PRAiSE: 규칙 기반 프로세스 중심 소프트웨어 공학 환경

(PRAiSE: A Rule-based Process-centered Software Engineering Environment)

이 형 원 † 이 승 진 ††
(hyungwon Lee) (Seungjin Lee)

요 약 규칙 기반 패러다임은 복잡한 프로세스를 처리할 수 있는 정형성과 융통성을 제공하기 때문에 여러 프로세스 중심 소프트웨어 공학 환경에 도입되어 왔다. 그러나, 기존의 규칙 기반 패러다임을 채택한 시스템들의 경우 프로세스 모델을 작성하거나 이해하기 어렵고 프로세스 모델링 언어가 확장 또는 개선될 때마다 추론 엔진을 수정하거나 최악의 경우에는 새로 개발하여야 한다.

본 논문에서는 빈번히 발생하는 프로세스 변경에 유동적으로 대처할 수 있으며 프로세스 모델을 규칙 기반 언어의 사실로 직관적으로 맵핑함으로써 프로세스의 병렬성을 효과적으로 제어할 수 있다는 규칙 기반 패러다임의 장점을 살리면서 기존 규칙 기반 PSEE의 단점인 사용의 용이성과 추론 엔진의 안정성 문제를 해결한 PRAiSE 시스템을 기술한다. PRAiSE에서는 RAiSE라는 그래픽 프로세스 모델링 언어를 제공하며 작성된 프로세스 모델은 규칙 기반 전문가 시스템 도구인 CLiPS로 구현한 프로세스 엔진에 의해 해석되고 실행된다.

키워드 : 소프트웨어 프로세스, 프로세스 중심 소프트웨어 공학 환경, 규칙 기반 패러다임

Abstract Rule-based paradigm is one of the principal types of software process modeling and enaction approaches, as they provide formality and flexibility sufficient to handle complex processes. However, the systems adopting rule-based paradigms are hard to define and understand process models, and their inference engine should be modified or redeveloped at worst according to the change of process language. In this paper, we describe a rule-based PSEE(Process-Centered Software Engineering Environment) PRAiSE that solves the above limitations of existing rule-based PSEEs as well as maintains the merits of rule-based paradigm such as the ability to incorporate the nature of software processes flexibly in which dynamic changes and parallelism are pervasive and prevalent. PRAiSE provides RAiSE, a graphical process modeling language, and defined process models are interpreted and enacted by process engine implemented using CLiPS, a rule based expert system tool.

Key words : Software Process, Process-centered Software Engineering Environment, Rule-based Paradigm

1. 서 론

소프트웨어 공학의 다양한 연구 분야에 있어서 최대의 관심사는 계획된 시간 내에 최적의 품질을 지닌 소프트웨어를 만들어내는 것이다. 보다 좋은 품질의 소프트웨어를 만들어내기 위해서는 소프트웨어 개발에 소요되는 모든 개발 단계를 효율적으로 관리해야 한다. 이러한 목적을 위해 연구되기 시작한 분야가 바로 소프트웨어 프로세스이다. 잘 정의된 소프트웨어 프로세스는 소프트웨어 개발 참여자들이 소프트웨어 개발을 위한 여러 단계를 명확히 이해하고, 그들 간에 의사소통 및 작업을 조율하며, 소프트웨어 개발에 요구되는 자원들을 효율적으로 분배하는데 커다란 도움을 준다[1].

이러한 장점들을 극대화하기 위해 프로세스 모델링 언어로 표현된 소프트웨어 프로세스를 해석하고 그 실행을 자동화하기 위한 다양한 프로세스 중심 소프트웨

† 종신회원 : 강릉대학교 컴퓨터공학과 교수
lhw@kangnung.ac.kr
†† 정 회 원 : 성공회대학교 소프트웨어공학과 교수
lsj@skhu.ac.kr
논문접수 : 2004년 10월 5일
심사완료 : 2005년 2월 28일

어 공학 환경(PSEE: Process-centered Software Engineering Environment)들이 개발되어 왔는데 이들은 지원하는 프로세스 모델링 언어와 시스템의 성격에 따라 상태 기반, 규칙 기반, 절차적 프로그래밍 언어 기반 등의 패러다임으로 분류할 수 있다. 이 가운데 규칙 기반 패러다임은 정형적으로 프로세스를 정의하고 자동화시킬 수 있으며 소프트웨어 프로세스의 특징인 병렬성을 효율적으로 지원할 수 있다는 장점을 갖기 때문에 여러 연구에서 채택되어 왔다[2-5].

그러나, 기존의 규칙 기반 PSEE들은 텍스트 형태의 규칙(rule)과 사실(fact)로 프로세스 모델을 정의해야 하기 때문에 쉽게 프로세스 모델을 작성할 수 없다는 단점을 갖는다. 같은 이유로, 프로세스 모델의 주요 활용 목적 가운데 하나인 프로세스 이해의 저하를 가져올 수 밖에 없다. 이는 대부분의 프로세스 모델 정의자가 규칙 기반 언어에 익숙하지 못한 비전문가라는 사실에 비추어볼 때 지금까지 규칙 기반 PSEE들이 널리 사용되지 못한 가장 큰 원인 중 하나다. 또한, 대부분의 규칙 기반 PSEE는 직접 추론 엔진을 구현했기 때문에 정의된 규칙과 사실을 처리함에 있어 오류를 발생할 가능성이 있으며 프로세스 모델링 언어가 확장 또는 개선될 때마다 추론 엔진을 수정하거나 최악의 경우에는 새로 개발하여야 한다.

본 논문에서는 빈번히 발생하는 프로세스 변경에 유동적으로 대체할 수 있으며 프로세스 모델을 규칙 기반 언어의 사실로 직관적으로 맵핑함으로써 프로세스의 병렬성을 효과적으로 제어할 수 있다는 규칙 기반 패러다임의 장점을 살리면서 기존 규칙 기반 PSEE의 단점인 사용의 용이성과 추론 엔진의 안정성 문제를 해결한 PRAiSE(PSEE for RAiSE) 시스템을 구축하였다. PRAiSE에서는 RAiSE[6]라는 그래픽 프로세스 모델링 언어를 제공하며 작성된 프로세스 모델은 규칙 기반 전문가 시스템 도구인 CLiPS[7]로 구현한 프로세스 엔진에 의해 해석되고 실행된다. 따라서, 시스템 내부의 처리 과정에서만 규칙 기반 패러다임이 사용되기 때문에 사용자 입장에서는 규칙 기반 언어의 개념을 전혀 몰라도 프로세스 모델을 작성할 수 있다.

본 논문의 2장에서는 프로세스 모델링의 기본 개념과 기존의 규칙 기반 PSEE들의 특징에 대해 다루고, 3장에서는 PRAiSE의 구성 요소인 프로세스 모델 편집기, 프로세스 엔진, 작업자 관리기에 대해 기술한다. 4장에서는 대표적인 벤치마크 프로세스인 ISPW-6 소프트웨어 프로세스 예제에 PRAiSE를 적용한 결과를 제시하며 5장에서는 기존 연구와의 차별성을 비교 분석한다. 마지막으로 6장에서는 결론 및 향후 연구 과제에 대해 다룬다.

