정형 명세를 이용한 웹 기반 은행 어플리케이션의 테스트 기법

안 영 희[†] · 최 은 만[††]

요 약

정형적 명세를 이용하면 원시코드의 복잡함에 방해받지 않고 필요한 구현 정보를 테스트 프로그래머가 얻을 수 있다. 특히 웹 기반 소프트웨어는 정형적 명세로 시스템에 대한 외부 입력과 반응을 잘 나타낼 수가 있다. 이 논문에서는 정형적 명세를 이용하여 테스트 데이터를 추출하는 방법을 제안한다. 복잡하고 구성요소가 다양한 웹 어플리케이션의 기능을 Object-Z 정형 명세언어를 이용하여 핵심적으로 나타낸다. 정형 명세에서부터 상태모델을 구성하고 최상위 레벨의 STD에서 세부적으로 STD를 추가하여 테스트 시나리오를 추출하였다. 실험 대상은 보안과 정확성을 요하는 웹 뱅킹 시스템으로 정하고 계좌이체 과정의 테스트 데이터를 추출하였다. 제안한 방법은 사용기반 테스트 기법과 결합하여 웹 소프트웨어의 테스트 자동화에 중요한 요소가 될 것이다.

A Testing Method for Web-Based Banking Applications Using Formal Specification

Younghee Ahn[†] · Eun Man Choi[††]

ABSTRACT

Programmers can be got the test-related information for implementation without interference of source code complexity by use of the formal specification. Especially the external inputs and system responses can be represented precisely by formal specification in testing phase of web-based software systems. This paper suggests a method of extracting test cases by use of formal specification. Object-Z formal specification represents various test-related information for complex functions of web-based applications. State Transition Models could be built from the formal specification so that test scenarios were extracted from STDs from the highest level to detail levels. The target system for verification of this method is a web-based banking system which is necessary to be secured and critical on errors. This method would be an important factor in automatizing test procedure for web-based application software systems combining the user-base test technique.

키워드 : 웹 기반 어플리케이션(Web-based Application), 소프트웨어 테스트(Software Test), 정형적 명세(Formal Specification), Object-Z, 테스트 자동화(Test Automation)

1. 서 론

여러 가지 다양한 자료의 접근이 쉬운 인터넷 환경에서 모든 소프트웨어를 사용한다는 것은 하나의 환경과 인터페이스로 통합한다는 의미에서 매력적이다. 또한 클라이언트나 서버에 극단적으로 부담을 주지 않고 적절히 컴퓨팅을 분산시키며 어느 곳에서나 접근하여 사용할 수 있다는 면에서 웹 기반 소프트웨어는 장점이 많다. 이러한 이유로 인터넷뿐만 아니라 인트라넷 환경에 웹 기반 소프트웨어들이 많이 개발되고 있다.

이러한 웹 기반 소프트웨어는 다음과 같은 몇 가지 이유로 화이트 박스 형태의 완벽한 테스트가 현실적으로 어렵다. 첫째로 웹 기반 소프트웨어는 구성 요소가 다양하다. 간단한 HTML로부터 Java Applet, Javascript, CGI, ASP에 이르기까지 다양한 컴포넌트들이 분산 존재하여 그의 추적 및 독립적인 단위 테스트와 분산된 컴포넌트의 통합 시험이 복잡하게 된다[1]. 둘째는 마크업 언어와 스크립트 언어는 절차적 프로그램과는 달리 실행 경로를 파악하기가 쉽지 않다는 문제가 있다. 즉 스크립트 타입의 언어에서 함수의 정의가 태그 안에서 사용될 때 어떤 순서로 사용될 것인지 파악하기가 어렵다.

정형적 명세를 이용하면 원시코드의 복잡함에 방해받지 않고 필요한 구현 정보를 테스트 프로그래머가 얻을 수 있다. 웹 기반 소프트웨어를 이루는 요소들의 통합을 위한 시험이 끝난 후 기능 시험을 위한 시스템 테스트 단계에는

† 준 회 원 : 동국대학교 대학원 컴퓨터공학과
†† 정 회 원 : 동국대학교 컴퓨터공학과 교수
논문접수 : 2003년 1월 27일, 심사완료 : 2004년 4월 20일

복잡한 원시코드를 볼 여유가 없다. 특히 웹 기반 소프트웨어는 복잡한 구성요소를 자세히 검토하는데 많은 노력과 시간이 소요되므로 시스템에 대한 외부 입력과 반응을 간결하게 나타내는 방법이 필요하다. 이것이 바로 정형적 명세이다.

