정보시스템 개발에 있어서 애플리케이션 프레임웍(Application Framework)의 재사용에 대한 연구

박광호, 이 인호
(한양대학교)

0 서론

1.1 연구의 배경

21세기를 목전에 둔 현대 사회는 빠른 속도로 정보화 사회(information society)로 옮겨가고 있으며 정보화사회의 도래에 따라 컴퓨터 보급이 증가하고 있으며 자연적으로 소프트웨어에 대한 수요도 급속히 늘어나고 있다. 현대인들은 더욱 새롭고 다양한 분야의 컴퓨터를 사용하기 바라며 교육, 의료, 생산, 오락, 출판 등 각 분야에서 효과적으로 이용 가능한 소프트웨어를 원하고 있다. 컴퓨터 분야의 기술 발전은 과거 보다 더 크고 복잡한 소프트웨어를 개발할 수 있게 하고, 소프트웨어의 발전은 더 많은 소프트웨어 요구를 창출한다. 1965년부터 20여년간 소프트웨어의 수요는 약 100배 증가하였고 매년 12.6%의 증가율을 나타내고 있다. Bohen의 통계에 의하면 1985년에는 전 세계가 소프트웨어 개발에 사용한 비용이 약 112조원, 1995년에는 348조원이었다고 한다. 하지만 소프트웨어를 생산하는 공급 측면의 발전은 이러한 수요의 증가를 따르지 못하고 있다. 미국에서의 실증조사에 의하면 소프트웨어 생산성 증가는 같은 기간에 2배를 넘지 못하고 있고 소프트웨어 생산 인력도 매년 4%정도의 증가에 그치고 있다. 또다른 조사에서는 지난 20여년간 소프트웨어에 대한 수요는 100배 이상 증가하였으며 앞으로 더욱 그 수요가 늘어날 것으로 예상되지만 같은 기간동안 소프트웨어 개발자들의 생산성은 단지 1.8배 정도의 향상에 불과하였고 개발자들의 공급도 10배 정도로 늘어난 것에 그쳤다고 한다.[1] 이상과 같은 소프트웨어의 낮은 생산성에 대한 심각한 인식이 바로 소프트웨어의 위기(Software crisis)이다. 또한 생산성 문제 이외에도 개발 비용 및 기간의 증가 등 많은 문제점을 안고 있다.

지금까지 등장한 개발도구와 방법론들이 부분적으로 소프트웨어의 품질 및 생산성 향상에 기여하였지만 갈수록 더욱 복잡하고 큰 규모의 소프트웨어를 필요로 하는 사회적 수요에 대해서 소프트웨어 개발 생산성을 혁신적으로 향상시키지는 못하였다. 이런 상황에서 소프트웨어의 위기에 대한 해결방안의 하나로서 주요하게 제기되고 있는 소프트웨어 재사용(Software Reuse) 역시 소프트웨어 개발의 생산성과 품질을 증대 시키기 위한 노력의 일환이다. 많은 현장조사 보고에 따르면 기존의 소프트웨어들간의 중복되는 부분이 상당하며 따라서 기존 소프트웨어를 충분히 재사용하고 나

[1] 이주헌, *실용 소프트웨어 공학론*, 법영사, 1993

머지 부분만을 집중적으로 개발하는 것이 효과적인 것을 알수 있다. 소프트웨어의 재사용에 대한 연구는 1969 년 소프트웨어의 부품화를 위해 상호 교환 가능한 표준화된 소스코드의 사용을 제기한 M.Mcllroy 의 주장에서 비롯되었다[2] 1970 년대 초반까지도 재사용에 대한 연구는 충분히 이루어지지 못하였지만 1970 년대 중반부터 소프트웨어의 위기에 대한 인식이 일반적으로 대두되면서 Raytheon, Toshiba 등의 기업에서 소프트웨어 재사용에 관한 연구가 본격적으로 진행되었으며 80 년 이후부터는 소프트웨어 개발에 있어 다양한 분야의 전문가들이 재사용의 개념, 인식, 방법론 등에 대해 집중적으로 연구되고 있다.

1.2 연구의 목적과 범위

Power Builder, Visual Basic, Delphi, Forte 등속과 같은 객체지향적 비주얼한 개발도구들이 정보시스템의 개발에서 이미 주요 도구들로 사용되고 있지만 객체지향의 잇점을 충분히 살리지 못하고 있으며 그 도구들이 나름대로 지원하거나 포함하고 있는 재사용 방안들에 대해서도 인식이 부족하고 충분히 활용되지 못하고 있는 실정이다. 대부분의 재사용이 GUI(Graphi User Interface)에 한정되고 있으며 디자인 및 코드에 대한 재사용은 종래의 Copy and Paste 방법 등을 답습하는데 그치고 있다. 이에 대해 적극적으로 클라이언트/서버 구조의 정보시스템 개발에 있어서 소프트웨어 재사용을 모색하기 위해 실제 사례로서 모 비영리단체의 직원을 대상으로 하는 급여관리시스템을 선정하였으며 Microsoft 사의 SQL Server 6.5 와 Visual Basic 5.0 을 주요 개발도구로서 이용하였다. 비영리단체의 성격과 150 여명이라는 비교적 소수의 직원들을 대상으로 한 것이기 때문에 일반 기업에 비해서 상대적으로 시스템이 간략한 구조를 갖기에 분석이 더욱 용이하다고 판단되었다. 애플리케이션 프레임웍을 우선 구축한 후 그것을 기반으로 하였다. 애플리케이션 프레임웍은 소프트웨어에 대한 전체적인 틀을 제공하지만 동시에 재사용을 위한 구조적 틀이기도 하다. 즉 클라이언트/서버 환경의 정보시스템의 개발에 있어서 재사용을 극대화하고 시스템의 구조적 단순화를 실현하여 유지보수를 용이하고자 하는 것이다. 현재 정보시스템 개발에서 일반적으로 이용되고 있는 비주얼한 개발도구의 하나인 Visual Basic 을 대상으로 개발환경을 조명하고 소프트웨어 재사용을 위해 지원되는 기능들을 분석하며 애플리케이션 프레임웍구축과 기반하의 구체적인 개발방법들을 제시하는데 본 연구의 주된 목적을 두었다.

II. 이론적 배경

2.1 소프트웨어 개발의 생산성과 품질 향상을 위한 제연구

소프트웨어의 위기에 대한 해결책은 신뢰도 높은 소프트웨어의 생산성향상에 있다. CASE 도구의 활

---

[2] M.Mcllroy, "Mass Produced Software Components", Proc. NATO Conf. of Software Eng. Petrocelli/Chater, NewYork, pp 88-98, 1969

용, 역공학, 소프트웨어 재사용 등과 같은 다양한 방안들이 소프트웨어의 위기를 해결하기 위해 제시되었다. 그 중에서 본 논문은 소프트웨어의 재사용에 주목하였고 소프트웨어의 재사용이란 시스템을 개발하는데 이미 사용된 디자인, 원시 코드, 문서 등 다양한 형태의 정보와 지식을 체계적인 방법을 이용하여 유사한 영역의 다른 시스템을 구축하는데 적용하는 기술로서 정의 내릴 수 있다.[3]

객체지향에서 재사용 단위로는 애플리케이션, 클래스 명세서, 클래스, 알고리즘, 분석/디자인 모델, 프레임웍, 애플리케이션 프레임웍, 패턴 등이 있다.[4] 본 연구는 애플리케이션 프레임웍을 기반으로 하는 정보시스템 구축을 목적으로 하며 프레임웍, 패턴 등이 주요 대상이다. 프레임웍은 상호작용하는 객체들로 구성되며 불완전하며 추상화 된 형태를 갖고 있다. 따라서 상황에 따라 필요한 컴포넌트와 서브클래스를 추가하여 완성될 수 있는 것이다. 패턴은 Gamma 등이 명명한 디자인 패턴으로 불리며 소프트웨어의 디자인에서 반복적으로 일어나는 문제들을 해결하기 위한 재사용 단위이다.[5] 다시 말해 패턴은 특정 환경과 조건을 가진 문제에 대한 일반적인 해결방안(Solution)이며 효과적인 객체지향 아키텍처는 일련의 패턴들을 포함하고 있다고 하였다.[6]

