EJB 에서 비즈니스 오퍼레이션의 설계 기법 및 패턴
(Techniques and Patterns for Designing Business Operations in EJB)

박 지 환 † 이 상 덕 †† 김 수 동 †††
(Ji Hwan Park) (Sang Duck Lee) (Soo Dong Kim)

요 약 객체 지향 모델링의 결과를 소스 코드로 매핑 할 때 구현 플랫폼에 맞는 정확하고 체계적인 매핑 기법이 요구된다. 또한, 모델링 자체는 구현 언어나 특정 플랫폼에 종속적이지 않기 때문에 특정 플랫폼이나 언어에 맞게 구현할 수 있는 효율적이며 순차적인 접근법이 필요하다. 모델링한 결과는 구현 상의 소스 코드로 정확하게 나타나야 하며 정확한 매핑을 위해서 본 논문에서는 EJB (Enterprise Java Beans) 2.0을 기준으로 하여 구현하고자 할 때, 모델링 단계에서 도출된 각 클래스가 가지는 비즈니스 오퍼레이션들이 EJB 2.0에서 지원하는 여러 가지 구현 가능한 장치들을 통해서 어떻게 나타날 수 있는지에 대한 절차적이며 구체적인 방법 및 디자인 패턴을 제시한다. 따라서 개발자로 하여금 이러한 기법들을 이용하여 비즈니스 오퍼레이션을 좀 더 체계적이며 정확하게 EJB 2.0 플랫폼에 맞도록 구현하여 설계 내용이 구현 소스 코드 상에 정확히 대응되어 나타날 수 있도록 지침을 제시한다.

키워드 : EJB, 비즈니스 오퍼레이션, EJB 인터페이스, EJB 패턴, 홈 비즈니스 메소드

Abstract Precise and systematic mapping techniques are required for mapping object-oriented artifacts into a platform-specific design. An effective and systematic mapping approach for an adequate platform or programming language is needed, because the characteristics of an initial design are independent from an implementation language and a platform.

In this paper, we propose systematic and concrete methods, guidelines, and design patterns that can be used to design business operations at EJB (Enterprise JavaBeans) source code level. We show how various EJB mechanisms can be utilized in designing business operations for beans. We believe these proposed methods can yield high-performance EJB applications that can also be well maintainable.

Key words : EJB, Business Operation, EJB Interface, EJB Pattern, Home Business Method

1. 서 론

UML[1, 2, 3]을 이용한 객체지향 기반의 모델링 결과를 EJB[4, 5, 6, 7]로 구현할 때 EJB 2.0 규격에 따라 그 내용을 반영시킬 경우에 엔티티 및 컨트롤러 클래스가 포함하는 비즈니스 오퍼레이션을 어떠한 인터페이스 및 메소드를 통해 명세를 하고 구현할 것인지에 대한 체계적이고 효율적인 지침이 요구된다. 또한, 모델링 단계에서 식별된 여러 종류의 클래스들은 EJB의 빈 클래스들로 매핑 될 때 EJB에서 지원하는 어떠한 인터페이스 및 메소드를 통하여 구현될 것인지에 대한 상세한 절차와 방법도 필요하다.

기존 EJB 1.1에서의 홈(Home) 인터페이스 및 리모트(Remote) 인터페이스를 통한 비즈니스 오퍼레이션에 대한 명세가 EJB 2.0 버전에서 새로이 추가 및 확장된 리모트(로컬) 홈 인터페이스와 리모트(로컬) 인터페이스 등 EJB 2.0의 여러 가지 가능한 오퍼레이션 구현 장치들로 나타날 수 있는데, 이들을 서로 매핑 시키는 것에 대한 지침이 필요하다. EJB 1.1에서는 홈 인터페이스와 리모트 인터페이스 만을 가지고 비즈니스 오퍼레이션에 대한 명세를 했었는데 EJB 2.0의 엔티티 빈에 새로이 추가된 홈 메소드라는 빈 인스턴스 전체에 공통적으로 적용할 수 있는 메소드와 기존의 엔티티 및 세션 빈 외에, 추가된 새로운 타입의 메시지 드리븐 빈(Message Driven Bean) 또한 비즈니스 오퍼레이션의 매핑에 관한 구체적인 지침이 요구된다. 따라서 본 논문에서는 체계적이며 효율적으로

† 비 회 원 : 숭실대학교 컴퓨터학과
perla7410@yahoo.co.kr
†† 비 회 원 : 한국정보통신기술협회
sdlee@tta.or.kr
††† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@comp.ssu.ac.kr
논문접수 : 2002년 4월 22일
심사완료 : 2002년 10월 31일

설계 모델에 대한 구현을 하기 위한 비즈니스 오퍼레이션의 설계 절차 및 구현 기법을 제시한다.

본 논문의 2장에서는 기존의 객체지향 방법론 및 EJB와 관련된 여러 책에서 제시된 비즈니스 매핑에 관련된 부분들의 문제점 및 한계성을 확인하고 이를 개선시킬 수 있는 점들을 명시한다. 3장에서는 새로이 추가 및 확장된 EJB 2.0의 기본적인 여러 오퍼레이션 구현 장치들에 대해서 알아보고, 4장에서는 본 논문에서 제시하는 비즈니스 오퍼레이션 및 클래스들에 대한 타입을 결정하는 방법과 각각의 비즈니스 오퍼레이션의 인터페이스 및 메소드로의 매핑에 관한 구체적인 절차 및 방법을 제시한다.

5장에서는 본 논문의 여러 가지 단계들을 통해서 매핑될 수 있는 EJB의 구현 가능한 인터페이스 및 메소드들에 대한 패턴을 분류 및 정의하고 구현 기법을 제시함으로써 모델링한 결과들이 체계적이고 효율적으로 EJB를 통해서 구현될 수 있도록 한다. 6장에서는 앞의 두 단계를 통해서 식별된 빈 타입 및 인터페이스 유형별로 어떠한 패턴을 적용시킬 것인지에 대한 전체적인 적용절차를 제시한다.

2. 관련 연구

2.1 유사 연구의 분석

표 1에서는 객체 지향 모델링 결과를 어떻게 EJB 플랫폼에 맞도록 구현할 수 있을지에 관한 기존 연구 방법들에 대한 특징 및 한계점을 개략적으로 설명한다.

기존의 연구들은 객체지향 모델링 결과를 체계적이며 절차적으로 EJB 라는 특정 플랫폼에 맞도록 구현할 수 있는 패턴 없이 단지 개발자의 경험 등에 의한 비체계적인 개발 방법으로 구현되었다. 따라서, 분석 및 설계 내용 자체를 정확하게 구현하는 데 구체적이고 상세한 방법 및 지침 등이 필요하다.

객체 지향 방법론을 이용한 모델링 결과를 특정 구현 플랫폼인 EJB 2.0에 기반하여 구현하고자 할 경우 설계 내용이 EJB에서 지원하는 구현 장치들을 이용하여 충분히 반영될 수 있어야 한다. 표 1에서와 같이 기존의 기법들은 EJB 1.1에 기반한 매핑 기법만을 제시하였거나 또는 클래스 오퍼레이션 수준까지의 매핑에 대한 지침은 제시하지 못하고 있다. 클래스 다이어그램을 이용하여 빈 타입 결정에 대한 지침만을 제시하고 있으므로, 상세하게 오퍼레이션 수준까지 어떻게 매핑 해야 하는지에 대한 방법이 필요하다.

또한 EJB 2.0의 홈 메소드나 새로운 타입의 빈 종류인 메시지 드리븐 빈 등과 같은 새롭게 지원되는 구현 장치들에 대한 적용 방법도 제시되어야 한다. 따라서, 더욱 다양해진 구현 장치들이 어떻게 이용될 수 있으며, 설계 내용이 더욱 효율적으로 구현될 수 있도록 하여야 한다.

표 1 기존 연구들의 특징 및 한계점

	특징	한계점
객체지향 개발방법론	• Booch, Coad/Yourdon, Jacobson 등의 방법론 있음. • 요구사항의 분석부터 정적, 기능적, 동적인 관점의 시스템 모델링을 통하여 주어진 도메인에 대한 분석 및 설계 가능.	• 설계된 모델들을 어떠한 기준 및 구현 기술에 종속적으로 특히, EJB 플랫폼의 구체적인 장치들을 이용할 수 있는지에 대한 지침 요구됨.
Arrington 기법[8]	• UML 기반의 요구사항 분석부터 구현에 이르는 객체지향 개발 과정 전반에 대하여 설계 내용을 EJB의 소스 코드에 이르기까지 매핑 하는 전체 과정을 나타냄. • 일반적인 자바 클래스, EJB 관련 클래스 및 사용자 인터페이스에 이르기 까지 모든 종류의 클래스를 Entity, Boundary, Control, Lifecycle 클래스라는 타입으로써 정의 후 각각에 대한 매핑을 함.	• 매핑시 EJB 1.1 버전으로의 세부적인 지침을 제시하고는 있지만, EJB 2.0 규격에서 제시하는 여러 가지 장치들을 이용한 매핑에 대한 지침이 요구됨.
Alur 기법[9]	• J2EE 기술에서 이용될 수 있는 여러 가지 패턴 기술을 제시함. • EJB 2.0에서 지원하는 빈 타입 및 지원 장치들을 이용하여 구현할 수 있는 복수개의 구현 모델들을 제시함. • 클라이언트 계층, 표현 계층, 비즈니스 계층, 통합계층 및 자원 계층의 총 5 가지 계층 중 표현, 비즈니스 및 통합 계층의 세 가지 경우에 있어서 고려할 수 있는 설계 패턴 및 소스 코드 수준에서의 매핑기법 제시함.	• 각 클래스의 클래스 오퍼레이션들을 EJB 2.0 규격에 맞도록 오퍼레이션 수준에서 어떻게 매핑 시키는지에 대한 상세한 지침이 요구됨.
Adatia 기법[10]	• EJB 2.0에서 새롭게 소개된 여러 가지 장치들에 대해서 설명. • 로컬 및 리모트, 세션 및 엔티티, 그리고 메시지 드리븐 빈 등의 여러 가지 장치들이 검토됨. • UML 상의 주요 다이어그램 즉, 클래스 다이어그램, Use Case 다이어그램, 시퀀스 다이어그램을 통한 모델링 결과들이 어떻게 여러 빈 타입으로 매핑 되는지에 대해 설명함. • 리모트와 로컬, 응집(cohesive) 클래스, 트랜잭션 경계, 디플로이먼트, 그리고 어플리케이션 객체의 다섯 가지 매핑 기준 제시.	• 빈 타입에 대한 매핑만을 위주로 함. • 모델링한 다이어그램의 내용들이 어떻게 비즈니스 오퍼레이션으로 매핑 되는지에 대한 상세한 지침 요구됨. • EJB 2.0의 여러 가지 오퍼레이션 구현 장치들을 이용한 정확한 매핑 과정 및 지침 제시 필요.

