라 관계(relation)에도 정의될 수 있으나 객체내부 상태에 관계된 오퍼레이션은 객체 내부에 정의하도록 하는 것이 보다 바람직하다.

APPL/A에서는 객체의 추상화와 캡슐화, 모듈화 등을 Ada 언어 자체에서 제공하는 기능을 사용하고 있다. 즉 APPL/A에서는 언어 차원에서 객체를 따로 정의해 놓거나 나타내지 않고 Ada 언어의 자료형으로 나타내며 패키지의 명세부 및 정의부를 나누어 캡슐화를 제공하고 있다. APPL/A에서는 소프트웨어 프로세스의 기술을 위해 이해하기 쉽고 읽기 쉬운 데이터 모형을 제시하는 대신에 확장된 관계 데이터 모형이 소프트웨어 생산물이나 이들 사이의 관계를 표현하는데 더욱 유용하다고 보고 있다. 따라서 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.

E3에서는 모든 객체는 기본 객체인 object로부터 상속을 받는다. E3의 미리 정의된 객체인 task, role, data, tool은 주어진 행위, 역할, 생산물, 도구에 대한 모형 객체를 나타낸다.

모형 객체는 추상화되어 있으며 직접 instantiate될 수 없다. 그러나 이 모형 객체는 사용자가 원하는 객체를 생성하기 위한 상위 객체로 동작하게 된다. 모형 객체인 task, rule, data, tool 등은 프로세스 프로그래밍에 적합한 기능을 기본적으로 제공하며, 사용자 정의 객체는 이러한 기능을 상속받아서 정의할 수 있다. 따라서 E3에서는 기본적인 추상화 및 캡슐화를 객체 클래스로 지원하며, 기본적인 프로세스 요소를 모형 객체인 task, role, data, tool 로 제공하여 사용자가 이 모형 객체로부터 상속받아 자신의 객체를 쉽게 정의할 수 있도록 하였다.

MASP/DL에서 객체는 PCTE의 객체 시스템인 OMS (Object Management System)의 객체 모형이며 타입을 가지는 객체 관계 다이어그램이다. 객체는 "source_file", "tool", "program_library"등과 같이 서로다른 객체 타입으로 나누어질 수 있으며 이들 객체는 공통적인 여러 속성을 가지고 있다. 예는 그림 1과 같다.

```
file:
SUBTYPE OF
object;

document:
SUBTYPE OF
file
with
    attribute
        author : string;
        title : string;
end
document;
```

그림 1. MASP/DL의 객체 모형

이러한 형태로 MASP/DL은 각각의 객체를 정의할 수 있도록 하며 subtype은 이미 정의된 객체를 재사용할 수 있도록 제공한다.

SPELL은 객체지향 패러다임을 따르고 있다. SPELL에서는 프로세스 모형을 나타내는 task와 생성물인 product를 하나의 객체 개념으로 표현하고 있다. SPELL의 프로세스 모형은 기본적으로 재구성 가능하고 서로 연결된 타입을 가진 네트워크로 나타낸다. 이 SPELL에서 제공하는 프로세스 객체 모형은 그림 2와 같이 나타낼 수 있다.

여기서 task는 모든 프로세스 행위를 기술하며 product는 프로세스에 의해 생성된 모든 소프트웨어 요소를 표현한다. E3에서 제공하는 모형 객체로는 TaskEntity와 DataEntity가 있다. PM_Entity는 어떤 프로세스 요소의 클래스이고 두가지 모형 객체인 TaskEntity와 DataEntity의 상위 객체 역할을 한다.