프레임웍의 일종인 애플리케이션 프레임웍은 클라이언트/서버 구조의 정보시스템 개발에 있어서 재사용의 극대화와 유지보수의 용이를 목적으로 한다. 클라이언트/서버 구조의 정보시스템 개발에서 기능별 코드 구성비율에 대한 Shaw 의 분석에 의하면[7] 기본 기능이 56%, 관리/지원 기능이 23%, 순수한 업무처리 기능이 21%를 차지하는 것으로 나타났다. 즉 전체 코드의 약 79%가 데이타베이스 처리, GUI 처리, 사용자 보안 등과 같은 비 업무처리 기능을 구현하기 위하여 작성되었으며 따라서 이 부분에 대한 노력을 최소화하고 순수 업무처리 기능 부분에 노력을 집중하는 것이 효과적인 것이다. 이 부분들은 업무와 독립적인 성격을 갖고 있으므로 재사용 부품으로 개발하여 개발 프로젝트마다 이를 이용함으로써 개발에서의 비용절감과 신속한 개발을 꾀하고자 하는 것이다. 기본 기능과 관리/지원기능을 정형화 하여 단순한 구조를 갖는 프레임웍들을 정의하고 개발하며 유사한 업무처리의 정형화를 통해 몇 개의 프레임웍 정의, 개발을 통해 코드 수준의 재사용에서 더 나아가 구조수준(Architecture Level)의 디자인 재사용까지 가능하게 하는 것이다.

2.2 소프트웨어 재사용 현황

객체지향 방법론과 프로그래밍을 중심으로 연구되고 있는 소프트웨어 부품의 재구성을 통한 소프트웨어 재사용은 국외에서는 현재 본격화되는 단계에 접어 들고 있다. 특히 소프트웨어 부품의 재구성 시스템은 Ada, Smalltalk 와 같은 특정 언어에서 상당한 생산성 향상이 보고 되고 있으며 코드 생성

[3] T.Biggerstaff & C. Richter, "Reusability Frame work, Assessment, and Directions", IEEE Software Vol. 4, No. 2., 1987

[4] Adele Goldberg & Kenneth S. Rubin, *Succeeding with Objects*, Addison Wesley, 1996

[5] Fowler, *Analysis Patterns,* Addison Wesley, 1996

[6] Grady Booch, *Object Solutions*, Addison Wesley, 1996

[7] Shaw, B., *Project Review Presentation*, Marcam Corp., 1996

(Code Generation)에 의한 재사용 기술은 이미 상품화되어 있다.

2.3 객체지향 패러다임과 소프트웨어 재사용

2.3.1 객체지향 패러다임

객체지향에서는 객체가 기본 단위가 되며 객체는 속성과 연산에 대한 정의인 클래스로부터 생성된다. 하나의 객체가 포함하고 있는 속성은 객체의 외부로부터 보호되며 그림에서처럼 객체자신의 연산(Method)에 의해서만 액세스가 가능하다. 외부 객체와의 관계는 인터페이스를 통하여 서비스를 요청하고 제공하는 것으로 이루어진다. 이러한 객체지향에서의 캡슐화는 중요한 의미를 갖고 있다. 내부 정보의 보호는 객체의 외부에 대하여 객체가 가지는 복잡성을 단순한 외양 즉 인터페이스로 변환하는 방식으로 이루어지는데 이를 통해 프로그램의 복잡성을 감소시키는 것이다.

한편 소프트웨어 표준화에 대한 노력이 몇 개의 국제 표준 기구와 대기업들의 연합체를 중심으로 활발히 전개되고 있다. 소프트웨어의 구조, 개발 방법, 사용자 인터페이스, 통신 프로토콜 등의 표준화를 제정한 후, 시스템 개발 시 이를 채택 적용하여 프로그램간의 호환 과 상호 연동성을 높인다는 전략이다. 이런 표준화 과정에서 주목할 점은 대부분의 표준들이 객체지향 방식을 채택하고 있다는 것이다.

2.3.2 유전관계와 구성관계

유전관계는 서로 다른 클래스들이 유사한 속성과 연산을 공유하기 위한 추상방법으로 일반화(Generalization), 특수화(Specialization) 관계로 불리기도 한다. 유전관계는 어떤 클래스와 그 클래스의 하나 이상의 세련된 버전(Refined Version)사이의 관계이다. 여기서 세련된 클래스를 슈퍼 클래스, 슈퍼 클래스로부터 세련된 클래스를 서브클래스라고 한다. 일반화는 슈퍼클래스가 서브클래스의 특성을 일반화한다는 사실에서, 특수화는 서브클래스가 슈퍼클래스를 세련시켜 특수화한다는 사실에서 각각 그 개념이 만들어 졌다.

구성관계(Aggregation)는 클래스간의 특수 형태의 관계로 구성요소가 되는 클래스들과 이들로 조합된 클래스 사이의 관계로 부품-제품(part-whole) 관계 또는 부품(part-of, consist-of) 관계로 불리기도 한다. 구성관계의 중요한 특성은 전이성(Transitivity)이다. 예를 들어 A가 B의 부품이고, B가 C의 부품이면 A는 C의 부품이 된다. 또한, 구성관계는 비대칭성을 갖는다. 즉 A가 B의 부품이면 B는 A의 부품이 될 수 없다. 구성관계는 여러 부속 객체들이 조립되어 하나의 객체로 구성되는 경우에 대한 것이다.

2.3.3 컴포넌트 지향 프로그래밍(COP: Component-Oriented Programming)

객체지향 기술을 이용한 최근의 많은 대규모 프로젝트들이 실패하였다. 그것은 객체지향 프로그래

밍(OOP:Object-Oriented Programming)이 알려진 바와는 달리 진정한 의미에서의 재사용을 목적으로 하는 소프트웨어를 생산하지 못하며, 대규모 프로젝트에는 적합하지 않고 캡슐화도 실패하기 때문이었다. 프로그램이 대형화될수록 프로그램의 복잡성(Complexity) 방어와 안정성(Stability)의 확보가 중요한데 OOP 에서는 그림에서처럼 효과적이지 못하다. 복잡성은 프로그램의 대형화에 따라 급속히 악화되며 개발 및 유지보수를 극도로 어렵게 하며 안정성은 프로그램 일부에 대한 변경이 프로그램 전체에 미치는 영향을 최소화함을 통해서 이루어진다.

COP 를 OOP 의 진화로 보는 Marko Krajnc 은 빌딩 블록(Building Block)의 기본 단위로서 객체 대신 객체들의 소규모 집합이며 인터페이스를 가진 컴포넌트를 제시한다. 컴포넌트 내부의 객체들은 제한없는 커뮤니케이션이 가능하지만 컴포넌트 외부의 객체와는 반드시 컴포넌트 인터페이스를 통해서만이 가능하다.

