DOI : 10.5392/JKCA.2011.11.3.036

디자인 패턴을 적용한 닷넷 리모팅 공통 프레임워크 설계 및 구현

Design and Implementation of .NET Remoting Common Framework Applied Design Pattern

강윤성, 이준환, 조한진
극동대학교 스마트모바일학과

Yun-Sung Kang(qsupertj@naver.com), Jun-Hwan Lee(hijuneh@hotmail.com),
Han-Jin Cho(hanjincho@hotmail.com)

요약

현재의 소프트웨어 개발의 주요 쟁점은 재사용 가능한 유연한 소프트웨어의 개발이다. 이미 많은 소프트웨어의 성공적인 개발 경험은 소프트웨어의 환경에 따라 또는 구현하려는 모듈의 성격에 따라 공통된 모습을 패턴으로 추출하여 제시되고 있다. 경험된 패턴을 개발하고자 하는 목적에 맞게 선택하여 이를 재사용하면 빠르고 정확하게 소프트웨어를 개발할 수 있다. 이러한 개발은 성공과 실패에 따라 다른 새로운 경험이 되고 다시 재사용 된다. 소프트웨어 개발에 있어서 디자인 패턴의 적용은 선택이 아닌 필수 사항이 되었다. 본 논문에서는 재사용 가능한 소프트웨어 개발을 위해 분산통신 서비스 기술 중 하나인 닷넷 리모팅 기술을 사용하여 디자인 패턴을 적용한 공통 프레임워크를 설계하고 구현한다.

■ 중심어 : | 디자인 패턴 | 닷넷 리모팅 | 프레임워크 | 소프트웨어 공학 |

Abstract

The main issue in the current software development is the development of a reusable and flexible software. Already many successful software development experiences have been proposed to extract patterns of common look, depending on the software environment or depending on the nature of the module you want to implement. Can develop the software quickly and accurately to select fit for the purpose of developing and reuse using experienced patterns. These developments are depending on the success and failure become a new experience and reuse again. Apply design pattern in software development is required, was not an option. In this paper, design and implement to a common framework applied design patterns for the development of reusable software using .NET Remoting in one of the technologies of distributed communication services

■ keyword : | Design Pattern | .NET Remoting | Framework | Software Engineering |

Ⅰ. 서 론

소프트웨어의 재사용 가능성은 개발기간의 단축, 개발자 수고의 감소, 개발비용의 절감으로 기업의 경쟁력을 높여주며 빠른 개발기간과 완성도 높은 제품으로 고객에게는 만족도를 높여준다.

잘 만들어진 소프트웨어는 잘 만들어진 프레임워크를 탑재하여 소프트웨어의 재사용 가능성을 높여준다.

접수번호 : #101103-005
접수일자 : 2010년 11월 03일
심사완료일 : 2011년 02월 16일
교신저자 : 조한진, e-mail : hanjincho@hotmail.com

소프트웨어의 재사용은 축적된 경험으로 소프트웨어 개발마다 공통적으로 반복되는 일련의 복잡한 과정을 줄여준다.

프레임워크는 소프트웨어의 기반이 되는 핵심 엔진이다. 프레임워크란 소프트웨어의 특정 클래스를 재사용할 수 있게 만드는 관련된 클래스들의 집합이다[1]. 프레임워크는 어플리케이션과 분리되어 독립된 형태로 아키텍처의 기술과 방향을 내포하며 어플리케이션을 조합하여 소프트웨어를 완성한다. 프레임워크는 재사용되거나 변경 또는 확장될 수 있도록 모듈화 한다[2].

프레임워크에 패턴이 접목되면 프레임워크의 구조는 더욱 가시적이며 탄탄해진다. 소프트웨어 공학적인 패턴에는 아키텍처 패턴과 디자인 패턴이 있으며 아키텍처 패턴은 아키텍처를 설계할 때 반복되는 패턴의 정의로서 큰 입자의 기능적인 역할과 관계만을 갖으며 구현은 하지 않는다. 반면 디자인 패턴은 클래스의 역할과 관계를 정의하고 코드의 구현까지 제시한다. 디자인 패턴이란 특정한 전후 관계에서 일반적 설계 문제를 해결하기 위해 상호 교류하는 수정 가능한 객체와 클래스들에 대한 설명이다[1][2]. 디자인 패턴은 클래스와 인스턴스를 식별하여 역할 또는 협력 관계를 정의하고 책임을 할당하여 하나의 디자인 패턴으로 정의된다. 디자인 패턴은 이름이 부여되어 명칭으로 설명되어 지며 당면한 문제와 해결 방법, 결과와 장단점을 제시한다. 이런 각각의 디자인 패턴들은 클래스의 설명과 기능을 더욱 명확하게 하고 프레임워크의 설계와 코드의 재사용 수준을 더욱더 높여준다. 디자인 패턴은 경험적 결과물로서 설계와 아키텍처를 설명하며 이를 재사용 가능하게 한다.

본 논문에서는 분산 환경에서 서비스 제공을 위한 닷넷 리모팅 기술을 사용하여 닷넷 기반의 공통 프레임워크를 구현한다. 또한 특정한 목적의 어플리케이션을 위한 프레임워크가 아닌 주제와 분리된 범용적인 프레임워크를 구현을 목적으로 한다. 우선 Ⅱ장에서는 닷넷 리모팅 아키텍처에서 필요로 하는 몇 가지 설정의 방식을 결정하고, Ⅲ장에서는 아키텍처 패턴을 선택한다. Ⅳ장에서는 알려진 디자인 패턴으로 클래스에 적용할 디자인 패턴을 추출하여 프레임워크 아키텍처를 설계한다. Ⅴ장에서는 프레임워크를 구현, Ⅵ장에서는 구현된 프레임워크가 적용된 프로젝트 사례, 끝으로 Ⅶ장에서 결론 및 향후 프레임워크의 확장 방향을 설명하고 결론을 맺는다.

Ⅱ. 닷넷 리모팅 아키텍처

과거 Windows기반의 분산통신은 DCOM을 통해 이루어졌으나 플랫폼 종속적이며 방화벽에 자유롭지 못하여 서비스 지향적이지 못하였다. 닷넷 리모팅은 사용가능한 프로토콜과 전송 포맷이 자유로워 웹서비스 보다 서비스 제공 방식의 선택이 다양하여 보다 나은 서비스 지향적 모델의 어플리케이션 구현이 가능하다.

