EJB 기반 컴포넌트의 가변성 맞춤화 기법

(A Method to Customize the Variability of EJB-Based Components)

민 현 기 † 김 성 안 †† 이 진 열 †† 김 수 동 †††
(Hyun Gi Min) (Sung Ahn Kim) (Jin Yeal Lee) (Soo Dong Kim)

요 약 컴포넌트 기반 소프트웨어 개발 (CBD) 기술은 재사용 가능한 컴포넌트를 조립하여, 효율적으로 소프트웨어를 개발함으로써 개발 노력과 상품화 시간을 줄여주는 새로운 기술로 정착되고 있다. 이러한 CBD 컴포넌트는 한 도메인의 표준이나 공통적인 기능을 제공하여야 재사용성이 높아진다. 특히, 공통성 안의 미세한 가변적인 부분도 모델링하고, 이러한 가변성을 각 어플리케이션의 특성에 적합하게 특화 할 수 있도록 설계되어야 한다. Enterprise JavaBeans(EJB)는 컴포넌트를 구현하는 최적의 환경으로 인식되어 왔다. 그러나 EJB는 컴포넌트를 특화 할 수 있는 설계 기법을 제공하지 않기 때문에 비즈니스 컴포넌트의 재사용성이 낮아진다. 따라서 본 논문에서는 EJB 환경에서 컴포넌트의 가변성을 설계하는 효율적인 기법을 제안한다. 세 가지 컴포넌트 특화 기법인 선택형 기법, 플러그인 기법, 외부 프로파일 기법을 적용하여 EJB를 위한 컴포넌트 특화 기법을 제안한다. 제시한 기법을 다른 연구의 다양한 기준과 비교하여 제시한 기법의 유용성에 대해서 평가한다.
키워드 : 컴포넌트 기반 개발(CBD), Enterprise JavaBeans(EJB), 컴포넌트 가변성, 컴포넌트 특화

Abstract Component-Based Development (CBD) has emerged as a new effective technology that reduces development cost and time-to-market by assembling reusable components in developing software. The degree of conformance to standards and common features in a domain largely determines the reusability of components. In addition, variability within commonality should also be modeled and customization mechanism for the variability should be designed into components. Enterprise JavaBeans (EJB) is considered a most suitable environment for implementing components. However, the reusability of EJB is limited because EJB does not have built-in variability design mechanisms. In this paper, we present efficient variability design techniques for implementing components in EJB. We propose a method to customize the variability of EJB-based components by applying three variability design mechanisms; *selection, plug-in, and external profile.* And we elaborate the suitable situations where each variability design mechanism can be applied, and conduct a technical comparison to other approaches available.
Key words : Component-Based Development (CBD), Enterprise JavaBeans (EJB), Component Variability, Component Customization

· 이 논문은 2005년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2004-005-D00172)
† 정 회 원 : 숭실대학교 컴퓨터학과
hgmin@otlab.ssu.ac.kr
†† 학생회원 : 숭실대학교 컴퓨터학과
sakim@otlab.ssu.ac.kr
jylee@otlab.ssu.ac.kr
††† 종신회원 : 숭실대학교 컴퓨터학과
sdkim@ssu.ac.kr
논문접수 : 2005년 12월 21일
심사완료 : 2006년 4월 18일

1. 서 론

어플리케이션 개발기술의 하나인 CBD 기술은 효율적인 재사용 기술이며 산업계에서 보편적으로 사용되는 패러다임이다[1]. 컴포넌트는 소프트웨어의 재사용 부품으로서 단일 객체보다 단위가 크고 특정 어플리케이션으로 특화하기 위한 기법을 적용함으로써 재사용성이 객체보다 좋다. 따라서 컴포넌트 기반 개발 환경에서 소프트웨어를 개발하는 것은 소프트웨어의 재사용 측면에서 비용과 노력을 줄이는 결과를 가져온다. 산업계에서

는 J2EE의 엔터프라이즈 자바 빈즈(Enterprise JavaBeans, EJB)[2]를 컴포넌트 개발 환경을 위한 최적의 솔루션으로서 채택하고 있다.

이러한 재사용을 높이기 위해서 컴포넌트 내의 가변적인 부분의 모델링과 이 가변성을 특정 어플리케이션에 맞게 특화 시킬 수 있는 기법이 필요하다. 일반적으로 알려진 가변성 특화 기법은 선택형 기법, 플러그인 기법 등이 있다. 그러나 EJB는 가변성을 표현하기 위한 장치와 이 가변성을 특화할 수 있는 기법을 제공하지 않는다. 또한 컴포넌트의 단위로서 단일 엔터프라이즈 빈을 사용하기 때문에 CBD에서 큰 재사용의 단위를 지원하지 못한다.

따라서 본 논문에서는 복합 컴포넌트의 설계 기법을 이용하기 위해 다수의 엔터프라이즈 빈을 내부에 숨기고 이를 중재하는 빈을 두어 컴포넌트 내부의 워크플로우를 클라이언트가 직접 접근하지 못하게 외부에 인터페이스만을 제공하는 기법을 소개한다. 그리고 세가지 타입의 인터페이스인 *Provided* 인터페이스, *Required* 인터페이스, *Customize* 인터페이스를 EJB에서 어떻게 설계할 것인가를 제시한다. 이러한 EJB에서의 컴포넌트 설계를 기반으로 EJB 기반 컴포넌트를 특화하기 위한 선택형 기법, 플러그인 기법, 외부 프로파일 기법을 제안하며 기존 연구와의 비교를 통해 제안된 기법을 평가한다.

본 논문의 2장 관련 연구에서는 기존의 가변성 기법에 대한 연구를 소개하고 3장 기반 연구에서는 컴포넌트의 가변성, 가변점 및 가변치에 대한 정의와 가변성의 다섯 가지 타입과 컴포넌트 가변성의 식별 단계에 따른 두 가지 영역에 대해서 설명한다. 그리고 컴포넌트 인터페이스의 세 가지 타입에 대해서 설명하고 컴포넌트 특화를 위한 세가지 기법에 대해서 설명한다. 4장은 EJB에서 CBD의 큰 단위의 재사용 컴포넌트를 지원하기 위한 복합 컴포넌트 설계모델과 세 가지 타입의 컴포넌트 인터페이스 설계 기법을 제안한다. 5장에서는 앞서 설명한 컴포넌트의 세 가지 특화기법에 대해서 EJB를 위한 컴포넌트 커스터마이제이션 기법을 제안한다. 6장에서는 제안된 기법과 기존 연구를 평가하며 본 논문의 결론은 7장에서 기술한다.