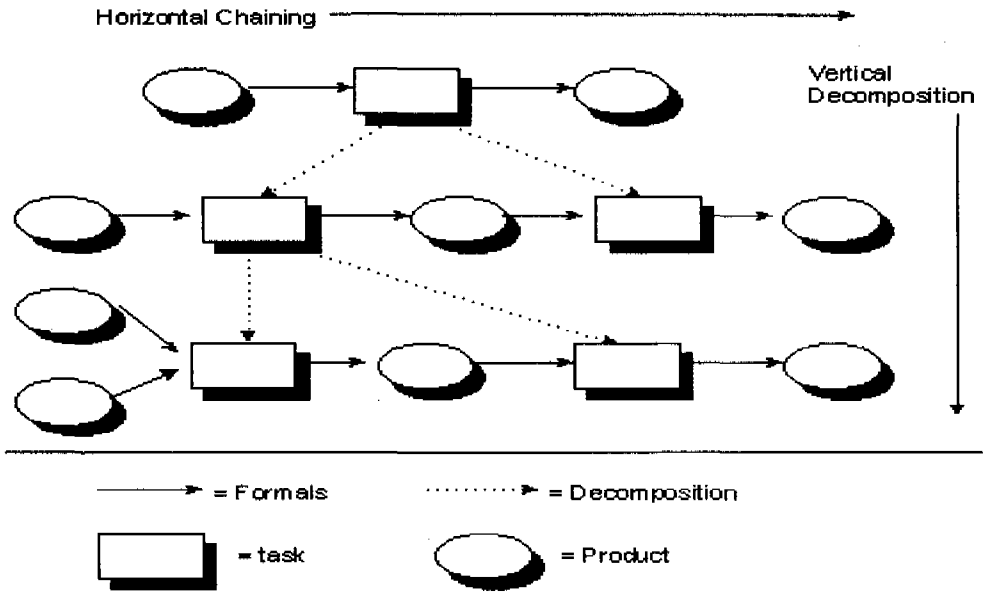

그림 2. 객체의 구성

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.

프로세스 프로그래밍 언어에서 관계를 표현하기 위해서는 상속 (inheritance), 일반적인 관계(general relation), 모형관계, 수행 메소드 등을 기술할 수 있어야 한다. 상속은 관계의 가장 일반적인 한 형태로 볼 수 있으며 객체의 재사용을 위해서는 필수적인 관계이다. 이 상속관계를 이용하여 기존에 정의되어 있는 객체로부터 새로운 객체를 손쉽게 생성할 수 있으며 또한 같은 상위 객체를 가지는 객체사이의 연관관계도 보다 쉽게 파악할 수 있게 된다. 일반적인 관계는 상속을 제외한 객체사이에 존재할 수 있는 관계를 나타낸다. 예를 들어 일반적인 관계로는 activity와 artifact사이에는 "input", "output" 등의 관계가 성립될 수 있으며, artifact와 tool사이에는 "use"의 관계가 성립될 수 있다. 이러한 일반적인 관계를 이용하여 사용자가 보다 다양하게 객체사이의 관련성을 기술할 수 있으며 따라서 객체사이의 관련성을 보다 명확하게 나타낼 수 있다.

관계에 있어서도 객체와 마찬가지로 사용자가 미리 정의된 관계를 사용할 수 있도록 모형 관계가 지원되기도 한다. 이 모형 관계는 각 모형 객체사이에서 발생할 수 있는 기본적인 관계를 나타내게 되며, 이 관계로는 "use", "input", "output", "responsible" 등이 있다. 그리고 각 관계에도 수행 메소드를 두어 객체사이에 발생할 수 있는 오퍼레이션을 수행하

도록 한다. 객체의 수행 메소드는 각 객체 내부에서 발생하는 오퍼레이션을 정의하는 반면, 관계에 대한 수행 메소드는 객체 사이에서 발생하는 오퍼레이션을 정의하여 관계된 두 객체의 상태를 한 오퍼레이션으로 변경시킬 수 있도록 한다.

APPL/A의 Relation unit은 추상화된 수학적인 표기법으로 표현된다. 이는 소프트웨어 객체와 생산물 사이의 관계를 기술하며 소프트웨어 생산물과 다른 소프트웨어 프로세스 데이터간에 나타나는 일반적이고 유연한 자료구조를 제공한다. 일반적인 데이타베이스와 유사하게 APPL/A의 관계는 영속적이며 여러 소프트웨어 프로세스들이 공유할 수 있다. 그러나 APPL/A의 관계는 일반적인 관계형 데이터 모형과는 다른 여러 가지 중요한 차이가 있다. APPL/A 관계내의 속성 타입은 추상화된 복합적인 타입을 가질 수 있으며, 자동적으로 관계내의 속성값이 계산될 수 있다. 또한 APPL/A의 관계는 프로그래밍이 가능하며 추상화되어 있다. APPL/A에 있는 관계의 또 다른 면은 활성화될 수 있다는 것이다. 즉 Ada의 타스크와 비슷하게 수행 흐름을 가지고 있어서 병행성 및 추론, 동적인 메모리의 할당과 같은 목적을 위해 자동적으로 수행될 수 있다. 이 관계에 대한 예는 그림 3과 같다.

