경영정보학연구
제8권 제1호
1998년 6월

정보시스템 개발을 위한 애플리케이션 프레임웍의 설계

- 재사용과 구조적 단순화를 목적으로 -

박 광 호*

Design of an Application Framework for Information Systems Development

- Aiming at Reuse and Architectural Simplicity -

Park, Kwangho

Typical multi-tier client/server architecture includes various and complex objects scattered around the network. Those objects must be clearly identified and designed. Then, their interface must be well-defined so that the integration can be done without major difficulties. This paper presents an application framework to increase reuse and to accomplish architectural simplicity in information systems development. Each component of the architecture is identified and designed. The proposed architecture of the application framework provides the fundamental structure for developing individual functions of an information system. Resulting architectural simplicity achieves effective and efficient maintenance throughout the lifecycle of the information systems. Although the framework aims at the reuse within a project, the productivity of information systems development across projects will be greatly improved eventually. Cases are discussed to evaluate the practicality of the architecture presented.

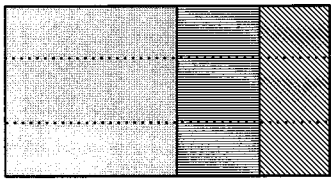
* 한양대학교 경영학부

◈ 이 논문은 1998년 1월 5일 접수하여 2차 수정을 거쳐 1998년 7월 10일 게재확정되었습니다.

I. 서 론

클라이언트/서버 구조의 정보시스템은 기본적으로 다양한 분산 객체로 구성되며 복잡한 인터페이스를 요구하고 있어 메인 프레임 컴퓨팅 구조의 정보시스템에 비해 보다 강력한 표준과 구조적 정형화 작업을 요구한다. 개발자간의 개발 표준 유지와 강제적 준수의 제도화, 공통적 구성요소에 대한 사전 부품화, 개발자의 개발 지식 공유를 통한 개발 생산성 향상 등을 실현 할 수 있는 방법이 정립되어야 하는 것이다. OMG(Object Management Group)의 CORBA (Common Object Request Broker Architecture)를 중심으로 한 객체 서비스 기능은 이런 요구에 대한 일련의 표준화 산출물이라 할 수 있다 [Orfali et al., 1996].

일반적인 클라이언트/서버 구조의 정보시스템은 첫째, 정보에 대한 입출력 기능을 제공하는 사용자 인터페이스, 둘째, 업무 규칙이나 처리 절차를 담당하는 비즈니스 로직, 마지막으로 정보를 장기적으로 저장, 액세스하는 기능을 제공하는 데이터베이스로 구성되는 3계층 구조가 보편화 되어 있다[Watterson, 1995]. Shaw[1996]는 <그림 1>과 같이 클라이언트/서버 환경의 정보시스템 개발 프로젝트에서의 기능별 코드 구성 비율을 보고하였는데, 기본 기능이 56%, 관리/지원 기능이 23%를 차지하며 순수한 업무처리 기능은 21%를 차지하고 있음을 알 수 있다. 여기서 주목할 점은 코드의 대부분이 비 업무처리를 위해 작성되었다는 것이다. 코드의 역할을 보다 구체적으로 구분해 보면 기본 기능 코드는 데이터베이스 관련 기능과 사용자 인터페이스에 해당되는 기능을 구현한 것이며 관리/지원 기능 코드는 사용자 보안, 에러처리, 사용자 인터페이스의 세련화, 복수 데이터베이스 인터페이스 지원, 계층간 인터페이스 기능을 구현한 것이다. 마지막으로 업무 처리 기능 코드는 업무 규칙을 만족시키기 위해 요구되는 데이터 검증, 데이터의 산술적, 논리적 연산, 상황에 따른 기본 값의 설정, 업무 흐름에 따른 절차 처리 등을 구현한 것이다. 이 분석에서 우리는 1차적으로 전체 코드 중 79%나 차지하고 있는 비 업무 처리 기능을 구현하기 위한 노력에 대한 당위성을 주장할 수 있다.



기본기능	관리/지원 기능	업무처리 기능
56%	23%	21%

<그림 1> 프로그램 코드 구성비

재사용을 통한 정보시스템 개발, 신속하고 효율적인 유지 보수 방법의 구현은 소프트웨어 엔지니어링의 오랜 목표이다[Jones, 1990]. 그러나, 재사용의 대상을 어떻게 정하느냐에 따라 재사용 시도의 성공 여부가 달려 있다고 해도 과언이 아니다. 우리는 정보시스템 코드 중 업무에 독립적인 부분을 재사용 대상으로 정하고, 클라이언트/서버 구조의 정보시스템의 내부 구조에 대한 분석에 따라 다음과 같은 목표를 설정하였다.

- 사용자 서비스, 비즈니스 서비스, 데이터 서비스의 3계층으로 구성되는 분산 객체 구조 하의 애플리케이션 프레임웍 (Application Framework, 이하 AF) 구조 정의
- 기본 기능과 관리/지원 기능을 정형화 시켜 재사용할 수 있는 AF의 설계, 개발
- 유사한 업무 처리를 정형화하여 몇 개의 개발 구조로 단순화 처리할 수 있는 AF의 설계, 개발이다.

결국, 이상의 목표는 재사용으로 집약되는데 코드 수준 (Code Level)의 단순 재사용에서 구

조 수준 (Architecture Level)의 설계 재사용까지 포함하게 된다. 일반적으로 재사용은 여러 단계로 발전되는데 본 재사용 성숙도 모델의 3번째 수준인 프로젝트 수준의 재사용[Bassett, 1996]을 목표로 한다. 이는 한 프로젝트 내에서 반복되는 코드 또는 구조를 패턴화 하여 재사용을 극대화하는 것이다. 본 논문에서는 위에서 설정된 클라이언트/서버 구조의 정보시스템 개발 방법의 목표를 효과적으로 실현하기 위해 AF를 제시하고 이를 통한 재사용과 프로젝트의 유지 보수 방법을 구체화 한다.

국내의 경우, 정보시스템 개발에 있어 AF를 적용하는 방법은 아직 초기 단계에 머무르고 있다. 최성운 외[1996], 최은만, 김진석[1996] 등의 연구에서 재사용의 방법으로 AF의 개념만을 소개한 정도이며 강문설, 김병기[1993]나 김태훈 외[1997] 등의 연구에서와 같이 정보시스템 보다는 시스템 소프트웨어에 객체지향 설계 방법을 적용하는 단계이다. 해외에서는 Gamma et al.[1990]의 패턴 연구와 대표적인 AF라 할 수 있는 CORBA는 공통 기능 (Common Facilities), 공통 객체 서비스 (Common Object Services) 등을 중심으로 다양한 형태의 AF 구조를 제시하고 있다[Johnson and Foote, 1988; Deutsch, 1989; Taligent, 1993; Wegscheider, 1996; Shelton, 1996; Bassett, 1996].

그러나, 이들 연구는 정보시스템 개발에 실질적으로 도움이 되는 AF 구조를 제시하기 보다는 개념적 단계나 일부 상업용 컴포넌트 개발의 개발 단계에 머물고 있다. 따라서, 본 논문은 AF나 객체지향 구조의 적용이 아직 시스템 소프트웨어 수준에서 벗어나지 못하여 정보시스템 개발에 있어서의 실용성에 의구심이 대두되고 있는 국내외 상황에 비추어 볼 때 정보시스템 개발에 있어 AF 적용의 실용성을 평가할 수 있는 계기를 마련한다고 볼 수 있다.

본 논문의 2장에서는 본 논문에서 설계할 AF를 정의하고, 개발 목표를 요약하며, 3장에서는 AF의 기본 구조를 제시하고 각 컴포넌트를 설계한다. 4장에서는 현재 AF를 적용하고 있는 4개 프로젝트를 통해 제시된 방법의 실용성을 논의한다. 마지막으로 5장에서는 향후 연구 과제와 결론을 맺고 있다.