웹 기반 소프트웨어를 상태 천이도로 나타내서 이를 바탕으로 테스트하려는 연구[14, 17]는 다음과 같은 문제를 안고 있다.

- 웹 응용의 규모가 커지면 상태천이도로 나타낼 때 상태가 매우 많아지고 이에 따라 테스트 경로를 찾기가 쉽지 않다.
- 상태천이도(State Transition Diagram)에만 의존하여 테스트 시나리오를 찾아내는 경우 각 기능이 모두 테스트되었는지 확인하기가 어렵다. 또한 상태 천이를 유도하는 입력이나 외부 자극을 찾아내기는 쉽지만 기능 수행 후 예상되는 결과는 상태천이도 만으로 쉽게 찾아낼 수가 없다.

이 연구에서는 정형적 명세를 웹 어플리케이션의 테스트 과정에 도입하여 이와 같은 문제점을 해결하였다. 정형적 명세를 이용하여 테스트하려는 연구는 일반적인 모형 기반 테스트 프레임 워크를 제안한 연구[24]와 실시간 제약 사항과 Z 명세 언어를 결합하여 시스템을 테스트하는 과정을 제안한 연구[8], 객체지향 프로그램의 테스트에 명세를 도입한 연구[21]가 있다. 하지만 웹 기반 소프트웨어에 적용하려는 노력은 없었다.

Obejct-Z 정형적 언어를 이용하여 웹 어플리케이션의 명세를 표현하고 여기서 상태 다이어그램을 추출한 후 테스트 시나리오를 작성하여 테스트하는 과정을 고안하였다. 실험 사례로 인터넷 뱅킹을 시험하였고 그 결과를 나타내었다.

2. 관련 연구

프로그램의 테스트 방법은 동적분석기법과 정적분석기법으로 분류할 수 있다. 동적분석은 테스트 케이스로 프로그램을 실행하여 출력 값을 분석하는 반면, 정적분석은 프로그램 그래프 분석이나 정형적 기법을 이용한다. 동적 테스트 방법에는 소프트웨어의 기능을 위주로 검사하는 테스트와 소프트웨어의 내부 구조를 기반으로 하는 테스트가 있다.

- 기능 위주의 테스트 - 출력 값이 맞는지 검사하기 위하여 입력 값의 조합을 변화시켜가면서 하위 레벨로부터 상위 레벨로 프로그램의 모든 기능을 테스트하는 것이다. 출력의 정확성은 프로그램의 기능 명세로부터 추정되는 예상 출력 값과 실제 출력 값을 비교함으로써 결정된다. 명세는 각 변수의 도메인이나 가능한 값의 집합을 정의하는데 이용되고 이 입출력 도메인에 근거하여 테스트 케이스를 선택한다.
- 구조를 기반으로 한 테스트 - 구조 테스트는 테스트 케이스를 선택할 때 프로그램의 내부 제어 구조를 이용하는 것이며, 화이트박스 테스트 또는 메트릭 기반 테스트라고도 한다. 커버리지 메트릭은 테스트 케이스에 의해 실행되는 프로그램의 구조적 단위의 수와 관련된다. 커버리지 메트릭에 근거한 테스트 방법은 테스트 케이스로 실행되는 프로그램에서 명령문과 분기, 경로의 수를 검사하는 것이다.

소프트웨어 개발 생명주기의 관점에서, 대개 구조적 테스트는 단위 레벨이나 모듈 레벨에 적용되고 기능 테스트는 통합 테스트나 인수 테스트와 같은 상위 레벨에 적용되지만 두 가지 모두 어느 레벨에나 적용될 수 있다. 단 하나의 테스트 어프로치로는 모든 테스트 문제를 해결할 수 없으므로 기능 테스트 기법과 구조 테스트 기법은 상호 보완적으로 사용하는 것이 일반화되어 있다.

