# GARDIAN: 실시간 내장형 소프트웨어 개발 방법론에서의 룰 기반의 모델링 평가 및 지원도구

## (GARDIAN: Rule Based Modeling Validation for Concurrent Object Modeling and Architectural Design mEThod(COMET))

김 순 태 † 김 진 태 †† 박 수 용 †††
(Suntae Kim) (Jintae Kim) (Sooyong Park)

**요 약** UML(Unified Modeling Language)은 대부분의 소프트웨어 개발 방법론에서 목표로 하는 소프트웨어를 분석·설계하기 위하여 널리 사용되며, UML로 작성된 산출물을 기반으로 목표 소프트웨어를 구축한다. 그러나 방법론에서 모델링에 대한 가이드라인이 보통 자연어로 기술되어 있기 때문에 목표 소프트웨어를 위한 모델이 이를 적절히 준수하고 있는가의 검증이 어렵다는 문제점을 가지고 있다. 본 논문에서는 실시간 내장형 시스템(Real-time Embedded System)을 위한 방법론인 COMET방법론을 대상으로 모델링의 가이드라인을 표현하고, 표현된 가이드라인을 기반으로 모델을 평가할 수 있는 룰 기반 COMET 방법론 가이드라인 평가 프레임워크인 GARDIAN을 제안한다. 제안된 프레임워크의 유용성을 검증하기 위하여 비전문가가 UML을 사용하여 분석·설계한 지능형 로봇의 주행 시스템에 프레임워크를 적용하여 보았다.

**키워드** : 전문가 시스템, COMET, 방법론, UML, 가이드라인, 프레임워크

***Abstract*** UML (Unified Modeling Language) is widely used to analyze and design target software. Developers also implement the target software based on the UML artifacts. However, it is difficult to validate whether the artifacts are generated to correspond to the modeling guidelines because the guidelines for UML modeling are described in natural language. This paper discusses rule based model checker focused on whether models are designed according to modeling methodology. We propose rules and their own checker, named GARDIAN, for UML model validation. The checkers are designed for COMET method for the real-time embedded system. We illustrate our checkers using Intelligent Robot system to validate our approach.

**Key words** : Expert System, COMET, Methodology, UML, Guideline, Framework

## 1. 서 론

대규모 프로젝트에서는 프로젝트의 산출물의 품질을 높이기 위하여, 프로젝트의 규모와 특성에 맞는 소프트웨어 개발 방법론을 선정하여 프로젝트를 진행한다. 엔터프라이즈 시스템(Enterprise System)에서는 USDP (Unified Software Development Process)[1]와 UML Component[2]가 대표적인 예이며 실시간 내장형 시스템에 대해서는 COMET(Concurrent Object Modeling and architectural design mEThod)[3] 방법론이 그 예 중 하나이다.

이런 방법론에서는 소프트웨어를 분석·설계하기 위하여 UML(Unified Modeling Language)[4]를 기반으로 한 모델(Model)을 사용하고, 이 모델을 기반으로 소프트웨어 시스템을 구축한다. 그래서, UML기반의 모델의 품질은 목표 소프트웨어의 품질과도 직결 되며, 특히 모델 기반 개발 (Model Driven Development)의 경우에는 그 모델의 품질은 더더욱 중요하다[5].

소프트웨어 품질을 향상하기 위해서는 구현단계에서 결함(defect)을 발견하고 수정하는 것보다 모델링 단계

† 학생회원 : 서강대학교 컴퓨터학과
jipsin08@sogang.ac.kr
†† 정 회 원 : 삼성전자 정보통신연구소
canon.kim@gmail.com
††† 정 회 원 : 서강대학교 컴퓨터학과 교수
sypark@sogang.ac.kr
논문접수 : 2007년 4월 25일
심사완료 : 2007년 6월 28일

에서 발견하여 개선하는 것이 개발 비용의 감소와 높은 품질의 소프트웨어 개발을 가능하게 한다. 모델링 수준에서 품질을 검증한다는 것은 모델이 주어진 방법론을 잘 준수하고 있는지 여부와 목표 시스템을 위한 설계가 해당 도메인의 지식을 반영하는지 여부로 나눌 수 있다. 방법론을 잘 준수하고 있는지 여부를 확인하기 위해 기존에 OCL[5], ArgoUML[6], Software Metric[7], Inspection, Review 또는 Workthrough[8] 방법들이 활용되었다. 그러나, 이러한 방법들은 전문가의 지식을 기술하는 방법이 어렵고, 모델이 지식의 위배 경우 설계자에게 전달하는 정보가 부족하다는 단점을 가지고 있기 때문에, 모델 수준에서 결함을 찾아 품질을 개선하기에 많은 도움을 주지 못한다. 그러므로 가이드라인을 정형적으로 표현하고, 표현된 가이드라인을 기반으로 자동적으로 모델을 평가할 수 있는 방법이 필요하다.

지식을 기반으로 모델을 평가하기 위해서는 지식이 추상적이지 않아야 한다. 엔터프라이즈 시스템을 대상으로 하는 방법론은 여러 도메인을 대상으로 하고 있기 때문에 가이드라인이 다양하게 해석이 가능하고, 추상적이기 때문에 객관적으로 검증하기가 어렵다. 이에 비해 실시간 내장형 시스템의 설계 방법론인 COMET방법론은 적용 영역이 실시간 내장형 시스템으로 한정되어 있으며, 타 방법론에 비해서 방법론의 각 단계별 모델의 가이드라인이 상당히 구체적이라는 특징을 가지고 있다. 그래서 본 논문에서는 연구의 범위를 COMET방법론의 가이드라인으로 제한하였다. COMET 방법론에서는 두 가지 종류의 가이드라인을 제시하고 있다. 첫째는, 실시간 내장형 시스템의 소프트웨어 생명 주기(Software Life Cycle)와 설계에 사용되는 새로운 개념들, 객체의 식별 기준, 그리고 개발 단계별 객체 정제의 기준들에 대한 가이드라인이며, 둘째는 앞서 제시된 개념들을 UML을 사용하여 표현하는 방법들에 대한 가이드라인이다. 두 가지 종류의 가이드라인 중에서 전자에 대한 위배여부의 판단은 사람의 지식과, 의사 결정이 많이 필요한 일이기 때문에 지식으로 선언적으로 정의하기 어렵다는 문제를 가지고 있다. 그래서, 본 논문에서는 연구의 범위를 후자, 즉 UML표기법에 대한 가이드라인으로 한정하고자 한다.