APPL/A에서의 관계는 아래의 Source_Compilations에서 보는 바와 같이 type이 tuple로써 정의될 수 있으며 자신이 어떠한 이벤트나 수행을 받을 수 있는 entry를 가지고 있다. 다른 객체나 관계에서 이 entry를 호출함으로써 관계에 대하여 원하는 오퍼레이션을 수행할 수 있다.E3에서 각 객체사이의 관계는 그래픽으로 처리된다. 프로세스 모형 클래스 사이에 주어진 모든 관계는 클래스 단계에서 구축된다. 그러나 수행 가능한 프로세스 모형을 생성할 때, 클래스 단계(클래스사이의 연결)와 인스턴스 단계(객체사이의 연결)의 관계를 구현하여야 한다. 각 인스턴스 단계의 관계는 자신의 클래스 단계의 관계에 의해 연결된 클래스에 속하는 객체와 연결되어야 한다. 모든 클래스 단계의 관계가 자신의 인스턴스 단계의 관계를 가질 필요는 없다.

```
Relation Source_Compilations is
type src_sompilations_tuple is tuple
    name : in name_type;
    src : in Code_Types.source_code;
    obj : out Code_Types.object_code;
    msgs : out messages;
end tuple;
entries
    insert(name : name_type; src : source_code);
    .....
dependencies
    determine obj, msgs by compile(src, obj, msgs);
End Source_Compilations;
```

그림 3. APPL/A의 관계 Source_Compilations

E3에서 제공하는 모형 관계로는 subtask, input, output, preorder, feedback, responsible, use가 있다. subtask는 한 타스크를 부분 타스크로 분해할 수 있는 관계이며 타스크와 타스크사이에 이루어지는 관계이다. input과 output은 타스크에서 사용되는 입력과 출력이며 타스크와 데이터사이에 이루어지는 관계이다. preorder는 두 타스크사이에서 이루어지는 관계로서 타스크사이의 수행순서를 결정해 주는 역할을 한다. feedback은 한 타스크가 다른 타스크에 대한 재수행을 야기시키는 관계이며 재수행된 타스크는 입력으로 feedback 문서를 가지도록 한다. responsible은 타스크와 역할사이에 이루어지는 관계로서 타스크의 수행을 책임지고 있는 사람을 나타낸다. use는 주어진 데이터를 조작하는 도구를 명시하며 데이터와 도구사이의 관계가 이루어진다. 이상에서 설명한 E3의 객체 및 관계에 대한 것은 그림 4와 같다.

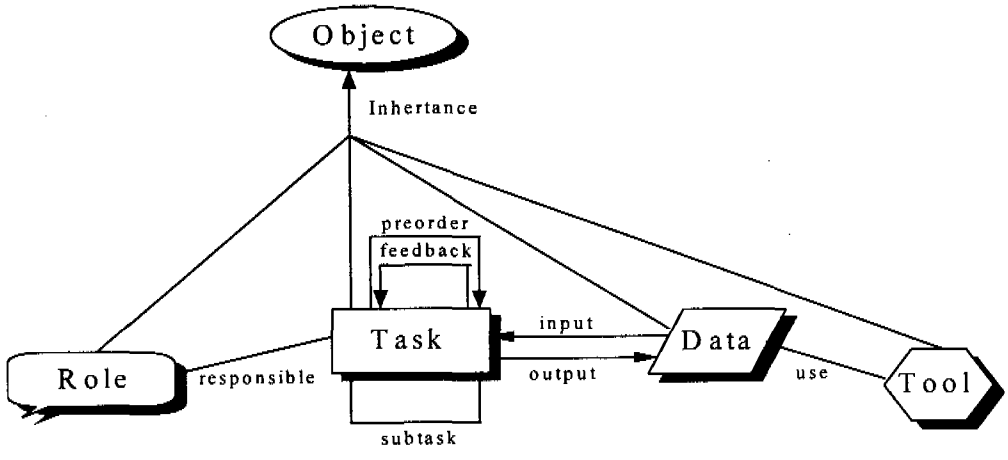

그림 4. E3의 객체 모형 및 관계

MASP/DL에서 관계 타입은 두 링크 타입의 쌍으로 정의한다. 이 관계 타입은 관계의 이름, 목적 객체(destination object)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. MASP/DL에서 사용하는 관계의 예는 다음 그림 5와 같다.

```
RELATIONSHIP(
    is_linked_from:
        COMPOSITION LINK(mod_name) TO object_module
        WITH code_number : INTEGER;
             creation_date : DATE;
    END is_linked_from;
    is_linked_in:
        COMPOSITION LINK TO exec_module
        WITH compilation_date : DATE;
   END is_linked_in)
```