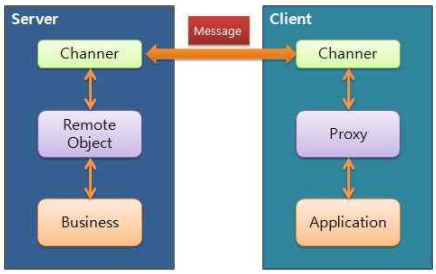

그림 1. 닷넷 리모팅 아키텍처

[그림 1]과 같이 닷넷 리모팅을 구현하기 위해서는 클라이언트가 서버와 통신하기 위해 사용할 원격 객체를 구현하고 전송 방식을 결정하기 위해 채널과 전송 포맷을 설정해야 한다[3].

표 1. 닷넷 리모팅의 설정 종류

설정 종류	방식
채널	HTTP / TCP
메시지	Marshal by value / Marshal by reference
원격 객체 생명관리	클라이언트 / 서버
원격 객체 활성화	Singleton / Single Call
서비스 모듈 실행	수동 / ASP.NET Hosting

[표 1]은 닷넷 리모팅을 사용하기 위해 필요한 설정들과 그 방식의 종류를 보여주고 있다. 채널은 HTTP,

TCP 채널을 사용할 수 있으며 전송 포맷은 SOAP, Binary 포맷이 있다. HTTP, TCP 채널 모두 전송 포맷의 설정이 자유롭다. 메시지를 전송하는 방식으로는 이기종간의 서로 다른 데이터 양식을 변환한다는 의미의 마샬링(marshal)이라는 단어를 사용하여 값에 의한 마샬링(Marshal by value), 참조에 의한 마샬링(Marshal by reference) 두가지 방법이 있다. 값에 의한 마샬링은 인스턴스가 직렬화되는 방식으로 클래스를 선언할 때 [Serializable] 속성을 부여한다. 참조에 의한 마샬링은 원격의 인스턴스의 참조 ID로 접근하는 방식으로 서비스가 이루어질 클래스는 MarshalByRefObject 클래스를 상속하여 선언한다. 또한 서버의 원격객체도 MarshalByRefObject 클래스를 상속하며 이것은 인스턴스가 응용프로그램 도메인 밖으로 나갈 수 없다는 것을 의미한다[3]. 서비스가 이루어질 원격 객체를 활성화 하는 방법으로는 클라이언트가 활성화 하는 방법과 서버가 활성화 하는 방법이 있으며 서버가 원격 객체를 활성화 하는 방법으로는 Singleton, Single Call 방식이 있다. Single Call 방식은 한정된 작업 양을 처리하고 상태 정보를 저장할 필요가 없는 경우에 사용한다. Singleton 방식은 서비스할 객체의 인스턴스를 단 하나만 활성화하는 방식으로 여러 클라이언트가 사용하므로 데이터 공유가 가능하며 이는 Gof 디자인 패턴에서도 설명하는 객체의 활성화 방식 중의 하나이다. 마지막으로 서버에서 서비스를 활성화하는 방법으로는 서비스 프로그램을 직접 실행하거나 ASP.NET이 자동으로 호스팅 하는 방법이 있다. 서비스 프로그램을 직접 실행하려면 채널, 프로토콜, 포맷을 코드로 선언해야 하지만 ASP.NET으로 호스팅하게 되면 어플리케이션의 구성 파일을 통해 설정하므로 변경이 쉽다[3]. 그리고 서비스의 시작을 ASP.NET 런타임이 직접 수행하므로서 더욱 서비스 지향적인 운영이 가능하다.

본 논문에서는 HTTP 채널과 Binary 포맷을 사용하고 ASP.NET으로 호스팅하는 닷넷 리모팅을 구현한다. 서비스 객체의 인스턴스화 방식은 프로젝트의 성격에 따라 적절히 선택한다.

Ⅲ. 아키텍처 패턴

아키텍처 패턴은 소프트웨어 시스템의 기본 골격인 소프트웨어 아키텍처를 구축할 때 적용되는 패턴[4]이며 소프트웨어 아키텍처는 소프트웨어 시스템의 구조 또는 구조들의 집합을 말하며, 이 구조는 소프트웨어의 구성요소와 각 요소의 가시적 특성, 그리고 요소들 간의 관계를 설명해준다[5].

본 논문의 아키텍처 패턴은 프레임워크가 사용되어질 어플리케이션을 고려하여 선정한다. 프레임워크는 닷넷 리모팅 서비스 어플리케이션을 목표로 하여 서비스 지향적이며 확장이 용이한 Three-Layered Service Application 패턴[6]을 따른다.

시스템을 기능별로 Presentation Layer, Business Layer, Data Layer로 분리하고 역할을 정의한다. Presentation Layer는 클라이언트에게 어플리케이션의 사용자 인터페이스를 제공하는 기능을 담당하며 Business Layer는 비즈니스의 기능이 구현되어진 비즈니스 컴포넌트가 위치하여 비즈니스 기능을 서비스 하게 되고 Data Layer는 데이터 액세스 컴포넌트가 위치하여 데이터베이스와의 연결을 담당한다.

어플리케이션을 기능별 Layer로 분리하면 시스템이 분리되어 Layer별 운영의 지속성을 보장하며, 확장과 변경이 용이하고, 보안성과 재사용성이 향상되어 더욱 강건한 시스템 구축을 가능하게 한다[6].

본 논문의 프레임워크는 사용자 인터페이스 기능의 Presentation Layer의 기능은 제외되며 Business Layer와 Data Layer만을 설계하고 구현한다.

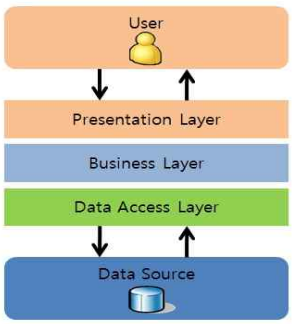

그림 2. Three-Layered Service Application

Ⅳ. 디자인 패턴 및 클래스 디자인

디자인 패턴은 특정 영역의 문제가 주어졌을 때 하나 또는 여러 개의 패턴들이 객체에 책임이 어떻게 할당되어야 하고 객체와 객체 간에는 어떤 관계가 존재해야 하는가에 대한 지침을 제공한다[7].

