DBC 접근법을 이용한 엔터프라이즈 빈즈 명세 기법

노혜민[0] 유철중 장옥배
전북대학교 컴퓨터과학과
hmino@cs.chonbuk.ac.kr {cjyoo, okjang}@moak.chonbuk.ac.kr

Enterprise Beans Specification Technique Using Design By Contracts Approach

Hye-Min Noh[0] Cheol-Jung Yoo Ok-Bae Chang
Dept. of Computer Science, Chonbuk National Univercity

요 약

분산 웹 애플리케이션에 대한 관심도의 증가에 따라 서버측 컴포넌트 모델인 엔터프라이즈 빈즈가 많이 사용되고는 있지만 소프트웨어의 정확성과 견고성의 향상을 위한 노력은 미비한 실정이다. 본 논문에서는 소프트웨어의 정확성과 견고성을 높여줄 수 있는 Design By Contracts 접근법을 이용하여 엔터프라이즈 빈즈를 명세하는 방법을 제안하고자 한다. 이와 같이 엔터프라이즈 빈즈 개발에 Design By Contracts 접근법을 적용한다면 소프트웨어의 정확성과 견고성면에서 보다 진보적인 효과를 기대할 수 있다.

1. 서 론

분산 웹 애플리케이션에 대한 관심도가 증가함에 따라서 복잡한 분산 프레임워크와 관련된 코드의 작성없이 비즈니스 코드 작성에 주력할 수 있도록 해 주는 서버측 Java 컴포넌트 아키텍처인 EJB[1]에 대한 관심도 또한 증가하고 있는 추세이다. 그러나 이러한 관심도의 증가에도 불구하고 지금까지는 이러한 시스템의 품질 평가를 위한 노력은 미비한 것이 현실이다.

소프트웨어에서의 주요 품질 요인은 신뢰성이다. 이러한 신뢰성은 시스템이 명세된 대로 동작하는지를 나타내는 정확성(correctness)과 비정상적인 상황을 다룰 수 있는 능력을 나타내는 견고성(robustness)으로 나타낼 수 있다. Design By Contracts(이하 DBC) 접근법은 이러한 신뢰성을 나타내는 정확성과 견고성을 높여줄 수 있는 기법이다[2].

컨트랙트는 프로그램과 관련된 정형 명세의 한 요소이며[3] 이러한 컨트랙트에 의한 설계 방법 즉, DBC 접근법은 개발 프로세스의 초기부터 신뢰성에 초점을 맞춘 소프트웨어 공학 접근법으로 소프트웨어 시스템을 정확하게 정의된 컨트랙트를 기반으로 상호작용하는 구성요소들의 집합으로 보는 접근법이다[4]. 즉, 공급자와 사용자의 권리 및 책임을 명확하게 명세하여 이에 따라 시스템이 동작하도록 함으로써 소프트웨어의 신뢰성을 높이기 위한 방법론이다. 현재는 서비스 설계자가 가능한 모든 상황에서 서비스의 정확성 보증에 전적으로 책임을 지는 시스템 형식(tolerant style)보다는 서비스의 사용자가 특정한 전제 조건을 따르게 하는 형식(demanding style)이 선호되고 있다[4]. 즉, 어떠한 설계자도 시스템 사용의 미래는 예측할 수 없기 때문에 시스템의 정확한 작동을 보증해 주는 환경의 명세가 더욱 효과적이라는 것이다. 이러한 관점에서 사전조건, 사후조건, 불변조건으로 구성되는 DBC 접근법은 신뢰성 있는 객체지향 시스템 개발에 많은 기여를 해왔다[2].

본 논문의 목적은 객체지향 시스템에서 정확성과 견고성을 높여줄 수 있는 접근법으로 증명되고 있는 DBC 접근법을 엔터프라이즈 빈즈 개발에 적용함으로써 정확성과 견고성이 높은 엔터프라이즈 빈즈를 개발하기 위함이다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 DBC 접근법을 지원하는 OCL(Object Constraint Language) 언어를 이용하여 엔터프라이즈 명세 기법을 제안하고, 3장에서는 은행의 입출금을 담당하는 Account 엔티티 빈을 예제로 하여 OCL로 실제 명세해 보고 이러한 명세가 갖는 설계로서의 의미와 문서로서의 의미를 파악해 본다. 마지막 4장에서는 결론과 향후 연구를 제시한다.

