# Sopclos : 객체지향 데이타베이스 관리 시스템을 위한 CLOS 인터페이스

# (Sopclos : CLOS interface for object-oriented database management system)

**성명** :

| | | | |
|---|---|---|---|
| 서울대학교 자연과학대학 | 전산과학전공 | 정 태선(Tae-Sun Chung) | 학생회원 |
| 서울대학교 자연과학대학 | 전산과학전공 | 조 은선(Eun-Sun Cho) | 학생회원 |
| 서울대학교 공과대학 | 컴퓨터공학과 | 김 형주(Hyoung-Joo Kim) | 종신회원 |

**연구 세부분야** : 프로그래밍 언어

**키워드** : 객체 지속성(object persistency), CLOS, 메타객체 프로토콜(metaobject protocol),개방형 구현(open implementation), 객체지향 프로그래밍 언어(object-oriented programming language)

**주소** : 서울 특별시 관악구 신림동 산 56-1
서울대학교 컴퓨터공학과 객체지향 연구실
정 태선 (우: 151-742)

(Tae-Sun Chung OOPSLA Lab.
Computer Engineering Seoul National University
Shilim-dong 56-1 Kwanak-ku Seoul 151-742)

**전화** : 880-1830

**FAX** : 882-0269

**e-mail** : tschung@papaya.snu.ac.kr

# "Sopclos" : 객체지향 데이타베이스 관리 시스템을 위한 CLOS 인터페이스

# ("Sopclos" : CLOS interface for object-oriented database management system)


**요약문**

객체지향 데이타베이스 관리 시스템은 프로그래밍 언어 상의 객체와 데이타베이스 객체간의 타입 불일치(impedance mismatch)가 관계형 데이타베이스 관리 시스템에 비하여 심하지 않기 때문에 데이타베이스 응용 프로그램을 작성하기가 용이하다고 알려져 있다. 그러나 객체지향 데이타베이스 관리 시스템에서도 기본적으로 프로그래밍 언어에 데이타베이스 기능을 추가하려면 언어의 의미와 구문을 확장시켜야 하기 때문에 여러 가지 문제들 즉, 직교적 지속성(orthogonal persistency), 투명한 지속성(transparent persistency), 이식성(portability) 등의 문제가 생긴다. Sopclos는 이러한 문제를 해결하는, 객체지향 데이타베이스 관리 시스템을 위한 객체지향형 Lisp(CLOS) 인터페이스이다. 본 논문에서는 CLOS의 메타객체 프로토콜을 이용하여 사용자에게는 거의 같은 구문을 제공하면서도 CLOS 객체가 객체지향 데이타베이스 시스템에 저장되고 관리될 수 있도록 한 Sopclos의 설계와 구현을 제안한다.

**Abstract**

It is well known that database application programming is easier in OODBMSs(Object Oriented Database Management Systems) than RDBMSs (Relational Database Management Systems) because the impedance mismatches between objects in disks and objects in programming languages are not more serious in OODBMSs. But OODBMSs also have problems in adding database functionalities to programming languages as expansion of the programming language semantics is required. For this, orthogonal persistency, transparent persistency, and portability are the issues to be addressed. Sopclos which is a CLOS interface to an OODBMS solves these problems. In this paper we suggest the design and implementation of Sopclos providing object persistency on OODBMS through *metaobject protocol* of CLOS without drastic modification of CLOS syntax.


## 1 서론

객체지향 데이타베이스 시스템이 관계형 데이타베이스 시스템에 비하여 우수한 점의 하나는 프로그래밍 언어와 좀더 완전하게 결합한다는 점이다[4]. 관계형 데이타베이스에서는 사용자 레벨에서 응용 프로그램을 작성할 수 있도록 내포형 (embedded) SQL/C나 내포형 SQL/PASCAL 등을 제공하지만 데이타베이스와 프로그래밍 언어 사이에 모델간의 불일치(impedance mismatch)가 크기 때문에[3] 사용

자가 투명하게 프로그래밍 언어를 보면서 데이타베이스 응용 프로그램을 짜기가 어렵다. 반면에 객체지향 데이타베이스 시스템에서는 객체지향 프로그래밍 언어와 같은 객체지향 모델을 사용하기 때문에, 프로그래밍 언어와 데이타베이스가 보다 밀접하게 연관되어 있다.

먼저 프로그래밍 언어가 데이타베이스 기능을 지원하기 위해서는 크게 두 가지 기능을 프로그래밍 언어에 첨가시켜야 한다. 첫째, 데이타에 지속성(persistency)을 주는 기능이다. 일반적 프로그래밍 언어의 모든 변수와 그 값은 프로그램의 수행이 끝나면 없어지지만, 데이타베이스 응용 프로그램에서는 프로그램이 끝난 후에도 이와 같은 데이타가 데이타베이스에 남아야 하기 때문이다. 둘째, 데이타베이스는 여러 사용자가 데이타를 공유할 수 있기 때문에 데이타의 일관성(consistency)을 유지하는 일이 필요하다. 이를 위해서 병행 제어, 고장 회복 등의 기능을 프로그래밍 언어가 포함해야 한다.

이와 같은 데이타베이스 기능을 프로그래밍 언어가 지원하려고 할 때 다음과 같은 점들을 고려해야 한다.

- 프로그래밍 언어가 위와 같은 데이타베이스 기능을 제공하더라도 사용자에게 투명하게 보여져야 한다. 즉 데이타베이스 기능을 제공하려고 하면 언어의 의미나 구문이 확장되어야 하는데, 이때 이 기존 언어와의 차이가 작아야 사용자가 거부감 없이 프로그래밍을 할 수 있다.
- 프로그래밍 언어의 어떤 객체도 그 타입에 상관없이 독립적으로 지속될 수 있는지에 대한 타입 직교적 지속성(type orthogonal persistency)[3, 20]의 문제이다. 타입 직교적 지속성을 가지지 못하는 시스템에서는 사용자가 지속적, 비지속적 데이타를 둘 다 지원하기 위해 같은 타입을 두 번 정의해야 한다. 또, 지속성을 지니지 않는 타입의 객체를 지속시키려고 할 때 시뮬레이션을 해 주어야 하기 때문에 성능 상의 문제가 생긴다.

이제까지 등장한 상업용 객체지향 데이타베이스의 데이타베이스 언어들은 주로 C++를 기반으로 하고 있다.[2, 11, 18, 19, 20]. 그러나, C++는 특성상, 프로그래밍 언어의 확장성보다는 실행 시간 효율성이 강조되어 설계된 언어이기 때문에 위와 같은 사항들을 잘 고려하기에는 한계가 있다[12, 16].

