정보처리학회 논문지 제5권 제9호(98.9)

시나리오를 이용한 객체 지향 시스템의 통합 테스트

김 은 주† · 최 은 만††

요 약

객체 지향 시스템의 통합 테스트를 위한 테스트 케이스는 Method/Message의 경로와 Method의 실행에 따른 시스템의 상태를 포함하여야 한다. 이런 테스트 케이스를 추출하기 위해서는 원시 코드만으로는 부족하고 시스템의 동적 모델링 단계에서 사용되었던 시나리오를 이용하여야 한다. 그 이유는 시나리오를 이용할 경우 사용자가 시스템에 행할 수 있는 동작의 종속성 분석을 통하여 테스트 케이스의 수를 줄일 수 있기 때문이다. 본 논문에서는 시나리오를 객체 지향 시스템의 통합 테스트에 활용하는 방안에 대하여 연구하였다. 이 방안을 이용하면 객체간의 상호작용을 테스트하기 위한 테스트 케이스의 생성과 테스트 결과의 비교를 컴퓨터를 이용하여 할 수 있고 테스트 케이스의 수도 줄일 수 있다.

Integrated Test of Object-Oriented System Using Scenario

Eun Joo Kim† · Eun Man Choi††

ABSTRACT

Test case for object-oriented system's integrated testing must be contained both method/message path and result of method execution. For this test case extraction, we should use scenario of dynamic modeling level. Because we can reduce account of test case through user action's dependency analysis. This paper suggest test case generation method using dynamic modeling scenario. There are two advantages of this method are two. First, we can use computer when we generate test case for object's interaction and test result compare. Second, we can reduce testcase amount.

1. 서 론

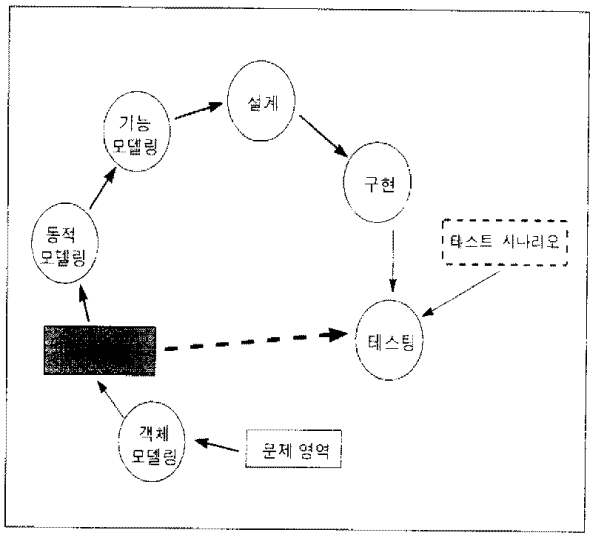
소프트웨어 테스팅은 소프트웨어 생명주기의 한 단계로서, 개발된 소프트웨어의 문제점을 찾아내는 과정이면서, 그 소프트웨어에 대한 신뢰도를 측정하는 방법이다[14].

객체 지향 방법은 구조적 방법과는 달리 상속, 캡슐화, 다형성과 같은 속성을 갖고 있으므로 객체 지향 방법으로 생성된 소프트웨어는 구조적인 방법으로 생성된 소프트웨어와는 다른 방법으로 테스트되어야 한다. 본 논문에서는 다루고자하는 통합 테스트의 경우 구조적 방법에서는 시스템을 구성하는 모듈의 인터페이스와 결합을 테스트하지만 객체 지향 시스템에서는 클래스에 속한 멤버 함수들의 메시지 전달에 의한 다른 멤버 함수의 호출 순서와 그 멤버 함수의 실행에 따른 시스템의 상태를 테스트한다. 이런 테스트 케이스를 추출하기 위해서는 원시 코드만으로는 테스트 케이스를 추출하는데 필요한 정보를 추출하기 어렵고 시나리오를 이용하여야한다. 그 이유는 시나리오가 연속된 멤버 함수의 집합으로 테스트 케이스를 형성할 때 순서를 결정하는데 도움을 주기 때문이다.

시나리오는 사용자와 시스템의 상호 작용을 사용자가 이해하기 쉽게 자연어로 기술한 것이다[1]. (그림 1)은

†준 회 원 : 세명대학교 강사
††정 회 원 : 동국대학교 컴퓨터공학과 교수
논문접수 : 1997년 8월 8일, 심사완료 : 1998년 7월 10일

OMT의 소프트웨어 개발 과정을 나타낸 것으로 크게 모델링, 설계, 구현, 테스팅의 네 단계로 구성된다. 모델링 단계에서 객체 모델링은 문제 영역에서 추출한 객체들간의 관계를 반영한 객체도를 생성하고 동적 모델링에서는 시스템에 요구되는 동적인 특성에 따라 상태도와 사건 추적도를 생성한다. 기능 모델링은 데이터가 어떤 흐름을 갖고 변형되는지를 보여주는 자료 흐름도를 생성한다.



(그림 1) 시나리오의 용도
(Fig. 1) The use of scenario

(그림 1)에서 시나리오는 동적 모델링 단계의 처음에 생성되어 정확한 동적 모형을 추출할 때만 사용되고 이후에 다시 쓰이지 않는다. 객체 지향 프로그램의 테스트 단계에서는 기능 테스트를 위하여 시스템의 기능을 테스트하기 위한 테스트 시나리오를 다시 작성하는 경우가 많다. 이 논문에서는 이런 중복되는 일을 하지 않도록 (그림 1)과 같이 테스트 시나리오를 작성하는 과정을 생략하고 동적 모델링 과정의 시나리오를 객체 지향 프로그램의 통합 테스트에 활용하는 방법과 이에 대한 지원 방안, 도구를 다루고자 한다.