본 연구는 한국과학재단 목적기초연구(2001-1-30300-017-1)지원 과제의 일부임

2. OCL을 이용한 엔터프라이즈 빈즈 명세 기법

본 논문에서는 명세 언어로 OCL을 선택하였다. 명세 언어에는 OCL이외에도 Object-Z, VDM++ 등과 같은 여러 명세 언어들이 존재하며 사용되고 있다. 본 논문에서 명세 언어로 OCL을 선택한 이유는 다른 언어들과는 달리 많은 수학적 배경지식을 필요로 하지 않기 때문에 관심 있는 많은 사람들이 쉽게 판독할 수 있다는 점에서이다[5].

OCL 언어는 객체지향 방법론에 적용되어온 언어이므로 엔터프라이즈 빈즈 명세에 OCL 언어를 아무런 가정이나 정의 없이 그냥 적용하기에는 무리가 있다. 따라서 본 절에서는 OCL 언어를 엔터프라이즈 빈즈 명세에 적용하기 위해 필요한 가정과 정의를 내려보고 이러한 가정과 정의를 기반으로 명세 규칙들을 제안한다.

본 논문에서 다루는 명세 기법은 여러 빈들로 구성되는 분산 애플리케이션 전체에 대한 명세가 아닌 하나의 엔터프라이즈 빈에 대하여 명세해 보는 것이다. 다소 협소한 명세일 수 있지만 이러한 하나의 빈에 대한 명세는 빈 제공자에게는 설계 방법론이 될 수 있으며 이러한 빈들을 이용하여 애플리케이션을 조립하는 애플리케이션 조립자에게는 훌륭한 문서가 될 수 있다. 본 논문에서 수행하는 하나의 빈에 대한 명세 기법에는 이러한 의미를 부여할 수 있다

2.1 가 정

OCL 언어를 엔터프라이즈 빈즈 명세에 이용하기 위해 다음과 같이 가정한다.

(가정 1) 엔터프라이즈 빈즈의 명세는 엔터프라이즈 빈즈의 클래스 다이어그램을 기반으로 작성한다.
(가정 2) 홈 인터페이스와 리모트 인터페이스에 대한 사항은 명세 대상에서 제외한다.
(가정 3) 엔티티 빈과 연관되는 데이터베이스의 테이블은 하나의 클래스로 간주한다.
(가정 4) 데이터베이스의 레코드들은 테이블(클래스)의 인스턴스들(객체)로 간주한다.
(가정 5) 세션 빈의 클래스 다이어그램은 빈 클래스에 대한 내용만으로 구성된다.
(가정 6) 엔티티 빈의 클래스 다이어그램은 빈 클래스와 프라이머리 키 클래스, 데이터베이스 레코드들과 그들간의 연관성, 다중성

(multiplicity)을 고려하여 구성된다.

이러한 가정들은 빈 클래스, 홈 인터페이스, 리모트 인터페이스, 프라이머리 키 클래스 등으로 구성되는 하나의 엔터프라이즈 빈즈를 연관관계를 갖는 객체들로 인식하기 위한 가정들로 이렇게 가정함으로써 객체지향 방법론에 적용되어온 OCL 언어를 엔터프라이즈 빈즈를 명세하는데 적용할 수 있도록 한다.

2.2 정 의

엔터프라이즈 빈즈에 (*가정 1*) ~ (*가정 6*)을 적용한다면 다음과 같은 정의들을 내릴 수 있다.

<정의 1> 엔티티 빈의 빈 클래스와 프라이머리 키 클래스와는 연관성을 가지며 다중성은 1 : 1이다.
<정의 2> 엔티티 빈의 빈 클래스와 데이터베이스 레코드들간에는 연관성을 가지며 다중성은 1 : 1..* 이다.
<정의 3> 불변조건은 빈 클래스가 항상 유지해야 하는 OCL 표현식으로 나타내어지는 조건들을 나타낸다.
<정의 4> 사전조건은 빈 클래스의 각 오퍼레이션이 수행되기 위해 갖추어야 할 조건들을 나타낸다.
<정의 5> 사후조건은 사전조건이 만족될 때 빈 클래스의 각 오퍼레이션이 제공해 주어야 하는 조건들을 나타낸다.
<정의 6> 빈 클래스의 오퍼레이션은 일반적인 비즈니스 오퍼레이션과 애플리케이션 서버의 컨테이너에 의해 호출되는 EJB Required 메소드로 구성된다.