Ⅱ. 애플리케이션 프레임웍

2.1 설계 대상 AF의 정의

프레임웍은 다수의 협력하는 클래스 집합이며, 그래픽 편집기, 컴파일러, 회계정보시스템, 투자자문시스템 등 특정 분류의 소프트웨어 개발에 있어 재사용할 수 있는 구조를 말한다[Deutsch, 1989; Johnson and Foote, 1988]. 추상 클래스 (Abstract Class)가 단일 객체에 대한 설계라면 프레임웍은 책임들을 수행하기 위해 협력하는 다수 객체들의 집합에 대한 구조적 설계이다. 따라서, 프레임웍은 추상 클래스보다 규모가 큰 설계이며 동시에 상위 수준의 설계를 재사용할 수 있는 방법이다[Johnson, 1991].

프레임웍은 애플리케이션의 구조를 결정한다. 전체적 구조, 클래스로의 분할, 클래스간의 협력관계, 절차 (Thread of Control) 등을 정의하는 것이다. 프레임웍이 주요 설계 매개변수를 사전에 정의해 줌으로써 개발자는 애플리케이션 도메인의 특성에만 집중할 수 있게 되는 것이다. 프레임웍은 애플리케이션 도메인에 공통적인 의사결정들을 포착한다. 따라서, 비록 프레임웍이 수행될 수 있는 칸크리트 서브 클래스 (Concrete Subclass)들을 포함하고 있으나 코드 재사용보다는 오히려, 설계 재사용을 강조한다고 볼 수 있다.

프레임웍을 기반으로 한 정보시스템 개발은 일반적인 라이브러리를 이용한 개발과 다르다. 라이브러리를 이용할 경우, 라이브러리에서 제공하는 컴포넌트를 작성 중인 프로그램에 포함시키고 호출하여 재사용한다. 이 때, 컴포넌트

의 내부 구조는 공개되지 않으며 수정할 수도 없다. 한편, 프레임웍을 사용하는 경우, 프레임웍의 구조를 기반으로 기능을 추가, 확장해야 하는 코드를 작성하게 된다. 프레임웍의 내부 구조는 공개되어 수정할 수 있다. 프레임웍에서 기능을 추가, 확장할 때, 라이브러리에서 제공하는 컴포넌트를 사용할 수도 있을 것이다. 이런 특성 때문에 프레임웍 기반 정보시스템 개발은 보다 신속하게 착수될 수 있으며 시스템들은 유사한 구조를 가지게 되는 것이다. 유지보수는 간편해지고 사용자에게 일관성 있는 인터페이스를 제공한다. 반면에 이미 구조적 의사결정은 내려져 있으므로 개발자의 독창적 개발의 자유는 상실하게 된다. 결국, 새로운 애플리케이션 개발 노력은 프레임웍과 애플리케이션과의 기능적 차이에 비례하여 각 기능에 대한 구현은 프레임웍의 인스턴스화 결과이며 고유의 코드는 기능 고유의 특성에 따라 결정된다.

프레임웍의 유형은 수평적 프레임웍과 수직적 프레임웍으로 구분할 수 있다. 수평적 프레임웍은 다양한 애플리케이션에 적용할 수 있는 전문지식을 추상한 것으로 사용자 인터페이스, 데이터베이스 등의 애플리케이션에 독립적인 기능들을 수평적으로 분할한 구조를 가진다. 수평적 프레임웍 중 파일 액세스, 분산 컴퓨팅 지원, 디바이스 드라이버 컨트롤 등과 같은 시스템 수준의 서비스는 지원 프레임웍 (Support Framework)이라 부르기도 한다. 반면에 수직적 프레임웍은 특정 문제 분야의 전문지식을 캡슐화한 것으로 제조, 금융, 서비스, 의료분야 등 수직적 기능의 분할 구조를 가진다. 수직적 프레임웍을 구성하는 객체를 비즈니스 객체라 부르는데 비즈니스 도메인에서 활동하고 있는 객체들을 표현한 것으로 비즈니스 이름, 정의, 속성, 행동, 관계, 제약 조건 등을 포함한다. 본 논문에서 설계하고 적용할 AF는 수평적 프레임웍으로 지원 프레임웍을 제외한 특정 업무 분야와 독립적이며 공통적인 객체들을 대상으로 한다.

2.2 AF의 설계 목표

본 논문에서 AF를 설계함에 있어 기본 목표를 Brooks[1965]가 지적한 바와 같이 개념적 무결성에 두었다. 즉, 설계 표준의 구체성 여부와 이에 대한 준수 여부로 정보시스템의 전체적 인터페이스 규격화를 실현하는 것이다. 일반적인 표준 규정은 분석/설계 작성, 개발 도구와 데이터베이스 구현에 대한 명명 표준 (Naming Convention), 윈도우 컨트롤, 메뉴 네비게이션, 툴바 전이 방법에 대한 사용자 인터페이스 표준 (User Interface Convention), 에러 메시지 처리, 사용자 로긴, 비밀 번호 관리, 사용자 사용 실적 관리 등 공통 기능 표준 (Common Function Convention) 등을 포함하고 있다.

이상과 같은 전체적인 표준에 따라 우리는 서론에서 언급한 AF 설계 목표에 따라 다음과 같이 설계 작업을 구체화하였다.

- 3계층 클라이언트/서버 구조에 있어서 설계, 적용하고자 하는 AF 대상 층의 정의
- 정보시스템에서 전체적으로 공통적인 부분과 개별 기능들의 공통적인 부분을 업무에 관련된 부분과 분리하여 정보시스템 개발 구조를 정의
- 도출된 구조와 개별 컴포넌트에 대한 설계

AF 개발의 궁극적 목표는 기업 더 나아가 산업, 그리고 업종에 공통적으로 사용할 수 있는 수평적 프레임웍인 AF의 개발일 것이다. 우리는 1차적으로 단일 프로젝트 내에서 AF를 정의하고 이를 점차 상위 수준으로 발전시키는 현실적인 방향으로 추진하고 있다.

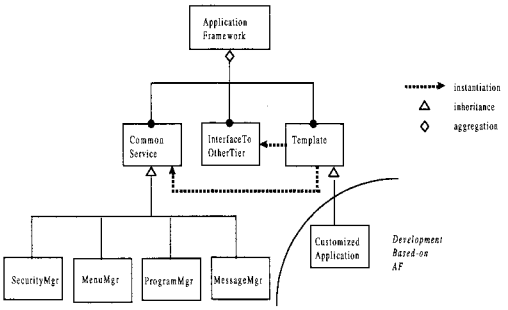
Ⅲ. AF 설계

3.1 3계층 클라이언트/서버 구조에서의 AF 설계 대상 층

일반적인 3계층 클라이언트/서버 구조의 정보시스템은 <그림 2>와 같이 다수의 윈도우, 비즈니스 객체, 데이터베이스 등으로 구성된다. 윈도우는 사용자 서비스 층을, 비즈니스 객체는 비즈니스 서비스 층을, 데이터베이스는 데이터 서비스 층을 각각 대변한 것이다. AF는 각 층별로 도메인에 독립적으로 반복되는 패턴을 도출하는데 있다고 볼 수 있다. 즉, 각 층별로 몇 개의 패턴을 정의하고 이들로 전체 시스템을 개발하는 것이다. 이를 통해 실현되는 구조적 단순성 (Architectural Simplicity)이 AF 개발 목적을 축약한 것이라 할 수 있다. 구조적 단순화의 실현 단계는 전체에서 층으로, 다시 층별로 역할로, 역할별로 클래스로, 클래스별로 속성과 연산으로 구체화, 세분화된다. 앞서 지적한 바와 같이 본 논문에서는 1차적으로 1계층의 윈도우를 주 대상으로 하며 2~3계층과의 인터페이스 패턴은 포함하나 2~3계층, 특히 2계층의 비즈니스 객체 패턴은 향후 연구에서 다루기로 한다.

