2001년 한국정보처리학회 추계 학술발표논문집 제8권 제2호

EJB 컴포넌트의 코드 자동 생성 도구의 개발

차정은*, 양영종*, 신석규*
*한국전자통신연구원 소프트웨어공학부 컴포넌트 SW 연구팀
e-mail : {nary2743, yangyj, shinsk}@etri.re.kr

Development of Automatic Code Generator for EJB Component

Jung-Eun Cha*, Young-Jung Yang*, Suk-Ku Shin*
*Dept. of Software Engineering, ETRI

요약

생산성 있는 소프트웨어 개발 및 관리를 위한 기술적, 산업적 전략은 웹 환경 하에서 컴포넌트 기반의 소프트웨어 개발(CBD : Component Based Development)로 점차 귀결되고 있다. 나아가 정보 공유의 투명성이나 비즈니스 로직의 독립적 패키징과 같은, 이 두 개의 기술들이 개별적으로 가지는 특성과 장점들을 결합한 웹 서비스로 전환되고 있다. 따라서 컴포넌트 중심의 웹 서비스를 위한 인프라 환경의 지원이 무엇보다 중요하게 되었다. J2EE는 이러한 요구를 수용하기 위한 가장 표준화된 플랫폼 중의 하나로, 다중 계층의 엔터프라이즈 솔루션을 위한 간단한 개발과 전개, 관리 방식을 보장한다.

본 논문에서는 CBD의 완성을 위한 실제적인 접근으로서 J2EE 환경의 EJB 컴포넌트를 위한 코드 자동 생성 도구를 개발한다. 이를 위해 EJB 컴포넌트를 구성하는 코드의 속성들을 분석하여 보편적인 EJB 컴포넌트의 템플리트를 설계하였다. 또한 도구 구축의 과정에서 세부적인 EJB 생성기의 구조와 기능적 특성을 명시화하고 개발에 필요한 EJB 컴포넌트 정보를 정의, 분류하였다.

1. 서론

비용 효과적인 소프트웨어 산업화를 위한 다양한 시도들은 웹 환경 하에서의 컴포넌트 기반 소프트웨어 개발(CBD:Component Based Development)로 점차 요약되고 있다. CBD는 비즈니스 로직들을 잘 정의된 인터페이스를 통해 패키징함으로써 이미 완성된 소프트웨어 부품들을 준비해 놓고, 특정 목적을 위해 조립, 통합함으로써 품질 보증된 고생산성의 재사용이 가능하다[1]. 또한 웹 응용 시스템들은 여러 종류의 자원으로부터 어떤 형태의 정보든지 쉬운 획득과 전달을 통해, 엔터프라이즈 솔루션의 확장성과 트랜잭션의 일치성, 대화적 시스템 운영과 같은 요구를 충족시킬 수 있다. 최근에는 엔터프라이즈 솔루션 구축을 위한 컴포넌트와 XML 지향의 웹 서비스의 접목을 통한 새로운 소프트웨어 환경 구성을 위한 아키텍쳐들이 핵심 이슈로서 논의되고 있다[2].

본 논문에서는 이러한 요구를 배경으로, CBD 패러다임의 구현물로서 점차 그 효과를 실제화하고 있는 J2EE 아키텍쳐를 위한 EJB 컴포넌트의 코드 자동 생성기를 개발하고자 한다. EJB 컴포넌트는 J2EE 어플리케이션을 위한 서버 측의 핵심 비즈니스 로직들을 표준에 따라 생성, 전개, 조립하도록 패키징한 소프트웨어 모듈이다. 따라서 본 논문에서는 EJB 컴포넌트 코드의 기본 구조를 파악하고 시그네이쳐와 제약 사항들을 정리함으로써 코드 상의 속성들을 분석하여 컴포넌트 각 구성 파일들에 대한 템플리트 파일을 정의하였다. 또한 도구화를 위해 EJB 생성기의 구조와 기능적 특성을 명시화하고 개발에 필요한 EJB 컴포넌트 정보를 정의, 분류하였다.

2. EJB 컴포넌트의 코드 특성

2.1 EJB 컴포넌트의 구성 요소

EJB 컴포넌트는 클라이언트/서버 모델에서 서버 부분에서 운영되는 비즈니스 로직을 포함하고 있는 자바 컴포넌트를 의미한다. 따라서 기본적인 EJB 컨테이너 서비스를 이용하여 전개, 관리되며 보완이나 트랜잭션과 같은 서비스를 이용한다. EJB 컴포넌트는 <표 1>과 같은 세 가지 요소로 구성된다[3].

