IE Interfaces
Vol. 16, Special Edition, pp. 127-131, December 2003.

닷넷 환경에서의 컴포넌트 기반의 파트너 관리 시스템 아키텍처

배성문[1†] · 이상천[2] · 최종태[3]

[1]경상대학교 산업시스템공학부 / [2]경상대학교 산업시스템공학부 / [3]LG CNS 연구개발센터

Component-based Partner Management System Architecture on .NET Environment

Sungmoon Bae[1] · SangCheon Lee[2] · JeongTae Choi[3]

[1]Division of Industrial and Systems Engineering, Gyeongsang National University, Jinju, 660-701

[2]Division of Industrial and Systems Engineering, Gyeongsang National University, Jinju, 660-701

[3]Research & Development Center, LG CNS, Seoul, 100-630

Component-based development approach is used in modern software system development projects that are very complex and large-scale. Technical components are more effective rather than business components in software development. However, on the Microsoft .net environment, the technical component-based development issues are not adopted. Moreover, software architecture on the .net environment is not studied yet because the development platform is recently used in the development of enterprise-level system.

This paper proposes a technical component-based software architecture on .net environment. Seven technical components - configuration, data access, exception, file I/O, log, message, and paging - are contained in the architecture. The proposed architecture enables developers to reduce development time and to concentrate business logic rather than architectural issues. To verify the proposed architecture and the components, a partner management system is developed based on the architecture.

Keywords: component-based development, .NET, partner management system, software architecture

1. 서 론

현대의 소프트웨어 시스템들은 복잡하고 대규모라는 특징으로 정의될 수 있는데, 이러한 특성으로 인해 시스템 개발 프로젝트는 주어진 예산으로 정해진 기간 내에 수행하기가 쉽지 않다. 시스템 개발에 필요한 비용을 감소시키고, 안정적인 시스템을 단기간에 개발하기 위해 소프트웨어 공학적인 다양한 연구가 수행되었다. 하지만, 현재의 소프트웨어 공학은 다음과 같은 문제점을 해결해야만 한다 (강교철, 2001).

- 소프트웨어 시스템이 특정 응용프로그램을 위해 구축됨: 유지 보수 및 재사용성의 어려움, 변화에 취약한 구조의 소프트웨어, 낮은 개발 생산성
- 기 개발된 시스템 및 개발 지식의 재사용성이 낮음: 시스템 개발시 재사용성을 고려하지 않음, 개발시에 사용된 지식이 코드나 문서로 축적되지 않음(재사용은 개발자에 의존적임)
- 재사용성이 프로세스로 정립되지 않음
- 사용자의 요구사항 정의와 검증의 시간 차가 너무 큼

†연락저자 : 배성문, 660-701 경남 진주시 가좌동 900번지 경상대학교 산업시스템공학부/공학연구원,
Fax : 055-762-6599, E-mail : bsm@gsnu.ac.kr

— 개발 시간 및 비용 산정의 어려움
— 디자인에 대한 정형화된 분석 방법의 부재
— 일시적이고, 복잡한 프로세스: 소프트웨어 개발 프로젝트의 모니터링, 컨트롤, 관리 및 개선이 어려움

많은 소프트웨어 개발 프로젝트에서는 위의 문제를 해결하기 위한 해법의 하나로 컴포넌트 기반 개발 방식을 도입하여 시스템을 개발하는데, 대부분 프로젝트에서 비즈니스 도메인을 분석하여 재사용이 가능한 비즈니스 컴포넌트를 기반으로 시스템을 구축하고자 시도하였다. 이러한 비즈니스 컴포넌트는 도메인에 전반적으로 사용 가능한 컴포넌트를 설계하기가 상당히 난해하며, 또한 유사한 도메인의 시스템을 개발하지 않는 한 재사용될 기회가 거의 없으므로 실제적인 생산성의 향상이나 안정적인 시스템 개발에 가시적인 성과를 제공하기 어려웠다. 하지만, 테크니컬 컴포넌트를 적용하여 시스템을 개발하는 경우 40%의 개발 생산성을 향상시킬 수 있다(Blechar, 1999).

