□특집□

e-Business를 위한 컴포넌트 소프트웨어 개발

배 두 환†

◈ 목 차 ◈

1. 서 론

21세기는 인터넷 시대라고 해도 과언이 아니다. 물론 과거에도 인터넷이 사용되기는 했지만 요즘에는 하루가 다르게 인터넷이 우리 생활 속으로 파고들어 인터넷 없는 우리의 일상 생활은 생각하기조차 힘든 실정이다. 이러한 인터넷은 기업의 생산성과 매출을 늘리고, 새로운 비즈니스의 창출을 통해 새로운 가치 창출에 커다란 공헌을 하고 있다. 또한, 인터넷은 기업의 구조를 바꾸는데도 일익을 하고 있다. 기존의 정보 기술은 기업의 조직 중심으로 운용되어 주로 내부 업무용으로 사용되었으며 기업 간의 정보 공유를 위해서 EDI가 사용됐다. 이러한 상황은 조직 중심에서 비즈니스 프로세스 중심으로 기업의 업무 형태가 변환됨에 따라서 고객에게 가치를 제공하는 e-Commerce와, 판매자, 공급자와 같은 관련 기업들이 정보 기술을 통해 하나로 엮어지는 e-Business로 변화할 것이다. 하지만, 인터넷이 내재하는 위험을 간과해서는 않된다. 인터넷은 매우 짧은 시간에 상대적으로 저렴한 비용으로 많은 사람들을 고객으로 끌어들일 수 있다는 것은 커다란 기회일 수 있지만, 충실한 고객들을 하루 아침에 다른 경쟁업체에 뺏길 수 있다는 위험도 인터넷으로 말미암아 가능한 것이다.

e-Business에서 기술이 더 중요한가, 아니면 관리와 경영이 더 중요한가 하는 논쟁을 자주 접하곤 한다. 과거 어느 분야에서도 그랬듯이 기술과 비즈니스가 조화를 이루어야만 성공할 수 있을 것이다. e-Business시대에서 생존을 위해서는 비즈니스에 대한 충분한 고려를 통해서 급속히 등장하는 신기술들과 새로운 표준들을 선택해야 한다. 다시 말하면 비즈니스와 기술이 적절히 어우러져야 한다는 것이다. 이것을 가능하게 하는 것이 바로 소프트웨어이다. 사람들은 신기술들과 e-Business에 대해서 매우 관심이 높지만, 정작 이러한 기술과 비즈니스를 연결해 주는 소프트웨어에 대해서는 관심이 부족한 것이 현실이다. 성공적인 e-Business를 위해서는 비즈니스 분석의 결과가 소프트웨어 설계에 반영되고 이러한 소프트웨어의 설계에 맞춰 성숙한 기술과 표준을 통하여 시스템으로 구현되어야 한다.

인터넷 시대의 정보화는 1) 기술의 급속한 발전, 2) 치열한 경쟁, 그리고 3) IT자원의 부족으로 특징 지워진다. 이러한 환경에서 살아 남기 위해서는 통합(Integration) 및 적응(Adaptation)을 잘해야 한다. 비즈니스 분석의 결과를 소프트웨어 설계에 반영함으로써 비즈니스와 이를 지원하는 소프트웨어를 적절히 통합하고 또 급속히 변하는 기술들을 적시에 반영할 수 있도록 적응력이 높

† 정 회 원 : 한국과학기술원 전자전산학과 부교수

은 소프트웨어 개발을 추구해야 한다.