반면에 Common Lisp[9, 10, 17]에 객체지향 모델을 포함한 언어인 CLOS는 C++와는 달리 그 언어 구현 자체가 객체지향적으로 디자인되어 있고 언어 자체를 확장할 수 있는 표준 인터페이스를 제공하는 개방형 구현[7]으로 디자인된 언어이다. 따라서 CLOS는 이식성을 가지면서 프로그래밍 언어를 확장하기에 매우 적합한 언어이다. CLOS의 이러한 인터페이스는 '메타객체 프로토콜(metaobject protocol)'[6, 16]의 형태로 제공된다. 이러한 메타객체 프로토콜을 이용하여 지속성을 준 예로는 PCLOS(Persistent Common Lisp Object System)[13, 14, 15]와 Metastore[7]가 있다. 이러한 시스템은 메타객체 프로토콜을 통하여 CLOS 구문의 많은 변화없이 투명한 지속성을 주었다고 할 수 있지만, 직교적 지속성 면에서는 슬랏[1] 레벨에서는 직교적 지속성을 제공한다고 할 수 있지만 객체 레벨의 직교적 지속성을 제공하

[1] CLOS에서는 다른 프로그래밍 언어의 속성(attribute)에 해당하는 용어로 슬랏(slot)이라는 말을 쓴다.

지는 못하고 있다. 또한 객체 지향 데이타베이스와의 연결 측면에서 모델상 완벽한 결합을 하지 못하고 있다. 이 논문에서는 이러한 단점을 극복하는, CLOS의 메타객체 프로토콜을 이용하여 CLOS를 확장하여 지속성을 부여한, 데이타베이스 언어인 Sopclos를 제안한다.

논문의 구성은 다음과 같다. 2절에서는 CLOS에 대하여 사용자 레벨과 메타레벨로 나누어서 설명하고, 3절에서는 Sopclos에 관해 프로그래밍 언어 확장과 데이타베이스 연결 측면에서 분석한 후, 4절에서 기존의 관련 연구들과 Sopclos를 비교하여 살펴보며, 5절에서 결론을 맺는다.

# 2 CLOS 개요

CLOS는 높은 생산성, 빠른 프로토타이핑, 높은 이식성을 가지기 때문에 복잡하고, 다양하고, 계속 변화하는, 현대의 데이타베이스 응용 분야에 적합한 프로그래밍 언어라고 할 수 있다. 특히 전문가 시스템, 기계 학습, 자연어 처리, 음성 인식, 화상 인식 등의 인공지능 분야에서 가장 많이 쓰이는 Lisp에 객체지향 개념이 추가된 언어라는 장점이 있다. 또한 CLOS는 프로그래밍 언어 내부를 어느 정도 보여주는 개방형 구현(open implementation)으로 디자인된 언어이기 때문에 언어의 확장이 용이하다. 본 절에서는 CLOS에 대하여 일반 사용자가 보는 프로그래밍 언어와, 시스템 내부를 들여다 보는 메타 프로그래머 입장에서의 프로그래밍 언어로 구분하여 살펴본다.

## 2.1 사용자 레벨의 CLOS

CLOS의 구문은 Lisp과 마찬가지로 함수 수행으로 나타내어진다. 예를 들면 가로(width)와 세로(height)의 길이를 가지는 사각형(rectangle)을 클래스로 정의하면 다음과 같다[6].

```
(defclass rectangle ()
    ((height :initform 0.0 :initarg :height)
    (width :initform 0.0 :initarg :width)))
```

여기서 가로와 세로의 길이는 앞서 언급했듯이 다른 프로그래밍 언어에서 속성(attribute)에 해당되는 것으로 CLOS에서는 슬랏(slot)으로 불린다. 클래스의 각 슬랏에 대한 정의는 여러 인자로 이루어진다. 먼저 `:initform`은 객체 생성시 그 슬랏의 초기 값을 0.0으로 결정할 것을 나타낸다. `:initarg`는 객체 생성시 사용자가 해당 슬랏의 초기 값을 줄 때 쓰이는 키워드를 정의하기 위한 것이다.

CLOS는 다른 Smalltalk[5] 기반 객체지향 언어들과는 달리 제너릭 함수(generic function) 중심의 다중 메소드 방식이다. 즉 Smalltalk에서는 객체에 메시지를 보내는 방식으로 모든 시스템이 설명되고 프로그램이 작성되지만, CLOS에서는 대신 추상적인 '제너릭 함수'가 있어서, 구체적인 객체 인자들을

가지고 이를 호출하는 방식을 취한다. 클래스 정의와 제너릭 함수가 있고, 제너릭 함수의 실제 구현인 각 제너릭 함수마다 특성화(specialized)된 클래스들을 형식 인자로 하는 여러 구체적인 '메소드'들을 정의하는데 실행 시에 해당 인자 객체들의 클래스와 형식 인자가 가장 맞는 특성화된 메소드가 호출됨으로써 프로그램이 동작한다. 그러므로 CLOS 프로그래밍은 기본적으로 클래스를 정의하고 해당 클래스에 바인딩되는 메소드를 일반 함수와 별도로 정의함으로써 이루어지게 된다.

다음은 어떤 객체에도 적용 가능한 제너릭 함수 paint에 대한 정의이다. 그리고 위에 정의한 사각형 클래스에 대하여 특성화(specialized)된 메소드 paint를 정의하는 방법은 다음과 같다.

```
(defgeneric paint (x))
(defmethod paint ((x rectangle))
    (verticle-stroke (slot-value x 'height)
                     (slot-value x 'width)))
```

즉 제너릭 함수 paint에 대하여 그 인자로 사각형 타입의 객체가 주어지면 사각형 클래스에 특성화된 메소드가 호출되게 된다. 위 예에서는 주어진 사각형 타입의 객체에 대하여 가로의 길이와 세로의 길이를 추출하여 세로 막대기를 그리는 함수를 호출한다. 위 예에서는 사각형 클래스에 특성화된 메소드만을 정의하였지만 같은 paint 제너릭 함수에 대하여 다른 클래스를 정의하고 해당 클래스에 특성화된 메소드를 정의할 수도 있다. 이와 같은 클래스 정의와 메소드가 있을 때 사각형 클래스의 객체를 생성하여 메소드를 호출하는 방식은 다음과 같다.

```
>(setq rectangle1 (make-instance 'rectangle :height 10 :width 10))
#<RECTANGLE 31716770>
>(paint rectagle1)
```

먼저 객체를 생성하는 `make-instance` 메소드를 호출하여 생성된 객체를 rectangle1에 치환한 후 paint 제너릭 함수를 호출하였다. 이때 paint 제너릭 함수는 rectangle1의 클래스를 찾아서 클래스 사각형에 특성화된 메소드를 호출하게 된다.

Snyder[21]의 분류 방법에 따르면 C++는 계승(inheritance)방식이 가상 계승(virtual inheritance)을 쓰는 경우 그래프 중심(graph oriented) 방식이거나 일반 계승인 경우 트리 중심(tree oriented) 방식인데 반하여 CLOS는 선형적(linear) 방식을 쓴다. 즉 각 클래스마다 상위클래스에 대한 클래스 우선 순위 리스트(class precedence list)를 유지하여 다중 계승에서 이름 충돌(name conflict)이 생기는 경우 이 클래스 우선 순위 리스트의 순서에 따라 계승받을 속성을 결정함으로써 이 문제를 해결한다.