그림 5. MASP/DL의 관계 is_linked_from, is_linked_in

SPELL은 객체사이의 이진 관계를 지원한다. SPELL의 관계형의 집합은 루트에 PM_Rel을 가진 계층으로 구성되어 있다. SPELL의 관계형은 다음 그림 6과 같이 정의된다.

모든 관계는 이진 관계이고 두 개의 연결된 개체형, 즉 목적(destination) 타입과 원시(source) 타입은 역할의 속성에 표현된다. 카디널은 최하와 최상의 범위로 표현되며 위의 그림에서는 원시 타입이 0과 1사이의 카디널을, 목적 타입이 1과 N사이의 카디널을 가지도록 되어 있다. 관계의 접근은 cre_rel, read_rel, del_rel 함수에 의해 객체에 연관되어 있는 관계를 수행시킨다. 관계는 수행중에 동적으로 바뀔 수 없으며 cre_rel 함수에 의해 생성되고 del_rel 함수에 의해 소멸된다.

```
RELATION_TYPE <relationName> : <supertype> {
    ROLES
        SOURCE = <sourceType>     CARD = 0 : 1
        DEST = <destType>         CARD = 1 : N
    ATTRIBUTES
        <attrName> : <domain> := <default>;
        .....
}
```

그림 6. SPELL 관계형의 정의

3.3 프로세스의 수행제어

현재 프로세스 프로그래밍에서 실제의 동작은 타스크나 행위에서 발생한다. 기존에 있는 대부분의 프로세스 프로그래밍 언어에서는 선행조건과 후행조건, 트리거 등을 지원하여 타스크간의 수행 흐름을 수행시에 동적으로 판단할 수 있도록 되어 있다. 그러나 실제의 프

로세스 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.

일반적으로 소프트웨어 프로세스의 수행 흐름을 볼 때, 프로세스 프로그래밍 언어를 이용하여 프로세스를 프로그래밍할 때는 정적인 타스크간의 수행 제어와 동적인 타스크간의 수행 제어를 동시에 제공해주어야 함을 알 수 있다.

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.

E3의 행위는 기본적으로 객체사이의 관계를 이용하여 수행제어가 발생하게 된다. 이 관계는 타스크의 앞, 뒤에 나타나는 데이터를 명시해 줌으로써 타스크의 동기화를 유도할 수 있다. 이와 함께 E3에서는 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. 이때 이 path expression에 나타나지 않은 행위들은 수행 순서에 관계없이 조건만 만족되면 자신의 수행을 계속해 나갈 수 있다.

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

3.4 선행조건(precondition) 및 후행조건(postcondition)의 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 이와함께 타스크간의 동기화도 부분적으로 이룰수 있다.

APPL/A에서 predicate 유닛은 관계상의 조건 명세를 허용한다. predicate은 함수와 비슷하게 호출될 수 있다. predicate은 또한 제약조건과 같이 강제성을 지닐 수도 있다. 이 predicate의 강제성은 APPL/A 내의 일관성을 정의하는데 사용될 수 있다. 일관된 상태는 관계가 그들 사이에 강제된 predicate를 만족할 때이며, 비일관성을 가진 상태는 그 predicate를 만족하지 못할 때이다. 따라서 predicate는 관계에 대해 일관된 상태를 정의하고 강제성을 부여하는데 사용되며 predicate의 강제성이 변화되는 것은 관계에 대한 일관성이 동적으로 변화함을 나타낸다. predicate가 관계에 적용이 되고, 또한 관계는 소프트웨어 프로세스와 생산물의 관련성을 나타내는 데 사용이 되므로 predicate는 소프트웨어 프로세스와 생산물의 일관성을 명시하고 일관성을 유지하기 위해 사용할 수 있다. APPL/A의 predicate 유닛은 그림 7과 같다.

```
predicate Source_Length_Within_Limits is
begin return
    every t1 in Source_Repository satisfies
        length(t1.src) <= Max_Length
    end every;
End Source_Length_Within_Limits;
```

그림 7. APPL/A의 Source_Length_Within_Limits predicate 유닛

E3는 관계나 객체에 대한 선행조건이나 후행조건을 제공하고 있지 않다.

