기업 환경에 최적화된 닷넷 기반 솔루션 구현

왕정훠
고려대학교 컴퓨터정보통신대학원
wang1113@korea.ac.kr

Realization of Solution Based on .NET Optimized in Enterprise Environment

Junghwe Wang,
Graduate School of Computer and Information Technology, Korea University

요 약

현대 기업에서는 다양하고 복잡한 업무 프로세스가 일반화 되어있고, 그에 따라 동일한 작업을 여러 장비 즉, PC, 노트북, PDA등 다양한 장비에서 하기를 원하는 사용자들의 요구가 늘어가는 추세이다. 하지만 대부분의 기업에서도 동일한 업무 프로세스를 유지보수나 재개발 하는데 있어서 많은 시간과 비용이 투자되어온 것이 사실이다. 본 논문에서는 이 같은 문제의 해결방안으로 최근 기술인 닷넷 아키텍처를 기반으로 연구된 자료를 토대로 기업 환경에 맞는 재사용성을 강조한 클래스를 구현하고 기존에 대부분의 기업에서 구현되고 있는 데이터계층, 비즈니스 계층, 유저인터페이스 접근계층으로 분류되는 3-Tier계층 모델에서 한 단계 진보한 닷넷 아키텍처를 응용한 새로운 기업환경에 맞는 닷넷 솔루션 모델을 제시함으로써 닷넷 솔루션의 성능과 팀 개발 업무에서의 비용과 시간을 개선시킬 수 있다는 것을 증명한다.

1. 서 론

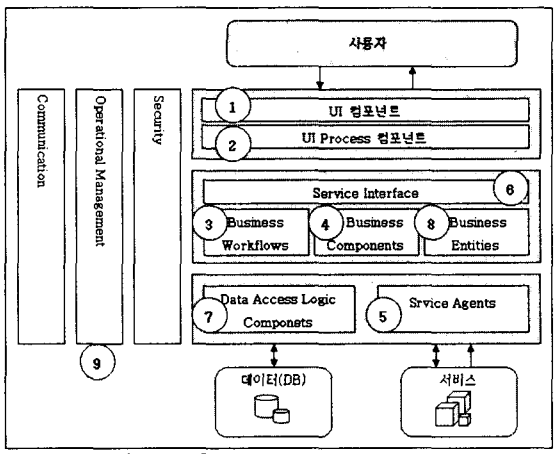
최근 기업 업무의 복잡성과 고급 사용자의 다양한 접근 방법의 필요성 증가로 인하여 효과적으로 비용과 시간을 줄일 수 있는 방법이 끊임없이 연구되어 왔다. 고급 사용자들은 웹사이트, 윈도우 응용프로그램, 모바일과 같은 다양한 방법으로 회사에 있는 시스템에 접근하고 싶어한다. 그러한 요구로 개발된 시스템의 재사용성 측면에서 많은 이슈가 대두되고 있다. 대부분의 기업에서는 이러한 사항을 고려하지 않고 유지보수나 재개발에 많은 비용과 시간이 투자되고 있다. 아직도 수많은 기업들이 동일한 업무 프로세스의 재사용성을 고려한 설계는 미흡한 수준에 머물고 있다.

2. 관련연구

일반적으로 닷넷의 어플리케이션 아키텍처 구조[1][3]는 데이터 접근을 위한 데이터 컴포넌트, 업무 규칙을 가진 컴포넌트, 사용자와의 상호작용을 담당하는 컴포넌트들로 구성되는데, [그림 1]은 일반적인 닷넷 프레임워크에서의 분산환경이다.