본 논문의 구성은 2장에서는 객체 지향 시스템의 테스트에 대한 분류와 기존의 통합 테스팅의 문제점에 대하여 살펴본다. 3장에서는 시나리오를 이용한 문제 해결 방법을 예를 들어 알아보고 4장에서는 자동화된 테스트 케이스를 추출하기 위한 시스템을 설계한다. 5장에서는 실제로 프로그램을 테스트 한 후 테스트 결과와 테스트 케이스를 비교하고 6장에서는 결론을 맺고자 한다.

2. 객체 지향 시스템의 테스트

2.1 객체 지향 시스템의 테스트 분류

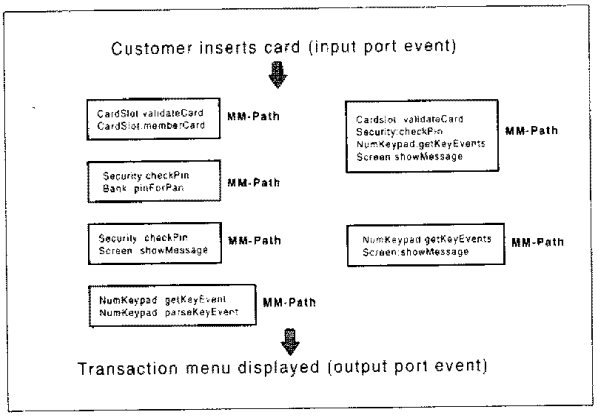
객체 지향 시스템의 특성은 추상화, 상속성, 다형성과 서로 다른 클래스 혹은 같은 클래스내의 멤버 함수들간의 다양한 형태의 상호작용에 의해 기능이 수행된다는 점이다. 이런 특성들이 테스트에서 갖는 의미는 추상화는 정보은닉을 통하여 수행 경로보다는 인터페이스의 입력과 출력을 강조하므로 화이트 박스 테스트보다 블랙 박스 테스트에 더 비중을 둔다는 것이고 상속성은 상위 클래스의 반복 테스트가 필요하게 하고 다형성은 동적 결합에 따른 수행 경로를 증가시켜 테스트를 복잡하게 한다는 것이다. 이런 객체 지향 소프트웨어의 특성은 기존의 구조적 테스트와는 다른 분류가 필요하므로 <표 1>과 같이 나타내었다. 테스트 단위에 따른 분류에서는 구조적 소프트웨어의 모듈과 비슷한 멤버 함수의 테스트가 가장 작은 단위이지만 멤버 함수를 클래스에 포함시켜서 클래스를 가장 작은 테스트 단위로 보는 경우도 있다. 상호 작용 테스트에서는 테스트 케이스가 멤버 함수의 호출 순서 또는 멤버 함수의 조합이 된다. 그 이유는 멤버 함수의 실행 경로가 입력 매개변수뿐만 아니라 객체의 상태에 의해서도 결정되는데 객체의 상태를 결정하는 데이터의 멤버는 다른 멤버 함수의 실행에 의존하기 때문이다.

〈표 1〉 객체 지향 시스템의 테스트 분류
〈Table 1〉 Test methods for object-oriented system

분류 기준	종 류	테스트 목 적
테스트 단 위	멤버 함수	멤버 함수의 기능
		멤버 함수의 구조
	클래스	멤버 값의 변화
		멤버 함수간의 상호 작용
		클래스의 기능
	통합	같은 클래스에 속한 멤버 함수간의 상호 작용
		서로 다른 클래스에 속한 멤버 함수간의 상호 작용
	시스템	시스템의 기능
테스트 케이스	블랙 박스	클래스의 기능
		클래스의 상호 작용에 따른 기능
	화이트 박스	원시 코드에 기반을 둔 경로
	상호 작용	같은 클래스에 속한 멤버 함수간의 상호 작용
		서로 다른 클래스에 속한 멤버 함수간의 상호 작용
테스트 대 상	명세 기반	요구 분석 명세에 프로그램이 얼마나 충실한가를 테스트
	코드 기반	멤버 함수의 논리적인 구조

2.2 객체 지향 시스템의 통합 테스팅의 테스트 케이스 생성과 문제점

객체 지향 시스템의 시스템 테스트와 통합 테스트의 차이는 시스템 테스트는 사용자와 시스템간의 눈에 보이는 인터페이스를 통해서만 시스템을 테스트하는 것이고 통합 테스트는 시스템 단계에서는 보이지 않는 메모리 이벤트나 조건도 테스트하는 것이다. 그러므로 기존의 시스템의 통합 테스트는 원시 코드로부터 (그림 2)와 같은 방향성 그래프를 추출하여 (그림 3)과 같은 MM Path를 구하여 이들의 조합으로 테스트 케이스를 생성하였다[10]. 그런데 이런 방법은 다음과 같은 문제점이 발생한다.



(그림 3) 메소드/메시지 패스[10]
(Fig. 3) Method/Message Paths[10]

① 메시지의 전달을 통한 멤버 함수의 호출 순서만 테스트하고 멤버 데이터의 경우는 배제하였다. 즉, 메시지에 의한 멤버 함수의 호출 순서만 테스트하고 그 멤버 함수의 실행 결과에 대한 테스트는 따로 했기나 그 멤버 함수가 맞게 실행한다는 가정 하에 테스팅이 이루어진다.

② 통합 테스팅은 객체 상호간의 동적인 작용을 테스팅하는것이므로 모든 경우를 고려하여 자동화된 테스팅을 할 수 없다.

③ 상태 기반 통합 테스팅에서 상대란 객체의 멤버의 값을 의미하지만 실제로 그 값들을 추출하기가 어렵다. 그 이유는 상태 표현에서 멤버 함수의 실행 결과를 구체적인 자료의 값이 아닌 추상적인 상태로 표현하기 때문이다. 그러므로 테스트 케이스를 이용하여 테스트를 한 후 그 결과를 컴퓨터를 이용하여 비교할 수가 없다.