2.1 정형적 명세

일반적으로 정형적 명세란 시스템의 속성이나 기능을 추상화하여 표현한 것으로 추상화 수준이나 개발 단계의 시스템에 대한 관심 대상에 따라 달라진다. 예를 들어 시간이 흐르면서 변하는 시스템의 동작을 나타내기 위하여 temporal logic을 이용하는 히스토리 기반 명세[4]가 있다. Z, VDM, B 등과 같이 어떤 순간에 시스템이 가져야 하는 변하지 않는 속성(invariant)과 전제조건(pre-condition) 및 종료조건(post-condition)을 나타내는 방법은 상태기반 방법이다. 또한 유한 상태 기계(finite state machine)로 표현하여 입력 이벤트에 대한 상태의 변환을 표현하는 변환 중심 명세(transition-based specification)와 논리 함수로 표현하는 기능 명세 방법이 있다.

네 가지 표현 방법 중 테스트에 잘 적용될 수 있는 것은 상태기반 방법과 변환 중심 명세 방법이다. 변환 중심 명세 방법은 테스트 경로가 무한으로 많아지며 상태를 줄이는 것이 쉽지 않다. 이에 비하여 Object-Z와 같은 상태기반 명세 방법은 구현된 소프트웨어가 가져야 할 기능적인 정보를 잘 나타내므로 블랙 박스의 기능 테스트 오러클을 찾아내기가 쉽다.

Object-Z는 Z의 객체지향 확장이다. Z에서 확장된 주요한 부분은 상태 스키마와 상태에 영향을 주는 오퍼레이션 스키마를 캡슐화하는 클래스이다. 클래스는 (그림 1)과 같은 박스 형태로 나타낸다.

Object-Z 정형적 명세는 단위 레벨 테스트에서부터 시스템 레벨 테스트, 리그레션 테스트까지 여러 레벨의 테스트에 사용될 수 있다. 그러나 다른 정형적 명세 언어와 마찬가지

로 Object-Z는 구현정보가 부족하기 때문에 화이트박스 테스트에는 부적합하며 기능적인 정보를 잘 나타내므로 블랙박스 테스트에 이용될 수 있다. (그림 1)에 Object-Z 오퍼레이션 스키마를 나타내었다. 오퍼레이션의 이름은 *Withdraw*이다.

Withdraw

$\Delta(balance)$

$amount?, amount! : \mathbb{R}$

$((balance > amount?) \wedge$
$(balance' = balance - amount?) \wedge$
$(amount! = amount?))$
$\vee$
$((balance \leq amount?) \wedge$
$(balance' = balance) \wedge$
$(amount! = 0))$

(그림 1) Object-Z 오퍼레이션 스키마

기호는 스키마에 의해 변경되는 객체의 데이터 속성을 의미한다. *Withdraw*오퍼레이션은 입력 값으로 *amount*?를 받아서 *amount*!를 출력하는데 '?'는 *amount*?가 스키마의 입력 파라미터, '!'는 *amount*!가 출력 파라미터라는 것을 의미한다. 스키마는 전제조건과 종료조건으로 표현되며 (그림 1)에서 (*balance*>*amount*?)는 전제조건이고 (*balance* = *balance amount*?)와 (*amout*! = *amount*?)는 종료조건이다. 이와 같이 Object-Z 명세는 오퍼레이션의 전제조건과 종료조건 뿐만 아니라 입출력 정보도 표현 가능하다.

2.2 일반적인 웹 어플리케이션의 테스트 방법

트랜젝션 기반 시스템이나 상태 기반 시스템들은 주로 유한 상태로 시스템을 표현하고 이를 테스트에 응용한다. 웹 기반 시스템도 상태 기반 시스템으로 간주하고 유한 상태 기계를 이용한 테스트 방법을 적용할 수 있다. 이 방법은 시스템의 행위를 유한상태의 집합으로 나타내고 여기서 접근 가능한 최소의 유한상태기계(minimal finite state machine)를 찾아내어 각 상태를 구동시킬 수 있는 트랜지션들을 찾아내는 것이다.

그러나 이러한 방법으로 테스트 시나리오를 찾아내는 경우 각 기능이 모두 테스트되었는지 확인하기가 어렵고 기능 수행 후 예상되는 결과를 쉽게 찾아 낼 수가 없다. 또한 웹 응용의 규모가 커지면 상태가 매우 많아지므로 테스트 경로를 찾기가 쉽지 않다.

따라서 본 논문에서는 이러한 문제점을 해결하고 시스템에 대한 외부 입력과 반응을 간결하게 나타내기 위한 방법으로 정형적 명세를 웹 프로그램의 테스트 과정에 도입한다. Object-Z를 이용하면 기능별로 시험할 수 있고 예상되는 결과가 표현되어 쉽게 비교할 수 있으며 테스트 데이터 찾기가 용이하다.

