디자인 패턴을 이용한 JiKU/XML 객체지향코드 생성기 설계 및 구현

선 수 균†

요 약

단일 시스템으로 개발된 기존 코드 생성 시스템은 분산 환경 상에서의 개발자나 유지 보수자들의 디자인패턴 정보를 공유하는 것이 원활하지 못했다. 본 논문에서는 웹환경 기반인 XML과 디자인 패턴을 이용한 JiKU/XML 객체지향코드 생성기를 설계하고 구현한다. 이것은 디자인 패턴 구조를 XML 코드로 변환하기 위해 UML을 이용하였으며 UML로 표현된 설계 정보를 XML 코드로 생성하기 위해 PIML 구문법에 맞게 코드생성을 한다. 이 JiKU/XML 객체지향코드 생성기는 열 단계로 코드를 생성하고 설계정보가 XML 코드로 생성되므로 웹 환경에 쉽게 적응시킬 수 있다. 기존의 생성기인 F77/J++생성기의 단점을 보완했으며 UML과 패턴 정보를 이용하기 때문에 설계의 표준화를 이룰 수 있다. 기존 시스템과 적용사례를 비교 분석하여 본 연구에서 제안한 생성기가 더욱 향상된 기능을 제공한다.

A Design and Implementation of JiKU/XML Object-oriented Code Generator Using for Design Pattern

Su-Kyun Sun†

ABSTRACT

The present code generation system, developing based on single system, is not easy for developers or maintenance men to share pattern design information in distribution environment. So in this paper, we design and implement XML as basis of web environment, and JiKU/XML object-oriented code generator using pattern design. We use UML to change pattern design to XML code, and create code, suitable to PIML command, to generate design information designed by UML into XML code. This JiKU/XML Object-oriented Code Generator makes 10-step codes, and can be easily applied to web environment. It complements the disadvantage of present generator, F77/J++, and makes standardization of design because it uses UML and design pattern information. We compare it with present system by implement cases, and as a result, generator suggested in this study gives more effective function.

키워드 : 객체지향 코드 생성기(Object-Oriented Code Generator), JiKU/XML(Join-Integrated-KU/eXtensible Markup Language), JiKU/XML 객체 관리 저장소(Object Management Repository), 디자인패턴(Pattern Design), XML(eXtensible Markup Language), 관계정보(Relation_information), UML(Unified Modeling Language)

1. 서 론

객체 지향 소프트웨어의 개발을 위한 객체 지향 방법론은 1995년 이후로 Rumbaugh의 OMT(Object MT), Jacobson의 Objectory, Booch의 OOA/OOD 이들의 방법을 취합한 UML(Unified Modeling Lanague)등이 제시되었다[5]. 본 연구의 설계는 패턴 구조를 가시적으로 모델링하기 위하여 UML기반의 패턴 정보를 이용하였다. 기존의 시스템으로는 인터넷상에서 널리 알려져 있는 OOTher, Rational Rose, Show CASE 도구등의 시스템이 있다[10]. 기존의 코드생성 시스템은 단일 시스템을 기반으로 개발되어 분산환경 상에서의 개발자나 유지보수자들의 디자인패턴 정보를 공유하는 것이 원활하지 못했다. 즉, F77/J++ 생성기(기존 생성기)는 설계정보를 XML 코드로 생성하지 못하므로 웹 환경에 쉽게 적응 할 수 없었다[7, 13].

따라서 본 논문에서는 디자인패턴을 이용한 객체지향 시스템 개발환경의 설계정보 재사용성을 높이고자 웹기반 통합 시스템인 JiKU/XML 객체지향 코드생성기를 설계하고 구현한다. 이 모델은 통합 객체 관리 모델의 단점을 보완하기 위해 XML과 PIML 구문법을 사용한다. 그리고 객체를 클래스 정보와 역할 정보를 분류하여 메타 모델함으로써 UML로 도식화가 가능하다. 그러므로 객체지향 코드 생성이 F77/J++ 생성기 보다 XML 코드 변환이 쉽고 메타 모델링이 가능하여 객체 내부까지 UML로 표현할 수 있다.

† 종신회원 : 동원대학 e-business과 교수
논문접수 : 2002년 2월 18일, 심사완료 : 2004년 2월 3일

본 논문에서 제안한 JiKU/XML 객체지향 코드 생성기는 열 단계로 나뉘어져 각 단계별로 거치는 동안 기존 시스템의 단점인 분산환경에서 개발자나 유지보수자들의 패턴 정보를 공유함으로서 객체 설계가 가능하다. 초기 단계에서는 디자인패턴 구조를 XML코드로 변환하기 위해 UML이라는 표준화된 표기법에 따라 도식화한다. 특히, 연결관계 상호 관련기를 두어 객체 내부 관계와 서로 관련성을 개발자가 쉽게 파악하여 재사용함으로서 개발자는 응용에 관련된 부분만을 개발할 수 있도록 지원한다. 이에 따라 객체지향 코드 생성기는 UML로 표현된 설계 정보를 여러 패턴을 이용하여 XML 코드로 생성하기 위해 PIML 구문법에 맞게 객체지향 코드가 생성된다. 또한 성능 평가는 제안한 모델을 중심으로 기존의 생성기인 F77/J++ 생성기와 비교 분석하여 적용 사례를 제시한다. JiKU/XML 객체지향코드 생성기는 효율적인 비교분석을 위해 기존 생성기에서 적용시킨 방법과 같은 방법으로 성능 평가를 실시한다. AMOSS 코드를 적용하여 그 결과 객체지향코드 생성기가 기존 생성기 보다 쉽게 객체지향 코드에 접근이 가능하다. 왜냐하면 순서 비교 알고리즘로 패턴 관계 정보를 쉽게 얻을 수 있어 객체코드 생성이 쉬우며 연결관계 상호 관련기에 있는 템플리트를 활용했기 때문에 개발자가 쉽게 코드 생성에 접근할 수 있어 향상된 기능을 제공할 수 있다. 즉, 이 모델은 UML과 XML을 사용하기 때문에 데이터 교환에 표준을 만들 수 있어 향후 통합 모델 구현과 소프트웨어 생산성을 향상시킬 수 있을 것으로 기대되는 모델이다.