3. 시나리오를 이용한 객체 지향 시스템의 통합 테스트

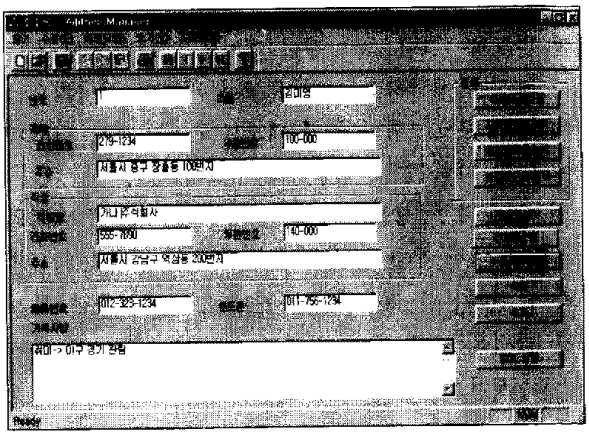
본 논문에서는 위의 문제점들을 해결하기 위하여 시나리오에서 테스트 데이터를 추출한 후 사용자의 시스템에 대한 작용의 종속성 여부를 결정하여 테스트 케이스를 생성하고자 한다. 종속성이란 일련의 문장들이 정해진 순서에 따라 수행하도록 하는 성질이다. 여기에서는 시나리오를 이용한 사용자의 시스템에 대한 동작의 종속성을 분석하여 테스트 케이스의 범위를 축소시킨 후 코드를 이용한 멤버 함수간의 종속성 분석을 통하여 테스트 케이스를 생성한다.

3.1 테스트 케이스의 생성 순서

step 1. 인스턴스화된 시나리오를 생성
step 2. 클래스의 멤버와 매개변수의 집합 추출
step 3. 인스턴스화된 시나리오를 객체 모델에 매핑
step 4. 시나리오 다이어그램에서 메시지/메소드 경로를 추출
step 5. 사용자 동작의 종속성 분석을 통하여 1차 테스트 케이스 생성
step 6. 멤버 함수의 종속성 분석
step 7. 멤버 함수의 종속성 분석 결과 추가

3.2 고객 관리 시스템의 예

예로 사용할 시스템은 고객 관리 프로그램이다. 사용자는 고객 개인의 신상을 기록하고 이름, 자택 전화, 회사명, 직장 전화로 검색할 수 있게 되어 있으며 레



(그림 4) 고객 관리 시스템
(Fig. 4) Customer Management System

고드를 리스트로 볼 수 있고 새로운 레코드를 수정할 수도 있다.

이 시스템의 클래스중 CExOdb2View클래스를 예로 들면 다음과 같은 멤버와 멤버 함수로 구성되어있다[4].

```
class CExOdb2View : public CRecordView
{
protected:
        CExOdb2View();
public:
        CString m_pNumber;
        CString m_pName;
        CString m_pHTel;
        CString m_pHPostNumber;
        CString m_pHAddress;
        CString m_pOffice;
        CString m_pOTel;
        CString m_pOPostNumber;
        CString m_pOAddress;
        CString m_pBeepNum;
        CString m_pHandNum;
        CString m_pDescription;
        BOOL m_bAddMode;
        BOOL m_bMode;
public:
        CExOdb2Doc* GetDocument();
        void UpdateResource();
        void UpdateDb();
        virtual CRecordset* OnGetRecordset();
        virtual BOOL OnMove(UINT nIDMoveCommand);
protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        virtual void OnInitialUpdate();
        virtual ~CExOdb2View();
        void OnAdd();
        void OnMode();
        void OnRecordFirst();
        void OnRecordLast();
                :
        void OnMove(UINT nIDMoveCommand);
        void OnChangeHtel();
        void OnChangeName();
```

(그림 5) CExOdb2View 클래스[4]
(Fig. 5) CEXOdb2View Class definition[4]

다음은 3.1의 순서에 따라 테스트 케이스를 구한 예이다.

① 일반 시나리오 (주소 레코드 추가)
사용자 : 프로그램을 실행한다.
시스템 : 기존의 레코드가 있으면 첫 번째 레코드를 화면에 보여주고 없으면 빈 화면을 보여준다.
사용자 : 화면에 보이는 고객 관리 프로그램의 추가 단추를 선택한다.
시스템 : 레코드의 입력란을 비운다.
사용자 : 각 레코드의 항목에 입력한다.
사용자 : 레코드 이동 단추를 누른다.

② 인스턴스화한 시나리오 (주소 레코드 추가)
사용자 : 시스템을 실행시킨다.
시스템 : 1번 레코드를 보여준다.
번호 : 1, 이름 : 김미영, 자택 전화번호 : 279 1234, 자택 우편번호 : 100-000,
자택 주소 : 서울시 중구 장충동 100번지, 직장명 : 가나 주식회사, 직장 전화번호 : 555 7890,
직장 우편번호 : 140-000, 직장 주소 : 서울시 강남구 역삼동 200번지, 삐삐번호 : 012 323-1234,
핸드폰 : 000-000-0000, 기록사항 : 취미→야구 경기 관람
사용자 : 화면에 보이는 고객 관리 프로그램의 추가 단추를 누른다.
시스템 : 레코드의 각 입력란을 비운다.
사용자 : 각 레코드의 항목을 입력한다.
번호 : 2, 이름 : 송 선주, 자택 전화번호 : 735 -0179, 자택 우편번호 : 135-000,
자택 주소 : 서울시 서대문구 홍제동 23번지, 직장명 : 가나 주식회사, 직장 전화번호 : 555 7890,
직장 우편번호 : 140-000, 직장 주소 : 서울시 강남구 역삼동 200번지, 삐삐번호 : 015-311-0179,
핸드폰 : 000-000-0000, 기록사항 : 취미→컴퓨터 게임
사용자 : First Record 단추를 누른다.
시스템 : 수정, 입력 완료 단추를 비활성화 시키고, 입력된 내용의 레코드를 데이터베이스에 추가한다.

