자율 컴퓨팅을 위한 OCL 기반 소프트웨어 결함 분석

장일규*, 유길종**, 이은석**
성균관대학교 컴퓨터공학과*
성균관대학교 전자전기컴퓨터공학과**
e-mail : ilgyu.jang@gmail.com, gjyoo@ece.skku.ac.kr, leees@skku.edu

OCL based Software Fault Analysis for Autonomic Computing

Ilgyu Jang*, Giljong Yoo**, Eunseok Lee**
Department of Computer Engineering, Sungkyunkwan University, Suwon, 440-746, Republic of Korea*
Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, 440-746, Republic of Korea**

요 약

현대의 시스템들은 매우 복잡해지고 있으며 그 규모 또한 점점 커지고 있다. 또한 기술의 발달로 서로 상호작용하는 시스템들이 많아졌으며, 비즈니스적 요구 사항의 증가로 인해 통합되는 시스템들도 많아지고 있다. 시스템의 유지 보수는 비용과 노력 등 시간이 갈수록 고객과 개발자에게 더 많은 부담이 되고 있다. 때문에 자율 컴퓨팅 시스템이 제안되었다. 자율 컴퓨팅 시스템은 시스템 스스로가 자신을 관리하는 자가 적응하는 시스템이다. 이러한 자율 컴퓨팅이 이루어 지기 위해서는 데이터베이스화 된 시스템의 결함 정보, 각 결함에 따른 하나 이상의 전략이 존재하여 적절한 선택을 할 수 있도록 하여야 한다. 본 논문은 OCL 을 기반으로 하여 설계 단계의 시스템 모델에서 예상 가능한 결함 정보를 추출하여 결함 리스트를 생성하는 연구를 내용으로 한다. 이를 통하여 개발자는 효율적으로 자가 적응 시스템을 구축할 수 있다. 그리고 연구의 효율성을 검증하기 위해 ATM 시스템을 대상으로 평가를 한다.

1. 서론

시스템 개발과 유지보수는 많은 비용과 엔지니어의 노력을 요구한다. 특히, 시스템의 개발부터, 유지보수, 폐기까지 이어지는 소프트웨어 공학의 주기에서, 유지보수에 들어 가는 비용과 노력의 비중은 매우 크다. 레거시 시스템(Legacy System) 과 같이 시스템의 생명이 매우 오랜 기간 유지되는 경우에는 유지보수의 비중이 더욱 커질 수 밖에 없다. 유지보수는 단순히 시스템이 고장 없이 작동하도록 하는 것만을 말하지 않는다. 예상치 않았던 고장에 대처, 새로운 컴포넌트 추가, 최신 버전으로 업데이트 등 항상 시스템의 성능을 최상으로 유지하기 위한 노력이 요구된다. 변화가 빠른 IT 산업에서 이러한 노력들은 매우 중요하고 신속하게 이루어져야 하는 일들이다[1].

유지보수는 시간이 지날수록 점점 더 어려워지는 작업이다. 유지보수가 오랜 기간 이루어지다 보면 시스템은 복잡해지고 점차 초기의 명확함은 많이 사라지게 된다. 그리고 비즈니스적 요구사항으로 인하여 시스템 확장, 상호 작용, 그리고 통합을 하게 되기도 한다. 고객의 요구가 서서히 변하면서 시스템도 서서히 변하기도 한다. 개발자마다 가지고 있는 구현 습관에 따른 사소한 차이가 존재하여 유지 보수에 어려움을 겪을 수도 있다. 이 외에 많은 요인들이 유지보수에 어려움을 증가시키는 이유가 될 수 있다.

이러한 어려움들을 해소하기 위해 IBM 은 시스템 스스로가 자신을 관리하여 사람의 관여를 최소로 하는 자율 컴퓨팅 시스템(Autonomic Computing System)을 제안하였다[1]. 또한, David Garlan 은 레인보우 프레임워크(Rainbow Framework)를 제시하여 자가 적응(Self-Adaptation)하는 자율 컴퓨팅 시스템을 제안하였다[2].