2.2 논문의 범위와 구성

본 논문에서는 이러한 비즈니스 오퍼레이션을 EJB 2.0 버전의 여러 장치들을 이용해서 구현하기 위해 필요한 매핑 절차 및 방법과 매핑 타입에 따른 패턴을 제시한다. EJB 2.0의 여러 가지 인터페이스 및 메소드로의 세부적인 매핑에 관한 지침이 요구되기 때문에 새로운 빈 타입인 메시지 드리븐 빈을 포함하여 어떠한 종류의 빈을 사용할 것인지, 빈 종류가 결정된 후 세부적인 각각의 오퍼레이션들은 EJB의 어떠한 인터페이스 혹은 메소드를 이용하여 명세 또는 구현할 것인지에 대한 지침이 필요하다. 또한 매핑된 결과를 가지고 구현을 할 때 지켜야 할 구현 제약사항 및 세부적인 구현 지침들을 제시하며, 각각의 매핑 타입에 따른 패턴도 분류 및 정의한다.

3. EJB 2.0의 오퍼레이션 구현 장치

3.1 리모트 인터페이스와 로컬 인터페이스[11]

리모트 인터페이스는 클라이언트가 서버에 배치되어 있는 빈 인스턴스가 가지고 있는 비즈니스 메소드를 Remote Method Invocation(RMI) 기술을 이용하여 접근하는데 필요한 인터페이스로써, 서로 다른 JVM상의 빈을 원격으로 호출할 때 이용한다. EJB 2.0에서는 리모트 홈 인터페이스와 리모트 인터페이스 두 가지 종류의 인터페이스를 정의한다.

로컬 인터페이스는 동일한 JVM 내의 빈 혹은 빈들 간의 호출을 위한 인터페이스로써 원격에서의 호출에 따른 네트웍 상의 오버헤드를 줄일 수 있으며, 'Pass-By-Reference'에 의한 매개변수의 전달이 가능하다. 그러므로, 기존의 EJB 1.1에서 가졌던 'Pass-By-Value'가 가지는 비효율성이나 융통성 측면의 문제점을 EJB 2.0 에서는 로컬 홈 인터페이스와 로컬 인터페이스 두 가지 종류의 인터페이스를 정의하여 해결한다. 클라이언트로부터 혹은 다른 빈들 간의 동일 JVM 내에서의 호출시에 야기되는 오버헤드 문제점을 로컬 인터페이스로써 줄일 수 있다.

3.2 홈 인터페이스와 컴포넌트 인터페이스

홈 인터페이스에서는 빈 인스턴스의 생성을 위한 create() 메소드, 삭제를 위한 remove() 메소드 그리고, 해당 빈 객체들을 찾기 위한 finder 메소드 등의 메소드에 대한 명세를 한다. 추가적으로 EJB 2.0 에서 새롭게 지원되는 홈 메소드에 대한 명세도 홈 인터페이스를 통해서 이루어진다. 세부적으로는 각 빈의 타입에 따라서 메소드가 가지는 매개변수 등에 차이가 있을 수 있다. 예를 들어서 무상태 세션 빈(Stateless Session Bean)의 경우에는 홈 인터페이스에 명세하는 create()와 빈 클래스에서 구현되는 ejbCreate() 메소드를 매개변수가 없는 단 하나의 형태만을 가질 수 있는데 반하여, 상태유지 세션 빈(Stateful Session Bean)에서는 서로 다른 매개변수 형태를 취하는 복수개의 create() 메소드를 정의할 수 있고, 또한 빈 클래스에서도 그에 대응되는 ejbCreate() 메소드를 동일한 매개변수의 형태로써 각각 가질 수 있다. 홈 인터페이스에는 리모트 홈 인터페이스와 로컬 홈 인터페이스의 두 가지 종류의 인터페이스가 있다.

로컬 인터페이스와 리모트 인터페이스를 통칭하는 컴포넌트(Component) 인터페이스는 홈 인터페이스 외의 기타 메소드에 해당되는 즉, 비즈니스 로직을 가지는 메소드에 대해 명세를 한다. 기존의 EJB 1.1 에서는 빈 클래스가 가지는 각 필드에 대한 getter와 setter 그리고 비즈니스 메소드에 대한 명세를 했었는데, EJB 2.0 버전에서는 엔티티 빈에 새롭게 지원되는 Container Managed Persistence(CMP) 및 Container Managed Relationship(CMR) 필드 등을 이용하기 때문에 비즈니스 메소드에 대한 getter 및 setter는 빈 클래스에 추상 메소드로 정의를 하고 별도로 인터페이스에는 명세하지 않는다. 따라서, 순수 비즈니스 로직을 가진 메소드만이 컴포넌트 인터페이스에 명세된다.

3.3 컴포넌트 비즈니스 메소드와 홈 비즈니스 메소드

EJB 1.1에서는 리모트 인터페이스와 홈 인터페이스의 두 가지 종류의 인터페이스에 대해서만 메소드의 명세를 했었으나, EJB 2.0에서는 리모트 인터페이스와 로컬 인터페이스라는 두 가지 종류의 장치를 정의하면서, 컴포넌트 비즈니스 메소드를 컴포넌트 인터페이스에 명세한다.

홈 비즈니스 메소드는 특정 빈 인스턴스에 대한 오퍼레이션의 명세를 하는 컴포넌트 인터페이스와는 달리 엔티티 빈의 모든 빈 인스턴스에 공통적으로 적용할 수 있는 메소드를 명세한다. 일반적인 자바 프로그램에서 모든 클래스에 공통적으로 적용되는 클래스 메소드가 이러한 타입의 오퍼레이션이다. EJB 2.0 에서는 리모트 홈(Remote Home) 인터페이스와 로컬 홈(Local Home) 인터페이스를 통해서 이러한 메소드에 대한 명세를 한다.

3.4 동기적 오퍼레이션과 비동기적 오퍼레이션

비즈니스 오퍼레이션을 수행하는 클라이언트 즉, 액터 입장에서의 수행 결과에 따라 동기적 혹은 비동기적 오퍼레이션으로 분류를 할 수 있다. 첫째, 비즈니스 오퍼레이션을 수행할 때 그 오퍼레이션의 수행 결과로써 어떠한 리턴값을 받거나 그 리턴 값을 받기까지 클라이언트의 상태가 블럭되는 경우 동기적 오퍼레이션이라 한

다. 기존의 EJB 1.1에서 지원하던 인터페이스 및 빈의 메소드들이 수행하는 오페레이션들은 모두 서버 상의 빈 인스턴스가 해당 오퍼레이션을 수행 후 그에 대한 결과값을 클라이언트 쪽으로 보내는 시간 동안 클라이언트가 블럭 상태가 되기 때문에 동기적인 오퍼레이션이다[12, 13].

그러나 EJB 2.0에서 새로운 빈 타입인 메시지 드리븐 빈(Message Driven Bean)은 기존의 J2EE 요소 기술 중의 하나인 JMS의 메시지 소비자(Consumer) 역할을 하는 또 하나의 클라이언트를 빈으로써 구현한다. 따라서 메시지 드리븐 빈에 대한 클라이언트는 JMS 메시지를 보냄으로써 비동기적인 오퍼레이션 형태의 구현을 한다. 또한 JMS의 비동기적인 메시지 호출에 대해서는 결과값이 없기 때문에 그에 따른 세션 빈에서의 메소드 호출과는 구별되는 다른 형태의 비즈니스 구현을 한다[14, 15, 16, 17].

4. 비즈니스 오퍼레이션의 설계 절차 및 기법

4.1 빈 타입 식별

객체지향 개발 방법론을 통해서 도메인에 대한 모델링한 결과를 EJB 플랫폼에 맞게 구현할 경우, 우선적으로 고려할 사항은 설계시 고려된 각 클래스들이 어떤 종류의 빈으로 매핑 되는가 이다[18, 19]. 따라서, 본 논문에서는 이러한 빈 타입을 선택할 때 고려 해야할 사항에 대해서 어떠한 순서와 항목들을 대상으로 하는지를 우선적으로 살펴본다.

대상이 되는 클래스에 대해서 우선적으로 그 클래스가 데이타 위주의 클래스인지 아닌지에 따라서 엔티티 빈 및 세션 빈으로 구분을 한다[20]. EJB 2.0에서의 새로운 타입의 빈인 메시지 드리븐 빈은 이 단계에서는 고려하지 않으며, 그러한 빈 타입은 두 번째 식별 단계인 비즈니스 오퍼레이션 타입 분석 단계에서 식별된다.

그림 1에서는 대상이 되는 클래스를 어떠한 빈 종류로 매핑 해야 하는지 고려하는 단계이다. 그 첫 번째 단계에서는 해당 클래스가 영구성 데이타를 가지고 있는지 확인한다. 메소드들이 해당 클래스가 가지는 여러 개의 속성들에 대한 영구적인 읽고 쓰기 위주의 메소드들이라면 우선적으로 엔티티 빈으로 분류한다.

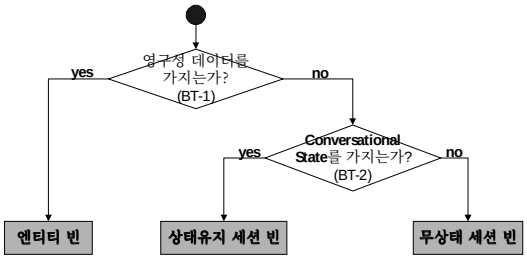

그림 1 빈 타입 식별

둘째, 영구성 데이타를 갖지 않는 클래스라면 세션 빈 또는 EJB 2.0에서 새롭게 추가된 또 하나의 빈 종류인 메시지 드리븐 빈으로 매핑할 수 있는데, 이 단계에서는 우선적으로 세션 빈으로만 분류한다.

세션 빈은 'Conversational State'라는 클라이언트의 어느 한 세션 동안에 클라이언트의 메소드 호출간에 유지 되는 빈 클래스가 가지는 변수 값이다. 이러한 상태를 기준으로 해서 상태유지(Sateful) 세션 빈과 무상태(Stateless) 세션 빈으로 나눌 수 있다.