COMET방법론의 UML표기에 대한 가이드라인을 효율적으로 표현하고, 평가하기 위해서는 다음의 세가지 고려사항이 존재한다. 첫째로는, 방법론 지식 표현의 용이성을 고려해야 한다. COMET 전문가가 방법론에 대한 지식을 표현하기 위해서 별도의 복잡한 언어를 배워야 한다면 별도의 지식 표현 전문가가 필요할 것이다. 그러므로, COMET전문가가 지식을 쉽게 기술할 수 있어야 한다. 둘째, 지식의 재사용성 및 상호 운용성을 고려해야한다. 만약 정의한 방법론의 하나의 프로젝트나 모델에만 국한된다면 매번 다시 정의해야 한다. 따라서, 지식의 재사용 및 상호 운용성이 보장되어야 한다. 마지막 고려사항은 지식의 자가 기술성(Self-Description)이다. 모델이 가이드라인에 위배된 경우 설계자는 자신의 설계 모델에서 정확히 어느 부분이 위배되었으며, 또한 무슨 이유로 위배되었는가를 알 수 있도록 지식의 유추 과정을 설명할 수 있어야 한다.

본 논문에서는 위에서 언급한 고려사항을 해결하기 위해서 전문가 시스템의 룰을 사용하여 전문가의 방법론의 지식을 표현하고, 표현된 지식을 기반으로 설계자의 UML모델을 검증하기 위한 프레임워크를 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구에 대한 관련 연구 및 그 문제점에 대해 살펴보고, 3장에서는 룰 기반 COMET지원 프레임워크와 지원도구를 제안한다. 4장에서는 실제로 프레임워크를 적용하여 모델의 가이드라인 위배를 추적하여 평가해보고, 5장에서는 결론과 향후 연구 방안을 논의하고자 한다.

## 2. 가이드라인 위배체크에 대한 관련 연구

모델의 가이드라인의 위배여부를 체크하기 위하여 기존에는 OCL(Object Constraints Language)[5]를 사용하거나, 도구를 사용[6] 또는, Software Metric[7,9]을 사용하는 방법 등이 있었다.

### 2.1 OCL을 사용한 가이드라인 위배 체크

[4]에서는 자동차 점화장치 설계 모델에서 모델링 규칙의 표준 준수여부를 체크하기 위하여 OCL을 활용하였다. 그러나 OCL을 사용하여 모델링의 가이드라인을 기술하는 데는 한계를 가지고 있다. OCL은 약 33개의 (UML 1.5기준)의 CFG(Context Free Grammar)상의 LHS(Left Hand Side)를 정의하고 있기 때문에 비 전문가가 사용하기에 복잡하다는 단점을 가지고 있다. 그래서 모델링 가이드라인의 지식을 OCL을 사용하여 정의하기 위해서는 가이드라인 전문가 이외에 OCL 전문가가 별도로 필요하다[5]. 또한, OCL을 사용하면 모델이 모델링 가이드라인을 위배한 경우 설계자가 위배의 위치와 원인에 대해 알 수 있도록 적절한 메시지 생성을 지원하지 않는다는 단점을 가지고 있다.

### 2.2 도구를 사용한 가이드라인 위배 체크

ArgoUML[6]에서는 모델요소에 대한 체크리스트를 제공하는 기능을 가지고 있다. 이 체크리스트를 제공하는 목적은 모델의 Review와 Inspection을 지원하기 위함이다. 이 모델링 도구는 이미 모든 모델 요소(Model Element)의 체크리스트가 미리 정의되어 있기 때문에, 방법론 전문가의 지식을 표현하고, 편집하는 것이 불가능하다. 그리고 도구에서 미리 기술된 가이드라인을 기

반으로 가이드라인이 평가되지 않으며, 단지 체크리스트만 제공함으로써, 실제의 모델에 대한 가이드라인 위배 체크는 사람이 직접 해야 한다는 단점을 가지고 있다.

### 2.3 Software Metric을 사용한 가이드라인 위배 체크

[9]에서는 Software Metric[7]을 적용하여 모델 정보를 분석해주는 SAAT(Software Architecture Analysis Tool)이라는 도구를 소개한다. 이 도구는 목표 소프트웨어를 설계한 UML모델을 기반으로 아래의 내용을 분석하여 사용자에게 보여준다.

- 유스케이스, 액터, 패키지, 객체, 클래스 등의 개수
- 유스케이스가 필요한 액터 혹은 시나리오 들

이 도구는 모델링 도구인 Rational Rose[10]를 사용하여 도식된 UML모델 정보를 데이타베이스에 삽입한 후, SQL(Structured Query Language)의 Select구문으로 질의하여 모델 정보를 검색하는 방식으로 소프트웨어를 분석한다. 그리고 그 검색 결과를 HTML(Hyper Text Markup Language)혹은 MetricView[11]를 통하여 사용자에게 보여준다. 그러나 SQL의 Select 구문으로 방법론 지식을 표현하기 어려우며, 방법론 전문가가 SQL을 모두 학습하기 어렵다는 점은 문제점이다.

이런 접근 방법들은 공통적으로 전문가의 지식을 기술하기 어렵다는 문제점을 가지고 있다. 또한, 가이드라인을 위배했을 때 그 위배에 대해서 설계자에게 알려주는 정보가 부족하다는 점은 또 다른 문제점이다. 그래서 본 논문에서는 이런 문제점을 보완하기 위해서 룰 기반의 지식 표현 방법 및 표현된 지식을 바탕으로 UML모델을 평가하기 위한 지원 도구를 제안한다.

## 3. 룰 기반 COMET지원 프레임워크

COMET방법론의 지식을 기술하기 위해서 방법론 지식 표현의 용이성, 지식의 재 사용성 및 상호운용성, 그리고 지식의 자가 기술성에 대해서 고찰해 보아야 한다. 이러한 사항을 다루기 위하여 본 논문에서는 그림 1과 같은 룰 기반의 COMET지원 프레임워크를 제안한다.

본 프레임워크의 흐름은 다음과 같다. COMET 방법론의 지식을 IF-THEN-ELSE형식의 룰 표현식을 사용하여 기술한 후(1), 기술된 룰을 룰 엔진 프레임워크에 입력(2)한다. 그리고 설계자가 입력한 UML모델의 결과를 XMI(Xml Metadata Interchange)[12]로 변환하여 룰 엔진 프레임워크에 입력(3)하면, 룰 엔진은 기술된 방법론의 지식을 기반으로 설계자가 설계한 UML모델을 평가한다. 룰 엔진 프레임워크에서는 평가의 결과를 가이드라인 위배 보고서 형식으로 생성하며(4), 설계자는 이를 기반으로 모델을 수정하여 모델의 품질을 개선할 수 있다(5).