2. 관련 연구

2.1 소프트웨어 프로세스 모델링

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. 특히, 소프트웨어 개발 비용의 대부분은 개발에 참여하는 사람들에 소요되기 때문에 소프트웨어 개발에 드는 노력의 절감이 곧 소프트웨어 개발 비용의 절감을 의미하며 명시적인 소프트웨어 프로세스의 표현은 불필요하고 비생산적인 행위가 무엇인지 그리고 자동화된 도구에 의해 수행될 수 있는 작업들이 무엇인지 식별하는데 사용될 수 있다. 따라서, 기존 프로그래밍 언어, Petri-Net, 상태 전이도, 자료 흐름도 등을 이용한 다양한 형태의 정형적 기법이 소프트웨어 프로세스를 모델링하는데 사용되어 왔으며 이러한 노력들은 소프트웨어 프로세스 모델링이 프로세스를 개념화하는데 커다란 도움을 주고, 개발자로 하여금 프로세스에 대한 의사 소통을 원활하게 하며, 프로세스 모델의 협동적 실행을 가능하게 한다는 사실을 입증하고 있다.

프로세스 모델링은 소프트웨어 생명 주기 모델을 충분히 상세화시켜 프로젝트 수행에 대한 명시적인 안내 지침을 제공하는 것이 목적이다. 따라서, 프로세스 모델은 무엇을, 누가, 언제, 어디서, 어떻게, 왜 수행하여야 하는가에 대한 정보를 제공하여야 한다. 기존 모델 및 환경들은 이러한 정보들 중에 어느 측면에 초점을 맞추는 가에 따라 구분된다. 즉, 수행하여야 할 작업이 무엇이며 이 작업들과 어떠한 정보 개체들이 연관성을 갖는지를 표현하는 기능(functional) 관점, 작업이 수행되는 시점과 순환, 반복, 선택 등을 통하여 작업이 수행되는 방법을 표현하는 행위(behavioral) 관점, 작업이 어디서 누구에 의해 수행되고 물리적인 의사 소통 수단과 저장 매체 등이 구체적으로 무엇인가를 표현하는 조직(organizational) 관점, 작업에 의해 생성되거나 조작되는 정보 개체의 구성 및 그들 간의 관계를 표현하는 정보(informational) 관점으로 구분할 수 있다.

소프트웨어 프로세스를 표현하고 실행시키기 위해서 프로세스 모델링 언어가 갖추어야 할 핵심 요소들은 행위, 산출물, 자원, 개체 간 관계성 등이다[1,8]. 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

데이타 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 상관 관계를 나타내기 위한 것으로 예를 들어 행위 간의 선행 관계, 산출물 간의 생성 관계, 행위/자원/산출물간의 연관 관계 등을 표현한다.

2.2 규칙 기반 패러다임의 분류

규칙 기반 패러다임의 공통적인 핵심 개념은 규칙과 사실의 집합이다. 사실들은 산출물들의 동적인 상태, 그들간의 관계, 속성을 나타내고 규칙은 프로세스 스텝에 해당하며 상태를 통해 간접적으로 서로 통신한다. 규칙 기반 시스템의 실행 의미론은 규칙을 선택하고 인스탄스화하고 적용하는 추론 엔진의 동작에 의해 정의된다. 소프트웨어 프로세스를 모델링하기 위해 사용되어온 규칙 기반 패러다임은 크게 전문가 시스템, 계획 시스템(Planning System), Prolog의 세 가지 타입으로 구분할 수 있다. 본 장에서는 이들 각각의 대표적인 PSEE인 MARVEL, GRAPPLE, Merlin에 대해 기술한다.

전문가 시스템 유형: MARVEL

MARVEL[1]은 전진 연쇄 추론과 후진 연쇄 추론을 수행하는 전문가 시스템을 프로세스 실행의 근간으로 하는 PSEE이다. 프로세스 모델을 해석하고 실행하는 프로세스 엔진의 수행 메커니즘은 MARVEL 규칙을 통해 이루어진다. MARVEL 규칙은 하나의 선행 조건, 하나의 동작, 하나 이상의 후행 조건으로 구성된다(그림 1). 선행 조건은 동작이 실행되기 전에 만족되어야 하는 조건을 나타내고 동작은 수행되어할 작업을 의미하며 후행 조건은 동작 수행 후 상태 변경이 발생할 때 선택될 수 있는 조건을 담고 있다. 만약 동작이 실행되면 후행 조건 중 하나에 따라 상태가 변경된다. 다중 후행 조건은 성공과 실패와 같이 실행 결과의 택일을 가능하게 한다.

전진 연쇄 추론은 현재 상태에서의 적용 가능한 규칙들을 결정하고 적용할 규칙을 선택한 다음 더 이상 적용할 규칙이 없을 때까지 반복하는 일을 담당한다. MARVEL에서의 전진 연쇄 추론은 이전 스텝들의 수행 결과로 선행 조건이 만족된 프로세스 스텝의 실행을 다룬다. 후진 연쇄 추론은 원하는 상태에 도달하기 위해 현재 상태에 적용할 수 있는 규칙의 집합을 구성한다. MARVEL에서의 후진 연쇄 추론은 선행 조건이 아직 만족되지 않은 스텝을 실행하려고 할 때 빠진 프로세스 스텝들을 채우는 데 사용된다. MARVEL은 병행성을 처리하도록 확장되어 왔으며 전략 언어(strategy language)를 이용하여 규칙의 집합을 모듈화하는 방법을 제공한다.

```
distribute_review_package (?p:rev_product):
(and (forall DEVELOPER ?d
      suchthat (member [?p.reviewers ?d])
       (exists DOC ?rp
      suchthat (linkto ?p.review_pkg ?rp))
:
(DISTRIBUTE mail ?d ?rp)
(?p.review_status = Ready);
(?preview_status = NotReady);
```

그림 1 MARVEL 규칙 예

계획 시스템 유형: GRAPPLE

계획 시스템도 규칙에 대해 전문가 시스템과 유사한 형식을 사용한다. 다만, 프로세스의 단위 스텝(atomic step)에 대응하는 규칙들뿐만 아니라 프로세스 추상화 수준의 상위 계층을 기술하는데 사용될 수 있는 부분 목표 규칙이 추가된다. GRAPPLE[1]에서는 프로세스 목표를 달성하기 위한 계획은 여러 수준의 부분 목표로 이루어져 있으며 이.부분 목표들은 단위 스텝에서 이루어져야 하는 부분 목표들로 세분화된다(그림 2).