마이크로소프트에서 제안하고 있는 닷넷 개발 환경은 자바에 비해 단기간에 급성장하여 기업의 시스템 구현 환경의 하나로 자리잡고 있다(Driver, 2002). 하지만, 기업 환경에 구현된 사례나 검증된 아키텍처에 대한 연구는 그다지 많지 않다. 이에 본 연구에서는 테크니컬 컴포넌트를 포함하면서, 닷넷 환경에 적합한 시스템 아키텍처를 제안하고 이를 기반으로 구현한 시스템에 대해 설명하고자 한다.

본 논문은 다음과 같은 순서로 기술된다. 먼저 2장에서는 컴포넌트 기반의 시스템 개발 방식에 대한 관련 연구를 정리한다. 그리고 본 연구에서 제안한 웹 기반 닷넷 시스템 아키텍처 및 구현한 테크니컬 컴포넌트에 대한 상세한 설명은 3장에서 기술한다. 4장에서는 파트너 관리 시스템에 대한 설명을 하고, 테크니컬 컴포넌트를 적용한 시스템 개발에 대해 기술한다. 마지막으로 5장에서는 본 연구의 결론에 대해 설명한다.

2. 관련 연구

재사용성을 바탕으로 하는 컴포넌트 기반의 시스템 개발 분야에는 많은 연구가 수행되고 있다(Jacobson *et.al*., 1997; D'Souza *et.al*., 1998; Herzum and Sims 2002). 하지만, 이러한 대부분의 연구는 가시적인 생산성의 향상을 증명하기 어렵고 또한 이론적인 측면이 많아 이를 해결하기 위해 소프트웨어 제품 라인(software product line)의 개념이 등장하기도 했다(Clements and Northrop, 2002). 소프트웨어 제품 라인은 광범위한 도메인에 전반적으로 적용 가능한 컴포넌트를 개발하는 데 발생하는 한계를 극복하고자 비즈니스 도메인을 특화하여 그 도메인에서만 사용 가능한 컴포넌트를 개발하고 개발 생산성을 증대시키고자 하는 측면이 강하다. 하지만, 이 또한 특정 도메인에 국한되어 사용되는 한계가 있다.

소프트웨어 아키텍처를 효과적으로 설계하여 안정된 시스템을 개발하고자 하는 연구도 있었다(Bosch, 2000). 시스템을 개발하기 전에 미리 시스템의 비 기능적인 요구사항을 고려하여 안정성 있는 소프트웨어 아키텍처를 구성함으로써 전체 시스템의 안정성을 향상시키고 개발비용을 절감하는 접근 방법이다. 이러한 소프트웨어 아키텍처적인 접근 방법과 컴포넌트를 결합하여 자바 환경에서 시스템을 구축한 연구는 이미 상당부분 진행되어 있다 (최종태 *et. al*., 2002). 하지만, 이러한 연구가 닷넷 환경으로 확장된 사례는 없었다. 이에 본 연구에서는 닷넷 환경에서 웹 기반 시스템을 아키텍처 중심적이고 비즈니스 도메인에 독립적인 테크니컬 컴포넌트를 기반으로 하는 시스템 아키텍처를 제시하고자 한다.

3. 닷넷 기반 테크니컬 컴포넌트 구현 및 시스템 아키텍처

본 연구에서 제시한 시스템 아키텍처는 시스템의 안정성 및 성능, 프로세스의 독립성 보장, 프로세스 간 통신 최소화를 통한 성능 향상, 관리의 용이성 및 테크니컬 컴포넌트의 재사용성을 고려하여 설계되었다(안현정, 2002). 이 아키텍처는 표현 계층(presentation layer), 퍼사드 계층(façade layer), 비즈니스 계층(business logic layer), 데이터 계층(data access layer)으로 구성되어 있으며, 닷넷 테크니컬 컴포넌트를 기반으로 하고 있다(<그림 1> 참고).

