제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

템플릿을 이용한 PSM에 독립적인 코드 자동 생성 기법에 관한 연구

최연준
한국전자통신연구원 컴퓨터소프트웨어연구소
e-mail : june@etri.kr

An Study on Implementation of Automatic Code Generation Independent on PSM Using Template

YeonJun Choi*, MinJeong Kim, Munsu Lee, SeokJin Yoon, Ocheon Kwon
ETRI-Computer & Software Technology Laboratory

요 약

엔터프라이즈 컴퓨팅 환경에서 넘쳐나는 다양한 플랫폼과 언어, 프레임워크가 소프트웨어 개발에 대한 중복 투자를 야기하고 있다. 이종 플랫폼, 나아가 이종 개발 플랫폼을 자유로이 연동시킬 수 있는 개발 방법에 대한 필요성이 대두되면서 등장한 MDA 개념은 개발된 모델을 특정 플랫폼에 알맞은 형태로 변환함으로써 개발 모델 및 코드의 재사용성을 극대화한다. 본 논문에서는 MDA 개발 방법에 있어서 특정 플랫폼에 알맞은 코드를 동적으로 생성하면서 PSM 이나 프로그래밍 언어에 대한 중립성을 가지기 위한 방안으로 템플릿과 언어 처리기를 복합적으로 사용하는 방안을 제시한다.

1. 서론

웹이 엔터프라이즈 영역의 비즈니스에 적용되면서 컴포넌트 개발 방법론과 각종 런타임 플랫폼이 개발되었다. 개발 용이성과 재사용성을 고려한 컴포넌트 개발 방법론은 이상적인 이론에 머물러야 했던 객체지향 개발 방법론을 한차원 실용적으로 끌어올린 개발 방법론으로 현재 가장 현실적인 개발 방법론으로 손꼽히고 있다.

그러나 구현에 초점을 둔 컴포넌트 개발 방법론도 이종 플랫폼 간의 변환에는 적합하지 않아 같은 영역의 유사한 비즈니스라고 해도 이중 개발이 불가피하게 되면서 구현 모델을 플랫폼 독립적으로 만들고자 하는 노력이 시작되었고 그 결과가 국제 기구인 OMG (Object Management Group)에서 만든 MDA(Model Driven Architecture)이다.

MDA 는 크게 비즈니스 모델, 플랫폼 독립 모델(Platform Independent Model, 이하 PIM), 플랫폼 종속 모델(Platform Specific Model, 이하 PSM), 모델로부터 생성되는 플랫폼 종속적 구현 언어 코드로 이루어지는데, MDA 역시 비즈니스 모델로부터 궁극적으로 코드를 생성하는 것을 목표로 한다.

본 논문에서는 전체 MDA 개발 단계 중 PSM 단계에서 만들어진 모델로부터 코드를 자동으로 생성하기 위해 템플릿과 특정 언어에 적합한 유틸리티를 복합적으로 사용하는 복합적인 방식을 사용하여 PSM 과 프로그래밍 언어에 대한 간섭을 줄이는 시도를 하였고 이를 EJB 1.1 profile for UML 에 적용하여 보았다.

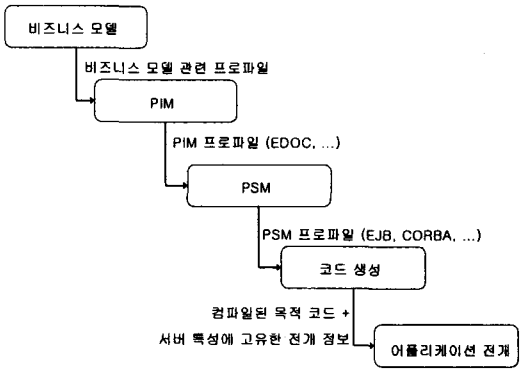
본 논문의 내용은 다음과 같다. 2 장에서는 연구 배경을 소개한다. 3 장에서 본 논문에서 제안하는 복합 방식을 설명하고, 4 장에서 구현 및 적용 예를 들고 마지막 5 장에서 결론을 맺는다.