2. 기반 연구

2.1 가변성 구현 기법

Anastasopoulos, M.과 Gacek, C.는 가변성 기법을 다음과 같이 제안하였다. 가변성 기법은 집합/위임(Aggregation/Delegation), 조건적 컴파일(Conditional Compilation), 동적 클래스 로딩(Dynamic Class Loading), 동적 링크 라이브러리(Dynamic Link Libraries, DLL), 프레임(Frame), 상속(Inheritance), 오버로딩(Overloading), 파라메터라이제이션(Parameterization), 속성(Property), 정적 라이브러리(Static Library), AOP로 분류하였다[4]. 집합/위임 기법은 가변적인 행위를 가지고 있는 다른 객체로 위임하는 기법이다. 조건적 컴파일은 선택된 코드 세그먼트를 포함시킬 것인지 제외할 것인지를 여러 코드 세그먼트를 컨트롤하여 편집하는 기법이다. 동적 클래스 로딩은 런타임시 필요할 때 메모리에 적절한 클래스를 구현하여 로드하는 기법이다. 프레임 기법은 적절한 행위를 명세한 프레임을 프로그램에 포함하는 기법이다. 파라메터라이제이션은 가변치인 특정값을 전달하기 위한 방법이고, 동적 라이브러리 기법은 컴파일 후에 어플리케이션에 연결될 수 있게 외부 함수의 집합을 미리 포함하고 있는 기법이다. 제안 된 기법 중에 집합/위임, 동적 링크 라이브러리, 파라메터라이제이션 기법만이 블랙박스 컴포넌트에서 적용 가능하다.

Svahnberg, M.과 Bosh, J.는 다음과 같은 5가지의 가변성 기법을 제안하였다. 제안 된 기법은 상속, 확장(Extension), 파라메터라이제이션, 구성(Configuration), 파생된 컴포넌트 생성(Generation of Derived Components)이다[5]. 확장 기법은 한 컴포넌트의 여러 부분들이 추가적인 행위에 의해 확장되고, 가변치들의 집합으로부터 선택될 때 사용된다. 구성 기법은 특화된 제품의 형태로 코드 저장소에 있는 파일 또는 소스 코드 세그먼트의 선택을 가능하게 한다. 파생된 컴포넌트 생성은 적절한 파라메터들의 집합을 직접 코딩하는 것이다. 그러나 상속과 컴포넌트 생성 기법은 블랙박스 컴포넌트에서 사용할 수 없다.

Keepence는 다음과 같이 3가지의 가변성 설계 패턴을 제안 하였다. 제안된 패턴은 단일 어댑터(Single Adapter), 다중 어댑터(Multiple Adapter), 선택 패턴(Option Patterns)이다[18]. 단일 어댑터는 일반적인 휘처들을 상위 기본 클래스에 두고 하위 클래스에서 상속받아 개별화 하는 것이다. 이때 하위 클래스는 하나만 인스턴스화 될 수 있다. 다중 어댑터는 단일 어댑터와 유사하지만 하나 이상의 하위 클래스들이 인스턴스화 될 수 있다는 점이 다르다. 선택 패턴은 가변성을 위한 두 개의 관련된 클래스를 생성한다. 그러나 Keepence의 세 가지 가변성 설계 패턴은 가변치의 범위를 제안하고 있지만 세부적인 구현 기법은 명세하고 있지 않다.

Catalysis는 가변성 구현 기법을 위해서 상속 및 템플릿 기법과 다형성 및 전달 기법을 제시 하였다. 상속 및 템플릿 기법은 기본 클래스에 대해서 다양한 하위 클래스를 두어 구현하는 방법이다. 다형성 및 전달 기법은 상속 및 템플릿 기법에서의 단점을 보완하여 전달되는 객체에 따라서 기능 수행시간에 동적으로 원하는

기능을 수행하도록 한다. 이 두 개의 기법은 화이트 박스 컴포넌트를 특화하기 위한 객체지향 구현 기법을 사용한다. 그러나 상속 및 템플릿 기법은 블랙 박스 컴포넌트를 특화하기 위한 기법을 가지고 있지 않다. 그리고 다형성 및 전달 기법은 클래스 리플렉션 기법을 지원하는 환경에서만 블랙박스 컴포넌트를 위한 특화가 가능하다.

2.2 컴포넌트 가변성

컴포넌트는 사용자의 공통적인 요구사항인 공통성(Commonality)을 가지고 있다. 컴포넌트의 가변성(Variability)은 어플리케이션에 따라 변할 수 있는 공통성 내에 존재하는 성질을 의미한다[6]. 컴포넌트의 가변점(Variation Point)은 컴포넌트를 사용하는 멤버들간에 서로 다른 부분인 가변성이 발생하는 지점이다. 이러한 가변점은 특정 어플리케이션에 요구사항을 위한 설정 값인 가변치(Variant)로 설정되어야 사용할 수 있다[7].

컴포넌트가 가질 수 있는 가변성의 타입은 속성(Attribute) 가변성, 로직(Logic) 가변성, 워크플로우(Workflow) 가변성, 영속성(Persistency) 가변성, 인터페이스(Interface) 가변성의 다섯 가지로 분류할 수 있다[9]. 먼저 속성 가변성은 가변점이 컴포넌트 내부의 속성에 존재하는 것을 나타낸다. 속성 가변성은 속성의 개수의 차이나 속성이 가지는 자료형의 차이의 형태로 나타난다. 로직 가변성은 알고리즘이나 논리적 순서에 가변점이 존재하는 것을 나타낸다. 멤버들 사이의 서로 다른 처리 로직이 존재하는 경우에 발생한다. 워크플로우 가변성은 로직을 수행하기 위한 메소드들의 호출순서가 멤버들 사이에 다른 경우에 이러한 일련의 메소드 호출 순서에 가변점이 존재하는 것을 나타낸다. 영속성 가변성은 가변점이 물리적인 스키마나 외부 저장장치에서의 영속적인 속성의 표현이 멤버들 사이에 다른 경우에 존재하는 것을 나타낸다. 마지막으로 인터페이스 가변성은 가변점이 인터페이스의 메소드 시그네처가 멤버들 사이에 다른 경우에 존재하는 것을 나타낸다[15].

컴포넌트 내부에 가변성은 식별 단계에 따라 두 개의 범위로 나눌 수 있다[8]. 분석 단계에서 멤버들이 가질 수 있는 모든 가변치들이 식별 가능하고 변경 가능성이 적어 모든 가변치를 컴포넌트 내부에 포함하는 것을 Close 영역이라 한다. 반면에 멤버들이 가질 수 있는 가변치의 개수가 다양하고 변화 가능성이 높아 컴포넌트 내부에 가변치를 가지는 것이 경제성이 떨어지거나 분석 단계에서 모든 가변치를 식별 할 수 없으므로 컴포넌트를 사용하는 멤버에 의해 정의되어 컴포넌트 외부에 가변치가 존재하는 것을 Open 영역이라 한다[9].

컴포넌트 사용자의 요구사항에 맞는 가변치가 컴포넌트의 가변점에 설정되어야 하는데 이것을 컴포넌트 특화(Customization)라고 한다. 다양한 어플리케이션의 비슷하지만 다른 비즈니스 업무를 수행하기 위해서는 컴포넌트를 특화 할 수 있는 장치가 포함되어야 하며 이를 통해 컴포넌트의 재사용성을 높일 수 있다.