MASP/DL에서 operator는 자신의 이름, 선행조건, 후행조건, 인자 등으로 구성되어 있다. 선행조건은 관련된 타입의 연산을 수행하기 전에 만족해야 하는 제약조건이다. 따라서 선행조건은 연산을 위한 수행 문맥을 허용한다. 만약 선행조건의 평가가 만족되지 않았다면 그 연산은 수행되지 않는다. MASP/DL에서 선행조건은 이벤트 부분이 비어있는 표현으로 볼 수 있다. 후행 조건은 연산의 수행후에 만족해야 하는 조건으로서 선행조건과 마찬가지로 이벤트부분이 비어있는 표현으로 볼 수 있다. MASP/DL에서 선행조건과 후행조건을 이용한 연산은 그림 8과 같다.

SPELL에서 타스크의 수행제어는 실제적으로 수행관리자(Execution Manager)에 의해 제어받는다. 이 수행관리자는 PRE_DYNAMIC, CODE, POST_DYNAMIC의 세부분을 이용하여 수행제어를 담당한다. PRE_DYNAMIC은 주어진 타스크를 수행하고자 할 때의 조건을 명시한다. CODE는 주어진 타스크가 달성해야 할 작업을 하는 순차적인 프로그램이다. 따라서 타스크의 수행은 자신의 CODE의 수행을 의미한다. POST_DYNAMIC은 에러를 다루는 부분이다.

```
audit:(IN _p:project; OUT _r:report)
PRECOND : autid_authorized(_p, _authorized) ∩ _authorized = TRUE
POSTCOND : audit(_p, _a) ∩ _a = TRUE ∩ proj_to_rep(_p, _r, NO_KEY)
KIND : INTERACTIVE
```

그림 8. MASP/DL의 연산

3.5 트리거(trigger)의 지원

트리거는 객체나 관계에 연관된 오퍼레이션이 수행될 때 발생하는 이벤트(event)에 대한 재반응(reactive reponse)로 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 흐름이다. 트리거는 소프트웨어 프로세스내의 병행 조작을 위해 사용될 수 있다. 그 예는 그림 9와 같다.

```
global trigger Maintain_Source_Compilations;    -- specification
trigger body Maintain_Source_Compilations is
    given_name : name_type;
    begin
        loop
            select
                upon
Source_Repository.insert(author:name_type, name:name_type, src:source_code);
                completion do
                    -- propagate name and source to Source_Compilations
                    Source_Compilations.insert(name, src);
                end upon
            or
            .....
            end select;
        end loop;
    end Maintain_Source_Compilations;
```

그림 9. APPL/A의 트리거 Maintain_Source_Compilations

E3와 MASP/DL에서는 트리거를 제공하지 않는다.

SPELL에서의 트리거는 예외상황 처리나 변화의 전달기능을 담당하며 객체내에 정의된 오퍼레이션이 호출되기 전 또는 호출된 후에 특별한 명령을 수행하는 것이다. 그 형태는 그림 10과 같다.

ON_PROC = <*procName*> **WHEN** = <*when*>
***COND** = <*condition*> **ACTION** = <*action*>
***OVERRIDE** = <*overide*>

그림 10. SPELL에서의 트리거

위의 그림에서 <procName>은 타입에 정의된 오퍼레이션의 이름이 되며 <when>에는 before나 after가 사용될 수 있다. <condition>은 수행할 수 있는 조건을 명시하여 <action>에는 조건이 만족되었을 때 수행해야 할 행위를 나타낸다. <override>는 상속성

에 대해 정의를 하게 되며 자신의 <action>만을 수행할지 상위타입의 트리거에 정의된 <action>을 아울러 수행할지 결정할 수 있도록 한다.

3.6 프로세스의 계층적 조합(hierarchical composition)

프로세스 프로그래밍을 한 후 나타나는 여러 가지 프로세스 모형을 다른 프로세스 프로그래밍에서 이용할 수 있도록 프로세스 모형을 계층적으로 구성할 수 있어야 한다. 이 프로세스 모형의 계층적인 조합은 다른 곳에서 사용된 프로세스 모형을 그대로 이용할 수 있도록 구조화된 모형을 제공하는 것을 의미하여 복잡한 프로세스 프로그래밍이 이루어진다면 이미 구성된 프로세스 모형을 효과적으로 이용하는 것이 바람직하다.