해당 클래스가 가지는 'Conversational State'의 유무를 판별해서, 그러한 상태를 갖는 경우 상태유지 세션 빈으로 구분하며, 그렇지 않은 경우 무상태 세션 빈으로써 분류를 한다. 이번 단계에서 식별되는 빈의 종류는 총 세 가지 타입으로써 엔티티 빈, 상태유지 세션 빈, 무상태 세션 빈 등으로 분류할 수 있다.

첫 번째 종류인 엔티티 빈의 경우, 모델링 단계에서의 클래스가 가지는 메소드가 데이타 위주의 성격을 갖는다. 설계 단계에서 식별되는 여러 클래스들 중 액터가 직접 접근하여 사용하는 즉, 컨트롤러 클래스의 역할을 하는 클래스가 아닌 직접 데이타 베이스에 영구 데이타를 읽고 쓰는 역할을 하는 클래스들이 이러한 종류에 해당된다.

두 번째 종류인 상태유지 세션 빈의 경우, 설계 단계에서의 클래스가 가지는 메소드들이 비즈니스 오퍼레이션의 성격을 가지며 또한 'Conversational State'도 갖는 경우이다. 이는 클라이언트로부터의 한 번의 메소드 호출에 의해서 그 기능을 수행하는 메소드 보다는 일정 세션 동안에 특정 클라이언트에 대한 상태를 저장하며 유지하는 경우이다.

세 번째 종류인 무상태 세션 빈의 경우, 클래스가 가지는 메소드들이 클라이언트로부터의 한 번의 메소드 호출로 그 기능을 수행하며 특정 클라이언트에 대해서 일정한 상태를 유지하지 않는 경우이다.

4.2 비즈니스 오퍼레이션 타입 분석

빈 타입을 식별하는 첫 번째 단계를 통해서 정해진 빈 종류에 이어서, 이번 단계에서는 각각의 클래스가 가지는 메소드들이 EJB의 어떠한 인터페이스 및 메소드를 통해서 구현될 수 있는지에 대한 비즈니스 오퍼레이션의 타입을 분석한다.

앞의 단계에서는 엔티티 빈, 상태유지 세션 빈, 그리

고 무상태 세션 빈 등의 모두 세 가지 종류의 빈으로만 매핑을 했는데 EJB 2.0에서는 이 외에도 추가적으로 새로운 빈의 종류인 메시지 드리븐 빈을 정의하고 있다. 이 메시지 드리븐 빈은 기존의 J2EE 요소기술의 하나인 Java Messaging Service(JMS)의 메시지 소비자(Message Consumer)에 해당하는 클라이언트 역할을 또 하나의 새로운 빈 타입으로서 정의된다[21].

이렇게 빈 타입 식별에서는 엔티티, 상태유지 세션, 무상태 세션 빈 등의 세 가지 종류의 빈만을 식별하였으나, 이번 단계를 통해서는 추가적으로 메시지 드리븐 빈 타입으로 구현될 클래스 및 비즈니스 오퍼레이션을 결정하고 나머지 엔티티, 상태유지 세션 및 무상태 세션 빈의 경우에 대해서는 매핑 될 각각의 세부적인 인터페이스 및 홈 메소드까지도 결정한다.

전 단계와 본 단계에서 결정되는 세 가지 빈 타입 및 네 가지 인터페이스 그리고 두 가지 경우의 홈 메소드 타입으로써 EJB 2.0에서 지원하는 비즈니스 오퍼레이션 구현을 위한 모든 장치를 나타낼 수 있다. 또한, 각 인터페이스 및 메시지 드리븐 빈을 통한 비즈니스 오퍼레이션의 명세 또한 각각 유일하게 매핑 되어야 한다.

그림 2는 비즈니스 오퍼레이션에 대한 타입을 결정하는 단계로써 첫째, 메시지 드리븐 빈으로써 식별될 수 있는 기준으로 해당 오퍼레이션이 동기적(Synchronous) 인지 비동기적(Asynchronous) 인지에 따라서 구분 한다. 오퍼레이션이 동기적일 경우 즉, 해당 오퍼레이션을 수행한 클라이언트의 상태가 그 결과값을 받을 때까지 블럭 상태가 되는지에 따라서 블럭 상태가 될 경우는 동기적인 오퍼레이션으로 분류를 한다.

둘째, 비즈니스 오퍼레이션이 비동기적 이고 메소드의 수행에 따른 리턴 값이 있을 경우, 엔티티 빈 및 상태유지 세션 빈 또는 무상태 세션 빈으로 분류한다. 만일 결과로써 어떠한 리턴 값을 기대하지 않는 메소드의 경우는 비동기적인 오퍼레이션의 수행 후 리턴 값이 없는 메시지 드리븐 빈의 메소드를 통해서 구현할 수 있다. 이는 클라이언트로부터의 JMS 메시지를 받는 경우에 해당되며, 그 메시지의 종류를 판별하여 빈 인스턴스가 해당 메시지의 종류에 따라 메시지에 대한 일을 처리하게 된다.

빈 타입 식별 단계에서 별도로 식별하지 않았던 메시지 드리븐 빈 형태의 메소드에 대한 식별은 비즈니스 오퍼레이션 타입 분석의 두 번째 단계에서 하고, 세 번째로 해당 메소드가 특정한 빈 인스턴스에 대해서만 적용되는 것인가를 기준으로 한다. 즉 일반적인 자바 프로그램에서 클래스의 모든 인스턴스에 공통적으로 적용시킬 수 있는 메소드인 클래스 메소드와 같이 EJB의 모든 빈 인스턴스에 대해서도 동일하게 적용할 수 있는 종류의 메소드가 이에 해당된다. 이러한 종류의 메소드의 경우, 홈 인터페이스(Home Interface)에 명세를 한다. 홈 메소드의 경우 또한, 리모트 홈 인터페이스(Remote Home Interface) 인지 혹은 로컬 홈 인터페이스(Local Home Interface) 인지에 따라서 두 가지의 인터페이스에 각각 명세한다.

넷째, 세부적인 기준으로는 홈 인터페이스에 명세를

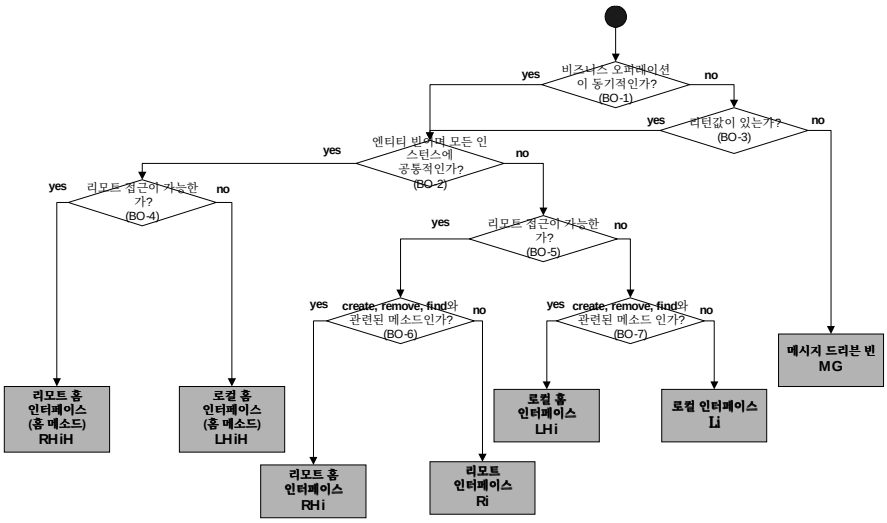

그림 2 비즈니스 오퍼레이션 타입 결정

하고 홈 인터페이스에 대한 레퍼런스를 통해서 그 메소드의 실행이 가능하다는 점은 동일하다. 그러나 접근할 수 있는 범위가 원격에서 하는지 혹은 로컬 즉, 동일한 자바 가상 머신(Java Virtual Machine) 내에서 이루어지는 지에 따라서 리모트 홈 인터페이스와 로컬 홈 인터페이스로 나눌 수 있다.

원격에서의 접근이 가능할 경우, 홈 메소드는 리모트 홈 인터페이스에 명세를 하며, 로컬에서만 접근할 경우는 로컬 홈 인터페이스에 명세를 하게 된다.

앞의 단계들에서는 메시지 드리븐 빈과 모든 빈 인스턴스에 대해서 공통적인 오퍼레이션을 수행하는 홈 메소드의 리모트 및 로컬 인터페이스로 명세하는 경우를 분석했다. 그러나 특정 빈 인스턴스에만 국한되는 메소드일 경우에 있어서 리모트 접근이 가능한 경우 크게 두 가지로 나눈다. 첫째, 메소드가 객체의 create, remove, find와 관련된 메소드 인지의 여부에 따라서 리모트 홈 인터페이스 및 리모트 인터페이스 등으로 나눌 수 있는데, 일반적인 비즈니스 로직을 가지고 있는 메소드의 경우 리모트 인터페이스를 통해서 명세를 한다.

비동기적인 오퍼레이션이면서, 모든 빈 인스턴스에 대하여 공통적인 오퍼레이션이 아니며, 리모트 접근이 가능하지 않을 경우 로컬 홈 인터페이스(Local Home Interface) 및 로컬 인터페이스(Local Interface) 두 가지의 인터페이스를 통하여 명세를 한다.

5. 비즈니스 오퍼레이션 설계 패턴 및 구현 기법

첫 번째 및 두 번째 단계를 통하여 식별되는 빈 타입과 그에 따른 인터페이스 및 메소드 들은 본 장에서 제시하는 비즈니스 오퍼레이션에 대한 8 가지의 설계 패턴을 통해 분류를 할 수가 있다.

5.1 Static Business(SB) 패턴

SB 패턴은 엔티티 빈의 모든 인스턴스에 공통적으로 수행할 수 있는 클래스 메소드 같은 경우에 해당되는 것으로써, 특정 빈 인스턴스가 아닌 모든 빈 인스턴스에 메소드를 수행하고자 할 때 이 패턴을 사용한다.

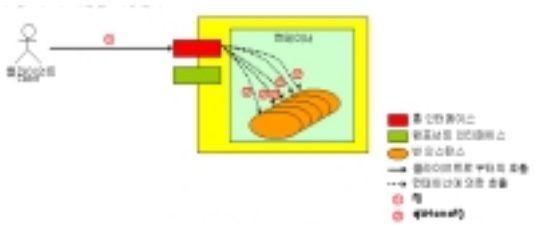

그림 3 Static Business(SB) 패턴

그림 3은 클라이언트가 홈 인터페이스에 명세 되어 있는 f()라는 이름의 메소드를 호출한 후 컨테이너에 의해서 다시 callback 메소드인 ejbHomeF() 라는 빈 클래스의 메소드를 실행 하므로써 이루어진다. f() 메소드는 그림 4의 SBBean의 모든 인스턴스에 대해 공통적으로 적용되는 메소드이다.

