JESS 규칙 기반 시스템을 이용한 특성 구성 검증☆

Feature Configuration Verification Using JESS Rule-based System

최 승 훈*
Seung-Hoon Choi

요 약

특성 모델(Feature Model)은 소프트웨어 제품 라인 개발 시 도메인 공학 단계에서 제품들 사이의 공통된 개념들과 서로 다른 개념들을 모델링하는데 널리 사용된다. 특성 모델로부터 특정 제품에 포함될 특성들을 선택한 결과를 특성 구성(Feature Configuration)이라고 하며, 이것은 특정 제품에 대한 요구 사항을 나타낸다. 현재 소프트웨어 제품 라인 개발 시 특성 모델과 특성 구성을 어떻게 구축하고 이용하는지에 대한 연구는 많이 되어 있지만, 이들에 대한 정형적 시맨틱과 논리적 추론에 대한 연구는 부족하다. 본 논문에서는 소프트웨어 제품 라인 공학에서의 표준 문제로 제안된 Graph Product Line을 예제로 하여 규칙 기반 시스템인 JESS를 이용한 특성 구성 검증 기법을 제안한다. 본 논문의 기법은 특성 구성의 불일치성을 일으키는 원인을 명확히 제시하는 장점을 가지며, 자바 언어와의 결합성이 뛰어난 JESS 시스템에 기반을 두었기 때문에 다른 소프트웨어 제품 라인 개발 환경과 쉽게 통합될 수 있다.

Abstract

Feature models are widely used in domain engineering phase of software product lines development to model the common and variable concepts among products. From the feature model, the feature configurations are generated by selecting the features to be included in target product. The feature configuration represents the requirements for the specific product to be implemented. Although there are a lot of researches on how to build and use the feature models and feature configurations, the researches on the formal semantics and reasoning of them are rather inactive. This paper proposes the feature configuration verification approach based on JESS, java-based rule-base system. The Graph Product Line, a standard problem for evaluating the software product line technologies, is used throughout the paper to illustrate this approach. The approach in this paper has advantage of presenting the exact reason causing inconsistency in the feature configuration. In addition, this approach should be easily applied into other software product lines development environments because JESS system can be easily integrated with Java language.

☞ keywords: Software Product Lines; Feature Modeling; Feature Configuration Verification; Rule-based System, 소프트웨어 프러덕트 라인; 특성 모델링; 특성 구성 검증; 규칙 기반 시스템

1. 서론 및 연구 배경

소프트웨어 제품 라인이란, 공통된 핵심 소프트웨어 자산(core software asset)들로부터 특정 목표나 시장을 위해서 개발된 소프트웨어 집약 시스템들의 집합을 의미한다[1]. 소프트웨어 제품 라인의 목적은, 특정 도메인에서 주로 사용되는 공통된 핵심 자산들을 초기 단계에 먼저 개발한 후, 소프트웨어 생산 시 정해진 방식에 의해 이들을 조립함으로써, 비슷한 특성을 가지고 있지만 특정 부분이 다른 일련의 다양한 소프트웨어들을 보다 빠르고 좋은 품질을 갖도록 생산하는 데에 있다. 즉, 소프트웨어 개발 단계 초기에 소프트웨어 패밀리에 속하는 멤버들 사이의 차이점과 공통점을 미리 예측하고 분석함으로써 보다 전략적인 재사용이 가능하도록 하여 소프트웨어 개발 생산성을 향상시키고자 하는 것이다.

이러한 소프트웨어 제품 라인을 구축하는데 있

* 종신회원 : 덕성여자대학교 정보공학대학 컴퓨터공학부
csh@duksung.ac.kr
[2007/08/14 투고 - 2007/08/16 심사 - 2007/08/29 심사완료]
☆ 본 연구는 덕성여자대학교 2006년도 교내연구비 지원에 의해 수행되었음

어서 먼저 도메인 공학(Domain Engineering) 단계를 거치게 된다. 이 단계에서는 도메인 분석 과정을 통해서 특정 도메인에 존재하는 재사용 가능한 자산들을 식별한다. 특성 모델(Feature Model)은 소프트웨어 제품 라인 개발 시 도메인 공학 단계에서 가장 널리 사용되는 모델로서, 특정 도메인에서 제품들 사이의 공통된 개념들과 서로 다른 개념들을 모델링하는데 사용된다. 즉, 개발 초기에 소프트웨어 제품 라인에 존재하는 멤버들 사이의 공통점과 차이점을 명확하게 식별하여 제품들 사이의 가변성을 지원하고자 하는 것이다.

어플리케이션 공학(Application Engineering) 단계에서는 도메인 공학 단계에서 구축된 재사용 가능한 자산을 이용하여 구체적인 제품을 생산한다. 구체적인 제품 생산 시 먼저 특성 모델에 표현되어 있는 가변적 특성들 중에서 어떤 특성을 목표로 하는 제품에 포함시킬 것인가에 대한 요구 사항을 결정해야 한다. 특성 모델에 표현되어 있는 가변적 특성에 대해 특정 제품에 포함될 특성들을 선택한 결과를 특성 구성(Feature Configuration)이라고 한다.