2.3 정형적 명세를 이용한 웹 프로그램의 테스트 방법

이 연구의 주된 아이디어는 정형적 명세를 웹 어플리케이션의 테스트 과정에 도입하는 것이다. 객체지향설계에서 기본 빌딩 블록은 클래스와 객체이다. 각 클래스는 속성과 오퍼레이션을 갖는다. 클래스 사이의 관계는 상속과 포함이며 클래스 간의 데이터 전달은 메시지 교환에 의해서 이루어진다. 객체지향 프로그램에서 객체의 오퍼레이션과 속성은 객체의 상태를 반영한다. Booch에 의하면 상태천이도(State Transition Diagram)는 주어진 클래스의 상태 공간과 트랜지션을 일으키는 이벤트, 그리고 상태 변화로 인한 액션을 나타내는데 사용되며 시스템의 행위도 나타낼 수 있다. 이 연구에서 상태천이도는 상태 모델로 표현되며 시스템의 동적 행위를 나타낸다.

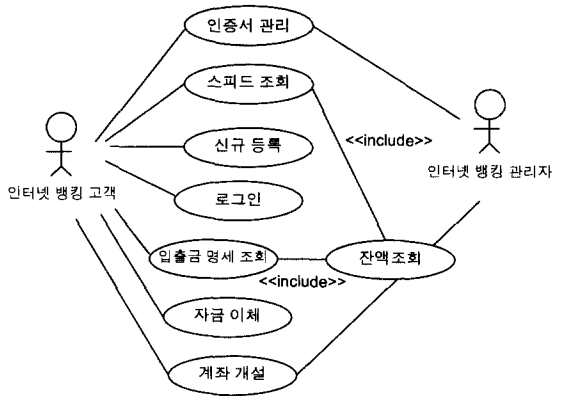
웹 프로그램의 테스트를 위한 프레임워크는 객체지향 정형적 명세, 오퍼레이션 사용 프로파일, 데이터 프로파일의 세 가지 요소를 포함한다. 정형적 명세를 이용하면 테스터가 소스 코드를 들여다보지 않고 구현 정보를 얻을 수 있으며 클래스 구조와 예상되는 행위의 정확한 표현이 가능하다. 대상 시스템의 오퍼레이션 시퀀스를 나타내는 상태천이도는 오퍼레이션 사용 프로파일로 이용될 수 있다. 오퍼레이션 사용 프로파일은 상태천이도나 유한상태기계라는 문법 형태로 나타낸다. 그래픽 표현은 시스템을 시각적으로 나타내는 방법을 제공하는 반면 문법 형태는 테스트 시나리오 추출의 자동화를 가능하게 한다. 이에 더하여 데이터 타입과 분포 즉, 사용 프로파일은 extended Backus-Naur form(EBNF)로 나타낼 수 있다.

테스트 시나리오는 오퍼레이션 사용 프로파일로부터 추출하고 테스트 시나리오를 작성한 후에는 데이터 프로파일에 따라 테스트 케이스를 작성한다. 그 단계는 다음과 같다.

① 웹 기반 소프트웨어의 중요 기능을 Object-Z로 명세화한다.
② 웹의 각 페이지와 매칭되는 상태천이도를 작성한다.
③ 트랜지션 경로(테스트 시나리오)를 선택한다.
④ 테스트 시나리오와 데이터 사용 프로파일을 이용하여 테스트 케이스를 작성한다.
⑤ 실행 경로를 검사하고 전제조건과 종료조건을 검증하여 테스트 결과를 모니터링한다.

3. 웹 프로그램의 명세화

정형적 명세는 프로그램의 품질을 향상시키는 데에 매우 효과적인 방법으로 알려져 있다. 그러나 실제로 웹 프로그램의 테스트에 정형적 명세가 이용된 예는 찾아보기 어렵다. 이 장에서는 정형적 명세가 웹 프로그램의 테스트에 어떻게 이용되는지 (그림 2)의 인터넷뱅킹 업무를 예로 들어 설명하겠다.



(그림 2) 테스트하려는 인터넷뱅킹의 사용사례

3.1 요구기능 명세

인터넷뱅킹은 사용자가 PC를 통한 인터넷 접속으로 자금 이체나 각종 조회를 할 수 있는 금융 시스템이다. 인터넷뱅킹 사용자는 서비스 신청 시 은행이 정한 출금 가능 과목 중 본인명의 계좌를 출금계좌로 지정해야 되며 이외의 계좌는 이체서비스를 이용할 수 없다. 사용자는 인터넷뱅킹을 이용하기 위해서 인증기관으로부터 인증서를 발급받아야 되며 발급자 이름과 일련번호로 인증서의 유일성을 식별한다.