성공적인 e-Business를 위한 소프트웨어의 요구사항을 만족시킬 수 있는 새로운 소프트웨어 개발 방법으로 컴포넌트 기반 개발(CBD: Component-Based Development)이 있다. 이 기법이 사용자 인터페이스 개발 분야에서 거둔 커다란 성공에 고무되어 서버 사이드의 비즈니스 로직을 구현하는데 적용하려는 노력과 비즈니스 환경의 변화에 발맞추어 Microsoft의 COM+와 Sun의 EJB를 수용한 어플리케이션 서버들이 e-Business를 구현하는 핵심 기술로 자리매김하고 있다.

e-Business를 위한 CBD에서의 소프트웨어 설계는 비즈니스 역량(Business Capability)을 지원하는 비즈니스 컴포넌트들의 네트웍으로 구성되며, 이를 통해서 비즈니스의 변화를 적시에 수용하게 된다. 이를 지원하는 핵심 개념이 인터페이스(Interface)이다. 인터페이스는 이를 사용하는 컴포넌트와 이를 구현하는 컴포넌트들 간의 계약(Contract)이다. CBD에서 컴포넌트들은 그들의 구현을 오직 인터페이스에 대해서만 의존하도록 하기 때문에 동일한 인터페이스를 구현하는 컴포넌트들을 서로 교체하는 것이 가능하다. 즉, 플러그-앤-플레이(Plug&Play)가 지원된다. 시스템을 구성하는 컴포넌트들과 그들이 구현하고 있는 인터페이스들, 그리고 그들 간의 의존 관계를 시스템의 아키텍처(architecture)라고 한다.

플러그-앤-플레이는 개발되는 시스템의 적응력을 향상시킨다. 그러나, CBD가 언제나 적응력을 향상시키는 것은 아니다. 적응력은 시스템에 발생하는 변화가 지역화(Localize)되어 있을 때 향상된다. 시스템에 대한 변화는 여러 컴포넌트에 영향을 미칠 수도 있으며, 이러한 영향은 의존 관계를 통해서 또 다른 컴포넌트들에게 영향을 미치게 된다. 최악의 경우, 시스템을 구성하는 모든 컴포넌트들이 변화에 영향을 받을 수도 있다. 이러한 문제를 완화하기 위해서는 변화에 영향을 받을 가능성이 있는 요소들은 지역화하는 것이 바람직하다. 이러한 구조를 강건 구조(Robust Structure)라고 한다[4]. 즉, 성공적인 CBD를 위해서는 구현하고자 하는 시스템에 대해서 강건한 아키텍처를 도출해야 한다.

본 논문에서는 강건한 아키텍처를 도출하기 위해서 고려되어야 할 사항들을 알아보고, 아키텍처의 강건성을 향상시키기 위한 전략을 제시하고자 한다. 본 논문의 구성은 다음과 같다. 2장에서 변화의 특성을 정리하고, 변화에 대처하기 위한 구조에 대해서 알아 본다. 3장에서는 2장에서 제시된 사항들을 포괄하는 간략한 CBD 방법론을 제시하며, 4장에서 끝을 맺도록 한다.

2. 아키텍처의 강건성

앞서 언급했듯이 시스템의 적응력을 높이기 위해서 시스템의 아키텍처는 강건해야 한다. 즉, 바꾸기가 쉬워야 한다. 이를 위해서는 변화의 가능성이 있는 요소들을 지역화 함으로써 되도록 시스템의 적은 부분에 영향이 미치도록 해야 한다. 가능한 변화는 크게 두 개로 분류될 수 있다[4].

1. 새로운 자료 형의 추가와 같은 데이터의 변화
2. 새로운 업무 절차의 추가와 같은 업무(Transaction)의 변화

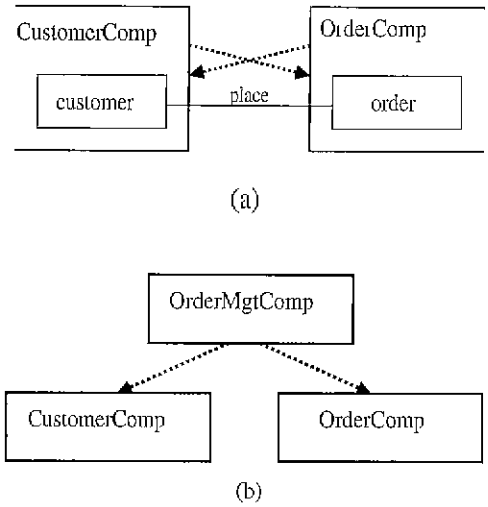
시스템의 변화는 이러한 두 가지 측면에 모두 영향을 미치는 것이 보통이다. 이러한 두 측면을 모두 고려하기 위해서 [4]에서는 제어(Control) 객체와 엔티티(Entity) 객체의 개념을 제시했다. 즉, 엔티티 객체가 데이터를 추상화 하고, 제어 객체가 업무에 대한 구현을 추상화함으로써 각 변화에 대한 영향을 지역화하는 것이다. 이 접근 방법에서 중요한 점은 엔티티 객체가 정확하고 재사용 가능한 연산들을 제공함으로써 업무에 대한 변화가 엔티티 객체로 전파되는 것을 최소화하는 것이다.