계획은 하나의 인스탄스화된 프로세스이며 프로세스의 상태 관련 정보를 포함한다. 이러한 명시적인 프로세스 상태를 이용함으로써 수행해야 할 프로세스 스텝의 목록과 이미 수행된 프로세스의 히스토리가 프로세스 수행자에게 보여진다. GRAPPLE은 목표를 달성하는데 필요한 프로세스 스텝들을 자동적으로 수행하기 위한 계획 생성과 프로세스 수행자에 의해 수행된 스텝들을 현재의 계획 집합에 첨부하기 위한 계획 인식을 결합한 방식을 사용한다.

```
GOAL
      reviewed(p)
Preconditions
      static: in_review_plan(p)
Subgoals
      preparation_complete(p)
      review_meeting_held(m)
      action_items_closed(i)
Constraints
      review_meeting_topic(m,p)
      action_items_listed(m,i)
Effects
      add reviewed(p)
```

그림 2 GRAPPLE 규칙 예

Prolog 유형: Merlin

Prolog에서의 추론 엔진은 내재된 백트랙킹을 갖는 후진 연쇄 추론에만 기반한다. 전문가 시스템과는 반대로, 규칙 작성자, 즉 프로세스를 기술하는 사람은 규칙이 평가되는 순서를 명시한다. Merlin 시스템[2,3]은

```
[specification, m1, module, m1, module, m2]
        for the parameter Read_Access,
[]
        for the parameter Write_Access and
[object_code, m1]
        for the parameter Execute_Access
```

그림 3 Merlin 규칙 예

Prolog 스타일의 규칙과 전진 연쇄 추론에 대한 별도의 문법으로 작성된 규칙을 결합한다. 후진 연쇄 추론 규칙들은 프로세스 수행자가 접근 가능한 모든 산출물과 수행 가능한 모든 작업들을 보여주고 관리하는데 사용되는 반면 전진 연쇄 추론 규칙들은 프로세스 스텝들의 자동화에 사용된다.

Merlin 시스템에서 프로세스 모델의 실행은 역할, 행위, 소프트웨어 객체의 세 구성 요소의 의해 이루어진다. 역할은 행위를 수행하는 담당자로서 사람 혹은 도구일 수 있다. 행위는 MARVEL의 동작과 같은 의미로 이루어져야 할 작업을 나타낸다. 마지막으로 소프트웨어 객체는 소프트웨어 개발 프로세스 단계동안 만들어지는 산출물들을 의미한다.

Merlin 시스템의 세 구성 요소가 서로 유기적으로 동작하기 위해 시스템 내부적으로 프로세스 모델을 Prolog 형태의 규칙과 사실로 표현한다. 여러 규칙의 제어는 추론 엔진이 담당하게 함으로써 변경이 수시로 발생하는 소프트웨어 프로세스에 유연하게 대처할 수 있다. 그림 3에 Merlin 시스템의 규칙의 예가 있다.

3. PRAiSE의 구성

규칙 기반 PSEE인 PRAiSE는 프로세스 모델을 작성하고 관리하기 위한 프로세스 모델 편집기, 작성된 프로세스 모델을 해석하여 실행시키는 프로세스 엔진, 프로세스 수행자와 PRAiSE 간의 통신을 위한 사용자 인터페이스인 작업자 관리기로 구성된다. 본 장에서는 이 구성 요소들의 설계와 구현에 대해 기술한다.

3.1 프로세스 모델 편집기

3.1.1 프로세스 모델링 언어 RAiSE

PRAiSE에서 지원하는 프로세스 모델링 언어 RAiSE[6]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 4)을 제공한다.

기본 개체

*스텝*은 누군가에게 할당된 작업을 말하며 다른 개체

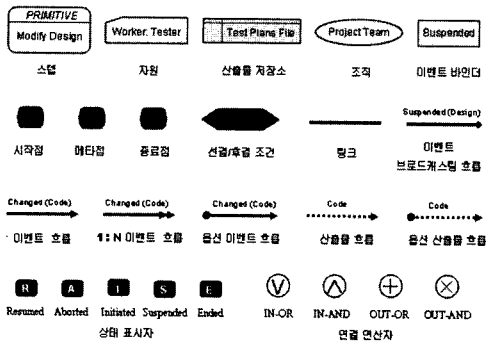

그림 4 RAiSE 표기법

들은 모두 스텝을 중심으로 연결된다. 스텝은 계층적 분할에 의해 또 다른 RAiSE 모델로 표현될 수 있다. *자원*은 한 스텝의 수행에 필요로 하는 개체를 말하며 각 스텝이 실행되기 전 획득되어야 하는 개체들을 표현하는데 사용된다. *산출물 저장소*는 산출물을 저장하는 물리적 장소를 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.

개체간 연결

*링크*는 주로 자원과 스텝 사이를 연결하는데 사용된다. *산출물 흐름*은 두 개체 사이를 연결하는 점선 화살표 및 부착된 산출물 이름으로 구성되며 개체의 유형에 따른 동작 메커니즘 및 두 개체 사이에 전달되는 산출물을 표현하는데 사용된다. *이벤트 흐름*은 두 개체 사이를 연결하는 실선 화살표 및 화살표에 부착된 이벤트로 구성되며 발생한 *이벤트를 메시지*의 형태로 전달하고자 할 때 사용된다. 이 메시지를 이벤트 메시지라 하며 특히, 스텝과 스텝 사이의 연결은 모두 이벤트 흐름을 이용하여 표현한다.

스텝의 상태

스텝이 인스탄스화 되고 나면, 다섯 가지 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 가지

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

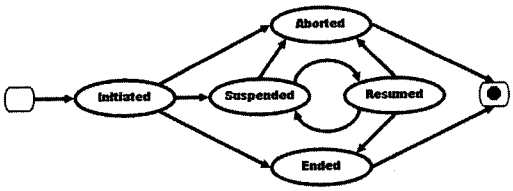

그림 5 스텝의 상태 전이

이다(그림 5). 각 상태에 해당하는 *상태 표시자*는 스텝에 부착되어 산출물 흐름이나 이벤트 흐름과 연결된다. 모든 스텝은 I 상태 표시자와 E와 A 중 적어도 하나의 상태 표시자를 갖는다.

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.

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 이벤트는 *이벤트 브로드캐스팅* 흐름을 사용하여 발생시키며 *이벤트 바인더*를 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 동일 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.

이벤트가 브로드캐스팅 방식으로 처리되기 때문에, 한 이벤트에 대해서 여러 스텝이 실행될 수 있으며 이들 간에는 자원 할당 등의 충돌이 발생할 수 있다. 이를 제어하기 위해 이벤트를 처리하는 스텝에 우선 순위를 설

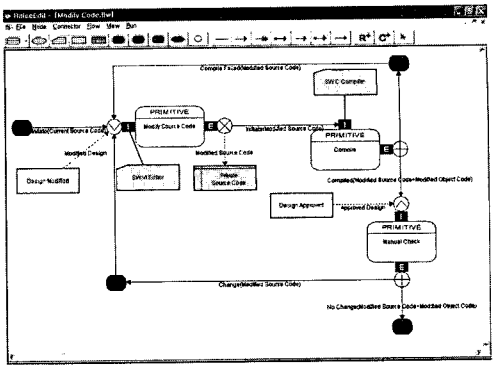

그림 6 프로세스 모델 편집기를 이용한 Modify Code 스텝의 표현

정할 수 있는데, 가장 높은 우선 순위를 갖는 스텝부터 실행을 시도하여 자원을 할당받는다. 우선 순위를 설정하지 않으면 모두 동시에 실행되며 실행 순서는 프로세스 엔진이 임의로 결정한다.