현재 MASP/DL 언어를 제외한 다른 APPL/A나 E3, SPELL 언어는 이러한 프로세스의 계층적인 조합을 제공하지 않는다.

4. 토의 및 결론

객체기반 프로세스 프로그래밍 언어인 APPL/A, E3, MASP/DL, SPELL 등의 기능에 대한 분석 및 평가를 통하여 객체기반 프로세스 프로그래밍 언어가 갖추고 있어야 할 여러 가지 요건에 대하여 살펴보았다. 전반적으로 객체와 객체 사이의 관계, 그리고 나머지 수행에 관계된 여러 가지 조건들을 살펴본 결과는 표 1과 같다.

요건 / 프로세스 프로그래밍언어	객체의 지원	관계의 지원	수행제어	선행/후행 조건	트리거	계층적 조합
APPL/A	△	O	O	O	O	X
E[3]	O	O	O	X	X	X
MASP/DL	△	O	O	O	X	O
SPELL	O	O	O	O	O	X

표 1. APPL/A, E3, MASP/DL, SPELL언어의 평가(O는 제공함, △는 부분적으로 제공함, X는 제공하지 않음을 나타낸다)

객체와 객체사이의 관계에 대해 살펴볼 때, E3와 SPELL이 이 두 개념을 지원하고 있음을 알수 있다. APPL/A에서 객체의 지원은 Ada의 자료형이 제공하는 수준에 그치고 있으므로 새로운 객체를 생성할 수는 없다. 마찬가지로 MASP/DL에서의 객체도 완전한 의미의 객체가 아니라 하나의 자료구조만으로서의 객체를 제공한다. 단 네가지 언어 모두 객체사이의 관계는 모두 제공을 하고 있으며 객체사이에 이진관계를 구성할 수 있다.

수행제어 관점에서 볼 때, 네가지 언어 모두 프로세스가 수행될 수 있도록 수행제어를 제공함을 알 수 있다. 다만 APPL/A 언어나 SPELL 언어는 선행조건과 후행조건을 제공하

여 수행제어를 담당하는 반면 E3에서는 preorder 관계를 타스크사이에 성립하여 수행제어를 제공하게 된다. MASP/DL 언어는 선행조건, 후행조건과 아울러 path expression에 나타난 타스크들을 외부에서 명시적으로 수행제어를 조작할 수 있도록 제공한다. 그리고 APPL/A에서의 트리거는 동적인 수행제어를 위해 사용될 수 있으며 SPELL에서의 트리거는 수행제어와는 상관없이 예외상황이나 변화에 대한 전달기능을 담당하고 있다.

특히 MASP/DL 언어에서는 한 객체의 모형이 다른 객체의 모형의 부분 객체로 이용될 수 있는 프로세스의 계층적인 조합을 제공한다. 프로세스의 계층적인 조합은 복잡도가 높은 프로세스를 모형화할 때 이미 개발된 프로세스 모형을 효과적으로 이용할 수 있는 기반을 제공하여 준다. MASP/DL 언어를 제외한 다른 APPL/A나 E3, SPELL 언어는 이러한 프로세스의 계층적인 조합을 제공하지 않는다.

이러한 네가지 객체기반 프로세스 프로그래밍 언어의 주요 특성을 비교, 분석하고 기존의 여타 프로세스 프로그래밍 언어의 기능을 참조하여 본 결과로서, 효과적이고 자연스러운 프로세스 프로그래밍을 지원하기 위해 프로세스 프로그래밍 언어가 갖추어야 할 특성은 다음과 같이 기술될 수 있겠다.