인터넷뱅킹의 예는 소프트웨어 시스템의 기능성에 초점을 둘 것이다. 본 논문에서 예로 드는 인터넷 뱅킹의 요구 명세는 다음과 같다.

① 보안성
- 인터넷뱅킹 홈페이지를 클릭하면 암호화 프로그램이 자동으로 다운로드되어 설치된다.
- 인증서 암호 5회 오류 시 로그인을 재시도한다.
- 이체비밀번호, 안전카드번호 3회 연속 오류 시 이체서비스가 정지된다.
- 계좌비밀번호 3회 연속 오류 시 서비스가 정지된다.
- 사용자의 정보 보호를 위하여 일정시간 거래가 없으면 인터넷뱅킹 거래를 자동 종료한다.

② 거래 기능
- 계좌 잔액을 조회한다.
- 입출금이 자유로운 보통예금, 저축예금, 당좌예금의 입출금명세를 조회한다.
- 사용자의 지시에 의해 은행에 등록된 인터넷뱅킹 출금 계좌에서 지시한 입금 계좌로 자금을 이체한다.
- 보통예금, 저축예금, 정기예금, 정기적금 계좌를 개설한다.

3.2 인터넷뱅킹의 Object-Z명세

명세의 핵심은 인터넷뱅킹의 제어 소프트웨어이며, 각 하드웨어 컴포넌트는 충분히 테스트 되었다는 가정 하에서 출발한다. 인터넷뱅킹 시스템에서 가장 기본적으로 처리되는 객체는 인증서이다. 인증서는 사용자와 발급자, 인증번호, 유효기간 등의 정보를 포함하고 있으며 인증기관으로부터 발급받을 때 지정된 디렉토리에 파일 형태로 저장된다. 인터넷뱅킹에서의 물리적 객체는 Object-Z클래스로 표현된다. 예를 들면 *WebBrowser, INIplugin, Keyboard, Mouse, Account, Bank, CertificateAuthority, IBProcess* 등이 이에 해당된다.

IBProcess

$bank : Bank$
$ca : CertificateAuthority$
$iniplugin : SecurityProgram$
$revokedCertificate : CertificateNo \pfun \mathbb{B}$
$State ::= idle \mid inipluginrunning \mid certificatechecked \mid certpwdverified \mid receivedinfo \mid transferpwdverified \mid logout$

$passbookAcc, savingsAcc, timeAcc, installmentAcc : Account$
$badTry : \mathbb{N}$
$state : State$

INIT

$state = idle$
$badTry = badTry1 = badTry2 = 0$

INIpluginStart

$\Delta(state)$
$start\_url? : URL$
$state = idle$
$state' = inipluginrunning$

CertificateCheck

$\Delta(state)$
$login\_url? : URL$
$certificateFilename! : FileName$

$state = inipluginrunning$
$certificateFilename! = INIplugin.ReadCertificate.certificateFilename$
$(certificateFilename! \in validCertificateFilenames \land state' = certificatechecked)$

CertificatePasswordVerify

$\Delta(state, badTry)$
$inCertPwd?: CertPwdType$
$certificateNo! : CertificateNo$

$state = certificatechecked$
$((certificateNo! \in ca.CRL \land state' = inipluginrunning) \lor$
$((badTry = 5 \land INIplugin.CloseCertificate \land badTry' = 0 \land state' = inipluginrunning)$
$\lor$
$((badTry < 5) \land$
$((inCertPwd? = bank.GetCertPwd \land badTry' = 0 \land state' = certpwdverified) \lor$
$(inCertPwd? \neq bank.GetCertPwd \land badTry' = badTry + 1 \land state' = state))))$

