CBD 기반 컴포넌트의 재사용성의 향상을 위한 Opportunity Tree 설계

신솔민°, 이은서, 이경환
중앙대학교 컴퓨터공학부
{smshin°, eslee, kwlee}@object.cau.ac.kr

Opportunity Tree of Component Categorization for Reusability on the CBD Process

Sol-min Shinn°, Eun-Ser Lee, Kyung-Whan Lee
Dept. of Computer Science and Engineering, Chung-Ang University

요 약

세계적인 SW 산업의 변화 추이로 볼 때, 우리나라 SW업체들도 SW개발 체제를 하루바삐 CBD로 전환해야겠다는 데 대해서는 의심의 여지가 없다. 기업의 CBD 능력은 지속적인 사업구조 조정 및 신규사업 창출이 불가피한 21세기의 경영환경에서 필수적으로 확보해야 할 기업 핵심 역량으로 간주되고 있다. 선진국에서는 CBD가 90년대 중반부터 본격화되어 이제는 기업 정보시스템 구축 방식의 주류가 되어있다. 국내에서도 금융, 국방 부분을 선도로 CBD에 의한 정보시스템 구축을 요구하기 시작했으며, 향후 그 수요가 급증할 전망이다. 그러나 문제는 CBD로의 전환에는 여러 가지 역경이 도사리고 있다는데 있다. 특히, 본 논문에서는 재사용성에 의해 기대되는 효과에 비해 실질적인 재사용의 결과도출을 이루지 못하는 문제점 개선을 위한 Opportunity Tree를 제안해본다. 또한, CBD로 전환함으로써 거둘 수 있는 이득이 무엇인지, 전환을 위해 갖추어야 할 조건이 무엇인지를 확인하고자 한다.

1. 서 론

소프트웨어 개발에 있어서 가장 큰 관심은 소프트웨어 개발 생산성을 높이는 것이다. 소프트웨어 개발 생산성을 높이기 위해서는 소프트웨어를 이루고 있는 여러 가지 구성단위들을 재사용하는 방법이 가장 효과적이라 판단된다. 재사용단위로서 컴포넌트의 개념이 도입되면서, 소프트웨어 개발 생산성은 비약적으로 높아지게 되었다. 컴포넌트를 사용함으로써 기존에 검증된 기능을 손쉽게 적은 비용으로 재사용이 가능하게 된 것이다. 이러한 과정에 의해 만들어진 컴포넌트 중에서 사용자는 컴포넌트를 선택하여 사용하게 된다. 그러나 컴포넌트의 재사용시에 문제점이 존재하게 된다. 그 문제점은 다음과 같다.

첫째, 사용자의 요구사항을 완전하게 만족시키는 컴포넌트는 존재가능성이 거의 없다. 실제적으로 이러한 불일치성은 전체 프로젝트의 Risk Management에도 치명적인 요소가 될 수 있다. 따라서 사용자의 요구사항이 변화되는 것을 예측하여 컴포넌트를 설계 할 필요가 있다.
둘째, 컴포넌트가 사용되어지면서 환경의 변화에 의한 경우 기존의 컴포넌트를 그대로 사용하여서는 원하는 결과를 얻을 수 없다. 이와 같은 경우는 현재 비즈니스 컴포넌트의 경우 대다수로 발생하는 경우이다. 그러므로 컴포넌트의 환경변화를 예측할 수 있는 추상화가 제공되어야 한다.
셋째, 컴포넌트를 재사용할 경우 요구되어지는 기능을 만족시키지 못하는 경우, 추가적인 기능을 만족시킬 수 있어야 한다. 기능 추가를 위한 컴포넌트의 구조가 제공되어야 한다[5].
따라서 본 논문은 이런 문제점의 해결책으로 컴포넌트에 요구되어지는 기능을 위한 도메인 중심의 비즈니스 컴포넌트 설계를 하여 서비스를 만족시키는 Opportunity Tree를 설계한다. 또한, 컴포넌트의 내부요소의 특성을 고려하여 환경에 적합한 컴포넌트 공용성과 가변성을 추출하고, 인터페이스 중심으로 컴포넌트를 구성하여 요구되는 기능을 서비스 할 수 있는 Glue code, Tailoring, System volatility, Interface System, Plug & Play 등을 제안한다.

