2013. 12. 9. [제76호]

워크플로우 기반의 제품라인 소프트웨어 개발 지원 환경

양진석(Jin-Seok Yang), 강교철(Kyo C. Kang)

KIPS transactions on software and data engineering v.2 no.6

소프트웨어공학센터 경영지원TF팀

Contents

정보통신산업진흥원 부설
nipa 소프트웨어공학센터

Ⅰ. 서론

최근 들어 자동차, 의료, 제조 등의 산업분야에서 ICT 기술을 적극적으로 활용한 융합소프트웨어 개발의 중요성이 강조되고 점차 많은 기업들이 세계시장으로 진출을 모색하고 있다. 소프트웨어 개발에 반영해야 하는 휘처(Feature)의 수는 증가하고 그에 따라 필연적으로 따라오는 다양성(Variation)을 소프트웨어에 반영하기 위한 노력을 기울이고 있다.

소프트웨어 제품라인 공학(Software Product Line Engineering)은 소프트웨어를 각각 개별적으로 개발하는 전통적인 소프트웨어 개발 방법의 단점을 보완하여 소프트웨어의 재사용에 따른 고품질과 빠른 시장적시성(Time-to-Market)을 보장하여 앞서 설명한 다양성 문제를 해결하는 방법을 제시하고 있다. 개발자는 개발 대상 소프트웨어의 기능 및 비기능 측면을 고려한 공통점과 차이점 분석(Commonality and Variability Analysis)을 통해서 핵심 자산(Core Asset)을 확보하고 이후 자산들을 조합하여 원하는 소프트웨어(제품)을 개발할 수 있다[1].

지난 몇 년 간 포항공과대학교 융합소프트웨어개발센터(CoSDEC)는 Kyo C. Kang이 제안한 FORM(Feature Oriented Reuse Method)[2]를 기반으로 하는 제품라인공학 기반의 융합소프트웨어 개발 방법을 제안하고, 방법론을 지원하는 개발도구인 벌컨 워크벤치(VULCAN Workbench) 개념을 제안하였다[3].

융합소프트웨어 제품라인 방법론에서는 소프트웨어 개발을 위한 아키텍처 모델(Architecture Model)을 제안하고 어플리케이션의 유형에 따라 명세(Specification) 기반으로 자동화 된 제어컴포넌트 개발(Control Component Development)이 될 수 있도록 제어컴포넌트의 행위(Behavior)를 명세할 수 있는 모두 네 가지의 방법을 제안하고 있다[4]. 그 가운데 워크플로우(Workflow) 명세방법은 트랜잭션을 주로 처리하는 소프트웨어의 제어 행위를 명세하기 위해서 제안되었다.

워크플로우 명세를 기반으로 제품라인 소프트웨어의 제어컴포넌트 개발이 효과적으로 진행되기 위해서는 도구의 지원이 반드시 필요하지만, 기존의 워크플로우 모델링 지원 도구들은 제품라인 공학 개념을 지원하지 않는다[5][6]. 그래서 융합소프트웨어 제품라인 방법론에서 제안하는 워크플로우 기반의 제품라인 소프트웨어 개발 지원도구를 앞서 소개한 벌컨 워크벤치에 포함시켜 개발을 진행하였다.

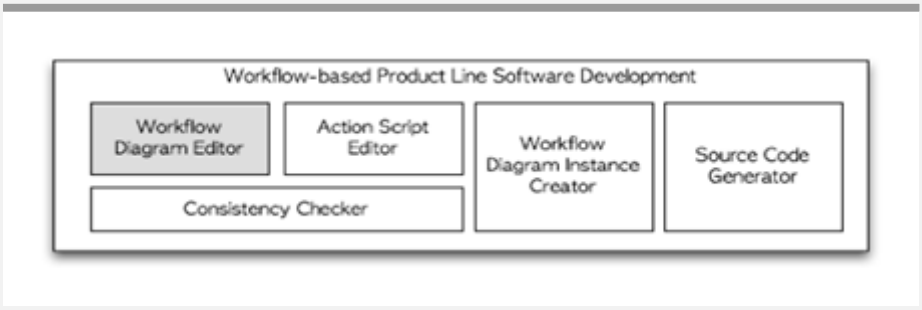
II. 개발 현장의 문제점

휘처 중심의 제품라인 공학 개념이 반영된 최대한으로 단순하면서 확장 가능한 워크플로우 기반 개발도구를 개발을 목표로 하고 그에 따라 결정된 기능은 다음과 같다.

