# 객체지향 폼 시스템(Form System)의 설계

음 두 헌†


## 요 약

대부분의 데이터베이스 사용자들은 폼들을 조작함으로써 데이터베이스 시스템을 사용한다. 본 논문에서는 이러한 폼들에 관한 기술 및 각 기술이 폼의 동작에 어떠한 효과를 미치는 가에 대해 논한다. 상용제품들이 지원하는 폼 시스템들을 고찰해 본 결과 어느 것도 이상적인 폼이 갖추어야 할 모든 기능을 제공하지 못한다. 본 논문에서는 위의 상용 폼 패키지들이 갖고있는 제한성들을 대부분 극복할 수 있는 새로운 설계 방법을 제시한다. 새로운 시스템은 객체지향, 사건중심 (event-driven)이며 계층 구조의 복합 블럭 (composite block)을 지원한다. 또한 기본적으로 모드리스 폼을 생성한다. 새로운 폼 시스템에서, 폼은 톱 레벨의 블럭이며 블럭은 어토믹 또는 복합 구조로 정의되는 데 메소드들이 직접 함께 포함될 수 있다. 메시지들은 이와 같은 폼 객체들 사이에서 상호 전달되어 처리된다. 새로운 시스템 하에서의 폼의 정의는 파스칼과 비슷한 문법에 의한 스크립트로써 이루어지므로 매우 간단하다.


# Design of Object-Oriented Form System

Doohun EUM†


## ABSTRACT

The majority of database users interact with database systems by manipulating forms. This paper discusses the fundamentals underlying forms and considers how these mechanisms affect the behavior of forms. We then review the forms supported by commercial products. None of the packages reviewed provide all of the features that make up an ideal form. We propose a new design that overcomes many of the limitations observed in the packages currently available. The new system is event-driven, object-oriented, supports a hierachy of composite blocks, and is primarily modeless. Forms are defined as top-level blocks and blocks can be either atomic or composite structures with methods directly included in their definition. Messages are passed among form objects. Defining forms with the proposed system is simple because form definitions are similar to type declarations in Pascal.


## 1. 서 론

데이터베이스 시스템은 프로그래머가 아닌 일반 사용자들이 쉽게 저장된 정보를 검색할 수 있도록 하는 기능들을 제공하여야 한다. 일반적으로 이러한 목적에 쓰이는 것이 데이터베이스의 폼인데 이를 이용하여 사용자들은 데이타의 저장, 검색, 삭제 그리고 갱신 등을 쉽게 수행한다.

많은 데이터베이스 사용자들은 그들의 일상 업무에서 폼을 사용하고 있으며 이러한 폼의 대표적인 예로는 비행기 예약, 물품 주문, 세금 등에 관한 것들이다. 데이터베이스의 폼은 데이터베이스에 저장된 정보들과 연계되어 동작하므로 기존의 업무에서 쓰는 서류 형태의 폼(양식)에 비해 강력한 기능을 지원한다.

대부분의 데이타 처리를 목적으로 하는 응용 프로그램들은 폼을 통한 데이타의 조작 기능을 제공한다고 볼 수 있다. 따라서, 이러한 응용 프로그램을 설계함에 있어 그 프로그램에서 쓰일 폼과 그들의 데이타 조작을 정의하는 과정은 필수이다[1,2,3]. Sue 등은 [4] 폼을 정보내포 객체(information-holding object)로 정의하고 있다.

†정 회 원: 덕성여자대학교 전산학과 조교수
논문접수: 1994년 3월 3일, 심사완료: 1994년 4월 30일

본 논문에서는 폼의 설계에 있어 기초 기술들 즉, 폼의 양상(modality), 구조, 제어 구조 그리고 객체지향 설계 방식 등을 논한다. 위의 각 설계 기술이 폼의 동작에 미치는 작용을 설명하고 이상적인 폼을 구현하기 위한 방법을 소개한다. 상용제품들이 지원하는 폼들을 고찰해본 결과 어느 것도 이상적인 폼이 갖추어야 할 모든 기능을 제공하지는 못한다.

이상적인 폼의 설계를 위해 본 논문에서 소개된 각 설계 기술들은 새로운 폼 시스템 상에서 정의될 명세서(specification) 형태로 통합되는데 이 새로운 폼 시스템은 객체지향, 사건중심(event-driven)이며 복합 자료 구조(complex data structure)를 지원한다.

제안된 새로운 폼 시스템은 폼을 블럭들의 조합으로 정의하고 있는데 각 블럭은 필드, 배열, 레이블, 레코드, 메뉴 또는 버튼 등과 같은 폼 객체(form obejct)를 말한다. 각 블럭에 메소드(method)들이 정의될 수 있으며 폼 내에 사건(event)이 발생하면 폼의 블럭에 의한 계층 구조(composition hierarchy)를 따라 메시지(message)가 전달되면서 해당되는 블럭에 연계된 메소드들을 수행시켜 마우스 또는 키보드 조작과 같은 사건을 처리한다. 제안된 폼 시스템에서, 폼의 정의는 파스칼(Pascal)의 타입 정의(type declaration) 문법과 유사하다.

## 2. 폼의 기초 기술

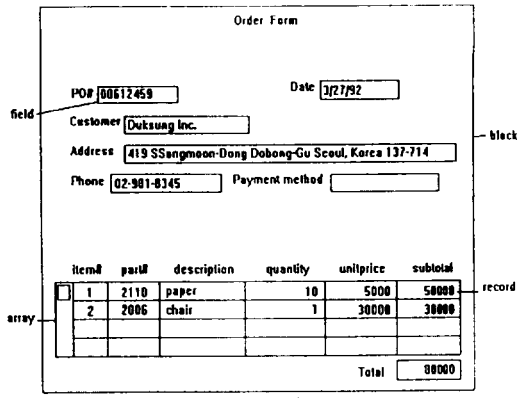
본 절에서는 폼에 관한 기초 기술들을 폼의 구성 요소, 양상(modality), 구조, 제어 구조, 객체지향 설계 방식의 순으로 고찰해 본다.

### 2.1 폼의 구성 요소

본 부절에서는 폼의 기본 개념 및 그 구성 방법에 대해 논한다. 폼의 예로서 고객과 그 고객의 물품 주문에 대한 정보를 포함하는 주문 폼을 (그림 1)에 보였다. 주문 폼은 대부분의 다른 폼들과 같이 다음에 기술되는 다양한 요소들로 구성된다.

1. 필드(field)는 데이타들을 삽입, 갱신 그리고 디스플레이할 수 있는 편집 가능 영역이다. 주문 폼 (그림 1)의 상반부는 고객과 구매에 대한 정보를 보여주는 필드들로 구성되었다.
2. 블럭이란 필드나 버튼과 같은 어토믹(atomic) 객체이거나 또는 배열이나 레코드와 같은 복합(composite) 객체를 말한다. 예로, (그림 1)의 주문 폼은 7개의 필드, 14개의 레이블 그리고 한개의 다중 레코드 블럭으로 구성된 톱 레블(top-level)의 복합 객체 블럭이다.
3. 몇개의 블럭들이 모여서 레코드나 배열과 같은 상위 블럭을 형성할 수 있다. 레코드 내의 각 구성 블럭은 필드 이름으로 접근할(access) 수 있으며 배열 내의 각 블럭은 배열의 색인(index)으로 접근한다.