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 **R**이면 프로세스 엔진은 작업자 인스탄스들 중 임의로(random) 결정하고, **W**이면 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.

3.1.2 프로세스 모델 편집기의 구현과 적용 예

프로세스 모델 편집기는 프로세스 모델의 작성과 시뮬레이션 기능을 갖는다. ISPW-6 벤치마크 프로세스[9]의 모델링 결과의 일부가 그림 6으로, 소스 코드를 수정하고 컴파일하는 과정을 담고 있다. Modify Code 스텝이 시작되면, Current Source Code를 입력으로 받아

2) 디폴트는 W이다.

Modify Source 스텝이 실행되고 종료 후 Source Code를 저장한 후 Compile 스텝이 실행된다. 컴파일이 실패하면 다시 Modify Source Code 스텝이 실행되며, 성공하면 Design Approved 이벤트가 발생할 때까지 기다렸다가 Manual Check 스텝이 실행된다. Manual Check 결과 코드와 설계 문서가 일치하면 성공적으로 종료되지만 일치하지 않으면 다시 Modify Source Code 스텝이 시작되며 이 때 메타점에서 스케줄링 등의 작업이 모델 소유자에 의해 이루어진다. 이 모델에 포함된 모든 스텝의 작업자는 Design Engineer이다.

편집기의 File 메뉴에는 프로세스 모델에 대한 저장하기 및 불러오기, Undo, Redo 기능이 있으며 Node, Connector, Flow 메뉴를 이용하면 개체, 연결 연산자, 산출물/이벤트 흐름들을 선택하여 화면에 그리고 서로 연결할 수 있다. View 메뉴는 화면의 비율을 지정하여 확대 및 축소할 경우에 사용하며 Run 메뉴는 실제 프로세스를 실행하거나 시뮬레이션할 때 사용한다. 메뉴 아래쪽에 있는 툴바에서는 그래픽 개체들을 직접 사용할 수 있도록 하였다.

프로세스 모델 편집기는 윈도우 XP 운영 체제하에서 실행되며 비쥬얼 베이직 6.0을 이용하여 개발되었다. 그래픽 심볼들을 표현하기 위해 Addflow ActiveX Control을 사용하였다.

3.1.3 프로세스 모델의 저장

작성된 프로세스 모델은 두 가지 형태로 저장된다. 첫째, 프로세스 모델 편집기가 필요로 하는 그래픽 정보들의 집합으로 저장되며 둘째, 프로세스 엔진에 의해 실행될 수 있도록 CLiPS 사실의 형태로 변환되어 저장된다. 후자의 경우 각 개체에 대한 사실은 유형 별로 설계된 템플릿에 실행에 필요한 개체 정보를 담고 있는 형태로 생성된다. 예를 들어, 가장 대표적인 개체인 스텝은 스텝 타입과 이름을 가지며 계층적 분할에 의해 또 다른 RAiSE 모델(대응 모델)로 표현될 수 있다. 이와 같은 정보를 담고 있는 스텝에 대한 CLiPS 사실 구조는 그림 7(a)와 같다. 또한, 이벤트 흐름의 경우 타입과 이름뿐만 아니라 이벤트 흐름에 의해 연결된 개체들, 전달되는 산출물의 이름 등에 관한 정보를 포함한다. 이벤트 흐름에 대한 CLiPS 사실 구조는 그림 7(b)와 같다.

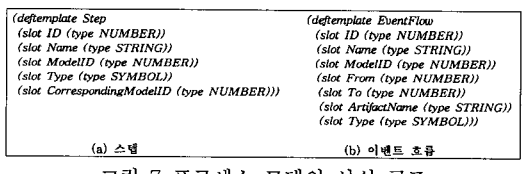

그림 7 프로세스 모델의 사실 구조

```
① Fact = "(OutputResourceInfo (ResourceOID " + resource_id
         + ")(ResourceName " + resource_name + ")(StepOID " + step_id + "))"
② CLiPSActiveXControl.AssertString (Fact)
   Fact = "(EndStep (StepOID " + step_id + "))"
   CLiPSActiveXControl.AssertString (Fact)
```

그림 8 CLiPS 사실의 추가

3.2 프로세스 엔진

PRAiSE의 핵심 구성 요소는 프로세스 모델을 해석하여 실행시키는 프로세스 엔진이다. 본 절에서는 프로세스 엔진과 CLiPS 시스템의 추론 엔진과의 상호 작용 메커니즘을 기술하고 프로세스 엔진의 실행 규칙에 대해 논한다.

3.2.1 CLiPS와 프로세스 엔진의 상호 작용

프로세스 엔진의 역할은 크게 두 가지로 하나는 프로세스 모델을 해석하는 것으로 CLiPS의 추론 엔진과 미리 정의된 실행 규칙들로 구성된 해석 모듈에 의해 이루어지며, 다른 하나는 해석 결과를 다른 구성 요소에게 알리고 다른 구성 요소들로부터의 정보를 수신하여 CLiPS에게 알려주는 것으로 비쥬얼 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 감지해내야 하고 동시에 CLiPS의 작업 메모리에 새로운 정보를 추가함으로써 계속적인 모델 해석 작업을 도와야 한다. 따라서, 프로세스 엔진과 CLiPS사이에 지속적인 정보 교류가 필요하며 이를 위해 본 연구에서는 CLiPS ActiveX Control을 사용한 통신 메커니즘을 도입하였다. 예로서, 스텝의 종료 시 CLiPS 사실이 추가되는 과정을 살펴보도록 한다. 스텝 수행이 완료되면 작업자는 프로세스 엔진에게 해당 스텝이 종료하였음을 알린다. 엔진은 이 사실을 통보 받은 후 스텝이 종료하였음을 의미하는 CLiPS 사실을 작업 메모리에 추가하는 일을 한다. 이렇게 함으로써 CLiPS 추론 엔진은 그 다음 스텝을 수행시키기 위한 모델을 해석하는 작업을 계속 이어나갈 수 있다. 그림 8은 CLiPS 사실을 작업 메모리에 추가하는 코드로 ①은 "Fact" 라는 스트링형의 변수에 추가할 CLiPS 사실을 만들어서, ②의 AssertString 메소드를 이용하여 CLiPS의 작업 메모리에 사실을 추가한다. CLiPS 추론 엔진은 이러한 사실을 바탕으로 스텝의 상태를 종료 상태로 변화시키고 다음에 수행될 스텝을 위한 준비를 한다.