- UML2.0 표기법을 활용한 최소 워크플로우 행위명세 지원
- 제어컴포넌트와 도메인컴포넌트와의 결합을 위한 방법제공
- 휘처 기반 워크플로우 행위 명세의 파라메터화(Parameterization)
- 휘처모델-아키텍처-워크플로우 행위 명세 사이의 일관성 검사
- 워크플로우 행위 명세 기반의 코드 생성

1. 도구의 구성과 도구들 사이의 관계

그림 1_지원도구의 개념적인 아키텍처



목표에 따라서 개발된 지원도구는 [그림 1]과 같이 다섯 개의 모듈로 구분되며 각 모듈의 주요기능은 다음과 같다.

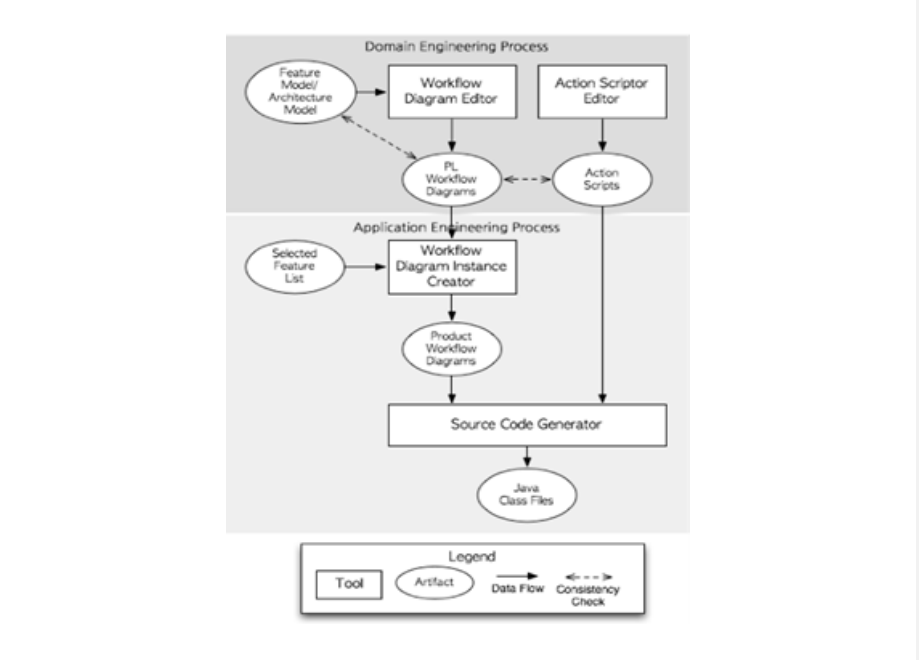
- Workflow Diagram Editor : 워크플로우 다이어그램 모델 편집 환경을 제공
- Action Script Editor : 워크플로우를 구성하는 특정 태스크가 활성화 되었을때 호출되는 특정 도메인 컴포넌트와의 관계를 정의하기 위한 액션 스크립트(Action Script)의 편집환경 제공
- Consistency Checker : 워크플로우 모델에 포함된 휘처정보와 휘처모델 사이의 일관성을 검증하는 기능을 제공

- Workflow Diagram Instance Creator : 휘처 컨피규레이션 정보에 따라서 제품라인 워크플로우 모델에서 제품 워크플로우 모델을 생성하는 기능을 제공
- Source Code Generator : 제품 워크플로우 모델과 액션스크립트를 이용하여 실제 구현을 위해서 워크플로우의 행위 의미(Semantics)를 반영한 소스코드를 자동으로 생성하는 기능을 제공

지원도구를 구성하는 모듈들 가운데 '워크플로우 다이어그램 편집기(Workflow Diagram Editor)'를 제외한 나머지 도구들은 모두 이클립스 IDE 플랫폼 기반의 이클립스 플러그-인(Eclipse Plug-in) 어플리케이션으로 개발되었다. '워크플로우 다이어그램 편집기'의 경우 UML 2.0 표기법을 지원하는 다이어그램의 편집 기능만 제공하면 되었기 때문에 오픈소스 도구인 StarUML 5.0을 이용하였다[7].