위의 주문 폼의 각 블럭들은 상호 연계되어 작동된다. 예를 들어 구매 번호(PO#)가 갱신되면 그 갱신된 구매에서 주문된 물품(item)들이 검색되어 배열에 디스플레이된다. 물품 배열 내의 서브토탈(subtotal)은 단가(unitprice)와 수량(quantity)으로 부터 자동적으로 계산되며 토탈(total) 또한 서브토탈들의 합으로 계산된다.




(그림 1) 폼의 구성 요소
(Fig. 1) Form Components

(그림 2)에는 폼 시스템을 설계할 때 고려해야 할 점들을 도식화하였다. 나머지 부절들에서는 양상(modality)의 모달(modal)과 모드리스

(modeless)의 두가지 방식의 장단점을 논한 뒤 이상적인 폼 시스템이 지원해야하는 복합 객체의 필요성에 대해 기술한다. 또한, 사건중심과 순차제어(sequential control)의 두가지 폼의 제어 구조 중 사건중심 방법의 이점들을 강조한다. 마지막으로 클래스 계층 구조(class hierarchy)와 구성 계층 구조(composition hierarchy)를 포함하는 폼의 객체지향 설계 방법을 논한다.

### 2.2 양상 (Modality)

양상은 모달과 모드리스의 두가지 방법이 있는데 모달 방법이란 사용자가 사용 중인 폼의 현 상태(state)에 의해 그 시점에서 사용자가 수행시킬 수 있는 명령어들이 한정되며 이러한 명령어들이 실제 수행과 어떻게 결합되는(binding) 가를 결정하는 방식을 말한다. 반면, 모드리스는 사용자의 수행 가능한 명령어가 폼의 현 상태와 무관하며 따라서 실제 수행과 명령어의 결합이 변화하지 않는 방법을 말한다.

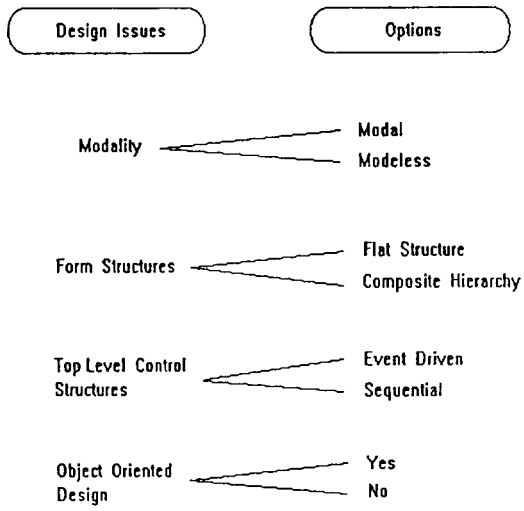
다음은 모달 폼을 이용하여 데이터베이스에 한개의 레코드를 추가시키는 절차의 예이다.

1. 사용자가 삽입 명령어를 선택한다.
2. 추가될 레코드의 정보를 입력한다.
3. 'exit' 명령어를 선택한다.(데이타는 이때 실제로 데이터베이스에 추가 됨) 질의(query)와 갱신을 하고자 할 때도 모달 폼에서는 명령어가 먼저 선택된다.

반면, 모드리스 폼의 예는 다음과 같다.

1. 사용자가 모드리스 폼에 데이타를 입력한다.
2. 삽입, 갱신 또는 질의 등의 수행하고자 하는 명령어를 선택한다. 삽입, 갱신 또는 질의 등의 명령어가 먼저 선택되지 않기 때문에 모드리스 폼에는 모드가 없으며 따라서 사용자는 폼을 사용하는 데 있어 어느 한 순간에 한가지 명령어에 국한되지 않고 여러개 중에서 선택할 수 있다.

모달 폼은 모드리스 폼에 비해 하고자 하는 일을




(그림 2) 폼의 설계시 고려점
(Fig. 2) Design Features of Forms

미리 계획하여 정해진 순서대로 명령어들을 수행시켜야 하므로 사용자 입장에서는 덜 유연하다. 또한 모달 폼은 주어진 일을 수행하는 데 있어 일반적으로 모드리스 폼보다 많은 명령어들을 수행시켜야 한다.

양상은 상대적인 것이며 따라서 완전한 모드리스 폼을 지원하는 것이 항상 바람직한 것은 아니다. 때로는 폼내의 블럭들을 모달 형태로 지원하여 어느 한 순간 한개의 블럭만을 사용할 수 있게 하는 것이 바람직할 때도 있다. 예로써, PC Oracle에서는 사용자가 다음 블럭으로 커서를 옮기기 위해 'Next Block' 명령어를 먼저 수행시켜야 한다. Accell에서는 윈도우(window)가 모달로 지원되며 사용자는 맨 앞의 윈도우 내의 폼만을 사용할 수 있다. 즉, 여러개의 윈도우들 중 아래의 윈도우 내의 폼을 사용하고자 할 때 사용자는 원하는 윈도우가 맨 앞의 윈도우가 될 때 까지 위의 윈도우들을 계속 닫아야 한다.

MacIntosh Oracle에서 지원하는 블럭은 모드리스 형태이다. 따라서 사용자는 디스플레이된 어떠한 블럭도 사용할 수 있으나 열쇄(lock)가 채워진 블럭은 사용할 수 없다. 이러한 열쇄는 응용 프로

그램 내에서 동적으로(dynamically) 제어될 수 있게 지원된다. 이와 같은 기술을 커서 삽입점 제어(cursor insertion point control)라 한다 [5,6]. 즉, 사용자가 필드 위에서 클릭(click)할때 커서의 삽입이 필드에 열쇄가 채워진 유무에 따라 커서가 삽입되는 것을 제어할 수 있다. 삽입 점을 제어함으로써 현재 수행되고 있는 테스크(task)에 적절한 명령어들 만을 허용하므로 이러한 방식은 오류 방지에 이용될 수 있다.

다이얼로그 박스(dialog box)는 사용자의 명령어 수행을 위해 필요한 입력 사항과 경고 메시지(warning message) 등을 디스플레이 할 수 있으므로 양상 제어에 이용될 수 있다.

다음은 이상적인 폼 시스템이 지원해야 할 양상 제어 방식을 위한 원칙이라 할 수 있다.

1. 프로그래머가 선택 제어할 수 있어야 한다.
2. 필요치 않을 경우 보다 쉽게 제거되어야 한다.