FODA(Feature-Oriented Domain Analysis)[2], FORM(Feature-Oriented Reuse Method)[3], FeatuRSEB[4] 등 특성 기반의 소프트웨어 제품 라인 방법론은 많이 제안되었다. 그러나 특성 및 특성 모델에 대한 정형적 시맨틱(formal semantics)을 정의하고 특성 구성의 정확성(correctness)을 검증하는 방법에 대한 연구는 상대적으로 부족한 상황이다.

가변적 특성의 종류가 많아지고 제한 조건이 복잡해지면 수작업으로 특성 구성의 정확성을 검증하기는 쉽지 않고 이를 위한 자동화 도구가 필수적이다. 본 논문에서는 JESS 규칙 기반 시스템을 이용한 특성 구성 검증 기법을 제안한다. 이 기법은, 특성 모델에 표현되어 있는 특성들에 대한 정보 및 특성들 사이의 제한 조건을 JESS 언어로 표현하고 JESS가 제공하는 규칙 엔진을 이용하여 특성 구성의 불일치성(inconsistency)을 검증한다.

본 논문의 구성은 다음과 같다. 제 2 장에서 특성 모델 및 특성 구성과 JESS 시스템에 대해서 간략히 살펴본다. 제 3 장에서 특성 구성을 검증하기 위한 JESS 프로그램을 Graph Product Line 예제 특성 모델을 통해서 살펴본다. 제 4 장에서는 특성 구성 검증 기법을 실행한 결과와 본 논문에서 제안한 기법의 장점을 살펴보고, 제 5 장에서 결론 및 향후 연구 과제를 기술한다.

2. 관련 연구

2.1 특성 모델과 특성 구성

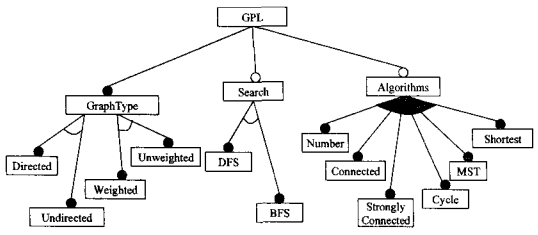
도메인 분석의 주요 결과물인 특성 모델(feature model)은 제품군에 속하는 멤버들 사이의 공통점과 차이점에 대한 고수준의 추상화를 제공하는 요구사항 모델이다. 여기에서 '특성'은, 각 소프트웨어 제품들에서 구현되고 테스트되고 배포, 유지되어야 하는 기능적 추상화를 뜻하는 것으로 고객이 중요하게 생각하는 기능이나 요구 사항을 의미한다.

특성 모델은 특성들 사이의 계층 구조를 나타내는 특성 다이어그램과 제한 조건 등을 나타내는 부가적인 정보로 구성된다. 특성 다이어그램에서 제품군에 속하는 멤버들이 공유하는 공통점은 필수적 특성(mandatory feature)으로 표현되며, 멤버들 간의 차이점은 선택적(optional) 특성과 택일적(alternative) 특성, OR 특성 등으로 표현된다.

필수적 특성은 어플리케이션에 반드시 포함되어야 하는 특성을 의미하고, 택일적 특성은 여러 개 중에서 하나 만이 선택되는 특성을 의미한다. 선택적 특성은 특정 제품에 포함되거나 또는 포함되지 않거나 둘 중의 하나로 선택되는 특성을 의미한다. OR 특성은 여러 개 중에서 적어도 하나가 선택되어야 하는 특성을 의미한다. 이러한 특성 모델은 응용 프로그램 개발에 있어서 '결정 공간(decision space)'을 정의한다[3].

그림 1은 소프트웨어 제품 라인 기술들에 대한 평가를 위해 [5]에서 제안한 표준 문제인 Graph Product Line(GPL) 특성 모델을 나타내며 본 논문에서는 이 특성 모델을 예제로 하여 특성 구성 검증을 위한 프로그램을 기술한다. 그림 1에서 까만 원은 필수적 특성, 빈 원은 선택적 특성을 의미한다. 빈 원호로 연결된 링크는 택일적 특성 그룹을 의미하며 까만 원호로 연결된 링크는 OR 특성 그룹을 위미한다.

그림 1의 GPL 특성 모델의 최상위 노드인 "GPL"은 개념(Concept)이라고 하며 목표로 하는 제품을 의미한다. "GraphType"는 필수적 특성으로서 GPL 생산 시 반드시 포함되어야 하는 특성이다. "Search"나 "Algorithms" 특성은 선택적 특성으로서 제품에 따라 포함될 수도 있고 포함되지 않을 수도 있는 특성들이다. "GraphType" 특성 하위에는 두 개의 택일적 특성 그룹이 존재한다. 하나의 택일적 특성 그룹에 속하는 "Directed"와 "Undirected" 특성은 택일적 특성으로서 두 특성 중 반드시 하나만 특정 제품 생산 시 포함되어야 한다. "Algorithms" 특성 하위에는 여러 특성들이 OR 특성 그룹을 형성하는데 이들 중 적어도 하나 이상의 특성이 제품 생산에 포함되어야 함을 의미한다.