소프트웨어 개발자들은 워크플로우 기반의 제품라인 소프트웨어의 제어컴포넌트 개발을 위해 [그림 2]와 같은 흐름으로 지원도구들을 사용할 수 있다.

그림 2_제품라인 공학 기반의 엔지니어링 절차에 따른 도구의 지원



개발자는 제품라인 소프트웨어의 자산(Asset)을 만드는 제품라인 공학의 도메인 엔지니어링(Domain Engineering) 과정에서 대상 소프트웨어의 제어컴포넌트의 행위명세를 작성하기 위해서 '워크플로우 다이어그램 편집기'와 '액션 스크립트 편집기'를 이용할 수 있다. 이 도구들을 이용하여 개발자는 가변성(Variation) 정보가 포함된 제품라인 워크플

로우 다이어그램(Product Line Workflow Diagram)과 특정 액티비티가 활성화 될 때 실행되는 도메인 컴포넌트를 연결하는 액션 스크립트 (Action Script)를 작성할 수 있다.

워크플로우를 이용하여 제어컴포넌트의 행위명세를 작성 할 때 개발자는 워크플로우의 액티비티(Activity)와 흐름(Flow)에 UML 스트레오타입(Stereotype)을 이용하여 휘처이름과 AND, OR연산자를 이용한 간단한 논리식(이하, 휘처로직)을 만듬으로써 가변정보를 포함 시킬 수 있다. 이러한 휘처로직은 어플리케이션 엔지니어링 과정에서 휘처선택의 결과에 따라 참/거짓이 결정되며, 참으로 판정난 요소들만 인스턴스 생성시 사용된다.

'일관성 검사기(Consistency Checker)'를 통해 개발자는 작성된 워크플로우 다이 어그램에 오류가 있는지에 대해서 확인 할 수 있다. 이 도구는 제품라인 워크플로우 다이어그램의 검증과 제품 워크플로우 다이어그램의 검증에 모두 사용된다.

전자의 경우, 도구는 워크플로우 행위명세에 포함된 가변 성 정보에 대한 검증을 진행한다. 이 검증 과정에서 도구는 휘처모델(Feature Model), 아키텍처(Architecture), 그리고 워크플로우 다이어그램을 입력으로 받아, 앞서 설명한 워크플로우에 포함된 휘처로직과 휘처와의 일관성 검사를 진행한다. 휘처모델-아키텍처-워크플로우 행위 명세 사이에 일관성 검사는 현재 아래의 네 가지 기본적인 규칙을 통해서 이뤄진다.

첫째, 모든 휘처는 적어도 한번은 아키텍처 구성 요소 및 워크플로우 행위명세에 연결되어야 한다. 둘째, 아키텍처를 구성하는 모든 구성요소는 휘처에 바인딩(Binding) 되어 있어야 한다. 셋째, 아키텍처와 워크플로우 행위명세에 연결된 휘처들로부터 도출되는 휘처들 사이의 계층적 관계는 휘처모델의 휘처들 사이의 계층적 관계와 동일해야 한다. 넷째, 대체적(Alternative) 관계의 휘처는 같은 요소에 연결될 수 없다.

후자의 경우, 도구는 비즈니스 프로세스 모델에서 검증 대상으로 삼는 문법적 오류(Syntax Error), 구조적 오류(Structural Error), 의미적 오류(Semantic Error)[8]에 대한 검증을 지원한다. 문법적 오류는 정의된 워크플로우 모델의 표기법들에 대한 오류로 워크플로우 다이어그램 편집도구에서 제공하는 사용 제한으로 오류를 검출할 수 있는 반면에 구조적 오류와 의미적 오류는 워크플로우에 내재된 논리상의 오류로 반드시 확인이 필요하다. 하지만 이번에 개발된 도구는 데드락(Deadlock), 결정적인 무한루프(Deterministic Infinite Loop), 동기화 손실(Lack of Synchronization), 종료 불가능 상태(Activity without Termination), 비활성 상태(Activity without Activation)와 같은 구조적 오류[9]에만 초점을 둔다. 이와 같은 구조적 오류를 검증하기 위해서 현재 [9][10]와 같이 다양한 방법이 연구되고 있지만 구조적 오류 검증에 가장 효율적인 방법인 페트리 넷(Petri Net)을 사용한다[11].