## 2.3 폼의 구조

한개의 폼은 여러개의 폼 객체들로 구성되는데, 여기서 폼 객체란 디스플레이 필드, 레이블, 아이콘(icon) 또는 버튼과 같은 어토믹 항목(item)이거나 또는 배열과 레코드와 같은 복합 구조를 갖는 항목일 수도 있다.

이러한 폼 객체들이 모여서 한개의 폼 블럭을 형성하는 데, 한개의 폼은 여러개 정의된 폼 레블의 블럭이다. 폼은 디스플레이 필드와 레이블들의 평적 단순 조합이거나 또는 하나의 계층 구조를 갖는 다중 레코드 블럭으로 정의될 수도 있다. 폼을 구성하기 위해 단일 레코드 블럭과 다중 레코드 블럭의 조합들을 자유자재로 지원하는 폼 시스템이 보다 유연성을 사용자에게 제공한다.

상용 제품들이 지원하는 폼들을 고찰해 본 결과 어느 것도 폼을 폼 객체들의 계층 구조로 지원하지 않음으로써 폼의 설계자에게 폼의 설계시 필요한 유연성을 충분히 제공하지 못한다.

예를 들면, 기존의 폼의 구조는 단종 레코드 블럭을 디스플레이 하는데 있어 일반적으로 스크롤러블(scrollable) 배열을 지원하지 않는다. 스크롤러블 배열은 한번에 디스플레이 하기에는 너무 많은 항목들의 디스플레이에 유용하다.

또한, 일반적으로 메뉴는 응용 패키지의 톱 레블에만 정의되나 블럭 레블의 메뉴도 필요하다. 예를 들면, 커서가 한 필드에 삽입되었을 때 메뉴가 디스플레이 되고 사용자는 메뉴에서 한 항목을 선택할 수 있는 블럭 레블의 메뉴가 바람직하다.

폼의 구성 요소를 필요에 따라 생성, 소멸시킬 수 있는 기능을 지원하는 것도 바람직하다. 예를 들어, 폼을 사용하는 동안 메뉴, 메시지 박스 그리고 알람(alarm) 등이 생성, 이용되고 소멸되는 것이 바람직하다. 즉, 전체 폼 또는 그를 포함하는 윈도우 차원의 가시성(visibility)의 제어가 아닌 구성 요소 레블의 가시성의 제어 기능이 유연하고 사용하기 쉬운 폼을 설계하는 데 있어 필수이다.

## 2.4 폼 레블 제어 구조

사용자 인터페이스인 폼을 이용하는 응용 프로그램을 개발할 때는 기존의 순차 제어와 사건중심 제어의 두가지 제어 방식을 쓸 수 있다. 순차 제어 방식으로 구현된 사용자 인터페이스에서는 사용자가 아닌 응용 프로그램이 다음에 이루어질 동작(action)을 결정한다. 즉, 이러한 방식의 사용자 인터페이스는 사용자가 어떻게 폼을 써야하는 가를 미리 지정해 놓고 있다.

반면, 사건중심 방법으로 구현된 인터페이스는 사용자의 다음 행동을 기다리고 그에 따라 적절히 동작한다. 하나의 테스크를 수행하기 위해 미리 지정된 순서에 의해 행동을 취하는 것이 아니라, 사용자에게 테스크들을 마음대로 바꿔가며 일을 수행할 수 있는 자유가 주어진다.

사건중심 인터페이스는 사건 루프(event loop)을 이용하여 구현된다. 사건 루프는 반복적으로 마우스 클릭이나 키스트로크(keystroke)와 같은 사건의 발생 유무를 검사한다. 사건이 발생하면 이 사건은 사건 루프 내에서 인지되며 처리를 위한 적절한 루틴(routine)이 수행된다. 사건 루프는 종료 플랙(finished flag)이 참일 때까지 반복된다.

다음은 사건 루프의 구조를 파스칼로 보인 것이다.

```
doneflag := FALSE;
WHILE (NOT doneflag) DO
  CASE event OF
    event1 : action1;
    event2 : action2;
    ...
    eventn : doneflag := TRUE;
  end;
```

사건중심 방법을 취하는 응용 프로그램은 모두 리스너 모델 형태로 동작한다. 모드리스 형태 일때, 사용자는 여러한 명령어도 수행시킬 수 있으며 이 명령어는 사전 루프에 의해 인지된다. 사전 루프에 의해 인지된 사용자의 명령어만을 만족하도록 설계되었을 때에 사건중심 응용 프로그램은 모델로 동작한다. 사용자가 필요한 메소드의 수행을 완료할 때까지 제어(control)는 사전 루프 내에 있으며 이 때 발생되는 다른 사전은 무시되거나 오류로 처리된다.

순차 제어 방식에 의해 구현된 응용 프로그램은 모델에서 지엽적인(locally) 사건중심 형태로 동작한다. 즉, 시스템은 특정의 모드에서 정해진 사전의 발생을 기다리며 정해진 사전이 발생될 때까지 사용자의 다른 명령어는 무시된다.

## 2.5 객체지향 설계

폼을 설계할 때 객체지향 설계 방법을 적용함으로써 얻을 수 있는 이점은 많다. 본 부절에서는 여러한 이점들을 살펴보고 폼의 객체지향 구현 방법을 제시한다. 객체지향 폼은 폼 객체들로 구성되며 각 폼 객체는 애트리뷰트들(attributes)과 행동들(behaviors)을 갖는다. 한 객체의 행동은 그 객체에 정의된 메소드들(methods)로 정의된다. 여러한 객체들은 상호 메시지(message)를 주고받음으로써 각 객체의 메소드의 수행 즉, 행동을 유도한다.

폼 객체들 중에는 설계시 자주 쓰이는 것들이 있고 여러한 객체들은 일반적으로 여러 상황에서 비슷한 행동을 갖는다. 이때 여러한 객체들을 묶어 하나의 클래스를 정의하고 이 클래스 상에 비슷한 행동에 대한 메소드들을 정의하여 객체들 간에 공유하게 하면 편리하다. 예를 들면, (고객 ID)의 배열 내의 각 물품 주문을 우리가 'item'이라 클래스를 정의하여 이 클래스의 상태(instance)로 생성할 수 있다. 즉, 클래스 'item'에 속하는 객체들이 정의되면 이들은 'item'의 모든 상태들에 의해 공유된다.

객체지향 기법은 객체의 행동을 메소드에 의해 구현하기 때문에 각 폼 객체와 메소드들은 객체와 같은 계층 구조 상에 쉽게 정의될 수 있다. 이러한 메소드들을 포함하는 폼 객체들을 묘역하는 하나의 전형 폼이 형성되며 이때 이러한 객체들이 결합되면 폼 형성에 정의된 메소드들을 자동적으로 모델 전체 폼와 행동에 정의된다.