이러한 시스템들은 결함에 따라 전략을 선택하여 자가 적응을 하기 위한 결함 정보가 필요하다. 하지만 시스템이 개발된 초기에는 이러한 결함 정보의 양이 없는 상태에서 시작하여 점점 쌓아 나가야 한다. 시스템 유지 보수의 초기에 많은 비용과 노력이 들 수 밖에 없다.

본 논문에서는 UML(Unified Modeling Language)를 사용하여 시스템을 설계하는 과정에서 OCL(Object Constraint Language)를 적용하여 시스템 설계 단계에서부터 자가적응을 적용시키는 방법에 대하여 연구한다. 이를 통하여 개발자는 자가적응 시스템을 구축함에 있어서 런타임에 발생하는 결함에 대한 해결 부담을 줄이고 결함 정보를 미리 확보하여 효율적으로 자가 적응할 수 있는 시스템을 구축할 수 있다.

본 논문의 구성은 2 장에서 관련 연구를 소개하고 3 장에서 본 연구에서 제안하는 OCL 을 기반으로 한

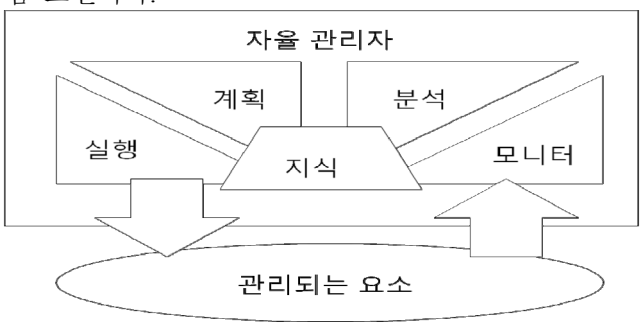
* 이 논문은 2010 년도 정부(지식경제부)의 재원으로 한국전자통신연구원(ETRI)의 지원을 받아 수행된 연구임(10035708)

소프트웨어 결함 리스트 작성에 대하여 소개한다. 4 장에서 사례연구를 통한 평가를 하며 마지막으로 결론과 보완점, 향후 연구 방향을 정리한다.

2. 관련 연구

2.1. 자율 컴퓨팅

자율 컴퓨팅 시스템은 시스템 스스로가 자신을 관리하는 시스템이다. 자율 시스템은 하나 또는 여러 개의 자율 요소(Autonomic element)로 구성되며, 자율 요소는 그림 1 에서와 같이 자율 관리자(Autonomic Manager)와 관리되는 요소(Managed element)로 구성된다. 관리되는 요소는 비자율 컴퓨팅 시스템의 일반적인 요소들과 같다고 볼 수 있고, 자율 관리자는 자가 적응을 위한 역할을 한다. IBM 은 이를 위하여 MAPE-K 모델을 제안하였다. MAPE-K 는 자가 적응을 위해 공동으로 활용되는 지식(knowledge)를 기반으로 모니터(Monitor), 분석(Analysis), 계획(Plan), 실행(Execute) 4 가지의 루프로 구성되는, 자율 컴퓨팅 시스템 모델이다.



(그림 1) 자율 요소의 구성

2.2. 소프트웨어 구조 기반 자가 적응 시스템

자가 적응 시스템은 모델을 이용하여 자가 적응을 한다. 시스템은 시스템 모델과 시스템 환경 모델을 가지고 있다. 이 모델들은 시스템의 상태를 파악하고 적응 전략을 세우는 기반이 된다.

2.2.1. 레인보우 프레임워크(Rainbow Framework)

David Garlan 은 자가 적응하는 시스템으로 레인보우 프레임워크를 제안하였다[2].

레인보우 프레임워크는 프로브, 번역기, 게이지, 모델 관리자, 구조 평가자, 적응 관리자, 전략 수행자, 이펙터로 구성된다. 프로브들이 수집한 시스템 정보는 번역기를 지나면서 모델 관리자가 가지고 있는 모델에 적용될 수 있도록 변환되고 게이지는 이 정보를 모델에 적용한다. 구조 평가자는 모델을 평가하여 시스템의 상태를 파악한다. 결함이 발생하면 적응 관리자에 의하여 적응 전략이 선택되고 전략 수행자가 이를 수행한다. 전략은 번역기를 지나면서 실제 시스템에 적용될 수 있도록 바뀌고 이펙터들이 이들을 실제 시스템에 적용하여 자가 적응이 이루어진다.