2. 기반 연구

2.1 컴포넌트와 재사용

소프트웨어 재사용에 대한 관심은 소프트웨어 위기가 발생하면서 중요한 연구의 대상으로 주목 받아왔다. 소프트웨어를 부품화 된 독립모듈로 개발하는 컴포넌트 기술은 소프트웨어의 재사용에 대한 문제점 극복에 도움을 주고, 소프트웨어 개발 시 많은 장점을 가져다주었다.

2.1.1 재사용성(Reusability)

컴포넌트가 갖는 가장 큰 장점중 하나는 재사용성이다. 한번 개발된 컴포넌트는 독립된 모듈로 여러 소프트웨어에서 재사용된다. 컴포넌트의 재사용은 많은 이익을 가져다준다. 첫째로 저렴한 개발비용과 개발기간을 단축시킬 수 있는 개발의 생산성이다. 둘째로 개발의 편리성이다, 시스템개발자는 재사용되는 컴포넌트의 내부구현 및 설계에 대한 자세한 모습에 대해 알 필요가 없다[2].

2.2.2 대체성(Replaceability)

컴포넌트 대체성은 기존 시스템에서 기능이 변경되거나 추가되었을 때, 전체 시스템을 변경하는 것이 아니라 변경된 기능에 해당하는 컴포넌트를 새로운 컴포넌트로 대체할 수 있는 것을 말한다. 컴포넌트의 대체성을 만족시키기 위해서는 지켜야 할 몇 가지가 있다.

첫째, 컴포넌트 인터페이스를 준수하여야 한다.

둘째, 계약에 만족해야 한다[2].

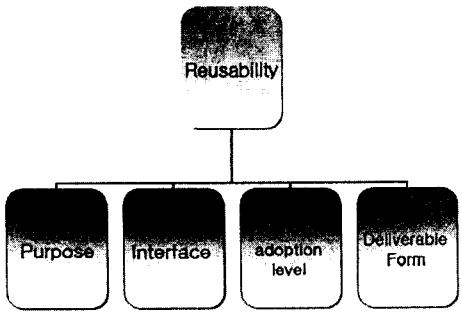
3. Opportunity Tree

3.1 Opportunity Tree 정의

Opportunity Tree는 해결하려는 목표를 설정하게 되고, 전체 목표를 해결하기 위하여 요구되는 하위 목표를 결정하게 된다. 즉, 하위 목표를 달성하게 되면 상위 목표가 달성될 수 있게 된다. 그리고 목표 설정과 함께 목표를 해결하기 위하여 요구되는 문서와 전문가의 노하우를 Opportunity Tree화하여 사용자에게 지침을 제공하고자 하는 것이 Opportunity Tree의 목적이 된다.

3.2 Opportunity Tree 설계 제안

재사용성 향상을 통하여 Business Goal의 달성을 위한 방법으로 Opportunity Tree를 설계한 그림은 아래와 같다. 각 항목들은 크게 목적과 범위(Purpose & Scope), 인터페이스(Interface), 사용내역 적용 레벨(Context adoption level), 배포형태(Deliverable Form) 등으로 구성되어 있다.



<Opportunity Tree>

3.3 목적과 범위(Purpose & Scope)

컴포넌트를 사용하는 목적을 나타낸다. 사용자는 자신이 어떤 이유로 그러한 컴포넌트를 필요로 하는지 이해하고 있어야 한다. 컴포넌트는 범위에 따라 문제를 기술한다는 측면과 구현을 지원한다는 측면으로 분류한다.