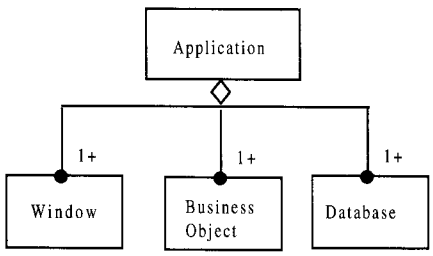
3.2 AF 기본 구조

본 논문에서 제시하는 AF의 기본 구조는 일반적으로 공통 서비스, 타 층으로의 인터페이스, 그리고 템플릿으로 구성된다<그림 3>. 공통 서비스, 타 층으로의 인터페이스는 모든 개별 기능에서 공통적으로 사용하는 전체적으로 공통적인 기능을 모아 놓은 것이며 템플릿은 개별 기능들간의 유사성을 기준으로 이들 내부적으로 공통적으로 사용한 기능을 모아 놓은 것이다. 따라서, 개별 기능의 구현은 템플릿을 기반으로 이루어 진다.

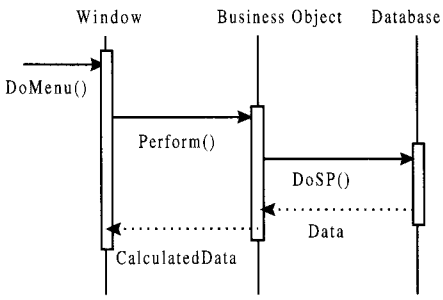


<그림 3> AF의 구성

공통 기능은 주로 사용자 로긴 기능, 보안 기능, 프로그램 관리 기능, 메시지 처리 기능 등으로 구성되며 템플릿 공통으로 사용하는 기능을 분류, 정의한 것이다 또한 템플릿은 기능의 사용 방법에 따라 공통적으로 반복되는 패턴을 추출하여 설계, 개발한 것이다. 공통기능과 템플릿은 서론에서 지적한 바와 같이 비업무적 내용으로 전체 코드의 2/3를 차지하고 있고 있으며 개별 메뉴를 구현할 때 반복적으로 발생하는 작업이다. 따라서, 사전에 이를 AF로 정의



(a) Application Architecture



(b) Typical Interaction among Objects

<그림 2> 3계층 클라이언트/서버 구조

할 때 반복되는 작업을 단순화시키며 전체적인 일관성을 유지할 수 있는 것이다. 이제 AF의 기본 구조에 정의된 각 컴포넌트 설계 내역을 설명하기로 한다.

3.3 공통 기능

1) 보안 관리자 (SecurityMgr)

보안 관리자는 사용자의 시스템 사용에 대한 보안을 관리하는 역할을 담당하는 컴포턴트로 로그인 윈도우 (Login Window)와 함께 정의된다. 보안 관리자의 주요 서비스(연산)는 비밀번호 확인, 비밀번호 변경, 프로그램 사용 권한 확인, 프로그램 사용 기록 여부 확인, 로그 인 시간 저장, 로그 아웃 시간 저장 등으로 정의될 수 있다.

2) 메뉴 관리자 (MenuMgr)

모든 정보시스템은 기본적으로 자주 사용하는 기능을 모아 도구 상자 (Tool Bar)로 정의하여 사용한다. 메뉴 관리자는 사용자가 개별 프로그램을 사용할 때, 도구 상자의 메뉴 항목들에 대한 활성화, 비활성화 작업을 담당하게 된다. 예를 들어, 프로그램의 초기 디스플레이 시, 신규, 조회 메뉴만 활성화하는 서비스나, 일단 신규 메뉴가 선택되면 저장, 삭제 등의 메뉴가 활성화되는 서비스 등을 정의할 수 있는 것이다. 이상과 같은 도구 상자의 개별 메뉴에 대한 활성화, 비활성화는 등록, 조회, 출력 등 프로그램 유형별로 다양하게 정의되며 상당량의 코드를 요구하게 되는 비 업무적 기능이다.

3) 프로그램 관리자 (ProgramMgr)

클라이언트/서버 구조의 정보시스템은 거의 모든 경우, 사용자가 동시에 다수의 윈도우를 열고 작업할 수 있도록 MDI(Multi-Document Interface) 기능을 사용하여 개발된다. 주문 등록을 하다가 생산계획 등록을 할 수도 있고, 작업지시 등록을 하다가 자재 입고 현황 조회를 할 경우도 있다. 즉, 한 업무를 처리하다가 다른 정보가 요구되면 굳이 현재 열려진 프로그램(윈도우, 또는 폼)을 닫고 새 프로그램을 여는 것이 아니라, 다수의 프로그램을 동시에 열고 작업할 수 있는 것이다. 프로그램 관리자는 이와 같이 다수의 프로그램을 열고 작업할 때 필요한 공통 서비스를 정의한 것이다. 프로그램 관리자는 윈도우를 다양한 방법으로 정렬, 분할한다든지, 프로그램을 열 때, 이미 열려져 있는가를 확인한다든지, 프로그램을 닫는 등의 서비스를 제공한다.

4) 메시지 관리자 (MessageMgr)

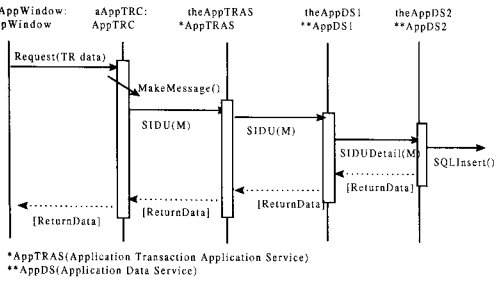
사용자가 시스템을 사용할 때, 시스템에서 적절한 시점에 적절한 메시지를 제공하는 것은 사용자 편리성을 향상시키는데 매우 중요하다. 메시지 관리자는 사용자에게 유용한 작업 정보를 제공하는 서비스를 제공한다. 메시지는 대부분 프로그램에서 공통적으로 발생한다. 따라서, 일련의 메시지를 메시지 코드와 함께 데이터베이스의 테이블로 저장해 놓고 해당 메시지가 필요할 때 마다 메시지 코드로 해당 메시지를 읽어 와 사용자에게 메시지 상자로 보여 주는 기능이 필요하게 된다. 메시지 관리자는 이와 같이 메시지 코드로 메시지를 읽어 와 윈도우에 보여 주는데 필요한 일련의 객체들을 모아 정의한 것이다.

3.4 타 층으로의 인터페이스

다층 클라이언트/서버 구조의 정보시스템 개발은 필연적으로 각 층별 객체 간의 인터페이스를 수반하게 된다. 사용자 서비스 층에서 트랜잭션이 발생하게 되면 이 트랜잭션의 처리를 위해 비즈니스 서비스 층의 해당 객체에 연결하게 된다. 즉, <그림 4>의 SK 신회계정보시스템의 예와 같이 클라이언트 윈도우에서 트랜잭

션이 발생하게 되면 트랜잭션 컨트롤 객체(aAppTRC)가 생성되고 이 객체는 프락시-리모트 (Proxy-Remote) 관계[Orfali et al., 1996]로 구현되어 서버의 해당 비즈니스 객체에게 트랜잭션 처리를 요청하는 Request(TRData), SIDU(M), SIDUDetail(M) 등의 연산을 호출하게 된다.

타 층으로의 인터페이스에서는 층 간 연결을 위한 메시지들을 표준화한다. 트랜잭션의 내용은 각각 다르지만, 인터페이스 자체는 공통적이므로 메시지(연산) 이름 자체를 표준화하여 전체적으로 일관성 있는 설계를 유지하게 되는 것이다. 이와 같은 인터페이스의 표준은 다층 클라이언트/서버 구조의 분산 객체로 개발된 정보시스템 개발의 유지 보수에 있어 필수적이다.



