JESS 시스템을 이용한 특성 모델링 도구 구현

지은미○ 정혜숙 곽미선 최승훈

덕성여자대학교

{emi1123, sunnyspot, onlyalbore, csh}@duksung.ac.kr

Implementation of Feature Modeling Tool using Jess System

Eun-Mi Ji, Hye-Sook Jung, Mi-Sun Kuak, Seung-Hoon Choi

Duksung Women's University

요 약

특성 모델(Feature Model)은 소프트웨어 제품 라인 개발 시 도메인 공학 단계에서 제품들 사이의 공통점과 차이점을 모델링하는데 널리 사용된다. 특성 구성(Feature Configuration)은 특성 모델로부터 특정 제품에 포함될 특성들을 선택한 결과이다. 특성 구성은, 특성 모델에 표현되어 있는 여러 가지 제한 조건을 만족해야 한다. 본 논문에서는 특성 모델 작성 기능과 특성 구성 정의 기능을 지원하고 특성 구성의 검증 기능을 지식 기반 시스템인 JESS를 활용하여 구현한 특성 모델링 도구를 제안한다. 본 도구는 자바 언어와의 결합성이 좋은 JESS 시스템을 이용하여 확장성이 좋으며 특성 구성에서의 오류 원인을 명확히 알려주는 장점을 가진다.

1. 서 론

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, 소프트웨어 개발 초기 단계에 소프트웨어 패밀리에 속하는 멤버들 사이의 차이점과 공통점들을 미리 예측하고 분석함으로써 전략적인 재사용이 가능하도록 하여 소프트웨어 개발 생산성을 향상시키는데 있다.

소프트웨어 제품 라인을 구축하는데 있어서 먼저 도메인 공학(Domain Engineering) 단계를 거친다. 이 단계에서는 도메인 분석 과정을 통해서 특정 도메인에 존재하는 재사용 가능한 자산들을 식별한다. *특성 모델(Feature Model)*은 도메인 공학 단계에서 가장 널리 사용되는 모델로서, 특정 도메인에서 제품들 사이의 공통된 개념들과 서로 다른 개념들을 모델링하는데 사용된다.

*특성 구성*이란 특성 모델로부터 특정 제품 개발자가 선택한 특성 선택 결과이다[2]. 즉, 특성 모델에 표현되어 있는 제품 라인 멤버들 사이의 차이점들 중에서 생성하고자 하는 특정 제품이 포함해야 할 특성들에 대한 요구 사항을 의미한다.

FODA(Feature Oriented Domain Analysis) [3], FORM(Feature Oriented Reuse Method)[4], FeatuRSEB[5] 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 않으며 이를 위한 자동화 도구가 필수적이다.

JESS [1] 는 Java Expert System Shell의 약자로서 Sandia 국립 연구소의 Ernest Friedman-Hill에 의해 자바 언어로 개발된 규칙 엔진이자 지식 기반 시스템 개발 환경이다. JESS 시스템을 사용하면 선언적인 규칙

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

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.

본 논문에서는 JESS 시스템을 이용한 특성 구성 검증 기법을 제안하고 이를 활용한 특성 모델링 도구를 구현한다. 본 도구는 특성 모델의 구축과 특성 구성의 작성을 지원하며, 특성 모델에 표현되어 있는 특성들에 대한 정보 및 특성들 사이의 제한 조건을 JESS 언어로 표현하고 JESS 가 제공하는 규칙 엔진을 이용하여 특성 구성의 불일치성(inconsistency)을 검증한다.

본 논문의 구성은 다음과 같다. 제 2 장에서 특성 모델링 도구의 아키텍처를 살펴보고 제 3 장에서 특성 구성 검증을 위한 JESS 활용 기법을 기술한다. 제 4 장에서는 본 논문에서 구현한 특성 모델링 도구를 설명하고 제 5 장에서 결론 및 향후 과제를 기술한다.

2. 특성 모델링 도구 아키텍처

본 논문에서는 제안하는 JESS 규칙 기반 시스템을 이용한 특성 모델링 도구는 크게 3 개의 모듈로 구성되며 전체 구조는 그림 1과 같다.

*특성 모델 작성 모듈*은 개발하고자 하는 소프트웨어 제품 라인의 특성 모델을 작성하기 위한 여러 가지 기능을 제공한다. 사용자는 각 특성을 생성하고 특성의 이름, 타입 등을 설정하고 특성들 사이의 계층 구조를 정의할 수 있다. 또한 특성들 사이의 제한 조건(requires 와 excludes)을 설정할 수 있는 기능을 제공한다.