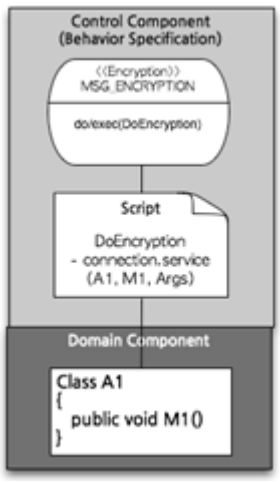
자산으로부터 제품을 만드는 제품라인 공학의 어플리케이션 엔지니어링(Application Engineering) 과정에서 개발자는 '워크플로우 다이어그램 인스턴스 생성기(Workflow Diagram Instance Creator)'를 이용해 휘처 컨피규레이션 과정에서 선택된 휘처 목록과 제품라인 워크플로우 다이어그램을 입력으로 지정하여 제품 워크플로우 다이어그램을 생성할 수 있다. 생성된 제품 워크플로우 다이어그램은 휘처 선택에 따른 가변요소에 대한 파라메터화가 적용된 제품라인 워크플로우 다이어그램의 인스턴스(Instance)이다. 인스턴스의 예는 3장에서 확인 할 수 있다.

2. 액션 스크립트를 이용한 제어컴포넌트와 도메인컴포넌트의 연결

도메인 엔지니어링 과정에 사용되는 도구는 제어컴포넌트의 행위모델을 작성하고, 명세 된 행위모델에서 워크플로우를 구성하는 특정 액티비티(Activity)가 활성화 되었을 때 도메인 컴포넌트 (도메인 컴포넌트는 메뉴얼하게 개발 된다고 가정한다.) 를 사용 또는 적용 할 수 있도록 도와준다. 이를 구현하기 위해서 도구에서는 UML2.0의 요소들 가운데 State Diagram과 State의 액션(Action)을 활용하고, 도메인 컴포넌트와의 결합을 위한 액션 스크립트(Action Script)를 추가로 제공한다. 이들 사이의 관계는 [그림 3]과 같다.

개발자는 State의 do액션속성에 exec(script_ name) 구문을 이용하여 액션 스크립트를 설정할 수 있다. 그리고 액션 스크립트의 connection.service(...) 구문을 이용해 특정 클래스의 메소드를 실행 할 수 있도록 할 수 있다.

그림 3_스크립트를 이용한 제어 컴포넌트와 도메인 컴포넌트 사이의 연결



정의된 액션 스크립트는 크게 내부변수 선언 부분과 로직을 작성할 수 있는 부분으로

구분되어 있으며 로직을 작성시 사용되는 타입과 구문은 자바(Java) 언어의 문법을 따르며 구문안에 @문자로 시작되는 매크로(Macro)를 사용할 수 있다. 정의된 매크로는 '코드생성기'를 통해서 특정 구문으로 치환이 된다. 매크로가 사용된 액션 스크립트의 예는 다음과 같다.

```
//@VAR
String orderID;
Boolean result;
//@BODY
Vector<String> args = new Vector<String>();
orderID=(String)@GETVALUE("orderID");
result = (Boolean)connector.service("UserManager","Validate", args);
```

위의 액션 스크립트는 매크로 "@GETVALUE"를 이용하여 orderID의 값을 반환받는 구문과 connection. service()를 이용하여 도메인 컴포넌트의 구현체인 User Manager 클래스의 Validate 메소드를 호출하는 예제이다.