3.3.1 비즈니스 컴포넌트

비즈니스 컴포넌트에는, 일반적인 비즈니스 영역내의 개념들과 문제점을 표현한다. 특정 비즈니스 기능들을 수행하는 컴포넌트들과 비즈니스 영역에서 만들어진 개념들에 대한 컴포넌트들, 그리고 프레임워크와 같은 컴포넌트들 간의 협력관계를 구현한 컴포넌트들이 포함된다.

3.3.2 기술 컴포넌트

이 컴포넌트는 기술적인 목적을 띄며, 소프트웨어 엔지니어링 요구사항들에 초점을 맞추고 있으며, 특정한 실세계 문제에 종속되지 않는다는 특징을 갖는다. 기술적인 컴포넌트는 주로 개발자에 의해 사용되어진다.
기술적인 컴포넌트에는, 사용자 인터페이스 요소들을 구현한 컴포넌트들과 같은 Infrastructure내에 속하는 컴포넌트들이 사용하게 되는 기술적인 서비스를 제공하는 컴포넌트들이 이 범주에 속한다[3].

3.3.3 명세 컴포넌트

명세 컴포넌트는 이 비즈니스 컴포넌트의 하위항목으로 둘 수 있다. 명세 컴포넌트는 비즈니스 문제나 기술적인 문제의 단어와 구조를 정의한 기술서라 할 수 있다. 명세 컴포넌트는 기술서를 제공하는데, 이 기술서는 특정 비즈니스 문제나 혹은 기술적인 문제의 단어와 구조들의 집합을 정의한다[3].

3.3.4 구현 컴포넌트

구현 컴포넌트 기술 컴포넌트의 하위 항목으로 특정 구현기술이 사용되어져 완성된 컴포넌트의 집합이며, 여기에 소스코드나 실행 가능한 코드, 컨트롤, 어플리케이션 프레임워크, 어플리케이션 객체 등이 포함된다.

3.4 인터페이스(Interface)

컴포넌트 조립 기법은 컴포넌트 조립활동에서 사용되는 기법으로 컴포넌트 코드가 산출된다. 컴포넌트 조립은 컴포넌트를 합성하여 사용자가 원하는 소프트웨어를 만드는 과정이다.

3.4.1 Plug & Play

Plug & Play란 각각의 컴포넌트들이 갖고 있는 기능들을 손쉽게 우리가 마치 레고 장난감을 만들듯이 하나하나 끼워서 쓰고, 빼면서 기능들을 독립적으로 활용한다. 그렇게 하므로 각각의 컴포넌트 기능들은 결국 전체적으로 하나의 큰 어플리케이션의 기능으로 합쳐져서 구동을 이루는 개념이다.

3.4.2 Glue Code

CBD로 개발한 시스템이 효과적으로 운영되려면 소프트웨어 상의 공간을 채워주는 역할이 필요하다. 독립적인 CBD방법론에 따른 개발이여도 컴포넌트와 컴포넌트의 결합에 있어 그 조건과 환경이 정확히 맞아 떨어지는 경우는 거의 불가능하다. 여러 가지 많은 요구사항과 변경으로 결합을 시도할 때 있을 수 있는 공간을 Glue Code라는 코드를 이용하여 COTS 컴포넌트와 응용 컴포넌트 사이에서 인터페이스를 시켜주는 개념을 말한다.

3.4.3 Tailoring

컴포넌트Tailoring은 컴포넌트 인터페이스를 중심으로 조정된다. 컴포넌트를 이용하여 결합 혹은 사용을 할 때 최소한의 기본적으로 제공되는 기능으로는 자신이 원하는 기능과 부합되기 힘들 때, 사용자는 그 기능을 자신의 요구사항에 맞게 변경해야 한다. 이때 쓰는 컴포넌트 기법이 Tailoring Component 기법이다. 컴포넌트 Tailoring 작업은 컴포넌트를 조립하기 위해 요구되는 컴포넌트 조정 작업을 수행한다.