*특성 구성 생성 모듈*은 이미 만들어진 특성 모델로부터 특정 제품에 포함될 특성들을 선택할 수 있는 기능을 제공한다. 선택적(optional) 특성인 경우 어떤 제품에는 포함되고 어떤 제품에는 포함되지 않을 수 있으므로 제품 개발자가 포함할 지 안 할지를 선택할 수 있는 기능을 제공한다. 택일적(alternative) 특성인 경우 택일적 특성 그룹에 속하는 특성들은 그 그룹 중에서 반드시 하나만이 특성 구성에 포함되어야 하며 사용자는 이러한 선택을 할 수 있다.

*특성 구성 검증 모듈*은 JESS 시스템을 사용하는데 JESS 시스템의 구조는 그림 1 의 오른쪽 부분과 같다. 규칙이 적용될 대상이 되는 데이터는 fact 라고 하는 기본 원소로 구성되며 working memory 또는 fact base 에 존재한다. 본 논문에서의 특성 모델과 특성 구성 결과에 대한 정보는 이 working memory 에 존재한다. Rule base 에는 지식을 나타내는 규칙의 집합이 존재하며, if-then 형태로 정의된다. 이러한 규칙은 working memory 에 존재하는 fact 들에 대해서 if 절이 만족하였을 때 then 에 정의된 행동을 실행한다. 본 논문에서는 특성 구성이 만족해야 할 제한 조건이 rule 로 표현된다.

Working memory 에 대해서 if 절이 만족되어 실행(fire)되어야 할 규칙이 존재하는지를 찾는 모듈이 Pattern Matcher 이며 찾아진 규칙을 저장하고 실행할 순서를 결정하는 모듈이 Agenda 이다. Execution Engine 은 agenda 에 존재하는 규칙들의 action 부분을 실제로 실행시키는 일을 한다.

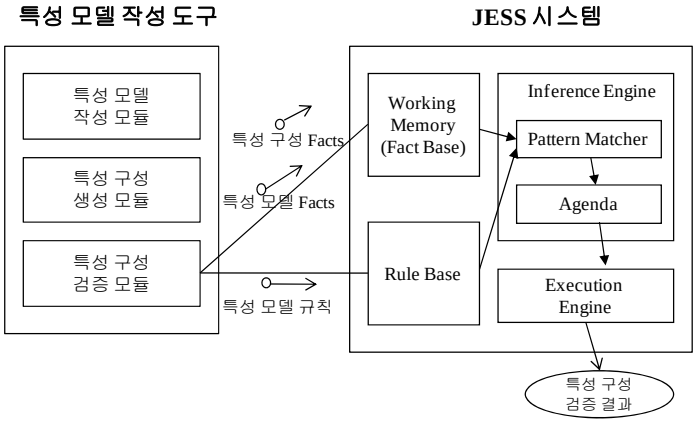

그림 1 특성 모델링 도구 전체 구조

3. 특성 구성 검증을 위한 JESS 활용

3.1 사례 연구: Graph Product Line

본 논문에서는 소프트웨어 제품 라인 기술들에 대한 평가를 위해 [6]에서 제안한 표준 문제인 Graph

Product Line(GPL) 특성 모델(그림 2)을 예제로 하여 특성 모델링 도구 프로그램을 기술한다.

그림 2의 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.

[6]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.

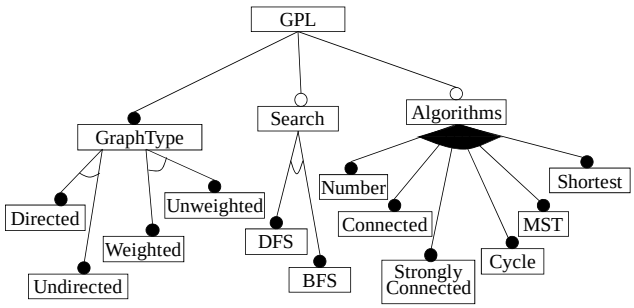

그림 2 Graph Product Line의 특성 모델

(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

3.2 특성 모델 및 특성 구성을 위한 JESS 템플릿

특성 모델과 특성 구성에 대한 fact들을 생성하기 위해서는 그 구조를 먼저 정의해야 한다. JESS에서는 (deftemplate)이라는 명령어를 이용하여 fact의 구조에 관한 템플릿을 정의한다. 그림 3은 특성 모델에 존재하는 각 특성의 정보를 표현하기 위한 템플릿 *feature*와 특성 구성에 포함된 특성의 이름을 가지는 템플릿 *selectedFeature*를 JESS 언어로 구현한 코드이다. *feature* 템플릿이 가지고 있는 슬롯(slot)의 의미는 표 2와 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 excluded 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 슬롯(slot)의 의미