기존의 객체지향 기법의 메소드 탐색(method search) 방법을 다음과 같이 객체지향 폼 시스템을 위해 보완할 수 있다. 메시지가 폼 객체에 전달되면 먼저 기존의 방식대로 클래스 계층 구조 상에서 자동적으로 처리된다. 여때, 클래스 계층 구조 상에서 처리할 수 없는 메시지는 구성 계층 구조 상의 상위 레블 객체에 전달된다. 위의 메시지는 구성 계층 구조 상에서 상응하는 메소드를 찾을 때까지 위로 계속 전달된다. 구성 계층 구조 상에서 메시지가 한 레블 위로 전달될 때마다 윗 레블의 객체가 "self(self object)"가 된다. 이와 같은 구성 계층 구조 상의 메시지 전달 방법은 메시지가 계층의 하위 레블로 부터 처리되는 것을 장려한다. 대부분의 상용 폼 패키지는 트리거(trigger) 기법에 의해 폼 객체에 절차(procedure)를 연계시킨다. 트리거는 폼, 블럭 또는 필드 레블에 연계될 수 있다. 예를 들어 Apcell에서는 트리거를 폼 객체에 연계시키기 위해서 그 객체의 시스템이 맡는 스크립트(script)를 고쳐야 한다. 예를 들어, 'customer_num' 필드에 트리거를 추가하여 'customer_num' 필드가 변화할 때 지정된 질의를 수행하게 하려면 다음과 같다.

```
FIELD customer_num
WHEN FIELD CHANGES
  select customer.name
  from customer
  where customer.# = customer.num
```

## 3. 상용 폼 패키지

본 절에서는 PC Oracle (version 2.0), Accell (version 1.3), 그리고 Informix (version 1.10) 등의 상용 폼 패키지들의 기능 및 특징을 고찰해 보고 <표 1>에 제안 시스템과 비교, 요약하였다.

<표 1> 상용 폼 패키지들과 제안 시스템의 기능 비교
<Table> Comparisons of Form Features for Commercial Packages and Proposed System

| 기 능 | Oracle (PC) | Accell | Informix | Proposed Form System |
|---|---|---|---|---|
| 전체 설계 | | | | |
| 폼 레벨 제어 구조 | | | | |
| 사건 중심 | X | X | | X |
| 시험설 | | | X | |
| 객체지향 설계 | | | | X |
| 상속 | | | | |
| 모드리스 | | | | X |
| 모달 | X | X | X | |
| 폼 객체 | | | | |
| 윈도우 | | X | X | X |
| 멀티 레코드 블럭 정의 | X | X | X | X |
| 폼당 다중 블럭 정의 | X | | X | X |
| 버튼 | | | | X |
| 메뉴 | | | X | X |
| 아이콘 | | | | X |
| 동작 | | | | |
| 나비게이션 중심 | X | X | X | X |
| 프레디키트 중심 | X | X | X | X |
| 트리거 | | | | |
| 폼 레벨 | X | X | | X |
| 블럭 레벨 | X | | | X |
| 필드 레벨 | X | X | X | X |
| 사용자 인터페이스 | | | | |
| 키 스트로크 | | | | |
| 함수 키 선택 | X | X | | X |
| 메뉴 선택 | | | X | X |
| 데이타 삽입 | X | X | X | X |
| 미우스 클릭 | | | | |
| 푸쉬 버튼 | | | | X |
| 필드에 커서 삽입 | | | | X |
| 메뉴 선택 | | | | X |
| 아이콘 선택 | | | | X |

PC Oracle은 텍스트 기반의 폼 패키지이며 기능 키 (function key)로써 데이타의 검색 및 브라우징을 한다 (browsing). 사용자는 기능 키들을 암기하거나 테이블을 이용하여야 한다. 윈도우가 지원되지 않기 때문에 화면 전체에 디스플레이 되는 한개의 폼을 조작한다. 폼은 한개 이상의 블럭으로 구성되며 같은 폼 내의 블럭들은 모달로 동작한다. 커서를 다른 블럭으로 옮기기 위해서 사용자는 'Next Block' 명령어를 먼저 선택해야 한다. PC Oracle에서는 폼, 블럭 또는 필드 레블에 트리거를 연계 시킬 수 있다. 폼 레블 트리거는 폼 빠져나오기 등과 같은 폼 사건이 발생하면 작동한다.

Accell에서 폼은 윈도우 상에 디스플레이 된다. 맨 앞의 윈도우가 동작 (active) 윈도우이며 동작 윈도우 밑의 윈도우를 동작시키려면 앞의 윈도우들을 닫아야 한다. 한개의 폼당 한개의 블럭만을 정의할 수 있다. 폼을 조작하기 위해 기능 키를 사용하며 기능 키 테이블이 스크린의 하단에 디스플레이 된다. 트리거는 필드와 폼 레블에 정의될 수 있다.

Informix는 텍스트 기반의 패키지이다. 사용자는 폼 위의 라인에 디스플레이 되는 메뉴를 이용하여 폼을 조작한다. 한개의 폼은 여러개의 블럭을 포함할 수 있으며 트리거는 블럭과 폼 레블에는 정의될 수 없으며 필드 레블에만 정의된다. Informix 응용 프로그램은 순차 제어 방식에 의해 구현되므로 트리거는 직접 필드에 연계될 수 없으며 직접 응용 프로그램 코드에 절차 (procedure)를 적절한 위치에 추가시켜야 한다. 따라서 트리거를 폼 객체에 직접 연계시키는 PC Oracle이나 Accell보다 트리거를 추가하기가 어렵다.

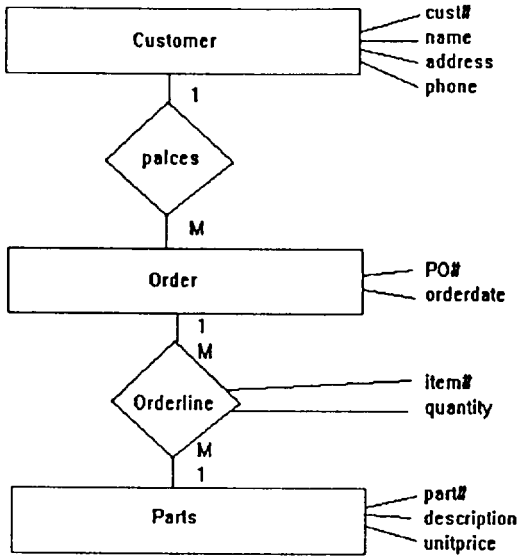
## 4. 폼 시스템의 설계

본 절에서는 폼을 폼 블럭들에 의해 구성되는 복합 자료구조로 정의하는 폼 시스템이 제안된다. 한개의 블럭은 디스플레이 필드나 버튼과 같은 단순 구조이거나 레코드나 배열과 같은 복합 구조일 수 있으며, 따라서, 하나의 폼은 폼 블럭들의 복합 계층 구조를 이루게 된다.

### 4.1 간단한 예

새로이 제안된 폼 시스템을 소개하기 위해 (그림 1)에 보인 주문 폼을 사용하는며 이 폼에 디스플레이 될 정보들은 (그림 3)의 엔티티 관계 도형 (entity-relationship diagram)과 관계 스키마 (schema)에 의해 구축된 데이터베이스로 부터 검색된다. 주문 폼은 고객 정보를 각각 지정된 필드에 디스플레이 하고 주문된 물품은 배열에 디스플레이 한다.