본 논문이 목표로 하는 프레임워크에서 구현되어야 할 기능적인 부분은 [그림 1]의 닷넷 리모팅 아키텍처에서 필요한 기능과 공통 프레임워크로서의 기능이다 . 클라이언트가 서버와 통신할 수 있는 클라이언트 객체, 서비스 되는 원격 객체, 비즈니스를 담당할 비즈니스 컨트롤 객체, 데이터베이스와 연동할 데이터 컨트롤 객체의 구현을 목적으로 각각의 객체의 역할에서 디자인 패턴을 추출하여 설계한다.

표 2. 서비스 인터페이스 공통 메소드 정의

명칭	역할
Excute	반환값 없는 명령
ExcuteScalar	단일값을 반환하는 명령
GetDataSet	데이터셋을 반환하는 명령
UpdateBatch	다중행에 대하여 개별적인 명령을 수행

[표 2]는 구현할 모듈 사이의 인터페이스에 사용할 메소드를 정의하고 있다. 앞으로 구현될 모듈은 정의된 인터페이스를 사용하도록 설계한다.

1. 브로커 패턴

클라이언트는 서버와 통신하기 위해 원격 객체의 형태를 알아야 메소드를 호출할 수 있다. 여기서 호출되는 메소드는 웹서비스의 WSDL과 같이 공개되어지는 서비스에 대한 규약으로서 원격 객체와 클라이언트간의 통신 규약이다. 클라이언트가 원격 객체의 형태를 알아야 한다는 것은 클라이언트 시스템이 빌드 되는 시점에 원격 객체가 필요하여 클라이언트 시스템이 원격 객체를 포함해야 함을 의미한다. 하지만 원격 객체가 클라이언트의 시스템에 위치하게 되면 원격 객체가 변경될 때마다 새로 배포해야하는 번거로움과 서버의 로직을 담고 있는 코드를 클라이언트에게 공개해야하는 부담감이 있다. 이때 클라이언트에게 원격 객체가 아닌 원격 객체의 인터페이스를 제공하여 재배포에 대한 번거로움과 코드 공개에 대한 부담감을 해결할 수 있다. 클라이언트 객체와 원격 객체 사이에 브로커(Broker) 패턴[8]을 적용하여 프록시 역할의 인터페이스가 되는 브로커 모듈을 추가한다.

그림 3. 브로커 패턴

[그림 3]은 브로커 패턴의 클래스 다이어그램을 논리적으로 표현하고 있다. 클라이언트와 서버사이에 브로커 인터페이스가 위치한다. 클라이언트는 브로커 인터페이스를 직접 참조하고 서버는 브로커 인터페이스를 인터페이스 상속한다.

브로커 패턴은 분산 시스템간의 원격 통신을 위해 사용되는 패턴으로 클라이언트와 서버 사이에 브로커 모듈이 위치하여 시스템을 한 단계 더 분리하고 클라이언트는 브로커 모듈을 통하여 서버와 간접적인 연결로 통신한다. 이로서 브로커 패턴은 서버 시스템의 내부를 보호하여 보안을 향상한다. 브로커 모듈은 인터페이스로 작성하며 클라이언트와 서버는 동일한 인터페이스로 통신하여 유지보수성을 향상하는 브로커 패턴의 장점을 취한다.

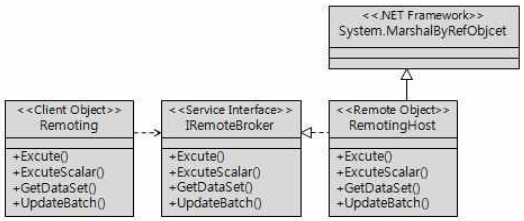

그림 4. 브로커 패턴의 적용

[그림 4]는 브로커 패턴의 적용을 시작으로 본 논문에서 구현될 실제 클래스 다이어그램을 나타내며 지속적으로 추가되는 기능과 디자인 패턴에 대한 클래스 다이어그램을 붙여 나간다. Remoting 클래스는 클라이언트 시스템에 배포되어야할 클라이언트 객체, IRemoteBroker 인

터페이스는 브로커 인터페이스, RemotingHost 클래스는 서버의 원격 객체이다. 원격 객체는 IRemoteBroker 인터페이스를 상속하고 분산통신이 가능하도록 MarshalByRefObject 클래스를 상속한다.

2. 퍼사드 패턴

클라이언트의 서비스 요청을 받은 원격 객체는 비즈니스를 선택해야 하는데 이때 각각의 비즈니스에 직접 연결하지 않고 원격 객체와 비즈니스 객체 사이에 비즈니스 퍼사드(Business Facade)[6] 객체를 두어 비즈니스 객체의 선택을 위임한다.

Gof에서 퍼사드 패턴은 구조 패턴으로 분류되어있다. 퍼사드 패턴은 복잡한 서브시스템을 합성하는 대수의 객체들에 대해 일관된 하나의 인터페이스만을 제공하여 쉽게 사용할 수 있도록 한다[1].

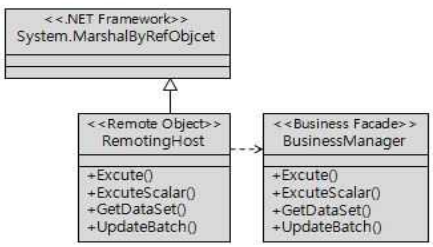

그림 5. 퍼사드 패턴의 적용

[그림 5]는 퍼사드 패턴의 적용으로 추가된 비즈니스 퍼사드 객체의 클래스 다이어그램이다. 비즈니스 퍼사드 객체는 BusinessMaganer로 명명한다.

