MDA기반 컴포넌트 설계정보 관리도구의 개발에 관한 연구

안용수*, 황상원*, 남영광*, 이병윤**, 권오천**
*연세대학교 전산학과
**한국전자통신연구원
e-mail: home@yonsei.ac.kr

Development of a tool for managing component model based on Model Driven Architecture

Yong-Soo Ahn*, Sang-Won Hwang*, Young-Kwang Nam*,
Byeong-Yun Lee**, Oh-Cheon Kwon**
*Dept. of Computer Science, Yonsei University
**Electronics and Telecommunications Research Institute.

요 약

MDA(Model Driven Architecture)는 추상적인 모델 계층을 사용하기 때문에 다양한 플랫폼에 적용가능하고, 각 모델 계층과 코드 생성의 자동화를 통해 개발의 효율성을 극대화한다. 본 연구에서는 XML 형태로 저장된 설계정보를 분석하여 MDA 기반 컴포넌트 설계 정보를 관리하는 도구를 개발하였다. 이 도구는 UML로 작성된 설계모델를 XMI(XML Metadata Interchage) 형태로 저장하여 각종 설계도구에서 Java, C++과 같은 언어에 대한 실제 프로그램 골격코드가 자동으로 생성되도록 하였다. 역으로 골격코드를 기반으로 구현된 콤포넌트의 원시코드를 수집하여 다시 컴포넌트 설계모델 정보를 추출하는 기능을 구현하였고, 이를 다시 시각적 정보로 재구성 하였다. 이러한 기능들은 기존의 단방향적 개발 구조 방식에서 벗어나 이미 개발되거나 개발 중인 프로그램에 대한 분석 및 평가 등을 통해서 재사용성을 높여주는 순환적인 개발 구조 방식을 제공한다.

1. 연구 배경

최근 소프트웨어 개발 환경에는 많은 프로그래밍 언어와 기술 플랫폼들이 존재한다. 기술적인 측면에서는 다양한 하드웨어와 운영체제, 네트워크들이 있고, C, C++, Java, C#, 각종 스크립트 언어 등 많은 언어가 존재한다. 그리고 COM, COM+, EJB 등 다양한 프레임워크의 구현 형태가 등장하였고, 지속적으로 요구되는 사용자 요구사항은 소프트웨어 개발에 어려움을 가중시키도 있다.

이런 소프트웨어 개발 환경에서 MDA는 기존의 전통적인 개발 방식과 달리 플랫폼 독립적 모델인 PIM(Platform Independent Model)과 플랫폼 종속적 모델인 PSM(Platform Specific Model)으로 추상화 계층을 분리함으로 플랫폼이나 언어가 바뀌었을 때 PIM을 수정하지 않고도 기능 변화에 신속하게 대응할 수 있고, 모델 단위의 재사용이 가능하다. 즉, 다양한 분야에 대해 동일한 기능 모듈 또는 모델를 개발하는데 있어서 중복적인 개발 노력을 없앨 수 있으며, 개발된 기능들을 인증하고 표준화시킴으로써 경쟁력을 높일 수 있다. 본 연구에서는 이러한 MDA의 장점을 이용하여 XML로 저장된 설계정보를 기반으로 골격코드를 생성하고, 콜격코드를 기반으로 구현된 콤포넌트 원시코드에서 다시 설계정보를 추출하는 시스템을 개발하여 설계와 원시 코스 간에 자연스럽게 연결이 될 수 있도록 하는 시스템을 Ecliple 환경하에서 개발하였다.

2. Model Driven Architecture(MDA)

2.1 MDA의 개념

MDA는 추상화 레벨로써 설계 모델과 구현 모델을 분리하였다. MDA 모델은 기본적으로 플랫폼 독립적으로 기술된 PIM과 플랫폼 종속적인 모델인 PSM로 이루어져있다. PIM은 시스템의 설계와 명세를 정형화된 모델로 기술한다. PSM은 정형화된 변환 법칙을 사용하여 PIM으로부터 생성되며 플랫폼에 종속적인 요구사항들이 포함된 시스템 모델 정보이다. 이 두 모델은 UML로 기술된다.