(그림 1) Graph Product Line의 특성 모델

특성 구성이란 특성 모델로부터 특정 제품 개발자가 선택한 특성 선택 결과를 의미한다[6]. 즉, 특성 모델에 표현되어 있는 제품 라인 멤버들 사이의 차이점들 중에서 생성하고자 하는 특정 제품이 포함해야할 특성들에 대한 요구 사항을 의미한다.

특성 모델에 포함된 특성들 사이에는 다음과 같이 두 가지의 추가적인 관계 또는 제한조건이 존재하며, 특성 구성은 이 제한 조건에 맞게 생성되어야 한다.

- *requires*(필요로 한다): 특성 구성에 어떤 특성이 포함된다면 다른 어떤 특성도 반드시 포함되어야 한다
- *excludes*(배제한다): 특성 구성에 어떤 특성이 포함된다면 다른 어떤 특성은 반드시 포함되어서는 안 된다.

[5]에 제시된 GPL의 특성들 사이에 존재하는 제한 조건은 표 1과 같다. 이 표에 따르면, Algorithms 특성의 하위 특성 중에서 StronglyConnected 특성이 선택되면, Directed 특성, Weighted 특성, Unweighted 특성, DFS 특성이 반드시 특성 구성에 포함되어야 한다.

(표 1) GPL에서의 특성들 사이의 제한 조건

Algorithms	Required Graph Type	Required Weight	Required Search
Vertex Numbering	Directed, Undirected	Weighted, Unweighted	BFS, DFS
Connected Components	Undirected	Weighted, Unweighted	BFS, DFS
Strongly Connected Components	Directed	Weighted, Unweighted	DFS
Cycle Checking	Directed, Undirected	Weighted, Unweighted	DFS
Minimum Spanning Tree	Undirected	Weighted	None
Single-Source Shortest Path	Directed	Weighted	None

본 논문에서는 특성 구성이 특성 모델에 표현되어 있는 여러 가지 제한 조건을 만족하는지를 검증하기 위해 JESS 규칙 기반 시스템을 이용하는 방법을 제안한다.

2.2 JESS 규칙 기반 시스템

JESS[7]은 Java Expert System Shell의 약자로써 Sandia 국립 연구소의 Ernest Friedman-Hill에 의해 자바 언어로 개발된 규칙 엔진이자 지식 기반 시스템 개발 환경이다. JESS 시스템을 사용하면 선언적인 규칙 형태로 제공되는 지식을 이용하여 새로운 사실을 추론하는 능력을 갖는 프로그램 개발이 가능하다. 또한, 자바 언어와 쉽게 결합 가능하여 JESS 언어에서 Java의 모든 API를 호출할 수 있을 뿐만 아니라 자바 언어에서 JESS 언어를 이용하여 논리적 프로그램을 작성할 수도 있다.

JESS 시스템의 구조는 그림 2와 같다[8]. 규칙이 적용될 대상이 되는 데이터는 fact라고 하는 기본 원소로 구성되며 working memory 또는 fact base에 존재한다. 본 논문에서의 특성 모델과 특성 구성 결과에 대한 정보는 이 working memory에 존재한다. Rule base에는 지식을 나타내는 규칙의 집합이 존재하며, if-then 형태로 정의된다. 이러한 규칙은 working memory에 존재하는 fact들에 대해서 if 절이 만족하였을 때 then에 정의된 행동을 실행한다. 본 논문에서는 특성 구성이 만족해야 할 제한 조건이 rule로 표현된다.

Working memory에 대해서 if 절이 만족되어 실행(fire)되어야 할 규칙이 존재하는지를 찾는 모듈이 Pattern Matcher이며 찾아진 규칙을 저장하고 실행할 순서를 결정하는 모듈이 Agenda이다. Execution Engine은 agenda에 존재하는 규칙들의 action 부분을 실제로 실행시키는 일을 한다.

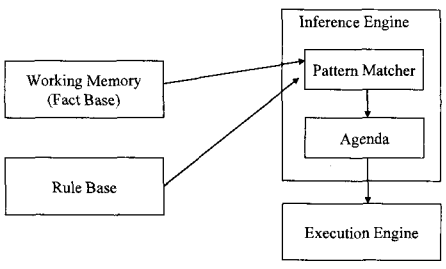


그림 2 JESS 시스템의 전체 구조

3. 특성 구성 검증을 위한 JESS 프로그램

본 절에서는 특성 구성을 검증하기 위한 JESS 프로그램을 기술한다. 2.1절에서 제시한 Graph Product Line(GPL) 특성 모델(그림 1 참조)을 예제로 하여 JESS 프로그램 개발 과정을 설명한다.