퍼사드 패턴이 적용되면 서브시스템을 사용하는 객체는 복잡한 서브시스템의 내부를 몰라도 쉽게 사용할 수 있으며 퍼사드 객체로만 서브시스템의 접근이 가능하므로 접근하는 객체와의 결합도를 줄여 코드간 의존성이 감소한다. 퍼사드 패턴과 브로커 패턴은 유사한 구조나 기능으로 퍼사드 객체를 브로커 객체로도 대신하여 사용이 가능하지만 본 논문에서는 브로커 객체는 서비스 인터페이스의 역할만을 담당하고 퍼사드 객체는 비즈니스 퍼사드로서 비즈니스 객체를 선별하고 트랜잭션 처리 역할을 담당한다. 또한 비즈니스 퍼사드 객체로 내부 시스템에 대한 성능, 로깅, 에러 처리를 담당할 수 있도록 확장이 가능하다.

3. 팩토리메소드 패턴

비즈니스 퍼사드 객체가 비즈니스 객체를 선택하는 방법으로는 팩토리메소드(FactoryMethod) 패턴[1]을 적용한다. 팩토리메소드 패턴은 Gof에서 생성 패턴으로 분류되어 객체의 생성을 서브클래스에 위임한다거나 생성할 객체의 타입을 예측할 수 없을 때 사용된다. 본 논문은 특정한 주제에 종속된 프레임워크가 아니므로 비즈니스 퍼사드 객체는 어떠한 비즈니스 객체가 사용될지, 어떠한 비즈니스 객체가 존재하는지 조차 모르는 상태에서 비즈니스 객체를 선택하도록 구현되어야 한다. 현재 대부분의 프로그래밍 언어에서는 이러한 상황에서 어플리케이션의 동적인 운영 변경을 가능하도록 리플렉션을 지원하며 닷넷에서는 System.Reflection 네임스페이스를 제공하여 런타임에 어셈블리명으로 어셈블리를 로드하고 클래스를 인스턴스화 하여 정의된 메소드를 선택하여 호출할 수 있다. 이는 닷넷이 모듈을 중간언어의 상태로 빌드하고 모듈의 정보, 형식, 타입등을 메타데이터로 제공하기 때문에 가능하다[9]. 리플렉션을 사용함으로써 비즈니스 퍼사드 객체와 비즈니스 객체간의 종속성이 완전히 사라지게 된다.

표 3. 서비스 인터페이스 파라메터 정의

형식	파라메터명	값
string	fullName	어셈블리명 + 클래스명
string	methodName	메소드명
string[]	param	저장 프로시저 파라메터
DataTable[]	changedDatatable	저장 프로시저 파라메터
TransactionType	tranType	트랜잭션 사용여부 (Required / NotSupport)

[표 3]은 비즈니스 퍼사드 객체에 팩토리메소드 패턴을 적용하게 되면서 결정된 서비스 인터페이스의 파라메터 형식을 정의하고 있다. [표 2]에서 정의한 서비스 인터페이스 메소드는 [표 3]의 파라메터를 사용하도록 정의한다.

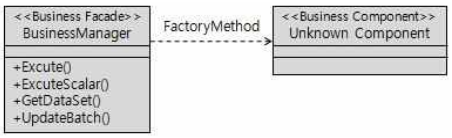

그림 6. 팩토리메소드 패턴의 적용

[그림 6]의 클래스 다이어그램은 BusinessManager 클래스가 비즈니스 컴포넌트 클래스를 리플렉션과 팩토리메소드 패턴을 통해 알 수 없는 형식의 비즈니스 컴포넌트를 참조하고 있음을 나타낸다. 이는 프로그램의 운영 중에 비즈니스 컴포넌트를 동적 참조하여 실행하는 것을 나타낸다.

4. 엔터프라이즈 라이브러리 어플리케이션 블록

마지막으로 비즈니스 객체는 데이터 액세스 객체를 통하여 데이터를 주고받는다. 마이크로소프트는 닷넷 개발자를 위해 엔터프라이즈 라이브러리 어플리케이션 블록(Enterprise Library Application Block)을 제공하여 더욱 손쉬운 닷넷 개발을 보장하고 있으며 항상 최신 버전의 닷넷 프레임워크를 지원하여 새로운 닷넷 프레임워크에 빠른 적응을 돕는다. 엔터프라이즈 라이브러리 어플리케이션 블록에는 로깅, 에러, 암호화, 보안 등의 9가지의 블록이 제공되며 이들은 하나의 프레임워크를 이룬다[10].

본 논문에서는 이중 데이터 액세스 어플리케이션 블록(Data Access Application Block)을 사용하여 데이터 액세스 객체를 구현한다. 데이터 액세스 어플리케이션 블록은 데이터베이스 연결을 더욱 쉽게 하고 구현 코드를 줄여주며 SQL뿐만 아니라 Oracle까지 지원하여 데이터베이스 선택 또는 변경에 유연하고 빠르게 대응한다.

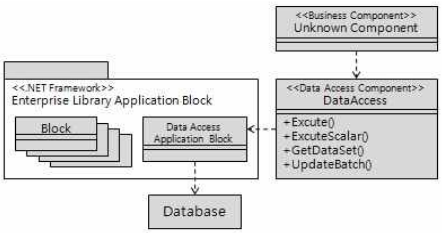

그림 7. 데이터 액세스 객체

위 그림은 데이터 액세스 객체와 데이터 액세스 어플리케이션 블록의 클래스 다이어그램을 나타낸다. DataAccess 클래스는 데이터 액세스 어플리케이션 블록을 호출하여 데이터베이스에 접근한다. 비즈니스 컴포넌트는 항상 DataAccess 클래스의 메소드를 사용하도록 작성한다.

데이터베이스로의 명령은 저장 프로시저(Store Procedure)를 통하여 실행되도록 한다. 저장 프로시저를 사용하면 SQL문 보다 유지보수가 쉬우며 한번 실행되면 데이터베이스가 실행계획을 최적화하고 캐시하여 다음번 사용에 더 나은 성능을 제공한다.

5. 디자인 패턴을 적용한 프레임워크 클래스 다이어그램

[그림 8]은 공통 프레임워크에 필요한 부분별 객체에 대한 설계를 조합하여 최종적인 공통 프레임워크의 클래스 다이어그램과 적용된 디자인 패턴을 보여주고 있다. 원격 객체에서 비즈니스 컴포넌트까지는 IIS에 의해 관리되고 MarshalByRefObject 클래스는 기반이 되는 닷넷 프레임워크에 위치함을 나타내었다.