2.2 MDA의 장점

- 기술 플랫폼 및 기능 변화에 대한 신속한 대응

PIM과 PSM을 분리했기 때문에 PIM을 변경하지 않고도 기술 플랫폼의 변화나 요구사항 변화에 대처할 수 있다. 새로운 플랫폼이 등장하여 해당 플랫폼에 대한 응용을 배포해야 하는 경우 PIM에는 수정이 필요 없다. 단지 PIM을 이용해서 새로운 플랫폼에 대한 PSM을 자동으로 생성하고, 이를 수정하여 코드를 재생성하는 과정을 통해 쉽게 새로운 기술 플랫폼에 대한 대응이 가능하다. 또한 기능의 변화를 요구하는 경우에도 PSM 수준에서 변경을 고려하지 않고 관련된 수정사항을 PIM에 적용하여 쉽게 대응 가능하다.

• 시스템의 지속성

플랫폼 의존적인 시스템은 새로 발생한 요구사항을 기존 아키텍처가 만족시키지 못하는 경우가 많지만, 본 연구에서는 기능과 아키텍처를 분리해서 정의함으로서 아키텍처의 변화가 있더라도 변환 과정을 거쳐 구현 과정으로 쉽게 진행할 수 있다. 이러한 과정에 의해 생명 주기가 연장되며 더 안정된 시스템 유지가 가능하다.

• 개발 생산성 증진

MDA는 모델의 자동화와 변환을 통해 여러 플랫폼을 쉽게 지원하고 코드 작성부분의 시간을 단축시킬 수 있다. 그리고 전체적인 유지보수가 용이하며 자동화에 의해 재사용성이 높아진다.

• 문서 작성이 용이

일반적으로 개념 모델에서부터 구현 모델 그리고 문서까지 동기화를 이루는 형태로 개발을 하면, 디자인 문서 갱신과 관련 코드의 수작업 관리와 같이 시간이 많이 걸리고 번거로운 작업을 줄여 생산성 향상에 도움을 준다.

• 품질 관리 비용의 감소

MDA 모델 자동화와 테스트 도구를 이용하면 개발자들이 응용을 모델 수준에서 테스트할 수 있기 때문에 디자인의 문제점을 쉽게 발견하여 품질 관리에 들어가는 비용을 줄일 수 있다.

• 양질의 시스템 구축

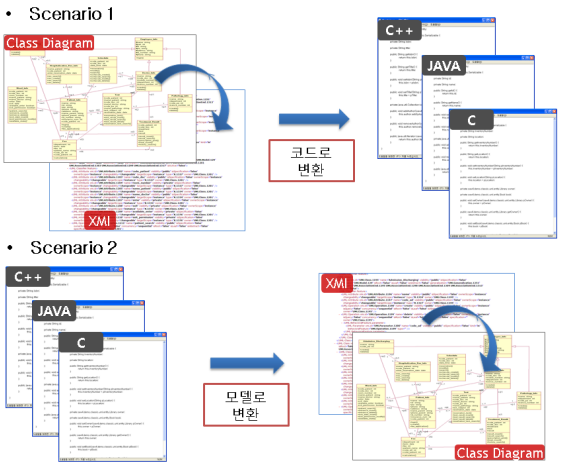
단순한 구조의 PIM을 이용하면 양질의 시스템 구축이 가능하다. 모델링은 팀 구성원들 사이의 의사소통을 원활히 하고 동시에 결점이 있을 때 이를 신속히 제거하도록 도와준다. 또한 MDA의 자동화 도구들은 잘 정리된 코딩 패턴을 모델에 적용하기 때문에 개발자가 직접 작성한 코드에 비해 결점이 적다.

3. MDA 기반 컴포넌트 설계정보 관리 도구

본 연구에서는 MDA 기반의 설계 정보를 저장하고 있는 XMI를 기반으로 하는 “MDA 기반 설계모델 및 코드 생성기”를 이클립스 플러그인 형태로 개발하였다. 이 플러그인은 PSM 거쳐 소스코드를 생성하기 보다는 XMI정보를 기반으로 하는 모델이라는 부분에 초점을 맞추고 있다. 여기서 생성된 정보는 이클립스 MDT와 연결되고, XMI 2.X 버전을 따른다.