슬롯 이름	슬롯이 가지는 정보
name	이 특성의 이름을 가진다.
type	이 특성의 타입으로써, mandatory, optional, alternative, or 중 하나의 값을 가진다.
parent	이 특성의 부모 특성의 이름을 가진다.
sameGroup Members	이 특성의 type 슬롯의 값이 mandatory이거나 optional인 경우에는 nil 값을 가지며 alternative 또는 or인 경우에는 같은 그룹에 속하는 특성들의 이름 리스트를 가진다.
requires	이 특성과 requires(필요로 한다) 관계에 있는 특성들의 이름 리스트를 가진다.
excludes	이 특성과 excludes(배제한다.) 관계에 있는 특성들의 이름 리스트를 가진다.

3.3 특성 구성 검증을 위한 Rule

특성 구성에서 검증해야 할 제한 조건은 아래와 같으며, JESS rule들은 특성 구성이 이러한 제한 조건들을 모두 만족하는지 검사하는 규칙을 제공한다.

1) 필수적(mandatory) 특성에 대한 제한 조건

특성 모델에서 필수적 제한 조건을 가진 특성은 특성 구성에 반드시 포함되어야 한다.

2) 선택적(optional) 특성에 대한 제한 조건

특성 모델에서 선택적 제한 조건을 가진 특성은 특성 구성에 포함되거나 포함되지 않을 수 있다. 특성 모델에서 아무런 제한 조건을 지정하지 않으면 자동적으로 선택적 제한 조건을 가진 특성이 된다.

3) 택일적(Alternative) 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.

4) Or 특성에 대한 제한 조건

특성 모델에서 Or특성 그룹에 속하는 특성들 중 적어도 하나의 특성은 특성 구성에 포함되어야 한다.

5) excludes(배제한다) 관계에 대한 제한 조건:

특성 모델에서 서로 excludes 관계에 있는 특성들은 특성 구성에 함께 포함될 수 없다.

6) requires(필요로 한다) 관계에 대한 제한 조건:

특성 모델에서 서로 requires 관계에 있는 특성들은 특성 구성에 반드시 함께 포함되어야 한다.

그림 4과 그림 5는 택일적 특성에 대한 제한 조건을 검사하기 위한 규칙을 JESS 언어로 구현한 코드이다. 그림 4은 같은 부모 밑에 존재하는 택일적 특성 그룹에서 하나의 특성만이 특성구성에 포함되었는지를 검사하는 규칙이다. 특성 구성에 포함된 특성과 함께 택일적 특성 그룹에 있는 나머지 특성들이 특성 구성에 포함되면 오류 메시지를 화면에 출력한다.

그림 5는 택일적 특성 그룹에서 어떠한 특성도 선택되지 않은 경우를 검사하는 규칙이다. 특성 모델에서 같은 부모 밑에 존재하는 택일적 특성 그룹에의 특성들 중 어떠한 특성도 특성 구성에 포함되지 않았을 경우에 오류 메시지를 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 out "Consistency Error[Alternative]: Both of " ?name1 " and " ?name2 " which are alternative features were selected to feature configuration." crlf)
        (modify ?*finalResult* (result inconsistent)))
```

그림 4 택일적 특성을 위한 규칙1

```
(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 5 택일적 특성을 위한 규칙2

```
(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 out "Consistency Error[Exclusive]: " ?e " which is exclusive feature of
" ?name1 " should not be selected to feature configuration." crlf)
(modify ?*finalResult* (result inconsistent))
) $?sameGroupMembers1) (requires $?requires1) (excludes $?excludes))
```

그림 6 excludes 관계를 위한 규칙

그림 6은 excludes 관계에 대한 제한 조건을 검사하기 위한 규칙을 JESS 언어로 구현한 코드이다. 특성 모델에서 서로 excludes 관계에 있는 특성들이 특성 구성에 함께 선택되면 오류 메시지를 출력한다.