2. 관련 연구

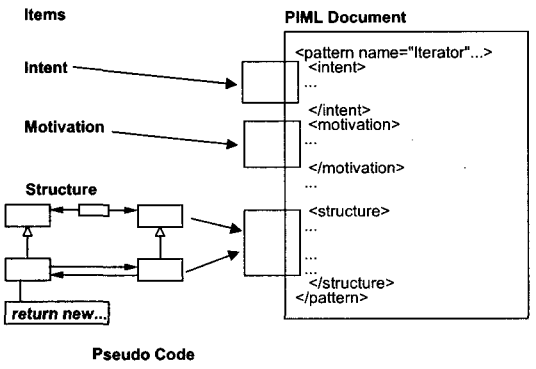
2.1 디자인패턴

디자인패턴은 객체지향 개발 연구에서 소프트웨어 설계의 일반적 문제를 시간에 관계없이 해결하기 위한 중요한 대안이며, 설계와 구조의 재사용을 쉽게 해준다. 일반적인 패턴들은 SmallTalk 언어를 사용해 Model-View-Controller (MVC) 프레임워크를 만들었다. 그것은 사용자 인터페이스 문제를 3가지 부분으로 나누었다. Data Model과 프로그램의 계산코드를 담고 있는 부분, View와 사용자 인터페이스를 표현하는 부분, Controller와 사용자와 View간의 상호작용을 담당하는 부분으로 나누어진다. 디자인패턴에 대한 연구는 1990년대 초반 Erich Gamma가 GUI 어플리케이션 프레임워크(ET++)에 패턴을 사용함으로써 시작되었다. Design Pattern(Gof)에서는 잘 알려져 있고 여러 개발 영역에서 쓰일 수 있는 23개의 디자인패턴을 기술하고 있다[3].

2.2 PIML 및 기존 생성기

디자인패턴은 각각의 클래스를 포함하는 구조로서 표현되며, 디자인패턴을 사용하면서 소스코드를 이해하거나 설계를 지원해 준다면 소프트웨어 개발에 효과적일 것이다. Mika Ohtsuki는 SGML 프레임워크를 사용하는 디자인패턴을 설명하기 위해 PIML(Pattern Information Markup Language)를 제안하였다[9].

디자인패턴에서의 PIML 설명은 세 부분으로 구성되어있다. PIML의 구성은 "Motivation"과 같은 설명 텍스트 부분, 구성요소 클래스와 클래스간의 관계같은 구조 정보 및 클래스간의 행위를 나타내는 "pseudo-code"로 구성되어 진다. 설명 텍스트와 구조 정보는 SGML 엘리먼트들을 사용하여 설명되어지며, "pseudo-code"는 간단한 프로그래밍 언어로 설명되어지거나 소스코드생성을 위해 사용되어 진다. 디자인패턴의 이름은 속성으로서 "pattern" 엘리먼트로 구성되어진다. "intent"와 "motivation" 항목은 (<intent>,</intent>)와(<motivation>,</motivation>)과 같은 한 쌍의 태그로 둘러 싸여져 있다[15]. 구조를 설명한 클래스 다이어그램은 (<structure>, </Structure>)와 같은 한 쌍의 태그로 중첩 설명으로 변환되어 진다.



(그림 1) 디자인패턴 항목과 PIML 설명 엘리먼트와의 대응관계

F77/J++ 생성기 구현단계는 아홉 가지의 재공학 전략으로 이루어지며 변형과정으로 나눌 수 있다[13]. 이 생성기는 자동 코드 생성 기능으로 노후코드(Legacy code)를 새로운 코드로 생성하여 소프트웨어 개발자가 새로운 개념을 쉽게 습득할 수 있도록 도움을 준다. F77/J++ 생성기는 포트란 언어를 입력하여 자바로 구현할 수 있으나 웹 환경에서 쉽게 적용할 수 없다. 이 시스템의 장점은 모듈별로 쉽게 적용시킬 수 있으며 자동코드 생성 기능으로 시스템 소프트웨어 개발 시간을 단축시킬 수 있으나, 복잡한 코드 처리가 불가능하고 객체지향 언어를 효율적으로 처리할 수 없다. 또 다른 다중언어 코드 생성기시스템은 XML 코드를 트리 정보로 변환하기 위해 MS-XML Parser를 사용한다[1, 12]. 이 생성기는 트리 구조로 변환된 설계정보를 추출하기 위해 XML-QL 질의어를 사용한다[8].

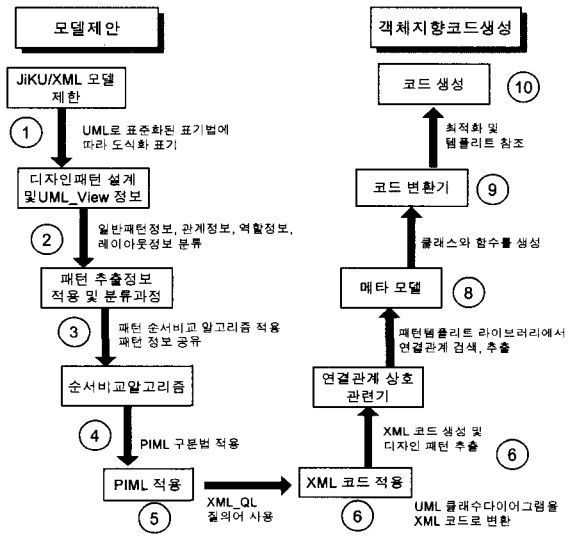
NetFree는 컴포넌트 프레임워크로, 컴포넌트를 설계하고 구현할 수 있는 기능을 제공한다. 또한 NetFree는 미들웨어로서 플랫폼(OS), 프로토콜, 프로그램 언어, 시스템 아키텍

처의 장벽을 느끼지 못하도록 하며, 사용하는 사람은 기초적인 프로그램 언어만 알고 있으면 고난도의 시스템과 신뢰성 있는 소프트웨어 컴포넌트를 만들 수 있다[11].

3. JiKU/XML 객체지향코드생성기 설계

본 논문에서는 JIKU/XML 통합 모델을 중심으로 생성기를 설계하는데 열 단계를 거치면서 각 단계의 기능을 사용하여 설계한다. 모델 중 하나인 생성기 설계는 자료 추상화, 예외처리, 객체지향 프로그래밍을 지원하는 C++을 사용하기 위한 테이블을 이용하여 코드생성기를 설계한다. 따라서 열 단계를 거쳐 노후 코드를 객체지향언어로 변환하는 JIKU/XML 객체지향코드 생성기를 설계하고 구현한다.

JIKU/XML 객체지향코드 생성기의 전체 시스템 구조는 (그림 2)와 같으며 열 단계로 구성되고 있다.