이 플러그인은 두 가지 방향성을 가지는 시나리오로 구성되어 있다. 첫 번째는 UML도구로 설계된 설계모델 정보 XMI를 코드로 변환하는 것이다. UML도구를 이용하여 작성된 설계모델 정보는 XMI형식으로 구성하는데, 이 XMI로부터 프로그램 언어(C, C++, Java)의 골격 코드를 생성한다. 이 시나리오는 MDA 과정 중 모델로부터 코드를 생성하는 순방향에 해당되는 부분이다.

두 번째는 저작도구나 로컬 저장소에 존재하는 코드를 읽어 그 코드로부터 컴포넌트 설계모델 정보를 담고 있는 XMI 파일을 생성한다. 이렇게 생성된 XMI는 다시 UML 도구를 이용해 모델의 단위로 재사용된다. 이 부분은 첫 번째 시나리오의 역방향으로, MDA과정 중 코드를 이용하여 모델을 생성하는 부분이다.



[그림 1] MDA 기반 설계모델 및 코드 생성기의 진행시나리오

MDA기반 설계모델/코드 변환 및 관리기는 설계모델로부터 코드를 생성하는 XML Analyzer, Code Generator와 구현코드로부터 컴포넌트 설계정보를 추출하기 위한 Code Analyzer, XMI Generator 그리고 그 중간 매개역할을 하는 Code Template로 이루어져있다.

3.1 코드 생성기

UML형태로 만들어진 설계 모델 정보는 XMI형식으로 저장된다. XML Analyzer는 설계 모델 정보를 재구성하며, Code Generator는 구성된 정보를 이용하여 C, C++, Java등의 해당 프로그램 언어로 골격 코드를 생성한다.

3.1.1. XML Analyzer

XMI는 기본적으로 XML의 형태를 따른다. 이 플러그인에서는 DOM(Document Object Model) 파서를 이용하여 설계 모델에서 필요한 정보를 추출하고 체계적으로 정리한다.

3.1.2. Code Generator

XML Analyzer를 통해 정리되어진 정보를 이용하여 C, C++, Java와 같은 타겟 언어에 대한 골격 코드를 구성한다. 현재 이 플러그인의 골격 코드 구현 범위는 클래스, 멤버 변수(타입, 이름), 함수(리턴타입, 이름, 파라미터)이다.

```
public class CtoX_Java_ActionDelegate {
    public Vector file_vec;
    public IWorkbenchPart targetPart;
```

```
    public IStructuredSelection selection;
    public void setActivePart(IAction action, IWorkbenchPart targetPart) {}
    public void run(IAction action) { }
    public void selectionChanged(IAction action, ISelection selection) { }
    public void recurse(File dirFile, int depth) { }
}
```

[표 1] 생성된 클래스의 예

3.2 모델정보 생성기

통합 개발환경에서 개발중인 프로젝트에 특정 언어의 코드가 적재되어 있는 경우, Code Analyzer는 해당 코드를 분석하여 모델정보 설계에 필요한 정보를 구성한다. 이 정보를 이용하여 XMI Generator는 설계정보를 담고 있는 XMI 형태의 파일을 생성한다.

3.2.1. Code Analyzer

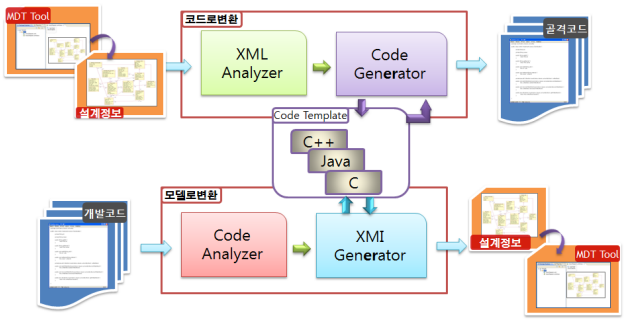
기존에 개발된 소스 파일을 분석하여 구조화된 형태의 코드 정보를 생성한다. 이 코드 분석기는 JavaCC를 이용하여 개발되었다. JavaCC는 Java로 만들어진 다른 파서에 비해 정보분석 수준과 코드에 대한 파싱 신뢰도가 높다. 그리고 추후 설계 모델 정보에 OCL이나 액티비티 다이어그램 등의 정보들이 추가되었을 경우 확장이 용이하여 비용을 절감 할 수 있다.