<정의 1>과 ***<정의 2>***는 가정들을 적용했을 경우 엔티티 빈의 빈 클래스와 프라이머리 키 클래스 및 데이터베이스 레코드들간의 다중성을 정의하고 있다. 이러한 정의는 명세에서 OCL 표현식의 작성 시 사전 정의된 OCL 타입의 사용 및 OCL 문법의 적용 근거가 된다.
<정의 3>, ***<정의 4>***, ***<정의 5>***는 엔터프라이즈 빈즈에서 불변조건, 사전조건, 사후조건이 갖는 의미를 정의하고 있으며 ***<정의 6>***은 빈 클래스가 지니는 메소드의 종류를 정의하고 있다.

2.3 규 칙

앞서 기술한 가정과 정의를 적용한 엔터프라이즈 빈즈의 OCL 명세 규칙은 다음과 같다.

[규칙 1] 세션 빈의 클래스 다이어그램은 (*가정 2*), (*가정 5*)에 의해 다음과 같이 하나의 빈 클래스와 관한 정보만을 가진다.

그림 1 세션 빈의 클래스 다이어그램

[규칙 2] 엔티티 빈의 클래스 다이어그램은 (*가정 2*), (*가정 3*), ***<정의 1>***, ***<정의 2>***에 의해 다음과 같은 형태로 작성되어진다.

그림 2 엔티티 빈의 클래스 다이어그램

[규칙 3] 클래스의 불변조건은 다음과 같은 형태로 작성한다.

context *BeanClassName* **inv**:
OCL expression

[규칙 4] 클래스의 오퍼레이션에 대한 사전조건 사후조건은 다음과 같은 형태로 작성한다.

context *BeanClassName::operationName(param1 : Type1, ...): ReturnType*
pre *{precondition name} : OCL expression*
post *{postcondition name} : OCL expression*

[규칙 5] 불변조건, 사전조건, 사후조건을 구성하는 OCL 표현식(expression)은 ***[규칙 1]***, ***[규칙 2]***에 의해 작성된 다이어그램을 기반으로 OCL 스펙[6]의 사전 정의된 OCL 타입 및 문법에 따라 작성한다.

[규칙 1]과 ***[규칙 2]***는 엔터프라이즈 빈즈 명세의 기반이 되는 클래스 다이어그램을 작성하기 위한 규칙이고, ***[규칙 3]***은 불변조건의 작성규칙이며, ***[규칙 4]***는 사전조건 및 사후조건의 작성 규칙이다. 마지막 ***[규칙 5]***는 불변조건, 사전조건, 사후조건을 표현하는 OCL 표현식의 작성 규칙을 나타낸다.

3. 명세 기법의 적용

본 장에서는 2장에서 제안한 명세 기법을 기반으로 실제 엔터프라이즈 빈즈를 명세한다.

3.1 명세 도메인

본 논문에서 제안한 명세 기법을 적용할 예는 은행업무에서 입출금을 담당해 주는 엔터프라이즈 빈즈이며, 표 1은 예제와 관련된 정보를 나타내고 있다.

빈 이름	빈 타입	퍼시스턴트 타입
Account	엔티티 빈	빈 관리 퍼시스턴트

표 1 예제 엔터프라이즈 빈즈 관련 정보

예제를 빈 관리 퍼시스턴트 엔티티 빈으로 설정한 이유는 데이터베이스의 데이터와 연결 정보를 표현해야 하는 엔티티 빈이 세션 빈보다 명세해야 할 사항이 더 다양하고, 컨테이너가 대부분의 작업을 처리해 주는 컨테이너 관리 퍼시스턴트 보다는 작성 시 많은 부분을 빈 개발자가 담당해야 하는 빈 관리 퍼시스턴트가 명세할 내용이 더 방대하기 때문에 다양하고 많은 정보를 명세해 보기 위해 선택하였다.