2. 연구 배경

MDA 는 각 비즈니스 영역(domain)의 핵심에 해당하는 부분을 플랫폼 독립 적인 모델로 한 번 작성하여 두고 필요한 시점, 즉 원하는 플랫폼에 어플리케이션을 전개(deploy)시키는 경우에 플랫폼 의존적인 부분을 플랫폼 독립적인 모델로부터 자동 생성함으로써 생산성을 극대화시키고자 하는 접근 방법이다.

이러한 MDA 를 지원하는 도구는 [그림 1]과 같은

주요한 동작 흐름을 가진다.



[그림 1] MDA 의 동작 흐름도

이 중, PIM 에 대한 프로파일(profile)로 OMG 에서는 EDOC 을 표준으로 권유하고 있으며[1], EJB, CORBA 등 PSM 단계의 여러가지 플랫폼에 대한 프로파일의 표준화 작업은 해당 벤더 및 관련 단체에서 추진하고 있다[2, 3].

현재 이를 지원하는 도구로 OptimalJ 가 있다. OptimalJ 는 PIM 작성 후 모델 내부의 메소드에 대한 세부 구현을 프로그래밍 언어로 직접 작성하게 되어 있으며, 권고안으로 되어 있는 PSM 대신 내부 PIM 으로부터 프로그래밍 코드로 직접 변환하는 자체 모델을 사용하여 모델 간 자유로운 변환이 용이하지 않다[4].

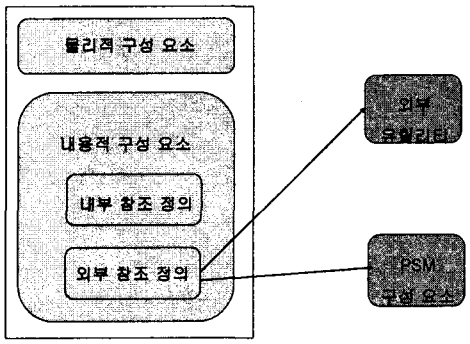
본 논문에서는 PSM 에 대한 UML 프로파일 정의를 알고 프로그래밍 언어나 프레임워크 특성을 알면 부가적인 프로그래밍을 하지 않고도 PSM 으로부터 프로그래밍 언어 소스 코드를 동적으로 생성할 수 있는 방안으로 템플릿과 유틸리티를 이용한 복합 방식을 소개한다.

3. 템플릿을 이용한 코드 생성

본 방안은 이미 PSM 이 모델링되어 있는 경우를 가정하고 있으며 XML 이나 XMI, 기타 방식을 이용한 PSM 단계의 모델링 결과물을 템플릿을 이용하여 소스 코드로 변환한다.

템플릿은 각종 PSM 의 코드 변환을 용이하게 하고, PSM 이 변경되더라도 개발된 프로그램의 변경 없이 템플릿 수정만으로도 코드 생성을 가능하도록 한다. 이 경우, 구조상 유사한 프레임워크를 가지는 경우 프로그래밍 언어 생성에 있어 부가적인 프로그램의 개발 필요 없이 텍스트 기반의 템플릿 작성만으로도 쉽게 모델로부터 소스 코드를 얻어낼 수 있다[5].

템플릿의 구성 요소는 [그림 2]와 같이 크게 프로파일에서 생성하는 소스 코드 파일에 관련된 물리적 구성 요소에 대한 정의와 소스 코드에 실제 반영될 내용 구성 요소에 대한 정의를 포함하며, 템플릿 외부에 특수 처리를 담당할 유틸리티와 PSM 의 모델 구성 요소가 존재한다.



[그림 2] 템플릿의 구성 요소

물리적 구성 요소는 파일 개수, 이름, 경로 등 물리적인 결과물이다.

소스 코드 내용적 구성 요소는 언어에 대한 예약어, 내부에서 참조되는 사용자 정의어, 참조되어질 외부 사용자 정의어를 포함한다.