3.1 특성 모델 및 특성 구성을 위한 템플릿 정의

먼저 특성 모델과 특성 구성을 표현하는데 사용될 fact들의 구조를 정의해야 한다. JESS에서는 (deftemplate)이라는 명령어를 이용하여 fact의 구조를 정의한다. 그림 3은 특성 모델에 존재하는 각 특성의 정보를 표현하기 위한 템플릿 "feature"와 특성 구성에 포함된 특성의 이름을 가지는 템플릿 "selectedFeature"을 정의하는 코드 부분이다. feature 템플릿이 가지고 있는 슬롯의 의미는 표 1과 같다.

```
; Meta model for Feature Model
(deftemplate feature ;; 특성 모델에서의 특성
        (slot name) ;; the name of this feature
        (slot type) ;; can be mandatory, optional,
alternative, or
        (slot parent) ;; the name of parent feature
        (multislot sameGroupMembers)
        (multislot requires) ;; the names of required
        features
        (multislot excludes))    ;; the names of
        exclued features

;; 특성 구성에 포함된 특성을 나타내기 위한 템플릿 정의
(deftemplate selectedFeature
        (slot name))
```

(그림 3) 특성 모델 및 특성 구성을 위한 fact 구조 정의

(표 2) "feature" 템플릿의 각 슬롯의 의미

슬롯 이름	슬롯이 저장하는 정보
name	이 특성의 이름
type	이 특성의 타입으로서, mandatory, optional, alternative, or 중 하나의 값을 가진다.
parent	이 특성의 부모 특성의 이름

sameGroup Members	이 특성의 type 슬롯의 값이 mandatory이거나 optional인 경우에는 nil 값을 가진다. 이 특성의 type 슬롯의 값이 alternative 또는 or인 경우에는, 같은 그룹에 속하는 특성들의 이름 리스트를 가진다.
requires	이 특성과 requires 관계의 있는 특성들의 이름 리스트
excludes	이 특성과 excludes 관계의 있는 특성들의 이름 리스트

3.2 특성 구성 검증을 위한 규칙 정의

특성 구성에서 검증해야 할 제한 조건은 다음과 같으며, JESS 프로그램은 이러한 제한 조건들을 만족하는지 검사하는 규칙을 제공한다.

1) 필수적 특성에 대한 제한 조건: 특성 모델에서 필수적 특성은 반드시 특성 구성에 포함되어야 한다.

2) 택일적 특성에 대한 제한 조건:
   - 특성 모델에서 택일적 특성 그룹에 속하는 특성들은 그 그룹 중에서 반드시 하나만이 특성 구성에 포함되어야 한다.
   - 또한, 택일적 특성 그룹 중에서 적어도 하나의 특성은 특성 구성에 포함되어야 한다.

3) OR 특성에 대한 제한 조건: 특성 모델에서 OR 특성 그룹을 이루는 특성들 중 적어도 하나의 특성은 특성 구성에 포함되어야 한다.

4) requires 관계에 대한 제한 조건: 특성 구성에 포함된 각 특성에 대해서, 특성 모델에서 이 특성과 requires 관계에 있는 모든 특성들은 특성 구성에 포함되어야 한다.

5) excludes 관계에 대한 제한 조건: 특성 구성에 포함된 각 특성에 대해서, 특성 모델에서 이 특성과 excludes 관계에 있는 모든 특성들은 특성 구성에서 제외되어야 한다.

• 필수적 특성을 위한 규칙

그림 4는, 필수적 특성이 모두 특성 구성에 포함되었는지를 검사하기 위한 규칙을 보여주는 코드 부분이다. 필수적 특성이면서 특성 구성에 포함되지 않았으면 오류 메시지를 화면에 출력한다.

```
(defrule checktMandatoryFeatures
        "Mandatory 특성들을 찾아서 이 특성이 특성 구성에 포함되었는지를 검사하는 규칙"
        (feature (name ?name) (type mandatory) (parent ?parent) (sameGroupMembers $?sameGroupMembers) (requires $?requires) (excludes $?excludes))
        (not (selectedFeature (name ?name)))
        =>
        (printout t "Consistency Error[Mandatory]: " ?name " which is mandatory should be selected to feature configuration." crlf)
        (modify ?*finalResult* (result inconsistent))
)
```

(그림 4) 필수적 특성을 위한 규칙

• 택일적 특성을 위한 규칙

그림 5와 그림 6은, 택일적 특성에 대한 검사를 수행하는 규칙을 정의하는 코드이다. 그림 5는, 택일적 특성 그룹에서 하나의 특성만이 특성 구성에 포함되었는지를 검사하는 규칙으로서, 같은 택일적 특성 그룹에 속하는 특성들 중에서 특성 구성에 동시에 존재하는 경우 오류 메시지를 화면에 출력한다.

```
(defrule checkAlternativeFeatures
        "Alternative 특성들을 찾아서 이 중에 하나만 특성 구성에 선택되었는지를 검사하는 규칙"
        (selectedFeature (name ?name1))
        (feature (name ?name1) (type alternative) (parent ?parent) (sameGroupMembers $?sameGroupMembers1) (requires $?requires1) (excludes $?excludes1))
        (selectedFeature (name ?name2&~?name1))
        (feature (name ?name2&:(member$ ?name2 $?sameGroupMembers1)) (type alternative) (parent ?parent)(sameGroupMembers $?sameGroupMembers2) (requires $?requires2) (excludes $?excludes2))
        =>
        (printout t "Consistency Error[Alternative]: Both of " ?name1 " and " ?name2 " which are alternative features were selected to feature configuration." crlf)
        (modify ?*finalResult* (result inconsistent))
)
```