<표 1> EJB 컴포넌트의 구성

구분	내용
엔터프라이즈 빈	· 컴포넌트(비즈니스 로직)의 자세한 구현 부분
홈 인터페이스	· EJB 객체의 생성, 식별, 제거 메소드 정의
리모트 인터페이스	· 클라이언트가 호출하는 비즈니스메소드 정의
DD 파일	· 빈 클래스를 위한 서비스(life-cycle, Persistence, Transaction, Security) 설정 · EJB 컴포넌트의 구조 및 조립 정보 선언

또한 EJB 컴포넌트는 관리 데이터의 연속성에 따라 두 종류로 분류된다. 세션 빈은 시스템의 워크플로우 처리를 위해 사용되며 데이터의 연속성을 위해서는 엔티티 빈을 사용한다. 트랜잭션 처리를 많이 포함하는 데이터는 엔티티 빈을 활용하며 그렇지 않는 경우는 섹션 빈을 통해 데이터베이스에 직접 연결한다. 더욱 세부적으로, 세션 빈은 인스턴스 정보를 유지하느냐에 따라 Stateful 빈과 Stateless 빈으로 구분되며 엔티티 빈은 데이터베이스 엑세스 부분의 처리를 컨테이너 서버가 자동적으로 처리하느냐에 따라 Bean Managed Persistence와 Container Managed Persistence로 그 유형이 분류된다.

2.2 EJB 컴포넌트의 코드 특성

세션 빈은 여러 빈 간의 관계를 표현하고 특정 작업을 구현하는 프로세스 부분에 중점을 두는 반면 엔티티 빈은 데이터베이스의 접근과 데이터의 참조 및 관리를 코드 부분에서 중요하게 고려해야한다. 섹션 빈과 엔티티 빈 클래스들은 각각 <표 2>와 같은 코드 상의 요구 사항을 만족해야만 하다. 특히, javax.ejb.SessionBean 인터페이스 구현 및 이 인터페이스 내에서 나열된 콜백(Callback) 메소드 제공해야 하며 하나 이상의 ejbCreate 메소드 정의를 통해 클라이언트로부터 전달된 파라미터로 빈을 초기화한다.

<표 2> 빈 클래스의 코드 상의 특징

세션 빈	· SessionBean 인터페이스 구현 · 클래스는 "public"으로 선언 · Javax.ejb.sessionBean 상속받음 · 하나 이상의 ejbCreate() 구현(Empty 생성자 포함) · setSessionContext(), ejbRemove() 메소드 구현 · 비즈니스 메소드 구현
엔티티 빈	· java.ejeb.EntityBean 인터페이스 구현하며 public 정의 · 한개 이상의 ejbCreate() 구현 : Javax.ejb.CreateException포함 · ejbPostCreate()구현 : getPrimaryKey(), getEJBObject() 호출 · Finder 메소드 구현(home interface에서 선언) : ejbFindByPrimaryKey() 포함 · ejbRemove(),ejbLoad(),ejbStore() : javax.ejb.NoSuchEntityException 포함 · 비즈니스 메소드 구현 및 empty 생성자 포함

<표 3> EJB 컴포넌트 인터페이스 코드의 특성

Home Interface
· EJBHome 인터페이스의 확장 · 클라이언트가 호출하는 create() 정의 · create()의 파라미터 타입과 수는 ejbCreate 메소드에 대응 · 엔터프라이즈 빈의 리모트 인터페이스 타입 반환 · Java.rmi.RemoteException과 javax.ejb.EJBExecption 포함
Remote Interface
· javax.ejb.Object의 확장 · 클라이언트가 호출할 수 있는 비즈니스 메소드 정의 · 메소드들은 빈 클래스에서 구현된 메소드와 매핑 · 빈 클래스에서 구현되어진 메소드의 시그네이쳐와 매핑 · Java.rmi.RemoteException 포함

만약 Stateless 빈일 경우는 파라미터 없는 ejbCreate()를 구현해야 하며, 선택적으로 인터페이스가 제공하는 행위적인 관점을 요구할 때는 sessionSynchronization 인터페이스를 확장시킨다. 섹션 빈과 엔티티 빈 모두에서의 각 인터페이스 코드 특성을 요약하면 <표 3>과 같다.

