어플리케이션 빌더용 클래스 라이브러리 관리자에 대한 설계 및 구현

이치근* · 김준하*

Design and Implementation of a Class Library Manager for Application Builders

Chikuhn Lee · Joonha Kim

〈Abstract〉

During the past few years, the productivity has been considered as one of the most important aims in developing softwares. Many practitioners and theorists are trying to reuse ready-made codes to increase the productivity of software development and the quality of their products. This paper introduces our experiences in developing a highly reusable class library and its reusing environment for building a database interface application. And the concepts, Structured Multi-Library and Dual Views of Class Inheritance, are also suggested, which consist our main conceptual framework in our work. Details in implementation are discussed in brief at the end.

1. 서론

1.1 소프트웨어 생산시스템

소프트웨어에 대한 환경의 변화는 높은 생산성과 유지보수의 편리함, 신뢰성보장이라는 요건을 만족시키는 개발방법을 요구하게 되었다. 소프트웨어에 대한 최근의 요건을 정리하면 다음과 같다.

- 사용방식의 고급화
- 유지보수의 용이화
- 확장성
- 이식성
- 고성능화
- 통합화
- 개방화

이와 같은 요구사항들을 만족하는 소프트웨어의 개발은 많은 경험과 체계적인 관리방식, 단위 구성부품의 표준화와 품질의 보장이 없이는 불가능한 것들이다. 그러나, 많은 경험, 체계적인 관리방식등은 수많은 실패, 교육등이 없이는 불가능한 것으로 이에 대한 비용의 지출 또한 엄청나다. 따라서, 경험이나 관리방식을 대신할 수 있는 소프트웨어 개발방법에 대한 새로운 개념이나 원하는 소프트웨어를 생산해주는 높은 생산성의 자동화된 소프트웨어 생산시스템에 대한 수요가 자연스럽게 발생하였다. 이에따라 전세계적으로 20여가지에 이르고 있는 시스템 빌더(System Builder)들이 상품화되었다. 국내에 소개된 대표적인 시스템 빌더로는 PowerSoft사의 PowerBuilder, Gupta Technology 사의 SQL Windows, Popkin Software & Systems 사의 System Architect, KnowledgeWare사의

* 삼성데이타시스템 정보기술연구소 S/W연구팀

ObjectView, IBM의 VisualAge, Borland사의 Delphi 등이 있다.

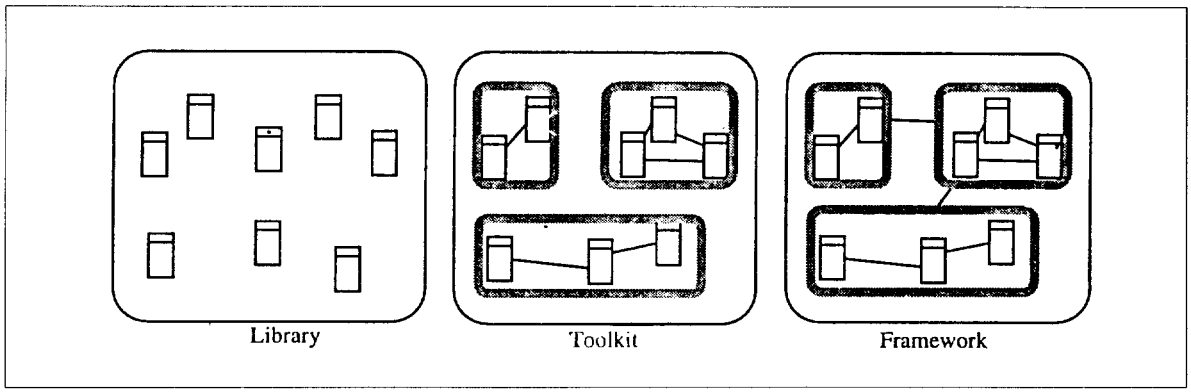
이들 어플리케이션 개발환경들은 공통적으로 사용용이성과 개발유연성(flexibility)사이에서 나름대로의 트레이드 오프(trade-off)를 취하고 있는데, 사용자의 업무분야에 따른 시스템 재정의와 상이한 사용능력을 가진 사용자가 자신의 상황에 맞게 빌더를 사용할 수 있도록 하는 기능이 없다. 본고에서 다루고 있는 빌더와 클래스라이브러리는 이와 같은 단점을 구조화된 다수라이브러리와 이중상속구조를 통해 해결하고자 한다.

2. 라이브러리

2.1 라이브러리, 프레임웍, 툴킷

사용해야 하는데, 이는 툴킷내의 클래스들은 하나의 작업에 대해 두가지 이상의 클래스가 관련되어 있기 때문이다. 이에 반하여 라이브러리내의 클래스는 특정 작업수행을 위해서 다른 클래스를 거의 사용하지 않는다. 툴킷을 구성하는 클래스군내에 속한 클래스간에는 종속관계가 존재하지만 다른 클래스군의 클래스간의 관계는 독립성이 보장된다. 프레임워크는 상호 밀접하게 연관된 툴킷과 라이브러리의 모임이다. 프레임워크는 각 부문에 대한 기능집합을 제공할 목적으로 사용되기보다는 어플리케이션의 전반적인 구조를 제공한다고 할 수 있다. 라이브러리, 툴킷, 프레임워크의 구조를 개념적으로 표현하면 다음의 〈그림 1〉과 같다.

라이브러리는 소속 클래스간의 독립성으로 인해 재사용을 위한 가장 적합한 재사용코드의 관리단위로



〈그림 1〉 라이브러리, 툴킷, 프레임워크의 개념

소프트웨어의 재사용에 관한 시도는 하위함수에 대한 호출이 가능해지면서 활발하게 진행되어 오고 있다. 코드의 재사용성 확보를 위한 가장 대표적인 방법이 라이브러리(Library)인데, 이와 유사한 개념으로 프레임워크(Framework)와 툴킷(Toolkit)이 있다. 라이브러리란 상호간에 대체로 독립적인 클래스의 모임으로 이들 클래스들은 공통의 작업을 목표로 하거나 이에 관련된 작업을 목표로 해야한다. 툴킷(Toolkit)이란 라이브러리의 경우보다도 더욱 밀접하게 연관된 클래스군의 모임으로 정의할 수 있다. 툴킷내의 한 클래스를 사용하면 이와 관련한 다른 클래스들도 함께 생각되어지고 있다. 본 연구에서는 라이브러리가 가지는 클래스간의 독립성과 이들을 재사용하는 환경으로서 프레임워크를 운영함으로써 재사용 가능한 어플리케이션 빌딩환경을 구축하였다.