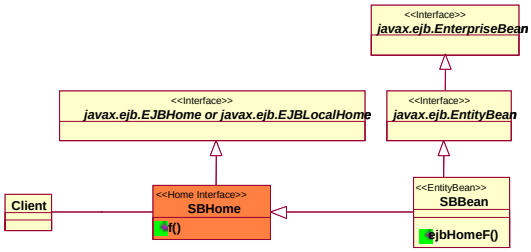

그림 4 엔티티 빈의 홈 메소드 관련 클래스 구조

f()라는 메소드는 그림 4와 같이 구현 시에 홈 인터페이스(SBHome)에 명세를 하며, 그것을 구현한 빈 클래스(SBBean)에서는 홈 메소드에 대한 구현(ejbHomeF())을 한다. 홈 메소드를 구현한 엔티티 빈은 javax.ejb.EntityBean이라는 인터페이스를 구현하며, 이 인터페이스는 javax.ejb.EnterpriseBean이라는 인터페이스를 상속한다.

SBHome이라는 SBBean의 홈 인터페이스가 구현하는 javax.ejb.EJBHome 또는 javax.ejb.EJBLocalHome 인터페이스는 리모트 또는 로컬 클라이언트가 빈 인스턴스의 create, find 및 remove 그리고 홈 메소드들에 대한 명세를 한다. javax.ejb.EntityBean 이라는 인터페이스는 모든 엔티티 빈 클래스가 구현해야 하는 인터페이스로써 빈 인스턴스에 대해 수행할 메소드들 즉, callback 메소드들에 대하여 명세한다.

```
<SBHome.java>
public interface SBHome extends EJBHome {
        . . .
        public <return_type> f() throws Remote
        Exception;
        . . .
}
<SBBean.java>
public class SBBean implements EntityBean {
        . . .
        public <return_type> ejbHomeF() {
        . . .
}
```

SBHome.java와 SBBean.java에서는 SB 패턴을 적용하여 구현할 경우 SBHome 이라는 홈 인터페이스의 홈 메소드와 SBBean이라는 빈 클래스 내의 ejbHomeF()라

는 메소드 구현 부분을 나타내고 있다. 홈 메소드는 엔티티 빈에만 해당되는 메소드로써 이를 구현한 빈 클래스에서는 EntityBean 이라는 인터페이스를 구현한다.

5.2 Persistence CRUD(PC) 패턴

PC 패턴은 엔티티 빈 인스턴스의 생성 및 소멸 그리고 생성된 빈 인스턴스에 대한 find하는 함수들에 대한 패턴을 나타낸다. 클라이언트로부터의 메소드 호출에 따라 컨테이너는 해당되는 각각의 callback 메소드를 수행하게 된다.

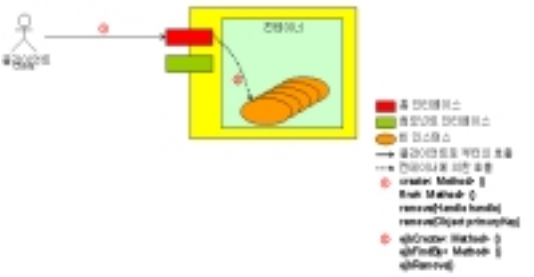

그림 5 Persistence CRUD(PC) 패턴

그림 5는 엔티티 빈의 빈 인스턴스 생성을 위한 메소드인 create<Method>(), 생성된 빈 인스턴스를 찾는 메소드인 find<Method>(), Handle을 이용하여 객체를 삭제하는 remove(Handle handle) 메소드, 그리고 엔티티 빈의 고유한 프라이머리 키로써 삭제를 하는 remove (Object primaryKey) 메소드 등과 같은 CRUD 성격의 메소드들에 대한 명세를 한다. 이러한 메소드들에 대한 호출로써 컨테이너에 의해서 각각 ejbCreate<Method>(), ejbFindBy<Mehtod>(), ejbRemove() 등의 callback 메소드가 수행된다.

이러한 CRUD 타입의 메소드들은 그림 6과 같이 홈 인터페이스인 PCHome 인터페이스에 각각의 메소드에

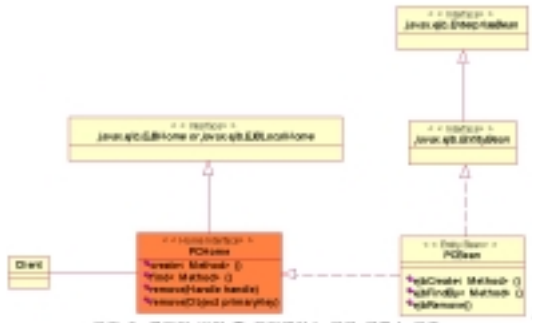

그림 6 엔티티 빈의 홈 인터페이스 관련 클래스 구조

대한 명세를 한 후, 빈 클래스인 PCBean에서 구현을 한다. 구현 클래스인 PCBean은 엔티티 빈으로써 javax.ejb.EnterpriseBean이라는 인터페이스를 상속한 javax.ejb.EntityBean이라는 인터페이스를 구현한다.

PCHome이라는 PCBean의 홈 인터페이스가 구현하는 javax.ejb.EJBHome 또는 javax.ejb.EJBLocalHome 인터페이스는 리모트 또는 로컬 클라이언트가 빈 인스턴스의 create, find 및 remove 그리고 홈 메소드들에 대하여 명세를 한다. javax.ejb.EntityBean 이라는 인터페이스는 모든 엔티티 빈 클래스가 구현해야 하는 인터페이스로써 빈 인스턴스에 대해 수행할 메소드들 즉, callback 메소드들에 대하여 명세를 한다.

```
<PCHome.java>
public interface PCHome extends EJBHome {
        . . .
        public PrimaryKeyClass create<Method>(. . .)
        throws RemoteException;
        public ComponentInterface
        findByPrimaryKey(Object primaryKey) throws
                RemoteException;
        public Collection findBy<Method>(. . .) throws
        RemoteException;
        public remove(Handle handle) throws
        RemoteException;
        public remove(Object primaryKey) throws
        RemoteException;
        . . .
}
<PCBean.java>
public class PCBean implements EntityBean {
        . . .
        public PrimaryKeyClass ejbCreate<Method>(. . .)
        {. . .}
}
```

PCHome.java와 PCBean.java에서는 PC 패턴을 이용할 경우 엔티티 빈의 홈 인터페이스에 명세를 하는 create 및 find 함수 그리고 remove 등의 메소드에 대한 명세와 빈 클래스에서의 구현 부분을 나타내고 있다. CMP 방식의 구현일 경우에 find 함수 및 remove 메소드에 대한 구현은 배치 디스크립터 내의 EJB-QL을 실행함으로써 수행되므로 빈 클래스에 해당 메소드의 구현을 나타내지 않으며, 반대로 BMP 방식의 구현일 경우에는 직접 빈 클래스의 메소드 부분에 JDBC를 이용한 코딩을 직접 한다.

5.3 Persistence Business(PB) 패턴

PB 패턴은 엔티티 빈이 가지는 비즈니스 로직을 구현한 메소드에 대한 패턴으로써, 이미 생성된 빈 인스턴스에 대해서 혹은 생성 후 얻어진 컴포넌트 인터페이스를 통해서 수행할 수 있는 메소드들에 대해서 이러한 패턴을 이용한다.

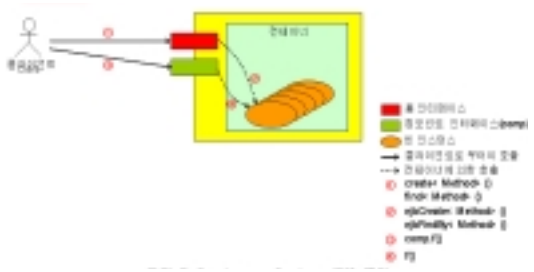

그림 7 Persistence Business(PB) 패턴

그림 7은 엔티티 빈의 CRUD 타입의 메소드 이외에 해당 클래스의 비즈니스 로직을 가지고 있는 메소드들의 수행을 나타낸다. 클라이언트는 홈 인터페이스의 create<Method>()를 통하여 해당 객체를 새로이 생성하거나 또는 이미 생성되어 있는 빈 인스턴스를 find<Method>() 메소드를 통하여 얻어진 컴포넌트 인터페이스를 통하여 비즈니스 메소드인 f()를 수행한다.

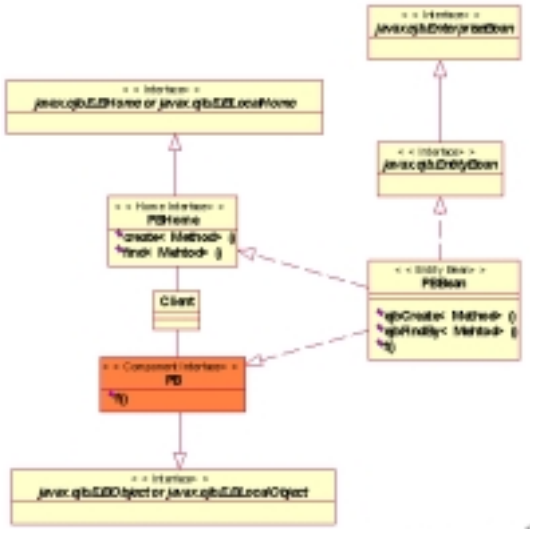

그림 8 엔티티 빈의 컴포넌트 인터페이스 관련 클래스 구조

그림 8은 클라이언트로부터의 create<Method>() 메소드 또는 find<Method>() 메소드를 통하여 얻은 컴포넌트 인터페이스를 통하여 빈 클래스가 가지는 비즈니스 로직에 대한 메소드 f()를 수행할 때 필요한 관련 클래스 및 인터페이스들을 나타내고 있다. 빈 클래스는 javax.ejb.EnterpriseBean 인터페이스를 상속한 javax.ejb.EntityBean 인터페이스를 구현한다.