3.2.2 프로세스 모델의 실행을 위한 규칙

사실로 저장된 프로세스 모델을 실행하기 위해서는 각 사실들이 의미하는 바가 무엇인지를 해석하는 과정이 필요한데, 이것은 RAiSE 언어의 의미론에 맞도록 프로세스 엔진에 규칙을 미리 정의해 놓음으로써 해결

할 수 있다. 이 규칙들은 CLiPS 추론 엔진에 의해 해석되어 그 결과가 프로세스 엔진에 전달된다. 여기에서는 프로세스 모델을 해석하기 위한 여러 규칙 가운데 스텝의 시작과 종료에 관한 규칙에 대해 기술한다.

스텝 시작에 관한 규칙

스텝 시작을 위해서는 먼저 스텝에 부착된 연결 연산자와 연산자에 연결된 이벤트 흐름이 해석되어야 한다. 따라서 스텝이 시작되는 순서는 이벤트 흐름, 연결 연산자, 스텝 순으로 스텝 시작을 위한 정보들이 새로 추가되면서 이루어진다(그림 9). '*EventFlow*' 규칙은 이벤트 흐름 사실과 이와 연관된 연결 연산자 사실이 존재하면 해당 이벤트 흐름이 해석되고, '*ConnectingOperator*' 규칙은 '*EventFlow*' 규칙에서 새롭게 추가된 사실과 이와 연관된 연결 연산자가 존재하면 해석된다. 마지막으로 '*Initiate_Step*' 규칙은 기 해석된 연결 연산자와 이벤트 흐름을 바탕으로 스텝 시작에 필요한 입력 산출물에 대한 사실(*InputArtifactInfo*)과 출력 산출물에 대한 사실(*OutputArtifactInfo*)이 존재하면 비로소 해당 스텝이 시작한다는 사실(*InitiateStep*)를 새로 추가함으로써 해당 스텝을 시작하게 된다.

스텝 종료에 관한 규칙

스텝 종료는 프로세스 엔진 외부로부터 해당 스텝이 종료되었다는 사실(*EndStep*)이 추가되면 그림 10의 규칙이 해석되면서 이루어진다. '*End_Step*' 규칙은 스텝이 종료되었다는 정보와 기존에 스텝이 시작될 때 추가된 사실(*OStep*)을 바탕으로 스텝 상태를 '*I*(*Initiate*)'에서 '*E*(*End*)'로 변경함으로써 마무리된다.

```
(defrule EventFlow
 (OModel (OID ?oid) (ModelID ?mid))
 (EventFlow (ID ?eid) (ModelID ?mid) (ArtifactName ?alist) (From ?cid) (To ?cid))
 (ConnectingOperator (ID ?cid) (ModelID ?mid) (StepID ?sid))
=> (assert (OEventFlow (OID ?*id*) (EventFlowID ?eid) (ModelOID ?oid)
          (ArtifactName ?alist) (Frequency 1))))

(defrule ConnectingOperator
 (OModel (OID ?oid) (ModelID ?mid))
 (ConnectingOperator (ID ?cid) (ModelID ?mid) (StepID ?sid))
 (OEventFlow (OID ?eoid) (EventFlowID ?eid) (ModelOID ?oid))
=> (assert (OConnectingOperator (OID ?*id*) (ModelID ?mid) (ModelOID ?oid)
          (ConnectingOperatorID ?cid) (StepID ?sid)))

(defrule Initiate_Step
 (OModel (OID ?oid) (ModelID ?mid))
 (Step (ID ?sid) (Name ?sname) (ModelID ?mid))
 (OStep (OID ?soid) (State Initiating) (StepID ?sid) (ModelOID ?oid))
 (InputArtifactInfo (Name ?iartifact) (ArtifactOID $?qfoid) (StepID ?sid) (ModelID ?mid))
 (OutputArtifactInfo (Name ?oartifact) (StepID ?sid) (ModelID ?mid) (Output Determined))
=> (assert (InitiateStep (StepName ?sname) (StepID ?sid) (ModelID ?mid))
   (assert (OStep (OID ?soid) (ModelID ?mid) (State I) (StepID ?sid) (ModelOID ?oid))))
```

그림 9 CLiPS 규칙: *Initiate_Step*

```
(defrule End_Step
 (OModel (OID ?oid) (ModelID ?mid))
 (Step (ID ?sid) (Name ?sname) (ModelID ?mid) (Type Primitive))
 ?x <- (OStep (OID ?soid) (State I) (StepID ?sid) (ModelOID ?oid))
 (EndStep (StepID ?sid) (ModelID ?mid))
=> (retract ?x)
   (assert (OStep (OID ?soid) (ModelID ?mid) (State E) (StepID ?sid) (ModelOID ?oid))))
```

그림 10 CLiPS 규칙: *End_Step*

3.3 작업자 관리기

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.

그림 11은 작업자가 할당받은 스텝의 상세 정보를 선택하였을 때 보여지는 화면이다. Assigned Step은 할당받은 작업들의 리스트를 보여주고 State는 할당받은 작업들의 현재 상태를 나타낸다. 할당받은 작업의 상태로는 Initiate, Resume, Suspend, Abort, End가 있다. 작업자는 모델에서 정의된 스텝의 입력 산출물과 출력 산출물을 스텝의 상세 정보에서 확인할 수 있으며 입력 산출물 리스트의 파일을 다운로드 받아 작업을 실행한 후 그 결과인 출력 산출물을 생성하게 된다. 그림 11의 경우, 작업자는 Modify Source Code 스텝을 선택하였으며 입력 산출물인 Current Source Code를 선택하면 해당 코드를 다운로드 받을 수 있다. 작업이 끝나면 결과물인 Modify Source Code라는 파일을 별도의 업로드 창을 통해 서버로 업로드하게 된다. 작업자는 작업의 진행상황을 판단하여 스텝의 상태를 변경(작업 보류, 작업 중지, 종료)할 수가 있으며 프로젝트 관리자는 작업자들에게 할당된 스텝의 상태를 보고 프로젝트의 진행 정도를 파악할 수 있다.