레인보우와 같은 자율 컴퓨팅 시스템들은 구현 단계에서부터 자가 적응이 이루어 진다. 하지만 사소한 결함이라도 그 결함이 처음 발생하였을 때는 해결에 어려움을 겪게 된다. 설계 단계에서 결함 정보들을 미리 파악하여 결함에 대한 적응 전략을 세우고 구현하면 이러한 부담을 줄일 수 있다.

2.2.2. Constraint checking

모델링은 소프트웨어 개발의 중요한 과정이다. 모델은 정형화된 형식 또는 언어로 표현되는 시스템의 표현이다. 정형화된 형식은 모델링 방법 패러다임(예, 객체지향, 절차지향)을 바탕으로 한다[3]. UML 은 유의미한 해석(meaningful interpretation)을 위해 OCL 을 제공한다.

모델은 해석이 이루어지는 레벨에 따라 가능한 해석의 양이 다르다. 해석 레벨은 제약사항의 범위에 의해 결정된다. 가장 다양한 해석을 가지는 모델은 패러다임 레벨에 존재하고 formalism 에 관한 제약사항으로 나타내어 지며 변하지 않는다. 여기에 의미 제약사항, 스테레오 타입 제약사항(패러다임 확장 레벨), 스타일 가이드 제약사항(모델링 프로세스 레벨) 등이 추가되면서 모델의 해석 가능한 결과는 점점 줄어든다. 즉, 해석에 대한 애매모호함이 줄어들게 된다. 모델링 된 영역에서의 제약사항은 현실 세계 영역의 제약사항들을 만족하는 모델을 생성한다. 마지막으로 구현 영역에서의 제약사항(예: balance > 0)은 유사한 클래스들간의 동일한 해석을 가능하게 해준다.

2.3. OCL(Object Constraint Language)

기존의 UML 을 사용한 객체 지향 모델링은 구체적이고 명확한 명세를 하기에는 충분하지 않았으며 특정 인스턴스에서의 제약 정보를 나타내지 못하였다. 이러한 추가적인 제약 정보를 기술할 방법이 필요했지만, 자연 언어는 애매모호한 문제가 있었기 때문에 명확한 제약 정보를 명세하는 데는 부적합하였다. 따라서, OMG(Object Modeloing Group)에 의해서 정형 명세 언어인 OCL 이 제안되었다[4].

OCL 은 시스템의 제약 정보를 UML 상에 명세할 수 있도록 하는 언어이다. UML 상에서 태그(tag) 형태로 기술되며 모델의 어떠한 것에도 영향을 주지 않는 순수 표현 언어이다. 또한 정형 명세 언어이기 때문에 애매모호성이 없으며, 각 표현식은 타입을 가지는 타입 언어이다.

OCL 은 클래스의 객체에 불변조건, 사전조건, 사후조건 등의 설정이 가능하다. 불변 조건은 항상 참이 되어야 하는 조건, 사전/사후 조건은 어떤 처리의 전후에 반드시 성립되어야 하는 조건이다.

OMG 는 현재까지 OCL 2.2 버전까지 발표하였다.

3. OCL 기반 소프트웨어 결함 분석

본 논문에서 제안하는 방법은 OCL 을 사용하여 제약 정보를 모델에 삽입하고 삽입된 OCL 을 파싱하여 시스템 설계단계에서부터 결함 모델을 파악한다. 미리 결함 모델을 파악하고 적응전략을 구현함으로써 런타임에 발생하는 결함을 최소화하고 효율적인 개발을 가능하게 한다.

3.1. 시스템 모델링

제약 정보를 표현하는 OCL 은 시스템 모델에 메모처럼 첨부되어 각 요소의 오퍼레이션이나 성분 등의 제약 조건을 나타낸다. 때문에 시스템 설계 단계에서 잘 만들어진 시스템 모델을 생성해야 한다.