2.3 컴포넌트 인터페이스

컴포넌트는 여러 개의 인터페이스를 가지며, 본 논문에서는 컴포넌트 인터페이스를 컴포넌트가 제공하는 기능을 정의하는 *Provided* 인터페이스, 컴포넌트가 필요로 하는 기능을 명세하는 *Required* 인터페이스, 컴포넌트를 커스터마이즈 하기 위한 *Customize* 인터페이스의 세 가지 유형으로 정의한 모델을 채택한다[7]. 이러한 인터페이스는 그림 1과 같이 표현된다. 인터페이스는 시그네처와 제약사항을 명세한다. 이러한 인터페이스 규약들에 따라 컴포넌트가 구현된다. *Provided* 인터페이스는 컴포넌트가 제공하는 기능군을 정의한다. 이는 컴포넌트의 주요 기능을 결정하는 인터페이스로 일반적인 의미의 컴포넌트 인터페이스를 가리킨다.

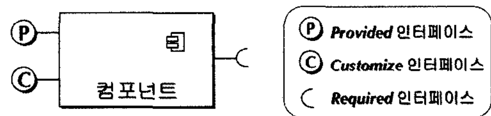

그림 1 컴포넌트 인터페이스

Required 인터페이스는 컴포넌트 실행 시 요구되는 다른 컴포넌트의 기능군을 명세 한다. 물리적인 인터페이스를 할당하지 않고 명세서를 제공하여 컴포넌트간 의존 관계를 명세한다. 즉, 실제 물리적인 컴포넌트 인터페이스로 구현되지 않는다.

Customize 인터페이스는 컴포넌트 내부의 가변점에 어플리케이션에서 요구하는 가변치를 설정하기 위한 인터페이스이다. *Customize* 인터페이스는 컴포넌트를 수정하지 않고 컴포넌트를 특화시킬 수 있기 때문에 확장성을 제공한다. 컴포넌트는 이러한 확장성을 제공하기 위한 커스터마이제이션 기법을 컴포넌트 내부에 구현하여야 한다. *Customize* 인터페이스를 통해 컴포넌트 사용자는 컴포넌트를 자신의 목적에 맞게 설정 한 후에 사용자는 컴포넌트의 기능은 *Provided* 인터페이스를 통해 컴포넌트 사용자에게 제공된다.

2.4 컴포넌트 커스터마이제이션 기법

컴포넌트의 가변성을 지원하기 위한 커스터마이제이션 기법은 그림 2와 같이 선택형(Selection), 플러그인(Plug-in) 기법, 외부 프로파일(External Profile) 기법이 있다[7]. 선택형 기법은 이미 식별된 가변치를 컴포넌트에 포함시킨 후에 내부의 가변치를 선택하는 기법이다. 플러그인 기법은 모든 가변치를 컴포넌트에 포함

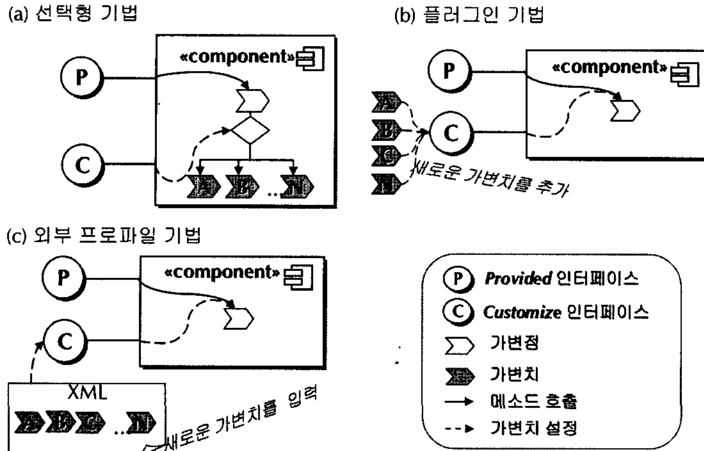

그림 2 컴포넌트 특화 기법

시킬 수 없으므로 외부의 가변치를 컴포넌트에 설정하는 기법이다. 외부 프로파일 기법은 XML 파일과 같은 외부 파일에 작성하고 프로그램에서 이를 읽어와 가변성을 설정하는 기법이다.

3. EJB에서의 복합 컴포넌트 설계

본 장에서는 CBD의 컴포넌트와 EJB 컴포넌트의 차이점을 해결하기 위한 EJB에서의 복합 컴포넌트 설계 모델을 제시한다. 복합 컴포넌트 설계를 이용하여 EJB에서 다수의 인터페이스를 지원할 수 있고 좀 더 큰 기능성을 가지는 컴포넌트를 설계할 수 있다.

EJB에서는 하나의 엔터프라이즈 빈이 컴포넌트의 단위로 사용되기 때문에 CBD의 큰 단위의 재사용 컴포넌트를 지원하지 못한다. CBD 개념에서 설명하는 복합 객체를 포함하고 있는 컴포넌트 크기(Granularity)와 차이점이 있다. 따라서 엔터프라이즈 빈을 이용하여 복합 컴포넌트를 만드는 설계 기법이 필요하다.

컴포넌트는 일반적으로 블랙박스 형식으로 제공된다. 하지만 엔터프라이즈 빈은 외부에 기능을 제공하기 위한 인터페이스를 가지고 있는 구조이다. 그러므로 엔터프라이즈 빈 여러 개를 이용하여 블랙박스 형식의 복합 컴포넌트를 만들더라도 컴포넌트 외부에 해당 인터페이스가 모두 공개되는 문제가 발생된다.

따라서 컴포넌트의 개략 설계에서는 블랙박스 형식의 복합 컴포넌트로 설계하지만, EJB 기반의 상세 설계에서는 컴포넌트 설계의 내부 객체를 엔터프라이즈 빈으로 각각 설계되고 구현된다. 따라서 실제로는 그림 3(a)와 같이 화이트박스 형식의 단일 컴포넌트로 설계되는 문제점이 발생된다. 이러한 문제는 그림 3(b)와 같은 설계로 해결 한다.

복합 컴포넌트의 내부에 숨겨져야 하는 컴포넌트는 여러 엔터프라이즈 빈들을 관장하는 퍼사드(Façade) 빈

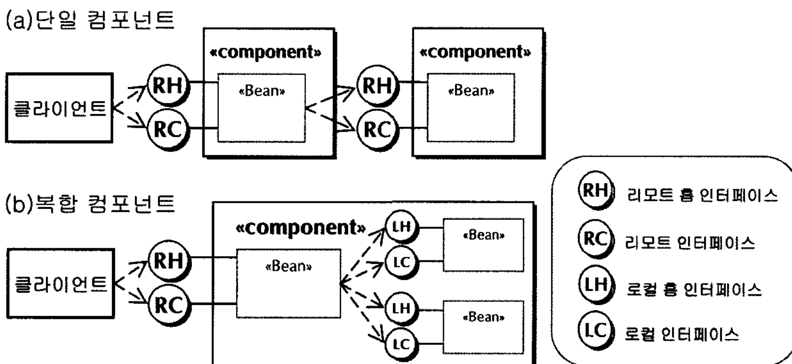

그림 3 EJB를 이용한 단일 및 복합 컴포넌트

[12]을 정의하여 설계할 수 있다. 디플로이먼트 디스크립터의 <ejb-client-jar>를 명시하고 클라이언트에게는 중재자 빈의 인터페이스만을 제공한다. 그러면 컴포넌트 내부의 인터페이스에는 클라이언트가 직접 접근할 수 없고 따라서 워크플로우를 직접 제어하지 못하며 반드시 외부의 리모트 인터페이스를 통해서만 컴포넌트에 접근하게 된다[13].