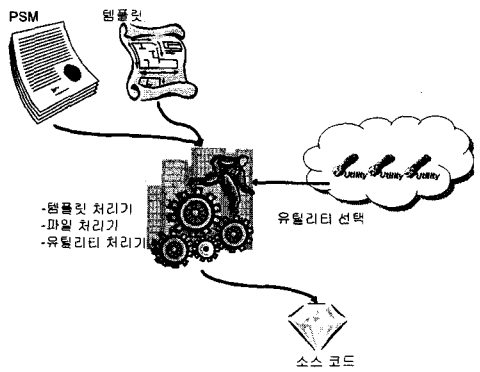
이를 지원하기 위한 템플릿 규칙은 [그림 3]과 같다.

```
[FileElement]   : 예약어로, 이 템플릿을 통해 생성
                  되는 파일 리스트를 알려줌
FileElement1.extension
FileElement2.extension
…

[FileElement1]  : FileElement 리스트에 있는 파일
                  내용으로, 차후 파일의 내용이 됨
…
… $CodeElement1$  : CodeElement 에 정의되어지
                    는 내용
… %ExternalElement1%  : 외부에서 참조되어지는
                        요소 정의
… $NL$      : new line 을 넣기 위한 예약어

[CodeElement1]  : CodeElement 요소 정의
…$CodeElement2$
…%ExternalElement2%

[CodeElement2]
#Utility1.0=value     : 외부 참조될 유틸리티 메소
                        드. 순서대로 파라미터
#Utility1.1=value
#Utility1.2=value
#Utility1.3=value
%ExternalElement3%    : 유틸리티에 의한 결과를
                        받기 위한 요소
```

[그림 3] 템플릿 규칙

[그림 4]은 템플릿 처리 과정을 보여 준다. 작성된 PSM 과 이에 알맞은 템플릿이 선택되면 템플릿 처리기에서 모델로부터 코드를 매핑하는 작업을 수행하고, 파일 처리기가 소스 코드를 해당 파일로의 수행을 담

당하게 된다. 템플릿 안에 포함된 외부 참조 요소는 유틸리티나 코드 구성요소가 되는데, 이는 각각 유틸리티 처리기와 템플릿 처리기에 의해 처리가 되어 소스 코드로 가공된다.



[그림 4] 템플릿 처리 과정

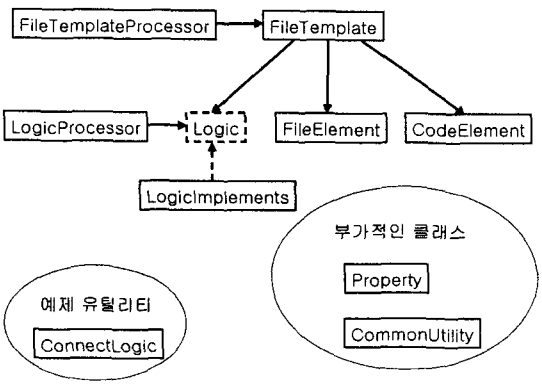
4. 구현 및 적용

본 연구를 위한 프로그램은 다음과 같은 환경에서 작성되었다.

- 작성 언어 : Java (JDK 1.3.1)
- 운영체제 : Windows XP

예제로 UML Profile for EJB 1.1[1]을 사용하여 EJB 및 자바 코드를 생성하여 보았다.

구현된 클래스의 구조는 [그림 5]와 같다.



[그림 5] 구현 클래스 구조

(1) 주요 클래스

1) FileTemplateProcessor

- 템플릿 파일을 이용하여 파일 템플릿을 가져 오고자 할 때 쓰임

2) FileTemplate

- 템플릿 파일을 분석하여 구성 요소(FileElement, CodeElement)를 만든다.

3) FileElement

- 템플릿에서 물리적 특성을 표현하는 데에 사용되며 템플릿의 FileElement 에 해당한다.

4) CodeElement

- 템플릿에서 소스 코드 구성 요소를 표현하는 부분이 되며 템플릿의 CodeElement 에 해당한다.

5) Logic

- 특정한 일을 수행하는 Logic 을 표현하는 클래스를 위한 인터페이스로, 이 인터페이스를 구현(implements)하면 템플릿 외부에서 역할을 수행하는 유틸리티 기능을 하게 된다.

6) LogicImplements

- Logic 을 구현하여 유틸리티 역할을 하게 된다.

7)) LogicProcessor

- Logic 인터페이스를 유틸리티 클래스를 구동하기 위한 처리기이다.

(2) 부가적인 클래스

1) Property

- 특정 데이터 종류에 따른 속성 리스트의 필드 이름과 그 값을 유지하기 위한 클래스로, 템플릿을 분석하거나 저장하기 위한 자료 구조이다.