2.1 UI(User Interface) 컴포넌트

사용자 인터페이스 컴포넌트는 일반 사용자와 어플리케이션간에 상호작용 즉, 사용자와 직접적으로 대화하는 컴포넌트를 말한다. 닷넷에서의 사용자 인터페이스 컴포넌트란 일반적으로 ASP.NET용 웹폼이나 Windows용 윈폼이다.[1]



[그림 1] 컴포넌트 타입 시나리오

2.2 UI Process 컴포넌트

UI(User Interface) 프로세스 컴포넌트는 사용자와 시스템간의 상호작용하는 프로세스의 플로어를 관리한다. 구현방법은 UI컴포넌트에 별도로 하드코딩 하지 않고, 클래스를 분리한 후에 다양한 UI(User Interface)에 재사용한다.[1]

2.3 비즈니스 워크플로우

비즈니스 워크플로우는 복잡하고 다양한 비즈니스 프로세스를 정해진 순서대로 조정한다.[1]

2.4 비즈니스 컴포넌트
업무규칙이나 업무로직을 구현한다.[1]

2.5 서비스 에이전트
비즈니스 컴포넌트가 외부의 서비스를 이용하고자 할 때 외부 서비스와의 통신(대화)을 담당한다. 즉, 내부서비스와 외부서비스 간의 데이터 포맷 매핑등을 구현한다.[1]

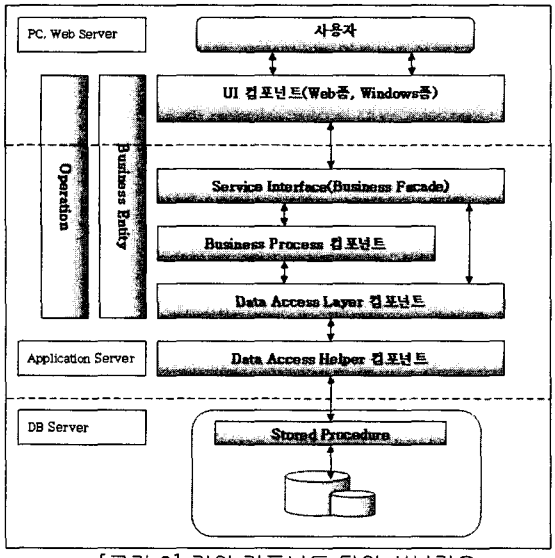
2.6 서비스 인터페이스
서비스 요구자에 따라 서로 다른 통신규약을 제공한다.[1]

2.7 DALC(Data Access Logic Component)
DB로부터 데이터를 CRUD(Create, Read, Update, Delete)와 비즈니스 로직에 관련된 연산을 구현한다.[1][2]

2.8 비즈니스 엔티티 컴포넌트
실제로 어플리케이션에서 다루는 데이터의 형태로 구현한다.[1]

2.9. 그 외의 컴포넌트
Security, Operational Management, Communication 컴포넌트는 예외처리, 어떤 업무에 대한 사용자 권한 관리, 통신정책을 구현한다.[1]

닷넷 분산환경에서 여러 컴포넌트의 모든 기능을 구현할 수 있지만 [그림 2]는 닷넷 분산환경을 재구성 했다.



[그림 2] 기업 컴포넌트 타입 시나리오

여기에서 비즈니스 엔티티와 DALC를 구분하는 가장 큰 장점은 물리적으로 대부분의 어플리케이션들은 객체보다는 비즈니스 데이터를 주고 받기 때문에 저장된 DB의 형태에 관계없이 어플리케이션을 보다 독립적으로 구현할 수 있다. [표 1]은 비즈니스 엔티티 종류를 설명한 것이다. [표 1]에서 보는 바와 같이 비즈니스 엔티티의 종류는 5가지로 나눌 수 있다.

[표 1] 비즈니스 엔티티의 종류

종류	설명
XML	XML 문자열 형 또는 XML DOM (Document Object Model) 객체 형
DataSet	ADO.NET에서 사용되는 DataSet
Typed DataSet	ADO.NET의 DataSet을 상속받아 재구성한 DataSet
Business Entity Component	사용자가 직접 작성한 Class, CRUD는 제외한다.
Business Entity Component with CRUD	CRUD연산을 포함한 Class