PRAiSE에서는 산출물 저장소에 입출력되는 문서들은 자동적으로 형상 관리의 대상이 된다. 즉, 해당 산출물을 산출물 저장소로부터 다운로드하거나 산출물 저장

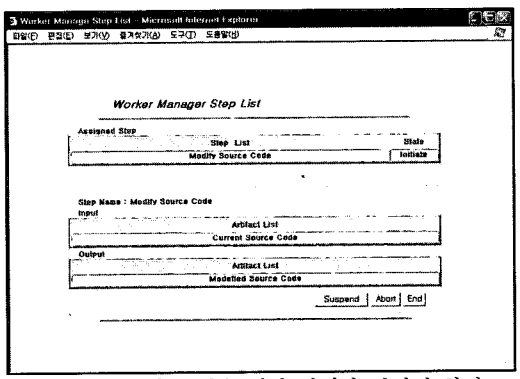

그림 11 스텝 수행을 위한 작업자 관리기 화면

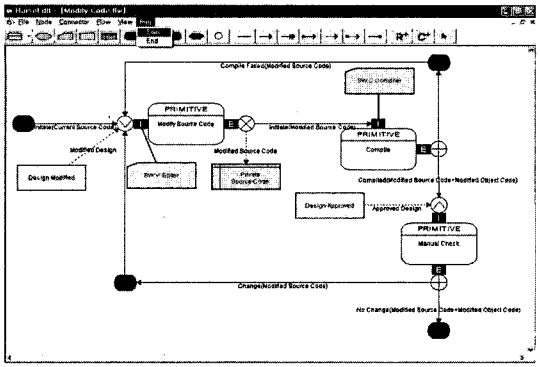

그림 12 프로세스 모델 실행

소로 업로드하는 경우, 체크-아웃이나 체크-인(커밋)이 가능하도록 CVS 웹 인터페이스인 TortoiseCVS가 실행된다. 다만, 어떤 버전을 체크-아웃할 것인가는 작업자의 결정에 따른다. 산출물 저장소는 CVS 리포지토리 내의 디렉토리에 해당하며 각 산출물들은 프로세스 모델에서 정의된 이름과 동일한 이름의 파일로 저장된다.

4. 프로세스 모델의 실행 예

본 장에서는 PRAiSE의 효율성 및 적용 가능성을 입증하기 위해 그림 6의 Modify Code 모델이 실행되는 과정을 기술한다. 프로세스 모델이 실행되는 과정은 프로세스 모델 편집기를 통해 모니터링할 수 있다.

1) Modify Code 모델의 실행

프로세스 모델 편집기를 통해 프로세스 모델을 불러온 후 Run 메뉴의 Start 항목을 선택하여 모델을 실행시킨다(그림 12). 위와 같은 프로세스 모델이 저장될 때 모델에 대한 CLiPS 사실이 파일 형태로 함께 저장되는데, 프로세스 엔진 내부에 정의된 CLiPS 규칙은 저장된 파일을 입력으로 하여 모델 해석 작업을 시작하게 된다.

2) Modify Code 모델의 입력 산출물 입력

그림 6의 Modify Code 모델을 수행하기 위해서는 모델의 입력 산출물이 존재해야 한다. 프로세스 모델의 시뮬레이션을 위해서 입력 산출물 대화 상자(그림 13)를 통해 해당 산출물을 직접 입력받아 모델이 실행된다. 프로세스 엔진은 입력된 산출물의 이름과 모델 내에 표현된 산출물의 이름이 동일한지를 판단한다. Modify Source Code 스텝에 부착된 연결 연산자는 IN-OR이기 때문에 이 연결 연산자로 향하는 입력 이벤트 흐름 및 산출물 흐름 중 하나라도 만족하면 Modify Source Code 스텝을 시작할 수 있다.

3) Modify Source Code 스텝의 시작

Modify Source Code 스텝을 시작하기 위해서는 스텝의 입출력 조건이 존재해야 한다. 그림 6에서 Modify

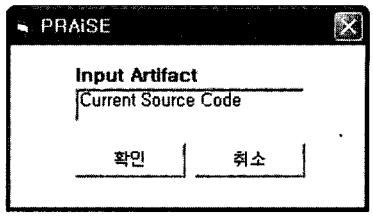

그림 13 입력 산출물 입력 화면

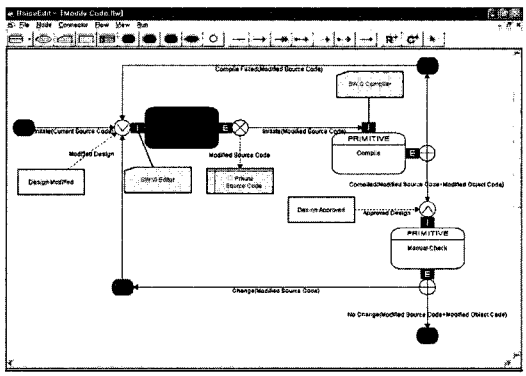

그림 14 스텝 시작

Source Code 스텝의 입력 조건은 산출물 Current Source Code이고, 출력 조건은 산출물 Modified Source Code이다. 또한 스텝 수행을 위한 작업자가 해당 스텝에 할당되어 있는지의 여부도 스텝을 수행시키기 위한 조건이 된다. 이 모든 작업을 프로세스 엔진이 사실과 규칙을 이용해 모델을 해석한 후 앞에서 언급한 모든 조건이 만족되면 Modify Source Code 스텝의 수행을 담당할 작업자의 작업자 관리기에게 스텝을 수행하라는 정보를 보내게 되고 Modify Source Code 스텝의 색이 반전됨으로써 시작되었음을 나타내게 된다(그림 14).

4) Modify Source Code 스텝의 수행

프로세스 엔진에 의해 스텝을 수행하라는 명령이 작업자 관리기로 전달되면 그림 11의 작업자 관리기 화면을 통해 작업 지시 사항을 전달받는다. 작업자가 소스 코드 변경을 완료한 후 End 버튼을 눌러 작업이 완료되었다는 정보를 보내면 해당 스텝은 종료하게 된다.

5) Modify Source Code 스텝의 종료

스텝이 종료되었다는 메시지를 작업자 관리기로부터 전달받으면 프로세스 엔진은 해당 스텝의 상태를 변경시킨 후 다음 스텝을 수행한다. 이 때 Modify Source Code 스텝에 부착된 상태 표시자 "E"가 반전됨으로써 스텝이 종료되었음을 나타낸다(그림 15).

6) Modify Code 모델의 나머지 스텝 수행

Modfiy Code 모델에 표현된 나머지 스텝들 -Compile, Manual Check-들도 3), 4), 5)의 과정을 통해 순차적으

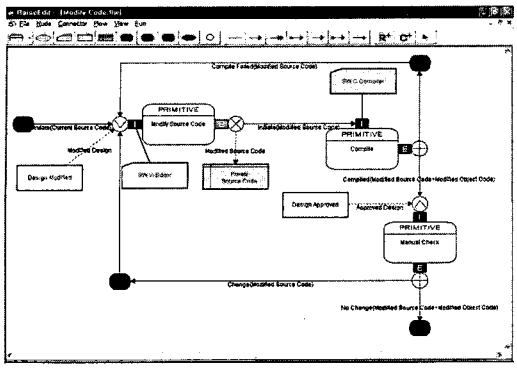

그림 15 스텝 종료

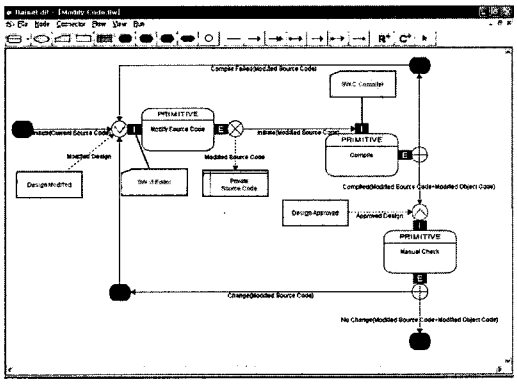

그림 16 프로세스 모델 종료

로 수행된다.