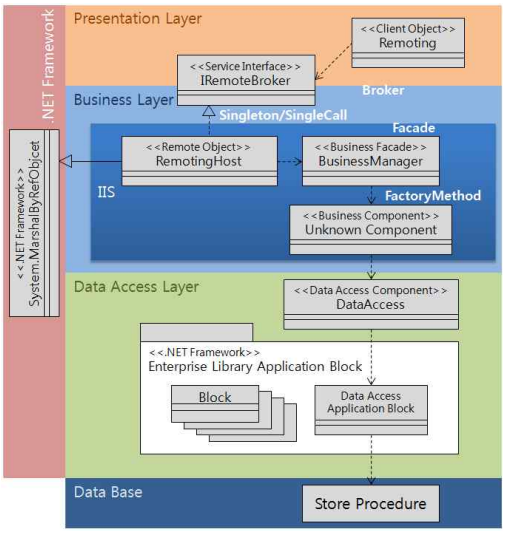

그림 8. 공통 프레임워크 클래스 다이어그램

V. 프레임워크 구현

1. 서비스 모듈 구현

클라이언트에게 제공되는 클라이언트 객체와 프록시 역할의 브로커 인터페이스, 서버에 위치하는 원격 객체를 구현한다.

```
public interface IRemoteBroker
{
    DataSet GetDataSet(string fullName, string methodName,
                       string[] param, TransactionType tranType);
    object ExecuteScalar(string fullName, string methodName,
                         string[] param, TransactionType tranType
    int Execute(string fullName, string methodName,
                string[] param, TransactionType tranType);
    bool UpdateBatch(string fullName, string methodName,
                     DataTable[] changedDataTable, TransactionTy
```

그림 9. 브로커 인터페이스 코드 구현

[그림 9]는 브로커 인터페이스의 코드에 대한 구현을 보여주고 있다. [표 2]에서 정의한 모듈간의 공통 메소드와 [표 3]에서 정의한 공통 파라메터를 따른다.

```
public static DataSet GetDataSet( string fullName, string methodNa
                                  string[] param,  TransactionType
{
    System.Data.DataSet dsReturn = null;
    try {
        ChannelServices.RegisterChannel(new HttpChannel(), false);
        IRemoteBroker oRemoting = null;
        oRemoting =
            (IRemoteBroker)Activator.GetObject(
                typeof(IRemoteBroker),
                m_RemotingHostURL
        );
        dsReturn =
            oRemoting.GetDataSet(fullName, methodName, param, trar
        return dsReturn;
```

그림 10. 클라이언트 객체 코드 구현

[그림 10]은 클라이언트 시스템에 같이 배포 되어야 할 클라이언트 객체 코드의 구현이다. HTTP채널을 사용하고 같은 시스템에 존재해야 할 IRemoteBroker 인터페이스를 통해 서버의 원격 객체의 인스턴스를 호출한다. m_RemotingHostURL 변수는 원격 객체의 위치에 대한 참조 URL로서 응용프로그램 구성 파일(App.config)에 명시하여 가져다 쓰거나 변수로 설정하여 사용한다.

```
public class RemotingHost :MarshalByRefObject, IRemoteBroker
{
    public RemotingHost() { }

    public DataSet GetDataSet(string fullName, string methodName,
                              string[] param, TransactionType tranTy
    {
        return BusinessManager.GetDataSet(fullName, methodName, para
    }

    public object ExecuteScalar(string fullName, string methodName,
                                string[] param, TransactionType trar
    {
        return BusinessManager.ExecuteScalar(fullName, methodName, p
```

그림 11. 원격 객체 코드 구현

[그림 8]과 같이 [그림 11]의 원격 객체는 브로커 인터페이스와 MarshalByRefObject 클래스를 상속하고 있다. 원격 객체는 브로커 인터페이스를 상속하여 형태가 동일하며 특별한 기능없이 BusinessManager 클래스를 호출하고 결과 값만을 반환한다.

2. 비즈니스 모듈 구현

비즈니스 모듈이 되는 비즈니스 퍼사드 객체를 구현한다. [표 3]의 서비스 인터페이스의 fullName, methodName, tranType 파라메터는 최종적으로 비즈니스 퍼사드 객체에서 사용되어진다.

```
public static DataSet GetDataSet(string fullName, string methodName,
                                 string[] parameters, TransactionType
{
    string assemblyName = fullName.Substring(0, fullName.LastIndexOf(
    try {
        DataSet ds = null;
        System.Reflection.Assembly asmbly = System.Reflection.Assembly
        object bizComponent = Activator.CreateInstance(asmbly.GetType(
        System.Reflection.MethodInfo service = asmbly.GetType(fullNam
        System.Reflection.ParameterInfo[] pinfo = service.GetParameter

        if (tranType == TransactionType.Required)
        {
            using (System.Transactions.TransactionScope tx =
                new System.Transactions.TransactionScope())
            {
                if (pinfo.Length == 1 && pinfo[0].ParameterType.FullNa
                {
                    object[] oArrParam = new object[1];
                    oArrParam[0] = parameters;
                    ds = (DataSet)service.Invoke(bizComponent, oArrPar
                }
                else { ds = (DataSet)service.Invoke(bizComponent, para

                tx.Complete();
```

그림 12. 비즈니스 퍼사드 객체 코드 구현

[그림 12]에서 fullName 파라메터는 어셈블리명을 포함하고 클래스를 인스턴스화하는데 사용하고 있다. 트랜잭션 사용여부를 결정하는 TransactionType 값이 Required라면 System.Transactions 네임스페이스의 TransactionScope 클래스를 사용하여 비즈니스 객체에서 실행될 저장 프로시저의 트랜잭션 범위를 비즈니스

퍼사드 객체에서 시작되도록 한다.