3.2. 제약 정보의 표현

결함이란 시스템 오류를 초래할 수 있는 사람의 실수나 시스템에 존재하는 특징이다. 결함은 항상 오류로 유발되지는 않지만 잠재적인 원인으로 존재하여 시스템을 불안정하게 만든다.

제약이란 특정 문제를 해결하는데 설정되는 조건이다. 프로그램에서 제약은 어떤 객체의 값 또는 상태에 대하여 범위나 조건을 부여해서 사용 가능한 영역을 제한하는 것이다. 제약 정보에 의해 제한된 영역을 벗어나게 되면 이는 시스템 결함을 유발하게 된다.

본 논문에서는 시스템의 제약 정보를 삽입하기 위해 표 1 과 같이 OCL 의 일부 키워드를 사용한다.

<표 1> OCL 키워드

(a) OCL 키워드들 중 일부 키워드

context	OCL 의 처음 부분에 기술될 수 있는 키워드로 특정 유형의 인스턴스를 나타낸다. 패키지, 클래스, 인터페이스, 컴포넌트, 오퍼레이션, 애트리뷰트 등이 될 수 있다.
inv	불변식으로 특정 인스턴스에서 항상 true 가 되어야 하는 조건
pre	어떤 실행 전에 항상 성립되어야 하는 조건
post	어떤 실행 후에 항상 성립되어야 하는 조건
init	속성이나 연관관계의 초기값을 나타내기 위해 사용
derive	속성이나 연관관계의 상속된 값을 나타내기 위해 사용
self	context 의 인스턴스를 참조하기 위해 사용
result	특정 실행의 결과값을 나타내기 위해 사용

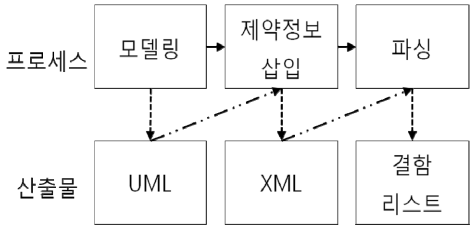
(b) 간단한 OCL 사용 예제

context Person inv: self.age>18	Person 의 age 라는 속성은 항상 18 보다 커야 함
context Person::price():Integer pre: self.age>=18 post: result<5000	Person 의 price()가 실행되기 전에 age 는 18 이상이 되고 실행 후에는 그 결과가 5000 보다 작아야 함
context Person::money:Integer init: 0	Person 의 money 라는 속성은 초기값이 0

본 논문에서는 결함 리스트를 추출하기 위해서 UML 을 사용하여 시스템을 표현한 모델에 OCL 을 이용하여 제약 정보를 삽입한다. 그리고 파서를 통해 OCL 을 분석하여 결함 정보를 추출한다.

3.3. 결함 리스트 추출 및 분석

그림 2 는 본 논문에서 제안하는 결함 리스트 추출 과정을 나타낸 그림이다. 모델링 과정에서 얻어진 모델에 OCL 로 제약 정보를 삽입한 후 모델 정보를 XML 로 추출한다. XML 로 잘 정돈된 정보들에서 OCL 정보를 파싱하여 결함 리스트를 추출하게 된다.



(그림 2) 결함 리스트 추출 과정 및 산출물

4. 평가

본 논문에서 제안한 OCL 기반 소프트웨어 결함 분석을 평가하기 위해, UML 을 기반으로 간단한 ATM 시스템을 모델링한다. 그리고 OCL 을 사용하여 모델에 시스템의 제약 정보를 삽입한다. 삽입된 정보는 XML 파일로 추출되고 XML 파일을 파싱하여 OCL 을 추출하고 이를 통하여 시스템의 제약 정보를 파악하여 결함 모델을 생성한다.