2.3 라이브러리의 속성 및 평가항목

재사용을 보장하기 위한 라이브러리가 갖추어야 할 속성들을 정리하면 〈표 1〉 과 같다[4].

라이브러리가 만족시켜야할 속성과 라이브러리를 평가할 항목간의 연관관계를 다음의 〈표 2〉에 정리하

〈표 1〉 라이브러리의 속성(Attributes)

Attributes	정 의
Complete	라이브러리는 개념전체를 처리할 수 있어야 한다.
Consistent	라이브러리는 일관된 접근방식을 따라야 한다. 즉, 문서의 양식, 각종 이름에 일관성이 보장되어야 한다.
Easy to learn	처음 사용하는 사용자가 쉽게 적응할 수 있도록 설계되고 구현되어야한다.
Easy to Use	사용 정보와 코드가 찾기 쉽고, 접근이 용이해야 한다.
Efficient	라이브러리는 구성하는 각종 요소코드는 전체적인 시스템의 수행성능을 개선하는 방향으로 영향을 미칠 수 있도록 가장 적합한 알고리즘으로 최적화되어 있어야 한다.
Extendable	최소의 노력으로 새로운 클래스를 첨가할 수 있도록 설계되어 있어야 한다.
Integrable	여러 공급자로부토 제공되는 다른 라이브러리들을 기본 소프트웨어에 쉽게 연결시킬 수 있어야 하고 동일 어플리케이션내에서 다양한 라이브러리내의 클래스를 자유롭게 사용할 수 있어야 한다.
Intuitive	분야별 전문가나 해당 분야의 통상적인 관례를 따라서 설계되어야 한다.
Robust	라이브러리를 구성하는 클래스들은 각종 비정상적인 상황에서도 적절히 대응할 수 있어야 한다.
Supported	새로운 버전으로 계속 제품을 향상시켜나가는 공급자의 제품이어야 한다.

였다. 〈표 2〉에서 "X"로 표시된 것은 좌측의 라이브러리리 속성(Attribute)과 상단의 평가기준(Criteria)이 명시적으로 관련이 되어있음을 의미하고"I"로 표시된 것은 암시적인 관련이 성립함을 의미한다.

3.어플리케이션 빌더와 라이브러리 관리자

3.1 어플리케이션 빌더의 개발 패러다임

시스템 빌더란 특정한 작업을 처리하기위한 특수목적의 소프트웨어가 아니고, 다양한 어플리케이션을 구축하는데에 사용되는 가장 일반적인 용도의 소프트웨어이므로 가장 기본적이면서 완벽한(complete) 기능집합을 제공해야만 한다. 그러나, 아무리 많은 기능을 제공한다고 하더라도 다양한 사용자의 요구를 모두 만족시킬 수는 없다. 따라서, 일반적인 어플리케이션을 정의하는데에 필요한 공통의 기능을 효율적으로 구조화시켜서 프레임워크로 구현하고 분야별 특수목적의 라이브러리를 제공하는 것이 바람직하다. 각 부문별 특수기능은 라이브러리의 형태로 전문화시켜나가는 구조로 프레임워크(Framework)을 설계했다. 즉, 여러종류의 라이브러리와의 접속기능이나 접속된 라이브러리의 사용을 지원하는 라이브러리관리자의 기능을 풍부하게 제공하는 것으로 설계방향을 설정하였다.

이와 같은 사용환경의 전체적인 구조는 〈그림 2〉와 같다.

3.2 메인프레임의 기능

메인프레임은 작업중인 내용을 저장하고 화일의 형태로 존재하는 작업내용을 메인 메모리로 올려서 작업을 수행하게 하고, 가용한 자원의 위치와 자원간의 관계를 정의함으로써 새로운 어플리케이션을 만들어나가는 전체적인 환경을 지원하고 있다. 라이브러리는 사용자로 하여금 새로운 클래스의 정의를 가능하게하고 존재하는 클래스들의 조회와 변경을 지원한다. 또한, 객체지향의 각종 메카니즘을 가능하게하는 임무로 수행하는데, 정보은닉(Information Hiding), 캡슐화(Encapsulation)등이 이에 해당한다.

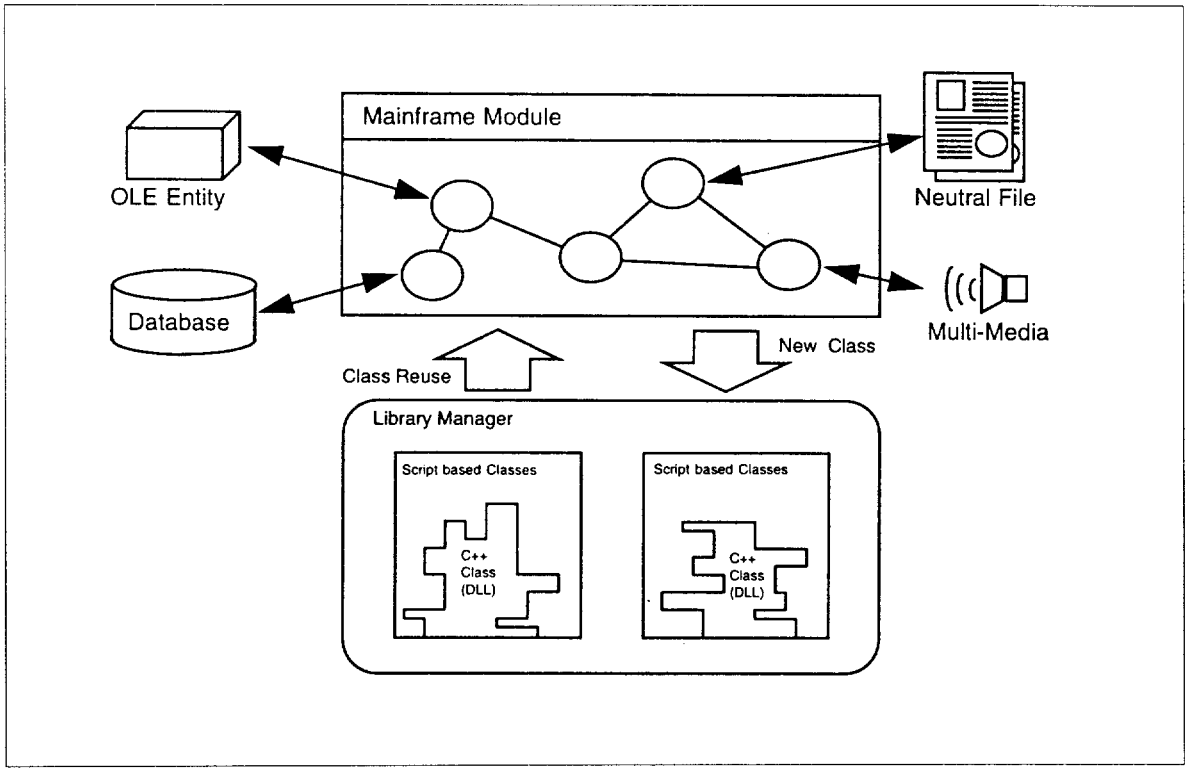
메인프레임의 임무는 어플리케이션의 조립과 이들의 관리, 스크립트 처리의 세가지로 요약할 수 있다.