현재 소프트웨어 시장에서는 Microsoft 사의 COM(OLE Component Object Model)이 객체지향 아키텍처에 기반하여 주도적으로 구현되고 있으며 OLE 가 객체기반 운영체제로의 전환을 꾀하는 Microsoft 사의 전략적 토대가 되고 있다. OLE 는 Microsoft 사가 제시한, 컴포넌트 기반의 소프트웨어 작업을 가능하게 하는 객체지향 기술의 구체적 형태이다. OLE 는 현재 Microsoft 사의 Windows 95/NT 운영체제에서 뿐만 아니라 APPLE 사의 Macintosh 플랫폼에서도 이용가능하며 COM 계열의 컴포넌트들이 모든 UNIX, VMS, VMS 운영체제의 모든 버전에서도 사용할 수 있게 되면서 많은 독립적인 소프트웨어 밴더들과 시스템 통합사업자, 정보시스템 담당자들이 OLE 에 기반한 솔루션들을 제공하기 시작하였다.[8] [9] [10]

2.3.4 프레임웍(Framework)에 대한 연구

프레임웍은 유사한 소프트웨어 구조를 나타내는 다수의 프로그램이 있을 때, 이를 보편적으로 적용할 수 있는 구조로 개발한 것이라고 정의 내릴 수 있다.[11] 또한 Ralph E. Johnson 는 프레임웍은 특정 문제들의 솔루션에 대한 추상적인 디자인인 클래스들의 집합이라고 했고 그것은 가장 일반적으로 받아들여지는 정의이다.[12]

프레임웍은 다양한 프로그램에서 적용할 수 있는 수평적 성격의 애플리케이션 프레임웍과 증권거래 시스템, 회계정보시스템 등 특정한 도메인에 한정하여 적용가능한 수직적 성격의 도메인 프레임웍

[8] Marko Krajnc, “Why Component-Oriented Programming ? ”, 오류! 참조 원본을 찾을 수 없습니다., 1997

[9] Marko Krajnc, “What is Component-Oriented Programming ? ”, 오류! 참조 원본을 찾을 수 없습니다., 1997

[10] Microsoft, “The Microsoft Object Technology Strategy : Component Software”, 오류! 참조 원본을 찾을 수 없습니다., 1996

[11] 최은만, 김진석, “객체지향 재사용과 CASE”, 정보과학회지, 14 권 10 호, 1996

[12] Ralph E. Johnson, “Designing Reusable Classes”, The Journal of Object-Oriented Programming, Vol.1, No.2, 1988

(Domain Framework), 그리고 파일처리, 디바이스 드라이버 등과 같은 시스템 수준의 지원 프레임웍(Support Framework)으로 구분할 수 있다. 또한 프레임웍으로부터 새로운 클래스를 유전받으며 멤버함수를 오버라이딩하여 이용하는 유전관계 프레임웍(Inheritance-Focused Framework)과 객체를 조합하여 이용하는 구성관계 프레임웍(Composition-Focused Framework)으로 나눌 수도 있다. 유전관계 프레임웍은 개발자가 상당한 양의 코드를 추가로 작성하여야 하기 때문에 프레임웍의 사용이 구성관계 프레임웍에 비해서 어렵다. 하지만 구성관계 프레임웍 또한 일반적으로 사용하기 쉬운 편이지만 그 사용이 제한적이라는 단점을 가진다. 그래서 사용하기 편리하면서도 기능을 확장할 수 있는 프레임웍 개발은 구성관계 프레임웍을 기반으로 유전관계 프레임웍의 성격을 부가하는 것이 적절하다.

III. 정보시스템 개발 환경의 변화와 재사용 전략

정보시스템은 현대 기업의 비즈니스 수행에 결정적인 영향을 미치고 있다. 우수한 정보시스템은 기업에게 보다 많은 비즈니스 기회를 제공하고 보다 신속하게 비즈니스 환경 변화에 적응할 수 있도록 한다. 하지만 하드웨어, 소프트웨어, 클라이언트/서버 기술, 분산 컴퓨팅, 그리고 보다 우수한 새로운 정보 기술들의 출현에도 불구하고 정보시스템 담당자들은 시스템의 복잡성, 비유연성(inflexibility), 저수준의 퍼포먼스 등의 문제에 직면하고 있다.

이러한 문제의 해결을 위한 방향들은 특히 소프트웨어의 재사용과 관련하여 비즈니스 모델링과 컴포넌트 기반의 애플리케이션 개발로서 제시되고 있다

3.1 3-Tier 아키텍처 클라이언트/서버

최근 몇 년 사이에 정보시스템은 메인프레임보다 유연하고 애플리케이션 사용이 편리한 클라이언트/서버 환경으로 옮겨가고 있다. 메인프레임에서는 가능하지 않았던 우수한 GUI 를 구현할 수 있었지만 2-Tier 클라이언트/서버는 상호충돌하고 중복되는 비즈니스 로직과 데이타들이 애플리케이션 전반에 흩어져 있었다. 그것은 아키텍처상의 취약성이었으며 3-Tier 아키텍처가 이러한 문제에 대한 해결책으로 제시되었다.

3-Tier 클라이언트/서버는 애플리케이션 분산 전략이며 종래의 2-Tier 아키텍처와는 미들(Middle)에서 구별된다. 비즈니스 로직을 클라이언트에서 분리하여 중간층(Middle-Tier)인 논리적 서버(Logical Server)에 배치시키는 것이다. 3-Tier 아키텍처에서는 많은 애플리케이션들이 중간층의 컴포넌트들을 공유하며 그러한 컴포넌트들을 통하여 데이타베이스가 존재하는 최종 서버(Back-End Server)에 보다 효율적이며 안전하게 액세스한다.

3.2 컴포넌트를 이용한 정보시스템 개발

객체지향의 새로운 형태인 컴포넌트 기반 프로그래밍은 정보시스템 개발과 유지보수에 보다 많은 혜택을 주고 있다. 외부 밴더에 의해 공급되거나 자체 개발한 소프트웨어 컴포넌트들을 조합하여 완

성된 정보시스템을 구현하는 방식이다. 이러한 컴포넌트들의 사용은 정보시스템 담당자들이 보다 유연하고 높은 품질의 시스템 구축을 용이하게 하며 시스템 프로그래밍, 실행, 유지보수 등을 위한 많은 자원과 시간을 절감할 수 있게 한다.

컴포넌트를 이용한 정보시스템 개발은 3-Tier 아키텍처 클라이언트/서버 구조에서 본격적으로 가능하다. 각 계층에 위치한 컴포넌트들이 서로 서비스를 요청하거나 요청받은 서비스를 제공하는 상호협력으로 애플리케이션의 목적을 달성하는 것이다. 3-Tier 는 애플리케이션 사용자에게 시각적인 인터페이스를 통하여 필요한 정보와 데이터의 입출력을 제공하는 User Service(1-Tier)층과 데이터베이스의 저장, 관리 등이 수행디는 Data Service(3-Tier)층, 그리고 User Service, Data Service 사이에서 비즈니스 로직을 통하여 서비스 및 데이터를 중개처리하는 Business Service(3-Tier)층으로 구성된다.

3.3 비즈니스 모델링(Business Modeling)을 통한 비즈니스 프레임웍 개발

세계적인 기업들의 정보시스템들은 비즈니스 객체를 활용하여 데이타 및 데이타처리의 무결성(Data Integrity), BPR, 메인프레임용 애플리케이션과 호스트 기반(Host-based)의 데이타 베이스의 클라이언트/서버 환경으로의 전환 등과 같은 문제를 해결하고자 하고 있다.

비즈니스 객체는 특정한 비즈니스 영역에서 운영되는 절차, 정보, 정책 및 규칙들을 표현하기 위해 정의되는 블랙박스이다. 그것은 코드로 작성되어 구현되며 동시에 특정한 비즈니스에 대한 추상화 및 디자인으로도 존재한다.

클라이언트/서버 환경의 정보시스템 구축에서 비즈니스 객체가 개발의 핵심이 될만큼 중요한 것은 첫째 데이타 및 데이타처리의 무결성 문제를 해결하는 강력한 도구이며 둘째, 비즈니스 객체 또는 비즈니스 컴포넌트들이 3-Tier 클라이언트/서버 아키텍처의 중간계층(Middle-Tier)에 배치될 이상적인 모듈의 단위가 되며 동시에 다른 애플리케이션에서의 재사용 또는 복수의 애플리케이션간의 공유의 단위가 되기 때문이다.