그림 1: 검은 상자 추상화

CLOS는 C++와는 달리 실행 시간에 클래스 재정의가 허용되는 동적 객체지향 언어이다. 이는 실행 시간에 객체의 구조와 동작을 점진적으로 바꿀 수 있음을 뜻한다. 클래스의 이러한 동적인 성질은 클래스 자체를 클래스의 클래스인 '메타클래스'의 한 객체로 봄으로써 가능하다. 그 밖에도 CLOS는 클래스뿐 아니라 슬랏, 메소드 등도 모두 객체로 보고 이를 위한 메타클래스들을 지원하고 있다. 이러한 객체들을 '메타객체'라고 하는데 다음절에서 이에 대해 자세히 설명한다.

## 2.2 메타레벨의 CLOS : 메타객체 프로토콜(metaobject protocol)

프로그래밍 언어를 제공하는 방법에 있어서 종래의 방법은 그림 1과 같이 입력과 그 출력만을 중요시하고 내부 구현은 사용자에게 숨기는 검은 상자 추상화(black box abstraction)[7, 8]형태가 대부분이다. 이러한 구현 방법은 프로그래밍 언어 자체가 망가질 염려가 없으므로 안전하다고 할 수 있지만, 프로그래밍 언어가 제공하는 기능이 고정적이므로, 언어에 지속성을 주는 것과 같이 언어 자체의 의미(semantic)를 바꾸는 경우에는 이식성을 가지면서 언어를 확장시키기에 어려움이 따른다.

반면에 프로그래밍 언어 구현 방법에 있어서 구현 내부를 사용자에게 어느 정도 보여주는 개방형 구현(open implementation)[7]의 방법은 언어의 의미 자체를 변경시키거나 확장시키는 것을 처음부터 지원한다고 볼 수 있다. 개방형 구현에서 프로그래밍 언어 구현자는 사용자에게 가릴 부분(closed part)과 보여줄 부분(open part)을 결정하게 되는데, 보여줄 부분으로는 내부는 가리고 기능만을 제공하는 일반 인터페이스와, 언어 구현의 내부를 보여주는 메타인터페이스(meta-interface)의 두 가지를 두게 된다. 사용자는 이 메타인터페이스를 통하여 객체지향 패러다임의 주요 개념인 일반화(generalization), 특성화(specialization), 계승(inheritance), 캡슐화(encapsulation)[21] 등을 통하여 프로그래밍 언어의 의미를 변경할 수 있게 된다.

앞서 언급 한대로 CLOS는 개방형 구현으로 설계된 언어이며, 메타인터페이스로서 '메타객체 프로토콜(metaobject protocol)'이란 것을 제공한다. CLOS의 메타객체 프로토콜을 간단하게 정리하면 다음과 같다.

- 메타객체 클래스(metaobject class) : CLOS는 크게 클래스(class), 슬랏(slot), 제너릭 함수(generic function), 메소드(method), 메소드 조합(method combination)의 다섯 가지의 기초 요소로 이루어진다. 이들 각각에 해당하는 기본적인 메타클래스가 존재하는데 이것을 '메타객체 클래스(metaobject

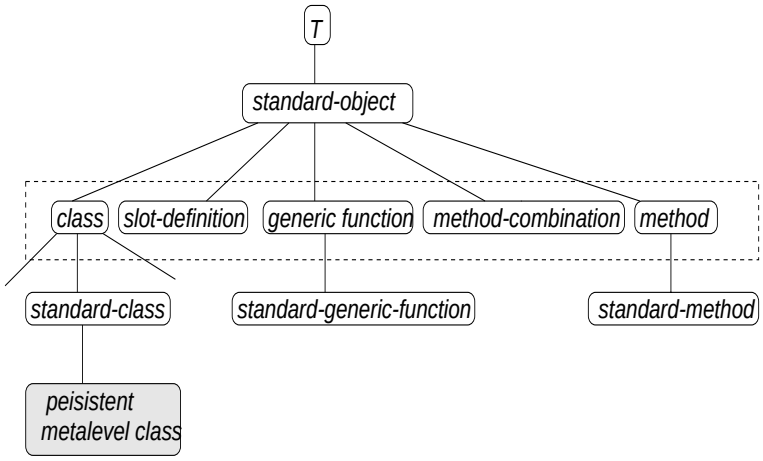


그림 2: 메타클래스 계층 구조

class)'라고 한다.

- 메타객체(metaobject) : 메타객체 클래스의 인스턴스를 말한다. 사용자 레벨의 클래스, 슬랏, 제너릭 함수, 메소드, 메소드 조합들이 이것에 해당한다. 즉, CLOS를 이루는 이러한 기초 요소들 자체를 각각 하나의 객체로 간주한다는 뜻이다.
- 메타객체 프로토콜(metaobject protocol) : CLOS의 메타인터페이스로서 CLOS 언어를 구성하는 메타객체 클래스의 계층 구조를 보여주고, 메타객체가 생성될 때 메시지와 인자의 전달을 구체적으로 보여주는 프로토콜을 말한다.

그림 2는 메타객체 프로토콜에서 제공하는 CLOS 내부의 시스템 클래스간의 계층 구조를 단순화하여 나타낸 그림이다. 객체지향 패러다임에서 클래스는 그 인스턴스들의 구조와 행위를 결정하므로, 이러한 CLOS 내부의 클래스 계층 구조는 언어 자체의 구조와 행위를 기술하는 중요한 의미를 갖는다. 그림 2에서 보면 CLOS의 최상위 클래스에는 `T`와 `standard-object`가 있고 그 하위 클래스로 CLOS를 구성하는 기본 요소인 클래스, 슬랏, 제너릭 함수, 메소드, 메소드 조합이 있다.

예를들어, 다음과 같이 사용자가 'Person'이라는 일반 클래스를 정의하였다고 하자.

```
>(defclass Person ()
    ((name :initarg :name)
    (age :initarg :age)
    (sex :initarg :sex)))
#<Standard-Class PERSON 32677730>
>
```