본 프레임워크의 특징을 다음과 같이 요약해 볼 수

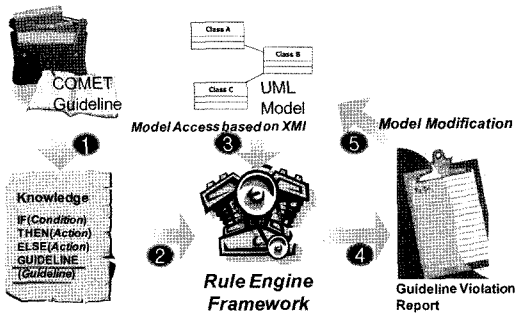



그림 1 COMET방법론을 위한 룰 프레임워크 개요

있다. 첫째, 전문가 시스템이 사용하고 있는 IF-THEN-ELSE형식의 지식 표현 방식[13]을 사용하여 방법론 전문가는 쉬운 지식 표현 방법을 사용하여 방법론의 지식을 표현할 수 있도록 지원하였다. 둘째, UML 모델에 대한 표준의 접근 방법인 XMI를 활용하여 모델을 접근하고, 이를 기반으로 전문가의 지식을 기술하기 때문에 모델링 도구와 상관없이 COMET방법론을 활용하는 모든 프로젝트에서 공통적으로 재사용이 가능하다. 셋째, 방법론의 지식을 기술하기 위하여 정의한 룰 표현식에서 설계자에게 전달할 메시지를 기술 할 수 있도록 가이드라인 부분을 확장하였다. 설계자는 이 부분에서 생성된 메시지를 본 후, 자신의 모델이 가이드라인에 위배된 이유를 알 수 있다. 다음 장부터는 위에서 기술한 연구의 이슈를 중심으로 룰 엔진 프레임워크에 대하여 상세하게 설명한다.

### 3.1 룰 표현식

본 프레임워크에서는 가이드라인을 표현하기 위하여 다음과 같은 룰 표현식을 제안하고, 사용하였으며, 이를 EBNF(Extended Backus-Naur Form)[14]를 통하여 표 1과 같이 기술 하였다.

표 1 룰 표현식

```
[Boolean|Double|String] RuleName() {
  IF(conditions)
    THEN(action)
    ELSE(action)
    RESULTINFO(expression)
}
```

룰의 리턴 타입은 Boolean, Double 과 String형식이 가능하다. 그리고 룰의 기본 형식은 IF-THEN-ELSE-RESULTINFO이다. 룰 표현식 중 IF구문에 룰의 수행을 결정하기 위한 조건문을 기술해주며, THEN구문에는 조건이 참이 될 때 수행될 Action을 기술하고, ELSE구문 이하에는 조건이 거짓인 경우 수행될 Action을 기술

한다. 그리고 RESULTINFO구문에는 IF구문의 조건이 거짓일 때 설계자에게 보여줄 가이드라인을 기술한다. 아래의 표 2에서 각 구문에서 쓸 수 있는 표현식을 EBNF로 기술하였다.

표 2 상세 룰 표현식

*conditions* → *andCondition* ( <OR> *andcondition*))*
*andcondition* → *unaryCondition* (<AND> *unaryCondition*)*
*unaryCondition* → <NOT> *booleanExpression* | *booleanExpression*
*booleanExpression* → *expression*( <RELATION_OP> *expression*)*
*expression* → *term*((<PLUS> | <MINUS>) *term*)* | <BOOLEANLITERAL>
*term* → *unaryFactor* ((<MULTIPLY>|<DIVIDE>|<MODULUS>) *unaryFactor*)*
*unaryFactor* → "-"*factor* | *factor*
*factor* → <CONSTANT> | ***callMethod*** | ***modelElements*** |<STRINGLITERAL> | *conditions*
*action* → *conditions*()

위의 룰의 표현식 중에서 *callMethod*는 사용자 정의 함수를 정의할 수 있는 확장 메커니즘으로, 본 프레임워크에서 기본적으로 제공하는 사용자 함수는 표 3과 같으며, 이는 룰을 정의하는데 있어서 보조적인 기능을 담당한다. *modelElements*는 룰 모델을 접근하기 위한 방법으로 다음 장에서 자세히 설명하도록 한다.

표 3 사용자 정의 함수 목록

| 내장 함수명 | 함수 설명 |
|---|---|
| List | 지정한 모델요소가 여러 개의 모델의 요소를 가진 경우 그 이름의 목록을 보여준다. |
| IsOperationFormat | 지정한 모델 요소의 값이 Operation 형식인지 확인한다. |
| CountContents | 모델요소의 내용의 개수를 센다. 만약에 공백이면 이를 제외하고 개수를 센다. |
| CountElements | 지정한 모델요소의 내용의 값이 공백인 것과 관계없이 모델 요소의 내용의 개수를 센다. |
| IsIncluded | 지정한 모델요소에 특정 값이 포함되는지 확인한다. |
| Name | 모델 요소의 Phase정보와 이름을 얻어온다. |

### 3.2 룰 모델

룰 모델은 룰 표현식 상에서 *modelElements*에 해당되며, 이는 전문가의 모델 정보를 접근하기 위한 방법이다. 이 룰 모델을 기반으로 방법론 전문가는 설계자가 정의한 모델을 접근 할 수 있다.

그러나 방법론을 정의하기 위하여 모든 UML의 모델 요소가 다 필요한 것은 아니다. 그러므로, 본 연구에서는 UML의 모델요소 중 COMET방법론에서 필요로 하는 모델 요소만을 추출하였다(그림 2 참조). UML은 XMI Schema[12]로 표현되며, 본 논문에서는 위에서 UML설계 물을 정의한 룰 모델로 변환하기 위해서 XSLT(eXtensible Stylesheet Language Transformations)를 사용하였다.

룰 모델은 가장 상위에 *Model* 정보가 있다. 하나의 *Model*은 하나이상의 *Phase*를 가질 수 있다. *Phase*는 방법상에서 목표 소프트웨어를 구현하기 위해 수행할 각 단계를 의미하며, 하나의 *Phase*는 하나 이상의 다른 *Phase*를 가질 수 있다. COMET방법론에서는 1.RequirementsModeling, 2.AnalysisModeling, 3.Architecture Design, 4.Design, 5.Detailed SoftwareDesign의 총 5개의 *Phase*를 갖는다. 그리고, 1.Requirements Modeling *Phase* 이외에 다른 *Phase*에는 내부의 또 다른 *Phase*를 정의하고 있다.