빈 클래스를 설명하고 컨테이너에게 컴포넌트 관리를 위한 정보를 설명하는 전개 서술(Deployment Description)은 필수 요소와 부가적인 요소로 분류된다. 필수 요소로는 환경저적 정보, 리소스 정보, 다른 빈의 참조 정보, JNDI 정보 들이 있으며 부가적인 요소에는 트란잭션 제어, 보완 정보 등이 있다. 표준 EJB 컴포넌트의 전개를 위한 DD는 XML 파일로 작성되며 기본적으로 <표 4>와 같은 정보를 구성한다

3. EJB 컴포넌트의 코드 템플리트

본 논문에서는 EJB 컴포넌트의 자동 생성을 위해 템플리트 기법을 사용한다. 즉, 특정 코드 골격을 작성한 후 여기에 전달된 생성 정보를 매핑시키고 사용자의 적절한 요구를 수용하는 방법을 이용한다. 그러므로 EJB 컴포넌트의 특성에 의해 빈 타입에 따라 크게 두 종류의 코드 템플리트를 정의하였으며 각각 3가지의 파일로 구성된다. 또한 DD를 위한 템플리트도 정의하였다[4].

<표 4> Deployment Description의 정보 구성

분류	서술	정보
Basic structure	개략적인 DD 구성 정보	EJB 이름, Home/Remote 인터페이스 이름, 빈/트랜잭션 타입
Environment Entry	정보의 name space 설명	구성 엔트리 이름과 타입, 값
Bean Reference	참조되는 빈의 JNDI 이름	참조 빈의 이름, 타입, 사용 권한. 인터페이스 이른
Resource Factory	DB연결, 트랜잭션 등의 자원 정보	참조 빈의 이름, 타입, 사용 권한
Fixed Elements	필수적으로 설명해야만 하는 정보	구성 클래스의 이름, 빈 타입, 트랜잭션 및 연속성 관리 타입

<표 5>는 엔티티 빈 클래스의 코드 템플리트를 정의한 것이고 <표 6>은 EJB 컴포넌트의 홈 인터페이스에 관한 코드 템플리트이다.

<표 5> Entity Bean 클래스의 코드 템플리트

```
import javax.sql.*;
import javax.ejb.*;
import javax.naming.*;
import javax.rmi.RemoteException;
import extendedSystemLibray;
import userLibray;
public class BeanClassName implements EntityBean {
   MemberAttributes; /*Instance Variables : visibility type name*/
    . . . . . .
   private String dbName = LogicalDBname
   /* Business Method */
   public BMtype BMname (BMpara_type BMpara_name, , , , )
     {  BM_Body;
        BM_Exception:     }
   public primaryKey_type ejbCreate
       (ejbCreate_para_type ejbCreate_para_name, , , , ) {
            ejbCreateMethod_Body;
       try {          (ejbCreate_para_name, , , , );
            } catch (Exception ex) {throw new EJBException();  }
           /*Instance Variables Initialization*/
           this.MemberAttributes = MemberAttributes;
           . . . . . .
           return primaryKey;   }
   public primaryKey_type ejbFindByPrimaryKey
      (primaryKey_type primaryKey) throws FinderException { }
/* Finder method : Database 조작 및 select 문을 위한 함수 표현*/
  public ejbFMtype ejbFMname(ejbFMpara_type ejbFMpara_name, , )
 {    ejbFM_Body;
       ejbFM_Exception;          }
    . . . . . .
 public void ejbRemove() { }
 public void setEntityContext(EntityContext context) { . . . . }
 public void ejbActivate()  {
   primaryKey = (String)context.getPrimaryKey();    }
 public void ejbPassivate() {  primaryKey = null;     }
 public void ejbLoad() {
          try {  loadRow();
          } catch (Exception ex) { throw new EJBException();  } }
 public void ejbStore() {
          try {  storeRow();
           } catch (Exception ex) { throw new EJBException();}
public void ejbPostCreate
          (ejbCreate_para_type ejbCreate_para_name, , , ) {  }
```

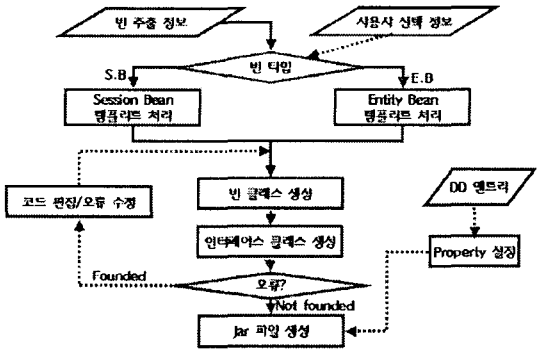
<표 6> Home Interface의 코드 템플리트

```
import java.rmi.RemoteException;
import javax.ejb.*;
import extendedSystemLibrary;
import userLibray;
public interface HomeInterfaceName extends EJBHome {
  public BeanClassName create
   (ejbCreate_para_type ejbCreate_para_name, , , , )
   ejbCreateMethod_Body,
   throws RemoteException, CreateException,.;
   public ejbFMtype ejbFMname(ejbFMpara_type ejbFMpara_name)
    ejbFM_Exception, RemoteException, createException;
 )
```

4. EJB 컴포넌트 코드 생성기

(1) 개요

본 논문에서 개발하는 EJB 컴포넌트 코드 생성기는 EJB 컴포넌트 구성 코드 각 부분에 대한 템플리트를 제시하고 적절한 생성 정보를 매핑시킴으로써 개별적인 빈 클래스를 생성한다. (그림 1)은 EJB 컴포넌트 생성기의 전체적인 이용 시나리오이다.