여기서 리턴 값에 주의하면 'Person'이라는 클래스는 자신의 클래스로 `standard-class`를 가지는 하

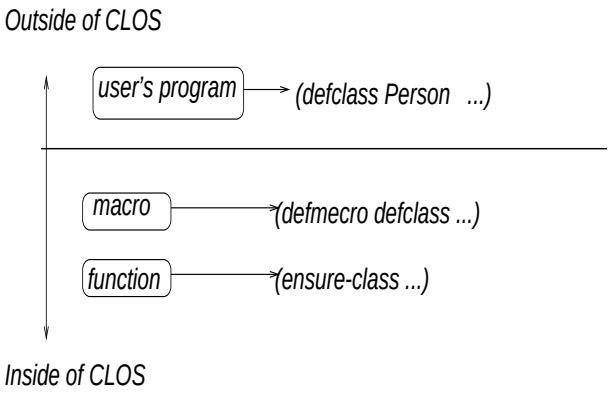


그림 3: defclass의 동작

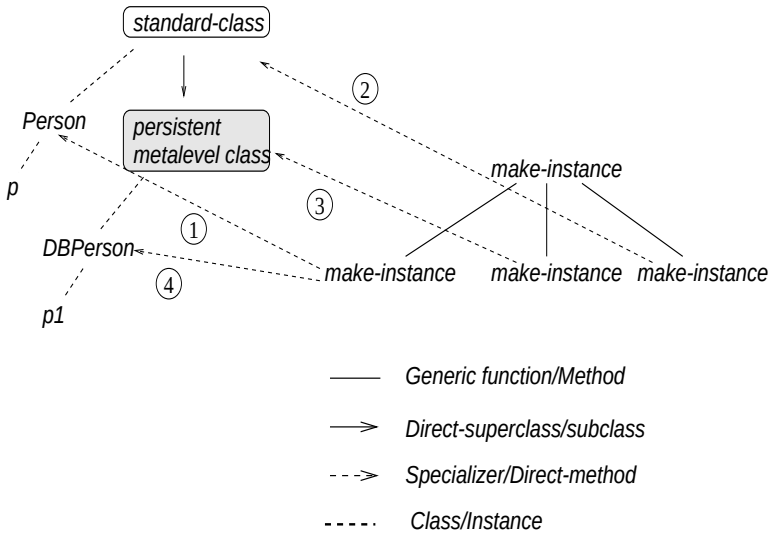


그림 4: 지속 객체 생성 프로토콜

나의 객체임을 알 수 있다. 메타인터페이스를 통하여 `defclass`의 동작을 좀더 자세히 살펴보면 그림 3과 같다. 즉, 사용자가 `defclass` 매크로를 이용하여 'Person' 클래스를 생성하려고 하면 CLOS 내부에서 매크로 확장을 거친후 '`ensure-class`'라는 함수를 호출하게 된다. 이 '`ensure-class`'는 `standard-class`에 특성화된 `make-instance`를 호출함으로써(그림 4의 2) 'Person'이라는 클래스 메타객체를 생성하여 이것을 리턴하게 된다.

다음으로 사용자가 일반 클래스 'Person'의 인스턴스인 'p' 객체를 생성한다고 하자.

```
(setq p (make-instance 'Person :name '홍길동 :age 20 :sex 'm))
```

즉, 사용자는 클래스 이름과, 여러 슬랏(이름, 나이, 성별)의 초기값을 인자로 하여 `make-instance` 메소드를 호출함으로써 객체를 생성하게 되는 것이다. 이때 메타인터페이스를 통하여 이 객체를 생성하는 과정을 좀더 자세히 살펴보면 다음과 같다.

**Step1** 심벌(Person)에 특성화된 make-instance 메소드 호출(그림 4의 1)

1-1 이 심벌(Person)의 클래스를 찾음 : `standard-class`

**Step2** `standard-class`를 인자로 다시 `make-instance` 호출(그림 4의 2)

2-1 객체를 위한 메모리를 주기억 장치(main memory)내에 할당

2-2 객체를 주어진 인자에 따라 주기억 장치내에서 초기화

2-3 객체를 리턴

이와 같이 CLOS의 메타인터페이스인 메타객체 프로토콜은 CLOS 내부의 동작을 메타 프로그래머에게 보여주고, 언어의 확장을 용이하게 하는데 다음절에서 이러한 메타객체 프로토콜을 Sopclos에서 어떻게 이용하고 있는지 살펴본다.

# 3 Sopclos

Sopclos는 프로그래밍 언어 CLOS(Common Lisp Object System) 해석기 (interpreter)에 ODMG[18] 기반 객체지향 데이타베이스인 SOP(SNU Object Oriented Database Platform)를 접합시켜 프로그래밍 언어에 지속성을 부여하고 데이타베이스 기능을 제공하는 새로운 해석기이다. 이 절에서는 Sopclos를 프로그래밍 언어의 확장과 데이타베이스와의 연결 측면에서 분석한다.

## 3.1 CLOS의 DBPL로의 확장

CLOS에 지속성을 주기 위하여 프로그래밍 언어 관점에서 필요한 일은 다음과 같이 크게 세 가지로 나눌 수 있다.

- 지속 클래스를 위한 메타 레벨 클래스 정의 : 기본적으로 객체가 지속성을 가질 수 있음을 결정하는 일은 그 객체가 속한 클래스로부터 결정된다. 이때 일반 클래스가 가지는 메타클래스인 `standard-class`와는 다른 클래스를 메타클래스로 가지도록 해야 한다.
- 지속 객체의 생성 : 지속성을 가지는 클래스의 객체는 데이타베이스에 생성하여야 한다. 이때 지속성을 위한 데이타 구조가 각 객체마다 내부적으로 생성되어야 한다.
- 지속 객체에의 접근 : 지속 객체에 대한 읽기나 쓰기를 할 때 데이타베이스에서 읽거나 써야 한다. 그러므로 지속 객체의 경우 CLOS 내부의 동작을 탐지하여 적당한 시점에서 데이타베이스와 연결하여야 한다.

이 밖에도 Sopclos는 다중 사용자간의 데이타의 일관성(consistency)을 위하여 트랜잭션 기능을 지원하는데 본 절에서는 CLOS가 제공하는 메타인터페이스인 메타객체 프로토콜을 통하여 언어의 의미(semantic)를 확장하는 과정을 살펴본다.

### 3.1.1 메타 레벨 클래스 정의와 지속 객체의 생성

Sopclos에서는 그림 4와 같이 `persistent-metalevel-class`를 `standard-class`의 하위 클래스로 정의한후 이 클래스에 특성화하여 `make-instance`를 새로이 정의함으로써(그림 4의 3) 객체가 지속성을 가지도록 하였다. 먼저 지속 클래스 'DBPerson'를 정의하는 구문은 다음과 같다.

```
>(defclass DBPerson ()
    ((name :transient nil :initarg :name)
    (age :transient nil :initarg :age)
    (sex :transient nil :initarg :sex))
    (:metaclass persistent-metalevel-class) )
#<Persistent-Metalevel-Class PERSON2 31145500>
```

앞서 정의한 일반 클래스 'Person'의 클래스 정의와 비교하면 두가지 점에서 차이가 남을 알 수 있다. 그 하나는 '`:transient`'라는 슬랏에 대한 옵션이 추가되었다는 점과 '`:metaclass`'옵션에 `persistent-metalevel-class`를 명시했다는 점이다[2]. '`:transient`' 옵션은 슬랏 레벨의 지속성을 제공하기 위하여 추가한 옵션인데 자세한 내용은 다음절에서 다룬다. 다음으로 'DBPerson' 타입의 지속성 객체 'p2'를 생성하는 구문은 다음과 같다.

```
>(setq p2 (make-instance 'DBPerson :is-persistent T
          :name '홍길동 :age 20 :sex 'm))
```

이 객체 생성 구문은 일반 CLOS 구문에 비하여 '`:is-persistemt`'라는 구문이 추가되었는데, 객체의 클래스가 그 메타클래스로 `persistent-metalevel-class`를 가지더라도 객체가 지속성을 가지지 않을 수 있도록 한다[3]. 즉 '`:is-persistent`' 옵션을 nil로 셋팅하면 일반 객체와 같이 주기억 장치내에 객체를 생성한다. 객체 생성과정의 프로토콜을 자세히 살펴보면 다음과 같다.