룰 표현식에 있는 *modelElement*는 그림 2의 룰 모델을 기반으로 정의할 수 있다. 예를 들어 그림 3에서 *Phase*가 '1.RequirementsModeling'인 *Phase*에 있는 모든 *Actor*의 이름을 검색하기 위해서는 다음과 같은 표현식을 사용하면 된다.

```
Model.Phase(name="1.RequirementsModeling").
Actor.name
```

### 3.3 COMET방법론의 룰 정의

실시간 내장형 시스템을 설계하기 위한 COMET방법론에는 여러 가지 모델링 규칙이 존재한다. COMET방법론에서 언급하고 있는 가이드라인의 종류는 크게 세 가지 이다. 첫째는 하나의 다이어그램 내의 모델 요소(Model Element)간의 관계에 대한 가이드라인 이며, 둘째는 모델 정제에 대한 가이드라인, 마지막으로 모델간의 일관성(Consistency)에 대한 가이드라인이다.

이들은 COMET방법론을 사용할 때 설계자가 지켜야 할 규칙이며, 전문가는 룰 표현식과 룰 모델을 사용하여 COMET 방법론의 가이드라인을 정의할 수 있다. 다음은 각각의 가이드라인 위배에 대한 예이다.

#### 3.3.1 하나의 다이어그램 내의 모델 요소 간의 관계에 대한 가이드라인

COMET방법론에서는 한 단계에 도식해야 할 다이어그램내부의 모델 요소들의 관계나 모델 요소가 가질 수 있는 스테레오 타입들에 대해서 정의하고 있다. 예를 들면 Static Modeling의 System Context Diagram에서는 표 4와 같은 표준의 Association이름을 갖는다.

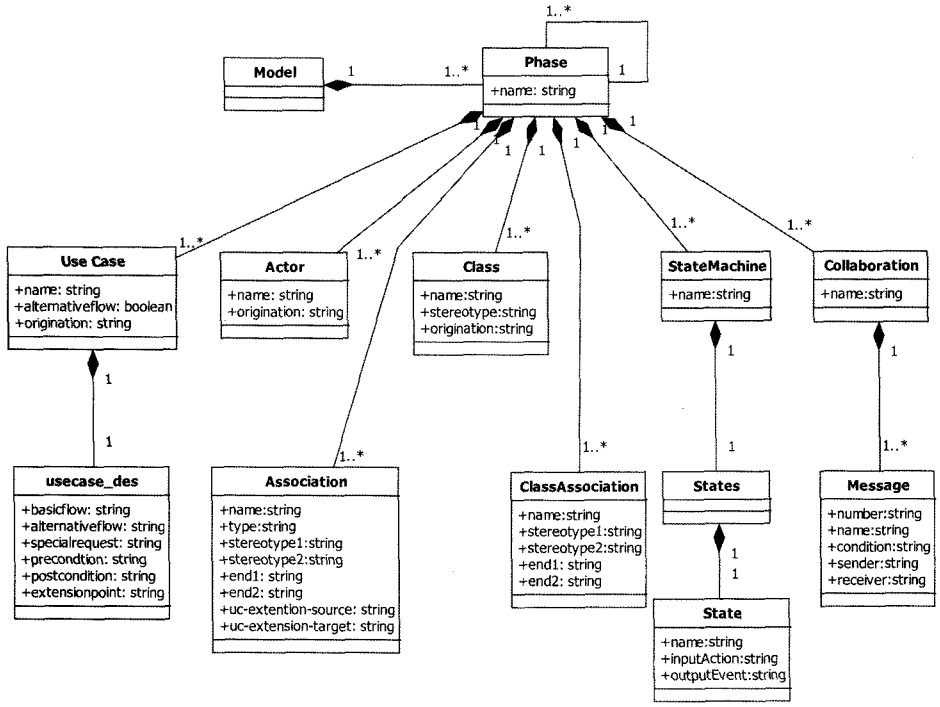



그림 2 룰 모델

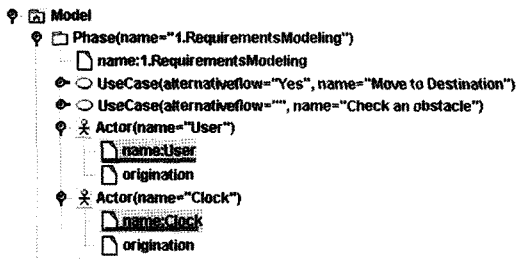



그림 3 룰 모델 예

표 4 System Context Diagram에서의 표준 Association 이름

| |
|---|
| <<external input device>> Inputs to <<system>> |
| <<system>> Outputs to <<external output device>> |
| <<external user>> Interacts with <<system>> |
| <<external system>> Interfaces to <<system>> |

COMET의 System Context Diagram에서 external input device의 스테레오타입을 갖는 클래스와 system의 스테레오타입을 갖는 클래스와의 Association의 이름은 'Inputs to'여야 한다고 정의하고 있다. 그림 4에서처럼 'external input device'의 스테레오타입을 가진 'Sensor' 클래스와 'system'스테레오타입을 가진 'Robot Navigation System' 클래스 사이의 관계가 'Inputs to'

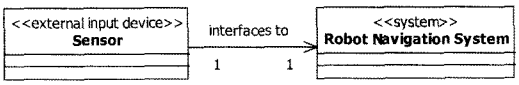



그림 4 모델요소간의 가이드라인 위배 사례

여야 하는데 'Interfaces to'로 되어 있기 때문에 이는 가이드라인 위배가 된다.

위의 규칙을 지키고 있는지 확인 하기 위해서는 모델요소가 <<external input device>> 혹은 <<system>> 이라는 스테레오타입을 갖고 있는가와, 해당 스테레오타입을 가진 요소 사이의 Association 의 이름이 'Inputs to' 인지이다. 이를 룰 표현식과, 룰 모델을 기반으로 정의하면 아래와 같다.

```
Boolean SM_ContextDiagram_Association체크룰()
{
   IF( Model.Phase(name="2.AnalysisModeling").Phase(name="2.1.StaticModeling").Phase(name="2.1.2.SystemContextDiagram").Association(stereotype2="system", stereotype1="external input device").name == "Inputs To" )
   THEN ( true )
   ELSE ( false )
   RESULTINFO("Analysis Modeling의 Object Structuring단계의 스테레오타입 external input device를 가진 Class ' " + Model.Phase(name="2.Analysis
```

```
Modeling"). Phase.(name="2.1.StaticModeling").Phase
(name="2.1.2.SystemContextDiagram"). Association
(stereotype2="system", stereotype1 ="external input
device").End1 + "' 와 system 스테레오타입을 가진
Class '" + Model. Phase(name="2.AnalysisModeling").
Phase (name="2.1.StaticModeling").Phase(name="2.1.2.
SystemContextDiagram").Association(stereotype2="system",
stereotype1 ="external input device").End2 +"' 의
관계는 Inputs to여야 합니다. 현재는 '"+ Model.
Phase(name= "2.AnalysisModeling").Phase(name=
"2.1.StaticModeling").Phase(name="2.1.2.SystemContextDi
agram").Association(stereotype2="system",
stereotype1="external input device").name +"'으로 되어
있습니다.")
}
```