〈표 2〉 속성 평가기준표

Attributes	Criteria																						
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Complete	×									×			I	×			I						
Consistent			I	I	I		×	×		×		×							I				
Easy to Learn	I	I	I	I		I				I		I			×	×	I	I	×	I			
Easy to Use	I	I			I	I	×	I		I			I	I	I	I		×	I				
Efficient											×												
Extendable	I	×		×																	×		×
Integrable					I																×		
Intuitive			×																				
Robust				I		I	I	I	×										I				
Supported																						×	

X : Explicit I : Implicit

1. Completeness : 완벽한 일반적인 모델을 제공할 수 있어야 한다.
2. Abstractions : 한두개 정도의 Key Abstraction을 제공하고 있어야 한다.
3. Standard : 적용대상 분야에 대한 표준화된 지식을 제공해야 한다.
4. Inheritance structure : generalization/specification 관계를 표현하는 상속관계를 정의해야 한다.
5. Purity : 다른 클래스와의 관계가 모호하거나 Procedural한 항목이 없는 클래스들로 구성되어야 한다.
6. Coupling : 클래스간의 Coupling이 약해야 한다.
7. Exceptions : 오류나 예외적인 상황에 대해서 일관되고 이해하기 쉬운방식이 수립되어있어야 한다.
8. Partial Functions : 부분적인 작업을 수행하는 함수에 대해서 선행조건을 점검하는 장치가 제공되어야 한다.
9. Integrity of the abstraction : 라이브러리 클래스에 의해 구현된 추상화를 그 사용자가 위반할 수 없어야 한다.
10. Complete Interface : 라이브러리내의 클래스들은 최소의 Interface를 가져야 한다.
11. Efficient : 클래스내부의 Method구현은 수행시간과 사용메모리에 있어서 가장 효율적으로 구현되어야 한다.
12. Consistency : 각종 품목에 대한 정의와 이름은 일관성을 가지고 있어야 한다.
13. Generes : 가능한 한 가장 일반적인 클래스를 제공해야 한다.
14. Full Implementation : 사용자가 클래스의 구현이 어느정도를 완벽하게 이어져 있는지를 알 수 있게해야 한다.
15. Organization : 문서는 라이브러리의 전반적인 구조를 쉽게 알 수 있도록 구성되어야 한.
16. Overview : 문서는 라이브러니의 내용과 구조를 포함해서 전반적인 것을 담고 있어야 한다.
17. Orientation : 다양한 수준의 사용자들에게 필요한 내용을 담고 있는 문서화가 이루어져야 한다.
18. Indexes : 문서는 최소한 3가지의 검색방법을 제공해야 한다. 즉, 철자순서에 의한 검색, 상속의 계층구조를 이용한 검색, 키워드를 이용한검색.
19. Formal Specification : 라이브러리의 구성요소들에 대한 정규화된 명세가 제공되어야 한다.
20. Accessing tools : 사용자가 각종 클래스를 참조하고 위치를 찾아내는 작업을 보조할 장치가 마련되어 있어야 한다.
21. Integration tools : 사용자가 새로운 클래스를 추가할 수 있어야 한다.
22. Support : 상용화된 라이브러리를사용할 수 있어야 한다.
23. Upgrade : 객체지향 라이브러리의 Upgrade는 필수적이다.



〈그림 2〉 개발중인 어플리케이션빌더의 구조

1) 어플리케이션의 조립

라이브러리내에는 가용한 자원이 저장되어 있다. 이들은 3차원 효과를 내는 버튼, 스크롤링이되는 리스트, 문자열이나 수치입력이 가능한 입력필드(Text field)등이다. 사용자는 메인프레임이이 제공하는 각종 환경을 이용해서 가용한 자원을 페이지위에 위치시키고, 이들간의 관계와 동작을 스크립트로 정의해나간다. 페이지간의 관계도 그위에 존재하는 자원이나 페이지 자체의 스크립트로 인해 정의가 가능하다.

2) 어플리케이션 관리

작업중인 어플리케이션을 영구기억장치에 저장하고, 영구기억장치로부터 작업할 어플리케이션을 불러들이는 작업 역시 메인프레임의 임무중 하나이다. 저장과 열기외에도 전체적인 페이지들의 구조와 인쇄작업, 클립보드(Clipboard)를 이용한 에디트(Edit)기능, 사용환경의 설정등이 어플리케이션 관리를 위해서 필요한 기능들이다.

3) 스크립트 처리

새로운 클래스의 정의는 속성의 정의와 메소드의 정의를 통해서 이루어 진다. 속성의 정의는 사용환경이 제공하는 일련의 메뉴항목의 선택과 다이알로크 박스에 간단한 내용의 정보를 입력함으로써 이루어지고, 메소드의 정의는 스크립트를 이용한 간단한 프로그래밍이 필요하다. 이렇게 정의된 스크립트코드는 라이브러리에 의해서 관리되지만 최종적인 수행의 책임은 메인프레임내에 존재하는 스크립트 인터프리터(Script Interpreter)에 의해서 이루어 진다.