4. EJB 컴포넌트의 커스터마이제이션 기법

본 절에서는 EJB 플랫폼을 이용한 컴포넌트의 가변성 설계 기법을 제안한다. EJB 플랫폼은 컴포넌트 특화를 위한 장치를 제공하지 않기 때문에 이를 지원하기 위한 기법이 필요하다. 제안하는 가변성 설계 기법을 이용하여 설계한 컴포넌트는 어플리케이션을 조립하는 사용자가 컴포넌트의 수정 없이 *Customize* 인터페이스만을 사용하여 컴포넌트를 특화 한다. 실제 어플리케이션 이용자에게 *Customize* 인터페이스는 공개하지 않고, *Provided* 인터페이스만을 제공하기 때문에 의도하지 않은 *Customize* 인터페이스의 호출을 막을 수 있다.

4.1 선택형 기법

선택형 기법은 컴포넌트 내부의 가변치를 이용해서 컴포넌트를 클라이언트의 요구에 맞도록 특화한다. 컴포넌트 내부의 가변치를 사용하는 기법은 어플리케이션에 따라 변할 수 있는 가변치를 미리 식별하여 컴포넌트 내부에 설계함으로써 컴포넌트의 사용자가 특화시에 빈 내부의 가변치 중에서 선택하여 사용할 수 있다.

4.1.1 제1단계 - 가변점이 있는 함수 선언

1단계에서는 컴포넌트 내에 가변점이 있는 함수를 선언한다. 가변점이 있는 함수에서는 가변치 선택값을 가지고 있는 객체에 접근하여 가변치 선택 정보를 얻는다. 해당 가변치 선택값을 이용하여 선택값에 알맞은 알고리즘을 수행한다. 이는 선택값에 따른 로직의 변화가 가능하도록 프로그램의 조건문을 사용하여 설계할 수 있다. 가변점에서 가변치 선택값을 구하는 방법은 그림 4와 같다. 가변점이 있는 foo1() 함수에서 정적 클래스인 Variation에 저장된 가변치 선택값을 loanVar변수를 통해 얻어와 switch 문을 이용하여 해당 가변치 선택값에 알맞은 알고리즘을 수행한다.

4.1.2 제2단계 - 가변치 설정 함수와 가변치 저장소 선언

2단계에서는 가변치를 설정하기 위한 함수와 가변치에 대한 정보를 유지하는 저장소를 선언한다. 첫째 외부에서 설정한 선택값 정보를 이용하여 여러 가변치 중 하나를 선택하는 가변치 설정 함수를 선언한다. 그림 4와 같이 VariabilityBean의 select1() 함수는 엔터프라이즈 빈의 비즈니스 로직을 수행하기 위한 함수가 아니다. 이는 사용자의 가변치 선택값을 설정하는 역할을 담당한다. 가변치를 설정하는 함수는 비즈니스 로직을 설정하는 함수와 별도의 엔터프라이즈 빈으로 설계하는데 이는 비즈지스 로직의 수행과 컴포넌트 특화의 책임을 분리하여 응집도를 높이기 위함이다.

둘째 가변치 저장소를 선언한다. 가변치 설정 함수에서 설정한 가변치 선택값을 영구적으로 보관하기 위해 사용된다. 가변치 선택값이 변경되지 않는 동안에는 해당 가변치 선택값을 지속적으로 유지해야 한다. 가변치 저장소는 가변치값을 효율적으로 제공하기 위해 static 변수를 사용한다. 엔터프라이즈 빈은 풀링(Pooling) 되기 때문에 static 변수를 사용하여 값을 저장할 수 없다. 따라서 일반 자바 객체(Plain Old Java Object, POJO)를 사용하여 설계한다.

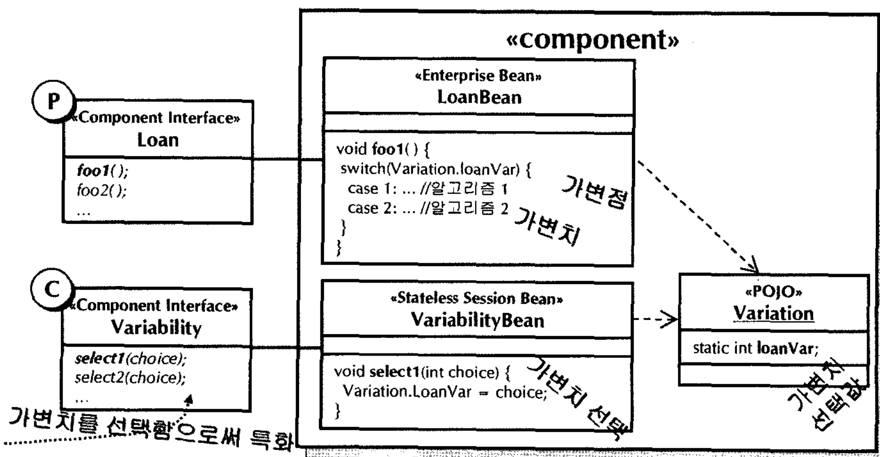

그림 4 선택형 가변성 설계 기법

4.1.3 제3단계 - Customize 인터페이스 정의

3단계는 *Customize* 인터페이스를 선언하는 작업이다. 컴포넌트 사용자에게 가변치 설정 함수를 제공하기 위해 선언한다. 컴포넌트의 특화를 위한 *Customize* 인터페이스는 컴포넌트 내부에 가변치 선택을 위한 세션빈의 컴포넌트 인터페이스를 이용해 설계한다. 선택형 기법에서는 가변치 선택값을 특화 시에 전달한다. 그림 4의 Variability는 *Customize* 인터페이스이다. 가변치 설정 함수를 별도의 엔터프라이즈 빈에 설계하였기 때문에 *Customize* 인터페이스는 비즈니스 로직을 가진 인터페이스로부터 독립될 수 있다. 이를 통해, 컴포넌트를 특화할 때만 *Customize* 인터페이스를 공개하고 컴포넌트의 사용시점에는 해당 정보를 공개하지 않을 수 있다.

4.1.4 제4단계 - 가변치 설정

가변치 설정을 위해서 EJB의 컴포넌트 인터페이스를 이용하여 설계된 *Customize* 인터페이스를 통해서 컴포넌트 내부의 가변치를 선택한다. 그림 4와 같이 *Customize* 인터페이스에서 정의된 select()함수에서 가변치를 선택하게 되면 일반 자바 객체인 Variation 클래스의 static 필드인 loanVar 변수에 가변치의 선택 정보를 저장한다. 컴포넌트의 실행 시 loanVar 변수의 가변치 정보를 참조하여 선택된 로직을 수행함으로써 특화된 정보를 이용한다.