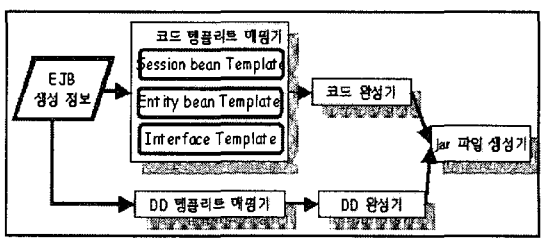
(그림 1) EJB 컴포넌트 생성기의 전체적인 프로세스

빈 생성 정보와 사용자 선택 정보들이 빈 타입에 따라 제공되는 템플리트에 매핑됨으로써 빈 클래스로 완성되고 컴포넌트 전개 정보를 DD 템플리트를 통해 확정지음으로써 기본 구조의 EJB 컴포넌트 코드를 생성한다. 또한 EJB 코드의 문법 오류 검사를 지원하며 최종적으로 컴포넌트의 표준 전개 형태인 Jar 파일로 패키징한다. (그림 2)는 EJB 컴포넌트 생성기의 세부 구성 모듈로 템플리트 매핑기를 통해 생성 정보들이 배치되고, 코드 완성기에 의해 일반 코드 옵션으로 완성되어 개별 파일로 컴파일 된 후 jar 파일로 패키징된다. 개별 모듈들의 개략적인 설명은 다음과 같다.

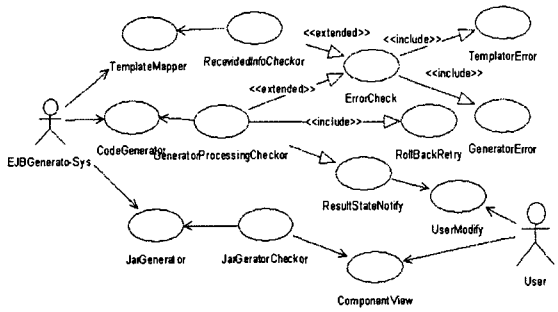
①코드 템플리트 매핑기 : EJB 컴포넌트에 맞는 템플리트 유형을 제공하고, 생성 정보를 유형에 맞도록 배치, 매핑
②코드 완성기 : 템플리트 매핑 결과에 라이브러리를 연결, 코드의 구현 부분을 삽입하고 필요한 곳을 링크
③DD 템플리트 매핑기 : DD 템플리트 제공하고 DD 메타 데이터를 배치, 매핑
④DD 완성기 : DD 메타 데이터를 DD 파일로 완성
⑤Jar 파일 생성기 : 빈 클래스, 인터페이스 클래스, DD 파일을 Jar 파일로 바인딩

(2) 생성기의 분석 및 설계

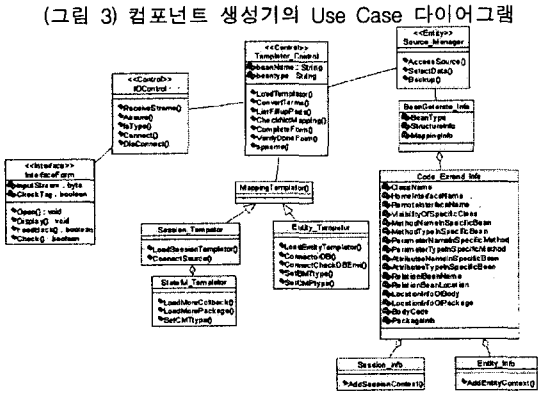
생성기 시스템의 문서화, 시각화하기 위해 UML (Unified Modeling Language)를 사용하였다. (그림 3)는 시스템의 Use Case 다이어그램이며 (그림 4)는 MVC 방식에 의한 코드 템플리트의 클래스 다이어그램이다.