(그림 2) JiKU/XML 객체지향코드 생성기 구조

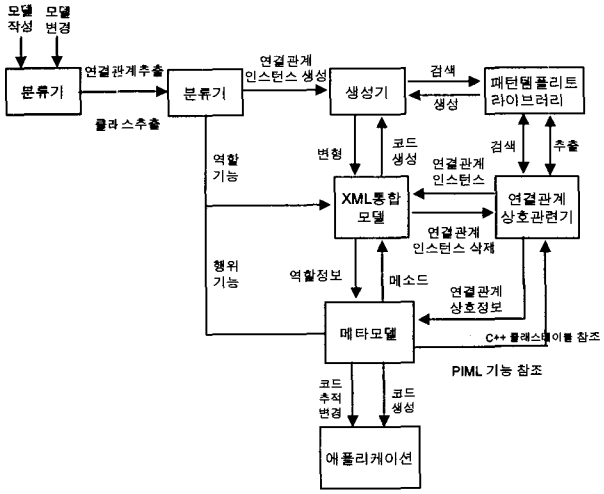
JiKU/XML 모델 제안은 객체지향 생성기를 구현하기 위한 첫 번째 단계로 객체 관리 저장소와 XML 검색기, 연결관계 상호관련기, XML 템플리트 라이브러리와 밀접한 관계를 이루고, 분류 방법 설계도 포함하고 있다. 이 중에서 객체 관리 저장소는 설계정보를 추출한다. 추출한 곳을 패턴별로 분류하는데, 이 단계는 디자인패턴 설계 정보를 분류하고 패싯분류 방법을 사용한다. 패싯은 패턴의 기본적인 기능과 목적 그리고 패턴을 구현하고 실행할 수 있는 실제 상황을 기술할 수 있도록 도메인 종속여부, 패턴이 속한 영역, 적용 목적, 적용 범위의 4개 패싯을 포함하도록 정의한다. 이것은 설계패턴을 UML로 모델링하고 패턴 모델 시키기 위하여 패턴 구조를 메타 데이터로 저장하여 패턴 메타 데이터베이스로 설계한다. (리스트 1)은 객체지향 코드 생성기의 처리기를 알고리즘으로 나타낸 것이다.

```
//처리기 알고리즘
    Repeat
        increase count
          If point is in any pattern
          Load pattern_name Perform data consistency
        Endif
    Until all total pattern count
     Repeat
      If exsit pattern
        If pattern_name ∈ pattern database
         // 순서비교 알고리즘 수행
          Write source linked pattern_name
            Perform sequence-relation
            Perform pattern-abstrat
      Endif Endif
    Until
    Repeat
     If exsit PIML// PIML 수행, 메타모델링, UML
        Perform PIML Document
          <intent> <motivation> <pattern>
            <structure> <relation> <roles>
              <operation>
        Perform Meta_Modeling
        Perform iLink-Relation/* 연결관계 상호*/
        Write Meta Database
        Perform UML
         set cnt : = 0//초기화
        Repeat
         add 1 to count
         If exist inheritance
           Repeat
           If exsit subClass
              x,y position move to/* UML 좌표이동*/
              center position of class box of top
        draw line between classes /* 클래스 상속관계 */
        draw line arrow /* 관계화살표 좌우 그리기 */
          End if
          Until cnt < tablesize
          End if
        Until ∀ total class count
      Perform UML-draw/* UML 도식화 */
      Perform C++create-process /* 코드 생성*/
     Write create code
     Endif
Until
```

(리스트 1) 객체지향코드 생성기의 처리기 알고리즘

첫 번째 설계는 데이터 연속성(data consistency)을 확실하게 하고 데이터를 전반적으로 사용할 수 있도록 포괄적인 데이터를 만든다. 또한 패턴 정보를 분류하고 UML로 표준화된 표기법에 따라 도식화한다. 두 번째 설계는 관계와 관계정보 알고리즘을 분류하여 패턴 순서 비교 알고리즘을 적용한다. 세 번째 설계는 과다한 코드를 제거하는 것과 PIML 적용이다. 즉 불필요한 코드를 제거하고 알고리즘을 간단하게 하여 패턴을 PIML로 변환한다. 네 번째 단계는 연결관계 상호 관련기 적용한다. 이것은 모든 논리적인 (logical) 변수를 변환시키고 제어의 건실한 사용이 확실하게 한다. C++로 변환하는데 도움이 되고 다른 loop 루틴을 만드는데 쉽게 해 준다. 예로써 "IF (<expr>) <statement>"가 "IF(<expr>) THEN <statement> ENDIF"으로 대신함

으로써 다음의 C++로 변형이 쉽게 만들어질 것이다. 이것을 가능하게 하는 것은 패턴템플리트 라이브러리가 있기에 가능하다. 또한 C++ 클래스 테이블을 참조해서 여러 루틴을 참조한다.



(그림 3) 연결관계 상호 관련기 구조

(그림 3)은 연결관계 상호 관련기의 상호 구조를 나타내고 있다. 다섯 번째 설계는 클래스정보와 역할정보 설계 및 메타 모델링이다. 이 단계는 디자인패턴 설계정보 및 패턴정보 적용이다. 이 패턴정보는 클래스, 오브젝트, 상속을 지원하며 강력한 형 체크를 지원하여 자료 추상화, 예외처리를 포함한다.

본 논문에서는 여러 패턴 중 Iterator패턴을 중심으로 코드 생성을 한다.

이 관계정보에서는 ConcreteIterator패턴, Iterator와 ConcreteAggregate, Client, 그리고 Aggregate 클래스간의 관계를 추출할 수 있으며 역할정보에서는 각 클래스의 오퍼레이션과 접근권한, 반환값 등의 정보를 추출할 수 있다. 그리고 레이아웃 정보에서는 패턴 내의 클래스 위치 정보를 추출할 수 있다. 구조적인 정보는 패턴 내의 각 클래스와 클래스 내의 오퍼레이션에 대한 정보, 패턴 내의 클래스들의 위치정보가 표현되어진다. 여섯 번째 설계는 메타데이터베이스 저장이다. 일곱 번째 설계는 디자인 패턴을 클래스 객체(object) UML 표현이다. 여덟 번째 설계는 클래스와 함수 생성이다. 여기에서는 클래스와 함수를 새로운 코드에 맞게 생성한다. 아홉 번째 설계는 마지막 단계로 최적화 및 코드 생성이다.

4. JiKU/XML 객체지향코드생성기 구현

본 장은 JIKU/XML 통합 모델을 중심을 이용하여 객체지향코드 생성기를 구현한다. 이 생성기 설계에서 중요한 알고리즘을 중심으로 구현과정을 설명한다. 여기서 가장 중요한 알고리즘은 패턴 순서비교 알고리즘과 패턴 추출 알고리즘 및 연결관계 상호 관련기 적용이다. 이것은 생성기를 구현한 핵심이 된다. 객체지향코드 생성기는 이 알고리즘을 적용하여 패턴을 추출하여 순서 비교함으로서 가장 적합한 패턴을 찾을 수 있다. 패턴 추출 정보 적용 및 분류 과정은 UML로 도식화한 표준화된 뷰에서 생성된 디자인패턴 설계 정보를 파싱하여 XML-QL 질의어에 의해 특정 정보가 쉽게 추출된다. 이 단계로 추출된 관계 정보를 JiKU/XML 객체 관리 저장소에 저장하여 활용할 수 있다.