**Step1** 심벌(DBPerson)에 특성화된 make-instance 메소드 호출(그림 4의 4)

1-1 이 심벌(DBPerson)의 클래스를 찾음 : `persistent-metalevel-class`

**Step2** `persistent-metalevel-class`를 인자로 다시 `make-instance` 호출(그림 4의 3)

2-1 객체를 위한 메모리를 데이타베이스내에 할당

2-2 객체를 주어진 인자에 따라 데이타베이스내에서 초기화

2-3 객체를 리턴

따라서, Sopclos에서 객체의 생성과정의 프로토콜은 알고리즘 1과 같다.

[2] 명시를 안한 경우에는 자동으로 메타 클래스가 `standard-class`가 된다.

[3] 옵션을 명시하지 않는 경우는 메타클래스의 종류에 따라서 지속성이 결정된다.

**Algorithm 1** Sopclos에서의 객체 생성 프로토콜

생성 객체의 클래스의 클래스인 메타 클래스를 찾음;

**if** 메타 클래스가 `persistent-metalevel-class` **then**

객체를 위한 메모리를 데이타베이스내에 할당;

객체를 주어진 인자에 따라 데이타베이스내에서 초기화;

**else** {메타 클래스가 `standard-class`}

객체를 위한 메모리를 주기억 장치내에 할당;

객체를 주어진 인자에 따라 주기억 장치내에서 초기화;

**end if**

객체를 리턴;

### 3.1.2 지속 객체의 접근

객체의 접근은 크게 읽는 경우와 쓰는 경우의 두 가지로 나눌 수 있다. CLOS에서 객체에 접근하려면 (`slot-value`)와 (`setf slot-value`)를 쓴다. 앞서 생성한 일반 객체 'p'의 이름 항목을 추출하려고 하면 다음과 같이 하면 된다.

```
(setf n (slot-value p 'name))
```

일반 객체 'p'의 이름 슬랏을 추출하여 n이라는 변수에 치환하였다. 이때 `slot-value`함수는 CLOS 내부에서 얻고자 하는 슬랏값이 적당한지를 검사한후, 인자로 받은 객체의 클래스를 찾고('Person' 클래스 메타객체) `slot-value-using-class`라는 메소드를 호출하게 된다. CLOS의 메타인터페이스 레벨에서 `slot-value-using-class`는 다중 메소드(multi-method)로써 다음과 같은 세가지의 인자를 가진다.

1. `standard-class`에 특성화(specialized)된 클래스 메타객체
2. 사용자가 인자로 넘긴 인스턴스
3. `standard-effective-slot-definition`에 특성화된 슬랏 메타객체

`standard-effective-slot-definition`은 CLOS 내부에서 슬랏 관리를 위해 정의된 시스템 클래스로 그림 2의 `slot-definition`의 하위 클래스로 정의되어서 슬랏의 이름, 접근 함수, 타입등의 정보를 유지하고 있다. Sopclos에서는 메타 레벨에서 `standard-effective-slot-definition` 클래스의 하위 클래스로 `persistent-standard-effective-slot-definition`을 정의하여 사용자가 '`:transient`'옵션을 사용하여 슬랏 레벨의 지속성을 부여할 수 있도록 하고, 메타 프로그래머에게 슬랏의 지속성 여부를 검사할 수 있도록 하였다. 그러므로 지속 객체의 접근을 위한 `slot-value-using-class` 메소드는 다음의 인자를 가진다.

1. `persistent-metalevel-classs`에 특성화(specialized)된 클래스 메타객체
2. 사용자가 인자로 넘긴 인스턴스
3. `persistent-standard-effective-slot-definition`에 특성화된 슬랏 메타객체

앞서 정의한 지속 객체 'p2' 의 이름 항목을 추출하려고 한다면, 일반 CLOS 구문과 같이 다음과 같이하면 된다.

```
(setf n2 (slot-value p2 'name))
```

이 경우에는 'p2'의 클래스가 클래스 메타객체 'DBPerson'로서 `persistent-metalevel-classs`의 인스턴스이므로 새로이 정의된 `slot-value-using-class`가 호출되게 되어 데이타베이스를 접근하게 된다. 이때 Sopclos에서 새로이 정의된 `slot-value-using-class`의 동작을 살펴보면 알고리즘 2과 같다.

**Algorithm 2** slot-value-using-class의 동작

**if** 슬랏의 '`:transient`'옵션이 TRUE **then**

다음으로 특성화된 클래스(`standard-class`)에 특성화된 메소드 호출;

**else if** 객체의 '`:is-persistent`' 옵션이 FALSE **then**

다음으로 특성화된 클래스(`standard-class`)에 특성화된 메소드 호출;

**else**

데이타베이스 함수 호출;

**end if**

여기서 클래스를 지속클래스로 가지더라도 사용자가 명시한 슬랏 레벨이나 객체 레벨의 지속성 부여 여부에 따라 다음으로 특성화된 메소드를 호출함으로써[4] 타입에 대한 직교적 지속성을 가지도록 하였음을 알 수 있다. 객체 슬랏의 내용을 바꾸는 (`setf slot-value`) 함수도 마찬가지로 구현되었다.

### 3.1.3 트랜잭션 처리

Sopclos에서는 SOP에서 제공하는 트랜잭션 처리 함수를 이용하여 `with-transaction`이라는 매크로를 정의하여 트랜잭션을 처리한다. 그 구문은 다음과 같다.

```
>(with-transaction
        ...                    ; 지속성 객체의 접근 구문
)
```

이러한 트랜잭션의 범위는, 중첩되지 않는 한도 내에서 사용자에 의해 자유롭게 명시될 수 있다.

[4] CLOS 에서는 다음으로 특성화된 메소드를 호출하는 `call-next-method`라는 함수를 제공한다.