(그림 2) EJB 컴포넌트 생성기의 구성 모듈



(그림 3) 컴포넌트 생성기의 Use Case 다이어그램

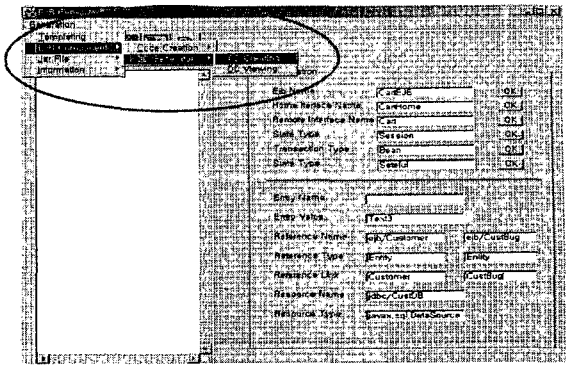


(그림 4) 컴포넌트 생성기의 클래스 다이어그램

EJB 컴포넌트 생성기 위해 요구되는 정보 즉, 코드 템플리트에 매핑되어야 하는 정보는 <표 7>과 같이 요약할 수 있다. 이들 정보는 개별적인 API로서 구현되어 코드 상에 포함된다.

<표 7> EJB 컴포넌트 생성기 개발을 위한 정보 항목

분류	정보
라이브러리	· userLibray 의 이름 및 위치 정보 · extendedSystemLibrary 의 이름 및 위치 정보 · Code-body 부분의 위치 정보
Deployment Description	· 빈 클래스 이름 · 홈 인터페이스, 리모트 인터페이스의 이름 · 빈 타입, 트랜잭션 타입 · 참조관계의 엔티티(참조 빈 구성 요소 이름, 타입, 위치 정보, 참조 DB 이름, 위치 정보) · 임계 상수의 이름 및 값, 타입
EJB 정보	· 빈 타입, 빈의 매핑 정보, 빈의 관계 정보
클래스	· 이름, 위치 정보 및 가시성 정보 · 인스턴스 변수의 타입, 이름, 초기값 · 메소드의 타입, 이름, 파라미터 정보 및 코드 위치 정보 · 메소드의 구현 코드 및 예외 처리 코드 · primaryKey의 이름, 타입 · 빈 초기화 메소드의 타입, 이름, 파라미터 정보 및 코드 위치 정보 · 빈 초기화 메소드의 구현 및 예외 처리 코드 · Finder 메소드의 타입, 이름, 파라미터 정보 및 코드 위치 정보 · Finder 메소드의 구현 코드, 구현 함수의 코드 및 예외 처리 코드 · 관계 있는 빈의 이름 및 위치 정보 · DB 접근 에트리뷰트 : 논리적/물리적 DB 이름



(그림 5) EJB 컴포넌트 생성기의 DD 생성 UI

(그림 5)는 본 논문에서 개발하는 EJB 생성기 시스템의 인터페이스 한 예로 <표 7>에서 정의한 DD 정보를 위한 필드들이 표현되어 있는 DD 생성 화면이다. 시스템 전체 메뉴는 크게 템플리트, 코드 생성, Jar 파일 생성 작업으로 분류하였으며 각기 세부 프로세스 모듈을 위한 항목들을 포함하고 있다.

5. 결론 및 향후 연구

본 논문에서는 CBD와 웹 기반 시스템의 결합되는 새로운 소프트웨어 패러다임의 실제적인 접근으로서 J2EE 환경의 EJB 컴포넌트 코드 생성을 자동화하는 도구를 개발하고자 하였다. 이를 위해 도구 구축의 과정에서 세부적인 EJB 생성기의 구조와 기능적 특성을 명시화하고 개발에 필요한 EJB 컴포넌트 정보를 정의, 분류하였다. 또한 실제 CBD 방법론에 준하여 시스템을 구축하였다. 본 논문에서는 템플리트 방식에 의한 자동 코드 생성 기법을 이용하였으며 현재 프로토타이핑 작업 중이다.

향후, EJB 응용 시스템 개발에 실제적인 적용을 위해 본 시스템의 구현 및 테스트가 수반될 것이며, 보다 더 큰 자동화를 위해 프레임워크 환경으로의 확장이 요구된다.

[참고 문헌]

[1]. Peter Herzum, *Business Component Factory*, John Wiley&Sons, 2000

[2] Chad Vawter and Ed Roman, "J2EE vs. Microsoft .NET : A Comparison of building XML-based web services", Sun Microsystems, http://www.theserverside.com/resources/articles/J2EE-vs-DOTNET/article.html

[3] Richard Monson-Haefel, *Enterprise JavaBeans*, O'REiLLY, 2000

[4] The Java™ 2 Enterpeise Endition Developer's Guide, Sun Microsystems, 2000