3.2.2. XMI Generator

Code Analyzer에서 분석된 구조화된 정보를 이용하여 컴포넌트 설계 모델 정보인 XMI를 구성한다.

```
<packagedElement xmi:type="uml:Class" xmi:id="UMLClass7" name="CKlineProcDataFrame">
    <ownedAttribute xmi:id="UMLAttribute.8" name="int color" visibility="protected" aggregation="composite">
    </ownedAttribute>
      ...Attribute...
    <ownedOperation xmi:id="UMLOperation.10" name="void RunModify" visibility="public">
      <ownedParameter xmi:id="UMLParameter.11" direction="return"/>
      <ownedParameter xmi:id="UMLParameter.12" name="int a">
      </ownedParameter>
      <ownedParameter xmi:id="UMLParameter.13" name="bool b">
      </ownedParameter>
    </ownedOperation>
    ...Operator...
</packagedElement>
```

[표 2] 생성된 XMI파일(Class 부분)



[그림 2] 코드 생성기 및 모델정보변환기의 세부 구성도

3.3. XMI의 구성

이 플러그인은 XMI 버전 2.X를 대상으로 하고 있다. XMI요소는 "xmi:id"에 유일한 식별자 값으로 설정된다. 따라서 이 값은 중복되지 않으며 한 예로

<... xmi:id="UMLClass2" name="Class_A" ...>

인 경우 XMI 문서 내에서 "Class_A"를 지칭하는 말은 "UMLClass2"이 된다. 구성요소 별 XMI는 다음과 같고, 각 주요 요소는 XMI를 파싱하거나 XMI를 생성할 때 주요 대상이 된다.

3.3.1. Class

```
<packagedElement xmi:type="uml:Class" xmi:id="UMLClass7" name="CKlineProcDataFrame">
    ...Attribute...
    ...Operation...
</packagedElement>
```

클래스에서 주요 속성은 클래스 아이디 "xmi:id", 클래스 이름 "name"이 있다. 이 클래스는 Attribute와 Operation을 하위요소로 가진다.

3.3.2. Attribute

```
<ownedAttribute xmi:id="UMLAttribute.8" name="int color" visibility="protected" aggregation="composite">
</ownedAttribute>
```

"ownedAttribute" 요소는 속성, 즉 클래스의 멤버변수를 나타낸다. "ownedAttribute"의 주요 속성은 변수 아이디 "xmi:id", 변수 이름 "name", 변수 가시성 "visibility"가 있다. 변수의 속성을 명시하는 "type"의 경우 "ownedAttribute"를 하위요소로 가지는데, 여기에는 따로 속성을 구분하지 않고 편의를 위해 변수 이름과 동일하게 정의하였다.

3.3.3. Operation

```
<ownedOperation xmi:id="UMLOperation.10" name="void RunModify" visi
```

```
bility="public">
    <ownedParameter xmi:id="UMLParameter.11" direction="return"/>
    <ownedParameter xmi:id="UMLParameter.12" name="int a">
    </ownedParameter>
</ownedOperation>
```

"ownedOperation" 요소는 클래스의 메소드를 말한다. "ownedOperation"요소의 경우 "ownedParameter"가 하위 요소다.

"ownedOperation"의 주요 속성는 메소드의 아이디 "xmi:id"와 메소드 이름 "name", 가시성 "visibility"이다. "ownedOperation"요소는 예시에서

public void RunModify(int a) { }

와 같이 굵게 표시되어 있는 부분을 나타낸다.

"ownedParameter"는 함수의 파라미터를 표현하는 요소이다. 추가적으로 함수의 반환타입도 이 요소를 이용해 표현한다. "ownedParameter"요소에서 반환타입의 경우에 "direction"속성은 "return"값을 가지고, "name"속성은 생략가능하며, 파라미터인 경우에는 반대로 "direction"속성이 생략가능하다. "ownedParameter"요소에서 "xmi:id"는 파라미터의 식별자를, "name"은 파라미터의 이름을 나타낸다.