제안된 폼 시스템에서는 (그림 4a)에 보인 것과 같이 레코드 정의와 비슷한 형태로 폼을 정의한다. 폼의 정의는 폼을 구성하는 블럭들의 정의로 이루어 지며 각 블럭의 정의는 그에 대한 레이블, 디스플레이 필드 그리고 메소드의 세 부분으로 이루어진다.




Customer (cust#, name, address, phone)
Order (PO#, cust#, orderdate)
Orderline (PO#, part#, item#, quantity)
Parts (part#, decription, unitprice)

(그림 3) 엔티티 관계 도형과 관계 스키마
(Fig. 3) Entity-Relationship Diagram and Relational Schema

```
MainForm = RECORD @(0,0)
LABELS
  'Order Form' @(100,40);
  'PO#' @(40,50);
  'Date' @(150,50);
  'Customer' @(40,60);
  'Address' @(40,70);
  'Phone' @(40,80);
  'Payment Method' @(130,80);
  'item#' @(40,100);
  'part#' @(60,100);
  'description' @(80,100);
  'unitprice' @(130,100);
  'quantity' @(160,100);
  'subtotal' @(185,100);
  'Total' @(185,130);
DISPLAY FIELDS
  PO# : NUMERIC @(50,50) WITH $length = 8 ENDWITH;
  orderdate : DATE @(150,50);
  customer : TEXT @(70,60) WITH $length = 18 ENDWITH;
  address : TEXT @(70,70) WITH $length = 36 ENDWITH;
  phone : TEXT @(70,80) WITH $length = 12 ENDWITH;
  total : MONEY WITH $length = 9 @(180,130);
  paymethod : TEXT @(170,80) WITH $length = 8 ENDWITH;
  items : SCROLLABLE ARRAY @(35,110) [1..4] OF item;
METHODS {MainForm}
  BEGIN
    ON update() FROM PO# DO
      select item#, part#, quantity
      from Orderline
      where Orderline.PO# = PO#
      into items[].item#, items[].part#, items[].quantity;
    ON update() FROM subtotal DO
      VAR i:INTEGER;
      BEGIN
        total := 0;
        i := 1;
        WHILE items[i] <> nil DO
          BEGIN
            total := total + items[i].subtotal;
            i := i + 1;
          END;
        END;
    END; {of MainForm methods}
  ENDRECORD; {of MainForm}
```

(그림 4a) MainForm의 정의
(Fig. 4a) MainForm Definition

먼저, 부모 블럭(parent block)인 'MainForm' 내의 각 레이블 이름과 그 상대적인 위치를 정의한다. 'Order Form' 레이블은 "Order Form"의 스트링(string)과 (100,40)의 상대 위치에 의해 결정된다. 다음으로 디스플레이 필드들은 그들의 타입, 위치 그리고 길이(length)로써 정의된다. 예를 들어, 'PO#' 필드는 'NUMERIC' 타입이며 위치는 (50,50)이고 길이는 8 (WITH $length=8)이다. 'WITH' 다음의 설정(setting)은 한 블럭 타입에 디폴트(defalt)로 설정된 값을 변경시키는 자격자(qualifier)이다.

위와 같은 자격자에 의한 덮어쓰기(overriding)는 실제적으로 단 한번 쓰이는 새로운 서브클래스

를 정의하게 되는 데 이 서브클래스에 대해서는 새로운 클래스 이름이 주어지지 않고 약간의 수정을 통한 기존의 클래스로 정의되기 때문에 편리하다.

스크린의 마지막 필드인 'items'는 (35,110)의 위치에 스크롤러블 배열로 정의되며 이 필드는 (그림 4b)에 정의된 'item' 블럭 타입의 네개의 요소(element)들을 갖는다. (그림 4b)에 'item' 타입을 한개의 레코드 블럭으로 정의하였다.

```
item = RECORD @(0,0)
DISPLAY FIELDS
  item#          : NUMERIC @(0,0) WITH $length = 5 ENDWITH;
  part#          : NUMERIC @(20,0) WITH $length = 5 ENDWITH;
  description    : TEXT @(40,0) WITH $length = 5 ENDWITH;
  unitprice      : MONEY @(80,0) WITH $length = 5 ENDWITH;
  quantity       : NUMERIC @(120,0) WITH $length = 5 ENDWITH;
  subtotal       : MONEY @(145,0) WITH $length = 5 ENDWITH;
METHODS
  BEGIN
    ON update() FROM unitprice, quantity DO
        subtotal := unitprice * quantity;
    ON update() FROM part# DO
        select unitprice, description
        from Part
        where Part.part# = part#
        into item.unitprice, item.description;
  END; {of methods}
ENDRECORD; {of item}
```

(그림 4b) Item의 정의
(Fig. 4b) Item Definition

'MainForm' (그림 4a)에 정의된 첫번째 메소드에 의해 구입 주문 번호(PO#)가 변경될 때마다 자동적으로 변경된 번호에 대해 주문된 물품들을 검색한다. 이 메소드는 구입 주문 번호(PO#) 객체에 연계되며 물품 정보를 검색하기 위해 내장된 SQL(embedded SQL) 문장을 이용하고 있다. 다음 메소드는 서브토탈(subtotal)과 토탈(total)의 갱신된 필드 값들을 다음과 같이 계산한다. 'Item' (그림 4b)의 수량(quantity)과 단가(unitprice)의 값이 갱신되면 이 갱신된 필드가 구성 계층 구조를 따라 상위로 메시지를 보낸다. 이 때, 'item'이 수량(quantity)과 단가(unitprice)로부터 메시지를 전달받아 서브토탈(subtotal)을 재계산하기 위한 다음의 메소드를 수행한다.

```
ON update() FROM unitprice, quantity DO
    subtotal := unitprice * quantity;
```

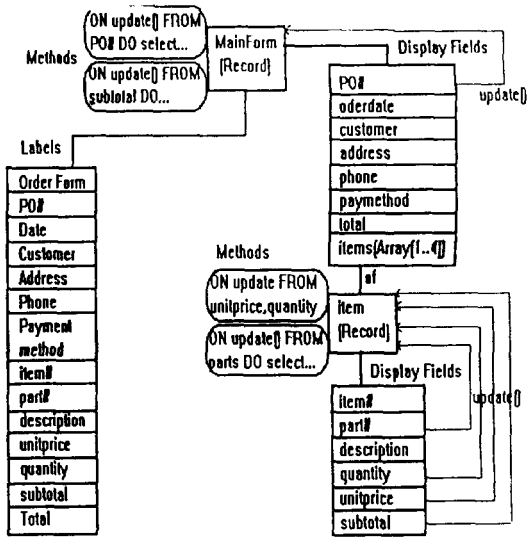
또한 'subtotal'이 갱신되면 'subtotal' 필드는 역시 구성 계층 구조를 따라 update 메시지를 한 단계 위로 보내며 이 메시지는 다음의 'total'을 계산하는 'MainForm'의 메소드(그림 4a)에 의해 부합되어(match) 수행된다.

```
ON update() FROM subtotal DO...
```

기존의 객체지향 방법과는 달리 수신자(receiver)가 송신자(sender)를 지정하고 송신자는 수신자를 명시하지 않는다. 송신자는 단지 폼 구조를 인식치 않고 메시지를 자신의 부모에게 방송(broadcasting)한다. 메시지는 계층 구조의 아래에서 위로의 단일 방향으로만 전달되며 메시지에 대한 회답(return) 값은 없다.