3.4.4 System Volatility

System의 외부에서나 내부에서의 요구사항 변경이 발생했을 경우나 Component volatility 때문에 발생하는 시스템의 변경작업이 필요할 때 적용되는 개념을 말한다.

3.4.5 공통성과 가변성

컴포넌트의 구조가 완성이 되면, 공통성과 가변성이 적용될 두 도메인을 선정한다. 도메인 선정 시에 현재 구축된 컴포넌트가 두 도메인에 공통적으로 사용 가능해야 한다는 것이 선행 조건이 된다. 이와 같이 두 도메인 간에 공통적으로 적용성이 있는지의 여부는 도메인 엔지니어링을 거쳐서 판별된다.
두 도메인에서 공통적인 부분과 가변적인 부분을 이미 구축된 컴포넌트에서 추출을 하여 컴포넌트 구조 상세화

의 인터페이스 구조를 다시 작성한다. 기존의 컴포넌트 구조상세화에서 가변적인 부분은 따로 추출하여 스테레오 타입으로 표기하여서 개발자들이 이를 참조하여 가변성 부분을 커스터마이징 할 수 있도록 한다[1].

3.5 사용내역 적용 레벨(Context adoption level)

컴포넌트의 특성은 일반적인 관점에서 논리적인 것과 물리적인 것으로 세분화한다. 논리적인 관점에서는 컴포넌트 특성은 비즈니스 도메인에서 실세계의 개념으로 모델링한 것을 의미한다. 비즈니스 컴포넌트의 물리적인 관점은 전형적인 개발자들이 가지고 있는 관점이다.

3.5.1 물리적인 관점

. 물리적인 관점에서 컴포넌트는 소프트웨어의 한 독립적인 조각이거나, 실세계의 비즈니스 개념으로 구현된 어플리케이션을 만드는 블록을 의미한다. 이러한 것을 부분화 한 것이 비즈니스 컴포넌트라고 한다[2].

3.5.2 논리적인 관점

논리적인 관점에서의 컴포넌트의 특성은 비즈니스 도메인에서 실세계의 개념으로 모델링 하는 것을 의미한다. 마치 모니터 컴포넌트가 정상적으로 동작하기 위해서는 커넥터가 외관상으로 본체에 정확하게 연결되는 것뿐만 아니라, 커넥터를 통하여 본체로부터 수신하는 신호를 받아들일 수 있어야 한다.

3.6 배포형태(Deliverable Form)

컴포넌트가 배포되어지는 형태에 따른 분류로서, 크게 설계 컴포넌트, 실행 가능 모듈, 소스코드 3가지로 분류된다.

3.6.1 설계 컴포넌트

패턴이란 특정 환경에서 여러 번 반복하여 발생하는 문제점들을 기술하고 각 문제점에 대한 핵심이 되는 해결책을 기술하여, 몇 번이고 이러한 해결책을 재사용 할 수 있도록 하는 것이다. 설계 패턴은 특정 문맥에서 일반적인 설계 문제점을 해결하기 위해 커스터마이징 되어질, 커뮤니케이션하는 객체와 클래식들에 대한 설명이라 할 수 있다. 시스템 구축 시 제안되었던 설계정보를 컴포넌트화 하여 이를 다른 어플리케이션의 개발에 재사용할 수 있다면, 이는 설계 컴포넌트의 분류에 들어갈 수 있다.

3.6.2 실행 가능 모듈

실행가능 모듈에는 정적라이브러리, 동적 링크 라이브러리, 혹은 실행 가능한 어플리케이션 조각들이 포함된다.\ 현재 거의 모든 벤더들은 이 실행가능 모듈 형태로 컴포넌트를 제공하고 있으며, 설계코드나 구현코드는 자산가치를 지니는 프로덕트로서 절대 외부유출을 하지 않고 보유하고 있는 경향이 많다.