비즈니스 엔티티와 데이터베이스를 매핑할 시에는 모든 테이블을 비즈니스 엔티티들로 분리하는 것 보다 어플리케이션에 사용하기 쉽도록 논리적인 비즈니스 엔티티로 구현한다. 특히, 다-대-다 테이블은 분리된 비즈니스 엔티티로 구현한다. 만약에 여러 데이터 소스를 이용한 집합 함수를 사용할 시에는 DALC와 분리하여 구현한다. 즉, CRUD와 같은 기능을 단독으로 구현한다.[2]

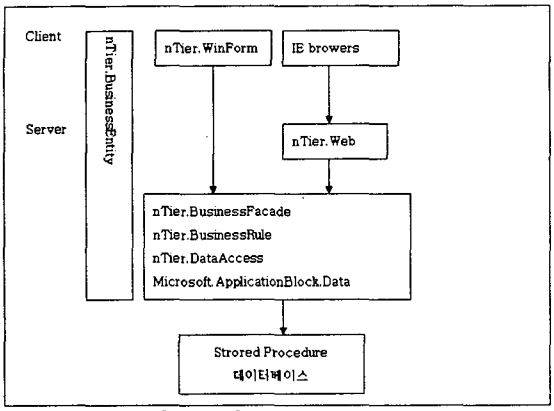
3. 제안모델

[그림 3]은 앞서 설명한 닷넷 기반의 서비스 계층 연구를 통해서 기업 환경에 맞는 프로젝트 배치도를 적용한 것이다. [그림 3]에서 보는 바와 같이 여러 계층의 서비스 레이어가 있다. 즉, BusinessFacade, BusinessRule, DataAccess, BusinessEntity로 구현했다. 실제로 COM+ 서비스 구현 후 사용시에는 BusinessRule과 DataAccess 클래스들 만이 등록되기 때문에 닷넷에서 기본제공 하는 System.EnterpriseService 클래스를 참조해서 사용된다.

이 중에서 DataAccess클래스는 비지니스 데이터를 접근하기 위해서 로직을 캡슐화하고, 마이크로소프트사에서 제공하고 있는 클래스인 Data Access Application Block for .NET을 이용하여 데이터 소스 연결 및 접근 API를 사용함으로써 Data Access Helper Component를 구현한다. 실제 데이터베이스에서 데이터에 대한 관리는 스토어드 프로시져가 담당한다.[4] 여기서 중요한 점은Service Interface(Business Façade)부분을 제외한 클래스들은 클라이언트에 관계없이 일관성 있는 인터페이스로 구현한다. 이때 클래스들은 OOP의 중심개념인 캡슐화, 상속성, 다형성을 최대한 활용하여 구성한다.[5]

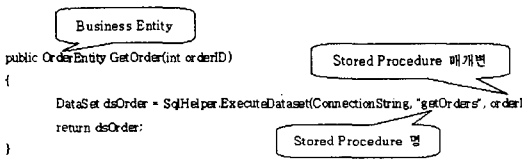
유지보수 측면에서의 우수성은 다양한 클라이언트들 즉, 웹, 윈도우 응용프로그램, 모바일등에서 동일한 업무를 구현시에 비즈니스 규칙은 BusinessRule 서비스 계층에 구현되기 때문에 재사용이 가능하므로 실제로 새로운 클라이언트에서 접근할 수 있는 프로그램을 구현할 시에 BusinessFacade 서비스 계층만을 구현함으로써 기존에

구현했던 BusinessRule을 닷넷프로젝트에서 참조해서 사용함으로써 재사용이 가능하므로 유지보수 하기에 편리하고 신속하게 기업 환경에 대응할 수가 있다.



[그림 3] 프로젝트 배치도

실제로 [그림 4]는 직접적인 데이터소스와의 연결을 Data Access Helper Component에서 SQL의 스토어드 프로시저를 호출하여 구현하였다.