3.2 Account 엔터프라이즈 빈즈의 명세

본 논문의 명세 기법은 클래스 다이어그램을 기반으로 한다 (*가정 1*). 따라서 먼저 Account 엔터프라이즈 빈즈의 클래스 다이어그램을 작성한다. ***[규칙 2]***를 적용하면 그림 3과 같은 클래스 다이어그램을 작성할 수 있다.

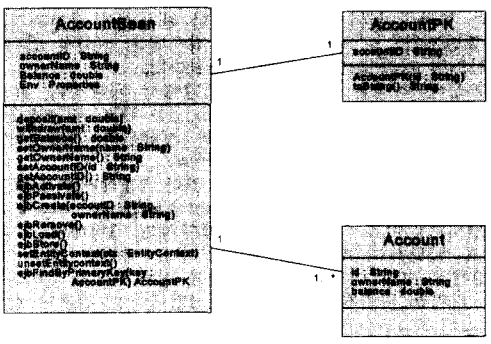

그림 3 Account 엔티티 빈의 클래스 다이어그램

이와 같이 작성된 클래스 다이어그램을 기반으로 Account 엔티티 빈을 명세해 보도록 한다.

먼저 빈 클래스 자체가 항상 유지해야할 제약사항인 불변조건은 *[규칙 3]*과 *[규칙 5]*에 의해 다음과 같이 나타낼 수 있다.

```
context AccountBean inv :
self.balance >= 0
```

다음으로 빈 클래스의 각 오퍼레이션에 대한 사전조건과 사후조건은 *[규칙 4]*와 *[규칙 5]*에 의해 기술한다. 앞의 *<정의 6>*에서 언급한 바와 같이 빈 클래스의 오퍼레이션은 비즈니스 메소드와 EJB Required 메소드로 구분된다.

먼저 빈 클래스가 갖는 비즈니스 메소드에 대한 명세는 데이터베이스 내의 데이터나 프라이머리 키 클래스를 고려하지 않아도 되기 때문에 클래스다이어그램의 연관관계나 다중성 등을 고려하지 않고 다음과 같이 명세할 수 있다.

```
context AccountBean::deposit(amt : double)
pre  : --none
post : self.balance = self.balance@pre + amt

context AccountBean::withdraw(amt : double)
pre  : amt <= self.balance
post : self.balance = self.balance@pre - amt

context AccountBean::getBalance() : double
pre  : --none
post : result = self.balance

context AccountBean::setOwnerName(name : String)
pre  : --none
post : self.ownerName = name
```

EJB Required 메소드는 비즈니스 메소드와는 다르게 데이터베이스내의 데이터나 프라이머리 키 클래스를 고려해야 하기 때문에 *[규칙 5]*를 신중하게 고려하여 명세해야 한다. 다음은 대표적인 EJB-Required 메소드인 ejbCreat()와 ejbRemove() 메소드에 대한 명세이다.

```
context AccountBean::ejbCreate(accountID : String, pOwnerName : String)
                                : AccountPK
pre  : self.account->select(accountID = id)->isEmpty
post : self.account->select(accountID = id and
                            ownerName = pOwnerName and
                            balance = 0)->size = 1
       result = self.accountPK.AccountPK(accountID)

context AccountBean::ejbRemove()
pre  : self.account->select(id1 | self.accountPK->select(id2 | id1.id =
                            id2.accountID)->size = 1
post : self.account->forAll(id1 | self.accountPK->forAll(id2 |
                            id1.id <> id2.accountID))
```

3.3 명세의 의의

본 논문에서 기술한 명세는 다음 두 가지 의미를 지닌다.

첫째는 소프트웨어의 설계 방법으로서 의미를 지닌다. 즉, 앞서 기술하였던 명세를 빈 제공자가 엔터프라이즈 빈즈를 개발하기 위한 설계 방법으로 채택할 수 있다는 의미이다. 이러한 소프트웨어 컨트랙트 개념은 상호 책임과 권리를 명확히 정의함으로써 모호함이 없이 소프트웨어를 설계할 수 있도록 해준다[7]. 따라서 앞서 제안한 명세 방법의 기반이 되는 DBC 접근법은 설계의 관점에서 보면 견고한 소프트웨어 개발의 기반이라고 할 수 있다.