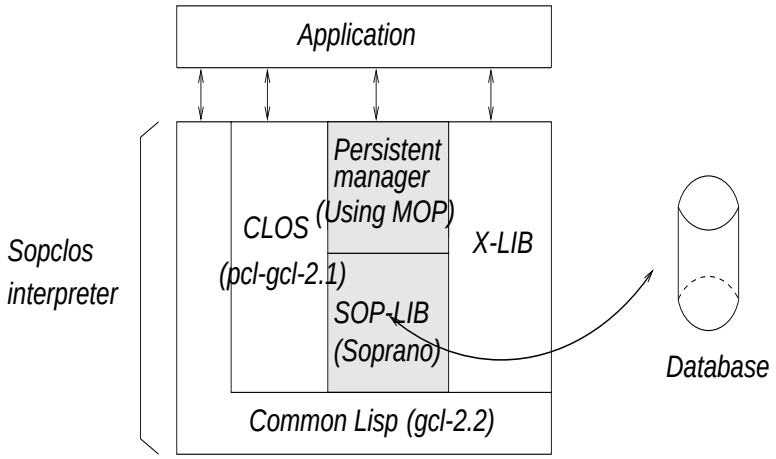


그림 5: Sopclos의 전체 구조

### 3.2 데이타베이스 시스템 연결

데이타베이스와의 연결을 살펴보기 위하여 Sopclos의 전체적인 구조를 먼저 살펴보면 그림 5와 같다. Sopclos 해석기의 기반이 되는 Common Lisp의 해석기는 gcl-2.2를 사용하였고, CLOS를 이루는 PCL(Portable Common Loops)은 pcl-gcl-2.1을 이용하였다. 또 Sopclos는 사용자가 X 윈도우 프로그래밍을 할 수 있도록 xgcl-2를 포함하였다. 따라서 응용 프로그래머는 사용자 인터페이스가 우수한 데이타베이스 응용 프로그램을 작성할 수 있다.

Sopclos의 구조상의 특징은 객체지향 데이타베이스의 라이브러리가 CLOS 해석기 속에 포함되어 있다는 점이다. 이때 Sopclos 해석기가 이용하게 되는 데이타베이스 함수들을 기능별로 정리하면 다음과 같다.

- 데이타베이스를 초기화하는 함수
- 트랜잭션에 관련된 함수
- 데이타베이스에 객체를 생성하는 함수
- 주어진 이름에 따라 데이타베이스의 객체를 읽는 함수
- 주어진 이름의 객체를 데이타베이스에 쓰는 함수
- 데이타베이스의 이용을 마치는 함수

다음에는 Sopclos의 기반 데이타베이스 관리 시스템인 SOP의 객체 관리자인 Soprano의 구조에 대해 간단히 알아보고, 데이타베이스 스키마를 등록하는 방법을 소개하고, C++ 기반의 데이타베이스를 Sopclos에서 어떻게 접근하는지에 대해 알아본다. 각 기능에 대한 Sopclos함수는 부록에 있다.

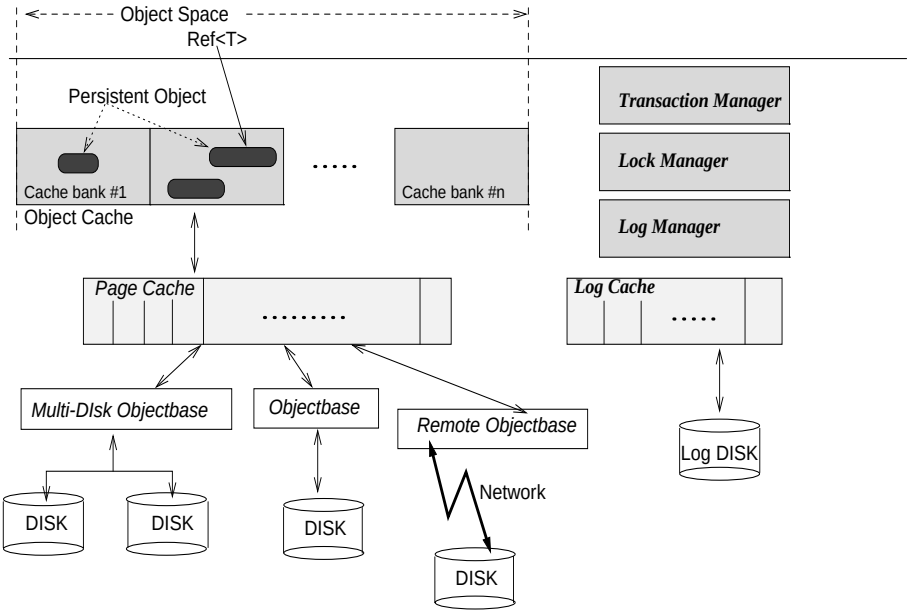


그림 6: SOP의 객체 관리자인 Soprano의 구조

### 3.2.1 객체 관리자의 구조

Sopclos가 주로 사용하는 SOP의 저장 시스템인 Soprano[1]의 구조는 그림 6과 같다. 그림에서와 같이 Soprano는 물리적 장치를 다루는 ObjectBase와 페이지 캐쉬, 객체 캐쉬가 있어서 객체를 캐싱하고 스위즐링(swizzling)[24]과 언스위즐링(unswizzling)을 하여 성능을 높인다. 그리고 병행 제어와 고장 회복을 위한 로크(lock) 관리자, 로그(log) 관리자, 트랜잭션 관리자 등 이 있다. SOP는 C++ 기반의 객체지향 데이타베이스 관리 시스템으로 지속 객체의 접근을 위하여 ODMG[18, 19]에서 제안한 영리한 포인터(smart pointer)인 Ref 핸들러를 쓰고 있다. Sopclos 내부에서도 궁극적으로는 이 Ref 핸들러를 통하여 지속 객체에 접근한다. Sopclos는 SOP가 제공하는 시스템 레벨 함수들을 직접 호출하여 데이타베이스 관리 시스템의 트랜잭션이나, 데이타 공유, 캐싱 등을 수행하게 된다.

### 3.2.2 데이타베이스 스키마 등록

데이타베이스 응용 프로그램을 작성하기 위해서는 먼저 데이타베이스에 스키마를 등록 시켜야 한다. SOP에서는 스키마 등록 도구인 Import 툴을 이용하여 스키마를 등록한다. 스키마 등록을 위해 사용되는 언어는 ODMG에서 제안한 스키마 정의 언어인 ODL(Object Definition Language)[18]이다. ODL은 객체지향 데이타베이스 관리 시스템의 스키마 이식성을 위한 언어로 특정 프로그래밍 언어에 의존하지 않는 독립적인 명세 언어(specification language)이다. 즉, 그림 7과 같이 ODL로 스키마를 정의하여 데이타베이스에 등록하면, ODL 처리기는 각 언어에 맞도록 클래스를 생성해 주게 된다. 현재는 SOP의 ODL 처리기에서 아직 CLOS 코드로의 변환이 지원되지 않으므로, 사용자가 직접 CLOS로 된

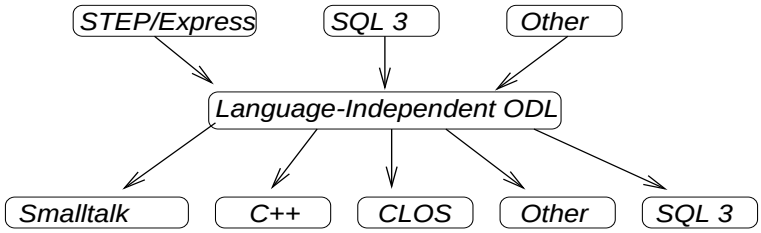


그림 7: ODL과 프로그래밍 언어

스키마 클래스를 정의하고 있다.