먼저 객체와 이들 객체사이에 대한 관계가 지원되어야 한다. 객체는 행위(activity), 생산물(artifact), 리소스(resource), 역할(role) 등과 같이 소프트웨어 프로세스를 구성하는 요소들을 기술하기 위한 기본이 된다. 이들 객체사이의 관계는 행위나 생산물, 역할과 행위 등에 존재할 수 있는 연관관계를 나타내기 위해 필요한 요소이다. 이와 아울러 가장 기본이 되고 사용자가 쉽게 상속받아서 이용할 수 있는 행위, 생산물, 리소스, 객체 등의 모형객체와 상속(inheritance), 사용(use), 책임(responsible) 등과 같은 모형관계를 지원하는 것이 바람직하다. 또한 SPELL과 같이 수행제어가 외부적으로 기술되지 않고 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. 선행조건 및 후행조건은 소프트웨어 프로세스가 수행되거나 수행된 후의 조건을 나타낸다. 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 바람직하다고 본다. 마지막으로 프로세스의 계층적인 조합은 가능한 한 지원하여 이미 구성된 모형을 재사용할 수 있도록 제공하는 것이 바람직하다.

참 고 문 헌

[1] 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.

[2] Mario Baldi, Silvano Gai, Maria Letizia jaccheri, and Patricia Lage, Obejct

Oriented Software Process Model Design in E3, November 1993

[3] Noureddinge Belkhatir, Jacky Estublier, and Walcelio L. Melo, Adele 2: A support to large software development process, In Proceedings of the First International Conference on the Software Process, pp. 159-170, 1991

[4] Sergio Bandinelli, Alfonso Fuggetta, and Sandro Grigolli, Process modelling in-the-large with SLANG, In Proceedings of the Second International Conference on the Software Process, pp. 75-83, 1993

[5] R.F. Bruynooghe, J.M. Parker, and J.S. Rowles, PSS: A system for process enactment, In Proceedings of the First International Conference on the Software Process, pp. 128-141, 1991

[6] 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, pages 98-118, Springer-Verlag, 1995

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

[8] Thomas E. Cheatham, The E-L system support for process programs, In Takuya Katayama, editor, Proceedings of the 6th International Software Process Workshop, October 1990

[9] Wolfgang Emmerich and Volker Gruhn, FUNSOFT Nets: A Petri-net based software process modelling language, In 6th International Workshop on Software Specification and Design, pages 175-184, October 1991

[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] H. Hunnekens, G. Junkermann, B. Peuschel, W. Schafer, and J. Vagts, A step towards knowledge-based software process modelling, In Proceedings of the First Conference on System Deveopment Environments and Factories, Pitman Publishing, 1989

[12] G. Junkermann, B. Peuschel, W. Schafer, and S. Wolf, Merlin: Supporting cooperation in software deveopment through a knowledge-based environment, In Anthony Finkelstein, Jeff Kramer, and Bashar Nuseibeh, editors, Software process Modelling and Technology, chapter 4, pages 103-129, Research Studies Press, Ltd., Taunton, Somerset, England, 1994

[13] G.E. Kaiser, P.H. Feiler, and S.S. Popovich, Intelligent assistant for software development and maintance, IEEE Software, 5(3):40-49, May 1988

[14] L.C. Liu and E. Horowitz, A formal model for software project management, IEEE Transactions on Software Engineering, 15(10), pp. 1280-1293, October 1989

[15] Amaury Legait, Flavio Oquendo, and Dan Oldfield, MASP: A Model for Assisted Software Processes, volume 467 of Lecture Notes on Computer Science, pages 57-67, Springer-Verlag, 1989

[16] Jacques Lonchamp, An assessment exercise, In Anthony Finkelstein, Jeff Kramer, and Bashar Nuseibeh, editors, Software process Modelling and Technology, chapter 13, pages 335-356, Research Studies Press, Ltd., Taunton, Somerset, England, 1994

[17] Peiwei Mi and Walt Scacchi, Modeling articulation work in software engineering processes, In Mark Dowson, editor, Prodeedings of the First International Conference on the Software Process, pages 188-201, 1991

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

[19] Stanley M. Sutton, Jr., Dennis Heimbigner, and Leon J. Osterweil, Language constructs for managing change in process-centred environments, In SIGSOFT 90 Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments, pages 206-217, ACM Press, December 1990

[20] Stanley M. Sutton, Jr., Accounting for Purpose in Specifying Requirements for Process Programs, CMPSCI Technical Rport 95-76, July 1995

[21] Terry Shephard, Steve Sibbald, and Colin Wortley, A visual software process language, Communcations of the ACM, 35(4):37-44, April 1992

[22] P. Scott Chun Young, Customizable Process Specification and Enactment for Technical and Non-Technical Users, Ph.D dissertation, University of California, Irvine, 1994