③ 클래스의 멤버와 매개 변수
3.1의 클래스에 대한 정의를 참고하여 각 멤버 함수에 대하여 영향을 미치는 클래스의 멤버와 매개 변수를 구한다. 여기서는 레코드 추가 함수인 OnAdd 함수에 영향을 미치는 멤버와 매개변수를 예로 구하였다. 이 경우 OnAdd함수의 매개 변수가 없으므로 클래스의 멤버만 구하면 된다.

```
CString m_pNumber;
CString m_pName;
CString m_pHTel;
```

CString m_pHPostNumber;
CString m_pHAddress;
CString m_pOffice;
CString m_pOTel;
CString m_pOPostNumber;
CString m_pOAddress;
CString m_pBeepNum;
CString m_pHandNum;
CString m_pDescription;
BOOL m_bAddMode;
BOOL m_bMode;

OnMove 함수는 OnAdd함수에서 구한 결과에 매개변수 nIDMoveCommand를 추가한다.

④ 객체 모델에 매핑

이 단계는 ②에서 구한 내용에 사용자가 시나리오를 보고 그 멤버 함수를 실행한 후의 값을 매핑한다. 예를 들어 인스턴스화된 시나리오에서 "시스템: 1번 레코드를 보여준다" 와 같은 부분은 다음과 같이 매핑이 된다.

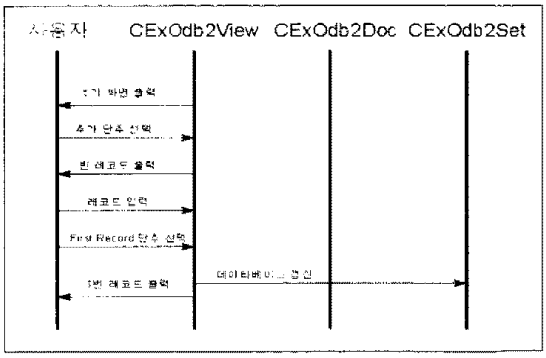
OnExOdb2View	OnInitialUpdate
m_pNumber=""	m_pNumber="1"
m_pName=""	m_pName="김미영"
m_pHTel=""	m_pHTel="279-1234"
m_pHPostNumber=""	m_pHPostNumber="100-000"
m_pHAddress=""	m_pHAddress="서울시 중구 상충동 2가 100번지"
m_pOffice=""	m_pOffice="가나 주식회사"
m_pOTel=""	m_pOTel="555-7800"
m_pOPostNumber=""	m_pOPostNumber="140-000"
m_pOAddress=""	m_pOAddress="서울시 강남구 역삼동 200번지"
m_pBeepNum=""	m_pBeepNum="012-323-1234"
m_pHandNum=""	m_pHandNum="011 756-1234"
m_pDescription=""	m_pDescription="유학 >아주 경기 관련"

⑤ Method/Message 경로 추출

(그림 6)은 사용자가 프로그램을 실행시킨 후 레코드를 추가하는 동작의 순서에 따라 고객 관리 시스템과 주고 받는 메시지를 나타낸 것이다.

레코드 추가 시나리오 다이어그램을 이용하여 ASF를 구한다.

【정의1】 MM-Path(Method/Message Path)
MM-Path는 message에 의해 연결된 연속된 method의 실행이다.



(그림 6) 고객 레코드 추가 시나리오 다이어그램
(Fig. 6) Scenario diagram for 'customer add'

【정의2】 ASF(Atomic System Function)
ASF는 input event, MM-Path의 집합, output event로 구성된다.

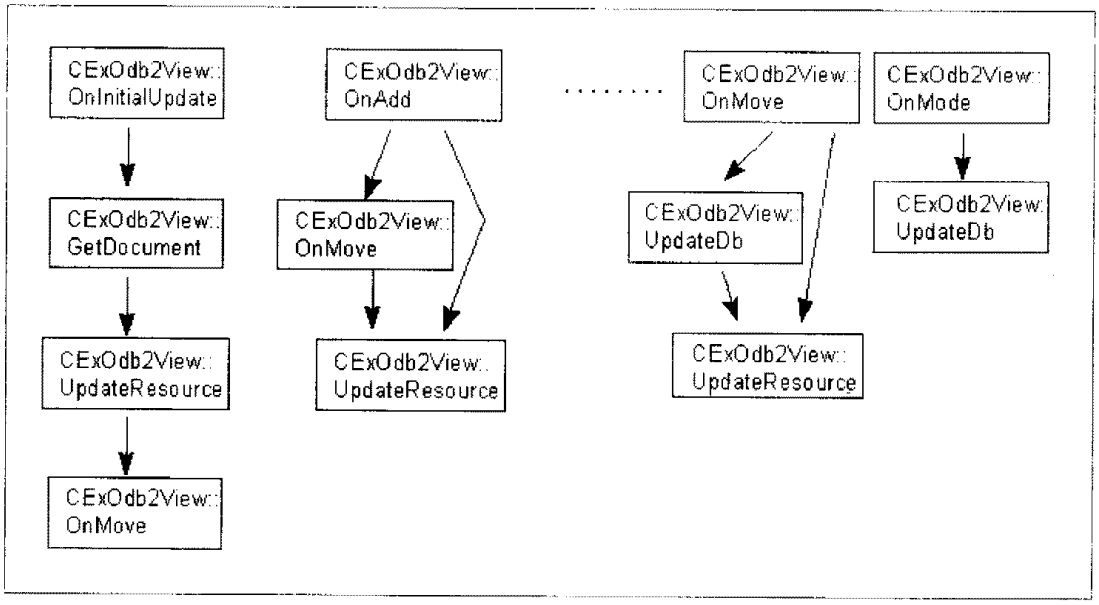
ASF 1) CExOdb2View::CExOdb2View() →
CExOdb2View::OnInitialUpdate()
ASF 2) CExOdb2View::OnAdd()
ASF 3) CExOdb2View::OnRecordFirst