이 방법은 다음과 같다. 객체와 객체간, 또는 객체 내부에 있는 클래스 사이의 순서는 (α, β)와 같은 순서쌍으로 표현 할 수 있다. 클래스 α에서 β사이로 관계가 존재한다는 의미이다.

〈표 1〉 클래스다이어그램에서의 관계 표시

관 계	기 호	저장표기
연관(association)	→	S
집합(Aggregation)	—◆	A
복합(Composite)	—◇	C
의존(Dependency)	⇢	D
일반화(Generalization)	—▷	G
실체화(Realization)	⇢▷	R

이 과정은 만약 클래스다이어그램에서 존재하는 클래스 사이의 관계가 클래스 α와 β사이의 일반화 관계(Generalization : G)가 존재할 때, $(\alpha, \beta)_G$와 같이 표기할 수 있다. 이것을 저장소에 저장할 때는 $G(\alpha$클래스 위치, β클래스 위치)에 저장된다. 이와 같은 도식화로 UML에서 클래스 사이에 존재하는 여러 가지 관계를 나타낸 것이 <표 1>의 저장표기이다.

이러한 클래스 다이어그램에서 α클래스와 β클래스를 이용하여 UML의 관계정보를 효율적으로 알 수 있다. 이런 결과를 알 수 있는 알고리즘이 패턴 순서비교 알고리즘이다. (리스트 2)는 이 알고리즘을 나타낸 것이다.

패턴 순서비교 알고리즘과 패턴 추출 알고리즘으로 추출된 정보를 가지고 패턴 정보를 공유한다. 이것은 클래스간의 관계를 비교함으로서 관계 정보를 파악하여 순서 기준 패턴을 생성한다. 앞에서 언급한 두 알고리즘을 활용하여 Iterator 패턴으로 구현결과를 설명하고PIML 구분법과 클래스 다이어그램을 XML 코드로 변환한다. 이것을 UML로 표현된 설계정보로 변환하는데 PIML 구문법을 사용하고 XML 코드로 변환한 것을 데이터베이스에 저장하게 되는 것이다.

```
1 : Repeat
 2 : set Da : /* 디자인패턴 테이블의 마지막레코드 */
 3 : set Tb : / * 테스트 패턴테이블의 마지막레코드*/
 4 : set (α, α) : /* 디자인패턴 클래스의 관계 */
 5 : set (β, β) : /* : 테스트 패턴 클래스의 관계*/
 6 : set n, m : /*T 테이블 모든 클래스와 비교
 7 : Until total_count
 8 :  If design pattern of index Not exist
 9 :  while ( D1 ← Da ){//D테이블 모든클래스와 비교
10 :   while ( T1 ← Tb ){ // 클래스 순서 비교
11 :     If (αn, αn+1)와 (βm, βm+1) 순서를 비교 동일then
12 :                 //클래스 관계 비교
13 :      if (αn, αn+1)∈R 와 (βm, βm+1)∈R 관계비교동일
14 :           then
15 :              동일 패턴 → 새로운 테이블 저장
16 :          else
17 :              동일 패턴 아님
18 :      endif
19 :          else
20 :      동일 패턴 아님
21 :   endif
22 :  }
23 : }
24 : Endif
25 :    Readindex from repository
26 :  Else/* index exist */
27 :    Repeat
28 :     retrievalindex-id
29 :    Until
30 :    Endif
```

(리스트 2) 패턴 추출 알고리즘

이 장에서는 패턴 순서비교 알고리즘과 패턴 추출 알고리즘을 사용하여 Iterator 패턴을 PIML로 변환한 구현 결과가 (리스트 3)와 (리스트 4)이다. 그리고 XML 코드로 변환된 패턴정보는 패싯검색이나 이름검색에 의해 검색되어 파싱된다. 또한, XSL과 DTD를 통하여 설계정보를 확장할 수 있으며, 트리 형태로 변환한 후 Node-ID로 연결됨으로 코드 생성을 쉽게 찾을 수 있다.

```
<?xml version = '1.0' ? encoding ="EUC-KR"?>
<?xml : stylesheet type="text/xsl" href="Iterator.xsl"?>
<LIBRARY>
    <pattern name='Iterator'>
      <structure>
        <relations>
        <inheritance origin='Concreteiterator'
                        target='Iterator'></inheritance>
        </relations>
        <roles>
        <role syslabel='Aggregate' abstract='abstract'>
        <operations>
        <operation constuctor='constructor' override='do'
access='public' return='nonreturn' syslabel='Creteleterator'>
                        </operations>
        </role>
        <role syslabel='Client' abstract='concrete'>
        </role>
        <role syslabel='Iterator' abstract='abstract'>
        <operations>
        <operation constuctor='constructor' override='do'
access='public'return='nonreturn' syslabel='First'></operation>
        <operation constuctor='constructor' override='do'
access='public'return='nonreturn' syslabel='Next'></operation>
        <operation constuctor='constructor' override='do'
access='public'
        </operation>
        </role>
        <role syslabel='Concreteiterator' abstract='abstract'>
        <layout rows='2' columns='3'>
        <box name='Aggregate'row='1'column='1'></box>
        </layout>
        </structure>
        </pattern>
  </LIBRARY>
```

(리스트 3) Iterator 패턴의 PIML 및 XML 변환 구현 결과

```
<?xml version="1.0" encoding="euc-kr"?>
<xsl : stylesheet xmlns : xsl="http://www.w3.org/TR/WD-xsl">
<xsl : template match="/">
<html>
    <head><title> iterator </title>
      <style>
       .com{color : blue ; }
      </style>
    </head>
    <body>
<xsl : if test="parant[.$eq$ 'S']">
    <font><xsl : value-of select="parat"/>S(0,*)</font></xsl : if>
<xsl : if test="parant[.$eq$ 'A']">
    <font><xsl : value-of select="parat"/>A(0,*)</font></xsl : if>
<xsl : if test="parant[.$eq$ 'C']">
    <font><xsl : value-of select="parat"/>C(0,*)</font></xsl : if>
<xsl : if test="parant[.$eq$ 'D']">
    <font><xsl : value-of select="parat"/>D(0,*)</font></xsl : if>
<xsl : if test="parant[.$eq$ 'G']">
    <font><xsl : value-of select="parat"/>G(0,*)</font></xsl : if>
<xsl : if test="parant[.$eq$ 'R']">
    <font><xsl : value-of select="parat"/>R(0,*)</font></xsl : if>
.....
</body>
  </html>
</xsl : template>
</xsl : stylesheet>
```