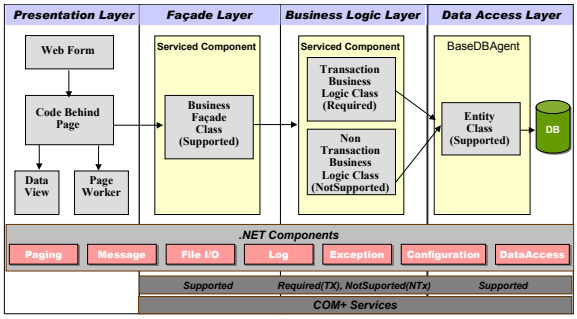

그림 1. 시스템 아키텍처.

먼저 각 계층별로 그 기능과 상세 역할을 정의하도록 한다. 표현 계층은 *web form*과 *code behind page*로 구성되고, 아래와 같은 역할을 수행한다.

— 사용자가 시스템에 접근하는 인터페이스를 제공(웹 기반 사용자 인터페이스와 윈도우 기반 사용자 인터페이스를 모두 지원)
— 사용자로부터 데이터를 입력받거나 출력
— 사용자 입력값의 클라이언트 측 유효성 검사
— 사용자의 입력값을 데이터셋(DataSet)으로 연결
— 서버 측 이벤트 발생 및 서버 측 유효성 검사
— 사용자의 세션 정보유지
— 사용자의 요청을 퍼사드 계층으로 전달 및 수행결과를

전달 받음
- 수행결과를 사용자에게 보여줌

*Web form*은 사용자의 인터페이스를 담당하며 닷넷의 웹 폼 컨트롤(Web Form Control)을 주로 사용하며, 클라이언트 측 유효성 검사를 위해 자바 스크립트를 사용한다. 그리고 *code behind page*는 사용자 인터페이스의 이벤트를 처리하기 위한 코드를 포함하고 있으며, 서버 측 유효성 검사를 담당한다.

퍼사드 계층은 표현 계층과 비즈니스 계층 간의 통신, 사용자의 인증 및 권한 관리, 보안, COM+ 서비스 오류 및 에러 처리, 시스템의 로그를 담당한다. 그리고 필요할 경우 웹 서비스로 비즈니스 로직을 포장할 수 있는 기반을 제공한다.

비즈니스 계층은 사용자의 비즈니스 요구 사항을 실제로 구현한 계층으로서 아래와 같은 역할을 수행한다.
- 비즈니스나 웹 서비스 퍼사드 계층으로부터 요청을 접수
- 코드화된 비즈니스 로직에 의하여 요청을 처리
- 트랜잭션 관리
- 비즈니스 로직 간의 흐름 제어
- 데이터를 접근하기 위하여 데이터 계층을 사용
- 처리 결과를 퍼사드 계층에 전달

비즈니스 계층은 *Transaction Business Logic Class*와 *NonTransaction Business Logic Class*로 나뉘어진다. 이처럼 비즈니스 클래스를 트랜잭션 속성에 따라 분류된 이유는 COM+ 서비스의 트랜잭션을 클래스별로 설정할 수 있기 때문이다. 즉, 구현할 시스템 기능이 동일한 업무영역으로 분류된다 하더라도 트랜잭션 관리가 필요한 업무 등록, 수정, 삭제 등의 업무와 트랜잭션 관리가 필요하지 않은 단순 조회성 업무로 나누어지기 때문에 트랜잭션 관리가 필요한 클래스를 위해 추상화 클래스로 *Transac- tion Business Logic Class*를 정의하였다.

데이터 계층은 데이터 접근과 연관된 부분을 비즈니스 로직과 분리하여 데이터베이스의 변화가 시스템 전체에 주는 영향을 최소화하는 데 있다. 그 역할은 아래와 같다.
- 모든 데이터 서비스에 대한 접근(생성, 변경, 조회, 삭제) 처리
- 비즈니스 수행시 단위 로직 수행
- 데이터 계층은 데이터 서비스와 다른 시스템을 분리할 수 있도록 하여, 데이터 서비스의 변화에 의한 비즈니스 계층 및 퍼사드 계층의 변화를 최소화

본 연구에서 개발한 테크니컬 컴포넌트는 웹 기반 시스템 구축시에 반복적으로 사용되는 모듈을 컴포넌트화하여 시스템 개발시 재사용성에 중점을 두었다. 각 컴포넌트에 대한 상세한 설명은 아래와 같다.