PBHome이라는 PBBean의 홈 인터페이스가 구현하는 javax.ejb.EJBHome 또는 javax.ejb.EJBLocalHome 인터페이스는 리모트 또는 로컬 클라이언트가 빈 인스턴스의 create, find 및 remove 그리고 홈 메소드를 수행할 수 있도록 하는 메소드들에 대한 명세를 한다. PB라는 컴포넌트 인터페이스는 특정 빈 인스턴스에 대해서 수행할 수 있는 비즈니스 메소드들에 대한 명세를 하는 인터페이스 이고, 엔터프라이즈 빈 컨테이너에 의해 구현된다. javax.ejb.EntityBean 이라는 인터페이스는 모든 엔티티 빈 클래스가 구현해야 하는 인터페이스로써 빈 인스턴스에 대해 수행할 메소드인 callback 메소드들에 대한 인터페이스이다.

```
<PB.java>
public interface PB extends EJBObject {
        . . .
        public <return_type> f() throws RemoteException;
        . . .
}
<PBBean.java>
public class PBBean implements EntityBean {
        . . .
        public <return_type> f(){. . .}
      . . .
}
```

PB.java와 PBBean.java에서는 PB 패턴을 이용하여 엔티티 빈의 비즈니스 로직을 구현한 메소드에 대한 컴포넌트 인터페이스로의 명세 및 빈 클래스에서의 구현 부분을 나타낸다. 비즈니스 로직을 가진 메소드를 컴포넌트 인터페이스인 PB에 명세를 하며, 메소드의 구현 부분을 PBBean 이라는 빈 클래스에 해주게 된다.

5.4 Instance Identifier CRUD(IIC) 패턴

IIC 패턴은 클라이언트와 특정 빈 인스턴스 사이의 'Conversational State'를 갖는 상태유지 세션 빈의 경우에 있어서 클라이언트로부터의 새로운 빈 인스턴스에 대한 생성 및 이미 생성된 빈 인스턴스에 대한 삭제 요청 등을 할 때 이용하는 패턴이다.

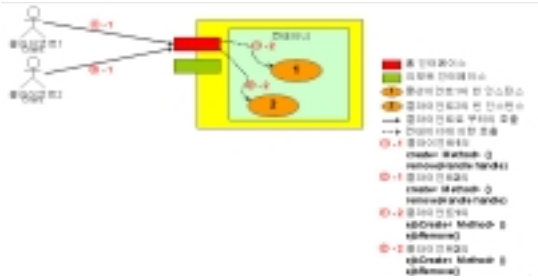

그림 9 Instance Identifier CRUD(IIC) 패턴

그림 9는 상태유지 세션 빈에서 클라이언트가 create<Method>() 메소드 혹은 remove(Handle handle) 메소드에 대한 호출을 했을 때 컨테이너에 의해서 호출되는 callback 메소드인 ejbCreate<Method>()와 ejbRemove() 메소드의 수행 관계를 나타내고 있다. 상태유지 세션 빈은 각각의 클라이언트에 대한 'Conversational State'를 저장하기 때문에 클라이언트에 대해서는 빈 인스턴스가 일대일로 대응된다.

그림 10 상태유지 세션 빈의 홈 인터페이스 관련 클래스 구조

그림 10은 상태유지 세션 빈의 create<Method>()와 remove(Handle handle)를 홈 인터페이스인 IICHome 이라는 인터페이스로 명세하는 것을 나타내며, 이를 구현한 빈 클래스는 IICBean으로써 javax.ejb.Enterprise 인터페이스를 상속한 javax.ejb.SessionBean 인터페이스를 구현한다.

IICHome이라는 IICBean의 홈 인터페이스가 구현하는 javax.ejb.EJBHome 또는 javax.ejb.EJBLocalHome 인터페이스는 리모트 또는 로컬 클라이언트가 빈 인스턴스의 create 및 remove 메소드를 수행할 수 있도록 하는 메소드들에 대한 명세를 한다. javax.ejb.SessionBean 이라는 인터페이스는 모든 세션 빈 클래스가 구현해야 하는 인터페이스로써 빈 인스턴스에 대해 수행할 메소드들 즉, callback 메소드들에 대하여 명세 한다.

```
<IICHome.java>
public interface IICHome extends EJBHome { . . .
        public ComponentInterface create<Method>()
                throws RemoteException;
        . . .
}
<IICBean.java>
public class IICBean implements SessionBean {
        . . .
        public void ejbCreate<Method>(. . .) {. . .}
        public void ejbRemove() {. . .}
        . . .
}
```

IICHome.java와 IICBean.java에서는 IIC 패턴을 적용할 경우 상태유지 세션 빈에서의 빈 인스턴스의 생성 및 삭제에 대한 메소드들의 명세와 IICBean 클래스에서의 구현 부분을 나타내고 있다. 빈 인스턴스의 생성 또는 삭제와 관련된 메소드를 IICHome 이라는 인터페이스에 명세를 하고, IICBean 이라는 빈 클래스에 해당 메소드에 대한 구현을 한다.

5.5 Instance Identifier Business(IIB) 패턴

IIB 패턴은 'Conversational State'를 갖는 상태유지 세션 빈이 가지는 비즈니스 로직을 구현할 경우의 패턴으로써 클라이언트는 create 등을 통해서 얻은 컴포넌트 인터페이스를 가진 메소드를 각각의 빈 인스턴스에 대하여 비즈니스 로직을 가진 메소드를 수행할 경우 이 패턴을 이용한다.

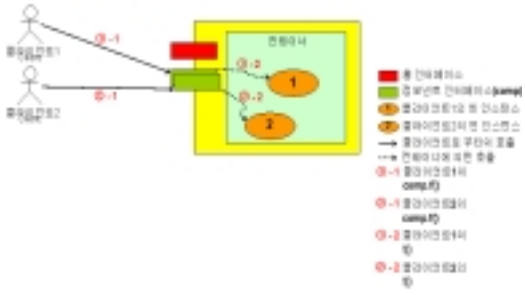

그림 11 Instance Identifier Business(IIB) 패턴

그림 11은 상태유지 세션 빈의 비즈니스 로직을 가진 메소드에 대한 수행을 나타내는 그림으로써, 홈 인터페이스를 통해서 얻은 컴포넌트 인터페이스를 통하여 f() 라는 메소드를 호출하게 되면 컨테이너에 의해서 빈 인스턴스에 대한 메소드가 수행된다.

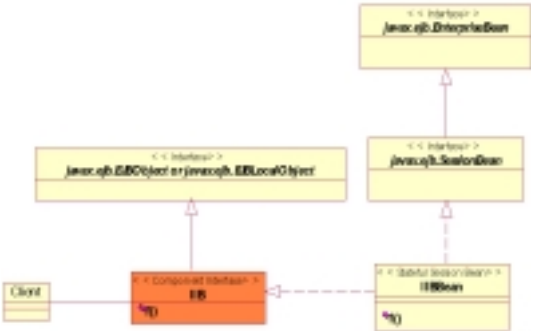

그림 12 상태유지 세션 빈의 컴포넌트 인터페이스 관련 클래스 구조

그림 12는 상태유지 세션 빈의 경우에 컴포넌트 인터페이스에 명세를 한 비즈니스 메소드에 대해서 이를 구현한 빈 클래스가 javax.ejb.EnterpriseBean 인터페이스를 상속한 javax.ejb.SessionBean 인터페이스를 구현한 관계를 나타내고 있다.

IIB라는 IIBBean의 홈 인터페이스가 구현하는 javax.ejb.EJBObject 또는 javax.ejb.EJBLocalObject 인터페이스는 리모트 또는 로컬 클라이언트가 EJB 오브젝트에 대해서 비즈니스 메소드를 수행할 수 있도록 한다. javax.ejb.SessionBean 이라는 인터페이스는 모든 세션 빈 클래스가 구현해야 하는 인터페이스로써 빈 인스턴스에 대해 수행할 메소드들을 명세 한다.

```
<IIB.java>
public interface IIB extends EJBObject {
        . . .
        public <return_type> f(. . .) throws
RemoteException;
        . . .
}
<IIBBean.java>
public class IIBBean implements SessionBean {
        . . .
        public <return_type> f(. . .) {. . .}
        . . .
}
```

IIB.java와 IIBBean.java에서는 IIB 패턴을 적용할 경우 상태유지 세션 빈에서의 비즈니스 로직을 수행하는 메소드인 f()는 컴포넌트 인터페이스인 IIB 인터페이스에 명세를 하며 이를 IIBBean 이라는 빈 클래스에서 메소드에 대한 구현을 한다.

5.6 Common Instance CRUD(CIC) 패턴

CIC 패턴은 'Conversational State'를 유지하는 상태유지 세션 빈과는 달리 모든 클라이언트에 대해서 빈 인스턴스의 구별이 없는 즉, 어떠한 빈 인스턴스라도 클라이언트의 요청에 대한 서비스를 해 줄 수 있는 무상태 세션 빈에 대한 패턴이다. 또한, 클라이언트로부터의 빈 인스턴스의 생성 및 삭제 요청의 경우 이러한 패턴을 사용한다.

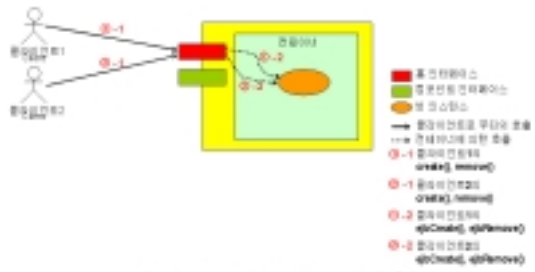

그림 13 Common Instance CRUD(CIC) 패턴

그림 13은 무상태 세션 빈이 가지는 객체 생성 및 삭제에 대한 메소드인 create(), remove() 메소드에 대한 클라이언트로부터의 호출에 따라 컨테이너가 수행하는 ejbCreate() 및 ejbRemove() 메소드의 수행관계를 나타낸 것이다.

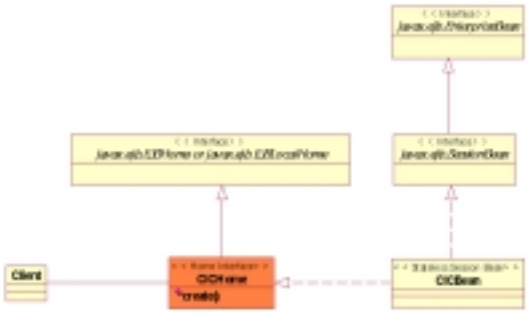

그림 14 무상태 세션 빈의 홈 인터페이스 관련 클래스 구조

그림 14는 무상태 세션 빈이 가지는 create() 메소드의 홈 인터페이스인 CICHome 인터페이스에 명세를 하고 javax.ejb.Enterprise 인터페이스를 상속한 javax.ejb.SessionBean 인터페이스를 구현한 CICBean 이라는 빈 클래스와의 관계를 나타내고 있다.