3. 가변정보의 연결

파라메터화 방법을 적용하여 워크플로우 행위모델의 인스턴스를 생성하기 위해서 다이어그램을 구성하는 State에 휘처를 연결(Binding)할 수 있는 방법으로 앞서 설명한 것과 같이 스트레오타입(Stereotype)을 사용한다. [그림 3]에서 "MSG_ENCRYPTION" State의 스트레오타입에 'Encryption'휘처가 연결되어 있다. 어플리케이션 엔지니어링 과정에서 선택된 휘처집합에 'Encryption'휘처가 없다면 해당 휘처가 연결된 모든 State와 State에 연결된 모든 Sequence Flow는 인스턴스를 생성하는 과정에서 사라지고 삭제과정에서 생성될 수 있는 Dead State(도달 할 수 없는 상태를 의미한다.)까지 사라지게 된다.

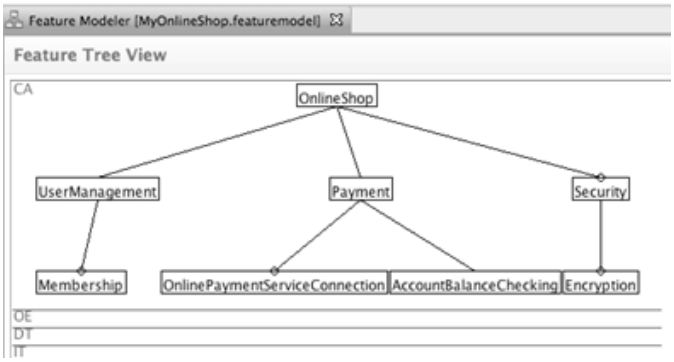
4. 소스코드 자동생성

마지막으로, '코드생성기'는 제품 워크플로우 다이어그램 과 액션 스크립트를 이용해 자바 소스코드를 자동으로 생성 한다. 다이어그램의 구성요소는 대응되는 클래스로 생성되며, 액션 스크립트 역시 전처리과정을 거친 후 하나의 클래스로 생성을 시킨다.

Ⅲ. 창의적인 요구사항 도출을 위한 제안

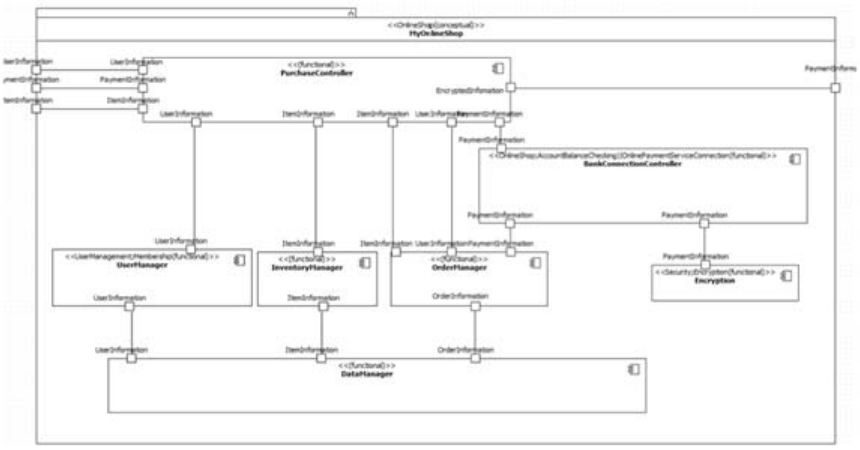
도구의 적용과정을 보여주기 위해서 간단한 물품 구매 소프트웨어를 활용 예제로 사용한다.

그림 4_물품구매 소프트웨어의 휘처모델(부분)



[그림 4]는 제품라인 기반으로 개발되는 물품구매 소프트웨어의 휘처모델을 보여준다. 가변요소는 'Membership', 'Online Payment Service Connection', 그리고 'Security' 및 'Encryption' 휘처로 정의되어 있다. 정의된 네 개의 가변 휘처는 모두 선택적 (Optional) 휘처들이다.

그림 5_물품구매 소프트웨어의 제품라인 아키텍처



FORM-UML 기법[13]을 이용하여 아키텍처 모델링을 진행하면서 도출된 컴포넌트들 가운데 두 개의 제어컴포넌트 "Purchase Controller"와 "Bank Connection Controller"에 대해서 워크플로우 모델을 이용하여 행위명세를 진행하였다. 아래의 [그림 5]는 "Bank Connection Controller"의 행위명세를 보여주며, 내부적으로 'Encryption'과 'Online Payment Service Connection' 선택적 휘처에 대한 가변요소가 포함되어 있다. 두 개의 제어컴포넌트들 제외한 다른 도메인 컴포넌트들은 개발자에 의해서 직접 개발이 이뤄졌다.