하지만, 아키텍처 상에서 인터페이스의 호출 관계-제어 구조(Control Structure)-를 고려하게 되면, 이 문제는 더욱 복잡해진다. 우선, 제어 구조에 대해서 알아보기에 앞서서 컴포넌트 기반 시스템의 아키텍처의 특성에 대해서 알아보자. 아키텍처에서 중요한 원칙중의 하나는 컴포넌트 간의 상호 의존관계(Mutual Dependency)를 제거하는 것이다. 상호 의존 관계에 있는 컴포넌트들은 서로 의존적이어서 독립적으로 사용될 수 없어 아키텍처의 재사용성과 적응력을 약화시킨다. 컴포넌트 간의 의존관계를 야기시키는 원인 또한 크게 두 가지로 구분할 수 있다.

1. 컴포넌트 간에 존재하는 클래스 간의 연관 관계(Association)
2. 컴포넌트 간에 호출 관계

제시된 원인들 중에서 컴포넌트 간의 호출 관계는 개발자들에 의해서 설계되는 것이므로, 개발 과정에서 발견되고 해소될 것이나, 컴포넌트 간 연관 관계는 업무 영역의 특성에서 야기되는 것이므로 다른 접근이 필요하다. 이 문제를 예를 통해서 살펴보도록 하자. (그림 1)의 (a)는 전형적인 컴포넌트 간 연관 관계를 나타내고 있다. 왼쪽의 컴포넌트 CustomerComp는 고객에 대한 정보를 관리하는 컴포넌트이며 오른쪽의 컴포넌트 OrderComp는 고객들의 주문에 대한 정보를 관리하고 있는 컴포넌트이다. 그림에서 보이듯이 컴포넌트 CustomerComp안에 있는 클래스 Customer와 컴포넌트 OrderComp에 속한 클래스 Order사이에는 연관관계 Place가 있다. 이 연관관계는 Customer와 Order가 서로에 대해서 일관되게 다루어져야함을 의미한다. 따라서, CustomerComp는 자신의 기능을 구현하는 과정에서 연관관계 Place의 일관성을 유지하기 위해서 OrderComp에 의존하며, OrderComp 또한 같은 이유로 OrderComp에 의존하게 된다.

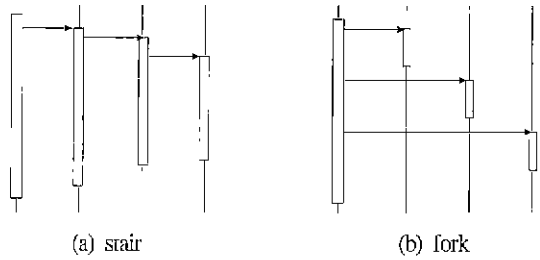
이러한 문제를 해소하는 일반적인 방법은 (그림 1)의 (b)와 같이 새로운 컴포넌트를 도입하고 이



(그림 1)

컴포넌트에 연관관계의 일관성을 유지하도록 하는 것이다. 결과적으로 상호 의존관계가 제거된 아키텍처는 계층적 구조를 갖게되며 컴포넌트와 그들 간의 의존관계는 DAG를 구성하게 된다. 앞으로, 컴포넌트 아키텍처는 이처럼 계층적 구조라고 가정한다.