3.3 라이브러리 관리자의 기능

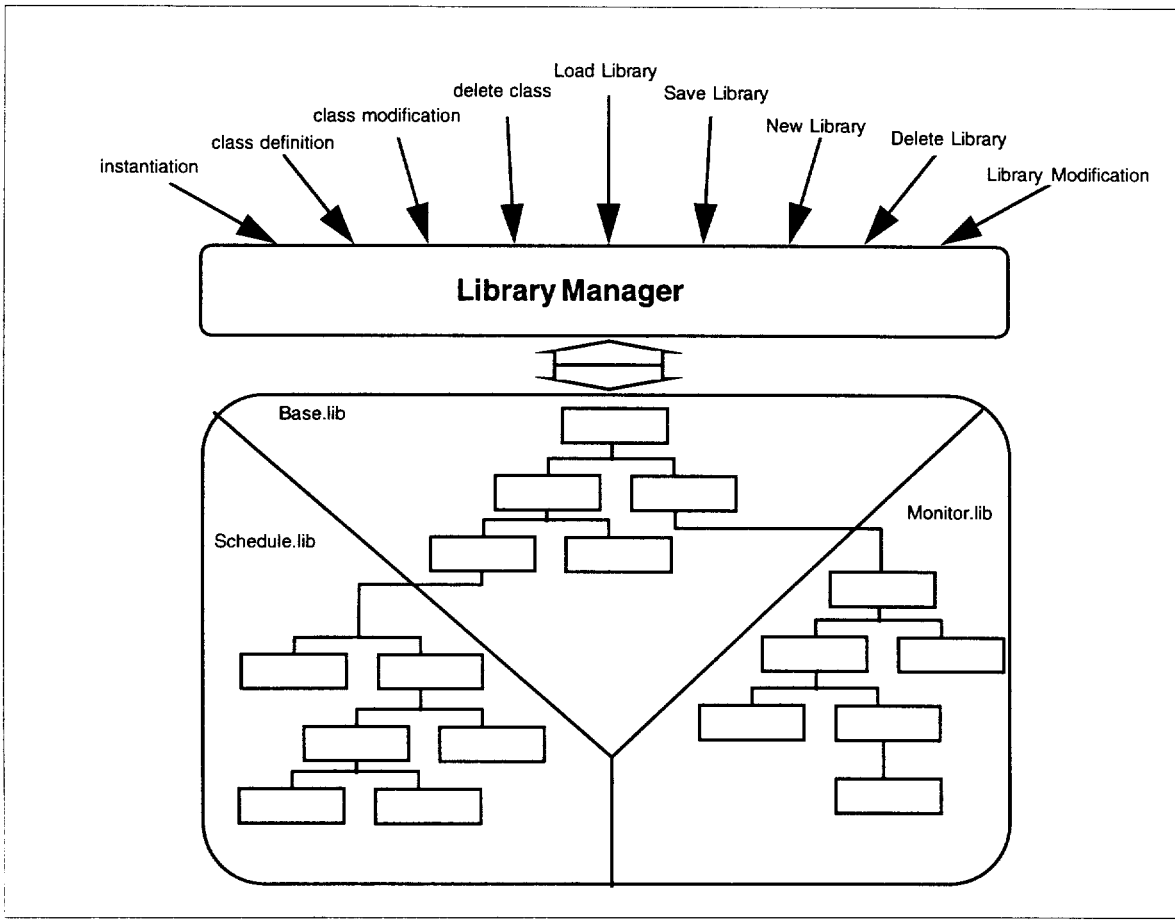
메인프레임이 사용할 자원의 관리와 새로운 자원의 등록, 기존 자원의 변경과 삭제, 새로운 라이브러리의 접속을 통한 확장, 라이브러리의 저장, 열기등의 기능을 수행하는 것이 라이브러리 관리자이다. 라이브러리 관리자의 기능은 다음과 같다.

- 실행시간 오브젝트의 생성 (Instantiation)
- 신규 클래스의 정의
- 기존 클래스의 정의 변경
- 기존 클래스의 삭제
- 라이브러리 열기
- 라이브러리 저장
- 신규 라이브러리의 생성
- 라이브러리의 변경과 삭제

이와 같은 라이브러리관리자의 임무와 자원의 저장 수단이 되는 라이브러리간의 구성관계는 다음의 〈그림 3〉과 같다.

1) 구조화된 다수 라이브러리

〈그림 3〉에서와 같이 라이브러리 관리자는 사용자가 정의한 여러개의 라이브러리를 하나의 단일구조로 통합하여 사용자가 여러개의 라이브러리를 사용하는 데에 따르는 불편함을 제거하고 있다. 이를 위해서는 시스템의 시동시에 관련 라이브러리의 구조를 읽어 들여서 분리된 정보를 하나로 통합하는 작업을 수행



〈그림 3〉 라이브러리 관리자의 기능

해야 한다. 각 라이브러리내의 클래스 계층구조는 다른 라이브러리내의 클래스 계층구조와 연계되어 정의되는데 정보의 참조방향에 따른 의존성이 존재하게 된다. 이와같은 의존성을 라이브러리간의 이진관계(binary relation)로 표현한다. 라이브러리 A의 클래스 중 하나이상이 라이브러리 B에서 정의된 클래스로부터 상속되었거나, 라이브러리 B내의 클래스 정보를 사용하고 있는 경우에, 라이브러리 B는 라이브러리 A를 "참조(Refer-to)"한다라고 정의한다.

다음의 〈그림 4〉은 3개의 라이브러리가 결합되어 사용자에게 여러개의 자원을 제공하고 있는 라이브러리 구조의 예를 나타내고 있다. Base.lib는 가장 기본적인 자원들로 구성된 라이브러리로 다른 라이브러리의 상속계층구조의 상부에 항상존재해야 하는 클래스들로 이루어져 있다.

2) 라이브러리의 사용자관점과 개발자 관점

라이브러리에 대한 해석은 두가지로 가능하다. 하나는, 이들자원을 사용할 어플리케이션 개발자, 즉, 어플리케이션빌더의 사용자관점이고, 또 다른 하나는 이들 클래스의 기능을 구현해서 사용자에게 공급할 시스템개발자의 관점이다. 사용자에게는 스크립트와 메인프레임을 이용해서 정의,수정가능한 부분만이 공개된다. 사용자들은 그들이 작성중인 어플리케이션내에서 존재할 소프트웨어의 단위 부품으로서의 클래스를 바라보게 되고, 이것은 목표로한 어플리케이션의 요소기능의 관점에서 해석되게 된다. 그러나, 이를 제공하는 시스템개발자의 관점에서는 클래스들의 당연하다고 여겨지는 행위와 상속과 정보은닉등의 기능과 사용자가 정의한 스크립트를 저장, 조회, 실행하는 것들과 같은 어플리케이션개발자가 기대하지 않는 클래스의 후면을 책임져야 한다. 따라서 클래스는 크게 어플리케이션개발자가 정의할 수 있는 부분과 어플리케이션 개발자의 정의부분을 지원하는 부분으로 구성된다. 이와 같은 이원적인 클래스의 구조를 나타내면 다음의 〈그림 5〉와 같다.