4.2 플러그인 기법

플러그인 기법은 컴포넌트 내부에 있는 가변점에 외부에서 가변치를 할당하여 컴포넌트를 특화한다. 컴포넌트 외부의 가변치를 설정하는 기법은 변화 지점인 가변점을 미리 식별하였지만 한정된 가변치를 가지지 않거나 가변치를 미리 식별하는 것이 불가능 한 경우 이를 컴포넌트를 사용하는 클라이언트가 어플리케이션의 목적에 맞는 가변치를 설정하여 컴포넌트를 특화할 수 있다.

4.2.1 제1단계 - 가변점이 있는 함수 선언

1단계에서는 컴포넌트 내에 가변점이 있는 함수를 선언한다. 가변점이 있는 함수는 가변치를 저장하고 있는 객체에 접근하여 플러그인 된 정보를 얻는다. 가변점이 있는 함수에서는 플러그인된 가변치를 저장하고 있는 객체에 접근하여 가변치의 정보를 얻는다. 얻은 정보의 플러그인된 알고리즘을 이용하여 컴포넌트가 특화된다. 가변점에서 플러그인된 가변치를 구하는 방법은 그림 5와 같다. 가변점이 있는 foo1() 함수에서 정적 클래스 Variation의 var1에 저장되어있는 플러그인된 객체를 얻어와 해당 객체의 알고리즘을 수행한다. 이는 객체의 동적 바인딩 속성을 이용한 기법이다.

4.2.2 제2단계 - 가변치 설정 함수와 가변치 저장소 선언

2단계에서는 플러그인된 가변치를 설정하는 함수와 해당 가변치에 대한 정보를 유지하는 저장소를 선언한다. 첫째 외부에서 플러그인된 가변치를 컴포넌트 내부에 설정하는 가변치 설정 함수를 선언한다. 가변치 설정 함수에서는 외부에서 플러그인된 객체를 가변치 저장소에 저장하는 역할을 담당한다. 선택형 기법과 같이 그림 5에서 가변치 설정 함수는 VariabilityBean에 정의된다. 가변치 설정 함수 plugin1()은 플러그인된 가변치를 가변치 저장소에 설정한다.

둘째 가변치 저장소를 선언한다. 가변치 설정 함수에서 설정한 가변치를 영구적으로 보관하기 위해 사용된다. 선택형 기법과 같이 새로운 가변치를 플러그인해서 컴포넌트를 다시 특화하기 이전까지 해당 가변치 정보를 저장하고 있는다. 가변치 저장소는 static 필드로 플러그인 될 객체의 슈퍼타입에 해당하는 변수를 가지고 있는다. 이는 플러그인 되는 정보가 무엇이던지 해당 타입의 서브클래스이면 업캐스팅(up-casting)해서 정보를

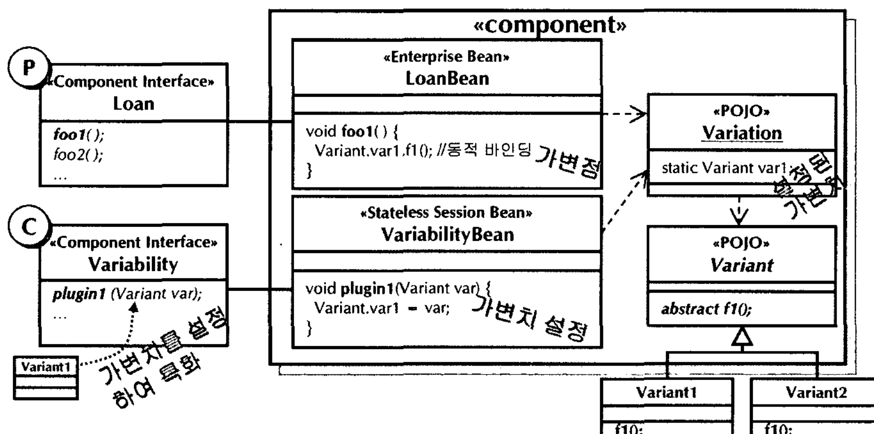

그림 5 플러그인을 이용한 가변성 설계 기법

사용할 수 있게 해준다.

4.2.3 제3단계 - Customize 인터페이스 정의

3단계는 선택형 기법과 같이 *Customize* 인터페이스를 정의하는 작업이다. 컴포넌트 사용자에게 가변치 설정 함수를 제공하기 위해 선언한다. *Customize* 인터페이스의 정의 방법은 선택형 기법에서와 같이 가변치 설정 함수를 포함하고 있는 엔터프라이즈 빈의 컴포넌트 인터페이스를 통해 정의한다. 플러그인 기법에서는 플러그인 될 가변치를 해당하는 객체의 슈퍼타입으로 업캐스팅해서 전달한다. 그림 5의 Variablity는 *Customize* 인터페이스이다. 플러그인 될 객체의 슈퍼타입인 Variant를 인자로 전달한다.

4.2.4 제4단계 - 가변치 설정

4단계는 컴포넌트 사용자가 플러그인 될 객체를 생성해서 *Customize* 인터페이스를 이용하여 가변치를 설정하는 단계이다. 가변치를 설정하기 위해서는 먼저 플러그인 될 객체를 설계하여야 한다. 그림 6과 같이 컴포넌트가 포함하고 있는 플러그인의 슈퍼클래스를 상속받아 플러그인 될 객체를 설계한다.

```
public class Variant1 extends Variant {
   public void f1() {
      // 어플리케이션 요구사항에 맞는 비즈니스 로직 수행
      ...
   }
}
```

그림 6 상속을 통한 플러그인 될 객체의 설계

플러그인 될 객체를 설계한 후 배포된 컴포넌트를 어플리케이션의 요구사항에 맞게 특화하기 위해서 플러그인 될 객체를 생성하고 컴포넌트의 *Customize* 인터페이스를 통해 이를 전달한다. 플러그인 될 객체를 생성하고 *Provided* 인터페이스를 통해 특화하는 방법은 그림 7과 같다.

```
public class VariantSetter {
   Variability variability;
   ...
   // 가변치 설정 메소드
   public void setVariant() {
      Variant var = new Variant1();
      variability.plugin(var);
   }
}
```

그림 7 *Customize* 인터페이스를 통한 컴포넌트 특화

4.3 외부 프로파일 기법

외부 프로파일 기법은 컴포넌트 내부에 있는 가변점을 가변치를 명세한 외부 프로파일을 이용해서 컴포넌트를 특화한다. 외부 프로파일은 XML 형식의 파일 사용이 가능하다. 외부 프로파일에 가변치를 명세한다. 명세된 외부 프로파일을 컴포넌트에서 분석하고, 해당 가변치를 컴포넌트 내부에서 반영구적으로 기억하여 컴포넌트를 특화시킨다. 외부 프로파일에 작성된 컴포넌트의 가변치를 변경하면 동적으로 컴포넌트를 특화 시킬 수 있다.