GetAccounts

$\Delta(state, passbookAcc, savingsAcc, timeAcc, installmentAcc)$

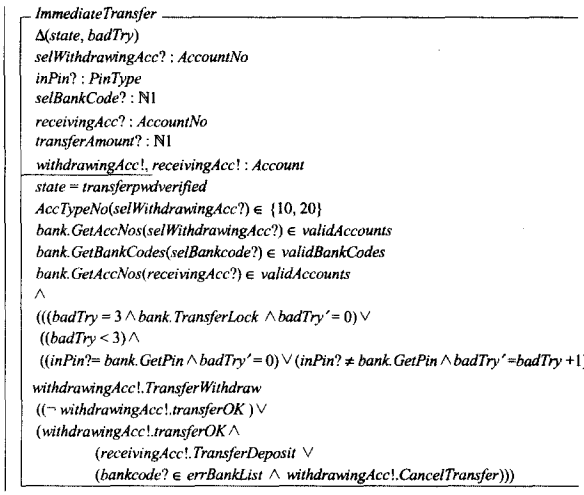
$state = certpwdverified$
$passbookAcc' = bank.GetPassbookAcc \parallel bank.GetAccInfo$
$savingsAcc' = bank.GetSavingsAcc \parallel bank.GetAccInfo$
$timeAcc' = bank.GetTimeAcc \parallel bank.GetAccInfo$
$installmentAcc' = bank.GetInstallmentAcc \parallel bank.GetAccInfo$
$state' = receivedinfo$

ProcessTrans

$\Delta(state)$
$sel? : URL$

$state = receivedinfo$
$sel? \in \{url1, url2, url3, url4, url5, url6\}$
$((sel? = url1 \land DepositWithdrawalInquiry \land state' = state) \lor$
$(sel? = url2 \land AccountBalanceInquiry \land state' = state) \lor$
$(sel? = url3 \land ImmediateTransfer \land state' = state) \lor$
$(sel? = url4 \land ReservationTransfer \land state' = state) \lor$
$(sel? = url5 \land OpenAccount \land state' = state) \lor$
$(sel? = url6 \land CloseAccount \land state' = state)$

(그림 3) 인터넷뱅킹의 정형적 표현



(그림 4) Object-Z 명세 : 즉시 이체

인터넷뱅킹 시스템을 정형화하기 위해 최상위 레벨인 *IB-Process* 클래스 명세로부터 시작하겠다. *IBProcess*의 속성은 bank, account, 효력 정지된 인증서를 나타내는 partial function을 포함한다. *IBProcess*는 주요한 일곱 가지 상태로 나타낼 수 있으며 상태간의 천이는 외부 이벤트와 내부 오퍼레이션에 의해서 일어난다.

인터넷뱅킹 홈페이지에 접속하기 전까지는 *idle* 상태이다. 인터넷뱅킹을 클릭하면 암호화 모듈 INIplugin이 설치되고 *IBProcess*는 *inipluginloaded* 상태가 된다. 로그인 전 초기 화면의 메뉴는 인증서 관리, 스피드조회, 신규등록이다. 초기 화면에서 로그인을 클릭하면 INIplugin이 인증서를 검색하여 *certificatechecked* 상태가 되고 인증서 암호가 은행 서버에 전달되어 확인되면 *certpwdverified* 상태가 되어 로그인 된다. 인증서 암호가 확인되면 내부 오퍼레이션에 의해 계좌정보를 받게 되고 로그인 후 웹 페이지에 예금종류, 계좌번호, 잔액 등 기본정보가 디스플레이 된다. 기본 정보 화면에서 계좌를 선택하여 클릭하면 레이어가 활성화되면서 숨겨져 있던 서브메뉴가 나타난다. 예금종류에 따라 서비스 가능한 메뉴가 다르게 나타나는데 자유입출금예금의 경우 주요 거래 기능은 입출금명세조회, 잔액조회, 즉시이체, 예약이체 등이다. 인터넷뱅킹의 Object-Z 명세를 (그림 3)에 나타내었다.

(그림 4)는 논문에서 예로 들 즉시이체 거래의 Object-Z 명세이다. *TransferPasswordVerify* 스키마는 계좌이체 및 예금신규 시 이체비밀번호를 확인하는 과정이다. 계좌 정보를 받은 *receivedinfo* 상태이며 입력 값은 이체비밀번호와 안전카드코드이다. *badTry*1과 *badTry*2는 각각 잘못 입력한 이체비밀번호와 안전카드코드를 의미한다. 이체비밀번호나 안전코드가 3번 연속 틀리면 해당 계좌의 이체 서비스가 중지되고 이체비밀번호와 안전카드코드가 확인되면 *transferpwdverified* 상태가 된다.