제어 컴포넌트의 제품라인 행위명세로부터 휘처 선택을 통해서 두 개의 제품을 생성할 수 있다. 아래의 [그림 6]은 제품라인 행위명세를, [그림 7]과 [그림 8]은 휘처 선택을 통해 자동으로 생성된 각 제품의 행위명세 인스턴스를 보여준다.

생성된 제품 워크플로우 다이어그램과인 액션스크립트를 이용해서 자바소스코드를 생성하고 간단한 시뮬레이션 이용해 생성된 제품 소프트웨어가 작성된 행위명세와 같이 동작하는지 생성된 소스코드에 포함된 디버깅 정보를 이용하여 시험하고 생성된 두 제품 모두 행위명세에 따라 제어가 되는 것을 확인 할 수 있었다.

그림 6_제품라인 Bank Connection Controller 컴포넌트의 행위명세

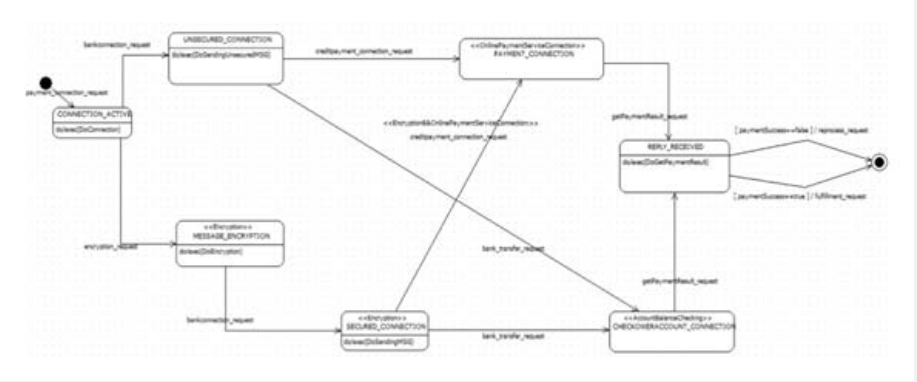


그림 7_Bank Connection Controller 컴포넌트 행위명세의 인스턴스 예 1 (Encryption 휘처를 선택하지 않은 경우)

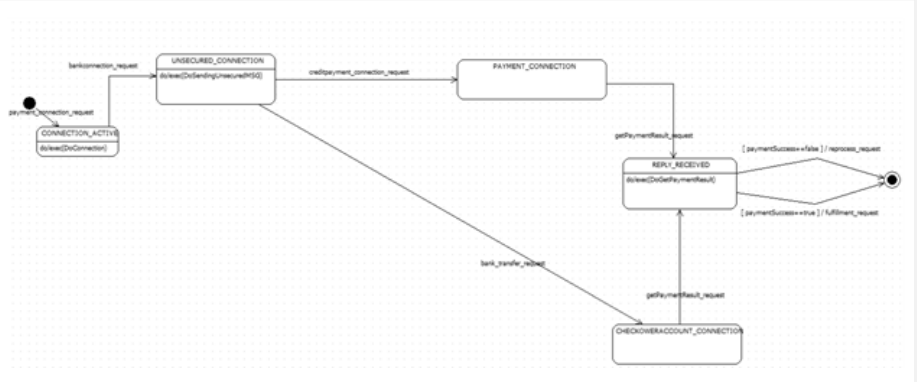
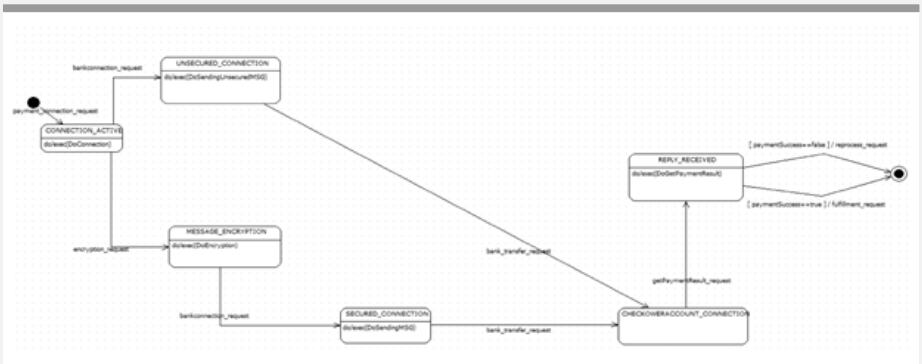