(리스트 4) Iterator 패턴의 XSL로 변환 구현

따라서 PIML로 변환된 패턴정보를 파싱함으로서파싱된 패턴 정보는 XML-QL 질의어에 의해 특정 정보가 추출되며 추출된 정보는 패턴 템플리트 라이브러리에 맵핑된다. 패턴 메타정보는 패턴을 UML로 모델링하는데 필요한 정보를 포함하고 있다. 설계패턴 구조를 생성하기 위해 패턴 등록시 그 구조를 UML로 모델링하여 구성 정보를 데이터베

이스화하였다. 특히, 연결관계 상호 관련기의 구현은 클래스정보와 클래스 사이의 관계정보가 객체저장소에 저장된다. 이때 이들 정보는 객체지향 다이어그램 도구를 위한 정보로 변환시켜야한다. 이들 정보는 클래스 정보와 클래스 사이의 관계정보, 기호정보, 위치정보, 다이어그램의 내부 구조 정보를 통합한다.

메타 모델링을 수행하는데 크게 세 가지가 있다. 그 하나가 다이어그램이고 둘은 패턴이며, 마지막으로는 컴포넌트이다. 이들 객체는 클래스 객체의 서브클래스로 모델링된다. 클래스 객체는 부모 버전과 아들 버전의 관계를 형성시킬 수 있으며, 연결 관계 정보 및 해당 패턴 정보를 함께 형성한다. (리스트 5)는 연결 상호 관련기의 연결 관계 표현이다.

```
abstract class Object
{
    attribute :
        object_id Oid ;
        version_number Number ;
        parent_version Oid ;
        child_version setof Oid ;
        relation_version setof Oid ;
pattern_version setof Oid ;
    method :
        derive() returns Object ;
        get_parent_version() returns Object ;
        get_child_version() returns Set<Object> ;
        get_relation_version() returns Set<Object> ;
        get_pattern_version() returns Set<Object> ;
class_method :
        new() returns Object ;
}
```

(리스트 5) 연결 상호 관련기의 연결 관계표현

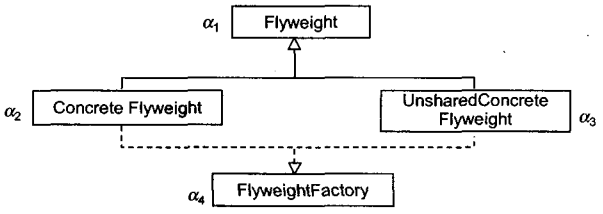
클래스 Diagram은 다이어그램의 원본 내용을, 클래스 Form은 양식의 원본 내용을, 클래스 Pattern은 패턴의 원본 내용을, 클래스 Component는 컴포넌트의 원본 내용을 해석없이 저장한다. 클래스가 컴포넌트 내부에는 컴포넌트의 형상 관리를 위한 속성과 메소드가 포함되어 있다. 속성 parent_component는 해당 컴포넌트를 포함하고 있는 상위 레벨의 컴포넌트를 표현하고, 속성 child_component는 해당 컴포넌트가 포함하고 있는 하위 레벨의 컴포넌트를 표현한다. 이러한 메타모델링 결과를 UML로 표현할 수 있다. 객체 내부의 메타모델링에서는 양식, 패턴이나 다이어그램의 내부 내용 구조를 세부적으로 표현하거나, 특정 양식이나 다이어그램간에 존재하는 종속(dependency) 관계를 표현할 수도 있다. 따라서, 클래스 다이어그램과 상태 다이어그램 사이에는 종속 관계가 성립한다. 즉, 역할과 행위도 포함시킬 수 있으며, 클래스 다이어그램에 속한 각 클래스들은 하나의 상태 다이어그램과 연결되어 있다.

```
class Diagram : Product
{
    attribute :
    content Graphic ;
}
class Form : Product
{
    attribute :
        content Text ;
}
class Pattern  : Product
{
    attribute :
        content Text ;
}
class Component : Product
{
    attribute :
        content BLOB ;
        parent_component Oid ;
        child_component setof Oid ;
        relation_component Oid
pattern_component Oid
    method :
        derive() returns Object ;
        get_parent_component() returns Object ;
        get_child_component() returns Object
        get_pattern_component() returns Object
        Set<Object> ;
}
```

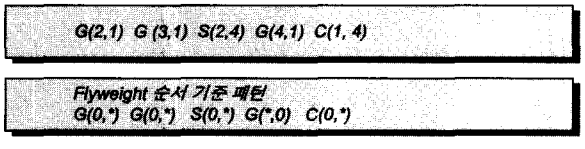
(리스트 6) 다이어 그램, 양식, 패턴, 컴포넌트의 상속 관계

클래스 다이어그램이 삭제되면 앞에서 언급한 역할 포인터와 행위 포인터에 의해서 이에 대한 상태 다이어그램들도 삭제되는데 자동적으로 이루어지도록 구현한다. 즉, 연결 관계상호 관련기는 객체 내부 관계와 서로 관련성을 개발자가 쉽게 파악할 수 있어 쉽게 구현할 수 있도록 지원한다.

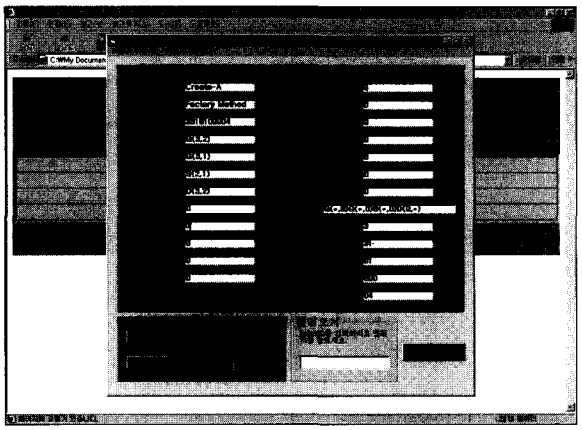
따라서 이러한 적용방법을 사용하여 플라이웨이트 패턴을 적용한것 결과 화면이 (그림 4)와 (그림 5)이다. 이 정보는 메타 데이터베이스에 저장하여 패턴을 검색하고 패턴을 재사용할 수 있다. 또한 이 정보를 활용하여 최적의 패턴을 추출해 내는데 사용된다.