따라서, 컴포넌트 간의 의존관계는 모두 제어 구조 - 인터페이스에서 제시되는 연산들을 어떻게 사용하는가 - 의 구성 문제로 변환될 수 있다. 제어 구조는 크게 두 가지 방식으로 구현된다. 하나는 분산 구조(Decentralized Structure)이며 또 하나는 집중 구조(Centralized Structure)이다. (그림 2)는 이러한 제어 구조의 예를 시퀀스 다이어그램을 통해서 보이고 있다. (그림 2)의 (a)는 분산 구



(그림 2) 제어 구조의 예

조의 예로 이를 계단(Stair) 구조라고 부르며 (그림 2)의 (b)는 집중 구조의 예로 이를 포크(Fork) 구조라 한다.

사실, 제어 구조는 두 방식들 중의 하나에 속하기보다는 절충된 형태를 갖게 되는데 이는 두 방식들은 각기 다른 장점을 가지기 때문이다. 우선, 분산 구조의 장점은 제어 구조의 알고리즘적 측면을 추상화할 수 있다는 것이다. 반면에 집중 구조는 제어 구조의 수행 순서를 추상화할 수 있는 장점이 있다. 따라서, 분산 구조는 수행 순서의 변화가 거의 없는 경우에 유리하여 쉽게 새로운 데이터와 그에 대한 연산을 추가하기가 쉽고, 집중 구조는 수행 순서를 변경하거나 새로운 연산을 추가하는 경우에 유리하다.

지금까지 변화의 영향을 지역화하기 위해서 고려되어야 할 사항들을 살펴보았다. 결론적으로 말하면, 가능한 변화의 형태에 따라서 서로 다른 형태의 제어 구조가 개발되어야 한다. 즉, 어떠한 변화가 보다 자주 발생할 것인가가 주된 고려사항이 된다. 아직까지 이 문제에 대한 적절한 분석은 없는 것으로 보인다. 대신, 요즘 대두되고 있는 비즈니스 컴포넌트(Business Component)의 개념[2,3]을 토대로 변화의 형태를 예측해 보도록 하자.

비즈니스 컴포넌트는 재사용의 단위를 확대시키는 과정에서 등장한 개념으로 이해될 수 있다. 재사용의 효과를 높이기 위해서는 재사용의 단위가 커야한다. 반면에 재사용의 단위가 크면 재사용 할 수 있는 기회가 감소하게 된다. 이러한 사항들을 고려하여 비즈니스 컴포넌트는 비즈니스 영역에서 발견되는 개념 수준까지 컴포넌트의 크기를 확대함으로써 재사용의 효과를 높이는 동시에 재사용 기회를 증대하려는 시도이다. 현재 제시되고 있는 비즈니스 영역의 개념들이 크게 행위(Activity/Process)와 엔티티(Entity)로 구분되며[2], 제어 객체와 엔티티 객체의 의미와 유사한 점은 강건한 구조를 얻는데 의미하는 바가 크다.

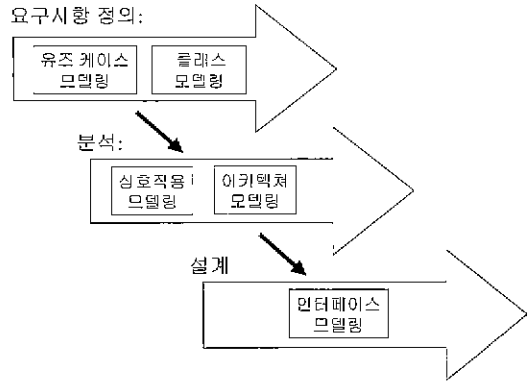
3. 강건성 향상을 위한 CBD 방법론

CBD 개발 과정에서 강건성을 향상 시키기 위해서는 시스템의 행위가 구현되는 과정에 대한 충분한 이해가 있어야 한다. 반면에 시스템의 구조, 즉 시스템의 아키텍쳐는 제어 구조에 많은 영향을 미친다. 계층적 구조를 갖기 위해서는 상호 의존관계가 없어야 하기 때문이다. 이러한 현상에 효율적으로 대처하기 위해서 본 절에서는 Catalysis[1]와 CBD'96[5]에 기반하여 강건성을 고려하는 CBD 개발 절차를 간략히 제시하고자 한다.