R. E. Shelton 은 비즈니스에 대한 추상화 작업인 비즈니스 모델링에서 비즈니스 객체에 대한 추상화를 먼저 제시하고 비즈니스 객체를 비즈니스 엔티티 객체(Business Entity Object), 비즈니스 프로세스 객체(Business Process Object), 비즈니스 이벤트 객체(Business Event Object)로 구분하였다. 비즈니스 엔티티 객체는 개념, 역할, 담당자, 장소, 사물 등을 나타내는 비즈니스 객체로서 고객, 주문, 제품, 보험 정책, 자동차 등을 예로 들 수 있다. 비즈니스 프로세스 객체는 하나의 비즈니스 프로세스를 구성하는 활동들을 나타내며 주문처리, 세금 계산서 처리, 입원 수속, 납품 등을 예로 들 수 있다. 마지막 비즈니스 이벤트 객체는 원인, 발생, 시간의 경과 등을 나타낸다. 비즈니스 엔티티 객체사이의 활동을 유발시키거나 활동의 결과로 나타나며 회계 마감, 안전 재고 부족, 작업 지시 취소, 주문 취소, 자재 발주 취소, 수금 만기, 접수 완료, 조립 완료, 납품 완료 등을 예로 들 수 이다.

비즈니스 모델링의 다음 단계는 비즈니스 패턴이다. BPR(Business Process Reengineering)에 대한 연구를 통해서 Hammer 와 Champy 는 모든 비즈니스들은 근본적으로 유사한 비즈니스 프로세스들로 구성

되어 있으며 다만 다른 방식으로 수행할 뿐이라고 하였다.[13] 이러한 점에서 공통적인 성격을 지닌 비즈니스 객체의 활용 가능성이 커지고 있다. R. E. Shelton 은 비즈니스 프로세스들 사이의 유사성을 BPP(Business Process Patterns)라 하였고 BPP 에는 행동 패턴(Behavioral Pattern), 구조적 패턴(Structural Pattern), 의미 패턴(Semantic Pattern)으로 구분하였다.

행동 패턴은 엔티티 객체들과 이들간의 상호작용들의 반복적인 집합, 주문처리와 같은 일반적인 버전의 비즈니스 프로세스에 대한 템플릿(Template)이다. 즉, 하나의 패턴으로 많은 비즈니스 프로세스를 설명할 수 있다. 구조적 패턴은 특수화 관계와 같이 상호 밀접하게 연관된 사물이나 개념들의 전체 집합을 의미한다. 마지막으로 의미 패턴은 엔티티 객체 유형과 정의, 의미에 대한 제약조건 등을 포착하는 2 개 객체간의 관계들의 반복적인 집합이다. 예를 들어, 항공기 운행편-구분-좌석의 수송 능력 패턴은 기차, 선박, 병원의 병실관리, 공장 조립 라인 등과 개념적 유사성을 가지고 있다.

비즈니스 모델링에 있어서 비즈니스 패턴은 다음 두 가지 점에서 중요하다. 첫째, 패턴은 모든 비즈니스 영역에 존재한다는 것이다. 많은 모델링 전문가들이 패턴들을 발견하고 있으며 패턴에 대한 포착을 결정적으로 가로막는 것은 잘못된 표기(pool naming), 모델링 대상에 대한 명확한 정의의 부재, 불충분한 추상화 작업들이다. 둘째, 비즈니스 패턴은 비즈니스 모델링을 크게 단순화한다. 새로운 모델을 추상화하는 노력 대신 존재하는 비즈니스 패턴을 발견하고 이해하는 것으로 비즈니스 모델링을 수행할 수 있는 것이다.[14] [15] 서브시스템에 대한 디자인으로서 프레임웍은 추상클래스, 칸크리트 클래스(Concret Class), 클래스들간의 인터페이스들로 구성되며 동시에 여러 개의 패턴을 포함하고 있다. 그 중 애플리케이션 프레임웍은 개발하고자 하는 애플리케이션의 GUI 디자인에 대한 패턴들인 추상, 칸크리트 클래스들의 집합이며 GUI windows 의 구조, 데이타 디스플레이, 데이타 입력 화면, 컨트롤 버튼, 메뉴들을 제공한다. 애플리케이션 프레임웍은 사용자의 필요에 따라 수정하여 작성할 수 있는 불완전한 코드와 구조를 함께 제공하기 때문에 코드 및 디자인수준의 재사용을 가능하게 하며 전체 애플리케이션의 구조적 단순화를 구현한다. 또한 애플리케이션 프레임웍은 비즈니스 객체, 컴포넌트들과 같은 블랙박스로서의 재사용단위를 포함하면서 그 스스로는 내부 구조가 공개되고 수정 가능한 화이트박스로서의 재사용단위가 된다.

비즈니스 객체에 기반한 애플리케이션 아키텍처는 복잡하고 역동적인 비즈니스 환경과 조건을 보다 잘 충족시키는 정보시스템의 구축에 기여하며 비즈니스 객체들은 직접적으로 비즈니스 모델을 구현하는 컴포넌트로 정의되어 재사용 및 공유가 이루어지게 된다.

IV. 애플리케이션 프레임웍을 이용한 정보시스템 개발

기존의 애플리케이션 프레임웍에 대한 연구 및 활용은 UI(User Interface)에 집중되어 있으며 분산 컴

[13] Hammer & Champy, *Reengineering the Corporation*, HarperBusiness, 1993

[14] Robert E. Shelton, “Business Object Frameworks and Patterns”, Data Management Review, May 1995

[15] Robert E. Shelton, “Business Objects Modeling with Business Patterns”, Data Management Review, May 1996

퓨팅 환경인 3-Tier 클라이언트/서버 구조 및 개발도구에 대한 고려가 부족하였다. 따라서 본 논문에서의 애플리케이션 프레임웍(이하 AF)은 개발도구가 지원하는 모든 재사용방안들을 적극적으로 고려하고 반영하였으며, 기존의 코드 수준에 머무르는 컴포넌트의 재사용을 AF 의 구조속에 포함시켜서 디자인 수준의 재사용을 실현하고자 하였다.

사용자 서비스층인 1-Tier 에서의 애플리케이션 개발은 주로 UI.와 관련되어 있다. UI 의 성공적인 구축은 프로젝트 성공에 있어 중요한 사항이 되며 일관성이 준수될 때 시스템에 대한 이해와 불필요한 유지보수 노력을 절감할 수 있으며 AF 의 개발과 이의 사용을 통하여 UI 의 일관성을 강제적으로 준수시켜 관리적으로 UI 의 일관성을 유지하려는 노력을 줄일 수 있다.

4.1 AF 의 목적

AF 기반 클라이언트/서버 환경의 정보시스템 개발의 목적은 재사용 극대화를 통하여 생산성을 높이고, 시스템의 구조적 단순화를 실현하여 시스템의 이해를 제고하여 유지보수 노력을 절감하기 위해서이다.

생산성 향상은 시스템의 유사 또는 중복되는 부분에 대한 불필요한 노력을 감소시키며 검증된 AF 의 재사용을 통해 시스템의 품질수준을 확보함으로써 이루어 진다. 또한 구조적 단순화는 코드 수준의 복잡성 감소 뿐만 아니라 디자인수준(Design Level)의 중복되는 복잡성을 제거함으로써 실현된다. 이러한 점에서 4GL, 코드생성기(Code generator), 클래스 라이브러리 등과 같은 다른 재사용기법과 결정적으로 구별되는 것이다. 4GL, 코드생성기는 프로시저 프로그래밍에 기반하여 디자인수준의 재사용이 쉽지 않으며 클래스 라이브러리는 코드재사용에 효과적이지만 저수준의 기능을 제공하는데 그치는 한계를 갖는다. 하지만 AF 는 디자인수준에서의 유연성을 갖고 있기 때문에 시스템 구조에 대한 수정과 기능확장이 용이한 효과적인 메커니즘을 제공한다. 따라서 시스템의 오류를 보다 빨리 찾을 수 있고 수정한 결과가 다른 부분에 주는 영향이 최소화 되므로 유지보수 노력을 절감할 수 있다.