4. 특성 모델링 도구 구현

본 절에서는 자바 언어와 JESS 규칙 기반 시스템을 결합하여 구현한 특성 모델링 도구에 대하여 기술한다.

4.1 특성 모델 작성 모듈

그림 7은 GPL 특성 모델을 본 논문의 특성 모델링 도구에서 작성한 결과를 보여주는 화면이다. 특성 모델링 도구는 크게 두 부분으로 나뉜다. 왼쪽 부분은 탐색기 역할을 하며 모든 제품 라인의 특성 모델과 특성 구성을 계층 구조 형태로 보여준다. 오른쪽 부분이 실제 작업 공간으로서 특성 모델을 작성하는 기능을 제공한다.

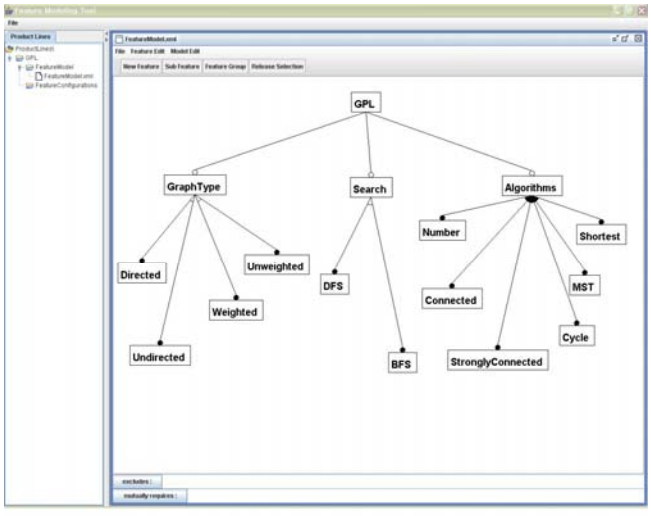

그림 7 특성 모델 작성 과정

4.2 특성 구성 생성 모듈

그림 8은 본 특성 모델링 도구에서 하나의 특성 구성을 작성하는 과정을 보여준다. 특성 구성 창에서는 특성 모델을 기반으로 각각의 특성을 클릭해서 선택할 수 있다. 각각의 특성마다 requires나 excludes 관계에 있는 특성을 추가할 수 있다.

4.3 특성 구성 검증 모듈

특성 구성의 검증 기능은 특성 구성 창에서 *JESS Validation* 버튼을 누름으로써 실행된다. 회색으로 표시된 특성은 특성 구성에 포함되었음을 의미한다. 이 버튼을 누르면 특성 모델과 특성 구성에 포함되어 있는 정보를 이용하여 fact base가 만들어지고 미리 정의되어 있는 JESS rule이 적용되어 특성 구성의 일치성(consistency)를 검증한다. 그림 8는 특성 모델에 대한 fact 일부를 보여주며, 그림 9은 특성 구성에 대한 fact를 보여준다.

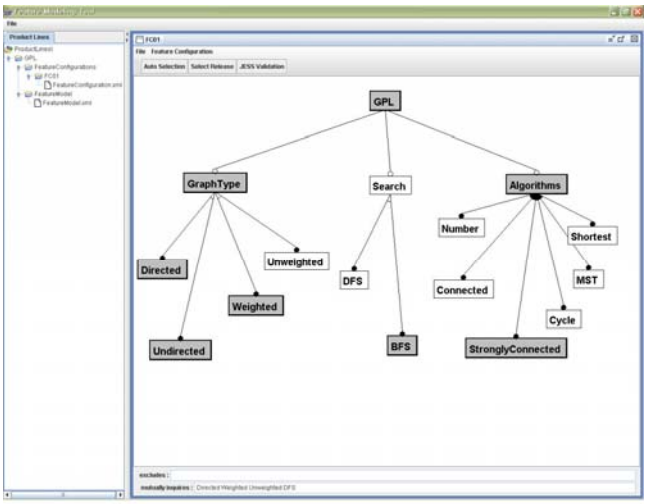

그림 8 GPL 특성 구성 생성 과정

```
;;GPL
(assert (feature (name GPL) (type mandatory) (parent nil) (sameGroupMembers nil)
(requires nil) (excludes nil)))
(assert (feature (name GraphType) (type mandatory) (parent GPL)
(sameGroupMembers GPLGraphTypeSearchAlgorithms) (requires nil) (excludes nil)))
(assert (feature (name Search) (type optional) (parent GPL) (sameGroupMembers
GPLGraphTypeSearchAlgorithms) (requires nil) (excludes nil)))
(assert (feature (name Algorithms) (type optional) (parent GPL)
(sameGroupMembers GPLGraphTypeSearchAlgorithms) (requires nil) (excludes
nil)))
(assert (feature (name Number) (type optional) (parent Algorithms)
(sameGroupMembers Connected StronglyConnected Cycle MST Shortest)
(requires Directed Undirected Weighted Unweighted BFS DFS) (excludes nil)))
```