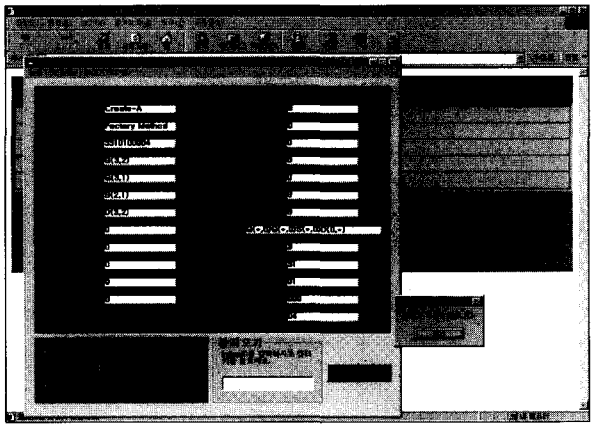
(그림 4) 플라이웨이트 패턴



(그림 5) 패턴 순서 비교 알고리즘을 적용한 결과화면



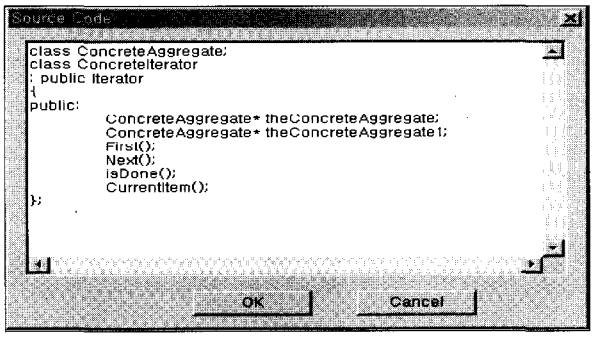
(그림 6) 패턴의 순서 기준 패턴 생성 구현화면(1)



(그림 7) 패턴의 순서 기준 패턴 생성 구현화면(2)

연결관계 상호 관련기적용은 메타모델, 패턴 템플리트라이브러리와 함께 연결하여 상호 맵핑되어 관련정보를 공유한다. 이 연결관계 상호 관련기를 활용하는데 PIML로 변환된 패턴정보를 파싱되며 파싱된 패턴 정보는 XML-QL 질의어에 의해 특정 정보가 추출된다. 추출된 정보는 연결관계 상호 관련기와 C++ 코드생성 템플릿을 통하여 패턴 템플리트 라이브러리에 맵핑된다.

코드 생성은 C++ 코드생성 알고리즘과 패턴 템플리트와 밀접한 관계를 형성한다. 파싱된 패턴 정보는 PIML 문서 내의 특정항목을 연결 관계 상호관련기로 추출한다. 추출된 정보는 클래스 테이블 저장구조에 맞게 입력되어 코드생성기 내의 각 코드 테이블과 C++ 패턴 테이블에 의해 소스코드가 생성된다. 변환된 코드는Iterator 패턴의 C++ 코드 변환 소스코드로 생성하는데 이것이 (리스트 5)이다. 이것은 Iterator 패턴 내의 Aggregate 클래스에 대한 각각의 헤더파일과 실행파일 소스 코드를 나타내고 있다. (그림 8)은 Client 클래스와 관계를 가지고 있는 Iterator 클래스와 Aggregate 클래스를 선언하고 Client 클래스가 상속을 받고 있는 일반 클래스인 Aggregate와 Iterator 클래스를 public으로 선언한 결과화면이다. 따라서 사용자가 이름검색에 의해 Iterator 패턴을 검색하여 코드생성 언어로 C++ 언어를 선택하면 코드생성이 되어 Iterator 패턴의 C++ 코드가 생성된다. 생성된 파일 중에서 ConcreteIterator.h 파일을 선택하여 소스코드를 살펴 볼 수 있다.

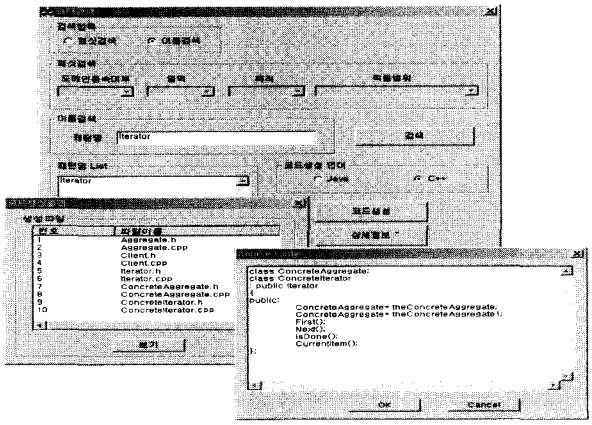


(그림 8) ConcreteIterator.h 소스코드 결과 화면

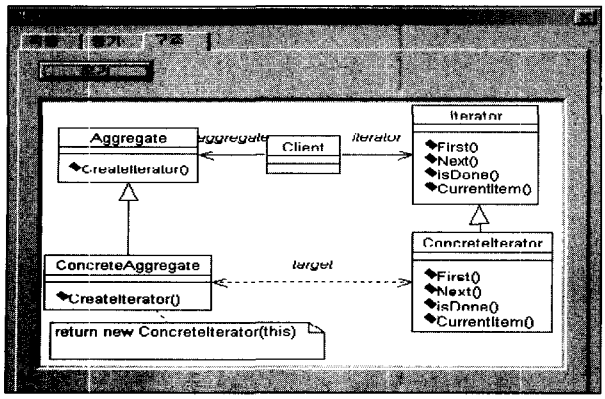
```
#include "ConcreteAggregate.h"
#include "ConcreteIterator.h"
ConcreteIterator :: First(){
}
ConcreteIterator :: Next(){
}
ConcreteIterator :: isDone(){
}
ConcreteIterator :: CurrentItem(){
}
```

(리스트 7) ConcreteIterator.cpp 소스코드

(리스트 7)는 클래스 선언으로 ConcreteIterator 클래스와 관계를 가지고 있는 ConcreteAggregate 클래스를 선언해 주며, 각각 해더화일에 속해 있다. 이것은 Iterator 패턴 내의 ConcreteIterator 클래스에 대한 헤더 파일과 실행 파일 소스 코드를 나타내고 있다.



(그림 9) ConcreteIterator.h 코드생성 결과 화면 뷰



(그림 10) Iterator 패턴의 UML 구현화면

```
while(ClasslistTableNUM) //연결관계 상호관련기의 파일(h. .c++)
생성
{
        make file "class name.h"
        make file "class name.cpp"
}
open classname.h //해더 함수코드 생성
if (inhertance) //inheritance 모드 확인
{
if(class.list.inheritance == classname.relation)
  #include "inheritance.parent.name.h"

if(pseudoccode.operation)
   class.pseudoccode.operation ;
   class classname
 : access.public inheritance.parent.name //inheritance
{
        opertion(public.private/protected).access
        while(refernce.no)
   refernce.Parent*thereferenace.Parent ;
          refernce.no-- ;
        while(opertion.no)
   opertion.name(opertion no) //Role table의opertion no정보
                    opertion .no-- ;
}
}
else
{.....
class classname//inheritance 외 일반적인 모드
{.....
}
```

(리스트 8) C++ 코드생성 알고리즘

또한 이름 검색에 의해 Iterator 패턴이 검색되며, 검색된 패턴은 코드생성 언어에서 사용자가 원하는 C++ 언어를 선택하여 코드생성을 하면, Iterator 패턴의 C++ 코드가 생성되는 결과는 (그림 9)와 같다.