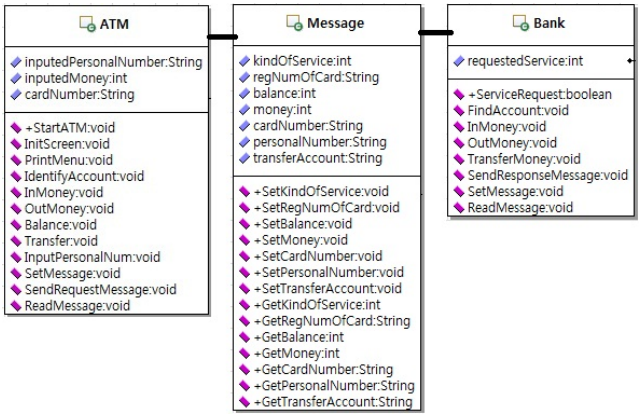
본 실험에는 OS 는 Windows 7 Ultimate K, CPU 는 AMD 애슬론 64 X2 듀얼 코어 프로세서 2.80GHz, RAM 은 4Gbyte 성능의 데스크톱 PC 를 사용하였다. 모델링 툴은 starUML 을 사용하였다.

4.1. ATM 모델링

ATM 의 일부 요구사항은 다음과 같다.

- 카드번호로 계좌를 조회
- 입금, 출금, 계좌 이체, 잔액 조회 기능

이를 바탕으로 그림 3 과 같은 모델을 도출하였다.



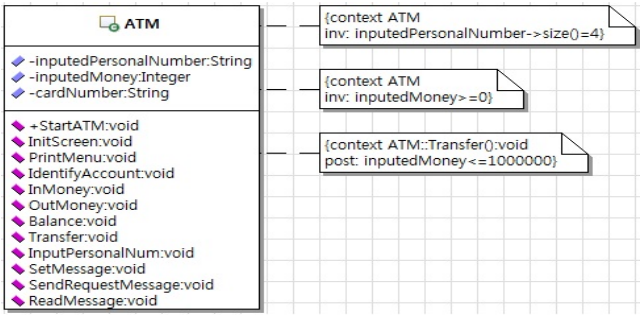
(그림 3) ATM 시스템의 클래스 다이어그램

4.2. 설계 모델 내에 제약 정보 삽입

ATM 시스템에서 발생할 수 있는 결함을 일부 선정하였다.

- 사용자가 입력하는 금액이 음수
- 비밀번호는 반드시 4 자리로 구성
- 이체 시 상한 금액보다 큰 금액 입력(100 만원)

그림 4 는 이들을 OCL 로 작성하여 모델에 삽입한 것이다.



(그림 4) ATM 클래스에 제약사항 삽입

4.3. 모델 해석 및 결함 정보 분석

제약정보가 삽입된 모델을 XML 로 추출하면 그림 5 와 같이 제약정보가 XML 에 표현된다.

```
- <UML:Constraint xmi.id="X.38" name="PersonalNumber size constraint" visibility="public" isSpecification="false"
    constrainedElement="UMLClass.5">
  - <UML:Constraint.body>
      <UML:BooleanExpression xmi.id="X.39" body="context ATM inv: inputedPersonalNumber->size() = 4" />
    </UML:Constraint.body>
  </UML:Constraint>
- <UML:Constraint xmi.id="X.40" name="inputedMoney value constraint" visibility="public" isSpecification="false"
    constrainedElement="UMLClass.5">
  - <UML:Constraint.body>
      <UML:BooleanExpression xmi.id="X.41" body="context ATM inv: inputedMoney >= 0" />
    </UML:Constraint.body>
  </UML:Constraint>
- <UML:Constraint xmi.id="X.42" name="inputedMoney maximum constraint" visibility="public" isSpecification="false"
    constrainedElement="UMLClass.5">
  - <UML:Constraint.body>
      <UML:BooleanExpression xmi.id="X.43" body="context ATM::Transfer():void post: inputedMoney <= 1000000" />
    </UML:Constraint.body>
  </UML:Constraint>
```

(그림 5) ATM 클래스에 제약사항 삽입

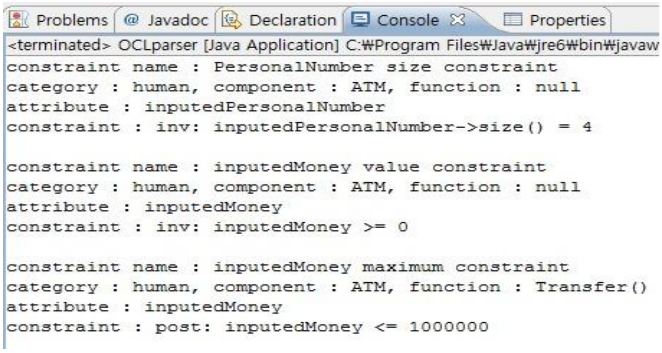
XML 을 파싱하여 OCL 들을 찾아서 그 정보를 읽어내면 어느 클래스에 어떤 속성이나 오퍼레이션에 어떤 제약 조건이 있는지를 알 수 있다. 이를 가지고 결함 리스트를 생성하게 된다.