Configuration 컴포넌트는 전체 시스템에 공통적으로 적용될 수 있는 환경정보를 통합 관리함으로써 시스템 구현 및 운영시 편의성을 제공한다. 즉, 해당 시스템의 전반적인 환경 설정(시스템, 데이터베이스, 파일 등)에 대한 정보를 파일을 통해 관리하며, 파일 입출력과 관련된 설정 정보(저장 위치 등)를 관리한다.

Exception 컴포넌트는 시스템에서 발생 가능한 모든 예외 사항을 미리 규정하여 통합 관리하는데, 분산 환경에서의 예외상황까지 처리할 수 있다. 이 컴포넌트는 비즈니스 예외상황과 시스템 예외상황 두 가지에 대해 처리할 수 있도록 구조를 설계하였다.

Message 컴포넌트는 에러 유형에 따른 메시지를 통합 관리하는데, 시스템의 에러 유형에 따라 생성된 코드를 해석하여 사용자 및 관리자에게 에러 메시지를 표시하는 부분과 메시지 관련된 정보를 담아두는 데이터셋 부분으로 구성된다.

Log는 시스템 개발시에는 디버깅을 지원하고 시스템 운영시에는 로깅 및 에러 추적 기능을 제공하는 컴포넌트로서, *LogManager*, *EventLog*, *LogDS*로 구성되어 있다. *LogManager*는 Exception과 연계하여 시스템 에러나 추적이 필요한 부분에 대한 로그를 시스템으로 관리하며, 지원하는 로그 형태는 데이터베이스와 XML 기반의 파일을 지원한다. *EventLog*는 윈도우에서 제공하는 이벤트 로그에 해당 로그 관리를 지원하며, 이벤트 뷰어를 통해 각 이벤트별 로그를 확인할 수 있다. 마지막으로 *LogDS*는 로그에 관련된 정보를 저장하는 데이터셋이다.

File I/O 컴포넌트는 파일을 시스템에 저장할 수 있는 기능을 제공하는 기능으로서 분산 환경을 지원한다. 즉, 웹 서버와 애플리케이션 서버가 분리된 환경에서도 닷넷 리모팅과 웹 서비스 기술을 사용하여 클라이언트에서 애플리케이션 서버로 파일을 전송할 수 있다.

Data access는 다양한 데이터베이스 접근을 용이하게 하고 데이터베이스 연결을 효율적으로 관리하기 위해 개발한 컴포넌트이다.

Paging 컴포넌트는 데이터베이스를 검색한 결과를 페이지 단위별로 검색할 때 필요한 기능을 구현한 컴포넌트로서 표현 계층에서 생성한 페이지 정보(현재 페이지, 정렬할 필드 명, 정렬 순서)를 이용하여 SQL문을 자동 생성한다.

본 연구에서 제시한 아키텍처의 적합성을 검증하기 위해서는 다양한 닷넷 프로젝트에 적용하여 그 경험을 토대로 해야 이를 증명해야 하지만, 현 단계에서 선택할 수 있는 대안은 아니다. 따라서 본 연구에서는 두 가지 방법을 통해 아키텍처의 적합성을 검증하였다. 먼저 다양한 닷넷 환경하에서 실제 운영되는 시스템을 구축한 경험을 가진 마이크로소프트 테크니컬 컨설턴트 그룹과 아키텍처 리뷰를 수행함으로써 본 연구에서 제시한 아키텍처가 실제 프로젝트에 적용 가능함을 현장 전문가를 경험을 통해 확인하였다. 그리고 파일럿 프로젝트를 선정하여 구현함으로써 본 아키텍처의 안정성 및 개발 생산성을 확인하였다. 파일럿 시스템 구현에 대한 상세한 내용은 다음 장에 설명된다. 실제 구현된 파일럿 시스템의 안정적인 운용을 확인한 비즈니스 프로세스 아웃소싱 시스템을 개발하는 프로젝트에서는 본 연구에서 제안한 아키텍처를 기본 모델로 채택하였다.