재사용기술의 적용은 대부분 반복적이거나 계속되는 다수의 프로젝트에서 이루어져 왔다. 그것은 재사용 컴포넌트, 프레임웍 등과 같은 재사용부품의 개발이 비용 및 시간이 많이 소모되어 하나의 프로젝트만을 위한 개발은 비경제적이기 때문이다. 하지만 프로젝트의 규모가 거대화 되고 복잡해질수록 재사용 부품들의 개발을 통한 긍정적 효과가 더욱 커지고 있으며 개발이 거듭될수록 재사용 부품의 개발기술 및 활용도가 확대될 것이다.

실제로 AF 의 개발을 통하여 정보시스템을 구축중인 기업들의 사례를 보면, 초기 개발 기간이 그렇지 않았을 때보다 1.5 배 정도 소요되어 프로젝트 관리에 위험요소가 되었다고 한다. 물론 프로젝트 후반기의 생산성 향상으로 전체 프로젝트 기간에는 문제가 되지 않지만 초기의 개발기간이 장기화된다는 것을 고려하여 전체 프로젝트 일정에 반영해야 한다고 하였다.

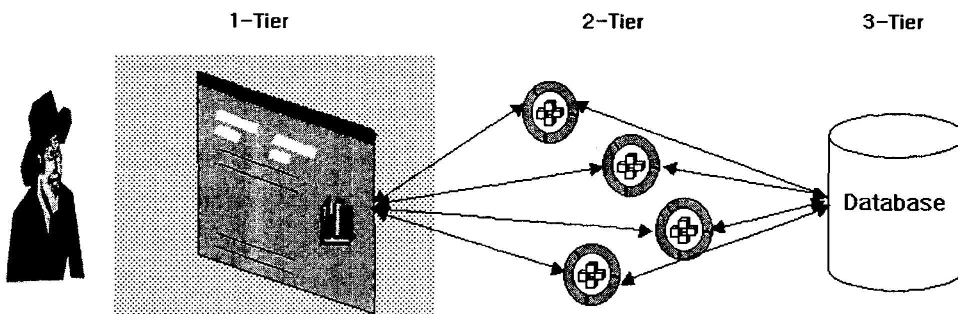
분산 컴퓨팅 환경에서 재사용은 다수의 프로젝트에서의 사용을 말하는 재사용과 다수의 애플리케이션에서 서비스를 함께 이용하는 공유로 나누어 볼 수 있다. 전자의 재사용은 디자인 단계에서 이루어지며 후자의 공유는 디자인 단계에서 코드를 통해 명시되지만 실제 공유는 애플리케이션의 실행시에 이루어 진다는 차이점을 갖고 있다. 또한 1 회의 프로젝트에서의 재사용과 다수 프로젝트에서의 재사

용으로 구분할 수도 있다. 재사용 부품의 개발에 필요한 노력을 감안할 때 경제성 측면에서 1 회 프로젝트에서의 재사용은 프로젝트가 일정 규모 이상이며 공통적인 사용 빈도가 많을 때 고려될 수 있다. 물론 이후의 프로젝트에서도 수정 등을 통하여 재사용이 가능하겠지만 현실적으로 프로젝트의 수행기간, 비용, 개발자들의 인식 부족 등 여러가지 이유로 소홀히 되고 있다. 그러나 1 회의 프로젝트에 있어서도 그 프로젝트의 규모가 클수록 AF 의 재사용 효과를 충분히 거둘수 있다. 그 효과는 프로젝트 후반기의 능률 향상과 UI 및 각종 변수, 프로시저, 함수의 호출 등의 일관성 유지로 유지보수 노력을 절감하는 것으로 나타나며 또한 이후의 프로젝트 추진시에도 이전의 AF 를 재사용할 수는 없더라도 개발자의 이해 증진 및 인식의 확산, 이전 AF 의 수정후 재사용 등으로 재사용 효과가 증가될 것이다.

결과적으로 이 논문에서의 AF 는 도메인에 독립적인 성격을 갖는다는 점에서 다수의 프로젝트 및 대규모 프로젝트에서의 재사용을 목적으로 개발된다. 또한 다수의 애플리케이션의 공유의 대상인 컴포넌트는 이용할 컴포넌트 및 이용방법 등에 대한 명시를 하는 것으로 AF 설계상에 포함되는 것이다.

4.2 AF 의 적용 환경

3-Tier 아키텍처의 특징은 네트웍상에 산재해 있는 객체들간의 협력작업으로 애플리케이션이 구현되며 애플리케이션의 목적을 달성하게 된다. AF 는 도메인 독립적인 성격을 갖는 프레임웍으로서 다양한 프로그램에서의 적용을 목적으로 하는 일반적 성격의 프레임웍이다. 따라서 공통적인 UI 와 컴포넌트들에 대한 디자인에 대한 추상으로서 사용자의 입출력을 처리하는 사용자 서비스(User Service)층인 1-Tier 에 주로 적용된다.



<그림 1> 3-Tier 아키텍처환경에서의 AF 의 적용영역

비즈니스 서비스(Business Service)층인 2-Tier 는 비즈니스 로직을 구현하는 컴포넌트들이 배치되며 다양한 비즈니스 프로세스의 공통적인 로직을 구현하는 비즈니스 객체에 대한 연구는 아직 본격화되지 않고 있다. 따라서 이번 논문에서의 AF 의 구현은 주로 1-Tier 에서의 재사용을 목적으로 하고 있다.

4.3 AF 설계 방법과 논리적 타당성

AF 의 대상은 첫째, 공통적인 것, 둘째, 도메인에 독립적인 것, 셋째, 자주 반복되는 것 등이다. AF 의 발견 및 선정은 시스템 요구분석 결과를 토대로 휴리스틱하게 이루어지며 유전관계 및 구성관계를 이용하여 개발되고 재사용된다.

DNA 는 생명체의 기본 구조로서 복잡한 생명체의 형성에 관한 정보를 담고 있다. DNA 정보를 바탕으로 최초의 세포를 형성하고 그 세포는 자신의 동일한 속성과 활동을 이어받는 세포들로 분열해 가고 결국 하나의 생명체를 구성하게 된다. 따라서 생명체의 신비를 이해하기 위해 우수한 조직과 세포들을 연구하는 대신 DNA 정보의 해석에 노력하는 것이다.

또한 카오스 이론에서도 비슷한 맥락을 찾을 수 있다. 카오스 이론은 1963 년 미국의 기상학자인 Edward Lorenz 에 제시되었으며 불규칙한 현상속에서도 질서가 있다는 것으로 요약될 수 있다. 단순한 질서를 발견해냄으로써 불규칙한 복잡성을 해소하여 이해하기 힘들거나 이해할 수 없는 현상을 이해 가능하게 하는 것이다.