위에서 구한 ASF를 조합하면 {(1,2,3), (1,3,2), (2,1,3,), (2,3,1), (3,1,2), (3,2,1)}의 6개의 테스트 케이스가 생긴다. 이때 시나리오를 이용하여 사용자 행동의 종속성 분석을 하여 테스트 케이스를 줄일 수 있다. 예를 들어 레코드 추가의 경우 추가 단추를 누른 후에 수정, 입력 단추를 누른다는 사용자의 동작에 대한 제한이 있고 생성자는 다른 모든 멤버 함수에 우선하여 실행하므로 1차 테스트 케이스는 다음과 같다.

CExOdb2View::CExOdb2View()
→CExOdb2View::OnInitialUpdate()
CExOdb2View::OnAdd()
CExOdb2View::OnRecordFirst

⑥ 멤버 함수의 종속성 분석 및 분석 결과 추가

위에서 구한 테스트 케이스는 시나리오만을 이용하여 추출한 것으로 ASF를 구성하는 모든 MM Path를 포함하고 있지 않다. 부족한 부분은 (그림 7)과 같이 원시 코드에서 멤버 함수간의 종속성 분석을 통하여 그래프를 생성한 후 ASF와 매칭되는 경로를 찾아서 완전한 MM Path로 구성된 테스트 케이스를 생성한다.



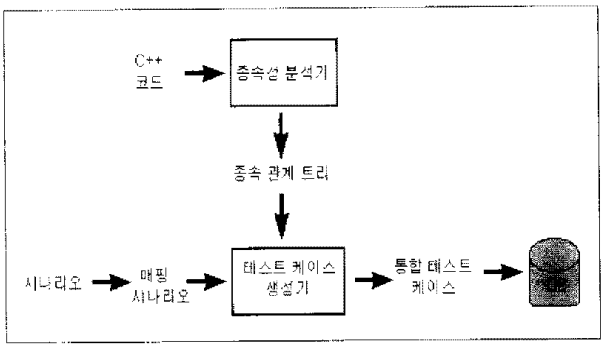
(그림 7) 멤버 함수의 호출 관계
(Fig. 7) Relationship of calling member functions

다음은 테스트 케이스로 생성된 결과이다.

```
CExOdb2View::CExOdb2View()
        m_pNumber=""                        m_pName=""
        m_pHTel=""                          m_pHPostNumber=""
        m_pHAddress=""                      m_pOffice=""
        m_pOTel=""                          m_pOPostNumber=""
        m_pOAddress=""                      m_pBeepNum=""
        m_pHandNum=""                       m_pDescription=""
        m_bAddMode=FALSE                    m_bMode=FALSE
CExOdb2View::OnInitialUpdate() -> CExOdb2View::GetDocument() -> CExOdb2View::UpdateResource() ->
CExOdb2View::OnMove(ID_RECIRD_FIRST) -> UpdateResource
        m_pNumber=1                         m_pName=김미영
        m_pHTel= 279-1234                   m_pHPostNumber=100-000
        m_pHAddress=서울시 중구 정동동 100번지   m_pOffice=가나 주식회사
        m_pOTel=555-7890                    m_pOPostNumber=140-000
        m_pOAddress=서울시 강남구 역삼동 200번지  m_pBeepNum=012-323-1234
        m_pHandNum=011-756-1234             m_pDescription=취미 -> 야구 정기 관람
        m_bAddMode=FALSE                    m_bMode=FALSE
CExOdb2View::OnAdd() -> CExOdb2View::UpdateResource()
        m_pNumber=""                        m_pName=""
        m_pHTel=""                          m_pHPostNumber=""
        m_pHAddress=""                      m_pOffice=""
        m_pOTel=""                          m_pOPostNumber=""
        m_pOAddress=""                      m_pBeepNum=""
        m_pHandNum=""                       m_pDescription=""
        m_bAddMode=TRUE                     m_bMode=FALSE
CExOdb2View::OnRecordFirst() -> CExOdb2View::OnMove() ->
CExOdb2View::OnUpdateDb() -> CExOdb2View::UpdateResource()
```

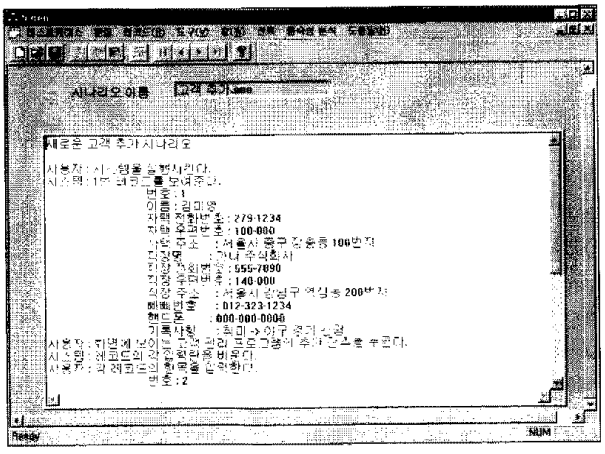
4. 객체 지향 시스템의 통합 테스트를 위한 시스템 설계

이 시스템의 메뉴는 파일, 선택, 종속성 분석, 테스트 케이스, 도움말로 구성되어 있다. 파일 메뉴는 생성된 테스트 케이스를 불러오거나 저장하고 테스트할 프로그램이나 시나리오를 선택에서 선택한다. 종속성 분석은 프로그램의 원시 코드로부터 호출 관계를 분석하여 멤버 함수의 종속성을 분석한다. 테스트 케이스 메뉴는 사용자가 작성한 매핑 시나리오를 1차 테스트 케이스로 입력받아서 종속성 분석을 한 결과와 합성하여 2차 테스트 케이스를 만든다. 생성된 테스트 케이스는 데이터 베이스에 저장하였다가 테스트 결과를 판단할 때 사용한다.