결함 리스트는 표 2 와 같은 여러 가지 유형의 정보를 포함한다. 이들 정보들을 나열하여 그림 6 과 같은 결함 리스트를 생성하게 된다.

<표 2> 결함 정보의 유형

constraint name	제약 정보의 이름
category	결함의 원인에 따른 분류 (예 : Human, Software, Hardware, External Attack)
component	결함이 발생한 요소(예 : 클래스)
function	요소 내에서 결함이 발생한 function
attribute	요소 내에서 결함이 발생한 성분
constraint	지켜져야 할 제약 사항

본 논문에서 제안한 방법은 설계 단계에서 결함 정보를 파악하여 해결하기 때문에 효율적으로 자가 적응 시스템을 개발할 수 있다. 그리고 설계 단계에서부터 쌓인 지식베이스가 있기 때문에 자가 적응 시스템의 효율적인 유지 보수도 가능하다. 또한, 제약 정보는 비기능적 요구사항이라고 볼 수 있는데 설계 단계에서 OCL 을 이용함으로써 모델에 이를 표현하기 때문에 더 효과적이고 통합된 시스템의 표현이 가능하다.



(그림 6) 결함 리스트

5. 결론

본 논문은 효율적인 자가 적응 시스템 개발과 유지보수를 위한 OCL 기반 시스템 결함 리스트 생성을 제안하였다. 이를 통해 시스템의 설치된 후가 아닌 설계 단계에서부터 자가 적응을 위한 지식 베이스를 쌓고 대응 전략을 구현할 수 있다. 이는 자가 적응 시스템의 효율을 한층 높여 줄 수 있다.

OCL 이 모델을 한층 더 명확하게 해주는 것은 사실이지만 인스턴스 단위의 값의 범위, 관계, 상태 등 여러 제약들을 나타내기 때문에 난해함을 가지고 있다. 또한 OCL 은 모델에 첨가되는 정보이기 때문에 모델이 얼마나 시스템을 잘 나타내는가도 중요하다. 이러한 이유로 숙련된 개발자를 필요로 하게 된다.

본 논문의 향후 연구로서 완전한 자동화와 결함의 연관관계 분석을 수행하고자 한다. 그림 6 에 사용된 결함의 정보들은 시스템의 특성에 따라 추가될 수도 있다. 하지만 OCL 을 추출하여 명백하게 알 수 있는 것은 component, function, attribute, constraint 이다. 나머지 정보들을 추출하기 위해서는 전문가가 개입되거나 새로운 방법이 개발되어야 한다. 본 논문은 결함을 독립적으로 취급하였다. 하지만 결함은 여러 요소가 연관되어 나타나는 경우도 많다. 이를 설계 단계에서 파악하기 위해서는 결함들 간의 연관 관계를 분석하는 방법이 필요하다.

참고문헌

[1] Jeffrey O. Kephart and David M. Chess, “The Vision of Autonomic Computing”, IEEE Computer, pp.41-50, Jan., 2003

[2] Garlan D, Schmerl B, and Cheng S.W, “Software Architecture-Based Self-Adaptation”, Autonomic Computing and Networking, Springer, pp.31-55, May, 2009

[3] Jean Louis Sourrouille and Guy Caplat, “Constraint Checking in UML Modeling”, In Procs. of the 14th international conference on Software engineering and knowledge engineering, pp.217-224, Jul.,2002

[4] Object Constraint Language ver.2.2, Feb., 2010, http://www.omg.org/spec/OCL/2.2

[5] Eoin Woods, “OCL Quick Reference Summary”, Jul., 2005, http://www.eoinwoods.info/doc/ocl_quick_reference.pdf