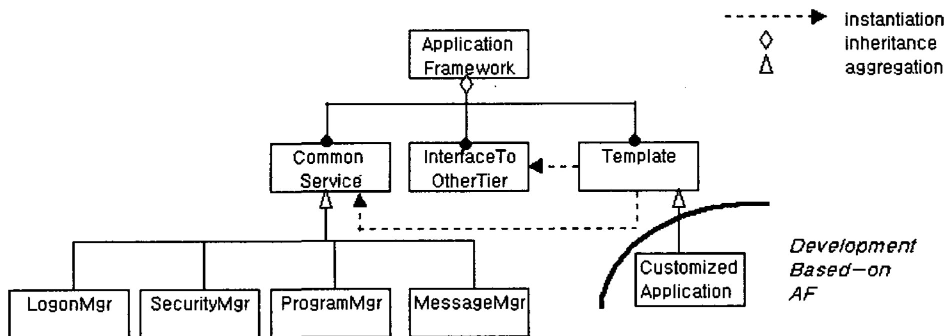
AF 의 개발과 재사용은 이와 같은 사실들과 같은 논리를 갖고 있다. 프로젝트 초기에 우수한 팀원들의 경험과 지식을 바탕으로 개발된 AF 는 유전을 통해 시스템의 전반적인 부분을 형성하며 따라서 구조적 일관성을 유지하여 시스템을 이해하기 위한 많은 노력을 절감할 수 있게 하는 것이다. 시스템에 대한 이해 증진 및 이해의 용이성은 개발 초기의 AF 에 대한 투자를 상쇄할만큼 그 이후의 생산성 향상에 기여하게 된다

4.4 AF 의 구성

AF 의 구성요소들의 성격은 템플릿을 중심으로 한 UI 와 특정 기능을 제공하는 공통 컴포넌트들로 크게 구분된다. UI 의 개발은 Visual Basic 에서 OCX 형태로 이루어 지며 공통 서비스의 제공은 Active X 컴포넌트로 개발되어 1-Tier 및 2-Tier 에 배치된다.

3-Tier 아키텍처에서 AF 의 개발은 각 층별로 공통적이면서 반복되며 도메인 독립적인 성격을 갖는 패턴을 도출하는 것으로 볼 수 있다. 각 층별로 존재하는 패턴들을 정의내리고 그 패턴들의 집합으로 전체 시스템을 개발하는 것이다.

패턴의 도출은 시스템의 중복되거나 유사한 부분을 단일화 함으로써 시스템의 복잡성을 감소시키고 구조적 단순성(Architecture Simplixity)을 실현한다.



<그림 2> AF 의 구성

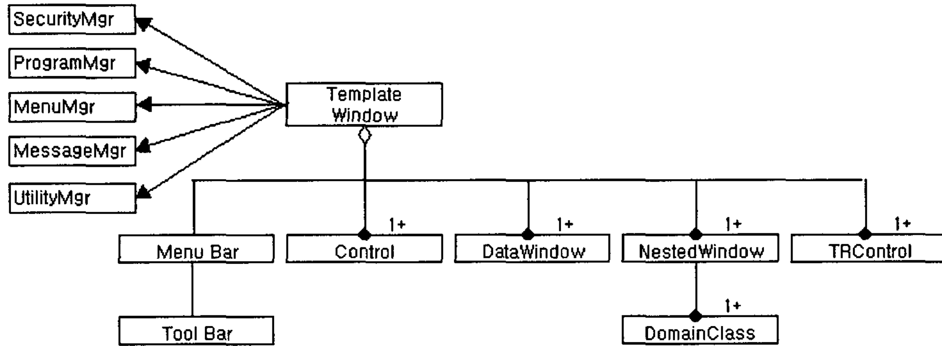
4.4.1 템플릿(Template)

템플릿은 반복되는 기능 유형을 정형화한 기능 수준의 재사용 단위로서 정보시스템의 메뉴와 같은 개별 기능을 개발하는 기본 모형을 말한다. 여기서의 템플릿은 정보시스템의 사용자가 접하는 하나의 화면이나 윈도우 자체를 대상으로 하며 직원 정보 조회, 급여 산출, 상여금 산출 등이 그 예가 된다.

템플릿 작성을 통하여 이후의 정보시스템 개발은 기존의 템플릿을 검색하여 유사한 템플릿을 선정하고 이를 유전받아 구조적 일관성을 유지하면서 필요한 기능들을 추가 또는 변경하여 사용하게 된다

템플릿은 다수의 컨트롤, 메뉴, 툴바, 내포 윈도우, 도메인 클래스 등으로 구성되며 공통적인 서비스를 제공하는 컴포넌트들을 호출하여 기능을 수행하는 구조를 가진다. 템플릿의 설계 방법은 특정한 도메인에 연관된 템플릿의 개발(Domain Approach)과 일반적으로 사용하기 위한 템플릿 개발(General Approach)로 구분할 수 있으며 여기서는 후자의 방법을 택한다. 선정한 템플릿에서 유전하여 재사용하지만, 유전관계를 지원하지 않는 Visual Basic 에서는 OCX 형태로 작성한 후 툴박스(Toolbox)에서 드래그 앤 드롭(Drag-and-Drop)으로 재사용하게 된다. 물론 유전관계에서처럼 수정 및 변경이 가능하다. 차이점이 있다면 유전관계를 통하여 수정 및 변경을 할 경우 유전관계의 대상인 템플릿이 가지고 있는 속성과 이벤트들이 그대로 유전되지만, OCX 로 개발되어 재사용할 경우 기본적인 속성과 이벤트들만이 제공되므로 추가적인 코드 작업이 OCX 개발시에 이루어 져야 한다.

따라서 정형화된 코드와 UI 디자인이 된 화면들을 프로젝트에 등록시켜두고 이에 대한 객체를 생성하여 이용하는 방법이 권장된다.



<그림 3> 템플릿의 구성

4.4.2 컨트롤(Control)

GUI 설계에 있어서 가장 기본적인 재사용 단위로서 화면에서 특정한 목적이나 기능을 수행하기 위해 시각적으로 표현되는 것을 컨트롤이라 하며 버튼, 텍스트 박스, 콤보 박스 등이 대표적이다. 이것은 비주얼한 개발도구들이 기본적으로 제공하고 있는 내장 컨트롤과 개발자가 자체적으로 개발하는 커스텀 컨트롤(Custom Control)로 구분되며 각 컨트롤들은 속성으로서 크기, 바탕색, 폰트, 폭과 같이 등을 가지며 각 컨트롤에 따라서 클릭, 더블클릭 등과 같은 이벤트(Event)를 지원한다. 이벤트는 사용자가 컨트롤에 가하는 동작에 따라서 기능을 하기 위한 것이다.

4.4.3 도메인 클래스(Domain Class)

도메인 클래스는 템플릿을 구성하는 최소 단위의 재사용 대상으로 속성과 이에 대한 컨트롤의 복합 클래스를 의미한다. 속성 값을 어떻게 화면에 디스플레이하며 또 입력된 값을 어떻게 변환하여 저장할 것인가를 결정하여 구현하고 이를 여러 윈도우 또는 다른 프로젝트에서 재사용한다.

이번 프로젝트에서 필요한 도메인 클래스는 사용자가 입력한 숫자를 자동적으로 세자리수마다 콤마를 삽입하는 것과 콤마가 표시된 숫자값을 데이타베이스에 저장할 경우에 이를 삭제하는 도메인 클래스, 그리고 은행계좌번호를 자동적으로 표시하는 도메인 클래스를 작성하였다. 이 도메인 클래스를 Visual Basic 5.0 에서는 OCX 형태로 구현 하였다. OCX(OLE Control Extensions)는 16 비트에서의 VBX 에서 32 비트로 진화한 것으로 기존의 Visual Basic 내에서만 사용가능하던 것을 윈도우 시스템의 자원으로서 서로 공유할 수 있도록 OLE 에 기반하여 작성된다. OCX 는 인터페이스 디자인을 위한 클래스로서 DLL 과 같은 구조의 함수 포인터 테이블이다. 또한 Active X Control 이기 때문에 인트라넷, 전자 상거래 등과 같은 인터넷상에서의 재사용 역시 가능하다.

4.4.4 내포 윈도우(Nested Window)

여러개의 컨트롤이 함께 묶여서 공통적으로 사용된다면 이를 통합하여 하나의 컨테이너 클래스(Container Class)로 만들어 이를 재사용하는 것이 효율적이다. 이것은 컨트롤과 도메인 클래스들의 집합으로 구성되므로 이들보다 보다 복잡한 상위의 재사용 단위가 된다. 그러므로 화면설계와 코드를 재사용하는 효과가 증대된다.