(그림 3)은 본 논문에서 제시하고 있는 CBD 개발 절차에 대한 도식화된 그림이다. 앞으로 각 절차에 대해서 설명할 것이다.

요구사항 정의: 요구사항 정의는 전통적인 객체지향 개발방법론과 유사하게 유즈 케이스(Use case)에 기반하여 진행한다. 즉, 사용자의 요구사항을 유즈 케이스 모델을 통해서 기술하고, 각 유즈 케이스에 대해서는 추가적으로 시나리오가 기술된다. 이 과정에서 중요한 점은 사용자의 용어를 적용하여 사용자와 개발자간의 오해를 최소화해야 한다는 것이다.

유즈 케이스에 대한 시나리오를 기술하는 과정에서 개발하고자 하는 시스템의 구조적 이해가



(그림 3) 개발 절차

어진다. 이러한 지식은 클래스 모델을 통해서 기술된다. 즉, 기술된 모든 시나리오들은 시스템의 구조를 클래스 모델에 기술된 바와 일관되게 반영하고 있어야 한다.

분석 : 분석 과정에서는 CBD의 핵심인 아키텍처가 개발된다. 이를 위해서 상호작용 모델링(Interaction Modeling)[1]이 행해진다. 이 과정에서 조인트 액션(joint action)[1]이 사용된다. 조인트 액션은 단지 어떠한 행위가 수행되는지, 그리고 이 행위에 어떠한 객체들이 참가하고 있는지 기술한다. 이를 보다 명확히 알아보기 위해서 상호작용 모델링 과정에서 널리 사용되는 시퀀스 다이어그램과 비교해보자. 시퀀스 다이어그램에서는 앞서 언급된 사항 뿐 아니라, 어떠한 객체가 행위를 시작하는지 까지 기술된다. 즉, 제어 구조 전체가 기술된다. 문제는 아키텍처를 얻기 전에 이러한 제어 구조를 설계하는 것은 의미가 적다는 것이다. 앞서 언급되었듯이 제어 구조는 아키텍처가 상호 의존관계를 갖지 않도록, 그리고 변화의 영향을 지역화하도록 구성되어야 한다. 따라서, 초기에 구성된 제어 구조는 전반적으로 수정될 가능성이 크다. 이를 방지하기 위해서 상호작용 모델링에서는 시퀀스 다이어그램 대신에 조인트 액션을 적용한다. 즉, 조인트 액션의 수행 순서만을 기술할 뿐, 제어구조를 제시하지 않는다. 상호작용 모델링은 각 유즈 케이스에 대해서 수행된다.

상호작용 모델링을 통해서 시스템의 행위가 파악되었으면, 이를 토대로 아키텍처 모델링(Architecture Modeling)을 수행한다. 이를 위해서 우선 서로 밀접하게 연관된 클래스들을 묶어 컴포넌트를 구성한다. 클래스들 간의 관계는 도출된 조인트 액션과 연관관계를 통해서 유추될 수 있다. 컴포넌트들이 도출되면, 조인트 액션들 간의 수행 순서를 고려해서 컴포넌트들 간의 의존관계가 설계된다. 이 의존관계를 설계하는 과정에서 제어구조에 대한 전반적인 사항이 결정된다.