[그림 4] DALC 구현 예

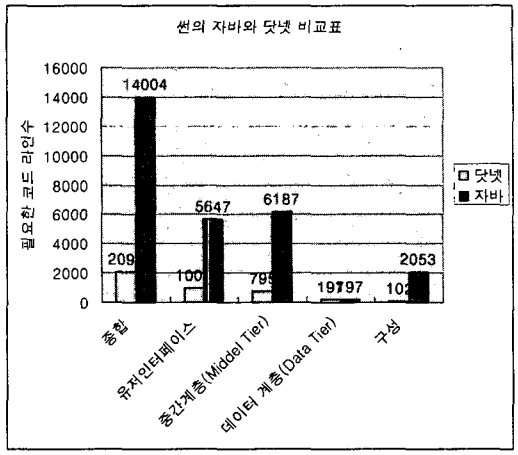
[그림 4]에서 보는 바와 같이 DALC(Data Access Logic Components)에서 스토어드 프로시저를 연결해서 사용하여 구현함으로써 얻게 되는 장점으로는

1. 성능 향상 측면에서 데이터베이스가 스토어드 프로시저의 데이터베이스계획을 최적화하고, 연속적으로 사용시 캐싱을 하기 때문에 처리속도 향상.
2. 스토어드 프로시저 자체로 보안기능을 대처.
3. 하드코드 SQL문 보다 유지보수 측면에서 효과적.
4. 스토어드 프로시저는 실제 데이터베이스의 스키마를 추상화하여 사용할 수 있음.
5. 네트워크 트래픽 줄임.

등이다.

4. 비교 검증

[그림 5]는 썬의 자바와 닷넷으로 같은 기능과 동일한 UI를 구성할 때 개발 코드 라인을 도표화 한 것이다. [그림 5]에서 보는 바와 같이 기존의 자바로 개발에 소요되는 시간이 닷넷에서는 코드의 양을 획기적으로 줄일 수 있는 장점을 가지고 있다. 또한 프로젝트 배치도에서 서비스 인터페이스 부분인 Business Façade의 클래스 구현시에 닷넷에서 제공하는 마샬링을 상속받기 때문에 이기종간에 데이터의 액세스를 자유롭게 구현할 수 있다.



[그림 5] 코드 라인수 비교 분석표

5. 결론 및 향후과제

기업 환경에 최적화 된 닷넷 기반의 솔루션을 실제로 대형 프로젝트에 초점을 맞추어서 구현하였다. 구현된 기술은 C#, ASP.NET, COM+(Enterprise Service)이다.

최근 기업의 업무가 다양하고 복잡해짐에 따라 어떻게 하면 성능을 향상시킬 수 있고, 유지보수 하기에 편리한 방법으로 아키텍쳐를 구성할 수 있는지에 대한 연구는 기업마다 나름대로 구현해왔다. 하지만 닷넷 기반 어플리케이션에 대한 특별한 표준안이 없고, 실제로 업무에 적용해 보고, 테스트해봄으로써 그 성능을 판단해 왔다. 이 논문에서는 닷넷 기반의 어플리케이션 아키텍쳐를 근간으로 기업 환경에 맞는 구현방법에 대해 제시했으며, 어플리케이션 구현 시 성능을 향상시킬 수 있는 방법을 소개하고, 재사용을 할 수 있는 클래스 구현으로 시스템을 확장하고, 최적화할 수 있는 방법을 연구하여 기업환경에 맞는 패턴을 다시 재정의 하였다. 본 논문에서 제안된 시스템은 대형 어플리케이션 설계시에 닷넷의 새로운 개념인 데이타셋의 도입으로 서버의 부하를 줄이고, 개발의 편이성에 중점을 두어 설계되었다. 향후에는 속도를 더 향상시킬 수 있도록 클래스에 대한 연구가 지속적으로 이루어져야 할 것이다.

6. 참고문헌

[1] Keith Short et al. " Application Architecture for .NET : Designing Applications and Services" , 2002, Microsoft

[2] Keith Short et al. " Designing Data Tier Components and Passing Data Through Tiers" , 2002, Microsoft

[3] http://msdn.microsoft.com

[4] http://taeyo.net/lecture/NET/DataAppBlock1.asp

[5] 장시형 역, " C#과 닷넷 플랫폼" , 2003, 사이텍미디어