이것은 템플릿에 의한 Iterator 패턴 내의 Concrete-Iterator 클래스의 해더파일에 대한 생성 결과 화면 뷰이다. C++ 코드생성 알고리즘은 (리스트 8)과 같다. (그림 10)은 실행결과로 검색된 디자인 패턴에 대한 클래스 객체로 Iterator 패턴에 대한 UML 구현화면이다.

5. 적용 사례 및 비교분석

본 연구에서 제안한 모델을 중심으로 기존의 생성기인 F77/J++ 생성기와 적용 사례를 비교 분석하여 제시한다. 기존 생성기의 자동차 관리는 포트란 언어로 구성돼 있다. 노후코드에 사용된 코드의 구성은 107 소스 화일인데 158루틴을 포함하고 있다. 총 라인 수는 22,450 라인으로 구성되어 있다. 따라서 본 논문에서는 많은 코드를 표시할 수 없어서 일 부분만을 선택하여 적용 사례를 들었으며 많은 부분을 생략할 수밖에 없다. 객체지향코드 생성기는 효율적인 비교분석을 위해 기존 생성기에서 적용시킨 방법과 같은 방법으로 성능 평가를 실시한다. 기존에 사용한 방법을 패턴화하여 AMOSS(AutoMobile Customer Service Shop)를 JiKU/XML 객체 관리 저장소와 객체지향코드 생성기에 활용한다. 이 생성기는 오라클 데이터베이스와 XML과의 결합으로 구현한다. 또한 SQL Server는 FOR XML 구문으로 XML 데이터로 결과 조회가 가능하며 OPEN XML 구문을 이용하여 XML 데이터의 갱신도 가능하게 구현할 수 있다. 각 루틴의 체크 방법은 구현한 과정을 단계별로 시간을 체크했으며 실험 결과와 분석자료를 기존 생성기와 같은 방법으로 실시했다. <표 2>는 기존 생성기에서 하드웨어와 컴파일러, 시스템을 비교한 것으로 Legacy는 기존 생성기(F77/J++생성기)를 의미하며 New는 본 논문에서 제시한 시스템을 설명한 것이다.

〈표 2〉 하드웨어, 컴파일러, 시스템 비교

Hardware	Compiler	System
586DX PC	Salford FORTRAN v2.74	Legacy
586DX PC	Java 1.2,	Legacy
586DX PC	C++	New
Sun classic	f77 v3.0, -O3-cg89	Legacy
Sun classic	J++ 1.2,	Legacy
Sun classic	C++	New

이것은 본 논문에서 제시한 객체지향코드 생성기와 F77/J++생성기와 비교함으로서 PC(586DX)와 Sun 시스템을 각각 실험 대상으로 실시했다. 또한 실험의 공정한 성능 평가를 위해 기존 생성기와 같은 방법으로 실행을 실시했다. 또한 <표 3>은 586DX PC에서 일정한 루틴을 각 시스템에 적용하여 100번 반복 소요되는 시간을 나타낸 것이다.

〈표 3〉 586DX PC에서 100번 반복 소요시간

System	Compiler	Time
586DX PC	Salford FORTRAN v2.74	12m07s
586DX PC	Java 1.2,	20m16s
586DX PC	C++	25m46s

<표 4>는 Sun 시스템에서 100번 반복하는데 소요되는 시간을 나타낸 것이다.

〈표 4〉 Sun 시스템에서 100번 반복 소요시간

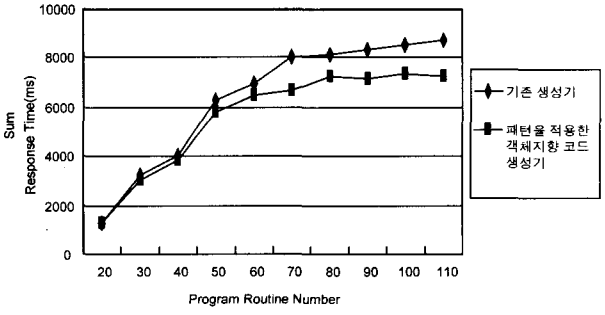
System	Compiler	Time
Sun classic	f77 v3.0, -O3-cg89,	5m10s
Sun classic	Java 1.2,	7m17s
Sun classic	C++	7m51s

새로운 시스템과 기존 생성기는 네 개의 소스와 13 해더 파일과 395루틴을 포함하고 있는 클래스 함수를 변형해서 적용했으며 소스 파일 안에는 11,250 라인과 1400 해더 파일로 로 구성되어 있다.

<표 5>는 기존 시스템과 패턴 디자인을 이용한 객체지향코드 생성기의 응답시간을 비교 분석한 것을 나타내고 있다. 이것은 568DX 시스템과 Sun시스템의 응답시간을 전부 합친 값을 그래프로 나타낸 것이다. 일정한 루틴을 각 시스템에 적용하여 20부터 10씩 증가하여 110까지를 100번 반복하였을 때 소요되는 시간을 나타낸 것이다. 이것을 그래프로 나타낸 것이 (그림 11)이다.

〈표 5〉 기존생성기와 객체지향코드생성기 응답시간비교

루틴 번호	기존 생성기 응답시간(ms)	객체지향코드 생성기 응답시간(ms)
20	1102	1084
30	3012	2802
40	4021	3860
50	6123	5433
60	6831	6182
70	7643	6542
80	7765	6998
90	7910	6812
100	8121	7122
110	8430	7233



(그림 11) 시스템 비교분석(568DX와 Sun 시스템)

즉 노후(Legacy) 시스템에서 새로운(New) 시스템으로 변형하는 과정을 서로 비교하여 시뮬레이션 결과를 나타낸 것이다. 그러나 본 논문에서 구현한 생성기는 각 단계를 거쳐서 마지막 부분에 프로그래머가 수정하는 단계를 거치므로 완전 자동화 단계를 이루지 못하는 단점도 있지만 분석 결과 객체지향코드 생성기가 기존 생성기 보다 향상된 기능을 제공함을 알 수 있다.