4. 라이브러리 관리자의 구현

4.1 전체적인 데이타구조

라이브러리모듈을 구성하는 주요 데이타구조체로는 아래와 같은 것들이 있다.

- zLibManager
- zLibHead
- zTreeHead
- zNode
- zEvtTbl
- zAtbNode
- zFunctionList

이들은 메인프레임의 각종 서비스요구와 질문들에 대응하기 위해서 가용자원을 저장하고 이들을 조회, 유지, 변경하기 위한 오퍼레이션과 정보콘테이너들로서 각각에 대한 내용은 다음과 같다.

zLibManager

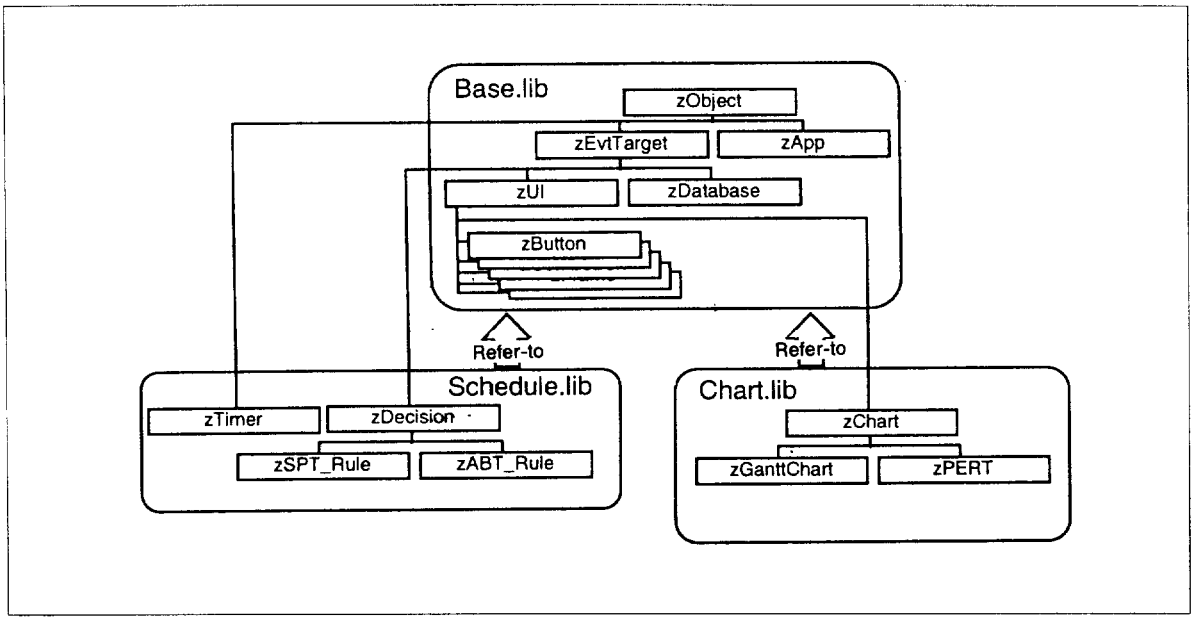
라이브러리에 대한 메인프레임의 작업요구를 접수하는 창구기능을 수행하는 자료구조로서 가능 작업내용에 대한 인터페이스를 제공한다. 여러개의 라이브러리로부터 읽어 들인 정보에 대한 엔트리 포인트(Entry Point)에 대한 포인터를 가지고 있다.

zLibHead

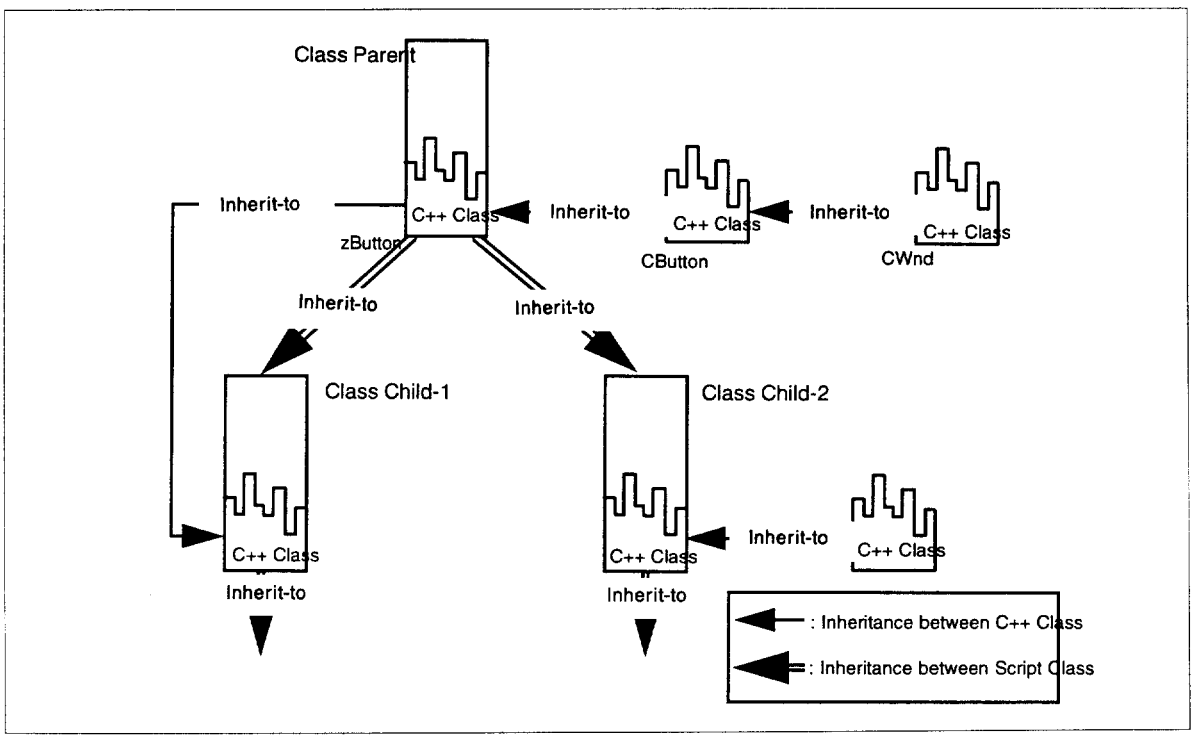
하나의 라이브러리에 대한 상세정보의 저장장소를 제공함과 동시에 이들을 저장하고 읽어들이는 메소드를 가지고 있다. 또한, 라이브러리내의 클래스 상속계층구조에 대한 관리책임을 지고 있으며, 해당 라이브러리내에 포함된 여러개의 클래스 계층구조에 대한 엔트리포인트를 관리한다.