4. 파트너 관리 시스템

4.1 비즈니스 요구사항

이 절에서는 컴포넌트 기반의 시스템 아키텍처를 적용하기 위한 국내의 한 SI 업체(이하 A사)의 요구사항을 바탕으로 파트너 관리 시스템을 개발한 내용을 설명한다. SI 업체의 경우 파트너로 정의할 수 있는 대상은 시스템 구축시 필요한 하드웨어 및 소프트웨어를 납품하는 다양한 국내외 기업, 그리고 시스템 개발에 참여하여 일정 부분을 담당하는 중소 아웃-소싱 업체들이다.

이 중에서 파트너 관리 시스템의 대상은 하드웨어 및 소프트웨어 공급업체로 한다. SI 프로젝트의 경우 대부분 프로젝트 단위로 모든 의사결정이 이루어지고, 업계의 특성상 이러한 파트너들에 대한 전사적인 관리체계가 부족하였다. 이에 A사에서는 파트너 관리를 담당할 그룹을 선정하고, 전사적인 파트너 관리의 일관된 채널을 확립하게 되었다.

파트너 관리 그룹은 파트너 관리 프로세스를 정립하고, 파트너별/영역별 전략을 수립하였다. 그리고 이러한 체계를 향후 지속적으로 추진하기 위해 파트너 관리 시스템을 개발하게 되었다. 파트너 관리 시스템에서 제공하는 기능은 아래와 같다.

- 파트너/솔루션 정보 : 파트너 및 파트너가 제공하는 솔루션에 대한 기본 정보 - 담당자, 연락처, 프로젝트 지원 내역 등 -을 제공한다.
- 제휴 현황 : 특정 파트너와 전략적 제휴를 맺은 경우 이에 대한 이력 정보, 제휴 조건 등을 제공한다.
- 프로세스 지원 : 구매/제휴/평가 등에 관련된 프로세스를 기존 레거시 시스템과 연계하여 지원한다.
- 파트너별 사업 현황 : 전사적인 관점에서 각 파트너의 사업 현황 정보를 제공해야 한다.
- 커뮤니티 : 파트너에 대한 정보를 교환할 수 있는 내부 커뮤니티와 파트너와 사내의 사용자를 연결하는 외부 커뮤니티를 제공하여 커뮤니케이션 장벽을 제거해야 한다.

4.2 시스템 구현

본 연구에서 구현한 파트너 관리 시스템의 개발 환경은 다음과 같다(<그림 2> 참고).

- 웹 / 애플리케이션 서버 운영체계 : Window 2000 Server
- 데이터베이스 : MS SQL Server 2000, Oracle 8.1
- 웹 애플리케이션 서버 : IIS 5.1
- 분석설계 도구 : Rational ROSE 2001A
- 개발 도구 : Visual Studio .NET 2002 Enterprise Architect
- 테스팅 도구 : Performance Studio 2001A

실제 구현한 시스템의 일부 기능을 이미지를 통해 설명하도록 한다. <그림 3>은 파트너 관리 시스템의 초기 화면으로서 사내 타 시스템과 연계하여 single-sign-on으로 구현하여 별도의 로그인이 필요하지 않으며 이미 설정된 권한에 따라 선택할 수 있는 메뉴 및 접근 권한이 동적으로 생성된다. <그림 4>는 파트너에 대한 정보를 제공하는 통합 뷰 화면이다. 이 화면을 통해 파트너에 대한 기본 정보, 솔루션 정보, 프로젝트 지원 이력, 제휴 현황 등을 조회할 수 있다. 또한, 사내의 지식 관리시스템과 연계하여 특정 파트너 관련 지식을 검색할 수도 있다.

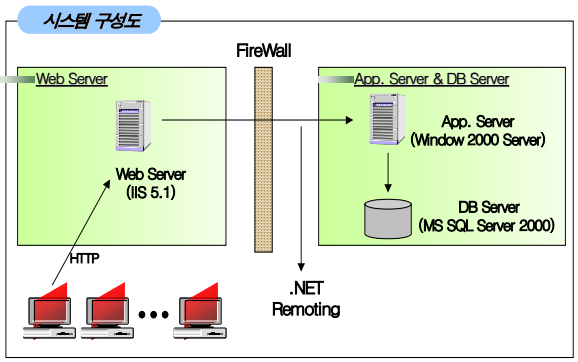

그림 2. 시스템 구현 환경.