<그림 4> 사용자 서비스 층과 비즈니스 서비스 층과의 인터페이스

SK 사례-1계층(VisualBasic)-2계층(MagnaX)

3.5 템플릿

1) 템플릿 정의

템플릿 (Template)이란 정보시스템 개별 기능(메뉴)을 개발하는 기본 모형을 말한다. 반복되는 기능 유형을 정형화시켜 등록한 기능 수준의 재사용 단위 (Functional Level Reuse Unit)이다. 일반적으로 템플릿은 메인 프레임 컴퓨팅이나 클라이언트/서버 컴퓨팅에 관계없이 화면이나 윈도우 자체를 대상으로 하게 된다. 전표 발행, 주문 등록, 입고 등록, 생산 계획 등록 등은 모두 개별 화면이나 윈도우로 개발되어 사용되기 때문이다. 따라서, 본 논문에서 템플릿이라 하면 하나의 독립된 기능을 구현할 수 있는 윈도우나 폼의 기본 틀을 의미하기로 한다.

템플릿을 이용한 정보시스템 개발은 개발자는 우선 개발하려는 기능(메뉴)에 맞는 템플릿을 선정하고 이 템플릿에서 유전 받아 기본 형태를 갖추고, 화면 컨트롤, 데이터윈도우[1] 또는 데이터 컨트롤[2], 컨트롤의 스크립트 등을 수정하여 완성하는 단계를 거치게 된다. 만일, 템플릿을 사용할 수 없는 전혀 새로운 형태의 기능일 경우에는 가장 유사한 템플릿의 이형으로 개발하여 등록한다.

2) 템플릿 설계

템플릿의 설계와 개발은 구성관계 (Aggregation) [Rumbaugh et al., 1991]를 이용하여 컴포넌트를 조립 작성하는 방법이 일반적일 것이다. 즉, 모든 템플릿에서 공통적으로 사용하는 컴포넌트를 사전에 개발해 놓고 이를 타입별로 구성하여 개발하는 것이다. 따라서, 템플릿 설계 시 구성관계를 기본 방법으로 사용하였다. 템플릿의 설계와 개발에 있어 반드시 고려해야 할 사항은 템플릿의 사용 방법이다. 앞서 설명한 바와 같이 개별 기능은 우선적으로 적절한 템플릿 타입에서 유전 받고, 이를 부분적으로 수정하여 구현하게 된다. 여기서 어떤 구조와 내용을 유전 받아 개발하게 되는지에 따라 개별 기능 구현의 편리성이 크게 좌우됨을 주지해야 할 것이다. 즉, 템플릿의 설계, 개발 방법에 따라 유전 받는 내용이 변경되므로 템플릿의 설계, 개발 방법과 템플릿을 이용한 개발 방법 사이에 직접적인 관계가 있음을 알 수 있다. 즉, 템플릿을 설계하고 구현하는 기본 방법은 구성관계 측면에서, 템플릿을 유전 받아 개별 기능을 구현하는 방법은 유전 관계로 볼 수 있는

1) 파워빌더 (PowerBuilder)가 제공하는 기능
2) 비주얼 베이직 (BisaBasic)이 제공하는 기능

<표 1> 템플릿 타입[3]

타입	분류	특 징
One Master	등록	한 화면에 한 건의 자료를 입력/수정/삭제/조회한다.
One Detail	등록	한 화면에 여러 건의 자료를 입력/수정/삭제/조회한다.
Two Detail	등록	2개(좌,우) 화면에 여러 건의 자료를 입력/수정/삭제/조회한다.
Master-Detail	등록	2개이상의(상,하) 화면에 여러 건의 자료를 입력/수정/삭제/조회한다. 마스터 테이블에 하나의 레코드, 디테일 테이블에 다수의 레코드를 신규로 등록하며 마스터 테이블의 레코드를 삭제하면 이에 대한 디테일 테이블의 레코드는 자동으로 삭제된다.
Process	처리	일마감, 월마감, 집계 처리 등 처리작업 프로그램 타입이다.
Query	조회	조회하는 모든 프로그램 타입이다.
Print	인쇄	프린터,화면,파일 등을 매체로 인쇄하는 모든 프로그램 타입이다.

것이다.

템플릿의 설계에 앞서 선행되어야 할 작업은 타입 (Type)의 설계이다. 템플릿 타입의 설계는 일반적 접근 방법 (General Approach)과 도메인 접근 방법 (Domain Approach)이 있다. 일반적 접근 방법은 업무에 무관하게 사용자의 시스템 행위를 유형별로 정형화시키는 것이다. 반면에 도메인 접근 방법은 특정 업무에 적용할 수 있는 템플릿을 설계하게 된다. 본 논문에서는 일반적 접근 방법을 택하여 사용자의 시스템 행위를 등록, 처리, 조회, 인쇄 등 유형별로 정형화시켜 템플릿 타입을 설계하였는데, AF를 적용하면서 정의되고 사용된 대표적인 템플릿 타입은 <표 1>과 같이 요약할 수 있다. 앞서 설명한 바와 같이 개별 템플릿을 설계, 구현함에 있어 기본적인 접근 방법은 구성 관계를 사용하여 컴포넌트를 조립, 구현하게 되는 것이다. 우리는 템플릿의 설계 과정에서 모든 템플릿에서 공통적으로 사용되는 루트 윈도우 (w_root), 루트 데이터 컨테이너 (dc_root), 자료사전 (w_dict), 조건 (uo_cond), 사용자 정의 컨트롤 (uo_control) 등을 정의하여 애플리케이션 윈도우를 개발하는데 사용하였다<그림 5>. 개별 템플릿 타입의 설계과정은 다음과 같이 정리할 수 있다.

- 루트 윈도우의 서브 클래스로 템플릿 윈도우를 정의한다.
- 템플릿 타입에 따라 템플릿 윈도우에 필요한 데이터 컨테이너의 수와 이들 간의 관계를 설계한다.
- 데이터 컨테이너에 대한 분석에 따라 각각 루트 데이터 컨테이너의 서브 클래스인 애플리케이션 데이터 윈도우를 정의하여 템플릿 윈도우에 붙인다.
- 자료사전, 조건, 그 외 필요한 사용자 정의 컨트롤을 각각 템플릿 윈도우에 붙인다.
- 템플릿 윈도우에 포함된 개별 컴포넌트들의 인터페이스를 정의한다.

이렇게 설계된 템플릿 타입을 이용하여 개별 프로그램을 개발하는 과정은 다음과 같다.

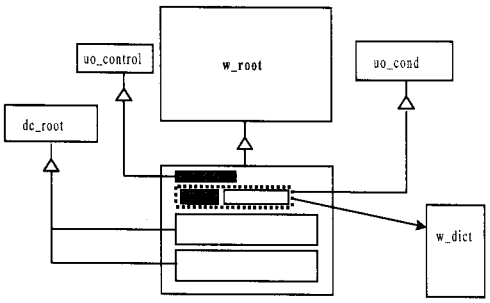
- 템플릿 윈도우를 유전 받아 애플리케이션 윈

3) 주문과 주문 내역 테이블과 같은 일대다의 관계를 갖는 두개의 데이블이 있을 때, 주문 테이블과 같이 피의존적 데이블을 마스터 (Master), 주문 내역과 같이 의존적 테이블을 디테일 (Detail) 테이블로 정의함.

도우, w_app를 만든다.

- 템플릿 윈도우에 포함된 컴포넌트 중에서 불필요한 것을 제거하고 추가로 필요한 사용자 정의 컨트롤을 추가한다.
- w_app의 open() 이벤트에서 컴포넌트간의 관계 및 초기 상태를 초기화하게 된다.
- 템플릿 윈도우에 포함된 컴포넌트(dc_root, w_dict, uo_cond, uo_control 등)의 각 연산에 대한 메소드를 작성한다.