3.3.4. Association

```
<packagedElement xmi:type="uml:Association" xmi:id="UML
Association20" visibility="public" memberEnd="UMLAssociat
ionEnd.21 UMLAssociationEnd.22">
    <ownedEnd xmi:id="UMLAssociationEnd.21" name="dst
" type="UMLClass2" association="UMLAssociation20">
        <upperValue xmi:type="uml:LiteralUnlimitedNatural" x
mi:id="UpaUMLAssociation20" value="1"/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="Dn
aUMLAssociation20" value="1"/>
    </ownedEnd>
</packagedElement>
```

관계는 클래스와 클래스 사이에 연관 관계를 표현한다. "uml:Association" 태그는 "ownedEnd"라는 요소를 하위로 가지고 있다. "ownedEnd"는 "upperValue"와 "lowerValue"를 통해서 양쪽의 관계를 표현한다. 주요 속성은 아이디를 나타내는 "xmi:id", 이름을 나타내는 "name", 관계를 나타내는 "type"이 있다.

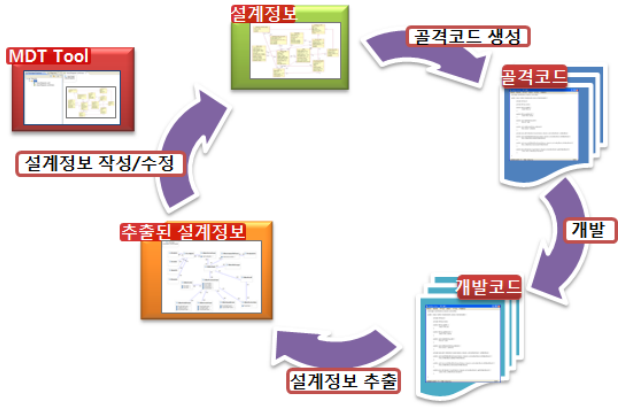
4. 결론

본 연구에서는 MDA 기반 컴포넌트 설계정보 관리 도구를 개발하였다. 이 도구는 이클립스 플러그인 형태로 개발하였고, MDA를 구현화한 MDT(Model Driven Tools)에 적용하였다. MDA방법론이 다양한 언어 및 플랫폼을 아우르는 방법론이라는 점에서 다양한 도메인의 개발 도구로써 발전하고 있는 이클립스 플러그인 형태의 개발은 큰 장점이 된다.

프로그램 개발에 있어서 모델링의 중요성이 크게 대두되고 있는 상황에 이러한 모델 정보를 표준화된 XMI를 통해 저장하게되면 모델 단위의 재사용성을 높일 수 있다. 그리고 XMI를 이용하여 골격 코드를 생성함으로써 모델에서 중요하게 설계된 변수나 함수들이 바로 코드로 적용되어 프로그램의 설계에서부터 구현까지 일관성을 제공한다.

이 도구는 모델정보로부터 순방향으로 골격코드를 생성하고 역방향으로 이미 생성되어 있는 코드를 분석하여 컴포넌트 모델정보를 추출할수 있다. 이종의 플랫폼으로 프로그램을 개발해야하는 경우 재설계 할 필요 없이 모델을 재활용하여 개발하기 때문에 개발 시간을 단축할 수 있다. 그리고 개발된 프로그램의 모델을 분석함으로써 문제점을 발견할 수 있고 보다 나은 프로그램을 위한 재료가 될 수 있다.

향후 이 도구는 OCL이나 액티비티 다이어그램을 분석하고 골격 코드를 보다 상세히 생성하여 프로그램 개발의 효율성이 높이는 연구가 필요할 것이다.



[그림 3] 전체흐름도

참고문헌

[1] JavaCC, "http://java.net/projects/javacc/"

[2] OMG, "MDA Guide Version 1.0.1 omg/2003-03-01", 12th June 2003.

[3] OMG, "MOF 2.0/XMI Mapping. Version 2.1.1", Decembet 2007.

[4] 김우식, 권오천, 신규상, "Model Driven Architecture 기술 소개", 2002년 12월, 전자통신동향분석

[5] 김준희, 윤현상, 이은석, "Model Driven Architecture 기반 소프트웨어 자동 생성을 위한 모델 변환기 자동 생성 도구", 2009년

[6] 김진만, 임좌상, "MDA를 적용한 클래스 다이어그램의 자바 소스코드 생성", 2008년

[7] 정지훈, "MDA는 소프트웨어 개발의 산업혁명을 가져올 것인가?", 2004년, 씨넷코리아