Visual Basic 에서 내포 윈도우의 구현 역시 OCX 를 통하여 이루어 진다. 단순히 상위의 재사용 단위를 작성한다는 점에서가 아니라 OCX 의 기본적인 속성과 이벤트가 한정되어 있기 때문에 필요한 속성 및 이벤트를 추가하여야 된다는 점에서 도메인 클래스의 제작 보다 복잡한 편이다.

4.4.5 추상 클래스(Abstract Class)

추상 클래스는 실제의 구현은 없고 다만 클래스의 속성과 연산에 대한 정의만을 제공함으로써 실질적인 서비스를 제공하지는 않지만 브룩스가 주창한 개념적 무결성 유지의 가장 기본적인 전제조건인 시스템 전반에 걸친 컨벤션의 준수를 보장하게 된다.[16] 그러나, 단순한 문법적 의미에서의 컨벤션만으로는 부족하기 때문에 계약 개념으로 확장하여 사용된다. 계약이란 연산에 기대하는 의미적 행동을 포착하는데 이것은 추상 클래스는 연산에 대한 구현을 칸크리트 클래스에 위임하지만 때로는 모든 행동을 가지는 것을 허용하지 않게 된다. 계약이란 클래스 연산들의 능력과 책임을 규정한 것이다. 즉, 다른 클래스와 어떤 협력 관계(연산 호출)를 가지며 서비스를 제공하는가를 규정한 것이다. 슈퍼 클래스의 계약은 서브 클래스에 유전된다. 서브 클래스는 유전받은 계약을 구체화시키나 깨뜨리지는 않는다. 서브 클래스는 슈퍼 클래스의 모든 책임을 완수해야 한다. 추상 클래스는 부분적으로 규현될 수 있으나 계약은 완전히 규정되어야 한다.

4.4.6 패턴(Pattern)

개념적 의미에서 패턴은 복잡한 현상이나 사건, 물질에서 어떤 규칙성 또는 유사성을 갖는 부분들이 존재하는 것을 말한다. 앞서의 DNA 와 카우스 이론의 비교에서처럼 AF 의 개발은 패턴의 발견으로 이루어진다. DNA 는 그 자체가 패턴을 의미한다고 볼 수 있고 생명체의 모든 부분에서 발견되며 카우스 이론 역시 복잡한 실세계의 아주 작은 부분에서 질서로 표현되는 단순한 구조를 발견하여 그 구조를 반복해서 재현함으로써 실세계를 묘사한다. 하지만 정보시스템 개발에 있어서 패턴의 발견은 이미 존재하는 시스템에서 패턴을 찾는 것이 아니라 요구분석의 다음 단계에서 찾아야 하고 발견한 패턴을 바탕으로 AF 를 개발하여 계속적으로 재사용해감으로써 복잡한 시스템을 구축하게 되는 것이다.

발견된 패턴은 AF 의 모든 구성요소들 또는 구성요소들의 결합의 형태로 실제 구현된다.

이와 같은 패턴은 요구분석의 다음 단계에서 휴리스틱하게 발견하게 되는데 생성 패턴(Creational

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

Patterns), 구조 패턴(Structural Patterns), 행동 패턴(Behavioral Patterns) 등 3 가지 유형의 패턴으로 구분하고 적용하였다

4.4.7 컴포넌트(Component)

컴포넌트의 대상은 사용자의 시스템 로그인 체크, 보안 기능, 프로그램 관리 기능, 메시지 처리 기능 등과 같은 다수의 템플릿에서 공통적으로 사용하는 기능을 분류, 정의한 것으로 1-Tier 에 배치되는 일반적이며 공통적인 서비스를 제공하는 컴포넌트와 주로 비즈니스 로직을 구현하기 위해 비즈니스 서비스층인 2-Tier 에 배치되는 비즈니스 컴포넌트로 구분된다. 즉 2-Tier 의 컴포넌트들은 1-Tier 에서 사용자의 요구사항에 따라 요청받은 트랜잭션 처리를 위해 데이터 서비스 층인 3-Tier 에 접속하여 데이터를 받아 가공하고 그 결과를 다시 1-Tier 에 전달을 수행한다.

4.4.8 Visual Basic 에서의 재사용 부품

재사용을 목적으로 작성되는 파일의 유형은 Active X DLL, Active X EXE, OCX, BAS 로 구분된다. 이중, Active X EXE 파일은 주로 비즈니스 로직을 구현하는 컴포넌트를 말하며, Active X DLL 은 동일한 시스템 내부에 배치되는 컴포넌트이다. BAS 는 일반적으로 가장 많이 사용되고 있는 공통적인 기능을 제공하는 함수와 프로시저, 그리고 변수들의 선언이 이루어 지는 곳이다. 반드시 프로젝트내에 포함되어 있어야 사용이 가능하다는 단점을 가지고 있다.

마지막으로 OCX 는 주로 UI 를 재사용하기 위해 사용되는 파일이다. 개발자의 고유한 필요에 따라서 제작되어 재사용할 수 있으며 현재 많은 상용 OCX 들이 개발되어 시장에서 구입할 수 있으며 Spread, Truegrid 등이 대표적인 상용 OCX 들이다. 상용 OCX 는 편리하고 다양한 기능을 제공하기 때문에 일반적으로 많이 사용되고 있지만, 추가적인 구입 비용의 부담과 굳이 개발자에게 필요하지 않는 많은 다른 기능들을 함께 포함하고 있기 때문에 시스템의 퍼포먼스에 나쁜 영향을 주는 단점이 있다.

OCX 는 마이크로소프트사의 전략적 표현의 하나로서 OCX 를 시스템 수준에서 공유하기 때문에 상이한 개발도구에서 개발된 OCX 도 사용가능하다는 장점을 가지고 있다. 즉 새로운 개발도구를 익혀야 한다는 부담감을 덜어주는 것이다.

4.5 AF 개발 및 AF 를 재사용하는 정보시스템 개발

AF 는 다수의 개발자들이 참여하는 큰 규모의 정보시스템 개발시 그 시스템에 한한 AF 의 개발과 재사용도 타당하다. 그것은 AF 가 단순히 재사용부품으로서의 의미만을 가진 것이 아니라 프로젝트 팀원중 가장 뛰어난 개발자의 경험과 지식을 바탕으로 하거나 모든 팀원의 바람직한 능력들을 기반으로 하여 시스템 구조에 대한 청사진을 우선 개발한 후 이를 재사용함으로써 시스템의 전반적인 품질 향상을 기대할 수 있다.

일단 AF 가 개발된 후 다음 단계는 프로젝트 팀원들이 충분히 AF 를 이해할 수 있는 방안과 AF 의

관리 방안이 마련되어야 한다. 즉 적절하고 충분한 도큐멘트, AF 관리 책임자의 선정, AF 의 개선 및 수정에 대한 지침 등이 그것이다.

AF 의 준수여부에 대한 확인작업은 이후 프로젝트관리에 있어서 가장 중요한 업무의 하나가 되어야 한다. AF 구조자체에 대한 변경은 특히 엄격하게 방지되어야만 하며 개발도구 자체가 유전관계를 지원하지 않아 Copy-and-Paste 에 의해 AF 를 재사용하는 경우 보다 많은 관리적 노력이 요구된다.

시스템 개발이 진행되면서 개발초기에 작성된 AF 의 단점 및 문제점이 발견되면 AF 의 수정작업이 팀원간의 확인하에 이루어지며 그때까지의 개발결과에 대한 수정까지 하게 된다. 이때 AF 의 재사용은 유전관계를 기반으로 이루어졌기 때문에 그 수정작업의 양이 대폭 줄어들 수 있게 된다.

AF 에 대한 도큐멘트들은 이후 완성된 시스템의 유지보수에도 큰 도움을 주게 된다. 시스템 전반의 공통적인 구조에 대한 정보를 집약적으로 담고 있기 때문에 시스템 이해에 대한 많은 노력이 절감되기 때문이다.