7) Modify Code 모델의 종료

프로세스 모델 내에 표현된 모든 스텝이 정상적으로 수행을 마치게 되면 모델의 종료점에 다다르게 된다. 그림 16은 모델 내에 표현된 세 개의 스텝이 모두 종료하여 종료점에 다다른 모습(종료점을 다른색으로 표시)을 나타내고 있다. 이로써 프로세스 실행이 종료된다.

PRAiSE는 직접 실행하기에 앞서서 여러 가지 가능한 시나리오에 따라 시뮬레이션할 수 있는 기능도 제공한다. 특히, 규모가 큰 소프트웨어 프로세스를 실행시키기 위해서는 많은 인력과 시간 및 예산이 소요되기 때문에 프로세스를 실제로 실행하기 전에 정의된 소프트웨어 프로세스 모델이 작업 환경에 적합한지를 평가하는 프로세스 시뮬레이션은 PSEE가 갖추어야 할 중요한 기능이다.

5. 분석

5.1 규칙 기반 패러다임의 도입 배경

본 연구에서는 소프트웨어 프로세스가 갖는 두 가지 중요한 특징을 반영하기 위해 규칙 기반 패러다임을 이용하여 PRAiSE를 개발하였다.

첫째, 소프트웨어와 마찬가지로 소프트웨어 프로세스는 요구사항 분석, 설계, 구현 및 테스트 등으로 구성된 생명 주기를 가진다[10]. 이와 같은 생명 주기 과정에 걸쳐서 소프트웨어 프로세스는 다양한 이유로 변경이 발생하며 프로세스의 변경은 프로세스 실행 중에 예기치 않게 발생할 수 있다. 예를 들어, 프로세스 모델링 과정에서 특정 산출물의 재검토 과정을 고려하지 못했을 수도 있다. 만약 프로세스가 실행된 이후에 이러한 상황이 발생하게 되면 그 산출물의 재검토 과정을 담당할 사람을 결정하고 그러한 과정을 수행할 수 있도록 프로세스 모델을 변경해야만 한다. 또한, 품질 보증을 담당한 사람이 갑자기 일을 그만 둔 경우 품질 보증 작업과 유사한 일에 익숙한 새로운 사람을 찾아 그 일을 대체할 수 있도록 조치를 취해야만 한다. 이와 같이 소프트웨어 프로세스 정의는 전체 생명 주기에 걸쳐 빈번한 변경이 발생하기 때문에 프로세스가 부분적으로 실행 상태에 있더라도 변경과 관련된 내용이 반드시 반영되어야 한다. 생성 규칙과 현재의 상태 혹은 사실을 대상으로 새로운 사실을 추론해내는 역할을 담당하는 추론 엔진이 분리된 구조를 띄고 있는 생성 시스템의 실행 메커니즘은 소프트웨어 프로세스 정의에 해당하는 상태만을 변경하고 프로세스의 실행을 위한 생성 규칙과 추론 엔진에는 영향을 주지 않음으로써 많은 변경을 내포하는 소프트웨어 프로세스의 정의 및 실행에 유연성을 제공하기에 충분하다.

둘째, 소프트웨어 프로세스의 또 다른 특징은 본질적으로 병렬성을 내포하고 있다는 것이다[1]. 프로세스에 정의된 많은 작업들은 여러 사람으로 구성된 팀을 바탕으로 수행된다. 여러 개발 참여자들이 제품의 검토 작업을 할 때 그 작업을 동시에 수행할 수 있고, 작업을 분할하여 각각의 작업을 특정 개발 참여자에게 수행하도록 함으로써 팀을 이루는 구성원들 간에 상호 협력적으로 작업을 수행할 수도 있을 것이다. 또한, 사람과 개발 도구가 함께 수행해야 하는 작업이 있을 수 있다. 예를 들어, 프로그램 오류 수정 작업의 경우는 사람이 오류에 대한 분석 및 수정 작업을 담당하고 컴파일러가 해당 코드를 컴파일 및 테스트하는 작업으로 구성될 수 있다. 심지어 팀 내의 구성원 한 사람과 관련된 프로세스의 경우에도 병렬성은 존재한다. 프로젝트 참여자 한 명의 경우 여러 가지 작업-새로운 기능의 설계, 오류 수정 작업이나 재검토 프로세스의 참여 등-과 관련될 수 있다. 순차적 흐름 구조인 전통적인 구조적 프로그래밍 기법을 이용하여 이러한 병렬성을 지닌 소프트웨어 프로세스를 자동화하는 시스템을 구축하는 것은 모델링 언어의 의미론이 다양해질수록 기하급수적으로 복잡해진다. 규칙 기반 개념은 병렬적인 구조를 지닌 소프트웨어 프

로세스 모델의 여러 작업들을 곧바로 규칙 기반 언어의 규칙과 사실로 맵핑하는 것이 가능하도록 지원한다.

이상에서 살펴본 바와 같이 규칙 기반 패러다임에 바탕을 둔 프로세스 자동화는 빈번히 발생하는 프로세스의 변경에 유동적으로 대처할 수 있을 뿐만 아니라 프로세스 모델을 규칙 기반 언어의 규칙과 사실을 이용하여 직관적으로 맵핑함으로써 프로세스의 병렬성을 효과적으로 제어할 수 있다.

5.2 기존 규칙 기반 PSEE와의 차별성

2장에서 언급한 규칙 기반 PSEE들과 PRAiSE는 규칙 기반 패러다임을 이용하여 프로세스 모델을 해석하고 실행한다는 점에서는 유사성이 존재하지만 그들과는 상반되는 차이점이 몇 가지 존재한다. 기존의 규칙 기반 패러다임을 도입한 PSEE들과 비교했을 때, PRAiSE만의 두드러진 특징은 다음과 같다.

5.2.1 프로세스 모델링과 프로세스 엔진의 분리

PRAiSE의 두드러진 첫 번째 특징은 프로세스 모델링을 프로세스 엔진과 엄격히 분리했다는 점이다. 기존 시스템들은 프로세스의 표현과 해석 및 실행이 결합된 형태를 띄고 있을 뿐만 아니라 이해하기 어려운 텍스트 형태의 규칙과 사실로 프로세스 모델을 정의한다. 즉, 기존의 연구는 프로세스 모델링에 직접 규칙 기반 언어를 사용해야 한다. 이러한 접근 방법은 쉽게 프로세스 모델을 작성할 수 없으며 프로세스를 이해하기도 매우 어렵다. 반면, PRAiSE에서의 프로세스 표현은 이해하기 쉬운 다이어그램을 이용하며 그 해석 및 실행을 위해 해당 모델이 저장될 때 자동적으로 사실 형태로 변환되게 구현함으로써 프로세스 모델링의 용이성 및 작성된 프로세스 모델에 대한 이해도를 높였다. 따라서, 규칙 기반 패러다임은 PRAiSE 내부의 효율성을 위해서만 사용되며 일반 사용자 입장에서는 규칙 기반 언어를 전혀 몰라도 프로세스 모델을 작성할 수 있다.