(그림 5) 택일적 특성을 위한 규칙 I

그림 6은, 택일적 특성 그룹에서 어떠한 특성도 선택되지 않은 경우를 검사하는 규칙으로서, 특성 구성에 선택되지 않은 각 택일적 특성에 대해 같은 그룹에 속하는 어떠한 택일적 특성도 특성 구성에 선택되지 않은 경우에 오류 메시지를 출력한다.

```
(defrule checkAlternativeFeatures2
        "alternative 특성 그룹을 찾아서 이 중에 적어도 하나의 특성이 특성 구성에 선택되었는지를 검사하는 규칙"
        (feature (name ?name1) (type alternative) (parent ?parent) (sameGroupMembers $?sameGroupMembers1) (requires $?requires1) (excludes $?excludes1))
        (not (selectedFeature (name ?name1)))
        (feature (name ?name2&:(member$ ?name2 $?sameGroupMembers1)) (type alternative) (parent ?parent)(sameGroupMembers $?sameGroupMembers2) (requires $?requires2) (excludes $?excludes2))
        (not (selectedFeature (name ?name2&~?name1)))
        =>
        (assert (finalAlternativeResult (result inconsistent) (parent ?parent)))

)
```

(그림 6) 택일적 특성을 위한 규칙 II

• OR 특성들을 위한 규칙

그림 7은, OR 특성 그룹에서 어떠한 특성도 선택되지 않은 경우를 검사하는 규칙으로서, 특성 구성에 선택되지 않은 각 OR 특성에 대해 같은 그룹에 속하는 어떠한 OR 특성도 특성 구성에 선택되지 않은 경우에 오류 메시지를 출력한다.

```
(defrule checkOrFeatures
        "Or 특성들을 찾아서 이 중에 적어도 하나의 특성이 특성 구성에 선택되었는지를 검사하는 규칙"
        (feature (name ?name1) (type or) (parent ?parent) (sameGroupMembers $?sameGroupMembers1) (requires $?requires1) (excludes $?excludes1))
        (not (selectedFeature (name ?name1)))
        (feature (name ?name2&:(member$ ?name2 $?sameGroupMembers1)) (type or) (parent ?parent)(sameGroupMembers $?sameGroupMembers2) (requires $?requires2) (excludes $?excludes2))
        (not (selectedFeature (name ?name2&~?name1)))
        =>
        ;(printout t "Consistency Error[OR]: At least one of OR features under " ?parent " should be selected to feature configuration." crlf)
        (assert (finalOrResult (result inconsistent) (parent ?parent)))
)
```

(그림 7) 택일적 특성을 위한 규칙

• requires 관계를 위한 규칙

그림 8은, 특성 구성이 requires 관계를 만족하도록 생성되었는지를 검사하는 규칙을 정의하는 코드 부분이다. 특성 구성에 선택된 각 특성에 대해서 requires 관계에 있는 특성이 선택이 되지 않은 경우 오류 메시지를 화면에 출력한다.

```
(defrule checktRequiredFeatures
        "각 특성의 required 특성들을 찾아서 이 특성이 특성 구성에 포함되었는지를 검사하는 규칙"
        (selectedFeature (name ?name1))
        (feature (name ?name1) (type ?type1) (parent ?parent1)(sameGroupMembers $?sameGroupMembers1) (requires $?requires) (excludes $?excludes1))
        (feature (name ?r&:(member$ ?r $?requires)) (type ?type2) (parent ?parent2)(sameGroupMembers $?sameGroupMembers2) (requires $?requires2) (excludes $?excludes2))
        (not (selectedFeature (name ?r)))
        =>
        (printout t "Consistency Error[Required]: " ?r" which is required feature of " ?name1 " should be selected to feature configuration." crlf)
        (modify ?*finalResult* (result inconsistent))
)
```

(그림 8) Requires 관계를 위한 규칙

• excludes 관계를 위한 규칙

그림 9은, 특성 구성이 excludes 관계를 만족하도록 생성되었는지를 검사하는 규칙을 정의하는 코드이다. 특성 구성에 선택된 하나의 특성에 대해서 excludes 관계에 있는 특성이 선택된 경우 오류 메시지를 화면에 출력한다.

```
(defrule checktExclusiveFeatures
        "각 특성의 required 특성들을 찾아서 이 특성이 특성 구성에 포함되었는지를 검사하는 규칙"
        (selectedFeature (name ?name1))
        (feature (name ?name1) (type ?type1) (parent ?parent1) (sameGroupMembers $?sameGroupMembers1) (requires $?requires1) (excludes $?excludes))
        (feature (name ?e&:(member$ ?e $?excludes)) (type ?type2) (parent ?parent2) (sameGroupMembers $?sameGroupMembers2) (requires $?requires2) (excludes $?excludes2))
        (selectedFeature (name ?e))
        =>
        (printout t "Consistency Error[Exclusive]: " ?e " which is exclusive feature of " ?name1 " should not be selected to feature configuration." crlf)
        (modify ?*finalResult* (result inconsistent))
)
```