이와같은 효과를 거둘수 있는 것은 AF 가 코드수준에서의 재사용이 아니라 보다 근본적인 시스템의 구조에 대한 정보를 제공하는 디자인수준의 재사용단위이기 때문이다.

도메인에 독립적인 성격을 갖는 AF 의 개발이 계속될수록 AF 의 개발경험과 지식의 축적으로 보다 우수하고 정교한 AF 의 개발이 가능해질 것이며 또한 특정한 도메인에 대한 AF 의 개발까지 경제적 측면에서 고려하더라도 가능해질 것이다. 그것은 AF 들 간의 공통적인 부분인 패턴을 발견할 수 있을 것이고 따라서 AF 의 공통적인 부분을 제외한 도메인의 특성을 반영하는 예외적인 부분들에 대한 개발에 집중할 수 있기 때문이다.

4.6 적용 사례

4.6.1 D 사

D 사는 339 개 기능(메뉴)을 13 개 템플릿을 사용하여 제조, 수급, 영업정보시스템을 성공적으로 구축하였다. NT 서버 환경에서 PowerBuilder 5.0 과 SQL Server 를 기본 개발도구로 한 AF 의 기본 템플릿은 기본적으로 유전관계를 이용한 윈도우 네비게이션, 툴바 작업, 데이타 디스플레이 등의 유형별 표준이다. 템플릿간의 유사성을 별도의 루트 템플릿으로 일반화 하지 않았으며 에러 메시지 처리, 메뉴관리, 보안 관리 등 비 가시적 모듈에 대한 재사용은 시도 되지 않았다. AF 를 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 발견되었다. 유전관계는 1 단계로 제한하여 성능 오버헤드를 줄였으며 스크립트의 추적도 용이한 상태이다. 또한, 동일한 템플릿을 유전받아 개발된 프로그램들의 공통 기능을 변경할 경우, 상위 템플릿만 수정하면 되기 때문에 유지보수가 간편하고 일관성있게 수행될 수 있다. D 사는 현재 2 단계로 회계정보시스템 구축 중에 있으며 지속적으로 템플릿의 구조 개선과 통합을 추진하고 있다.

4.6.2 C 사

D 사와 동일 그룹 계열사인 C 사는 동일한 환경, 개발도구로 통합 정보시스템을 구축 중인데 D 사와는 달리 공통 기능을 구현한 루트 템플릿을 기초로 이와 유전관계로 연결되는 7 개 구현 템플릿을 개발하고 실제 기능 구현시 이를 복사하여 각 연산별로 사전에 표시된 영역을 수정하여 개발하는 방법을 사용하였다. 1 단계 유전관계를 사용하였으며 중복 코드를 최소화 하였다. 또한, 템플릿은 예를 들어, 입고, 출고와 같은 업무 프로세스의 콘트롤 구조의 패턴을 발견한 후 개발하여 전반적으로 D 사에 비해 템플릿 수가 적으며 일반성도 향상되었다.

4.6.3 K 사

K 사는 전체 3000 개의 기능으로 구성된 병원 프로젝트에 AF 를 적용하고 있는데 기능상 배타적인 업무 영역인 원무, 진료, PACS, 일반관리 등 4 개 서브 프로젝트팀으로 분리하여 추진되었다. UNIX 환경에 Forte, Oracle 을 툴로 사용하는 전형적인 3 계층 클라이언트/서버 구조로 개발되었는데 약 30:1(1 개 템플릿을 30 개 기능에 적용)의 비율의 템플릿 사용율을 보이고 있다.

4.6.4 Y 사

Y 사는 IBM 메인 프레임에 COBOL/CICS-DB2, UNIX 기능서버에 MagnaX 4 세대 언어를, PC Windows95 에 VisualBasic 을 사용한 전형적인 IBM 호스트 중심의 3 계층 클라이언트/서버 구조로 신회계시스템을 개발 중인데 클라이언트에 1 개 기본 템플릿을 개발하여 이를 기반으로 해당 모듈을 수정 개발하는 방법을 사용하고 있다. 개발도구가 유전관계를 지원하지 않기 때문에 Copy-and-Paste 에 의해 개별기능을 개발하고 있다.

V. 결론

5.1 이번 연구의 의미와 결론

이번에 제시된 애플리케이션 프레임웍은 아직 완전한 형태를 갖추지 못하고 있으며 적용사례도 충분하지 못하지만 다음과 같은 의미를 가진다.

첫째, 기존의 UI 중심의 애플리케이션 프레임웍 개발, 재사용에서 공통 기능 및 비즈니스 로직을 구현한 컴포넌트들을 포함하는 애플리케이션 프레임웍을 제안하고 구체적인 개발과정을 밝혔다.

둘째, 코드수준의 재사용에 머무르는 컴포넌트를 AF 에 포함시킴으로서 디자인수준에서의 보다 적극적인 재사용을 수행할 수 있다.

셋째, 개발 초기에 시스템의 구조적 청사진을 제시함으로써 개발자들 시스템에 대한 구체적인 목적과 내용을 동일하게 인식하게 함으로써 시스템의 중복과 복잡성을 최소화할수 있을 것이다.

네째, AF 를 재사용하여 구조적 단순성을 실현함으로써 예상되는 긍정적 효과들의 규명을 시도하였

다.

5.2 이번 연구의 한계와 이후 연구 방향

애플리케이션 프레임웍의 연구는 앞으로도 계속될 것이며 다음과 같은 방향으로 진행될 것이다.

첫째, AF의 경제적 손익에 대한 실증조사가 아직은 적용사례가 충분하지 않기 때문에 곤란하지만 이후에는 가능해질 것이다.

둘째, AF의 적용에 따른 프로젝트 관리의 변화와 AF의 관리 방안에 대한 연구가 필요하다.

셋째, 개발도구에 독립적인 AF의 개발의 연구를 통해 AF의 개발을 보다 광범위하게 촉진할 수 있게 될 것이다.

넷째, 특정 도메인에 적합한 프레임웍의 개발을 모색할 수 있다.

다섯째, AF의 적용사례가 충분해지면 재사용한 시스템의 유지보수노력의 절감에 대한 연구가 가능해질 것이다.

[참고문헌]

1. 이주헌, *실용 소프트웨어 공학론*, 법영사, 1993
2. 최은만, 김진석, "객체지향 재사용과 CASE", 정보과학회지, 14권 10호, 1996
3. Adele Goldberg & Kenneth S. Rubin, *Succeeding with Objects*, Addison Wesley, 1996
4. Brooks, J., *Mythical Man-Month*, Addison-Wesly, Reading, NY, 1965
5. Fowler, *Analysis Patterns*, Addison Wesley, 1996
6. Grady Booch, *Object Solutions*, Addison Wesley, 1996
7. Hammer & Champy, *Reengineering the Corporation*, HarperBusiness, 1993
8. Marko Krajnc, "Why Component-Oriented Programming ? ", 1997
9. Marko Krajnc, "What is Component-Oriented Programming ? ", 1997
10. Microsoft, "The Microsoft Object Technology Strategy : Component Software", 1996
11. M.McIlroy, "Mass Produced Software Components", Proc. NATO Conf. of Software Eng. Petrocelli/Chater, NewYork, pp 88-98, 1969
12. Ralph E. Johnson, "Designing Reusable Classes", The Journal of Object-Oriented Programming, Vol.1, No.2, 1988
13. Robert E. Shelton, "Business Object Frameworks and Patterns", Data Management Review, May 1995
14. Robert E. Shelton, "Business Objects Modeling with Business Patterns", Data Management Review, May 1996
15. Shaw, B., *Project Review Presentation*, Marcam Corp., 1996
16. T.Biggerstaff & C. Richter, "Reusability Frame work, Assessment, and Directions", IEEE Software Vol. 4, No. 2. 1987