CICHome이라는 CICBean의 홈 인터페이스가 구현하는 javax.ejb.EJBHome 또는 javax.ejb.EJBLocalHome 인터페이스는 리모트 또는 로컬 클라이언트가 EJB 오브젝트의 create 및 remove 메소드에 대한 명세를 한다. javax.ejb.SessionBean이라는 인터페이스는 모든 세션 빈 클래스가 구현해야 하는 인터페이스로써 빈 인스턴스에 대한 callback 메소드를 명세한다.

```
<CICHome.java>
public interface CICHome extends EJBHome {
        . . .
        public ComponentInterface create() throws
                RemoteException;
        . . .
}
<CICBean.java>
public class CICBean implements SessionBean {
        . . .
        public void ejbCreate() {. . .}
        . . .
}
```

CICHome.java와 CICBean.java에서는 CIC 패턴을 이용하여 구현할 경우, 무상태 세션 빈 인스턴스의 생성을 하는 create 메소드에 대한 홈 인터페이스로의 명세와 이에 대한 callback 메소드인 ejbCreate 메소드를 빈

클래스에 구현한 부분을 나타내고 있다.

5.7 Common Instance Business(CIB) 패턴

CIB 패턴은 무상태 세션 빈의 비즈니스 메소드를 구현할 경우 이용한다. 무상태 세션 빈은 특정 빈 인스턴스에 종속적이지 않기 때문에 컨테이너는 클라이언트의 요청에 대한 서비스를 하기 위해 빈 인스턴스를 선택할 때 가용한 어떠한 인스턴스라도 사용할 수 있다.

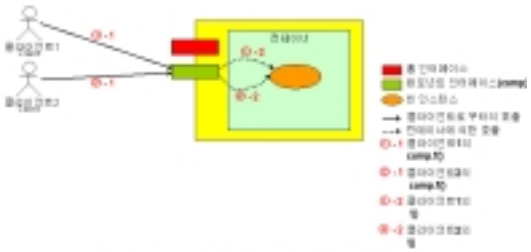

그림 15 Common Instance Business(CIB) 패턴

그림 15는 무상태 세션 빈이 가지는 비즈니스 로직을 수행하는 메소드에 대해서 클라이언트로부터의 호출과 그에 따른 컨테이너의 메소드의 수행관계를 나타내고 있다. 무상태 세션 빈의 경우는 클라이언트에 대한 구별되는 특정 빈 인스턴스가 없기 때문에 하나의 빈 인스턴스만 가지고도 여러 클라이언트로부터의 메소드 수행 요청을 실행할 수 있다.

그림 16 무상태 세션 빈의 컴포넌트 인터페이스 관련 클래스 구조

그림 16은 무상태 세션 빈이 가지는 비즈니스 로직에 대한 메소드의 컴포넌트 인터페이스로의 명세와 javax.ejb.EnterpriseBean 인터페이스를 상속하며 javax.ejb.Sessionbean 인터페이스를 구현한 빈 클래스인 CIB Bean 클래스와의 관계를 나타내고 있다.

CIB라는 CIBBean의 컴포넌트 인터페이스가 구현하는 javax.ejb.EJBObject 또는 javax.ejb.EJBLocalObject 인터페이스는 리모트 또는 로컬 클라이언트가 수행하는 비즈니스 메소드들에 대한 명세를 한다. javax.ejb.SessionBean이라는 인터페이스는 모든 세션 빈 클래스가 구현해야 하는 인터페이스로써 빈 인스턴스에 대해 수행할 callback 메소드들에 대한 명세를 한다.

```
<CIB.java>
public interface CIB extends EJBObject {
        . . .
        public void f() throws RemoteException;
        . . .
}
<CIBBean.java>
public class CIBBean implements SessionBean {
        . . .
        public void f() {. . .}
        . . .
}
```

CIB.java와 CIBBean.java에서는 CIB 패턴을 적용하여 구현할 경우 무상태 세션 빈이 가지는 비즈니스 메소드를 컴포넌트 인터페이스인 CIB라는 인터페이스에 명세를 하고, 이에 대한 구현을 CIBBean이라는 빈 클래스를 통해서 구현을 한다.

5.8 Message Grasp(MG) 패턴

MG 패턴은 JMS의 메시지 소비자 역할을 EJB 2.0에서의 새로운 빈 타입인 메시지 드리븐 빈이 하는 것으로써, 클라이언트로부터의 Destination을 통한 메시지를 빈의 메소드를 통해서 구현할 때 사용하는 패턴이다.

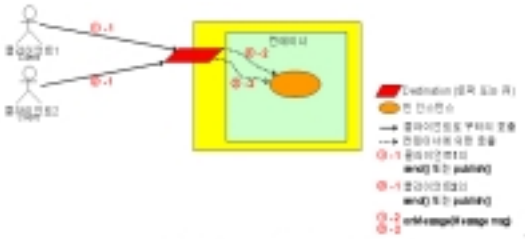

그림 17 Message Grasp(MG) 패턴

그림 17은 EJB 2.0에서 새롭게 소개된 메시지 드리븐 빈에서의 클라이언트로부터의 호출과 그에 따른 컨테이너로부터의 메소드 수행관계를 나타내고 있다. 메시지 드리븐 빈은 엔티티 빈 및 세션 빈에서와 같이 홈 및 컴포넌트 인터페이스를 갖지 않고 큐 또는 토픽이라는 Destination을 갖기 때문에 이를 통하여 클라이언트는 메시지를 send() 또는 publish()라는 메소드를 통하여 보내고, 메시지 리스너가 이를 빈 인스턴스의 onMessage

(Message msg) 메소드에게 전달한다. 메시지 드리븐 빈에서의 메소드는 onMessage(Message msg) 하나의 메소드를 통해서 이루어 진다.

그림 18 메시지 드리븐 빈 관련 클래스 구조

그림 18은 메시지 드리븐 빈에서의 Destination과 이를 상속한 큐 또는 토픽 그리고 javax.ejb.MessageDrivenBean이라는 인터페이스를 구현한 빈 클래스인 MGBean이라는 클래스와의 관계를 나타내고 있다. 메시지 드리븐 빈 클래스의 onMessage(Message msg) 메소드는 javax.jms.MessageListener 라는 인터페이스를 통해서 구현되며, 메시지 드리븐 빈은 엔티티 및 세션 빈과 같이 컴포넌트 및 홈 인터페이스를 갖지 않는다.

클라이언트는 보내고자 하는 메시지를 큐 또는 토픽을 통해서 보내는데 이들의 상위 인터페이스는 javax.jms.Destination이며 MGBean이라는 빈 클래스는 onMessage(Message msg)에 대한 선언을 해놓은 javax.jms.MessageListener 인터페이스와 javax.ejb.MessageDrivenBean 인터페이스를 반드시 구현해야 한다.

```
<MGBean.java>
public class MGBean implements MessageDrivenBean,
        MessageListener {
        . . .
        public void onMessage(Message msg) {. . .}
        . . .
}
```

MG 패턴을 이용하여 구현할 경우 메시지 드리븐 빈 클래스 내에서의 onMessage 메소드에 대한 구현 부분을 나타낸 것으로써, 메시지 드리븐 빈은 javax.ejb.MessageDrivenBean 및 javax.jms.MessageListener 라는 두 개의 인터페이스를 구현한다.

6. 적용절차

본 논문에서는 모델링한 결과를 구현할 때 어떻게 EJB 2.0 규격에 맞는 구현을 할 것인지에 대한 상세한 절차와 방법 그리고 각각의 타입에 따른 패턴을 분류 및 정의하였다. 두 가지 단계로써 빈 타입을 식별하고 각각에 따른 세부 오퍼레이션의 타입을 분류해서 최종적으로 비즈니스 오퍼레이션들이 소스 코드 상으로는 EJB 2.0에서 지원하는 여러 가지 비즈니스 오퍼레이션의 구현 장치 중 어떠한 인터페이스 혹은 메소드를 통하여 구현되는지에 대한 지침을 나타내었다.

그림 19 단계 및 패턴 적용

모델링한 결과가 소스 상으로 나타나기까지의 과정은 구체적이고 체계적인 방법이 필요하며, 본 논문은 이러한 절차와 방법에 대해서 절차적이며 체계적인 방법과 매핑 타입에 따른 정형화된 패턴을 제시하여 EJB 개발자 입장에서는 이 기법을 통해서 좀 더 정형화되고 효율적으로 EJB 소스 코드 상으로의 매핑을 할 수가 있다. 또한, EJB 2.0에서 지원하는 여러 장치들을 개발자가 충분히 활용할 수 있도록 하는 지침이 된다.

그림 19는 본 논문에서 제시한 두 단계와 비즈니스 오퍼레이션의 각 유형별 설계 패턴을 적용하는 전체적인 절차를 나타내고 있다. 빈 타입을 식별하는데 있어서 단계 I에서 제시한 방법을 이용하여 어떠한 빈으로 매핑할 것인지를 결정하고 병렬적으로 비즈니스 오퍼레이션 유형에 따른 설계 패턴을 적용함으로써, 식별된 각각의 빈 타입에 따라 클래스 내부의 메소드 들이 EJB가 제공하는 어떠한 장치들을 이용하여 매핑 되는지 결정한다. 마지막으로는 각각의 인터페이스 및 빈을 어떻게 구현하며 어떠한 구현 제약 사항들이 있는지에 대한 단계를 나타낸다.

그림 20은 앞의 두 단계를 거치면서 식별된 엔티티 빈, 상태유지 세션 빈, 무상태 세션 빈, 그리고 메시지 드리븐 빈 등의 총 4 가지 빈 클래스의 유형과 그에 따라 구현될 각각의 인터페이스 및 메소드들에 적용시키는 패턴들을 나타내고 있다. 두 번째 단계에서 결정된 인터페이스의 유형은 본 논문에서 제시한 8 가지의 패턴으로 대응시킬 수 있다. 각 패턴은 빈 클래스의 타입

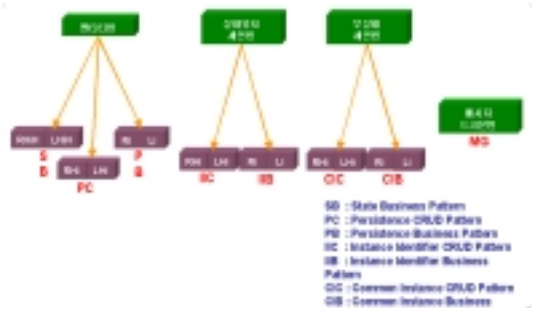

그림 20 각 인터페이스별 패턴 적용 유형

결정 후 분류되는 인터페이스 유형 각각에 대하여 비즈니스 오퍼레이션들을 서로 적용시킨다.