따라서 본 논문에서 제안한 생성기는 웹 환경에 적용시킬 수 있게 변형시켰으며, 새로운 환경에 쉽게 적용할 수 있는 모델제시와 소프트웨어 개발 시간을 단축시켰다는 장점을 가지고 있다.

본 논문에서 제시한 객체지향코드생성기와 비교한 시스템으로는 'IBM의 DPL'[12]과 CASE 도구인 'OODesigner 시스템', 'NetFree'[11], Omnibuilder[2], 그리고 ModelMaker [6]과 비교하였다. <표 6>에서 제시한 각 기준을 살펴보면, 먼저 패턴 추가에 대한 기준에서는 IBM의 DPL과 NetFree 시스템에서는 제공하지 못하고 있다. IBM의 DPL 시스템에서는 추가가 불가능하며, 단지 Gamma 분류법에 의한 패턴에 대한 분류만이 가능하다. 그리고 NetFree 시스템은 코드 생성은 지원하지만, 추가는 지원하지 않으며 디자인패턴에 대한 코드생성과 실시간 미들웨어 시스템을 지원한다. 생성 코드 언어는 IBM의 DPL 시스템과 OODesigner이 C++코드로 변환이 가능하고, Code Farms의 PTL 시스템과 NetFree는 Java 코드로 각각 변환이 가능하다. 따라서 기존 시스템과 객체지향코드 생성기를 종합적으로 비교한 결과는 <표 6>과 같다. 그러므로 본 시스템에서 제안한 객체지향코드 코드생성기 시스템에서는 웹상에서 표준화된 언어인 XML 코드 변환이 제공되며 디자인패턴 정보를 XML 코드로 변환하여 분산환경상의 개발자들을 위해 C++언어로 생성이 가능하다.

〈표 6〉 기존 시스템과의 비교

항목 / 시스템	컴포넌트	분산환경	분류모델	패턴/컴포넌트 추가	생성코드	모델링 도구 지원
IBM DPL[12]	클래스	×	열거+패싯	○	C++	컴포넌트(클래스)
OODesigner[12]	클래스	×	열거	○	C++	컴포넌트(UML)
NetFree[11]	클래스	○	열거	×	J++	×
Omnibuilder[2]	패턴	×	객체	○	×	템플릿(UML)
ModelMaker[6]	패턴	×	Gamma	○	×	템플릿(UML)
본 논문의 제시 방법	패턴	○	Gamma+패싯	○	C++	패턴(UML)

6. 결 론

본 논문에서 제안한 생성기는 열 단계를 거치면서 객체지향 코드가 생성된다. 이것은 UML로 표현된 설계 정보와 패턴을 이용하여 XML 코드로 생성하기 위해 PIML 구문법에 맞게 객체지향 코드가 생성된다. 이 모델은 디자인 패턴 재사용을 위한 JiKU/XML 객체지향코드 생성기를 설계하고 간단한 패턴 구조로 예로 들어 구현한다. 그리고 성능 평가를 위하여 기존 생성기와 비교 분석하는데 AMOSS로 적용사례를 제시하여 기존 생성기에서 적용한 같은 방법으로 적용한다.

기존 시스템과 디자인 패턴을 이용한 객체지향코드 생성기와 비교 분석하였다. 그래서 568DX 시스템과 Sun시스템의 응답시간을 체크하여 성능평가를 하였다. 그 결과 객체지향코드 생성기가 기존 생성기 보다 향상된 기능을 제공함을 알 수 있다.

따라서 객체지향코드 생성기가 기존 생성기 보다 다음과 같은 잇점이 있다. 첫째는 단일 시스템을 기반이 아닌 분산 환경 상에서의 개발자 및 유지 보수자들의 패턴 정보를 공유할 수 있다. 둘째는 기존 시스템과 달리 XML과 PIML 구문법을 사용했기 때문에 메타 모델함으로써 UML로 도식화가 가능하다. 셋째는 기존의 단점을 보완하기 위해서 XML기반으로 한 통합 관리 시스템을 제안하여 웹 환경에 쉽게 적응할 수 있다. 그러나 완전 자동화를 이루지 못함을 단점이라고 지적할 수 있다. 최적화 단계에서 프로그래머가 전반적인 수정을 다시 검증을 해야 한다.

향후 연구과제로는 다중언어 생성기를 구현하는 일이고, 객체 지향 프로세스에서 발생하는 다양한 산출물을 데이터베이스화하여 완전 자동화를 이루는 것이다.

참 고 문 헌

[1] Alin Deutsch, Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, David Maier, Dan Suciu : "Querying XML Data", IEEE Data Engineering Bulletin, Vol.22, No3, pp.10-18, 1999.

[2] http://www.omnibuilder.com/.

[3] E.Gamma, R. Helm, R. Johnson and J.Vlissides, "Design Patterns : Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995.

[4] F. J Budinsky, M. A. Finnie, JM. Vissides, P. S. Yu, "Automated code generation from design patterns," Object technology, IBM Systems, Vol.35, No.2, Journal, 1996.

[5] Grady Booch, Ivar Jacobson, and James Rumbaugh, Unified Modeling Language, Rational Software Corporation, January, 2001, Version 2.1.

[6] http://www.modelmaker.demon.nl/.

[7] Jon Meyer & Troy Downing "JAVA Virtual Machine," O'REILLY, 1997.

[8] J. Robie et al., "XML Qerry Language(XQL)," http://www.w3.org/TandS/QL/QL98/pp/xql.html, 1998.

[9] M. Ohtsuki, N. Yoshida, "A Source Code Generation Support System Using Design Pattern Documents Based on SGML," Proc. of the APSEC'98, 1998.

[10] Rational Soft, Corp, http://www.rational.co.kr/Product/Rose/.

[11] Solution Star, NetFree, http://www.mahanet.co.kr

[12] 김진향, 송영재, "디자인 패턴 재사용을 위한 다중언어 코드 생성기 설계에 관한 연구", 한국정보처리학회 학술발표대회 논문지, 2001.

[13] 선수균, 송영재, "통합 객체 관리 모델을위한 F77/J++ 생성기에 관한 연구", 정보처리논문지, 제7권 제10호, pp.3064-3074, Oct., 2000.

선 수 균

e-mail : sksun@tongwon.ac.kr

1988년 경희대학교 전자계산공학과(공학사)

1994년 경희대학교 전자계산학과(공학석사)

2002년 경희대학교 전자계산공학과 (공학박사)

1988년~1996년 경희대학교 전자계산소 프로그래머 근무

1996년~1997년 세경대학 전자계산과 교수

1997년~현재 동원대학 e-business과 교수

관심분야 : 소프트웨어 공학, 전자상거래, e-비즈니스, S/W 재사용