2) CommonUtility

- 빈번하게 사용되는 메소드를 모은 클래스이다.

(3) 예제 유틸리티 클래스

1) ConnectLogic

- 특정 조건에 따라 연결되는 문자열을 만들기 위한 유틸리티로, 템플릿의 "Connect"를 처리하기 위한 유틸리티 클래스이다.

- throws, implements, 각종 오퍼레이션 및 그에 포함된 파라미터 리스트 등 특정 규칙에 따라 연결성을 가지는 문자열을 처리하는 기능을 가진다.

다음 [그림 6]은 예제로 사용한 EJB 1.1 에 대한 템플릿의 일부이다.

```
[FileElement]
EJBPrimaryKey.java
EJBRemoteInterface.java
EJBSessionHomeInterface.java
EJBEntityHomeInterface.java
EJBSessionBean.java
EJBEntityBean.java

[EJBPrimaryKey]
$JavaPackage$
import java.io.Serializable;

public class %Name% $JavaImplements$ {
$Operations$
$Attributes$
}

[EJBRemoteInterface]
$JavaPackage$

public class %Name% $Extends$ {
$EJBRemoteMethods$
}
...
```

[1] 현재 EJB 1.1 에 대한 프로파일이 존재하며, 2.0 에 대한 프로파일은 현재 나와 있지 않다

```
[Attributes]
#ConnectLogic.2=newline
#ConnectLogic.4=$Attribute$
%Attributes%
...
```

[그림 6] EJB 1.1 PSM 에 대한 템플릿

예제로 사용한 EJB PSM 은 다양한 테스트를 하기 위해 [표 1]과 같은 특징을 가지도록 구성하였다.

[표 1] 예제로 사용한 EJB PSM

	Remote Interface	Home Interface	Session Bean
name	Hello	HelloHome	HelloEJB
package	test	test	test
imports	javax.ejb.* javax.rmi.RemoteException	좌동	좌동
extends	EJBObject	EJBHome	
implements			SessionBean SBean2
attributes			context
operation (일부)	greeting greeting2	create	ejbCreate setSessoinContext greeting greeting2
operation 의 특성(일부)			
name	greeting	greeting2	
modifier	public	public	
parameters	String name	String name double time	
throws	RemoteException	좌동	
return	String	String	
attribute 의 특성(일부)			
name	context	verbose	
type	SessionContext	boolean	
modifier	private	private	
value	null	true	

시험 결과, 만들어진 EJB 소스 코드는 EJB 1.1 의 표준을 준수하도록 생성되었다[2]. 생성된 코드는 프로그래밍 언어 편집기에서 비즈니스 로직이 기술된 후 컴파일된다.

5. 결론

본 연구에서는 UML profile for EJB 1.1 및 UML profile for Java 일부를 지원하는 템플릿과 유틸리티를 구성하여 시험해 보았고, 이에 대한 만족할 만한 결과를 얻을 수 있었다. 이는 CORBA IDL 등 특정 프레임워크를 지원하는 PSM 도 변환 가능함을 암시한다.

그러나, PSM 의 완전한 자동 변환을 위해서는 프로그래밍 언어에 의존적인 현재 방식을 떠나, 언어 독립적인 실행 메커니즘을 프로그래밍 언어로 변환하는 방법이 필요하다.

현재 프로그램 소스의 전체 구조 뿐 아니라 오퍼레이션 단위의 행위를 기술할 수 있는 ASL(Action Semantic Language) 프로그래밍 언어 변환에 대한 변환을 고려하고 있다.

[2] 결과 소스 코드의 양이 많아 본문에 첨부되지 않음

참고문헌

[1] OMG, UML Profile for Enterprise Distributed Object Computing Specification Part II Supporting Annexes, OMG, 2001.s

[2] OMG, UML Profile for Enterprise Distributed Object Computing Specification, OMG, 2002.

[3] OMG, UML Profile for CORBA Specification, OMG, 2000

[4] Jonathan Stephenson, "Product Report - OptimalJ from Compuware", CBDi journal, 2002.

[5] 최연준, 권오천, 신규상, "PSM 으로부터 EJB 코드 생성을 자동화하는 방안에 관한 연구", 한국정보과학회 춘계학술대회, 2003.