zTreeHead

상속에의한 하나의 트리구조를 형성한 클래스들의 집합체를 관리한다. 즉, 상속구조에 대한 정보를 저장하고 있는 트리구조의 관리자기능을 수행한다. 트리구조내에 존재하는 클래스정보에 접근하기 위해서 반드시 통과해야 하는 트리구조의 루트노드(Root Node)



〈그림 4〉 구조화된 다수 라이브러리의 예



〈그림 5〉 클래스의 이원 상속 구조

에 대한 포인터(Pointer)와 트리구조자체에 대한 기술 정보를 보유하고 있다.

zNode

하나의 클래스에 관한 정보를 저장할 정보의 최소 단위로 클래스의 이름과 속성, 메소드, 처리가능한 이벤트등에 관한 실질적인 정보를 저장하고 있다. 또한, 상속구조의 정의에 필요한 각종정보, 즉, 부모클래스(Parent Node, Base Class)와 자식클래스(Child Node, Derived Class)에 대한 정보를 포인터로 저장한다.

zEvtTbl

해당 클래스가 처리할 수 있는 이벤트와 이들에 대한 처리함수에 대한 정보를 기록할 저장장소이다. 이 자료구조는 기본적으로 테이블의 형태를 갖추고 있으며, 단위 이벤트 정보와 이에 대응하기 위한 처리함수에 대한 포인터를 하나의 쌍으로 관리하고 있다. 대응함수가 없는 이벤트 역시 처리가능함을 나타내기 위해서 처리함수를 NULL로 표현함으로써 같은 구조에 정보를 저장하고 있다.

zAtbNode

클래스가 가지는 속성에 대한 정보를 저장하기 위한 저장장소이다. 속성의 이름과 속성의 타입, 속성의 기본값을 저장한다. 하나의 클래스는 여러개의 속성을 가지므로 이를 저장하기 위한 자료구조는 동적저장구조중 하나인 리스트형태를 갖추고 있다.

zFunctionList

클래스의 행위를 정의하는 것은 사용자가 스크립트를 이용해서 정의한 메소드들이다. 이들 메소드는 스크립트 인터프리터의 해석을 용이하게 하기위해서 특별한 구조와 형태로 저장되는데, 이와 같은 용도로 사용되는 저장장소가 바로 zFunctionList이다. 메소드는 이벤트에 대응하는 핸들러와 다른 객체에 의해 호출되는 단순 메소드로 나뉘는데, zFunctionList는 이들을 똑같은 방식으로 저장하고 관리한다. 또한, zFunctionList는 하나의 클래스에 의해 사유되는 형태가 아니라 모든 클래스에 의해 공유되는 개념으로 설계되었으므로, 여러 클래스의 메소드들이 물리적으로 근접하게 위치하게 된다. 이상에서 설명한 라이브러리 관련 자료구조체들간의 구조를 다음의 〈그림 6〉에 나타내었다.

〈그림 6〉에서와 같이 관련자료구조체들은 각 객체들의 책임에 따라 제공해야하는 써비스의 효율성을 고려하여 각 정보에 대한 접근경로를 설정하고 있다. 즉, 라이브러리 관리자는 사용중인 각각의 라이브러리에 대한 관리책임과 이들을 이용한 서비스의 제공을 수행하여야 함으로, 라이브러리구조에 대한 포인터를 보유하고 있는 등과 같다.

라이브러리관리자의 실제적인 구현인 zLibManager는 다음과 같은 책임을 가지고 있는 클래스이다.

- 실행시간 오브젝트의 생성(Instantiation)
- 신규 클래스의 정의
- 기존 클래스의 정의 변경
- 기존 클래스의 삭제
- 라이브러리 열기
- 라이브러리 저장
- 신규 라이브러리의 생성
- 라이브러리의 변경과 삭제