5.2.2 프로세스 모델링 언어의 효율성

입도(granularity)

정의된 프로세스는 스텝, 산출물, 자원의 표현에 있어 다양한 수준의 입도 스펙트럼을 지원해야 한다. 그러나, MARVEL, Merlin, GRAPPLE 등의 기존 규칙 기반 PSEE들이 제공하는 프로세스 모델링 언어들은 계층적으로 프로세스를 정의하도록 지원하지만 스텝의 계층은 규칙 베이스로부터 추출되어야 하며 이는 쉬운 일이 아니다. 또한, 프로세스 자체에서 실제 산출물을 직접 접근할 수 없으며 도구의 호출을 통해서만 가능하다[8]. PRAiSE에서 제공하는 프로세스 모델링 언어 RAiSE는 스텝, 산출물, 자원을 다양한 수준에서 명시적으로 표현하도록 지원한다.

추상화 수준

규칙 기반 PSEE들은 완전히 실행 가능한 프로세스 프로그램을 산출하는 것이 주목적이기 때문에 상대적으로 복잡한 문법과 엄격한 의미론을 지원한다. 따라서, 모델 자체가 코드의 모습의 갖기 때문에 추상화 수준이 매우 낮으며 프로세스를 개발자 공동의 참조 모델로 사용하기 어렵다. PRAiSE에서는 다양한 추상화 수준에서 가시적인 그래픽 형태로 지원하면서도 실행 가능한 프로세스 모델을 생성한다.

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 규칙 기반 PSEE는 반응 제어는 잘 지원하는 반면 순방향 제어는 모델 작성자가 정의한 선행 조건 또는 후행 조건에 의해 시뮬레이션할 수 있을 뿐이다. RAiSE에서는 순방향 제어는 이벤트 흐름을 이용하여, 반응 제어는 이벤트 브로드캐스팅 흐름을 이용하여 명시적으로 표현할 수 있다.

5.2.3 추론 엔진의 안정성

규칙 기반 패러다임을 이용한 프로세스의 실행은 수시로 변경이 발생하는 소프트웨어 프로세스에 유연하게 대처할 수 있는 이점이 있다. 그 이유는 프로세스 정의에 해당하는 사실만 변경되고, 실행을 담당하는 추론 엔진에는 전혀 영향을 끼치지 않기 때문이다. 기존의 규칙 기반 PSEE에서는 이러한 추론 엔진을 직접 구현하였다. 이는 상당히 어려운 작업일 뿐만 아니라 그 내부에 예기치 못한 오류를 내포할 수 있다. 이러한 오류들은 잘못된 프로세스 모델의 해석으로 인해 여러 프로젝트 참여자들로 하여금 프로세스에 대한 혼동을 야기할 수 있고 자원 할당의 오류 및 프로세스 스케줄의 지연을 초래할 수도 있다. 특히, 프로세스 모델링 언어가 확장 또는 개선될 때마다 추론 엔진을 수정하거나 최악의 경우에는 새로 개발하여야 한다. PRAiSE에서는 규칙과 사실만을 정의하고 추론을 통한 프로세스 모델의 해석은 여러 다른 개발 분야에서 현재 이용되고 있고, 그 성능이 입증된 전문가 시스템 CLiPS가 담당하도록 분리함으로써 이러한 문제를 해결하였다.

6. 결 론

PSEE에서 요구되는 가장 중요한 두 가지 측면은 제공되는 프로세스 모델링 언어의 특성과 PSEE 구성 요소의 효율성이다. 프로세스 모델링 언어에 따라 시스템

의 사용성, 프로세스 모델의 표현력, 시스템의 여러 분석 및 기타 주요한 기능을 제공할 수 있는 능력이 좌우되며 PSEE의 구성 요소에 따라 프로세스 모델 실행의 정확성, 프로세스 실행의 효율성, 사용의 용이성 등이 크게 영향을 받는다. 그러나, 현재까지의 PSEE는 이 두 가지 요구 사항을 성공적으로 만족시키지 못하고 있다. 본 논문에서는 이러한 1세대 PSEE가 갖는 단점을 극복한 차세대 PSEE인 PRAiSE에 대하여 그 구성 요소들의 설계와 구현을 설명하였고 시뮬레이션 예제를 통해 사용의 용이성과 적용 가능성을 기술하였다. PRAiSE에서 사용된 규칙 기반 패러다임은 정형적으로 프로세스를 정의하고 자동화시킬 수 있으며 소프트웨어 프로세스의 특징인 병렬성을 효율적으로 지원할 수 있다는 장점을 가지며 기존 규칙 기반 PSEE의 단점인 사용의 용이성과 추론 엔진의 안정성 문제를 해결할 수 있다.

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 모델링하여 PRAiSE에 직접 적용해 봄으로써 프로세스 모델링 및 실행 시 발견되는 문제점을 파악하여 지속적으로 시스템을 향상시켜 나가는 작업도 이어져야 할 것이다.

참 고 문 헌

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

[2] Peuschel, B, Schfer, W. and Wolf, S., "Knowledge-based Software Development Environment Supporting Cooperative Work," Journal on Software Engineering and Knowledge Engineering, 1992.

[3] Junkermann, G., Peuschel, B., Schfer W. and Wolf, S., "MERLIN: Supporting Cooperation in Software Development Through a Knowledge-Based Environment," Software Process Modeling and Technology, Research Studies Press Limited, 1994.

[4] Alan, M. C, Software Process Automation, Springer, 1995.

[5] Burkhard, P. and Wilhelm, S., "Concepts and Implementation of a Rule-based Process Engine," Proceedings of 14th International Conference on Software Engineering, 1992.

[6] 이형원, 최상일, "RAiSE: 다중 패러다임을 결합한 프로세스 모델링 언어", 2003년도 정보처리학회춘계학술발표논문집, 10권 1호, No.1, pp. 1665-1668, May 2003.

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

[8] 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.

[9] 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 First International Conference on Software Process, pp. 176-186, 1991.

[10] Osterweil, L., "Software Processes Are Software Too," Proceedings of the 9th International Conference on Software Engineering, pp. 2-13, 1987.

이 형 원

1987년 서울대학교 계산통계학과(학사) 1990년 서울대학교 계산통계학과(석사) 1995년 서울대학교 계산통계학과(박사) 1998년~1999년 University of Massachusetts at Amherst 방문연구원. 1993년~현재 강릉대학교 정보전자공학부 컴퓨터공학전공 교수. 관심 분야는 소프트웨어 프로세스, 형상관리 등

이 승 진

1991년 서울대학교 해양학과(학사). 1996년 서울대학교 전산과학과(석사). 1999년 서울대학교 전산과학과 박사수료. 2000년~2003년 중앙M&S 연구소장. 2003년~현재 성공회대학교 초빙교수. 관심분야는 소프트웨어 공학