그림 8_Bank Connection Controller 컴포넌트 행위명세의 인스턴스 예 2
(Online Payment Service Connection 취처를 선택하지 않은 경우)



생성된 제품 워크플로우 다이어그램과인 액션스크립트를 이용해서 자바소스코드를 생성하고 간단한 시뮬레이션 이용해 생성된 제품 소프트웨어가 작성된 행위명세와 같이 동작하는지 생성된 소스코드에 포함된 디버깅 정보를 이용하여 시험하고 생성된 두 제품 모두 행위명세에 따라 제어가 되는 것을 확인 할 수 있었다.

Ⅳ. 결론

지금까지 워크플로우 기반의 제품라인 소프트웨어 개발을 지원할 수 있는 도구에 대해서 소개했다. 소개된 도구를 이용해 기존 워크플로우 모델링 도구에서 부족한 제품라인 개념을 적용하여 소프트웨어를 개발할 수 있는 환경을 제공 할 수 있었다.

하지만 아직 워크플로우 모델링을 위한 최소한의 요소만 제공하기 때문에 추가요소를 반영할 필요가 있으며, 추가 요소 반영에 따른 검증기 개선 및 검증기의 정확성 향상을 위해 잘 알려진 오류 예제에 대한 시험이 진행되어야 할 것이다. 이러한 부족한 부분에 대해서는 도구를 실질적인 소프트웨어 개발에 적용하는 과정을 통해 기능을 개선 보완할 예정이다.

* 본 원고는 해당 논문의 요약본임에 따라 자세한 내용은 논문원문을 참조하시기 바랍니다.

참고 자료

1. P.Clemens, L.Northrop, Software product lines: practices and patterns, Addison-Wesly Professional, Aug., 2001.
2. K.Kang, S.Kim, J.Lee, K.Kim, E.Shin, and M.Huh, "FORM: a feature-oriented reuse method with domain-specific reference architecture," Annals of Software Engineering, Vol.5, pp.143-168, May, 1998.
3. Hyesun Lee, Jin-Seok Yang, and Kyo C.Kang, "VULCAN: Architecture-Model-Based Software Development Work bench," The Joint 10th Working IEEE/IFIP Conference on Software Architecture and 6th European Conference on Software Architecture, Aug., 20-24, 2012.
4. Hyesun Lee, Jin-Seok Yang, and Kyo C. Kang, "VULCAN : Architecture-Model-Based Workbench for Product Line Engineering," The 16th International Software Product Line Conference, Sep., 02-07, 2012.
5. www.omg.org, "Business Process Model and Notation", Jan., 2011.
6. Visual Paradigm, "Business Process Visual ARCHITECT (BP-VA)"
7. staruml.sourceforge.net/ko, "StarUML 5.0-OpenSource UML/MDA Platform"
8. Gun-Woo Kim, Jeong-Wha Lee, Jin Hyun Son, "Design Anomalies in the Business Process Modeling", Journal of KIISE, Vol.14, No.9, pp.850-863, 2008. 12.
9. H.Bi, J.Zhao, "Applying propositional logic to workflow verification," Information Technology and Management, 5(3-4), pp.293-318, 2004.
10. Eshuis, H., Semantics and Verification of UML Activity Diagrams for Workflow Modeling, PhD thesis, Universty of Twente. CTIT Ph.D.-thesis series No.02-04.
12. W.M.P. van der Aalst, "Workflow verification: Finding control-flow errors using Petri-net-based techniques", Business Process Management, pp.161-183, 2000.
13. Kwan Woo Lee, "Managing and Modeling Variability of UML Based FORM Architectures Through Feature-Architecture Mapping," KIPS Transactions, Vol.19-D, No.1, pp.81-94, 2012. 2.