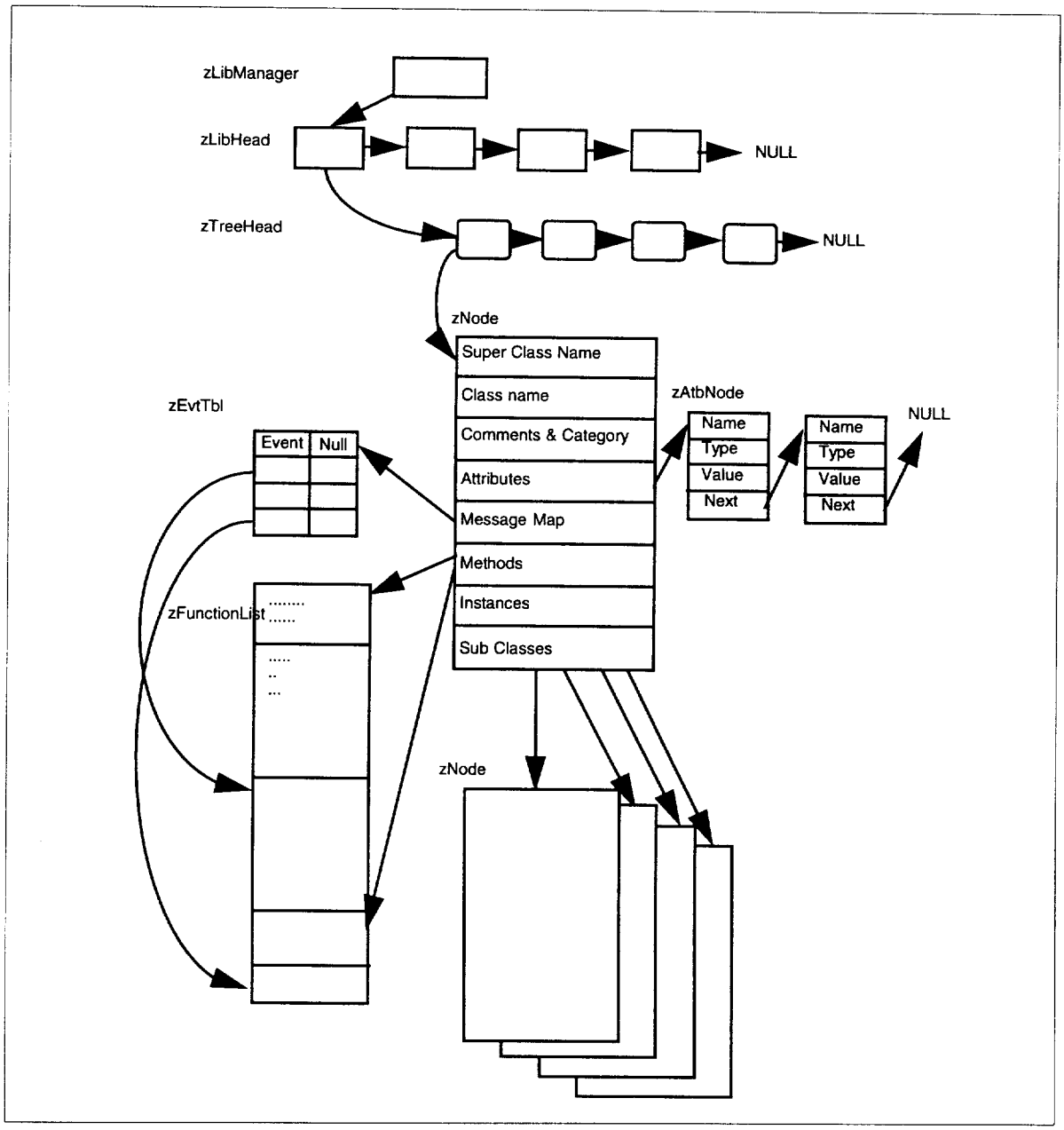
라이브러리관리자의 책임중 대부분은 라이브러리관리자에의해 관리되고 있는 다른 객체내에 구현되어있으며, 라이브러리관리자에는 이들 구현부와의 연결을 구현하고 있는 인터페이스만이 정의되어있다.

4.2 Runtime Instance의 구조

zRuntime Instance 는 라이브러리가 보유하고 있는 클래스의 객체화된 상태를 표현할 클래스이다. 라이브러리가 보유하고 있는 클래스는 정의에 해당하는 것으로 메시지를 받거나 보내는 일이나 각자의 속성값을 가질수 없다. 어플리케이션의 실행중에 존재하는 Runtime 객체가 이와 같은 역할을 수행하는 데, 이 객체의 구조는 〈그림 7〉과 같다.

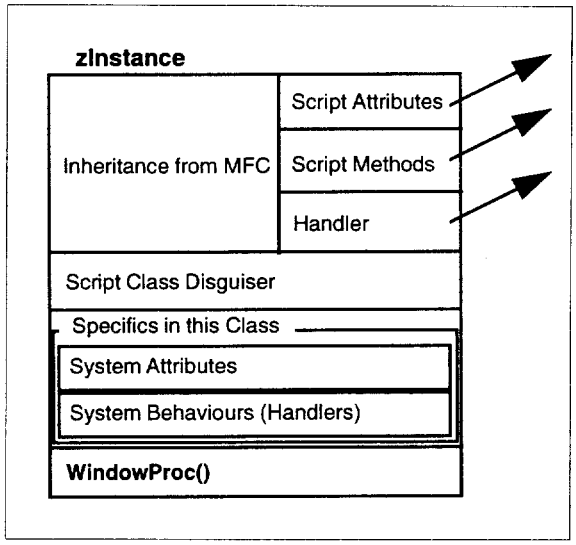
1) Inheritance from MFC

MFC Class Hierarchy내에서 해당 Class의 Base Class로 부터 상속된 C++ Code이다. Windows에서 사용



〈그림 6〉 라이브러리 모듈의 자료구조도

가능한 GUI 관련 Resource들과 이들 각각에 대응하는 MFC의 Class들은 다음과 같다. 개발중인 어플리케이션 빌더에서 제공되는 각종 GUI 용 Class들을 그 형태와 기능에 따라 이상에서 소개된 MFC Class중 하나로 부터 (혹은 두개 이상으로 부터) 상속된 C++ 의 Derived Class로 구현된다. 따라서 사용자에게 제공되는 Class에는 관련 MFC 의 Class로부터 상속된 Code가 존재한다. 이로인해서 각각의 GUI Object는 표준 윈도우의 콘트롤이 가지는 속성을 그대로 가지게되므로 사용자가 사용하기 쉽다.



〈그림 7〉 Runtime 객체의 구조

2) Script Attribute

사용자는 어플리케이션빌더가 제공하는 기본 클래스를 출발점으로 해서 새로운 클래스를 정의할 수 있다. 새로운 클래스의 이름을 정의하고 기본 클래스에는 없는 새로운 멤버 데이타나 새로운 기능(Behaviour, Method)를 정의함으로써 새로운 클래스를 정의할 수 있다. 이와 같이 사용자가 정의하는 멤버데이타는 C++의 클래스와는 다른 형태로 표현되는데, 실행시간중에도 클래스의 정의를 바꾸기 위해서는 C++을 이용한 클래스 콘테이너를 구현하였다. 사용자가 접하는 클래스는 C++로 구현된 클래스 콘테이너에 저장된 사용자관점의 클래스인것이다.

3) Script Methods

스크립트 속성과 마찬가지로 클래스의 행위 또한 클래스 콘테이너에 저장되는데, 이를 스크립트 메소드라 한다. 스크립트속성과 스크립트 메소드는 클래스 콘테이너를 정의하고 있는 C++ 코드에 의해서 작동된다.

4) Handler

사용자가 마우스나 키보드를 이용하여 각종 정보를 입력하면 윈도우즈는 이들을 기정의된 표준 메시지로 전환하여 화면상에 존재하는 해당 윈도우에 전달한다. 이러한 메시지는 Windows Messages, Notification Messages의 두가지로 나눌 수 있다. 이들 메시지를 받는 윈도우는 각 메시지에 대응하여 수행하여야 할 작업을 함수의 형태로 가지고 있는데, 이러한 함수를 핸들러라한다. 핸들러는 C++ 코드로 구현된 것과 사용자가 정의한 스크립트로 정의된 핸들러가 있다.