4.3.1 제1단계 - 가변점이 있는 함수 선언

1단계에서는 컴포넌트내에 가변점이 있는 함수를 선언한다. 컴포넌트에서 가변치를 읽어 들인 후에는 플러그인 기법과 유사하다. 가변점이 있는 함수에서는 이미 가변치를 저장하고 있는 객체에 접근하여 가변치의 정보를 얻는다. 얻은 정보를 이용하여 컴포넌트가 특화된다. 가변점에서 가변치값을 구하는 방법은 그림 8과 같다. 가변점이 있는 foo1() 함수에서 정적 클래스 Variation의 var1 변수에 저장된 정보를 이용하여 foo1()의 로직을 수행한다.

4.3.2 제2단계 - 가변치 설정 함수와 가변치 저장소 선언

2단계에서는 가변치를 외부 프로파일에서 분석하는 부분과 분석된 가변치를 저장하는 저장소를 선언한다. 첫째 가변치를 외부에서 읽어들이는 가변치 설정 함수를 선언한다. 가변치 설정함수에서는 외부 프로파일에 설정된 가변치를 XML 파서등을 사용하여 분석한다. 그림 8과 같이 setVar1() 함수는 엔터프라이즈 빈의 비즈니스 로직을 수행하는 함수가 아니다. 디플로이먼트 디스크립터(Deployment Descriptor)와 같은 XML 파일로부터 <env-entry> 태그를 이용한 환경 변수의 값을 가변치로 분석하고, 분석된 가변치를 설정하는 기능을 수행한다.

이때 가변치 설정 함수가 가변점 foo1()이 호출 될때마다 XML파일을 읽는 부분이호출되면 안 된다. 기본 EJB에서 사용하고 있는 디플로이먼트 디스크립터 기법에 단일화 패턴(Singleton Pattern)을 사용해야 한다. 해당 가변점 함수가 처음 호출되기 전에 한번만 가변치 설정 함수를 호출한다. 한번만 호출하는 방법으로는 그림 8과 같이 사용자가 *Customize* 인터페이스를 통해 한번만 setVar1()을 호출 할 수 도 있다. 또 다른 방법은 엔터프라이즈 빈의 ejbCreate()와 같은 콜백 메소드에서 아직 가변치가 설정이 안된 경우에만 자동으로 호출하는 방법이 있다. 다른 방법으로는 setVar1(file xml) 오퍼레이션에 가변치를 설정한 XML 파일을 전달하여 가변치를 분석할 수도 있다.

둘째 가변치 저장소를 선언한다. 가변치 설정 함수에서 분석된 가변치를 영구적으로 보관하기 위해 사용된다. 역시 다른 가변성 기법과 같이 가변치가 변경되기 전까지는 가변치 저장소는 한번만 설정되는 것이 효율

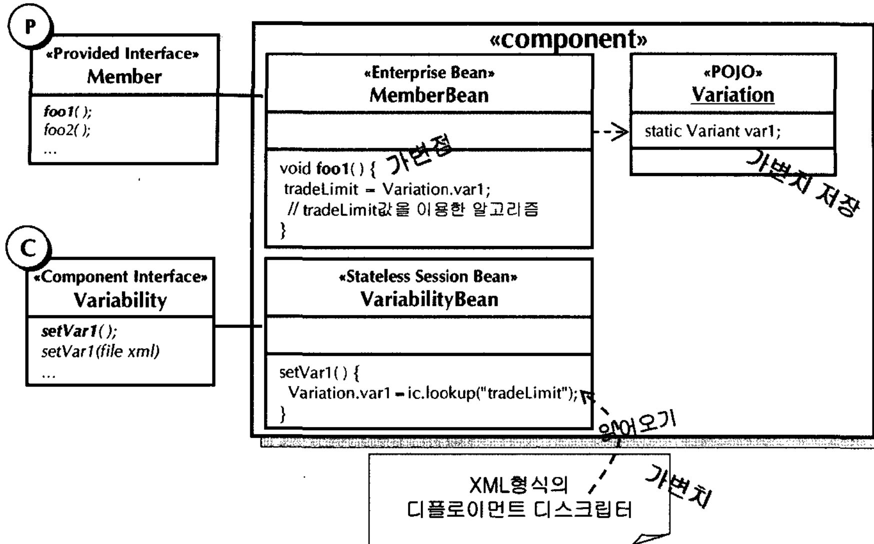

그림 8 외부 프로파일을 이용한 가변성 설계 기법

적이다. 가변치값을 효율적으로 제공하기 위해 static 변수를 사용한다.

4.3.3 제3단계 - Customize 인터페이스 정의

3단계는 선택형과 플러그인 기법과 같이 *Customize* 인터페이스를 선언하는 작업이다. 컴포넌트 사용자에 의해 가변치 설정함수가 호출되는 경우에는 존재하지만, ejbCreate()와 같은 콜백 함수를 통해 가변치 설정 함수가 호출된다면, 외부 프로파일 기법에서는 *Customize* 인터페이스가 필요 없을 수 있다.

4.3.4 제4단계 - 가변치 설정

4단계는 컴포넌트 사용자가 XML 파일에 가변치를 선언하는 작업이다. EJB에서는 환경 변수를 설정하기 위한 XML형식의 디플로이먼트 디스크립터를 제공한다. 그림 9와 같이 <env-entry> 엘리먼트를 이용하여 선언이 가능하다. <env-entiry-name>에 설정된 값을 이용하여 <env-entry-type>에 설정된 데이타 형식으로 <env-entry-value>에 선언된 값을 읽을 수 있다.

외부 프로파일 기법에서의 주요 특징은 XML 형식의 텍스트 파일에 가변치를 기술하는 것이다. 선언된 가변

```
<enterprise-beans>
  <session>
   <env-entry>
    <env-entry-name>tradeLimit</env-entry-name>
    <env-entry-type>java.lang.Integer</env-entry-type>
    <env-entry-value>500</env-entry-value>
   </env-entry>
  </session>
</enterprise-beans>
```

그림 9 디플로이먼트 디스크립터를 이용한 외부 프로파일 선언

치가 선택형 기법에서와 같이 이미 구현된 가변치를 선택하기 위한 가변치의 설정 값을 넣을 수 있고, 플러그인 기법과 같이 가변치를 직접 선언할 수 있다. Model Driven Architecture(MDA) 에서 사용하는 액션 의미언어(Action Semantic Language)와 같은 알고리즘을 기술하는 언어를 사용하여 가변적 로직을 선언 할 수도 있다.

또 다른 응용된 방법으로는 EJB에서 제공하는 디플로이먼트 디스크립터가 아니라 복잡한 가변치를 설정하기 위해 XML 파일을 사용자가 정의해서 사용할 수 있다. XML의 기법에는 Document Object Model(DOM)과 Simple API for XML(SAX) 기법이 있다. DOM은 XML 문서의 모든 노드를 트리 구조로 메모리에 로드한다. 따라서, DOM은 많은 메모리를 필요로 하고 많은 API를 갖는다. SAX는 DOM에 비해 간단한 구조를 갖는다. 개발자는 단지 4가지의 SAX API를 사용하면 된다. SAX는 XML 전체 노드를 메모리에 로드하지 않는다. 따라서 많은 메모리를 필요로 하지 않는다. 단지 노드의 순서에 따라 데이타를 얻는다.