그림 9 GPL 특성 모델에 대한 fact

```
(assert (selectedFeature (name GPL)))
(assert (selectedFeature (name GraphType)))
(assert (selectedFeature (name Search)))
(assert (selectedFeature (name Directed)))
(assert (selectedFeature (name DFS)))
(assert (selectedFeature (name Shortest)))
(assert (selectedFeature (name Weighted)))
```

그림 10 GPL 특성 구성에 대한 fact

그림 10은 GPL 특성 구성 창에서 검증을 한 후의 결과를 보여주는 화면이다. 현재 특성 구성에는 다음과 같이 특성 모델에 정의되어 있는 제한 조건을 위배한 사실이 포함되어 있다.

- *StronglyConnected* 특성과 requires 관계에 있는 DFS 특성과 *Unweighted* 특성이 특성 구성에 포함되어 있지 않다.

- *GraphType* 특성의 하위의 *Alternative* 특성 그룹들 중에 *Directed* 특성과 *Undirected* 특성 모두 특성 구성에 포함되어 있다.

그림 10의 특성 구성 검증 결과를 보면 위에서 언급한 모든 오류들을 정확히 찾아냈음을 알 수 있다.

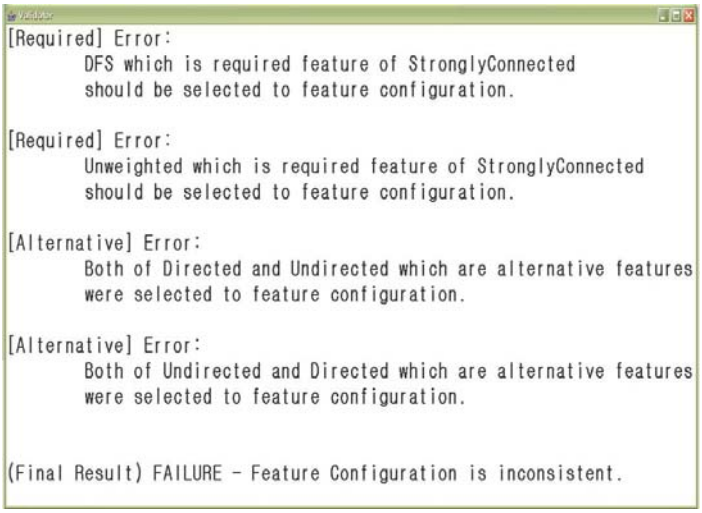

그림 10 GPL 특성 구성 검증

5. 결론

본 논문에서는 JESS 규칙 시스템을 이용한 특성 모델링 도구를 제안하였다. 이 도구는 특성 모델 작성 기능, 특성 구성 생성 기능 및 특성 구성 검증 기능을 제공한다.

본 도구의 특징은 자바 언어로 구현된 규칙 기반 시스템인 JESS 를 이용하여 특성 모델에 표현되어 있는 특성 및 특성들 사이의 관계/제한 조건을 JESS 언어로 정의하고 JESS 가 제공하는 규칙 엔진을 이용하여 특성 구성의 불일치성(inconsistency)을 검증하는데 있다.

본 논문에서 구현된 도구는 특성 모델과 특성 구성에 대한 정보는 fact 로서 표현되고 특성 구성이 가져야 할 제한 조건은 rule 로 표현되기 때문에, 특성 구성에서 검증되어야 할 새로운 규칙이 필요한 경우 이를 쉽게 추가할 수 있는 장점을 가진다. 또한, 규칙에 위배되어 불일치성을 일으키는 원인을 정확히 알려주는 특징을 가진다.

향후 연구 과제로는, 시맨틱 웹 표준 언어를 지원하는 도구인 Protege-OWL 과의 결합, 보다 큰 예제 시스템으로의 적용, 컴포넌트 기반 소프트웨어 제품 라인 공학과의 통합 방법에 대한 연구 등이 있다.

참고문헌

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

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

[3] 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.

[4] 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.

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

[6] 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.