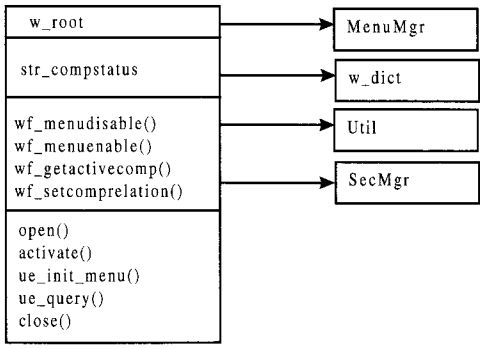
컴포넌트의 연산에 대한 메소드를 구현하는데 있어 공통적인 부분의 경우, 이미 루트 컴포넌트에 구현해 놓는 설계 전략을 채택할 수 있다. 이럴 경우, 각 메소드에 대해 루트 컴포넌트에 구현되어 있는 공통 코드 부분과 개별 기능별로 특수화 시켜야 하는 부분을 구분하기 위해 연산(이벤트나 함수)을 분리하여 정의할 수 있다. 첫째, 개별 기능을 구현할 때, 특수화 시켜야 하는 부분을 정의하기 위해 이벤트나 메소드 이름의 후반부에 _01(), _02()와 같이 연속 숫자를 첨부하거나 둘째, 공통 부분은 we_*event*(), we_*method*() 등과 같이 we_를 접두어로 붙이고, 특수화시켜야 하는 부분은 ue_*event*(), ue_*method*()와 같이 ue_를 접두어로 붙여 정의하는 방법을 사용할 수 있다. 이제 템플릿을 설계하는데 사용한 개별 컴포넌트를 설명하기로 한다.



<그림 5> 구성관계를 이용한 템플릿 설계/구현

3) 루트 윈도우 (w_root)

루트 윈도우는 모든 템플릿 타입의 공통 인터페이스를 추상한 템플릿 개발을 위한 메타 템플릿 (Meta Template)이라고 할 수 있다<그림 6>. 다양한 컴포넌트를 구성하여 개별 템플릿 타입을 구현하게 되는 기본 틀인 것이다. 루트 윈도우는 첫째, 포함하고 있는 컴포넌트의 상태를 저장하기 위한 속성, 둘째, 메뉴의 활성, 비활성을 처리하기 위한 함수, 현재 액티브한 컴포넌트(주로, 데이터 윈도우)를 돌려 주는 함수, 컴포넌트 간의 관계를 설정해 주는 함수, 마지막으로 초기화, 조회 (Query) 처리를 위한 이벤트 등으로 구성된다. 또한, 공통 기능으로 제공되는 보안 관리자, 프로그램 관리자, 메뉴 관리자, 메시지 관리자, 유틸리티 관리자 등을 사용하게 된다.



<그림 6> 루트 윈도우 객체도

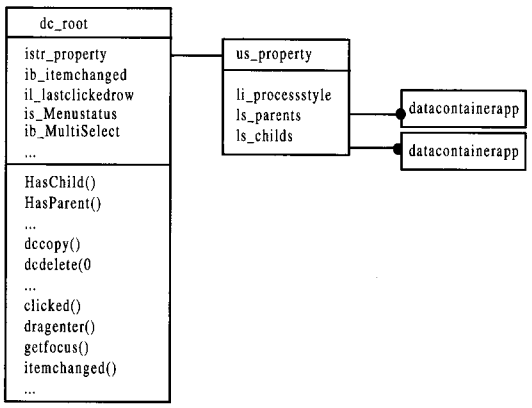
4) 루트 데이터 컨테이너 (dc_root)

데이터 컨테이너는 파워빌더 (PowerBuilder), 비주얼 베이직 (VisualBasic), 델파이 (Delphi), 디벨로퍼2000 (Developer2000) 등과 같은 비주얼 클라이언트 개발 도구에서 일반적으로 제공하는 기능으로 데이터베이스의 테이블에서 데이터를 읽어 와 화면에 보여 주고 사용자가 데이터를 수정, 삭제, 추가할 수 있는 작업 공간이 되어 주는 편리한 기능이다.

루트 데이터 컨테이너는 데이터 컨테이너의

공통 속성과 이벤트, 함수 등을 추상하여 정의, 구현한 컴포넌트이다. 루트 데이터 컨테이너의 설계에 있어 여러 구조적 패턴[Gamma et al., 1995]이 도출되었는데 대표적인 것으로는 마스터-디테일 데이터 윈도우 간의 저장, 수정, 삭제 등의 유기적 데이터 처리를 패턴화한 마스터-디테일 패턴이다. 또한, 마스터-디테일 패턴에 있어 사용되는 일련의 연산 패턴을 행동 패턴 중 하나인 커맨드 패턴[Gamma et al., 1995]을 사용하여 정의하였다.

<그림 7>과 같이 루트 데이터 컨테이너의 주요 속성으로는 마스터-디테일 데이터 컨테이너 관계에서 정의되는 패어런트 데이터 컨테이너(Parent Data Container)의 리스트를 나타내는 ls_parents, 차일드 데이터 컨테이너 (Child Data Container)를 나타내는 ls_childs, 차일드 데이터 윈도우 값의 변경 여부를 나타내는 ib_itemchanged, 마지막으로 클릭된 행을 나타내는 il_lastclickedrow, 메뉴의 활성, 비활성 상태를 나타내는 is_MenuStatus 등이 있다



<그림 7> 루트 데이터 윈도우 객체도

루트 데이터 컨테이너의 함수로는 차일드 데이터 컨테이너를 가지고 있는지를 나타내는 HasChild(), 데이터 윈도우의 한 행을 복사하여 신규 행에 넣어 주는 dwcopy(), 행을 삭제할 때 마스터-디테일 관계까지 고려하여 관련된 모든 행을 삭제하는 dwdelete() 등이 있다. 루트 데이터 컨테이너의 이벤트로는 데이터 컨테이너를 클릭했을 때 발생하는 clicked(), 행을 드래깅 (Dragging) 할 때 발생하는 dragenter(), 포커스가 갈 때 발생하는 getfocus(), 값이 변경되었을 때 발생하는 itemchanged() 등이 있다.

5) 자료 사전 (w_dict)

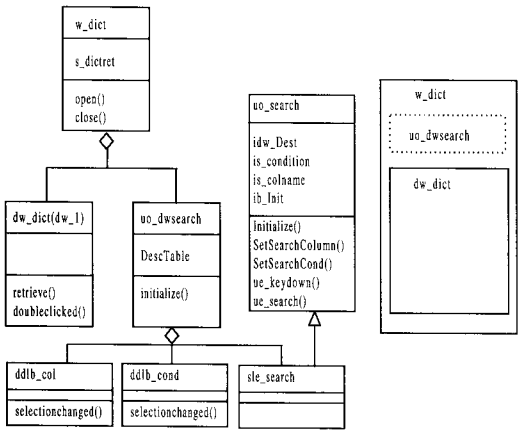
사용자는 개별 프로그램을 사용할 때 조건을 입력하게 된다. 이 때, 사용자가 데이터를 직접 입력하게 하지 않고 이미 등록된 데이터를 보여 주고 마우스로 선택할 수 있도록 하는 자료 사전을 사용하게 된다. 자료 사전의 사용 방법은 동일하며 다만 자료 사전에서 보여 주는 데이터 목록과 선택 시 해당 컨트롤에 채워 주는 속성만이 다를 뿐이다. 우리는 팩토리 메소드 (Factory Method) 생성 패턴[Gamma et al., 1990]을 적용하여 자료 사전을 설계하였다. 즉, 자료사전을 사용하는 입력 컨트롤에는 사용하는 자료 사전과 조건 컬럼들이 초기화 되어 있고 사용자가 입력 컨트롤[4]을 더블 클릭하면 자료 사전이 자료를 데이터 베이스에서 읽어 와 화면에 디스플레이 한다. 사용자가 해당 데이터를 더블 클릭하면 자료 사전은 해당 컬럼 명과 컬럼 값을 리턴하고 사라진다. 마지막으로 입력 컨트롤은 리턴된 값을 채워 놓아 입력 작업을 끝내는 것이다.