아키텍처 모델링과 제어 구조를 결정하는 과정에서 어떠한 변화가 발생할 것인지 추측하여 이를 반영함으로써 아키텍처의 강건성을 향상시킬 수 있다. 앞서 언급했듯이 변화의 종류에 대한 체계적인 자료는 부족한 실정이며, 시스템의 적용되는 영역에 따라서 각기 다른 변화의 형태들이 나타날 것이다. 여기에서는 e-Business에서 제시되고 있는 비즈니스 컴포넌트의 개념에 따르면, 컴포넌트들을 비즈니스 행위(Business Process) 컴포넌트와 비즈니스 엔티티(Business Entity) 컴포넌트들로 구분하고 비즈니스 행위 컴포넌트들이 집중 구조로 제어 구조를 구성하는 것이 변화의 영향을 지역화할 수 있을 것으로 예상된다. 이 과정에서 중요한 점은 엔티티 객체에 요구되었던 것과 같이 비즈니스 엔티티 컴포넌트가 정확하고 재사용 가능한 연산들을 지원함으로써 업무에 대한 변화의 영향을 비즈니스 행위 컴포넌트에 지역화하는 것이다.

설계 : 설계 과정에서는 분석 과정에서 도출된 아키텍처를 토대로, 정확한 인터페이스의 정의를 기술하는 인터페이스 모델링(Interface Modeling)이 수행된다. 즉, 인터페이스를 구성하는 메소드들의 정확한 구성과 사전조건, 사후조건이 개발된다. 사전/사후조건에 대한 사항들은 Meyer에 의해서 제시된 계약(Contract)을 따른다. 이를 위해서 Catalysis에서 제안된 인터페이스 다이어그램(Interface Diagram) 을 적용한다. 이 다이어그램의 특징은 컴포넌트 내부를 구성하는 클래스들을 일부 보여줌으로써 정확한 사전/사후조건의 기술을 지원한다는 점이다.

4. 결 론

지금까지 CBD 개발 과정을 통해서 강건성에

1) CBD'96에서는 이를 도메인 모델링(domain modeling)이라고 부르나, 요구사항에서 개발되는 클래스 모델과 혼동될 우려가 있어, 상호작용 모델링(interaction modeling)이라고 부르도록 한다.

대해서 고찰해 보았다. 성공적인 e-Busines를 위해서는 적응력이 높은 시스템이 개발되어야 하며, 이를 위해서는 강건한 컴포넌트 아키텍쳐가 구성되어야 한다. 국내의 소프트웨어 개발 실태를 살펴 보면 아직도 많은 소프트웨어 개발 과제들이 구현 중심의 개발 단계에서 벗어나지 못하고 있는 듯하다. 일단 빨리 만들고 보자는 개발자와 관리자의 심리와 열악한 개발 환경이 이러한 과오를 계속 범하게 하고 있는 것이다.

구현 중심의 개발로는 인터넷 시대의 e-Business에 맞는 효율적인 소프트웨어 개발의 필요성을 충족시키지 못하게 될 것이며 이는 바로 기업의 경쟁력 약화와 기회의 상실로 이어질 것이다.

성공적인 CBD의 정착이란 구현 중심의 소프트웨어 개발이 아닌 분석 및 설계 중심의 소프트웨어 개발로의 진화를 의미하는 것으로 CBD는 국내의 소프트웨어 개발 수준을 한 단계 끌어올릴 수 있는 좋은 기회가 될 것이다.

참고문헌

[1] Desmond F. D'souza and Alan C. Wills, Objects, Components, and Frameworks with UML: The Catalysis Approach, Addison-Wesley, 1999.

[2] Paul Harmon, "Components for the E-Business Age," *Component Development Strategies,* Mar., 2000.

[3] Peter Herzum, Oliver Sims, *Business Component Factory: A Comprehensive Overview of ComponentBased Development for the Enterprise,* OMG Press, 2000.

[4] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach,* AddisonWesley, 1992.

[5] Sterling Corp., The CBD96 Standard, version 2.0, Dec., 1997.

배 두 환

1980년 서울대학교 조선공학과 (공학사)
1987년 University of Wisconsin-Milwaukee(공학석사)
1992년 University of Florida (공학박사)
1992년-1994년 University of Florida, Assistant Professor
1994년-1996년 한국과학기술원, 정보 및 통신학과 조교수
1996년-현재 한국과학기술원, 전자전산학과 부교수
관심분야 : 객체지향 기술, 소프트웨어 프로세스 및 개발방법론, 컴포넌트 기반 개발