IF구문이 의미 하는 것은 2.AnalysisModeling/ 2.1. StaticModeling/2.1.2.SystemContextClassDiagram 단계에 있는 스테레오타입1이 'system'이고, 스테레오타입2가 'external input device'인 Association의 이름을 검색해서 그 값이 'Inputs to'인지 조건을 파악 하는 것이다. 만약에 조건문의 수행 결과가 참이면 THEN구문을 실행하여 'true'가 반환되고, 거짓이면 ELSE구문을 실행하여 'false'가 반환되며 이와 함께 RESULTINFO가 실행된다. RESULTINFO구문은 설계자에게 가이드라인 위배를 설명하는 구문이므로, 정확히 무엇이 틀렸는지에 대하여 충실히 설명하고 있어야 한다. 다음은 위의 룰을 수행하였을 때 수행 결과이다.

```
Result::false
[Rule:SM_ContextDiagram_Association체크룰]Analysis
Modeling의 Object Structuring단계의 스테레오타입
external input device를 가진 Class 'Sensor' 와 system
스테레오타입을 가진 Class 'Robot Navigation System'
의 관계는 Inputs to여야 합니다. 현재는 'interfaces
to'으로 되어 있습니다.
```

이 결과를 통해서 현재 모델이 COMET 방법론에서 이야기하는 가이드라인을 적절히 준수하고 있는지 파악해 볼 수 있다.

3.3.2 모델 정제에 대한 가이드라인

COMET방법론에서는 소프트웨어 개발 프로세스에서 한 단계에서의 모델 요소가 다음 단계에는 어떻게 정제가 가능한지에 대한 가이드라인을 정의하고 있다. 그 예로, COMET에서는 Object Structuring단계의 객체가 Design Model의 Task Structuring단계에서 어떻게 정제될 수 있는가에 대해서 미리 정의하고 있다. 표 5에서 그 예를 볼 수 있다.

그림 5와 같이 Object Structuring단계의 state dependent control의 속성을 가지고 있는 'RNS Control' 객체는 Task structuring단계에는 '주행Control'이름을 가진 control혹은 control clustering객체로 정제 될 수 있다.

표 5 Object Structuring단계와 Task Structuring단계의 정제가능 객체 Mapping

| Object Structuring | Task Structuring |
|---|---|
| state dependent control | control |
| | control clustering |
| coordinator | coordinator |
| | sequential clustering |
| business logic | asynchronous business logic |
| | periodic business logic |

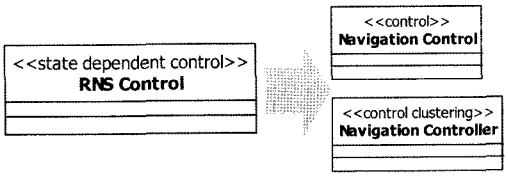



그림 5 모델 정제에 대한 가이드라인 사례

모델 요소는 프로젝트가 진행 될수록 자세히 정제된다. 그런데, 나중 단계의 모델 요소가 이전 단계의 어떤 모델 요소로부터 정제가 되었는지는 설계자가 기술해 주지 않으면 나중 단계의 모델 요소의 기원(origination)을 알 수 없다. 그래서 본 논문에서는 StarUML[15]과, UML Profile[4]을 사용하여 모델링 도구를 모델 요소의 기원에 대해 기술 할 수 있도록 확장하였다. 모델 요소의 기원은 모든 모델요소의 'origination'이라는 Tagged Value를 활용하였으며, 설계자는 여기에 해당 모델요소의 기원을 기입하면 된다.

모델요소의 기입은 다음과 같은 형식을 따라 기입하도록 하였다. 즉 Phase의 이름을 나열하고 Phase

```
(/Phase Name/)*Class Name
```

간 의 구분은 '/'로 하며 제일 마지막에 Class이름을 기입하면 된다. 예를 들면 다음과 같다.

```
/2.AnalysisModeling/2.1.StaticModeling/2.1.4.ObjectStruct
uring/Collision Detection
```

이것을 규칙으로 표현하기 위해서는 하위 룰 모델 요소의 기원('origination')을 찾고, 그것이 상위 룰 모델의 이름과 일치하는지 파악하면 정제가 정확히 되었는지 파악해 볼 수 있다. 이를 위해서 본 프레임워크에서는 모델 요소의 이름을 얻어오는 사용자 정의함수 'Name'을 지원하였다. Name함수는 모델 요소를 Name함수의 파라 미터로 입력해주면 그 결과로 모델 요소가 갖는 이름을 반환해준다. 다음은 모델 정제에 대한 가이드라

인 위배사례를 체크하는 룰이다.

```
Boolean DS_TS_Clustering정제체크룰()
{
  IF( (Model.Phase(name="4.Design").Phase
  (name="4.1.TaskStructuring").Class
  (stereotype="control clustering").origination
  ==
  Name(Model.Phase(name="2.AnalysisModeling").
  Phase(name="2.1.StaticModeling").Phase
  (name="2.1.4.ObjectStructuring").Class
  (stereotype="state dependent control").name)) )
  THEN( true )
  ELSE( false )
  RESULTINFO("Task Structuring단계의 control
clustering은 Static Modeling단계의 state dependent
control로 부터 정제되어야 합니다. 현재 control
clustering객체" + Model.Phase(name="4.Design").Phase
(name="4.1.TaskStructuring").Class(stereotype="control
clustering").name + "은 " + Model.Phase(name=
"4.Design").Phase(name="4.1.TaskStructuring").Class(stere
otype="control clustering").origination + " 으로부터
정제되어 있습니다.")
}
```