3. 데이터 액세스 모듈 구현

비즈니스 객체가 사용하게 될 데이터 액세스 모듈이 되는 데이터 액세스 객체를 구현한다. 데이터 액세스 객체에 데이터베이스 연결 설정에 사용할 변수를 정의한다.

```
public class DataAccess
{
    public DataAccess() { }

    internal static int COMMANDTIMEOUT =
        int.Parse(ConfigurationManager.AppSettings["CommandTimeout"

    private static Microsoft.Practices.EnterpriseLibrary.Data.Data
    private static Hashtable htConAlias = new Hashtable();
```

그림 13. 데이터 액세스 객체 변수 정의

[그림 13]의 COMMANDTIMEOUT 변수는 데이터베이스 연결 제한 시간을 어플리케이션의 구성 파일에서 읽어와 설정한다. 추가로 데이터베이스와 연결되는 데이터베이스 객체를 선언한다. 데이터베이스 객체는 Microsoft.Practices.EnterpriseLibrary.Data 네임스페이스의 DataBase 클래스로 부터 인스턴스화 된다.

```
private static Database GetDataBase(string aliasName)
{
    try
    {
        string strConnectionString = string.Empty;
        if (db == null)
        {
            if (!htConAlias.ContainsKey(aliasName))
            {
                strConnectionString =
                    ConfigurationManager.ConnectionStrings[aliasName].T
                if (!htConAlias.ContainsKey(aliasName))
                    htConAlias.Add(aliasName, strConnectionString);
            }
            else
            {
                strConnectionString = htConAlias[aliasName].ToString();
            }

            if (ConfigurationManager.ConnectionStrings[aliasName].Provi
                "System.Data.OracleClient")
                db = new OracleDatabase(strConnectionString);
            else
                db = new SqlDatabase(strConnectionString);
```

그림 14. 데이터베이스 객체 생성 코드

[그림 14]는 데이터베이스와 연결되어질 데이터베이스 인스턴스를 생성하는 GetDataBase 메소드를 정의하고 있다. 구성 파일에서 설정한 데이터베이스 공급자 정보에 따라 SQL 또는 Oracle로 변경이 가능하다.

```
private static DbCommand GetDatabseCommand(Database db, string comm
    DbParameter[] dbParameters, int commandTimeOut, CommandType com
{
    DbCommand dbCommand = null;
    dbCommand = db.GetStoredProcCommand(commandText);
    dbCommand.CommandTimeout = commandTimeOut;

    if (dbParameters != null)
    {
        foreach (DbParameter param in dbParameters)
        {
            AddParameter(dbCommand, param);
        }
    }
    return dbCommand;
}

private static void AddParameter(DbCommand dbcmd, DbParameter param
{
    if (param.Value == null)
        param.Value = System.DBNull.Value;

    dbcmd.Parameters.Add(param);
```

그림 15. 데이터베이스 연결 설정 메소드 정의

[그림 15]는 데이터베이스 인스턴스에 명령을 설정하는 GetDatabaseCommand 메소드와 파라메터를 추가하는 AddParameter 메소드를 정의하고 있다.

마지막으로 아래 [그림 16]은 비즈니스 객체가 사용하는 데이터베이스 액세스 메소드를 정의하고 있다.

```
public static DataSet GetDataSet(string connectionAlias, string com
    DbParameter[] dbParameters, int commandTimeOut, CommandType com
{
    try
    {
        Database db = GetDataBase(connectionAlias);
        DbCommand dbCmd =
            GetDatabseCommand(db, commandText, dbParameters, comman
        DataSet dsReturn = null;
        dsReturn = db.ExecuteDataSet(dbCmd);
        return dsReturn;
    }
    catch (Exception ex)
    {
        throw ex;
    }
}

public static int Execute(string connectionAlias, string commandTex
    DbParameter[] dbParameters, int commandTimeOut, CommandType com
{
    Database db = GetDataBase(connectionAlias);
    DbCommand dbCmd =
        GetDatabseCommand(db, commandText, dbParameters, commandTim
    return db.ExecuteNonQuery(dbCmd);
```

그림 16. 데이터베이스 액세스 코드 구현

각각의 메소드는 동일하게 앞서 정의한 GetDataBase 메소드에 구성파일의 데이터베이스 연결정보를 파라메터값으로 넘겨 DataBase 인스턴스를 얻고 이와 함께 최종적으로 비즈니스 객체에서 넘어온 저장 프로시저명, 파라메터값을 GetDatabaseCommand 메소드를 통해 명령 정보를 설정한 뒤 메소드 성격에 맞는 실행 메소드를 호출하여 반환값을 넘겨준다.

4. 환경 설정

데이터 액세스 어플리케이션 블록을 사용하려면 어플리케이션의 구성 파일에 엔터프라이즈 라이브러리 어플리케이션 블록 네임스페이스를 등록해야한다[10].

```
<configSections>
  <section name="dataConfiguration"
           type="Microsoft.Practices.EnterpriseLibrary.Data.Configu
                 Microsoft.Practices.EnterpriseLibrary.Data,
                 Version=2.0.0.0, Culture=neutral,
                 PublicKeyToken=null" />
```

그림 17. 구성 파일에 엔터프라이즈 라이브러리 어플리케이션 블록 네임스페이스를 등록

닷넷 리모팅 서비스를 ASP.NET으로 호스팅 하기 위해서는 원격 객체를 IIS의 웹 사이트를 통해 서비스 하고 이 URL은 클라이언트 객체에 노출한다. IIS를 통해 노출된 원격 객체 경로의 구성 파일에 II장에서 결정한 닷넷 리모팅 설정 방식을 적용한다. [그림 14]의 변수 m_RemotingHostURL은 이 공개된 URL을 나타낸다.

[그림 18]을 보면 닷넷 리모팅의 모든 설정을 구성 파일을 통해 쉽게 변경 가능함을 알 수 있다.

```
      <wellknown mode="Singleton"
                 type="RemotingHost.RemotingHost, RemotingHost"
                 objectUri="RemotingHost.rem" />
    </service>
    <channels>
      <channel ref="http">
        <serviceProviders>
          <formatter ref="binary" />
```

그림 18. IIS 원격 객체 경로의 구성파일 설정

VI. 적용사례 및 비교평가

1. 적용사례

본 논문의 프레임워크를 학사관리 C/S(Client/Server) 시스템에 적용하여 공통 프레임워크로의 가능성을 확인하였다. 적용된 학사관리 C/S 시스템에 웹 기반의 학사 정보 시스템을 추가하여 분리된 두 가지 어플리케이션을 통합 시스템으로 구축함으로써 C/S, 웹 어플리케이션 또는 C/S와 웹 어플리케이션의 통합 시스템 모두 공통 프레임워크의 적용 가능함을 확인할 수 있었다.