자료 사전 <그림 8>은 데이터 윈도우(dw_dict)와 탐색 컨트롤 (uo_dwsearch)로 구성되는데 dw_dict은 자료 리스트를 담는 데이터 컨테이너이며, 탐색 컨트롤은 자료 리스트가 길 경우, 조건을 입력하여 자료의 위치를 찾기 위한 것이다. 탐색 컨트롤은 다시, 탐색 조건 컬럼 (ddlb_col), 탐색 조건 (ddlb_cond)[5], 조건 값(sle_search) 등으로 구성된다. 자료 사전은 보

4) 예, Single Line Edit Box(파워빌더), Text Box(비주얼 베이직)

5) =, <>, <, >, >=, <=, Like 등

통 다음에 설명하는 조건 컨트롤 (uo_cond)과 같은 입력 컨트롤에 의해 사용되는 수동적 컴포넌트이다.

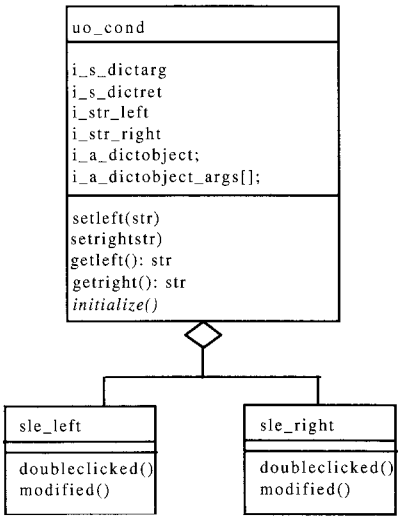


<그림 8> 자료 사전 객체 도

6) 조건 (uo_cond) 컨트롤

조건 컨트롤은 자료 사전과 연동하여 데이터를 입력하는 컨트롤이다. 예를 들어, 전표 등록 윈도우의 경우, 계정 코드와 계정 명 입력을 위한 컨트롤, 사원 등록 윈도우의 경우, 사번과 사원명을 입력하기 위한 컨트롤과 같이 코드와 값의 쌍으로 데이터를 입력할 때 사용하는 컨트롤을 일반화 시켜 정의한 것이다. 따라서 조건 컨트롤은 두개의 입력 컨트롤로 구성되며 좌측 컨트롤이 코드를 우측 컨트롤이 코드에 따른 명칭을 나타낸다. 조건 컨트롤은 초기화 시에 사용하는 자료 사전과 조건으로 사용하는 컬럼명 등을 지정하게 된다.

조건 컨트롤은 <그림 9>와 같이 자료 사전을 열 때 사용하는 아규먼트 (Arguments)를 담는 i_s_dictarg, 리턴 값을 담는 i_s_dictret, 좌측 컨트롤 값을 나타내는 i_str_left 등의 속성을 가진다. 조건 컨트롤의 함수로는 좌, 우측 컨트롤의 값을 정하거나 읽어 오는 setleft(), setright(), getleft(), getright() 등의 함수와 초기화를 위한 이벤트인 initialize() 이벤트가 있다. Initialize() 이벤트는 조건 컨트롤을 윈도우에서 사용할 때 자료 사전과 조건 컬럼명을 지정하기 위해 재구현될 이벤트이다. 조건 컨트롤은 2개의 데이터 입력 컨트롤(파워빌더의 경우, single line edit box, 비쥬얼 베이직의 경우, text box 사용)로 구성된다. 조건 컨트롤의 사용 시 사용자는 입력 컨트롤을 더블 클릭하여 자료 사전을 띄우고 데이터를 입력하거나, 데이터를 직접 입력하게 된다. 따라서, 입력 컨트롤은 더블 클릭 시 처리 작업을 doubleclicked() 이벤트에 구현하고 사용자가 데이터를 입력했을 때, 입력 데이터의 적합성을 체크하기 위한 작업을 modified() 이벤트에 구현하게 된다.



<그림 9> 조건 컨트롤의 객체도

7) 사용자 정의 컨트롤 (uo_control)

사용자 정의 컨트롤이란 개발 도구가 제공하는 일반적 컨트롤 (Generic Controls)을 사이즈, 글꼴체, 글꼴 크기, 마스크 (Mask), 배경 색 등을 지정하여 표준화시켜 모든 윈도우에서 사용하는 것으로 도메인 클래스[Forte, 1994]로 부르기도 한다. 사용자 정의 컨트롤은 스크립트보다는 외형 표준화를 통한 GUI(Graphical User Interface)의 설계 일관성 유지와 유지 보수의

편리성을 위한 목적이 크다. 사용자 정의 컨트롤은 재사용의 최소 단위 컴포넌트로서 이를 조합하여 상위 수준의 재사용 컴포넌트 설계에 사용된다.

AF의 설계 과정에서 일반 도메인 클래스, 조합 도메인 클래스, 비즈니스 도메인 클래스 등의 사용자 정의 컨트롤이 도출되었다. 일반 도메인 클래스 패턴에는 값을 입력하는 EC 패턴 (Edit Control Pattern), 다수의 컨트롤을 담는 CC 패턴 (Container Control Pattern), 버튼 형태를 갖는 BC 패턴 (Button Control Pattern), 단순히 정적으로 문자를 디스플레이 하는 SC 패턴 (Static Control Pattern), 데이터베이스의 데이터를 담는 DC 패턴 (Data Container Pattern) 등이 정의되었다. 다수의 컨트롤로 구성된 보다 복잡한 형태의 조합 도메인 클래스 패턴에는 조건을 표시하기 위해 다수의 컨트롤이 조합된 CD 패턴 (ConDition control Pattern)과 다수의 컨트롤로 조합된 코드를 나타내기 위한 CC 패턴 (Composite Code Pattern) 등이 정의되었다. 마지막으로 비즈니스 도메인 클래스는 일반 도메인 클래스와 달리 애플리케이션 특성을 반영한 패턴이다. 데이터베이스 테이블의 특정 컬럼과 매핑되는 컨트롤은 비즈니스 로직과 직접적으로 관련된다. 비즈니스 도메인 클래스에는 자재코드, 제품코드, 전표번호 등과 같이 테이블의 주키 컬럼에 매핑되는 PK패턴 (Primary Key Pattern)과 근속년수, 나이와 같이 테이블에 저장된 입사일, 생년월일 값이 함수에 의해 계산되어 디스플레이 되는 CV패턴 (Computed Value) 패턴 등이 각각 정의되었다.

3.6 개발 과정

대형 프로젝트는 다수의 개발팀으로 나뉘어 추진될 수 있는데 이 경우, AF도 해당 개발팀별로 구분하여 구축될 수 있다. 다수의 개발팀으로 나뉘어 개발이 추진되면 추가적인 노력이 요구되는데 이는 첫째, 프로그래머가 전체 프로젝트의 한 측면에만 초점을 두어 전체적으로 이해하지 못하기 때문이며, 둘째, 구조적 일관성이 팀간에 유지되어야 하고, 세째, AF간의 의존이 병목을 유발할 수 있기 때문이다. 이 문제를 해결하기 위해서, 첫째, 전체 구조를 유지하는 프로젝트 기술자를 지정하여 AF의 작동을 보장해야 한다. 둘째, 표준 설계, 코딩 지침이 준수되어야 한다. 세째, AF간의 의존관계를 중간 클래스로 분리시켜야 한다. 이는 어떤 AF가 다른 AF의 서비스를 요구할 때, 연결이 하나의 인터페이스나 서버 객체로 구현되어 이 객체만이 다른 AF에 의존하게 되는 것이다. 한편, 성공적인 AF의 개발 조건은 다음과 같다.

- 완전성 (Complete)
  AF는 클라이언트가 요구하는 모든 기능을 지원해야 하며 기본적인 구현이나 기능을 가능하다면 제공해야 한다.
- 유연성 (Flexible)
  AF는 다양한 상황에서 사용될 수 있어야 한다.
- 확장성 (Extensible)
  클라이언트는 쉽게 기능을 추가, 수정할 수 있어야 한다. 즉, AF에서 새로운 클래스를 파생하여 원하는 행동을 커스터마이즈 해야 한다.
- 설명성 (Understandable)
  AF와의 클라이언트 상호작용은 명확해야 하며 AF는 문서화가 충실해야 하며 표준 설계 코딩 지침과 예제 애플리케이션을 제공하여 AF의 사용방법을 교육해야 한다.