각 스키마 클래스에 해당하는 CLOS 클래스는 앞서 언급한 바와 같이 메타클래스를 `persistent-metalevel-class`로 하여 정의된다. 예를 들어 앞서 정의한 'DBPerson'클래스의 하위 클래스로 'Employee'라는 지속 클래스를 정의 한다면 다음과 같이 하면 된다.

```
(defclass  Employee (DBPerson)
    ((salary :transient nil :initarg :salary))
    (:metaclass persistent-metalevel-class) )
```

### 3.2.3 데이타베이스 접근

SOP는 대다수의 ODMG 기반의 시스템과 마찬가지로 C++를 기반으로 하고 있기 때문에 Lisp와 C++와의 교신을 필요로 한다. 앞서 밝혔듯이 Sopclos에서는 Lisp에서 직접 데이타베이스 함수를 호출하는 방식을 쓴다. 이 외에도 CLOS 상에서 C++ 기반의 데이타베이스 함수를 호출하는 방법으로, 간접적으로 RPC(Remote Procedure Call)나 소켓을 이용하는 방법[22] 등을 생각해 볼 수 있다. 그러나 이 경우에는 다음과 같은 단점이 있다.

- 성능 상의 문제이다. 전송되는 데이타가 네트워크를 통해야 하므로 직접 호출하는 방식에 비하여 성능 면에서 불리하다.
- 전송되는 데이타의 타입이 제한된다. 데이타가 네트워크를 타려면 데이타가 일정한 형식을 가져야 하므로 전송되는 데이타의 타입이 제한된다.

Sopclos가 사용한 gcl은 그 구현이 C를 포함하므로 C 언어와의 교신은 비교적 수월하다. Sopclos는 gcl이 제공하는 `defentry` 인터페이스를 이용하여 Lisp에서 C 함수를 부른다. `defentry`는 C 함수를 인자와 리턴 값을 명시하면 Lisp 함수처럼 쓸 수 있게 해주는 함수이다. 예를 들어 정수 타입의 두 인자를 받아서 합을 리턴하는 `add` 함수를 C로 정의하여 Lisp 해석기 상에서 부르려면 다음과 같이 하면 된다.

```
(clines "
```

```
int add(int, int);
int add(a , b)
int a;
int b;
{
    return a+b;
}
")
(defentry add (int int) (int "add"))
```

defentry는 이와 같이 Lisp와 C 언어간의 타입 차이를 내부적으로 해결하여 Lisp에서 C 함수를 부를 수 있도록 해준다. 타입이 문자열인 경우는 더 복잡해지는데 내부에서 object라는 C 언어의 유니언 타입을 정의하여 처리한다.

아직까지 Lisp에서 C++에 대한 인터페이스는 제공하지 않으므로 Sopclos는 extern을 사용하여 C++ 함수를 C로 재정의하는 방법으로 Lisp에서 C++ 함수를 부른다.

# 4 관련 연구와의 비교

## 4.1 C++ 바인딩과의 비교

C++에 지속성을 주는 방법은 크게 두 가지로 분류할 수 있다[23]. 하나는 클래스의 계승을 이용한 클래스 라이브러리 방식이고, 또 하나는 프로그래밍 언어에 새로운 구문을 추가하는 DML 바인딩이다. 앞서 언급 한대로 C++는 언어 디자인 단계에서 언어의 확장을 고려하고 디자인된 언어가 아니므로 어느 방식을 사용하던지 직교적 지속성이나 투명한 지속성의 문제가 생긴다.

먼저 클래스 라이브러리 바인딩 방법에서는 지속성을 위한 여러 데이타 구조를 가지는 루트 클래스를 정의하여, 지속성을 가지는 클래스는 이 클래스로부터 계승받도록 함으로써 클래스의 객체가 지속성을 가지도록 한다. 이 방법은 클래스의 타입에 따라 지속성이 결정되기 때문에 타입에 직교적인 지속성을 보장하기 힘들게 된다.

다음으로 새로운 키워드를 추가하여 타입에 직교적 지속성을 가지도록 하는 DML 바인딩에서는 전처리 과정을 거치거나(pre-processing) 확장된 C++ 컴파일러를 필요로 하므로 이식성이 떨어지고 사용자에게 투명한 지속성을 주지 못한다.

## 4.2 CLOS 바인딩과의 비교

### 4.2.1 PCLOS

PCLOS(Persistent Common Lisp Object System)[13, 14, 15]는 여러 종류의 데이타베이스와 여러 프로그래밍 언어를 독립적으로 지원하는 것을 디자인 목표로 하여 만들어진 시스템으로 프로그래밍 언어 단계와 가상 데이타베이스(virtual database)단계, 실제 데이타베이스 시스템의 세 단계로 이루어진 시스템이다.

Sopclos와 비교하여 PCLOS의 가장 큰 단점은 데이타베이스와의 결합도가 떨어지고, 객체지향 모델의 장점을 살리지 못한다는 점이다. Sopclos는 데이타베이스의 저장 시스템과 직접 연결되어서 데이타베이스가 제공하는 데이타 캐슁, 트랙잭션, 포인터 탐색등의 기능의 장점을 직접 이용할 수 있지만 PCLOS의 경우에는 기본적으로 객체가 세가지 포멧(프로그래밍 언어, 가상 데이타베이스, 실제 데이타베이스)으로 존재하기 때문에 이들간 매핑과 일관성의 유지를 위한 성능상의 문제가 있다. 또한 가상 데이타베이스 단계는 CLOS의 클래스를 데이타베이스의 타입으로, CLOS의 인스턴스를 데이타베이스의 인스턴스, CLOS의 슬랏을 데이타베이스의 함수와 매핑시키는 테이블 형태로 이루어지는데 테이블 기반의 관계형 모델이므로 객체지향 모델을 지원하기 위한 부가의 데이타 구조를 필요로 한다.

### 4.2.2 Metastore

Metastore[7]는 CLOS로 짜여진 CAD 응용 프로그램이 크고 복잡한 객체들을 다루기 때문에 주기억장치만으로는 객체의 관리가 어렵다는 문제로부터 출발한 CLOS 기반 지속성 지원 시스템이다.

이 시스템의 가장 큰 특징은 상업용 데이타베이스 관리 시스템을 쓰지 않고 시스템 자체에 데이타베이스 기능을 하는 Lisp 모듈을 가지고 있다는 것이다. 따라서 이 시스템은 타입 불일치가 거의 없는 장점이 있지만 데이타베이스 기능을 완벽하게 지원하지 못한다. 즉 다중 사용자 환경이나 트랜잭션, 고장복구 등의 기능이 미미하다.