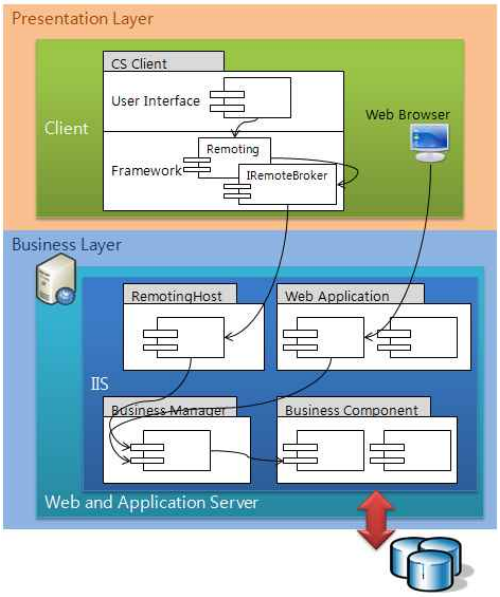

그림 19. 프레임워크가 적용된 어플리케이션 아키텍처

[그림 19]는 적용된 시스템의 간단한 어플리케이션 아키텍처를 보여준다. 본 논문에서 구현된 닷넷 리모팅을 위한 프레임워크가 IIS를 통해 호스팅하게 되면서 이미 구현된 시스템에 웹 어플리케이션의 모듈 부분만 추가되어 기존의 자원을 그대로 이용하거나 웹 어플리케이션에서 서비스되어야 할 비즈니스 컴포넌트만 구현하여 프레임워크의 변경 없이 하나의 프레임워크가 두 가지 방식의 응용프로그램을 서비스하게 되었다. 웹 응용프로그램은 ASP.NET으로 작성하고 이하 로직은 C/S시스템과 동일한 인터페이스로 BusinessManager 클래스만 호출하면 된다.

그림 20. C/S 시스템의 학적정보 프로그램

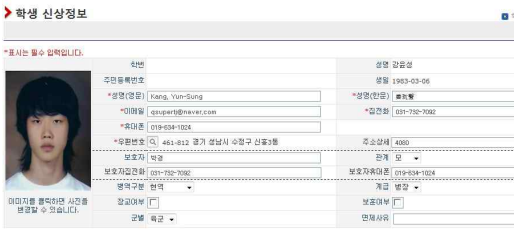

그림 21. 웹 어플리케이션의 학적정보 페이지

[그림 20]과 [그림 21]은 동일한 학적정보를 C/S 시스템과 웹페이지로 다르게 나타내고 있다. 이 같은 상황에서 두 서비스 가 동일한 비즈니스 컴포넌트를 사용할 수 도 있고 한 개, 혹은 여러 개의 비즈니스 컴포넌트의 조합된 사용이 가능하다.

결론적으로 프레임워크의 아키텍처 설계와 이를 구성하는 모듈이 기능별로 잘 분리되어 있어 서브시스템이 추가되는 상황에서도 아키텍처를 변경해야 한다거나 모듈을 수정해야하는 번거로움이 없었고 프레임워크의 유연성을 확인하였으며 재사용 가능성을 잘 나타내었음을 확인할 수 있었다.

2. 비교평가

본 논문에서 구현된 프레임워크에 적용된 기술과 효과를 중심으로 기존에 구현된 시스템을 비교한다.

표 4. 비교평가

비교내용	기존 시스템	구현된 프레임워크 또는 적용 시스템
아키텍처 패턴	Tier 패턴	Layer 패턴
디자인 패턴의 적용	코드가 시스템에 종속되어 재사용성 저하	기능(모듈)이 확실히 분리되어 재사용성 향상
서비스 인터페이스와 퍼사드	서비스 인터페이스 또는 퍼사드가 모든 역할을 담당	서비스 인터페이스와 퍼사드가 분리되어 역할이 분명하고 변경이 용이
C/S 시스템	배포시 비즈니스 로직, 데이터 액세스 로직의 포함	비즈니스 로직, 데이터 액세스 로직을 비포함

이전 시스템에서는 N-Tier의 아키텍처 패턴이 사용되어 왔었다[11][12]. Tier 패턴과 Layer 패턴은 흡사하지만 아키텍처를 물리적 측면에서 배포에 용이하여 아키텍처가 매우 직관적인 장점이 있으나 변경에 있어서는 물리적인 대응책과 매치가 되어야 설명이 가능해지는 불편함이 있다. Layer 패턴은 아키텍처를 논리적인 측면으로만 설계하고 변경한다. 설계를 추상적인 관점으로 바라보기 때문에 변경과 확장이 매우 자유롭다.

디자인 패턴이 적용된 경우를 비교해 보면 디자인 패턴을 전체적으로 특정 시스템 수준에서 적용하였을 때 기능, 클래스의 경계가 모호해지며 항상 시스템에 맞는 요구사항대로 수정이 불가피하다. 하지만 본 논문은 UI와 특정 비즈니스의 기능을 배제한 채로 프레임워크 수준에서 디자인 패턴을 적용하여 프레임워크가 수정과 성능개선을 거쳐 버전이 결정된 이후 에는 시스템의 변경을 요구하지 않으며 기존 버전에 대한 갱신으로 수정이 간단하다. 아울러 확실한 기능의 분리로 디자인 패턴이 적용된 효과가 극대화 되어 구현된 프레임워크는 닷넷 리모팅, 동적 참조, 데이터 액세스, 세 기능이 모듈화 되었다.

미들웨어의 서비스 구현의 설계적인 측면에서 흔히 사용하는 패턴은 서비스 인터페이스, 브로커, 퍼사드, 이 세 가지 패턴을 많이 사용하지만 비슷할 수도 있는 이들 패턴은 각각의 역할과 기능이 서로 다름이 분명하다. 클라이언트와 서버의 연결 구성에서 서비스 인터페이스 또는 퍼사드만이 존재하거나 비즈니스별로 여러 개의 인터페이스가 존재할 경우 서비스 로직 또는 비즈니스 로직의 변경에 대해 많은 수정을 요구하게 된다. 본 논문에서는 서비스 인터페이스와 퍼사드를 분리하고 퍼사드를 비즈니스 퍼사드로 사용함으로써 시스템의 구성이 더 직관적이며 기능이 잘 분리되어 서비스 로직의 변경, 비즈니스 처리 로직의 변경에 더 효과적으로 대처할 수 있다.