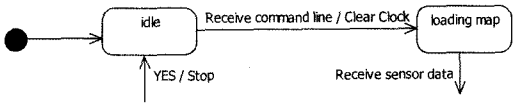
이 예는 Task Structuring단계에서의 모델 요소가 Object Structuring단계의 모델 요소로부터 올바르게 정제가 되었는지 체크하는 룰이다. 그래서 만약 Task Structuring 단계의 control clustering객체의 origination과 Object Structuring단계의 state dependent control이란 스테레오 타입을 갖는 객체의 이름이 서로 동일하다면 true의 결과를 반환하고, 그렇지 않으면 false의 결과와 함께 가이드라인 정보를 반환하게 된다. 다음은 그 가이드라인 위배 정보이다.

```
Result::false
[Rule:DS_TS_Clustering정제체크룰] Task Structuring
단계의 control clustering은 Static Modeling단계의 state
dependent control로부터 정제되어야 합니다. 현재
control clustering객체 주행 control은 /2.AnalysisModeling/
2.1.StaticModeling/ 2.1.4.ObjectStructuring/Collision
Detection 으로 부터 정제되어 있습니다.
```

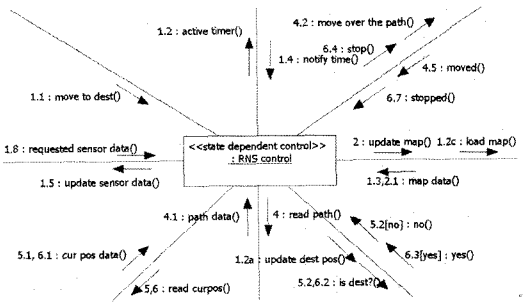
설계자는 이 가이드라인 위배 메시지를 보고, 자신의 모델의 문제점을 개선 할 수 있다.

3.3.3 모델간의 일관성(Consistency)에 대한 가이드라인

모델의 일관성은 모델의 품질에서 상당히 중요한 요소이다. 특히 COMET방법론 에서는 Analysis Modeling의 Dynamic Modeling에서 사용되는 State Chart와 Collaboration다이어그램에서 메시지와 이벤트 간의




(a) State Chart




(b) Collaboration Diagram

그림 6 모델의 일관성에 대한 가이드라인 사례

일관성은 상당히 중요한 사항이다[16].

그림 6은 동기화를 위배하고 있는 예를 보여준다. 먼저 State Chart(a)는 시작과 함께 'Idle'상태로 전이하게 되고 'Receive Command Line'라는 Event를 발생하고 'Clear Clock'이라는 Action을 수행하면서 'loading Map'상태로 전이 하게 된다.

이때 'Receive command line' 이벤트는 Collaboration Diagram(b)내의 'state dependent control'객체의 입력 메시지(예: move to dest, requested sensor data 등) 중 하나여야 한다. 이에 대한 처리를 본 프레임워크를 적용하여 다음과 같이 정의할 수 있다.

```
Boolean SM_SM_CD_SM출력Event동기화체크룰()
{
  IF(
Model.Phase(name="2.AnalysisModeling").Phase(name="2.
2.DynamicModeling").Phase(name="2.2.2.StateCharts").Sta
teMachine.States.State.OutputEvent
      ==
      Model.Phase(name="2.AnalysisModeling").Phase(na
      me="2.2.DynamicModeling").Phase(name="2.2.1.Col
      laborationDiagram").Collaboration.Message(Receiver
      =Model.Phase(name="2.AnalysisModeling").Phase(n
      ame="2.1.StaticModeling").Phase(name="2.1.4.Objec
      tStructuring").Class(stereotype="state dependent
      control").name).name )
  THEN( true )
  ELSE( false )
  RESULTINFO(
    "Static Modeling의 State Chart에 있는 State의
출력 Event와 Collaboration다이어그램에서 Receiver의
메시지가 동일해야 합니다." + IsIncluded( Model.Phase
```

```
(name="2.AnalysisModeling").Phase(name="2.2.Dynamic
Modeling").Phase(name="2.2.2.StateCharts").StateMachine.
States.State.OutputEvent,Model.Phase(name="2.AnalysisM
odeling").Phase(name="2.2.DynamicModeling").Phase(nam
e="2.2.1.CollaborationDiagram").Collaboration.Message(Re
ceiver=Model.Phase(name="2.AnalysisModeling").Phase(n
ame="2.1.StaticModeling").Phase(name="2.1.4.ObjectStruct
uring").Class(stereotype="state dependent
control").name).name )
    )
}
```

각각의 State Chart의 Event에 대해서 Collaboration 다이어그램 내의 state dependent control클래스에게 입력되는 Message의 이름과 동일한지 파악하여, 가이드라인에 위배된 사항을 다음과 같은 방식으로 보여준다.

```
Result::false
[Rule:2_SM_SM_CD_SM출력Event동기화체크룰]Receive
sensor data은 [path data, moved, no, move to dest,
requested sensor data, notify time, stopped, yes, map
data, cur pos data]에 없습니다.
 Receive command line은 [path data, moved, no, move
to dest, requested sensor data, notify time, stopped, yes,
map data, cur pos data]에 없습니다.
[Rule:2_SM_SM_CD_SM출력Event동기화체크룰]Static
Modeling의 State Chart에 있는 State의 출력 Event와
Collaboration다이어그램에서 Receiver의 메시지가
동일해야 합니다.
```

### 3.4 프레임워크 지원 도구 - GARDIAN

룰 기반 COMET방법론 지원 프레임워크인 GARDIAN은 크게 GUI(Graphical User Interface)부분과 Model Check Framework으로 구분되어 있다. GARDIAN의 전체적인 흐름은 그림 7에서 볼 수 있다.

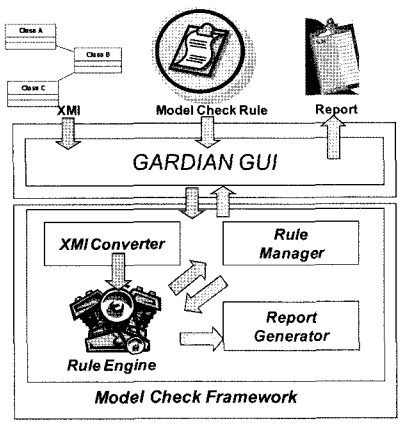



그림 7 GARDIAN의 개념도

GUI부분은 사용자와 상호작용 하는 부분으로, XMI형식의 UML모델을 불러오고, 방법론의 지식을 룰을 사용하여 정의할 수 있는 기능을 제공한다. 또한, Model Check Framework에게 UML모델이 룰을 준수하고 있는지 질의하여, 그 결과를 사용자에게 보여줄 수 있는 기능을 가지고 있다. Model Check Framework는 주어진 XMI기반의 UML모델을 룰을 기반으로 평가하며, 가이드라인위배 보고서를 생성하여 이것을 GUI에게 보내주는 역할을 한다. Rule Engine은 XML과 룰 표현식으로 표현된 룰을 입력 받아, 하나의 룰을 파싱하는 역할을 한다. Rule Manager는 여러 개의 룰을 관리하는 역할을 하며, XMI Converter는 XMI를 Rule Engine이 해석 가능한 XML로 변경하고, Report Generator는 룰 엔진의 수행 결과를 보고서 형식으로 만드는 역할을 담당한다. Model Check Framework중 룰 엔진 부분은 JavaCC[17]를 사용하여 구현되었으며, 다른 부분은 Java언어를 활용하여 구현되었다.