Ⅳ. 적용 사례

본 논문에서 제시한 AF 구조는 4개사의 정보시스템 개발 프로젝트에 각각 정의되었다. 이미 언급된 바와 같이 각 프로젝트별로 개별적인 AF를 설계, 적용하였다. <표 2>는 4개 프로젝트에서 설계된 AF의 특징을 비교, 분석한 것이다.

제시된 구조에 따라 AF를 설계하고 적용한 최초 사례인 대성산소의 경우, 총 845본의 기능(메뉴)을 20개 템플릿 타입을 사용하여 제조, 수급, 영업, 회계, 사업계획, 신규투자, 고정자산, 시스템 정보 등을 포함한 통합 경영정보시스템을 개발하였다. 대성산소는 완벽한 3계층 클라이언트/서버 환경으로 개발되지는 않았으나, 데이터베이스 관련 작업을 대부분 저장 프로시져(Stored Procedure)로 처리하여 3계층에 가까운 구조로 개발되었다. NT 서버 환경에서 파워빌더 (PowerBuilder) 5.0과 SQL 서버 6.5를 기본 도구로 개발한 AF의 기본 템플릿은 윈도우 네비게이션, 툴바 작업, 데이터 디스플레이 등의 유형별 표준을 정의, 설계하였다. 템플릿 간의 유사성을 별도의 루트 윈도우로 일반화 하지 않았으며 에러 메시지 처리, 메뉴 관리, 보안 관리 등 비 가시적 모듈에 대한 재사용은 시도되지 않았다. AF를 이용한 개발 방법은 가장 적합한 템플릿을 선정하고 이를 유전받아 _01, _02, ... 로 표시된 연산을 재구현하고 적절한 메소드를 추가, 작성하는 방법을 사용하였다. 개발 생산성은 개발 후기로 갈수록 가속화되었으나 중복 코드가 다수 발견되어 이를 제거하였다. 실제 기능 구현 시 유전 관계는 1단계로 제한하여 성능 오버헤드를 줄였으며 스크립트의 추적도 용이한 상태이다. 또한, 동일한 템플릿을 유전 받아 개발된 프로그램들의 공통 기능을 변경할 경우, 상위 템플릿만 수정하면 되기 때문에 유지 보수가 간편하고 일관성있게 수행될 수 있다.

대성 산소와 동일 그룹 계열사인 캠브리지 필터는 동일한 환경, 툴로 통합 정보 시스템을 구축 중인데 대성 산소와는 달리 다수의 템플릿을 설계, 개발하지 않고, 모든 개별 기능(단위 윈도우) 개발에 공통적인 기능을 단일 루트 템플릿으로 모아 개발하고, 개별 기능은 루트 템플릿에서 유전 받아 개발하는 방식을 취하고 있다. 또한, 다수의 사용자 정의 클래스, 루트 데이터 컨테이너, 자료 사전, 조건 컨트롤 등 다양한 형태의 컴포넌트를 설계, 개발하여 이를 재사용하는 방식으로 정보 시스템을 개발하여 반복적으로 발생하는 패턴을 찾아 템플릿으로 정형화시키는 작업을 계획하고 있다.

<표 2> AF 설계와 적용 사례 요약

사례	응용분야	D/S구조	개발환경	템플릿
대성산소	통합MIS	Semi 3 Tier	NT/Win95	20 타입
캠브리지필터	통합MIS	Semi 3 Tier	NT/Win95	1 타입
KCC	의료정보	3 Tier	UNIX/Win95	30 타입
SK	회계정보	3 Tier	MVS/UNIS/ Win95	1 타입

KCC는 약 3000본의 기능으로 구성된 분당 제생 병원 프로젝트에 AF를 설계하여 적용하고 있는데 기능상 배타적인 업무 영역인 원무, 진료, PACS, 일반 관리 등 4개 서브 프로젝트팀으로 분리하여 추진되었다. UNIX 환경에 포르테 (Forte), 오라클 (Oracle)을 툴로 사용하는 전형적인 3계층 클라이언트/서버 구조로 개발 중인데 분산 객체 구조를 기반으로 객체 분석, 설계 작업을 완료하였다. 전 분야에 공통적인 템플릿과 서브 프로젝트 팀별로 독자적인 템플릿을 설계, 개발하여 프로젝트의 전체적인 일관성을 유지하면서 동시에 업무별로 특성을 반영하는 하이브리드 형태의 AF를 적용, 개발하고 있다.

SK는 IBM 메인 프레임에 COBOL/CICS-DB2, UNIX 기능서버에 마그나엑스 (MagnaX) 4세대 언어를, PC Windows95에 비주얼 베이직 (Visual Basic)을 사용한 전형적인 IBM 호스트 중심의 3계층 클라이언트/서버 구조로 신 회계정보시스템을 개발하였다. 1계층의 클라이언트의 비주얼 베이직 (VisualBasic) 윈도우와 2계층의 마그나엑스 (MagnaX) 기능 서버, 그리고 3계층의 코볼 (COBOL) 프로그램 간의 인터페이스가 모든 개별 단위 기능 개발에 있어 동일한 패턴을 사용한다는 점을 발견하고 이 패턴에 존재하는 추상 클래스를 개발하였으며, 이를 기반으로

해당 모듈을 수정하는 방법으로 개별 기능을 개발하고 있다. 개발 도구의 특성상 유전 관계를 적용하지 못하고, 복사-붙여넣기 (Copy & Paste) 기법으로 개별 기능을 개발하고 있으나, 각 계층별 인터페이스 패턴의 추상화를 통한 구조적 단순화를 실현하여 개발 생산성 향상은 물론 향후, 효과적인 유지 보수를 예상하고 있다.

AF 적용 성과를 정량적으로 비교, 분석하는 것은 매우 어려운 일이다. 동일한 프로젝트를 AF를 적용하지 않고 개발했을 때와 비교한다면 정확한 효과에 대한 분석이 가능하기 때문이다. 그러나, 각 사례는 공통적으로 AF를 적용함으로써 구조적 단순화와 재사용을 실현하였음을 다음의 사실에서 검증할 수 있었다. 대성산소의 경우, 템플릿 유형별로 평균 42.25본의 프로그램을 개발하여 높은 재사용율을 보이고 있으며, 캠브리지 필터는 루트 템플릿을 포함한 17개 컴포넌트로 374본의 프로그램을 구현하여 컴포넌트의 높은 재사용율을 보이고 있다. KCC는 원무, 진료, 일반관리 등 주요 병원 업무별 서브 시스템에 약 30개의 공통 템플릿과 기능별 템플릿을 설계하여 약 3,000 본의 기능을 구현하였다. SK의 경우, 약 400본의 온라인 회계 작업 기능을 단일 템플릿과 타 층과의 인터페이스 표준화를 통해 구조적 단순화를 실현하였다. 또한, 각 사별로 GUI 중심의 재사용에서 점차 비즈니스 로직으로 재사용의 대상을 발전시켜 나감으로써 보다 핵심적이고 효과적인 재사용 전략을 추진하고 있다.

AF 적용의 두번째 성과는 개발 생산성의 향상이다. 대성산소의 경우, 착수 1년간 339본의 프로그램을 구현하였으나, 2차년도에는 506본의 프로그램을 구현하여 약 50%의 개발 생산성 향상을 이루었다. 또한 SK의 경우, 배치 (Batch) 프로그램 600본을 포함한 총 1,000본의 프로그램을 전체 19개월 개발 일정 중 마지막 6개월간에 구현하여 AF 기반 개발이 분석, 계설 기간을 장기화시키나 상대적으로 구현 기간은 대폭 단축함을 보여 주고 있다.