(그림 9) excludes 관계를 위한 규칙

4. 특성 구성 검증 결과 및 평가

4.1 특성 모델 및 특성 구성을 위한 Fact 작성

특성 구성을 검증하기 위해서는 먼저 특성 모델에 대한 fact를 선언(assert)해야 한다. 3.1 절에서 정의된 특성을 위한 템플릿 정의에 따라 특성 모델에 존재하는 모든 특성에 대한 fact를 선언한다. 그림 10은 그림 1에서 보인 GPL 특성 모델에 대한 fact를 선언하는 코드를 보여준다.

다음 단계로 검증하고자 하는 특성 구성에 대한 fact를 선언한다. 특성 구성은 생산하고자 하는 제품의 특성에 따라 다양하며, 그림 9는, 다음과 같은 여러 가지 제한 조건을 위배한 특성 구성의 한 예를 나타내는 코드이다.

- StronglyConnected 특성과 requires 관계에 있는 DFS 특성과 Unweighted 특성이 특성 구성에 포함되어 있지 않다.
- Algorithms 특성 하위의 OR 특성들 중에서 하나의 특성도 특성 구성에 포함되어 있지 않다.
- 같은 택일적 특성 그룹에 속하는 Undirected 특성과 Directed 특성이 동시에 특성 구성에 포함되어 있다.

```
;; GPL
(assert (feature (name GPL) (type mandatory) (parent nil) (sameGroupMembers nil ) (requires nil) (excludes nil)))

;; GraphType + 2 Alternative Groups
(assert (feature (name GraphType) (type mandatory) (parent GPL) (sameGroupMembers nil ) (requires nil) (excludes nil)))
(assert (feature (name Directed) (type alternative) (parent GraphType) (sameGroupMembers Undirected) (requires nil) (excludes nil)))
(assert (feature (name Undirected) (type alternative) (parent GraphType) (sameGroupMembers Directed) (requires nil) (excludes nil)))
(assert (feature (name Weighted) (type alternative) (parent GraphType) (sameGroupMembers Unweighted) (requires nil) (excludes nil)))
(assert (feature (name Unweighted) (type alternative) (parent GraphType) (sameGroupMembers Weighted) (requires nil) (excludes nil)))

;; Search + Alternative Group
(assert (feature (name Search) (type optional) (parent GPL) (sameGroupMembers nil ) (requires nil) (excludes nil)))
(assert (feature (name DFS) (type alternative) (parent Search) (sameGroupMembers BFS) (requires nil) (excludes nil)))
(assert (feature (name BFS) (type alternative) (parent Search) (sameGroupMembers DFS) (requires nil) (excludes nil)))

;; Algorithms + OR Group
(assert (feature (name Algorithms) (type mandatory) (parent GPL) (sameGroupMembers nil ) (requires nil) (excludes nil)))
(assert (feature (name Number) (type or) (parent Algorithms) (sameGroupMembers Connected StronglyConnected Cycle MST Shortest) (requires DFS BFS Directed Undirected Weighted Unweighted) (excludes nil)))
(assert (feature (name Connected) (type or) (parent Algorithms) (sameGroupMembers Number StronglyConnected Cycle MST Shortest) (requires DFS BFS Undirected Weighted Unweighted) (excludes nil)))
```

```
(assert (feature (name StronglyConnected) (type or) (parent Algorithms) (sameGroupMembers Number Connected Cycle MST Shortest) (requires DFS Directed Weighted Unweighted) (excludes nil)))
(assert (feature (name Cycle) (type or) (parent Algorithms) (sameGroupMembers Number Connected StronglyConnected MST Shortest) (requires DFS Directed Undirected Weighted Unweighted) (excludes nil)))
(assert (feature (name MST) (type or) (parent Algorithms) (sameGroupMembers Number Connected StronglyConnected Cycle Shortest) (requires Undirected Weighted) (excludes nil)))
(assert (feature (name Shortest) (type or) (parent Algorithms) (sameGroupMembers Number Connected StronglyConnected Cycle MST) (requires Directed Weighted) (excludes nil)))
```

(그림 10) GPL 특성 모델에 대한 fact

```
;; FeatureConfguration #1
(assert (selectedFeature (name GPL)))
(assert (selectedFeature (name Algorithms)))
(assert (selectedFeature (name BFS)))
(assert (selectedFeature (name GraphType)))
(assert (selectedFeature (name Weighted)))
(assert (selectedFeature (name Undirected)))
(assert (selectedFeature (name Directed)))
(assert (selectedFeature (name StronglyConnected)))
```