GARDIAN은 그림 8과 같이 가장 왼쪽으로 정의한 룰 목록, 중간에는 룰을 편집할 수 있는 편집 창, 그리고 제일 오른 쪽에 위치한 모델 경로 도우미(Model Navigation Helper)로 이루어져 있다. 이 도구를 통해 룰 목록 관리, 단위 룰 보기와 실행, 모델 정보 보여주기, 전체 룰 실행이 가능하다. 본 프레임워크가 대상으로 하는 것이 모델을 평가하는 것이기 때문에 룰을 정의할 때 모델의 경로를 찾는 일은 아주 빈번한 일이다. 그래서 본 논문에서 제안하는 도구에서는 모델의 경로를 쉽게 찾기 위해서 모델 경로 도우미(그림 8의 오른쪽 아래)를 제공하고 있다.

이 지원 도구에서는 단위 룰과 전체 룰을 모두 실행할 수 있는 기능을 제공하고 있다. 단위 룰의 실행은 전문가가 모델을 기반으로 룰을 정의할 때, 그것이 적절하

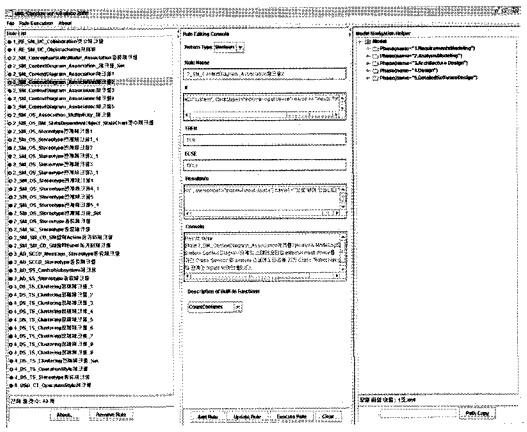


그림 8 UML Checker 도구 화면

게 정의되어 있는가를 파악하기 위한 용도로 유용하다. 또한 전체 룰 실행은 저장된 룰을 기반으로 모델을 평가하기 위한 용도로 활용 될 수 있다.

## 4. GARDIAN 프레임워크 적용 사례

본 장에서는 GARDIAN 프레임워크가 정의된 지식에 따라 설계자의 UML모델을 적절히 평가하는지 검증하기 위해서 본 프레임워크를 지능형 서비스 로봇 시스템[18]에 적용한 사례를 살펴본다. 지능형 서비스 로봇은 다양한 공학 분야의 전문가들이 함께 작업하여 만들어야 하는 복잡한 시스템 중 하나이다. 우리는 [18]에서, 서비스 로봇의 다양한 서비스 중 주행 시스템을 COMET방법론을 사용하여 서비스 로봇의 통합을 시도하였다. COMET방법론을 사용하여 서비스 로봇을 통합하기 위해서는 COMET방법론의 교육은 필수적인 것이며, 설계자가 방법론의 가이드라인을 적절히 준수하고 있는지의 평가는 상당히 큰 이슈 중 하나이다.

### 4.1 검증 방법

본 프레임워크는 이런 이슈를 해결하기 위하여 제안되었으며, 프레임워크를 검증하기 위하여 로봇 주행 시스템을 대상으로 하여 "프레임워크를 통한 방법론 위배 체크와 전문가의 방법론 위배 체크와의 비교 실험"을 수행하였다. 적용 대상은 로봇 시스템에 대해서 약 1주 동안 교육을 받고, 약 3개월 동안 COMET교육을 받은 54명의 COMET비전문가가 7개의 팀이 되어 수행한 모델을 그 대상으로 하였다. 로봇 주행 중 모델의 대상은 "지능형 로봇의 주행 및 충돌장애물 회피 모델링"[18]이며, 산출물의 규모는 약 2개의 Use Case와 10개의 다이어그램이다.

프레임워크가 설계자의 UML모델의 결점을 적절히 식별하는지 파악하기 위해서 COMET을 사용하여 로봇 주행시스템을 수행해 본 경험이 있는 COMET전문가가 프레임워크에 약 43개의 룰을 사전에 정의·입력하였고, 7개 팀의 모델링 결과를 본 프레임워크에 입력하여 그 가이드라인 위배 여부를 파악하였다. 그리고 이 결과와 비교 하기 위하여, COMET방법론 전문가가 모델링을 평가하였다.

### 4.2 검증 결과 및 분석

그림 9는 프레임워크와 전문가의 평가를 비교한 결과이다. 룰은 계층적 구조를 가질 수 있으므로, 총 43개의 룰 중 19개의 상위 룰을 호출 하면 전체 룰이 모두 호출 된다. 그러므로, 수행 결과는 19개의 룰을 호출한 결과와 전문가가 평가한 결과를 보여준다. 프레임워크와 전문가 모두 평균 10.57개의 가이드라인 위배 사항을 체크하였으므로, 평가의 양적인 측면에서 전문가와 동일한 결과가 나왔다.

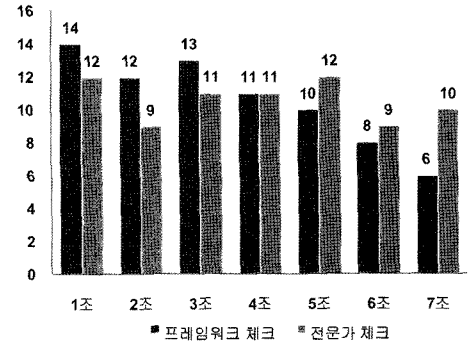



그림 9 프레임워크와 전문가의 모델 평가 비교

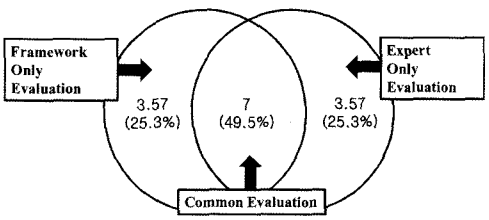



그림 10 프레임워크와 전문가의 모델 평가 비교

그림 10에서는 평가의 내용을 파악하여, 프레임워크와 전문가가 공통적으로 체크한 것과, 프레임워크만 체크한 것, 그리고 전문가만 체크한 결과를 도식하였다. 이 그림에서 볼 수 있듯이 전문가와 프레임워크가 공통적으로 약 50%의 가이드라인 위배를 체크하였으며, 25%는 전문가가 체크하지 못한 것을 프레임워크가 체크하였고, 또 나머지 25%는 전문가가 체크한 것을 프레임워크는 체크 하지 못하였다. 이것으로, 전문가가 단독으로 모델을 평가하는 것보다 약 25%정도 프레임워크가 더 도움을 줄 수 있음을 알 수 있다.