본 논문의 닷넷 리모팅 프레임워크를 C/S 시스템에 적용하면서 기존 C/S 시스템의 단점을 보완한 스마트 클라이언트(Smart Client)[13] 기반의 C/S 시스템에 운용하게 됨으로써 프리젠테이션 레이어 이하의 비즈니스 레이어, 데이터 액세스 레이어의 모듈이 배포에서 제외되어 클라이언트의 설치 프로그램이 한결 더 가벼워지며 프리젠테이션 레이어 이하의 레이어에 대한 코드를 공개하지 않는 효과를 얻게 됨으로써 기존의 C/S

시스템과 차별화된 성능을 얻을 수 있었다.

Ⅶ. 결론 및 향후 연구

본 논문에서는 닷넷 리모팅 시스템의 공통 프레임워크의 구현이 목적이었다. 하지만 어떤 특정 기술을 위한 프레임워크는 그 기술에 종속되어 프레임워크에 대한 가치는 기술이 변해감에 따라 점차 잃어갈 것이다. 따라서 프레임워크가 특정 기술에 종속되지 않고 유연하고 재사용 가능하도록 설계하고 구현하기 위해 잘 알려진 아키텍처 패턴과 디자인 패턴을 적용하였다.

구현된 프레임워크는 모듈별 기능이 분명하여 프레임워크를 닷넷 리모팅 모듈, 재빌드 없이 운영 중 로직을 변경할 수 있는 동적 참조 모듈, 데이터 액세스 모듈로 분리할 수 있으며 재구성 또한 가능하게 되었다.

현재 분산통신에 관한 닷넷 기술은 닷넷 리모팅에서 WCF(Windows Communication Foundation)로 이동하고 있으며 웹 기반 기술은 ASP.NET, ASP.NET MVC, 닷넷 웹서비스, 실버라이트(Silverlight)등 다양한 기술이 존재하여 여러 환경에서 소프트웨어를 개발할 수 있다. 본 논문에서 구현한 프레임워크를 특정한 기술과의 의존성 분리를 확실히 하고 마이크로소프트의 여러 기술에도 능동적으로 대처하여 새로운 기술에 대한 프레임워크 모듈과 기존에 구현된 공통 프레임워크 모듈과의 다양한 조합이 가능하도록 구현하고자 한다.

끝으로 본 논문의 프레임워크는 다소 운영적인 측면이 반영되지 않아 추가적으로 암호화, 성능, 로깅, 에러처리 등의 기능이 추가되어 운영에 무리가 없도록 보완되어야 하며 프레임워크의 다른 기능으로 코드의 구현 속도를 높이기 위해 자주 사용되는 메소드를 지속적으로 공통 메소드로 추출하여 프레임워크가 가져가야할 것이다.

참 고 문 헌

[1] Erich. Gamma, Richard Helm And Ralph Johnson, John Vlissides, *Design Patterns*, Addison-Wesley, 1995.

[2] 전병선, *.NET Enterprise System CBD 개발 방법론*, 영진닷컴, 2004.

[3] Simon.Robinson, *Professional C#*, Wrox, 2001.

[4] 궁상환, "USN 미들웨어 설계사례를 통한 패턴지향 아키텍처 설계방법의 개선", 한국콘텐츠학회논문지, 제7권, 제11호, pp.1-8, 2007.

[5] 궁상환, "UML을 응용한 GLORY 소프트웨어 아키텍처의 표현", 한국산학기술학회논문지, 제10권, 제8호, pp.1970-1976, 2009.

[6] Edward A. Jezierski, *Application Architecture for .NET Designing Applications and Services*, Microsoft patterns & practices, 2002.

[7] 최진명, 류성열, "패턴 기반 소프트웨어 개발을 위한 효과적인 패턴 선정 프로세스", 정보과학회논문지, 제32권, 제5호, pp.346-356, 2005.

[8] David Trowbridge, Dave Mancini And Dave Quick, *Enterprise Solution Patterns Using Microsoft .NET*, Microsoft patterns & practices, 2003.

[9] Joel Pobar, *Dodge Common Performance Pitfalls to Craft Speedy Applications*, Microsoft MSDN Magazine, 2005.

[10] http://msdn.microsoft.com/en-us/library/ff664433(v=PandP.50).aspx

[11] 이환진, 최병엽, "COM+ 기반의 다중 계층 아키텍처 환경", 한국정보처리학회 학술대회논문집, 제12권, 제1호, pp.975-978, 2005.

[12] 권오현, "유비쿼터스 환경의 물류관리업무를 대상으로 한 계층구조 컴포넌트의 설계 및 구현", 한국멀티미디어학회논문지, 제9권, 제10호, pp.1361-1370, 2006.

[13] 유경상, *닷넷 스마트 클라이언트:스마트 클라이언트가 작동하는 원리 살펴보기*, 마이크로소프트웨어, 2007.

저 자 소 개

강 윤 성(Yun-Sung Kang) 준회원

- 2009년 2월 : 극동대학교 정보통신학과(공학사)
- 2009년 3월 ~ 현재 : 극동대학교 정보통신학과(공학석사)

<관심분야> : 닷넷, 소프트웨어 공학, 디자인 패턴

이 준 환(Jun-Hwan Lee) 정회원

- 1999년 : 단국대학교 전자공학과(공학석사)
- 2001년 : 단국대학교 전자공학과(공학박사)
- 2001년 ~ 현재 : 극동대학교 스마트모바일학과 교수

<관심분야> : 스마트 앱 콘텐츠, 머신비전, 생체인식

조 한 진(Han-Jin Cho) 종신회원

- 1999년 : 한남대학교 컴퓨터공학과(공학석사)
- 2002년 : 한남대학교 컴퓨터공학과(공학박사)
- 2002년 ~ 현재 : 극동대학교 스마트모바일학과 교수

<관심분야> : 정보보호, 스마트폰 보안, 모바일 콘텐츠