## 4.3 Sopclos와의 비교

이상에서 살펴본것과 같이 Sopclos는 CLOS의 메타객체 프로토콜을 사용하여 CLOS 프로그래밍 언어의 의미를 확장하여 사용자에게 투명한 지속성을 제공하고, 슬랏 레벨의 '`:transient`'라는 옵션과 객체 레벨의 '`is-persistent`'옵션을 제공함으로써 객체 레벨의 직교적 지속성을 제공한다. 또한 Sopclos는 객체지향 데이타베이스인 SOP의 저장 시스템과 직접 연결함으로써 데이타베이스 기능을 성능상의 문제없이 이용할 수 있는 장점이 있다. C++ 기반으로 지속성을 주는 시스템들과 PCLOS와 Metastore를 Sopclos와 비교하면 표 1과 같다.

| | | DML (C++) | Class library (C++) | Metastore | PCLOS | Sopclos |
|---|---|---|---|---|---|---|
| 직교적 지속성 | 슬랏 레벨 | O | X | O | O | O |
| | 객체 레벨 | O | X | X | X | O |
| 투명한 지속성 | | X | O | O | O | O |
| 트랜잭션 지원 | | O | O | X | O | O |
| 객체지향 모델 | | O | O | O | X | O |
| 데이타베이스 결합도 | | O | O | O | X | O |
| 동적 스키마 변환 | | X | X | O | X | X |
| ODMG-93 모델 지원 | | O | O | X | X | O |

표 1: Sopclos와의 비교

# 5 결론

본 논문에서는 CLOS의 메타객체 프로토콜을 이용하여 언어 구문은 거의 변화시키지 않으면서 CLOS 프로그래밍 언어에 데이타베이스 기능을 확장한 Sopclos를 소개하였다. 메타객체 프로토콜은 사용자 레벨의 클래스의 클래스인 메타클래스를 사용하여 이식성을 가지면서 언어의 의미를 확장할 수 있는 방법으로 시스템 내부의 클래스 계층 구조를 보여주고 메타객체의 정보를 프로그래머에게 보여준다.

Sopclos는 CLOS 해석기에 데이타베이스 라이브러리를 로딩하여 직접 데이타베이스 함수를 부를 수 있는 구조를 가진다. 트랜잭션이나 객체의 캐슁 등을 데이타베이스가 해주므로 쉽게 데이타베이스 기능을 제공할 수 있다.

향후에는 동적 스키마 변환, 디스크 상의 쓰레기 수집(garbage collection) 등에 관하여 연구할 계획이다.

# 참고문헌


[1] 안정호, 이강우, 송하주, 김형주. Soprano: 객체 저장 시스템의 설계 및 구현. **정보과학회논문지**, 제2권(제3호), 1996.

[2] Agrawal, R. and Gehani, N. H. Rationale for the Design of Persistence and Query Processing Facilities in the Database Programming Languages. In *2nd Int'l Workshop on Database Programming Languages*, 1989.

[3] Atkinson, M. P. and Buneman, O. P. Types and Persistence in Database Programming Languages. *ACM Computing Surveys*, 1987.

[4] Mary E.S.Loomis. *Object Databases : The Essentials.* Addison-Wesley Publishing Company, 1995.

[5] Adele Goldberg and David Robson. *Smalltalk-80 : The Language and its implementation.* Addison-Wesley Publishing Company, 1983.

[6] Daniel G.Bobrow Gregor Kiczales, Jim des Rivieres. *The Art of The Metaobject Protocol.* MIT Press, 1991.

[7] Arthur H.Lee and Joseph L. Zachary. Reflections on Metaprogramming. *IEEE Transaction on Software Engineering*, 1995.

[8] Ian Sommerville. *Software Engineering.* Addison-Wesley Publishing Company, 1989.

[9] Guy L. Steele JR. *Common LISP.* Digital Press, 1984.

[10] David J.Steele. *Golden Common LISP.* Addison-Wesley Publishing Company, 1989.

[11] Lamb, C. The ObjectStore Database System. *Comm. of the ACM*, 1991.

[12] Margaret A. Ellis, Bjarne Stroustrup. *The Annotated C++ Reference Manual.* Addison-Wesley Publishing Company, 1991.

[13] Andreas Paepcke. PCLOS: A Flexible Implementation of CLOS Persistence. In *Proceedings of the European Conference on Object-Oriented Programming*, pages 374–389, 1988.

[14] Andreas Paepcke. PCLOS: A Critical Review. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*, 1989.

[15] Andreas Paepcke. PCLOS: Stress Testing CLOS: Experiencing the Metaobject Protocol. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*, 1990.

[16] Andreas Paepcke. *Object Oriented Programming : The CLOS Perspective.* MIT Press, 1993.

[17] Bertbold Klaus Paul Horn Patrick Henry Winston. *LISP.* Addison-Wesley Publishing Company, 1993.

[18] R.G.G. Cattell. *The Object Database Standard: ODMG-93.* Morgan Kaufmann Publishers, 1994.

[19] Richard Mark Soley. *Object Management Architecture Guide. Object Management Group.* Inc., 1992.

[20] J. E. Richardson and M. J. Carey. Persistence in E Language: Issues and Implementation. *Software-Practice and Experience*, 1989.

[21] Alan Snyder. Encapsulation and Inheritance in Object-Oriented Programming Languages. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*, 1986.

[22] W. Richard Stevens. *UNIX Network Programming.* Prentice-Hall, 1994.

[23] Atwood T. Two Approaches to Adding Persistence to C++. In *4th Int'l Workshop on Persistent Object Systems*, 1990.

[24] S. J. White. *Pointer Swizzling Techniques for Object Oriented Database Systems.* PhD thesis, University of Wisconsin-Madison, 1994.

**부록A. Sopclos에 추가된 사용자 인터페이스**

1. `:transient` 옵션 : `defclass`를 이용한 클래스 선언시 `:metaclass`옵션[5]이 `persistent-meta-level-class`로 정의되어 있으면 슬랏 레벨의 지속성 여부를 부여할 수 있음.
2. `:is-persistent` 옵션 : `make-instance`를 이용한 객체 생성시 객체 단위의 지속성을 다시 부여할 수 있음.
3. `(with-transaction)` 매크로 : 지속객체를 접근할때 사용자에게 트랜잭션을 명시할 수 있도록 함.

**부록B. Sopclos에서 사용하는 데이타베이스 접근 함수**

1. `(sopclos-start)` : SOP 데이타베이스 시스템 초기화 함수.
2. `(sopclos-create (class &rest initargs))` : SOP 데이타베이스에 객체 생성 함수.
3. `(slot-value-from-database class object slotd)` : SOP 데이타베이스에서 슬랏 값의 추출 함수.
4. `(slot-value-to-database new-value class object slotd)` : SOP 데이타베이스의 슬랏 값 변경 함수.
5. `(sopclos-finish)` : SOP 데이타베이스 종료 함수.

[5]CLOS 문법에 있는 옵션이다.