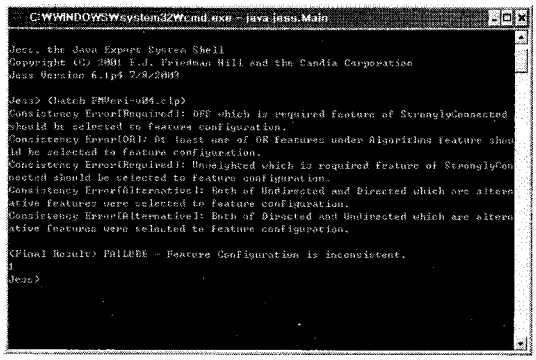
(그림 11) 특성 구성에 대한 fact 추가

4.2 프로그램 실행 결과

그림 12는 그림 11의 특성 구성에 대해 JESS 특성 구성 검증 프로그램을 실행한 결과를 보여주는 화면이다. 실행 결과를 보면 4.1절에서 언급한 모든 오류들을 정확히 찾아냈음을 알 수 있다.

4.3 평가

본 논문의 기법과 관련하여 여러 가지 기존 연구가 존재한다. [9]은 특성 모델을 기능적 특성(functional feature) 뿐 만 아니라 비기능적 특성(extra-functional feature)을 모델링할 수 있도록 확장하였다. 확장된 특성 모델을 Contraint



(그림 12) GPL 특성 구성에 대한 검증 결과

Satisfaction Problem(CSP)로 변환하고, constraint programming을 통하여 특성 모델에 대해서 여러 가지 정보를 추론하는 방법을 제안하였다. 추출 가능한 정보에는, 특성 모델로부터 만들어 질 수 있는 모든 제품의 개수, 비기능적 특성에 대한 제한 조건을 만족하는 모든 특성 구성의 종류 등이 포함된다. 이 기법은 각 특성들이 가지는 비기능적 요구 사항을 만족하는 제품 또는 특성 구성을 검색하는데 유용하지만, 특성들 사이의 requires와 excludes 관계를 고려하지 않는 단점이 있다.

[10, 11]에서는 시맨틱 웹 기반의 특성 구성 검증 기법이 제안되었다. 이 기법은, 특성 모델과 특성 구성을 Protege 온톨로지 편집 도구[12]를 이용하여 OWL[13] 언어로 표현하고 RACER[14]나 FaCT[15]와 같은 추론 엔진을 사용하여 특성 구성이 일관적인지를 검증한다. 본 기법은 시멘틱 웹 언어의 표준으로 떠오르는 OWL 언어를 이용하여 특성 모델 및 특성 구성을 표현하였기 때문에 시맨틱 웹에서의 특성 모델 저장 및 교환, 특성 구성 검증 기능 등의 구현을 용이하게 한다.

이 기법에서는 추론 엔진 자체 만으로는 불일치성을 발생시키는 원인을 명확히 파악하지 못하므로 따로 OWL 디버깅 도구를 구현하여 특성 구성에 존재하는 불일치성의 원인을 파악할 수 있도록 하였다. 또한, 특성 모델을 OWL로 변환하는 도구는 구현했지만, 특성 구성에 포함되지 않는 모든 특성에 대해서도 이를 표현하는 필요 조

건을 추가해야 하는 등 특성 구성에 대한 OWL 표현이 복잡한 작업인데도 이를 위한 도구는 구현하지 않았다. 본 논문의 기법은 특성 구성에 포함된 특성에 대한 fact만 선언하면 되므로 보다 간단히 특성 구성을 표현할 수 있다.

본 논문에서 제안한 JESS 프로그램에 기반을 둔 특성 구성 검증 기법은 다음과 같은 장점을 가진다.

• 다른 도구와의 결합성

JESS 시스템이 가지는 자바 언어와의 결합 능력으로 인해 본 논문에서 제안한 특성 구성 검증 기법을 기존의 소프트웨어 제품 라인 지원 도구나 새로운 도구 개발 시 쉽게 통합할 수 있다.

• 확장성

특성 모델과 특성 구성에 대한 데이터는 fact로서 표현되고 특성 구성이 가져야 할 제한 조건은 rule로 표현되기 때문에, 특성 구성에서 검증되어야 할 새로운 규칙이 필요한 경위 이를 쉽게 추가할 수 있다. 또한. 특성 모델이나 특성 구성이 확장되거나 변경되는 경우 fact를 수정함으로써 변경 사항을 쉽게 반영할 수 있다.

• 불일치성 원인 명시

특성 구성에서 규칙에 위배되는 상황이 발견될 때마다 오류 메시지를 일으키는 원인이 바로 출력되므로 불일치성을 일으키는 원인을 쉽게 알 수 있다.

5. 결론 및 향후 과제

소프트웨어 제품 라인 개발 시 제품들 사이의 공통점과 차이점을 표현하기 위하여 도메인 공학 단계에서 가장 널리 사용되는 모델이 특성 모델이다. 또한 특정 제품 생산 시 이에 포함될 특성들을 선택한 결과가 특성 구성이다.