5) Script Class Disguiser

스크립트 클래스 구현자(Script Class Disguiser)란 사용자가 정의한 클래스간의 상속관계나 정보은닉(Information Hiding), 캡슐화(Encapsulation)등의 객체지향 메카니즘을 가능하게 하는 부분이다. 스크립트 클래스 구현자의 임무는 다음과 같다.

각종 속성값의 설정
각종 속성값에 대한 조회
각종 메시지의 수령 및 해당 핸들러 함수와의 연결

6) Specifics in this class

클래스의 기본적인 형태나 동작은 MFC에서 정의된 것을 따르고 있다. MFC에서 정의된 속성이나 동작방식을 더욱 세분화하여 정의해서 그 사용 대상을 더욱 구체적으로 결정해서 사용자에게 제공하고 있는데, 바로 이와같은 더욱 구체적인 부분에 대한 속성/메소드 정의를 구현하고 있는 부분이 바로 Specifics in this class이다.

가. System Attributes

MFC가 정의하고 있는 속성에 추가된 속성들이다. 이것은 구체적인 클래스의 정의와 사용자 관점에 따라 결정되므로 모든 클래스에 대한 내용을 일괄적으로 설명할 수는 없다.

나. System Behaviours (Handlers)

클래스의 동작에 대한 정의로 클래스마다 다른 내용을 담고 있으므로 일괄하여 설명할 수 없다. MFC가 정의하고 있는 표준 핸들러를 구체화시키는 것으로 사용자에게 제공되는 구체적인 클래스의 행위를 정의하고 있다.

〈표 3〉 GUI관련 Resource들과 MFC의 Class

자 원	MFC 클래스	내 용
Window	CWnd, CFrameWnd,...	어플리케이션의 기본구성단위로 각각의 화면에 해당한다.
Dialog Box	CDialog	사용자의 자료입력, 작업수행상태, 경고 등 사용자와의 의사교환의 수단으로 사용된다.
Text	CStatic	화면을 구성하는 요소로 기능이 없는 단순문장이다.
Frame		토글버튼을 하나의 그룹으로 묶어 주는 기능을 수행한다.
Check Box	CListBox	일련의 값들을 수직으로 나열하여 표시하고 사용자로 하여금 표시된 값중 하나를 선택할 수 있게 한다.
Combo Box	CComboBox	사용자가 클릭하면 리스트박스를 제공한다.
VertScrollBar	CScrollBar	연결된 GUI객체의 내용을 수직으로 Scrolling한다.
HoriScrollBar	CScrollBar	연결된 GUI객체의 내용을 수평으로 Scrolling한다.
Bitmap		Bitmap정보를 화면에 표시한다.
Edit field	CEdit	사용자가 키보드를 이용해서 값을 입력, 수정할 수 있게 한다.
Button	CButton	사용자가 특정기능의 수행을 시작시킬수 있는 의사 표현의 수단을 제공한다.

7) WindowProc()

윈도우즈나 사용자가 스크립트로 전달하는 각종 메시지를 해당 핸들러로 연결시켜주는 기능을 수행하는 부분으로 시스템의 메시지전달체계의 가장 중요한 부분에 해당한다. 이 함수는 상위의 클래스에서 가상함수로 정의되어 있는데, 이를 하위의 클래스에서 오버라이드(Override)하여 구현한다.

4. 결론

개발중인 어플리케이션빌더는 어플리케이션 구성의 근간을 형성하는 프레임워크부분과 각 부문별 특화된 기능을 담고 있는 라이브러리들로 구성된다. 프레임워크부분은 다시 그래픽환경을 지원하기 위한 그래픽 에디터(Graphic Editor)와 윈도우즈 어플리케이션을 위한 메뉴 생성기, 다중문서 편집기, 스크립트 해석기, 이벤트 관리엔진과 제공되는 시스템 라이브러리와의 인터페이스를 제공하고 사용자의 라이브러리 사용을 지원할 라이브러리 관리자등으로 구성된다. 라이브러리는 각 부문의 기능을 담을 컨테이너로 가장 기본적인 그래픽 기능 정의를 위한 기초 라이브러리가 기본적으로 제공되고 각 부문별로 특화된 기능은 차후에 제공될 예정이다.

각 부문별 기능의 확장은 라이브러리를 통해서 이루어지도록 되어 있으므로, 개발시스템의 프레임워크의 설계에 있어서 가장 큰 비중을 두었던 부분이 바로 라이브러리를 통한 확장가능성이었다. 이를 위해서 라이브러리가 갖추어야할 각종 특성(Attributes)과 설계기준(Criteria)을 정의하였으며, 정의된 라이브러리의 설계기준을 토대로 라이브러리 관리자의 기능을 설계 하였다. 또한, 기본적인 GUI(Graphic User Interface)의 정의를 위한 클래스들로 이루어진 라이브러리의 설계에도 이상의 설계기준들을 참고하였다. 코드의 재사용성과 라이브러리 자체의 확장성, 사용자의 편의성등이 라이브러리설계의 기준이였다.

【참고문헌】

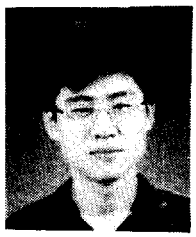
[1] Brad J. Cox, Object-Oriented Programming : An Evolutionary Approach, Addison-Wesley, 1986

[2] 김형주, 알기쉬운 객체지향시스템, 동아출판사, 1993

[3] Bertrand Meyer, "Genericity versus Inheritance", in *Proceedings of OOPSLA '86*, pp. 391-405, 1986

[4] Timothy D. Korson, "Evaluating and Constructing

Class Libraries", in Tutorial Note of OOPSLA '94, 1994.



이치근
서울대학교 산업공학과를 90년에 졸업하고 한국과학기술원 산업공학과에서 92년 석사학위를 취득하였다. 현재 삼성데이타시스템의 정보기술연구소에서 전임연구원으로 재직중이며, 관심분야는 소프트웨어재사용, 객체지향시스템, 인공지능, Geometric Modeling등이다.

송희석
서울대학교 산업공학과를 86년에 졸업하고 한국과학기술원 산업공학과에서 88년 석사학위를 취득하였다. 현재 삼성데이타시스템의 정보기술연구소에서 책임연구원으로 재직중이며, 관심분야는 인공지능, 공정계획수립, 객체지향시스템, 통계분석 등이다.