둘째는 소프트웨어의 문서로서의 의미를 지닌다. 즉, 앞서 기술하였던 명세를 애플리케이션 조립자가 애플리케이션을 조립하기 위하여 각각의 엔터프라이즈 빈즈를 정확히 이해하기 위한 문서로서 참조할 수 있는 의미이다. 바꿔 말하면 빈을 조립하기 위한 정보로 명세를 활용할 수 있다는 것이다. 그러나 위에서 명세한 내용을 빈들을 조립하기 위한 정보로 활용하기에는 다소 부족한 점이 있다. 대표적인 것이 각 오퍼레이션의 트랜잭션과 관련된 사항이다. EJB는 개발자가 비즈니스 로직에만 전념하도록 하기 위해 트랜잭션 코드를 작성하지 않고 배치 디스크립터에 트랜잭션 속성(transaction attribute)과 트랜잭션 격리 수준(transaction isolation level)을 정의하도록 하고 있다. 따라서 빈 제공자가 이러한 트랜잭션 관련 사항을 명세에 정의해 놓는다면 애플리케이션 조립자가 명세를 더욱 효율적으로 활용할 수 있을 것이다.

Account 엔티티 빈에서 트랜잭션이 필요한 부분은 withdraw()와 deposit() 오퍼레이션이다. 따라서 다음과 같은 제약조건을 명시적으로 각각의 오퍼레이션의 사전조건에 추가한다면 더욱 애플리케이션 조립자가 빈을 이해하는데 도움이 될 수 있을 것이다.

```
transactionAttribute = #Required
transactionIsolationLevel = #TRANSACTION_READ_COMMITTED
```

4. 결론 및 향후 연구

몇몇 연구자들은 소프트웨어공학에서 유명한 실패 사례로 자주 인용되는 Ariane 5 로켓의 폭발 원인을 재사용 명세의 에러로 보고 있으며, 이러한 재사용 명세 에러를 해결하기 위한 방법으로 DBC 접근법을 제안하고 있다[8]. 이러한 관점에서 고려해 보면 본 논문에서 제안하고 있는 DBC 접근법을 이용한 엔터프라이즈 빈즈의 명세는 견고성 있고 정확히 동작하는 엔터프라이즈 빈즈를 구축하는데 많은 도움이 될 수 있을 것이다. 또한 본 논문에서는 언급하지 않았지만 최근 이슈가 되고 있는 컴포넌트의 테스팅 분야에서 테스팅 사례를 추출하는데 DBC를 이용한 명세는 충분한 활용 가치가 있다.

Beugnard[9] 등은 컨트랙트를 네 가지 레벨(basic contracts, behavioral contracts, synchronizational contracts, quality-of-services contracts)로 분류하여 정의하였다. 본 논문에서 명세한 명세 수준은 그들이 정의한 네 가지 레벨 중 하위 두 레벨만을 다룰 수 있는 명세이다. 나머지 두 가지 레벨을 다룰 수 있는 명세 방법과 하나의 빈이 아닌 여러 엔터프라이즈 빈즈간의 연관관계를 고려한 명세 방법에 관한 지속적인 연구가 필요하다.

참고문헌

[1] Ed Roman, *Mastering Enterprise JavaBeans*, Wiley, 2000.

[2] ISE, Building bug-free O-O software : An introduction to Design By Contract, http://www.eiffel.com/doc/manuals/technoloty/contract/page.html.

[3] B. Meyer, "Toward More Expressive contracts", JOOP, pp. 39-43, July-August, 2000.

[4] Yu Liu, "From UML to Design By Contracts", JOOP, April, 2001.

[5] Warmer, Kleppe, *The Object Constraint Language*, Addison Wesley, 1998.

[6] OMG, OMG Unified Modeling Language Specification, http://www.rational.co.kr/leadership/uml_resource_center.ap, 1999.

[7] Benoit Baudry, Yves Le Traon, Jean-Marc Jézéquel, "Robustness and Diagnosability of OO Systems Designed By Contracts", Software Metrics Symposium, pp. 272-284, 2001.

[8] Jean-Marc Jézéquel, and Bertrand Meyer, "Design By Contract: The Lessons of Ariane", *Computer*, pp. 129-130, Jan, 1997.

[9] Antoine Beugnard, Jean-Marc Jézéquel, Nöel Plouzeau, Damien Watkins, "Making Components Contract Aware", *Computer*, July, 1999.