'Item'에 추가된 마지막 메시지는 부품 번호(part number)가 변경되면 물품의 단가(unitprice)와 기술(description) 값을 검색하는 것이다.

(그림 5)는 주문 폼을 구성하는 블럭들의 계층 구조를 보인 것이다.




(그림 5) 주문 폼 블럭들의 계층 구조
(Fig. 5) Hierarchy of Order Form Blocks

폼 객체들은 사각형으로 메소드들은 모서리가 굽은 사각형으로 표시하였으며 화살표로 표시된 선들

은 계층 구조를 따라 위로 전달되는 메시지들을 나타낸다.

### 4.2 폼에 추가되는 기능

위의 주문 폼의 예에서는 레이블, 필드 그리고 메소드 등을 정의하였으며 본 부절에서는 버튼과 메뉴를 정의하는 블럭들을 소개한다. 기존의 폼 시스템에서는 이러한 블럭들을 톱 레블에서만 정의할 수 있는 데 반해 여기서 제안되는 새로운 시스템에서는 어느 레블에서도 이들의 정의가 가능하다. 또한 블럭들의 가시성과 촛점(focus)을 제어할 수 있게 하였다.

버튼은 레이블, 위치(location) 그리고 버튼을 눌렀을 때 동작되는 메소드로 이루어진 블럭이다. 메뉴 역시 블럭으로 정의되며 버튼의 배열로 정의된다. 메뉴 내의 버튼들은 동질성(homogeneous)이며 메소드가 추가되거나 기존의 설정(setting)이 바뀌게 되면 이질성(heterogeneous) 객체들로 변환된다. 메뉴 내의 각 버튼에 레이블을 삽입시키기 위해 다음(그림 6)과 같이 각 배열의 색인에 'METHOD'와 'WITH' 문장을 연계시킨다.

```
editmenu: ARRAY @(30,0)
 METHODS
  BEGIN
ON init() DO visibility := 0
  END
 [1..4] OF BUTTON
 [1] @(0,5) WITH $blabel = "insert" ENDWITH
  METHODS
   BEGIN ON pushed() DO insert() END;
 [2] @(0,10) WITH $blabel = "update" ENDWITH
  METHODS
   BEGIN ON pushed() DO update() END;
 [3] @(0,15) WITH $blabel = "delete" ENDWITH
  METHODS
   BEGIN ON pushed() DO delete() END;
 [4] @(0,20) WITH $blabel = "clear" ENDWITH
  METHODS
   BEGIN ON pushed() DO clear() END;
```

(그림 6) editmenu의 정의

(Fig. 6) editmenu Definition

또한, 블럭들의 디스플레이 및 활성(active) 상태를 각 블럭에 가시성(visibility)과 활성 플레그(active flag)를 연계시킴으로써 제어할 수 있다. 가시성 플레그는 블럭의 디스프레이 여부를 결정하고 활성 플레그는 활성 여부를 결정한다. 즉, 활성 플레그는 블럭의 양상을 제어하는 데 쓰인다.

위의 두 플레그를 이용해 다음과 같은 블럭의 세 가지 상태를 정의할 수 있다.

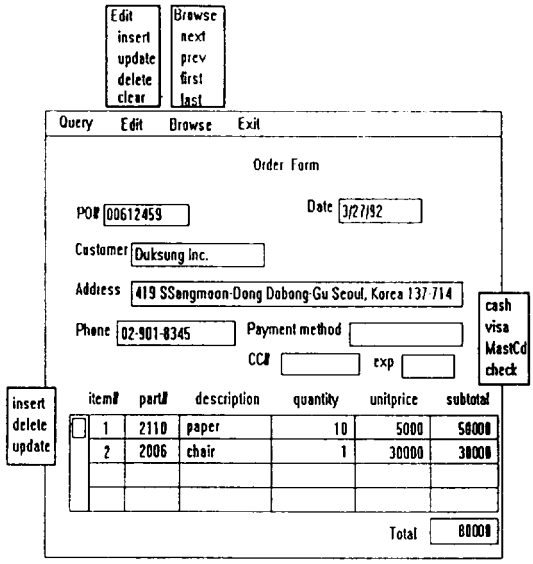
| state | visibility | active/inactive |
|---|---|---|
| invisible | 0 | 0 |
| unfocused | 1 | 0 |
| focused | 1 | 1 |

가시성 플레그로 팝업(pop-up) 메뉴를 지원할 수 있으며 활성 플레그를 이용하면 모달 메뉴를 지원할 수 있다. 즉, 활성화 하고자 하는 메뉴 이외의 모든 메뉴의 활성 플레그를 오프(off) 상태로 하면 된다. 디스플레이 필드는 가시성 플래그를 토글(toggle)함으로써 필요할 때에만 보이게 할 수 있다.

사용자 입력의 촛점(focus)이란 폼 내의 커서의 위치를 말하는 데 본 절에서 제안된 폼 시스템에서는 마우스 커서를 쓰는 것을 기본으로 하되 이는 프로그래머가 원한다면 구성 계층 구조의 모든 레블에서 프로그램으로 제어할 수 있어야 한다.

(그림 7)에 풀다운(pull-dowm) 그리고 팝업(pop-up) 디스플레이 필드들이 추가된 주문 폼을 보였다.

주문 폼의 상단에 메뉴 바(menu bar)가 추가되었으며 'Edit'과 'Browse'에 풀다운 메뉴 형태가 지원된다. 왼쪽 하단에는 주문된 물품에 대한 정보를 편집하기 위한 팝업 메뉴가 지원되며 오른쪽 중간의 팝업 메뉴는 커서가 'payment method' 필드에 삽입될 때 디스플레이되어 사용자가 메뉴 항목들 중 하나를 선택할 수 있게 한다. 사용자가 신용카드를 선택하면 신용 카드 번호(CC#)와 만기일(exp) 필드가 디스플레이되어 추가 정보를 수집한다.