본 논문에서 제공한 커스터마이제이션 기법에서는 가변치를 한번만 읽게 된다. 따라서 메모리에 노드를 읽어놓을 필요없이, 순서에 따라 한번만 호출하면 된다. 그러므로 DOM보다는 SAX를 사용하는 것이 더 효율적이다. 자바 기반의 SAX API를 이용한 외부화 기법의 구현은 그림 10과 같다.

클래스 A의 ReadProperty( ) 함수에서는 가변치가 설정되어 있는 XML 파일을 파싱한다. 그리고 선택된 가변치값에 따라서 해당 가변성 정보를 읽는다. 예를 들

```
public class A {
    private String attrType, logicType, wfType; //가변치 저장
    public static ReadProperty(int choice) {  //가변치 값을 읽는다.
        VariabilityHandler variability = new VariabilityHandler();
        XMLReader parser = new SAXParser();
        parser.setContentHandler(variability);
        parser.parse(getProfileName());          // XML 프로파일 파싱
        if(choice == 1) attrType = variability.getAttr1Type();
        if(choice == 2) attrType = variability.getAttr2Type();
        logicType = variability.getLogicType();
        ...
    }
    void f1( ) { ... }
}
public class VariabiltyHandler extends DefaultHandler { //SAX를 이용하여 가변치 값을 읽는다.
    private String attr1Type, attr2Type, logicType, wfType;
    public static final int ATTR1_TYPE = 1;
    public static final int ATTR2_TYPE = 2;
    public String getAttr1Type(){return attr1Type};
    ...
    public void startElement(String uri, String localName, String rawName,
                             Attributes attributes) throws SAXException{
        if(localName.equals("AttrType")) {
            if(attributes.getValue("variantType").equals("var1"))
                state = ATTR1_TYPE;
            if(attributes.getValue("variantType").equals("var2"))
                state = ATTR2_TYPE;
        }
        if(localName.equals("LogicType"))
            state = LOGIC_TYPE;
        ...
    }
    public void characters(char ch[], int start, int length) {
        if(state == ATTR1_TYPE) attr1Type = new String(ch, start,length).trim();
        if(state == ATTR2_TYPE) attr2Type = new String(ch, start,length).trim();
        if(state == LOGIC_TYPE) logicType = new String(ch, start,length).trim();
        ...
    }
}
```

그림 10 SAX를 이용한 외부화 파일을 분석하는 구현 예

면 사용자가 1번을 선택하면 XML 파일의 1번 엘리먼트를 읽는다. SAX를 사용하여 가변성 정보를 읽는 소스는 VariabilityHanlder() 함수이다. 본 논문의 예제에서는 어트리뷰트 가변성, 로직 가변성, 워크플로우 가변성을 식별하고 SAX를 이용하여 해당 엘리먼트 값을 읽는다. 분석된 가변성 정보로 f1() 함수는 사용자에 의해서 특화된다.

5. 평 가

본 논문에서는 EJB 컴포넌트를 위한 가변성 설계와 이를 특화 시키기 위한 기법을 제안 하였다. 컴포넌트의 특화 기법인 선택형 기법, 플러그인 기법, 외부 프로파일 기법을 이용하여 EJB를 위한 특화 기법을 제안했다. 표 1과 같이 기존의 가변성 연구에서 제안한 가변성 항목을 본 논문에서는 지원한다.

Anastasopoulos의 연구에서는 가변성과 휘처(Feature)를 제안하였다. 제안된 기법 중에 집합/위임, 동적 링크 라이브러리, 파라메터라이제이션 기법만이 블랙박스 컴포넌트에서 적용 가능하다. Svahnberg는 다섯 가지의 가변성 타입을 제안 하였으나 상속과 컴포넌트 생성 기법은 블랙박스 컴포넌트에서 사용할 수 없으며 구체적인 구현 기법도 제공하지 않는다. 그러나 본 논문에서는 블랙박스 수준에서 EJB의 복합 컴포넌트 설계로부터 세 가지 특화기법을 제안하였으며 세부적인 구현 기법도 제시하였다.

Catalysis 방법론은 플러그인 기법을 이용하여 컴포넌트를 특화 시키는 기법을 제안하였다. 그러나 Keepence의 세 가지 가변성 설계 패턴은 가변치의 범위를 제안

표 1 본 논문에서 지원하는 가변성 관련 항목 (✔ : 제공)

지원 항목 \ 가변성 연구	Anastasopoulos[4]	Svahnberg[5]	Catalysis[17]	Keepence[18]	DAO/EJB[19]	본 논문의 기법
컴포넌트를 위한 가변성 지원	✔	✔	✔	✔	✔	✔
블랙박스 컴포넌트 지원	✔	✔	✔	✔	✔	✔
가변성의 구현 수준 제안	✔		✔	✔	✔	✔
가변성 타입 지원	✔	✔				✔
가변성 영역 지원	✔	✔		✔		✔
선택형 기법 지원	✔	✔		✔		✔
플러그인 기법 지원	✔					✔
외부 프로파일 기법 지원		✔				✔
특정 컴포넌트 모델 지원					✔	✔

하고 있지만 세부적인 구현 기법은 명세하고 있지 않다. Keepence의 세 가지 가변성 설계 패턴은 가변치의 범위를 제안하고 있지만 세부적인 구현 기법은 명세하고 있지 않다. 이러한 기존의 연구는 EJB를 위한 가변성 설계 및 구현 기법은 제공하지 않지만, 본 논문에서는 EJB 컴포넌트를 위한 가변성 설계 기법을 제안했다.

EJB를 위한 커스터마이제이션 기법을 제안한[19] 논문은 데이타베이스 변화에 따른 영속성 관리 메커니즘의 변화를 J2EE 패턴[12]인 DAO(Data Access Object) 패턴을 이용해 컴포넌트가 데이타베이스에 따른 쿼리를 정의한 DAO를 선택 운용할 수 있다. 이러한 기존의 연구는 EJB를 위한 커스터마이제이션 기법을 제공하고 있지만 영속성 관리를 담당하는 엔티티 빈 중에서도 사용자가 직접 영속성을 관리해야 하는 BMP에 한정되어 있다. 이러한 DAO 기법은 본 논문에서 제시한 기법을 BMP에 적용할 시 *Customize* 인터페이스를 통해 DAO를 선택하거나 플러그인 함으로써 적용할 수 있다. 본 논문에서는 비즈니스 로직의 처리를 위한 세션 빈과 영속성 관리를 위한 엔티티 빈에서 모두 사용 가능한 커스터마이제이션 기법을 제공한다.