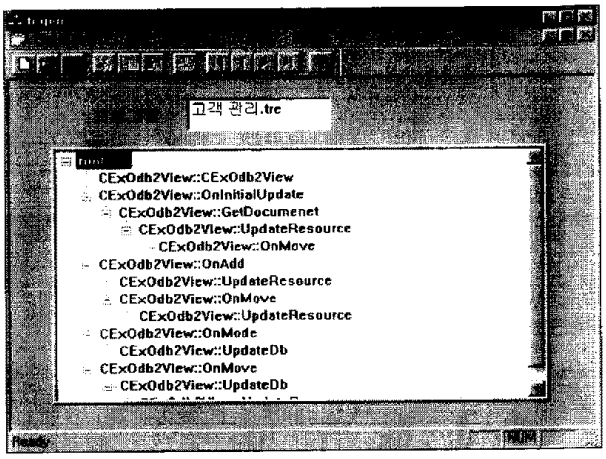
(그림 8) 시스템의 구성
(Fig. 8) System architecture

선택 메뉴를 선택하면 테스트 케이스를 생성하고자 하는 인스턴스화된 시나리오를 (그림 9)와 같이 보여준다.

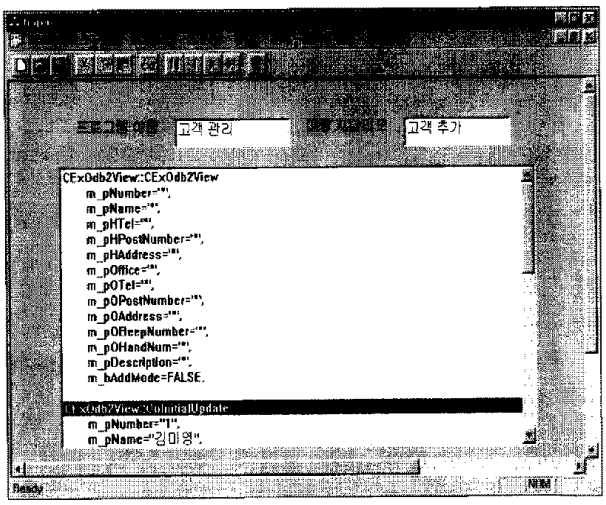


(그림 9) 인스턴스화 된 시나리오의 출력
(Fig. 9) Output of scenario including instance

종속성 분석 메뉴를 선택하면 (그림 9)의 시나리오와 관련 있는 모든 클래스의 멤버 함수들의 종속성 분석을 한 결과를 (그림 10)과 같이 계층관계로 보여주어 사용자의 이해를 돕는다.

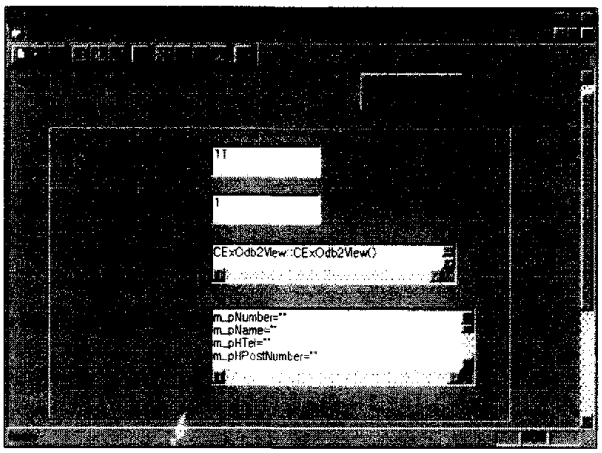


(그림 10) 멤버 함수의 종속성을 분석한 계층구조
(Fig. 10) Hierarchy of member functions dependency



(그림 11) 매핑 시나리오 입력 화면
(Fig. 11) Input window for displaying mapping scenario

(그림 11)은 객체 모델에 시나리오에서 추출한 테스트 데이터를 매핑한 결과를 입출력하는 화면이다. 이 화면을 이용하여 입력된 내용은 데이터 베이스에 레코드 형태로 저장되었다가 다음 단계에서 사용된다. (그림 11)의 매핑 시나리오와 (그림 10)의 멤버 함수의 종속 관계를 합성하여 (그림 12)에 나타난 것과 같은 테스트 케이스를 생성하여 보여준다. (그림 12)의 내용은 데이터 베이스에 저장되었다가 실제 프로그램의 실행 결과와 테스트 케이스를 비교할 때 다시 사용된다.