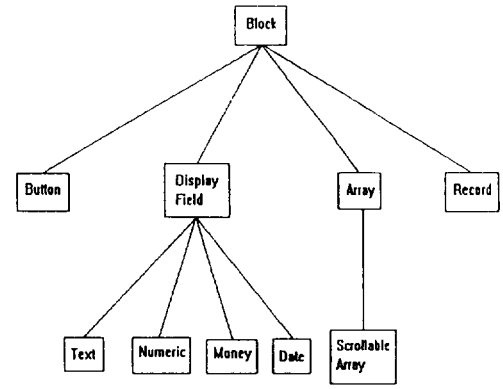

(그림 7) 메뉴와 팝업 필드가 추가된 주문 폼
(Fig. 7) Order Form with Menus and Pop Up Fields

### 4.3 객체지향 계층 구조

새로운 폼 시스템은 두개의 계층 구조를 사용한다. 앞에서 소개된 구성 계층 구조 외에 블럭 계층 구조를 지원하는 데 이는 기존의 객체지향 프로그래밍 언어의 클래스 계층 구조와 비슷하다. 폼을 위한 블럭 계층 구조를 (그림 8)에 보였다.

애트리뷰트들과 메소드들은 계층 구조를 따라 상속된다. 모든 블럭은 위치(location) 정보를 갖고 있으므로 이 위치 애트리뷰트는 루트(root)인 블럭(Block) 클래스에서 정의된다. 예를 들어 'Numeric' 클래스는 길이(length)와 위치 애트리뷰트를 갖게 되는 데 이는 블럭 클래스에서 정의되어 상속 받은 것이다.

메시지가 한 블럭에 전달되면 이 메시지의 처리가 위의 블럭 계층 구조 상에서 기존의 객체지향 기법에 의해 시도 된다. 만일 메시지가 블럭 계층 구조 상에서 처리가 되지 못하면 메시지를 이번에는 앞에서 소개한 구성 계층 구조 상의 수신자의 상위 블럭에 전달되고 이 계층 구조를 따라 처리될 때까지 전송 된다.




(그림 8) 블럭 계층 구조
(Fig. 8) Block Hierarchy

### 4.4 문법(syntax)과 의미(semantics)

폼의 구조를 정의하기 위한 스크립트에 대한 문법을 (그림 9a)에 정리하였고 (그림 9b)에는 메소드를 정의하기 위한 문법을 소개하였는 데 메소드 정의 문법은 파스칼(Pascal) 문법을 그대로 사용하였다.

```
block_declaration -> block_id = block;
block -> TEXT @loc {settings}{methods}
    | NUMERIC @loc {settings}{methods}
    | MONEY @loc {settings}{methods}
    | DATE @loc {settings}{methods}
    | RECORD @loc {lables}{display_fields}
      {settings}{methods} ENDRECORD
    | {SCROLLABLE} ARRAY @loc
      {settings}{methods} [range] OF block
    | BUTTON {settings}{methods}
      { [integer] @loc {settings}{methods} }*
    | block_id
lables -> LABELS 'string' @loc {; 'string' @loc}*
display_fields -> DISPLAY_FIELDS field_id:block {; field_id:block}*
settings -> WITH $parameter = integer-constant
{; $parameter = integer-constant}* ENDWITH
methods -> METHODS BEGIN method {; method}* END
method -> ON event DO {type_declaration}{var_declaration} method_statements
method_statements -> statement | BEGIN statement {; statement}* END
loc -> (integer-constant, integer-constant)
block_id -> id
field_id -> id
parameter -> id
```

(그림 9a) 폼의 문법
(Fig. 9a) Syntax of Forms

```
type_declaration -> TYPE type definition {;type_definition}*
type_definition -> type id = type
type -> type_id
    | INTEGER
    | REAL
    | CHARACTER
    | BOOLEAN
    | ARRAY [range] OF type
    | RECORD declaration_list END
```

```
range -> integer-constant..integer-constant [,integer-constant..integer-constant]*
var declaration -> VAR declaration_list
declaration_list -> declaration [; declaration]*
declaration -> identifier_list : type
identifier_list -> id [, id]*
type_id -> id
var id -> id
```

(그림 9b) 메소드 정의를 위한 파스칼 문법
(Fig. 9b) Pascal Syntax for Method Definitions

## 5. 결 론

상용 폼 패키지들을 분석해 본 결과 그들로 생성시킬 수 있는 폼들은 제한적이었다. 즉, 폼의 구조는 플랫(flat) 형태이고 사용자의 유연성을 제한하는 모달 방식이 많았으며 메뉴를 블럭 레블에서 정의할 수 없었다. 분석된 네개 상용 폼 시스템들중 세개는 사건중심, 한개는 순차 제어 방식을 사용한다.

위의 분석을 바탕으로 이상적인 폼의 명세서(specification)을 제시하였는 데 이는 사건중심 제어 방식, 객체지향 설계 등을 채택하며 복합 자료구조를 지원하다. 또한 기본적으로 모드리스 폼을 생성한다. 새로운 폼 시스템에서 폼은 툽 레블의 블럭이며 블럭은 어토믹 또는 복합 구조로 정의되는 데 메소드들이 직접 함께 포함될 수 있다. 메시지들은 이와 같은 폼 객체들 사이에서 전달된다. 메시지는 먼저 블럭(클래스) 계층 구조 상에서 처리가 시도되며 이 계층 구조에서 처리가 되지 않는 메시지는 구성 계층 구조 상에서 처리될 때까지 위로 전달된다. 따라서 한 폼의 행위(behavior)는 그 구성 요소들 간의 메시지 교환에 의한 상호 작용에 의해 정의된다.

새로운 시스템 하에서의 폼의 정의는 파스칼과 비슷한 문법에 의한 스크립트로써 이루어지므로 매우 간단하다.

## 참 고 문 헌


[1] Parsaye, K., Chignell, M., Khosrafian, S., and Wong, H., *Intelligent Databases*, John Wiley & Sons, pp. 72-83, 1989.

[2] Embley, D.W., NFQL: The Natural Forms Query Language, ACM trans. on Database System, Vol.14, No.2, June 1989, pp.168-211.

[3] Wilson E., A Comparison of Interfaces: Computer, Designer, and User, DEXA' 92, pp.326-331, 1992.

[4] Shu, N.C., Wong, H.K.T., and Lum, V. Y., Forms Approach to Requirements Specification for Database Design, SIGMOD 1983, pp. 161-172, 1983.

[5] *Oracle For MacIntosh Manual*, Version 1. 0, Part Number 5117-V1.0, Oracle Corporation, 1988.

[6] Rolland, F.D., *Relational Database Management with Oracle*, Addison-Wesley, 1990.

[7] Batini C., Ceri S., and Navathe S., *Conceptual Database Design*, Benjamin Cummings, 1992.

[8] Cattell, R.G.G., *Object Data Management*, Addison-Wesley, 1991.

[9] Date, C.J., *Database Systems*, Addison-Wesley, 1986.

[10] *Informix-4GL User's Guide*, Version 1. 10, Informix part number 200-501-0004, Informix Corporation, 1988.

[11] *Oracle SQL* Forms Designer's Tutorial*, Oracle Reference Manual Number 3302-V2.0, Oracle Corporation, 1987.