본 연구에서는 소스 코드를 직접적으로 접근하지 않는 블랙박스 형태의 EJB 컴포넌트를 특화 할 수 있는 기법을 제안 하였다. EJB 컴포넌트를 위한 커스터마이제이션 기법 중 선택형 기법은 식별 가능한 가변치, 즉 Close영역일 경우 미리 컴포넌트 내부에 설계함으로써 컴포넌트의 사용자가 특화 시에 빈 내부의 가변치 중에서 선택하여 사용할 수 있다. 만약 가변점을 미리 식별하였지만 한정된 가변치를 가지지 않거나 가변치를 미리 식별하는 것이 불가능 한 경우, 즉 Open영역일 때 본 논문에서 제안한 플러그인 기법을 이용하여 컴포넌트를 특화 할 수 있다. 외부 프로파일 기법은 명세된 외부 프로파일을 컴포넌트에서 분석하고, 해당 가변치를 컴포넌트 내부에서 반영구적으로 기억하여 컴포넌트를 특화시킨다. 외부 프로파일에 작성된 컴포넌트의 가변치를 변경하면 동적으로 컴포넌트를 특화 시킬 수 있다. 따라서 인터페이스 가변성을 제외한 4가지 가변성 타입을 지원한다. 그리고 본 논문의 특화 기법을 EJB 컴포넌트를 위한 코드 수준으로 제안함으로써 EJB를 이용하여 실용적인 컴포넌트를 개발할 수 있다.

6. 결 론

산업계와 학계 모두 CBD의 중요성이 인식되고 있다. CBD 기술은 효율적인 재사용 기술이며 대형 시스템 개발 시 사용되고 있다. EJB는 엔터프라이즈 환경에서 분산 컴포넌트 어플리케이션을 개발하기 위한 자바 기반의 서버 측 컴포넌트 아키텍처이다. EJB는 분산 서비스, 트랜잭션, 영속성, 보안을 미들웨어 차원에서 제공하여 개발자의 노력을 줄일 수 있기 때문에 CBD를 위한 최적의 플랫폼으로 인식되어 왔다. 컴포넌트는 다양한 사용자에 의해 사용되기 때문에 컴포넌트를 특정 환경에 맞춰 재사용성을 높이기 위한 특화 기법이 필요하다.

본 논문에서는 EJB 컴포넌트를 특정 환경에 맞도록 특화하기 위한 세 가지 기법을 제안한다. 미리 정의된 가변치를 컴포넌트 내부에 설계하여 특화 시 선택하여 사용하는 선택형 기법, 가변치의 식별이 완벽히 불가능한 경우나 선택 가능한 가변치가 너무나 많은 경우에는 컴포넌트 외부에 설계한 가변치를 플러그인하여 특화시키는 플러그인 기법, 프로그램이 아닌 XML 파일과 같은 외부의 가변치 정의 파일을 사용하여 가변치를 설정하는 외부 프로파일 기법을 제공하였다.

본 논문의 기법을 이용하여 사용자 특화가 가능한 효율적인 EJB 컴포넌트를 설계 및 구현 할 수 있도록 한다. 또한 기존의 CBD 연구에서의 특화 기법과 제시한 기법을 비교 평가하여 제시한 기법의 적합성을 검증했다. 제시한 EJB 컴포넌트 가변성을 위한 설계 및 구현 기법을 이용하여 EJB 환경에서 EJB 컴포넌트를 재사용성, 활용성, 이식성을 더욱 증가 시킬 것으로 기대한다.

참 고 문 헌

[1] Kim, S., "Lesson Learned from a Nationwide CBD Promotion Project," *Communications of the ACM*, Vol. 45, No. 10, pp. 83-87, 2002.

[2] DeMichiel, L., *Sun Microsystems, Enterprise JavaBeans™ Specification, Version 2.1*, Sun Microsystems, pp. 1-635, 2002.

[3] Roman, E., *Mastering Enterprise JavaBeans Third Edition Wiley*, 2005.

[4] Anastasopoulos, M. and Gacek, c., "Implementing Product Line Variabilities," *Proceedings of the 2001 symposium on Software reusability, Toronto, Ontario, Canada*, pp. 109-117, 2001.

[5] Svanhnberg, M., and Bosch, J., "Issues Concerning Variability in Software Product Lines," Lecture Notes in Computer Science 1951, Proceedings of the Third International Workshop on Software Architectures for Product Families, 2000.

[6] Muthig, D. and Atkinson, C., "Model-Driven Product Line Architectures," *SPLC2 2002, LNCS Vol. 2379*, pp. 110-129, 2002.

[7] Kim, S., Min, H., and Rhew, S., "Variability Design and Customization Mechanisms for COTS Components," *Lecture Notes in Computer Science Vol. 3480*, pp. 57-66, May, 2005.

[8] Sinnema, M., "COVAMOF: A Framework for Modeling Variability in Software Product Families," *LNCS 3154*, pp. 197-312, 2004.

[9] Kim, S., Her, J., and Chang, S., "A Theoretical Foundation of Variability in Component-based Development," Information and Software Technology, Vol. 47, pp. 663-673, July, 2005.

[10] Heineman, G. and Councill, W., *Component-Based Software Engineering*, Addison Wesley, 2001.

[11] Manolescu, D.A., and Johnoson, R.E., "A Micro Workflow Framework for Compositional Object-Oriented Software Development," Workshop on the Implementation and Application of Object-Oriented Workflow Management Systems II, OOPSLA, 1999.

[12] Alur, D., Crupi, J., and Malks D., *Core J2EE Patterns* 2nd, Prentice Hall, 2003.

[13] 김수동, 민현기, 이진열, 김성안, "Enterprise JavaBeans (EJB)에서 효율적인 CBD 컴포넌트 설계 기법," 정보과학회논문지: 소프트웨어 및 응용 제 33권 제 1호, pp. 32-43, 2006.

[14] Atkinson, C., et al., *Component-based Product Line Engineering with UML*, Addison-Wesley, 2001.

[15] Kim, S., Her, J., and Chang, H., "A theoretical foundation of variability in component-based development," *Information and Software Technology 47*, pp. 663-673, 2005.

[16] Chessman, J., and Daniels, J., *UML Component*, Addison-Wesley, 2001.

[17] D'Souza, D. and Wills, A. C., *Objects, Components, and Frameworks with UML*, Addison Wesley Longman, Inc. 1999.

[18] Keepence, B., and Mannion, M., "Using patterns to model variability in product families," IEEE Software, Vol. 16, Issue. 4, July-Aug., 1999.

[19] 이용원, 장윤정, 이경환, "EJB 기반 엔티티 컴포넌트 커스터마이제이션", 한국정보과학회 2001년 춘계학술대회, VOL. 28, pp. 667-669, 2001.

민 현 기
정보과학회논문지 : 소프트웨어 및 응용 제 33 권 제 1 호 참조

김 성 안
정보과학회논문지 : 소프트웨어 및 응용 제 33 권 제 1 호 참조

이 진 열
정보과학회논문지 : 소프트웨어 및 응용 제 33 권 제 1 호 참조

김 수 동
정보과학회논문지 : 소프트웨어 및 응용 제 33 권 제 1 호 참조