(그림 12) 테스트 케이스 출력 화면
(Fig. 12) Output window for test case

5. 테스트 케이스의 검증 및 테스트의 실행

테스트의 실행은 (그림 13)과 같은 클래스를 정의하여 실행한다. 테스트를 실행할 프로그램에 정의된 각각의 멤버 함수의 시작 부분에 (그림 14)와 같은 코드를 삽입한다. 멤버 함수의 끝 부분에는 멤버 함수의 이름을 출력하는 부분을 제외하고 삽입한다. 삽입후 프로그램을 실행시키면 그 멤버 함수가 실행될 때마다 CTest 클래스에 정의된 파일에 멤버 함수의 실행전 상태와 지금 실행하는 멤버 함수의 이름, 그리고 그 멤버 함수의 실행 결과를 출력하게 된다.

```
// Test.h
class CTest : public CWnd
{
public:
        CTest();
        void testWrite(CString testResult);
        virtual ~CTest();
};

// Test.cpp
#include "stdio.h"
#include "stdafx.h"
#include "ExOdb2.h"
#include "Test.h"

CTest::CTest()
{
}

CTest::~CTest()
{
}

void CTest::testWrite(CString testResult)
{
        CStdioFile fp;
        fp.Open("testlist.lst", CFile::modeReadWrite | CFile::typeText);
        fp.SeekToEnd();
        fp.WriteString(testResult);
        fp.WriteString("\n");
}
```

(그림 13) CTest 클래스
(Fig. 13) Ctest class

```
void CExOdb2View::TestResultWrite()
{
        CTest pTest;

        pTest.testWrite("        m_pNumber=" + m_pNumber);
        pTest.testWrite("        m_pName=" + m_pName);
        pTest.testWrite("        m_pHTel=" + m_pHTel);
        pTest.testWrite("        m_pHPostNumber=" + m_pHPostNumber);
        pTest.testWrite("        m_pHAddress=" + m_pHAddress);
        pTest.testWrite("        m_pOffice=" + m_pOffice);
        pTest.testWrite("        m_pOTel=" + m_pOTel);
        pTest.testWrite("        m_pOPostNumber=" + m_pOPostNumber);
        pTest.testWrite("        m_pOAddress=" + m_pOAddress);
        pTest.testWrite("        m_pBeepNum=" + m_pBeepNum);
        pTest.testWrite("        m_pHandNum=" + m_pHandNum);
        pTest.testWrite("        m_pDescription=" + m_pDescription);
        if(m_bAddMode)
                pTest.testWrite("        m_bAddMode=TRUE");
        else
                pTest.testWrite("        m_bAddMode=FALSE");
        if(m_bMode)
                pTest.testWrite("        m_bMode=FALSE");
        else
                pTest.testWrite("        m_bMode=FALSE");
```

(그림 14) 멤버 함수에 삽입될 내용
(Fig. 14) Instrumented code for member function

프로그램을 실행시킨 결과를 테스트 케이스와 비교하여 멤버 함수간의 인터페이스와 멤버 함수의 실행 결과를 테스트한다. 부록에 있는 내용은 고객 관리 프로그램을 실행시켜 레코드를 추가하는 기능에 대한 결과를 나타낸 것이다. 이 결과가 테스트 케이스와 멤버 함수의 실행 순서가 같고 실행 결과가 같은지를 비교한다.

6. 결 론

본 논문에서는 테스팅 단계에서 필요한 시나리오를 별도로 생성하지 않고 동적 모델링 단계의 시나리오를 이용하여 객체 지향 시스템의 통합 테스팅에 사용하는 방법을 제안하고 그 방법을 지원하기 위한 도구를 설계 구현하였다. 본 논문에서 제안한 방법은 동적 모델링 과정의 시나리오를 이미 만들어진 객체 모델에 매핑하여 일차적인 테스트 케이스를 추출하고 부족한 부분을 원시 코드에 나타난 멤버 함수간의 호출 관계를 이용하여 서로 다른 클래스의 멤버 함수간의 상호 작용을 테스트하기 위한 통합 테스트 케이스를 생성하는 방법을 보였다.

이 방법은 시나리오를 이용하여 사용자 행동의 종속성 분석을 통하여 1차 테스트 케이스를 생성한 후 2차 테스트 케이스를 생성한다. 그러므로 시나리오를

이용하지 않고 원시 코드에서 추출한 멤버 함수의 경로를 조합하였을 때와 비교하면 테스트 케이스의 수를 줄일 수 있을 뿐만 아니라 시나리오에서 제공하는 구체적인 테스트 데이터를 이용하여 멤버 함수의 경로뿐만 아니라 그 멤버 함수의 실행 결과까지 포함한 테스트 케이스를 생성할 수 있다.

앞으로 연구할 과제는 매핑 시나리오를 작성하는 사용자의 노력을 덜기 위한 지원 방안을 개발하는 것이다. 매핑 시나리오를 구체적이고 정확하게 작성하는 일은 테스트 케이스를 생성하기 위한 처음 단계로서 그 다음 단계에 영향을 미치므로 매우 중요하다고 할 수 있다. 그러므로 사용자가 도구를 이용하여 매핑 시나리오를 쉽고 정확하게 작성할 수 있도록 도와주는 도구의 개발이 필요하다.

참 고 문 헌

[1] 최은만, "컴퓨터를 이용한 시나리오 응용 방안", *1996년 한국 정보처리 학회 춘계 학술 발표 논문집 제3권 제1호*, pp.335-338

[2] 김희득, "객체 지향 소프트웨어의 스레드 테스팅을 위한 구성 개체의 종속 관계 정보 모델", *1996년 정보 과학회 가을 학술 발표 논문집(B)*, pp.1519-1526

[3] 홍형석, "유한 상태 기계 형태의 명세를 사용하는 클래스의 기능적 테스팅", *1994년 정보 과학회 가을 학술 발표 논문집(B)*, pp.802-805

[4] 이상엽, Visual C++ *Programming Bible Ver 4.x*, 영진 출판사, 1997, pp624-664

[5] Becky Winant and Mike Frankel, "The event dictionary: What your methodologist forget to tell you", *JOOP*, 1996년 10월, pp.33-37

[6] P.Hsia, "Formal Approach to Scenario Analysis", *IEEE Software*, 1994년 3월, pp.33-41