평가항목의 세부내용을 파악해 보면, 전문가가 중점적으로 평가하는 항목은 기본적인 COMET/UML의 표기법과 함께, COMET에서 제시하는 여러 개념에 따라 객체가 적절히 식별되었는가, 혹은 정제가 제대로 되었는가를 중점적으로 평가하였다. 그리고 본 논문에서 제안하는 프레임워크가 평가하는 모델은 COMET개념에 따라 UML표기법이 올바르게 표기되었는가를 중점적으로 평가하였다. 그래서, 모델을 평가할 때 방법론의 전문가는 의미론적인 객체 식별이나 정제의 평가에 중점을 두고, 프레임워크는 UML표기법을 중점적으로 평가하는데 사용하면 유용할 것이다.

## 5. 결 론

많은 방법론에서는 UML을 기반으로 하여 소프트웨어를 분석·설계하며, 이 결과물을 기반으로 소프트웨어

를 구현하게 되므로 UML기반의 모델은 목표 소프트웨어의 품질과도 직결될 정도로 중요하다. 그러나 방법론에서 모델에 대한 가이드라인은 자연어로 기술되어 있기 때문에, 모델이 가이드라인을 적절히 준수하고 있는지 파악 하기 위하여 사람이 직접 확인하는 방법을 사용하고 있다. 하지만 이러한 방법은 여러 전문가가 필요할 뿐만 아니라, 사람마다 기준이 다를 수 있다는 문제점을 가지고 있다.

본 논문에서는 방법론 중에 실시간 내장형 시스템 분석에 사용되는 COMET방법론을 대상으로, 전문가 시스템을 사용하여 방법론의 가이드라인을 기술할 수 있는 프레임워크를 제안하였다. 이 프레임워크를 기반으로 전문가는 가이드라인을 기술할 수 있으며, 다양한 프로젝트의 산출물인 모델이 가이드라인을 적절히 준수하고 있는지 평가할 수 있고, 가이드라인의 위배사항에 대해서 설계자에게 적절한 지시를 할 수 있다. 이렇게 모델에 대한 가이드라인의 위배사항을 알려줌으로써, 소프트웨어 개발 프로세스에서의 산출물의 품질을 높일 수 있으며, 최종 품질을 소프트웨어의 품질도 높일 수 있다.

향후에는 보다 많은 전문가의 협조를 구하여 현재보다 많은 룰을 입력하고, 프레임워크를 개선할 예정이며, 보다 규모 있는 프로젝트에 적용하여 프레임워크의 유용성을 검증하려 한다.

## 참 고 문 헌


[1] Jacobson, Booch, Rumbaugh, The United Software Development Process, Addion-Wesley, 1999.

[2] John Cheesman, John Daniels. UML Components, Intervision, 2000.

[3] H. Gomaa, Designing Concurrent, Distributed, and Real-Time Application with UML, Addison-Wesley, 2000.

[4] Unified Modeling Language OMG Homepage http://www.omg.org/technology/documents/formal/uml.htm, 2006.

[5] Tibor Farkas, Christian Hein, Tom Ritter. "Automic Evalutation of Modelling Rules and Design Guidelines," The Second Workshop "From Code Centric to model Centric Software Engineering: Practices, Implications and ROI," July 11th, Bibao Spain, 2006.

[6] http://argouml.tigris.org, 2006.

[7] N.E. Fenton and S.L. Pfleeger, "Software Metrics: A Rigorous and Practical Approach," Int'l Thomson Computer Press, 1996.

[8] Software Engineering Body of Knowledge, IEEE Computer Society, 2004.

[9] Software Architecture Analysis Tool(SAAT) http://www.laquso.com, 2006.

[10] IBM Rational Rose Homepage http://www-306.ibm.com/software/awdtools/developer/rose/

[11] Christian F.J. Larnge, Michel R.V. Chanudron, Johan Muskens. In Prictice:UML Software Architecture and Design Description. IEEE Software. 2006.

[12] XMI OMG Homepage http://www.omg.org/ technology/xml/index.htm, 2007.

[13] Joseph c. Giarratano, Gary D. Riley Expert Systems principles and programming. Thomson course technology, 2005.

[14] ISO/IEC 14977:1996:Information technology - Syntactic metalanguage - Extended BNF. International Organization for Standardization, 1996.

[15] StarUML homepage, http://staruml. sourceforge.net/ko, 2006.

[16] Joh Whittle, Paveen K. Jayaraman, "Generating Hierarchical State Machines from Use Case Charts," IEEE International Conference on Requirement Engineering, 2006.

[17] JavaCC https://javacc.dev.java.net, 2006.

[18] Minseong Kim, Suntae Kim, Sooyong Park, Mun-Teak Choi, Munsang Kim, Hassan Gomaa. "UML-Based Service Robot Software Development: A Case Study," ICSE, 2006.



김 순 태

2003년 중앙대학교 컴퓨터학과 졸업(학사). 2003년~2004년 (주) 소프트웨어 크패프트, 선임 컨설턴트. 2007년 서강대학교 컴퓨터학과 졸업(석사). 2007년~현재 서강대학교 컴퓨터학과 박사과정. 관심분야는 아키텍처, 객체지향 방법론, 소프트웨어 프로덕트 라인, 임베디드 시스템

김 진 태

1998년 서강대학교 컴퓨터학 학사. 2000년 서강대학교 컴퓨터학 석사. 2001년 데이콤시스템 테크놀로지 소프트웨어 엔지니어. 2005년 서강대학교 컴퓨터학 박사 현재 삼성전자 정보통신연구소 책임연구원. 관심분야는 Empirical software engineering, 요구공학, 아키텍처, 리엔지니어닝, 소프트웨어 프로덕트 라인

박 수 용

정보과학회논문지 : 소프트웨어 및 응용 제 34 권 제 4 호 참조