이러한 특성 구성은 특성 모델에 표현되어 있는 여러 가지 제한 조건을 만족하는지 검증할 필요가 있다. 특히, 가변적 특성의 종류가 많아지고 제한 조건의 개수가 커지면 특성 구성을 자동으로 검증하는 도구가 반드시 필요한데, 본 논문에서는 이를 자동화하기 위한 방법으로 JESS 규칙 기반 시스템을 이용한 특성 구성 검증 기법을 제안하였다. 이 기법은, 특성 모델에 표현되어 있는 특성들에 대한 정보 및 특성들 사이의 제한 조건을 JESS 언어로 표현하고 JESS가 제공하는 규칙 엔진을 이용하여 특성 구성의 불일치성(inconsistency)을 검증한다. 본 논문의 기법의 유효성을 보여주기 위해, 소프트웨어 제품 라인의 기술들을 평가하기 위한 표준 문제로 제안된 Graph Product Line를 사례 연구로 사용하였다.

본 연구 결과는 자바 언어와의 결합성이 뛰어난 JESS 시스템에 기반을 두었기 때문에 다른 소프트웨어 개발 환경과 쉽게 통합될 수 있다. 또한, 특성 구성의 불일치성을 일으키는 원인과 위치에 대한 정보를 명확히 제공하며 특성 모델 또는 특성 구성이 변경되거나 새로운 제한 조건이 필요한 경우 쉽게 확장가능하다.

향후 연구 과제로는, 그래픽 특성 모델 작성 도구와의 통합, 시맨틱 웹 표준 언어를 지원하는 도구인 Protege-OWL과의 결합, 보다 큰 예제 시스템으로의 적용, 컴포넌트 기반 소프트웨어 제품 라인 공학 지원 방법 등에 대한 연구가 있다.

참 고 문 헌

[1] P. Clements and L. Northrop, "Software Product Lines: Practices and Patterns", Addison Wesley, 2002.

[2] Kyo C. Kang, S. Cohen, J. Hess,W. Nowak, and S. Peterson. Feature-oriented domain analysis(FODA) feasibility study. Technical Report CMU/SEI-90TR-21, Software EngineeringInstitute, Carnegie Mellon University, Pittsburgh, PA, November 1990.

[3] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh., FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures.Annals of Software Engineering, 5:143-168, 1998.

[4] M. Griss, J. Favaro, and M. d''Alessandro. Integrating feature modeling with the RSEB. InThe 5th International Conference on Software Reuse, pages 76--85, Vancouver, BC, Canada, June 1998.

[5] Roberto E. Lopez-Herrejon and Don S. Batory. A standard problem for evaluating productline methodologies. In *Proceedings of the Third International Conference on Generative and Component-Based Software Engineering*, pages 10-24, Erfurt, Germany, September 2001. Springer-Verlag.

[6] S. Thiel and A. Hein, "Systematic Integration of Variability into Product Line Architecture Design", SPLC2 2002, LNCS 2379, pp.130-153, 2002, Springer-Verlag.

[7] http://www.jessrules.com/jess/index.shtml

[8] Ernest Friedman-Hill, "JESS in Action", Manning, 2003.

[9] 200703_05. Benavides, D., Trinidad, P., Ruiz-Cort´es, A.: Automated Reasoning on Feature Models. In: Proceedings of the 17th Conference on Advanced Information System Engineering (CAiSE'05), Porto, Portugal (2004)

[10] Hai Wang, LI Yuan Fang, Jing Sun, Hongyu Zhang and Jeff Z. Pan. A Semantic Web Approach to Feature Modeling and Verification. In Proc. of the ISWC2005 Workshop on Semantic Web Enabled Software Engineering (SWESE). 2005.

[11] H. H. Wang, Y. F. Li, J. Sun, H. Zhang and J. Pan. Verifying Feature Models Using OWL. In Journal of Web Semantics: Science, Services and Agents on the World Wide Web, 5(2):117-129, June 2007. [bib+pdf+abstract]

[12] J. Gennari, M.A. Musen, R.W. Fergerson, W.E. Grosso, M. Crubezy, H. Eriksson, N.F. Noy, S.W. Tu, The evolution of protege: an environment for knowledge-based systems development, Tech. Rep. SMI-2002-0943, Stanford Medical Informatics, Stanford University, 2002.

[13] D.L. McGuinness, F. van Harmelen (Eds.), O WL Web Ontology Language Overview, 2003. http://www.w3.org/TR/2003/PR-owl-features-20031215/.

[14] V. Haarslev, R. M¨oller, RACER User''s Guide and Reference Manual: Version 1.7.6, 2002.

[15] I. Horrocks, Fact++ web site. http://owl.man.ac.uk/factplusplus/.

◐ 저 자 소 개 ◑

최 승 훈(Seung-Hoon Choi)

1990년 서울대학교 계산통계학과 졸업(학사)
1994년 서울대학교 대학원 계산통계학과 졸업(석사)
1999년 서울대학교 대학원 계산통계학과 졸업(박사)
2000년~현재 : 덕성여자대학교 정보공학대학 교수
2005.8~2006.7 : George Mason University 방문 연구
관심분야 : 소프트웨어 프러덕트 라인, 자동 생성 프로그래밍, 온톨로지
E-mail : csh@duksung.ac.kr