[7] John D.McGregor and Timothy D.Korson, "Integrated Object-Oriented Testing and Development Process", *CACM*, 1994년 9월, Vol.37 No.9, pp.59-77

[8] John D. McGregor, "Planning for testing", *JOOP*, 1997년 2월, pp.8-12

[9] Kai Koskimies, "Scene : Using Scenario Diagrams and Active Text for Illustrating O-bject-oriented Programs", *18th International Software Engineering Conference*, 1996, pp.366-377

[10] Paul C. Jorgensen et al, "Object-Oriented Integration Testing" *CACM*, 1994년 9월, Vol.37 No.9, pp.30-38

[11] C. Potts, "Inquiry-Based Requirements Analysis", *IEEE Software*, 1994년 3월, pp.21-32

[12] Robert M.Poston, "Automated Testing from Object Models", *CACM*, 1994년 9월 Vol.37 No.9 pp.48-57

[13] Shekhar Kirani and W.T. Tsai, "Method sequence specification and verification of classes", *JOOP*, 1994년 10월, pp.28-37

[14] Myers, G., *The Art of Software Testing*, Wiley & Sons, 1979

[15] I. Jacobson, *Object-Oriented Software Engineering-AUse-Case Driven Approach*, 1992, Addison-Wesley

부록 : ExOdb2 프로그램의 실행 결과

(멤버 함수의 실행 결과가 그 이전에 실행된 멤버 함수의 결과와 동일할 경우는 생략하였음)

```
CExOdb2Doc::CExOdb2Doc()
CExOdb2View::CExOdb2View()
        m_pNumber=""                        m_pName=""
        m_pHTel=""                          m_pHPostNumber=""
        m_pHAddress=""                      m_pOffice=""
        m_pOTel=""                          m_pOPostNumber=""
        m_pOAddress=""                      m_pBeepNum=""
        m_pHandNum=""                       m_pDescription=""
        m_bAddMode=FALSE                    m_bMode=FALSE
```

```
CExOdb2Doc::OnNewDocument()
CExOdb2View::OnInitialUpdate()
CExOdb2View::GetDocument()
CExOdb2View::UpdateResource()
CExOdb2View::OnMove()
CExOdb2View::UpdateResource()
        m_pNumber=1                          m_pName=김미영
        m_pHTel= 279-1357                    m_pHPostNumber=100-392
        m_pHAddress=서울시 중구 장충동 100번지   m_pOffice=가나 주식회사
        m_pOTel=555-1234                     m_pOPostNumber=135-000
        m_pOAddress=서울시 강남구 역삼동 200번지  m_pBeepNum=
        m_pHandNum=                          m_pDescription=취미 : 야구
        m_bAddMode=FALSE                     m_bMode=FALSE
CExOdb2View::OnAdd()
        m_pNumber=""                         m_pName=""
        m_pHTel=""                           m_pHPostNumber=""
        m_pHAddress=""                       m_pOffice=""
        m_pOTel=""                           m_pOPostNumber=""
        m_pOAddress=""                       m_pBeepNum=""
        m_pHandNum=""                        m_pDescription=""
        m_bAddMode=TRUE                      m_bMode=FALSE
CExOdb2View::UpdateResource()
        m_pNumber=""                         m_pName=""
        m_pHTel=""                           m_pHPostNumber=""
        m_pHAddress=""                       m_pOffice=""
        m_pOTel=""                           m_pOPostNumber=""
        m_pOAddress=""                       m_pBeepNum=""
        m_pHandNum=""                        m_pDescription=""
        m_bAddMode=TRUE                      m_bMode=TRUE
CExOdb2View::OnMode
CExOdb2View::OnRecordFirst()
CExOdb2View::OnMove()
        m_pNumber=""                         m_pName=""
        m_pHTel=""                           m_pHPostNumber=""
        m_pHAddress=""                       m_pOffice=""
        m_pOTel=""                           m_pOPostNumber=""
        m_pOAddress=""                       m_pBeepNum=""
        m_pHandNum=""                        m_pDescription=""
        m_bAddMode=TRUE                      m_bMode=FALSE
CExOdb2View::UpdateDb()
        m_pNumber=10                         m_pName=aa
        m_pHTel=                             m_pHPostNumber=
        m_pHAddress=                         m_pOffice=
        m_pOTel=                             m_pOPostNumber=
        m_pOAddress=                         m_pBeepNum=
        m_pHandNum=                          m_pDescription=
        m_bAddMode=TRUE                      m_bMode=FALSE
CExOdb2View::UpdateResource()
        m_pNumber=1                          m_pName=김미영
        m_pHTel= 279-1357                    m_pHPostNumber=100-392
        m_pHAddress=서울시 중구 장충동 100번지   m_pOffice=가나다 주식회사
        m_pOTel=555-1234                     m_pOPostNumber=135-000
        m_pOAddress=서울시 강남구 역삼동 200번지  m_pBeepNum=
        m_pHandNum=                          m_pDescription=취미 -> 야구 정기 관람
        m_bAddMode=FALSE                     m_bMode=FALSE
CExOdb2View::~CExOdb2View()
```

김 은 주

1991년 동국대학교 전자계산학과 (학사)
1997년 동국대학교 컴퓨터공학과 대학원(석사)
1998년 현 세명대학교 출강
관심분야 : 객체지향 테스팅, 객체지향 프로그래밍

최 은 만

1982년 동국대학교 전산학과(학사)
1985년 한국과학기술원 전산학과 (석사)
1993년 미국 일라노이 공대 전산학과(박사)
1985년~1988년 한국표준연구소 연구원
1988년~1989년 데이콤 주임연구원
1993년~현재 동국대학교 정보산업학부 조교수
관심분야 : 객체지향 소프트웨어 공학, 소프트웨어 유지보수, Y_2K 문제, 소프트웨어 재사용, 리엔지니어링