V. 향후 연구과제 및 결론

정보시스템 개발에 있어 유연성, 규모성, 사용 적합성은 주요 평가기준이자 이상적 목표라 할 수 있다[Wegscheider, 1995]. 유연성은 다양한 집단의 요구를 만족시키면서 변화하는 요구사항에 대응할 수 있는 기능적 유연성과 전체 정책에 위반하지 않는 선에서 개인적 선호를 만족시킬 수 있는 선호적 유연성으로 구분된다. 일반적으로 기능적 유연성은 분석, 설계 단계에서 고려되나 선호적 유연성은 시간 제약으로 무시되는 것이 보통이다. 규모성은 기존의 서비스를 유지하면서 내부 구조가 물리적으로 복잡해져 갈 수 있는 확장성과 보다 단순한 요구사항으로 개인을 만족시킬 수 있도록 구조를 단순화하는 축소성을 의미한다. 마지막으로, 사용 적합성이란 적시성, 편리성, 완전성, 정확성, 친밀성 등 사용자가 사용하기에 적합한 기능을 제공함을 의미한다.

이런 이상적 목표를 실현하고자 제시된 AF의 설계와 개발은 1계층의 사용자 서비스 층을 대상으로 한 템플릿 중심으로 추진되었다. 비록, 2~3계층의 비즈니스 객체까지 개발 대상에 포함하지 못했으나 계층간 인터페이스의 일관성을 분산 객체 구조 정의와 적용으로 실현하였고, 객체지향 패러다임의 정보시스템 개발에 대한 현실적 적용을 실현했다고 볼 수 있다. 그러나, 현재까지 적용 과정에서 나타난 문제점을 요약하면 다음과 같다.

- 초기 개발 기간이 1.5배 정도 소요되어 프로젝트 관리에 위험 요소가 되었다. 물론, 프로젝트 후반기에 생산성 향상으로 전체 납기에는 문제가 없었으나 초기에 개발기간이 장기화된다는 것을 인식하고 이를 프로젝트 일정

관리에 반영해야 할 것이다.

- 유전 관계를 통해 템플릿을 재사용할 때, 무리한 코드 재사용으로 코드 자체가 너무 잘게 쪼개지는 경향이 발견되었다. 하나의 이벤트나 함수로 처리될 수 있는 것이 다수의 작은 이벤트나 함수로 구성되어 시스템 성능을 저하시키는 결과를 낳았다.
- 비즈니스 객체에 대한 패턴은 프로젝트가 종료되는 시점에서 패턴을 발견할 수 있어 이를 피드백 시킬 수 없었다.
- 다층 개발 구조에 따른 유지 보수의 문제점이 발견되어, 타 계층과의 인터페이스의 설계의 중요성이 크게 대두되었다. 즉, 오류가 발생했을 때, 어떤 층에서 발생하였는지 즉시 알 수 있는 방법이 없고, 또한, 오류 수정 후, 타 층과의 통합을 위한 절차가 정의되지 않아 유지 보수의 효과적 수행에 대한 우려가 있다.

이상과 같은 문제점을 인식하고 본 연구는 초기 단계의 결과를 토대로 다음과 같이 주제에 대한 연구를 추진하고 있다.

- 2~3계층의 비즈니스 객체 대상으로 한 패턴 정의와 설계를 추진하고 있다.
- 멀티 프로세서와 다층 클라이언트/서버 컴퓨팅의 활성화, 그리고 웹 기반 사용자 인터페이스 (WUI: Web User Interface)를 대비한 다양한 프레임웍 구조를 개발하고 있다.
- 유전 관계를 이용한 템플릿 재사용의 단점인 코드 조각화 (Fragmentation)를 해결하기 위해 Basset[1996]이 제시한 프레임 기술의 적용을 추진하고 있다.
- AF의 구성 요소와 템플릿의 관리 방법과 자동화에 대한 연구를 추진하고 있다.
- AF에 기반한 정보시스템 개발에 있어 재사용율에 대한 측정과 평가를 통해 방법론의 효과를 검증할 계획이다.
- 다층 개발 구조에 따른 유지 보수 방법과 절차에 대한 표준화 작업이 추진되고 있다.

〈참 고 문 헌〉

강문설, 김병기, "재사용 가능한 소프트웨어 부품의 검색 및 이해," *정보과학회 논문지*, 10권, 20호, 1993, pp. 1519-1559.

김태훈 외, "분산 객체지향 소프트웨어 개발 환경의 설계 및 구현," *정보과학논문지*, 3권 2호, 1997, pp. 139-151.

최성운 외, "객체지향 소프트웨어 개발 방법론 동향," *정보과학회지*, 14권 10호, 1996, pp. 30-46.

최은만, 김진석, "객체지향 재사용과 CASE," *정보과학회지*, 14권 10호, 1996, pp. 47-54.

Bassett, P., *Framing Software Reuse*, Prentice Hall, Upper Saddle River, NJ, 1996.

Brooks, J., *Mythical Man-Month*, Addison-Wesly, Reading, NY, 1965.

Deutsch, L. P., "Design Reuse and Frameworks in the Smalltalk-80 System," in Biggerstaff, T.J. and Perlis, A.J. editors, *Software Reusablility, Volumn II:Application and Experience*, pp.57-71. Addision Wesley, Reading, MA, 1989.

Forte, *Forte Object-Oriented Analysis and Design Training Manual*, Forte Software, Inc., 1994.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns*, Addison-Wesley, Reading, MA, 1995.

Jones, G. W., *Software Engineering*, John Wiley & Sons, New York, NY, 1990.

Johnson, R., "Reusing Object-Oriented Design, University of Illinois," *Technical Report UIUCDCS 91-1696*, 1991.

Johnson, R. and Foote, B., "Designing Reusable Classes," *Journal of Object-Oriented Programming*, Vol. 7, No. 2, June/July 1988, pp. 22-35.

Orfali, R., Harkey, D., and Edwards, J., *The Essential Distributed Objects Survival Guide*, Wiley, New York, NY, 1996.

Rumbaugh, J. et al., *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, NJ, 1991.

Shaw, B., *Project Review Presentation*, Marcam Corp., 1996.

Snyder, A., "Encapsulation and Inheritance in Object-Oriented Languages," *Proceedings of Object-Oriented Programming Systems, Languages, and Applications Conference*, Portland, OR, November, 1986, pp. 624-632.

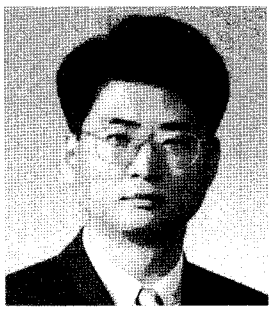
Shelton, R., "Business Objects - Modeling with Business Patterns," White Paper, *www.openeng.com/patterns.html*, May, 1996.

Taligent, "Leveraging Object-Oriented Frameworks," White Paper, *http:\www. taligent.com*, 1993.

Watterson, K., *Client/Server Technology for Managers*, Addison Wesley, Reading, MA, 1995.

Wegscheider, E., "Next-Generation Business Application Development Environments," White Paper, *http:\www.objectquest.com*, 1996.

◆ 저자소개 ◆



박 광 호 (Park, Kwangho)
저자는 한양대학교 경영학과를 졸업하고 University of Iowa에서 경영학 석사(MBA), 경영학박사(전공: MIS) 학위를 취득하였으며 현재, 한양대학교 경상대학 경영학부 부교수로 재직중이다. 삼성 SDS에서 AI실장, 소프트웨어연구팀장 등을 역임하며 전문가시스템, 객체지향기술 관련 프로젝트를 수행한 바 있다. 최근에는 객체지향 비즈니스 엔지니어링 프로젝트를 통해 프로세스의 표준화, 혁신, 이에 따른 정보시스템 개발을 추진 중이다. Naval Research Logistic, International Jounal of Production Research, International Journal of Manufacturing System Design, 데이터베이스저널, 정보시스템 연구 등에 논문을 게재하였다.