[12] Pohl, I., *C++ for C Programmers*, Benjamin/Cummings, 1989.

[13] Rolland, F.D., *Relational Database Management with Oracle*, 2nd Edition, Addison-Wesley, 1992.

[14] *Accell Integrated Application Development System*, Unify Reference Manual Number 254A, Release 1.3, Unify Corporation, 1987.


## 부 록

(그림 1)에 보인 간단한 주문 폼에(그림7)과 같이 풀 다운과 팝업 메뉴 및 팝업 필드들인 CC#와 exp들이 추가된 폼에 대한 정의를 다음에 보인다.

```
MainForm = RECORD @(0,0)

 LABLES
  'Order Form' @(100,40);
  'PO#' @(40,50);
  'Date' @(150,50);
  'Customer' @(40,60);
  'Address' @(40,70);
  'Phone' @(40,80);
  'Payment method' @(130,80);
  'item#' @(40,100);
  'part#' @(60,100);
  'description' (80,100);
  'unitprice' @(130,100);
  'quantity' @(160,100);
  'subtotal' @(185,100);
  'Total' @(185,130);

 DISPLAY FIELDS
  PO#  : NUMERIC @(50,50) WITH $length = 8 ENDWITH;
  orderdate : DATE @(150,50);
  customer : TEXT @70,60) WITH $length = 18 ENDWITH;
  address  : TEXT @(70,70) WITH $length = 36 ENDWITH;
  phone  : TEXT @(70,80) WITH $length = 12 ENDWITH;
  total  : MONEY WITH $length = 9 @(180,130);
  items  : SCROLLABLE ARRAY @(35,110) [1..4] OF item;
  paymethod : TEXT @(170,80) WITH $length = 8 ENDWITH;

  CCinfo  : REDORD @(120,90)
   LABELS
    'cc#' @(0,0);
    'exp' @(60,0);
   DISPLAY FIELDS
    CC#  : TEXT @(10,0) WITH $length = 8 ENDWITH;
    expdate : TEXT @(70,0) WITH $length = 5 ENDWITH;

    METHODS
     BEGIN
      ON init() DO visibility := 0;
      ON update() DO if(expDate > orderdate) THEN
       PRINT('credit card expired');
     END
ENDRECORD; {CC info fields}

mainmenu : ARRAY @(30,30) [1..4] OF BUTTON
 [1] @(0,0) WITH $blabel = 'Query' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO query—procedure() END;
 [2] @(30,0) WITH $blabel = 'Edit' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO editmenu.visibility := 1 END;
 [3] @(55, 0) WITH $blabel() = 'Browse' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO browse.visibility := 1 END;
 [4] @(80,0) WITH $blabel = 'Exit' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO exit—form() END;

editmenu : ARRAY @(30,0)
 METHODS BEGIN ON init() DO visibility := 0 END
 [1..4] OF BUTTON
 [1] @(0,5) WITH $blabel = 'insert' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO insert() END;
 [2] @(0,10) WITH $blabel = 'update' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO update() END;
 [3] @(0,15) WITH $blabel = 'delete' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO delete() END;
 [4] @(0,20) WITH $blabel = 'clear' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO clear() END;

browsemenu : ARRAY @(55,0)
 METHODS BEGIN ON init() DO visibility := 0 END
 [1..4] OF BUTTON
 [1] @(0,5) WITH $blabel = 'next' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO next() END;
 [2] @(0,10) WITH $blabel = 'prev' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO prev() END;
 [3] @(0,15) WITH $blabel = 'first' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO first() END;
 [4] @(0,20) WITH $blabel = 'last' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO last() END;

paymenu  : ARRAY @(225,65) [1..6] OF BUTTON
 METHODS
  BEGIN ON PUSHED() DO paymethod := $blabel END;
 [1] @(0,0) WITH $blabel = 'cash' ENDWITH;
 [2] @(0,10) WITH $blabel = 'charge' ENDWITH;
 [3] @(0,20) WITH $blabel = 'check' ENDWITH;
 [4] @(0,30) WITH $blabel = 'VISA' ENDWITH;
 [5] @(0,40) WITH $blabel = 'AmExp' ENDWITH;
 [6] @(0,50) WITH $blabel = 'MastCd' ENDWITH;

METHODS {MainForm}
 BEGIN
 ON update() FROM PO# DO
  select item#, part#, quantity
  from Orderline
  where Orderline.PO# = PO#
  into items[].item#, items[].part#, items[].quantity;

 ON update() FROM subtotal DO
  VAR i:INTEGER;
  BEGIN
   total := 0;
   i := 1;
   WHILE items[i] <> nil DO
    BEGIN
     total := total + items[i].subtotal;
     i := i + 1;
    END;
  END;

 ON CURSOR ENTERS paymethods DO paymenu.visibility := 1;
   ON ((paymethod = VISA) OR (paymethod = AmExp) OR
(paymethod = MastCd)) DO
    CCinfo.visibility := 1;
 END; {of MainForm methods}
ENDRECORD; {of MainForm}


item = RECORD @(0,0)
 DISPLAY FIELDS
  item#  : NUMERIC @(0,0) WITH $length = 5 ENDWITH;
  part#  : NUMERIC @(20,0) WITH $length = 5 ENDWITH;
  description : TEXT @(40,0) WITH $length = 5 ENDWITH;
  unitprice : MONEY @(80,0) WITH $length = 5 ENDWITH;
  quantity : NUMERIC @(120,0) WITH $length = 5 ENDWITH;
  subtotal : MONEY @(145,0) WITH $length = 5 ENDWITH;
```

```
itemmenu : ARRAY @(0,110)
 METHODS
  BEGIN
   ON init() DO visibility := 0;
  END
 [1..3] OF BUTTONS
 [1] @(0,0) WITH $blabel = 'insert' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO insert() END;
 [2] @(0,10) WITH $blabel = 'update' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO update() END;
 [3] @(0,20) WITH $blabel = 'delete' ENDWITH
  METHODS
   BEGIN ON PUSHED() DO delete() END;

METHODS
 BEGIN

 ON update() FROM unitprice, quantity DO
  subtotal := unitprice * quantity;

 ON update() FROM part# DO
  BEGIN
   select unitprice, description
   from Part
   where Part.part# = part#
   into item.unitprice, item.description;
  END;

 ON CURSOR ENTERS item# DO itemmenu.visibility := 1;

END; {of methods}
```


**음 두 헌**

1984년 서강대학교 전자공학과 졸업(학사).
1987년 오레곤 주립대학교 대학원 컴퓨터공학과(공학석사).
1990년 오레곤 주립대학교 대학원 컴퓨터공학과(공학박사).
1991년~1992년 한국전자통신연구소 선임연구원.
1992년~현재 덕성여자대학교 전산학과 조교수.
관심분야 : 객체지향 시스템 (특히, 데이타베이스 사용자 인터페이스 및 프로그래밍 환경)