두 가지 단계 및 패턴을 적용해서 최종적으로 분석 단계에서의 각 비즈니스 오퍼레이션들이 EJB 2.0 플랫폼 상에서 어떻게 구현되는지에 대한 지침을 제시하였는데, 이러한 지침과 패턴을 기반으로 하여 비즈니스 오퍼레이션들은 EJB 2.0에서 지원하는 여러 가지 장치들을 충분히 활용하여 체계적이며 효율적으로 매핑될 수 있다.

7. 사례연구

본 논문에서 제시된 기법들의 유용성을 시험하기 위하여 적용된 사례연구는 병원관리시스템(HMS)이다. HMS는 크게 환자의 등록 및 환자 정보 변경을 위한 환자관리, 진료를 위한 예약의 등록, 변경 및 삭제 등을 하는 예약 관리, 진료 스케줄 작성 및 삭제에 관련된 스케줄 관리, 환자 진료에 관한 진료 관리 및 진료 내용에 대한 청구서 작성, 전송 등에 관한 청구서 관리 등의 기능으로 이루어져 있다.

7.1 객체 지향 모델링

객체 지향 모델링은 기능성 위주의 Use Case 다이어그램, 데이타 구조 위주의 클래스 다이어그램 및 객체들 간의 메시지 흐름을 나타내는 시퀀스 다이어그램 등으로 이루어진다.

그림 21에서와 같이 HMS는 총 21개의 Use Case로 구성되어 있으며, 이 중 대표적으로 '예약하다' 라는 기능을 나타내는 'Make Appointment'와 '주기적인 청구서 발행'이라는 기능을 나타내는 'Generate Bill on Cycle'이라는 Use Case에 관련된 비즈니스 오퍼레이션의 구현에 대하여 알아본다. Use Case 다이어그램 상에 각각 독립된 기능으로 식별된 두 가지의 Use Case에 대하여 클래스 다이어그램 및 시퀀스 다이어그램을 함께 이용하여 어떻게 비즈니스 오퍼레이션 구현이 되는지 살펴본다.

그림 21 병원 관리 시스템의 Use Case 다이어그램

그림 22는 HMS의 전체 클래스들을 나타내는 클래스 다이어그램이다. 클래스의 이름만으로 표기하였으며, 구현 이전 단계에 도출되는 상세 클래스 다이어그램이다.

그림 22 병원 관리 시스템의 클래스 다이어그램 1

그림 23은 병원 관리 시스템의 클래스 다이어그램 중 주요한 기능들을 수행하는 Treatment, AppointmentClt, GenerateBillCtl 등의 클래스를 포함한 클래스 다이어그램의 일부분이다. 각각의 클래스들은 어트리뷰트와 오퍼레이션들을 포함한 상세 수준의 클래스들이다.

그림 24에서는 병원 관리 시스템의 기능들 중 'Make Appointment'라는 기능에 대한 메시지 흐름을 나타내었다. Use Case 다이어그램 상에 나타난 기능들 중 독립된 하나의 기능에 대하여 참여하는 객체들 간의 메시지 흐름이 어떠한가를 알 수 있다. 이를 통하여 기능 수행에 있어 필요한 세부 오퍼레이션들이 이후의 매핑 과정

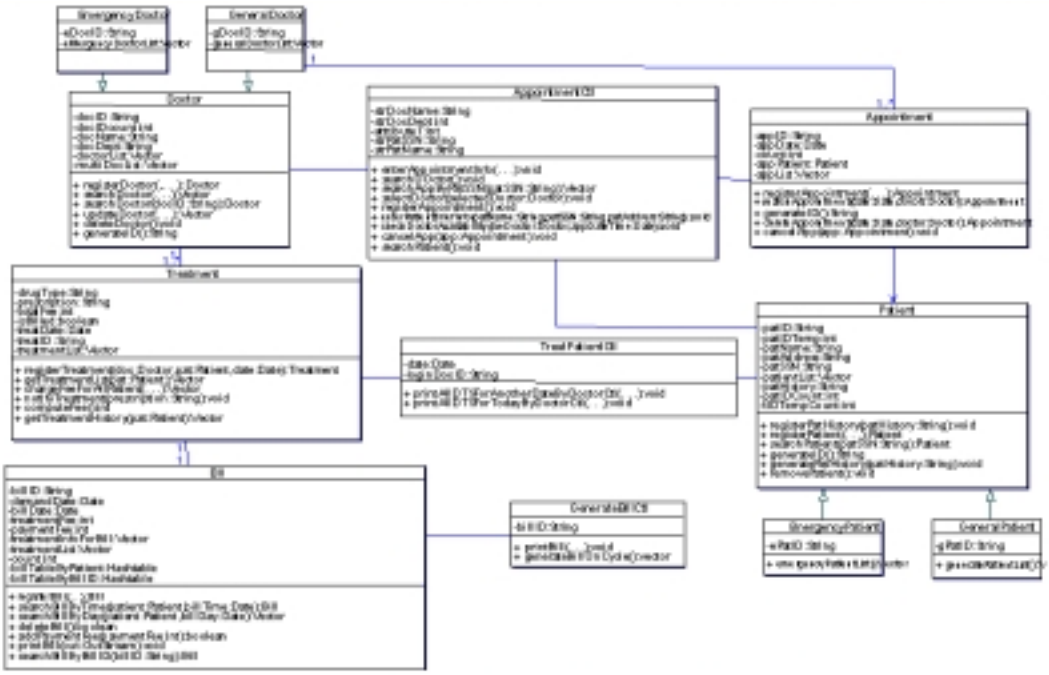

그림 23 병원 관리 시스템의 클래스 다이어그램 2

그림 24 병원 관리 시스템의 시퀀스 다이어그램 1

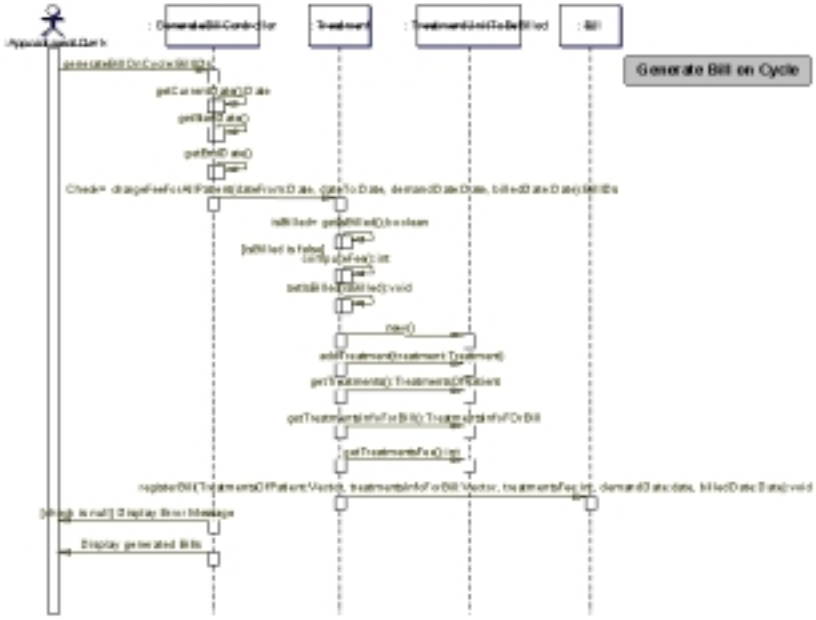

그림 25 병원 관리 시스템의 시퀀스 다이어그램 2

에서 어떻게 EJB에 종속적인 코드로써 나타나는지를 확인할 수 있다.

그림 25에서는 'Generate Bill on Cycle'이라는 기능에 대한 메시지 흐름을 나타낸다. 비즈니스 오퍼레이션을 구현하기 위한 방법 중 메시지 드리븐 빈으로 매핑이 이루어지는 경우에는 해당 메시지가 리턴 값이 없으면서 비동기적으로 수행될 수 있는 성격의 오퍼레이션이어야 한다. 따라서, 이러한 오퍼레이션들의 성격을 파악하기 위하여 시퀀스 다이어그램을 이용하여 분류한다. 본 Use Case에서는 'AppointmentClerk'으로 부터의 기능 수행 오퍼레이션인 'generateBillOnCycle'이라는 메시지를 이러한 성격의 오퍼레이션으로 간주한다.

7.2 비즈니스 오퍼레이션의 상세 설계

7.2.1 빈 타입 식별

본 절은 비즈니스 오퍼레이션 구현을 위한 첫 번째 단계로써 주어진 객체 지향 모델링의 클래스 다이어그램에 나타난 클래스들을 그림 1의 빈 타입 식별 방법 및 기준에 따라 EJB에서의 빈 타입으로 각각 식별하는 것이다. 총 네 가지의 가능한 빈 타입들 중에서 세 가지 빈 타입이 식별되었다.

병원 관리 시스템에 대하여 비즈니스 오퍼레이션에 대한 상세 설계를 위한 첫 번째 단계로써 빈 타입을 식별한다. 그림 1에서의 식별 기준에 따라 표 2에서와 같이 엔티티 빈, 상태유지 세션 빈 및 메시지 드리븐 빈 등 세 가지 종류의 빈 타입이 식별되었다. 메시지 드리븐 빈의 경우는 그림 2에서의 식별 기준이 적용되어 분류 되었다.