그림 3. 파트너 관리 시스템 초기 화면.

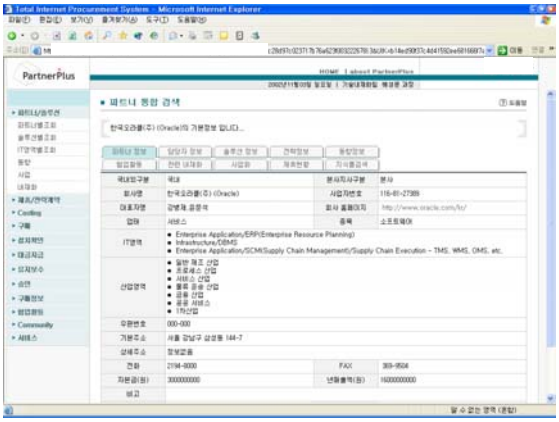

그림 4. 파트너 통합 정보 제공 화면.

파트너 관리시스템을 구현하면서 본 연구에서 제시한 아키텍처와 테크니컬 컴포넌트를 이용하였다. 개발자들에게 시스템 아키텍처와 골격에 해당하는 코딩 표준 및 템플릿을 제공함으로써 개발자들이 좀 더 비즈니스 로직에 전념할 수 있었고, 테크니컬 컴포넌트를 통해 개발 생산성을 경험할 수 있었다. 하지만, 초기에 컴포넌트에 대한 이해를 위해 개발자 교육이 필요했으며, 아키텍처 설계에 대한 검증작업이 선행되어야 하는 어려움도 있었다. 하지만, 이러한 웹 기반 아키텍처나 테크니컬 컴포넌트의 경우는 많은 시스템에 반복적으로 사용될 수 있기 때문에 실제 프로젝트에서는 안정성 및 생산성을 올릴 수 있는 좋은 대안이 될 수 있다.

5. 결 론

본 논문에서는 닷넷 개발 환경에서의 컴포넌트 기반 웹 시스템의 아키텍처를 제시하고, 개발 생산성을 향상시킬 수 있는 테크니컬 컴포넌트 7종을 개발하였다. 그리고 파트너 관리 시스템의 개발을 통해 아키텍처의 안정성 및 개발 생산성을 검증하였다. 본 연구에서 제시한 아키텍처 및 컴포넌트는 반복적이고 일정한 패턴을 가지는 웹 기반 시스템에 적용 가능하며 기업의 시스템 개발비용 및 개발기간을 단축할 수 있다.

참고문헌

강교철 (2001), Product Line Software Engineering, 제2회 SEEK 소프트웨어 공학 워크샵

안현정 (2002), 파트너 관리 시스템 어플리케이션 구조도 설계서, LG CNS 연구개발센터 테크니컬 리포트

최종태, 임미영, 배성문, 김영온, 민병석 (2002), 인사시스템 구축을 통한 컴포넌트 도출 프로세스 적용: J2EE 기반의 EJB 아키텍쳐 적용, *제4회 한국 소프트웨어공학 학술대회 논문집*, 170-179.

Desmond Francis D'Souza, Alan Cameron Wills (1998), Objects, Components, and Frameworks with UML *The Catalysis Approach*, Addison-Wesley

Ivar Jacobson, Martin Griss, Patrik Jonsson (1997), Software Reuse: Architecture, *Process and Organization for Business Success*, Addison-Wesley

Jan Bosch (2000), Design and Use of Software Architectures, Addison-Wesley

M. Blechar (1999), Chasing the Elusive Productivy of Components, Gartner Group

Mark Driver (2002), Java vs. .NET: Competition or Cooperation?, Gartner Symposium/ITxpo 2002

Paul Clements, Linda Northrop (2002), *Software Product Line*, Addison-Wesley