3.6.3 소스코드

소스코드는 다른 프로젝트에서 재사용되어지거나 테스트되어지는 데에 사용된다. 재사용되어지는 영역으로는 같은 조직이나 같은 영역 내에서 존재하는 다른 어플리케이션들에 적용되어지는 경우가 많으며, 소스코드 자체를 분석하여 더욱 향상된 소스코드 컴포넌트를 만들어 낼 수 있다.
소프트웨어 컴포넌트들의 주요 목적중 하나는 소프트웨어 재사용이다. 소프트웨어 재사용의 주요 타입은 화이트 박스 재사용과 블랙박스 재사용이다.

4. 결론 및 향후 연구방향

생산 공정의 효율화를 꾀하기 위해 공학자들의 고민이 있어 왔듯이 소프트웨어 분야에서도 이러한 발전의 한 가운데에서 소프트웨어를 좀 더 효율적이고 효과적으로 개발하고 유지하려는 고민을 해온 사람들이 있었다. 오늘날 우리는 이런 대안으로 소프트웨어 컴포넌트를 생각하고 컴포넌트를 기반으로 하는 개발을 이른바 소프트웨어 위기에 대한 하나의 대안으로 가능하다는 확신을 가지고 있다. 비즈니스 환경의 변화에 따른 소프트웨어 요구사항의 변화를 고려하여 소프트웨어를 보다 유연하고 (Flexible) 재사용(Reusable)가능하게 구축하는 방법을 연구해보았다. 재사용을 달성하기 위하여 극복해야 할 문제점들에 대한 고찰과 그에 상응하는 해결책을 제시하고, Opportunity Tree의 설계를 통하여 재사용 성과를 최대한 높일 수 있는 요소들과 정량적 방법들을 연구해보았다. 이와 더불어 Opportunity Tree를 이용하여 실제 컴포넌트 프로젝트에 적용을 한 후, 결함 제거의 효율성을 따져보았고, 그 결과는 다음과 같다.

개발 단계	[illegible]	[illegible]
요구사항	156	135
설계	346	211
코딩	522	287

또한, 개발 단계별로 결함제거의 효율성에 대한 결과는

0.536 = 156 / (156+135) (요구사항 단계)

0.621 = 346 / (346+211) (설계 단계)

0.645 = 522 / (522+287) (코딩 단계)

이다.
결론적으로 프로젝트의 결과는 53% ~ 65% 정도의 값이 산출되었다. 이것은 요구사항 단계의 결함제거 효율이 많이 낮기 때문이다.

CBD의 최종목적이 재사용을 통한 경영차원의 이득에 있음은 누구도 부인할 수 없는 현실이다. 결국 이런 연구는 경영적으로도 최종목표인 Business Goal을 달성하는 과정으로 보다 적은 노력과 짧은 일정으로 더 많은 성과를 나타낼 수 있는 CBD 컴포넌트의 설계에도 크게 부합하여 컴포넌트 기반 연구에 많은 도움이 될 것이다.

5. 참고서적

1. 이은서 " 컴포넌트 재사용을 위한 공통성과 가변성 추출에 관한 연구" 중앙대학교 컴퓨터공학과, 2000
2. 박진구 "재사용을 위한 컴포넌트 분류체계와 UML을 이용한 컴포넌트 분류체계 표기법에 관한 연구" 2000
3. "실전 CBD Project" ,영진닷컴, 2004
4. "객체지향 CBD개발 Bible", 한빛미디어, 2003
5. "컴포넌트란 무엇인가? 알기쉬운 소프트웨어 컴포넌트", 한국소프트웨어컴포넌트컨소시엄, 2004
6. Padmal Vitharana "Design, Retrieval and Assembly in Component-based Software Development" 2003
7. Ali H.Dogru. Murat M.Tanik "A Process Model for Component-Oriented Software Engineering, March/April 2003