비즈니스 오퍼레이션 타입 분석 및 패턴 적용

본 절은 두 번째 단계로써 식별된 빈 타입 각각에 대하여 각 클래스들이 가지는 오퍼레이션들을 어떠한 타입으로 식별을 하여 적절한 패턴을 적용시킬지를 결정하는 단계이다. 그림 2의 비즈니스 오퍼레이션 타입 결정 방법 및 절차에 따라 각 오퍼레이션에 대하여 오퍼

식별 기준	해당 클래스	빈 타입
BT-1(yes)	Staff, Clerk, AccountClerk, AppointmentClerk, ScheduleClerk, Doctor, EmergencyDoctor, GeneralDoctor, Treatment, Bill, Payment, PaymentThroughBank, PaymentByCash, Patient, EmergencyPatient, GeneralPatient, Appointment, DailyTreatmentSchedule	엔티티 빈
BT-1(no) -> BT-2(yes)	TreatmentCtl, AppointmentCtl	상태유지 세션 빈
BO-1(no) -> BO-3(no)	GenerateBillCtl	메시지드리븐 빈

표 2 병원 관리 시스템에 대한 빈 타입 식별

표 3 비즈니스 오퍼레이션의 타입 분석 및 적용 패턴

클래스명	오퍼레이션명	식별 기준	오퍼레이션타입	적용패턴
Treatment	registerTreatment	BO-1(yes) -> BO-2(no) ->BO-5(no) -> BO-7(yes)	LHi	PC
	getTreatmentList	BO-1(yes) -> BO-2(no) ->BO-5(no) -> BO-7(yes)	LHi	PC
	chargeFeeForAllPatient	BO-1(yes) -> BO-2(no) ->BO-5(no) -> BO-7(no)	Li	PB
	notifyTreatment	BO-1(yes) -> BO-2(no) ->BO-5(no) -> BO-7(no)	Li	PB
	computeFee	BO-1(yes) -> BO-2(yes) -> BO-4(no)	LHiH	SB
	getTreatmentHistory	BO-1(yes) -> BO-2(no) ->BO-5(no) -> BO-7(no)	Li	PB
AppointmentCtl	enterAppointmentInfo	BO-1(yes) -> BO-2(no) -> BO-5(yes) -> BO-6(no)	Ri	IIB
	searchDoctor	BO-1(yes) -> BO-2(no) -> BO-5(yes) -> BO-6(no)	Ri	IIB
	searchAppByPatSSN	BO-1(yes) -> BO-2(no) -> BO-5(yes) -> BO-6(no)	Ri	IIB
	selectDoctor	BO-1(yes) -> BO-2(no) -> BO-5(yes) -> BO-6(no)	Ri	IIB
	registerAppointment	BO-1(yes) -> BO-2(no) -> BO-5(yes) -> BO-6(yes)	RHi	IIC
	enterPatientBriefInfo	BO-1(yes) -> BO-2(no) -> BO-5(yes) -> BO-6(no)	Ri	IIB
	checkDoctorAvailability	BO-1(yes) -> BO-2(no) -> BO-5(yes) -> BO-6(no)	Ri	IIB
	cancelApp	BO-1(yes) -> BO-2(no) -> BO-5(yes) -> BO-6(no)	Ri	IIB
	searchPatient	BO-1(yes) -> BO-2(no) -> BO-5(yes) -> BO-6(no)	Ri	IIB
GenerateBillCtl	printBill	BO-1(no) -> BO-3(no)	MG	MG
	generateBill	BO-1(no) -> BO-3(no)	MG	MG

레이션 타입을 결정하고 5장에서 제시된 패턴들을 각각 적용한다.

표 3에서와 같이 Treatment 클래스의 register Treatment와 AppointmentCtl 클래스의 registerAppointment 메소드는 모두 홈 인터페이스 상에 명세를 한다. 또한, TreatPatientCtl 클래스는 클라이언트에게 직접 노출이 되어 접근 가능한 리모트 빈으로써 구현되며, Treatment 클래스는 이러한 리모트 클래스를 통하여 접근 가능한 로컬 빈 형태로 구현된다. GenerateBillCtl 클래스는 그림 25에서와 같이 generateBillOnCycle 메소드를 비동기적이며 리턴 값이 없는 메시지 드리븐 빈의 메소드로 매핑할 수 있다.

7.2.3 인터페이스 및 빈 클래스 구현

표 3에서 식별된 각각의 클래스들이 가지는 오퍼레이션의 타입 및 적용 패턴을 이용하여 다음과 같이 홈 인터페이스, 컴포넌트 인터페이스 및 빈 클래스를 구현하게 된다.

그림 26 Treatment 클래스의 구현

그림 27 AppointmentCtl 클래스의 구현

'GenerateBillCtl' 클래스는 메시지 드리븐 빈으로 매핑이 이루어지며, 메시지 드리븐 빈의 특성상 다른 빈과는 다르게 홈 인터페이스나 컴포넌트 인터페이스를 갖지 않고 빈 클래스만을 갖는다.

Treatment, AppoitmentCtl 및 GenerateCtl 세 개의 클래스에 대하여 구현에 필요한 홈 인터페이스, 컴포넌

그림 28 GenerateBillCtl 클래스의 구현

트 인터페이스 및 빈 클래스에 대한 코드를 나타내었다. 객체 지향 모델링의 결과를 표 2와 표 3을 통하여 식별 및 적용 패턴을 구분하여 최종적으로 소스 코드로 어떻게 구현하는지를 나타내었다. 예제를 통하여 나타낸 세 개의 클래스들은 빈 타입을 나타내기 위한 대표적인 클래스들이며, 본 논문에서 제시한 8가지 적용 가능한 패턴 중 6가지의 패턴에 대한 적용 사례를 나타내었다.

8. 맺음말

객체지향 분석을 통해 도출된 각 클래스들이 가지는 오퍼레이션들은 구현 시에 어떻게 매핑 될 것인지가 중요하며 이러한 분석 내용들이 소스 코드로 정확하게 반영되는 것 또한 중요하다. 본 논문에서는 분석한 결과를 EJB를 이용하여 구현할 때 각각의 비즈니스 오퍼레이션들이 어떻게 EJB에서 지원하는 여러 장치들을 이용하여 구현될 수 있는지에 대한 절차와 방법들을 제시하였다.

빈 타입 단계에서는 EJB의 빈으로 매핑 될 수 있도록 각각의 클래스들에 대한 정확한 빈 타입을 식별하고, 두 번째 단계에서는 식별된 빈 타입에 따라 비즈니스 오퍼레이션들이 EJB에서 지원하는 여러 가지 인터페이스 및 메소드 들을 이용하여 어떻게 구현될 수 있는지에 대한 식별 과정을 체계적이며 절차적인 방법으로 나타내었다. 따라서, 이 두 단계를 거치면서 분석된 클래스들에 대한 오퍼레이션들의 구현 가능한 인터페이스 및 메소드를 결정할 수 있다.

빈 타입과 구현 가능한 인터페이스 및 메소드를 결정하면 그에 따른 구현을 할 때, 어떠한 제약 사항들을 지켜야 하며 구현 종속적인 사항들은 어떤 것들이 있는지에 대해서 나타내었다. 또한, 두 단계를 거치면서 결정되는 각각의 인터페이스 및 메소드 들에 대해서 패턴을 분류 및 정의 하였다. 마지막으로 병원관리시스템에 대한 사례 연구를 통하여 제시된 기법 중에서 네 가지의 구현 가능한 빈 타입 중 세 가지의 빈 타입과 8가지의 적용 가능한 패턴 중 6가지의 패턴에 대한 적용 및 구현에 대한 예를 나타내었다. 따라서, 본 논문에서 제시하는 총 8 가지의 패턴을 사용함으로써 EJB를 이용한 구현을 할 때 필요한 비즈니스 오퍼레이션에 대한 매핑을 좀 더 체계적이며 절차적인 방법에 따라 할 수 있다.

참 고 문 헌

[1] Marshall, C., *Enterprise Modeling with UML*, Addison Wesley, 2000.

[2] Eriksson, H., *Business Modeling with UML*, John Wiley and Sons, Inc., 2000.

[3] Larman, C., *Applying UML and Patterns, An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall PTR, 2002.

[4] Matena, V., *Applying Enterprise JavaBeans*, Addison Wesley, 2001.

[5] Gomez, P., *Professional Java 2 Enterprise Edition with BEA Weblogic Server*, Wrox Press, 2000.

[6] Perrone, P., *Building Java Enterprise Systems with J2EE*, Sams Publishing, 2000.

[7] Girdley, M., *J2EE Applications and BEA Weblogic Server*, Prentice Hall PTR, 2002.

[8] Arrington, C., *Enterprise Java with UML*, John Wiley and Sons, Inc., 2001.

[9] Alur, D., *CORE J2EE Patterns*, Sun Microsystems Press/Prentice Hall PTR, 2001.

[10] Adatia, R., *Professional EJB*, Wrox Press, 2001.

[11] Roman, E., *Mastering Enterprise JavaBeans Second Edition*, John Wiley and Sons, Inc., 2002.

[12] Giotta, P., *Professional JMS Programming*, Wrox Press, 2000.

[13] Monson-Haefel, R., *JAVA Message Service*, O'Reilly & Associates, Inc., 2001.

[14] Marinescu, F., *EJB Design Patterns*, John Wiley and Sons, Inc., 2002.

[15] Gamma, E., *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.

[16] Grand, M., *Patterns in Java Volume 2*, John Wiley and Sons, Inc., 1999.

[17] Grand, M., *Java Enterprise Design Patterns*, John Wiley and Sons, Inc., 2002.

[18] Flanagan, D., *Java Enterprise in a Nutshell*, O'Reilly & Associates, Inc., 1999.

[19] Cooper, J., *The Design Patterns*, Addison Wesley, 1998.

[20] Stark, S., *JBoss Administration and Development*, JBoss Group, 2001.

[21] Enterprise JavaBeans Specification, Version 2.0 Final Release, Sun Microsystems, 2001.

박 지 환

2001년 숭실대학교 컴퓨터학부 졸업. 2001년 3월 ~ 현재 숭실대학교 컴퓨터학과 석사과정 재학중. 관심분야는 EJB 기반 개발 기법, 컴포넌트 개발 기법

이 상 덕

1978년 KIST 연구원. 1980년 부산대학교 대학원 경영학과 졸업(MIS 전공). 1985년 ~ 1988년 사우디 국립과학기술연구소 파견근무(선임연구원). 1990년 시스템공학연구소 의료정보개발실장. 1994년 시스템공학연구소 책임연구원. 1997년 시스템공학연구소 개방형 S/W연구실장. 1998년 ETRI 컴포넌트기반 S/W 연구 팀장. 2000년 ETRI S/W 시험센터 센터장. 2001년 12월 ~ 현재 TTA IT시험연구소 S/W 시험센터 센터장

김 수 동

1984년 Northeast Missouri State University 전산학 학사. 1988년 The University of Iowa 전산학 석사. 1991년 The University of Iowa 전산학 박사. 1991년 ~ 1993년 한국통신 연구개발단 선임연구원. 1993년 ~ 1994년 The University of Iowa 교환교수. 1994년 ~ 1995년 현대전자 소프트웨어연구소 책임연구원. 1995년 9월 ~ 현재 숭실대학교 컴퓨터학부 부교수. 관심분야는 컴포넌트 개발 방법론, 객체지향 개발 방법론, 분산 시스템, 컴포넌트 정형명세, 소프트웨어 시험