ImmediateTransfer 스키마는 이체비밀번호가 확인된 상태이며 출금계좌와 입금계좌를 선택한다. 출금계좌의 통장 비밀번호를 확인하는 오퍼레이션은 *bank* 스키마에도 포함되어 있다. 사용자에게는 맞는 PIN을 입력할 세 번의 기회가 주어지고, 만일 3번 연속으로 틀리면 그 계좌의 모든 서비스가 정지된다. PIN의 오류회수가 3번 미만인 경우 올바른 PIN이 입력되면 오류회수를 0으로 초기화하고 잘못된 PIN이면 오류회수는 1 증가된다. 이체금액이 유효하면 출금계좌의 *TransferWithdraw*가 호출되어 출금계좌에서 자금이 인출되고 계좌잔액이 변경되며 입금계좌로 이체된다. 그러나 입금은행이 서비스 장애 중이면 출금계좌의 *CancelTransfer* 오퍼레이션이 호출되어 즉시이체가 취소된다.

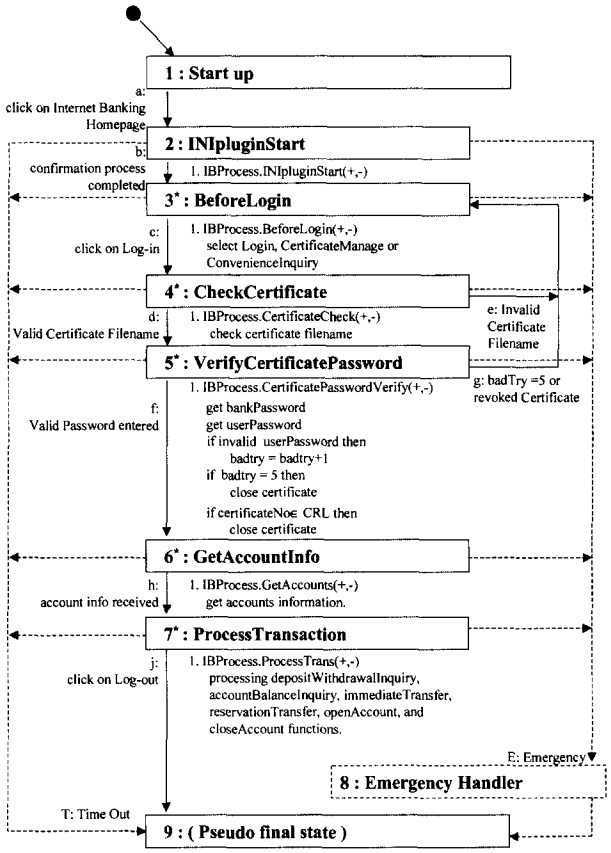
4. 명세 이용 테스트

4.1 시스템 상태천이도 작성

STD는 시스템의 상태 모델을 표현하는 데 사용되며 시스템의 정형적 명세와 사용 정보로부터 작성한다. 상태 모델에는 계층, 사용 정보, 파라미터 정보가 포함된다. 계층적 특징은 상태 다음에 '*'를 사용하여 상태가 하위 레벨로 더 상세화 됨을 나타낼 수 있다. 이것은 명세의 계층적 구성을 이용하여 정형적 명세와 상태 모델 간의 정확한 매핑이 가능하게 한다. 또한 이러한 계층적 특성으로 인해 통합 시스템의 상태 폭발 문제도 제어할 수 있다. 입출력 파라미터와 호출되는 함수, 전제조건과 종료조건 역시 상태천이도에 표현 가능하다. 입출력 파라미터명은 입력 데이터를 생성하기 위한 테스트 데이터 제너레이터나 출력 결과를 분석하기 위한 실행 모니터에 이용된다. 전제조건과 종료조건은 테스트 엔지니어가 Object-Z명세로부터 추출하여 해석하거나 오러클 툴을 이용할 수도 있다.

IBProcess 명세와 사용 정보를 기반으로 최상위 레벨 STD를 작성하면 (그림 5)와 같다. 상태를 변화시키는 트랜지션

은 대부분 *IBProcess*에 포함된 프로세스, *GetAccount*, *Process-Transaction*(웹 뱅킹의 경우 입출금명세조회, 잔액조회, 즉시이체, 예약이체, 예금신규 등)과 같은 것이다. 명세의 각 상태 스키마는 웹 페이지에 매칭된다. 상태를 천이시키는 트랜지션에 표시된 +는 프로세스가 수행에 들어갔다는 것을 의미하며 -는 퇴장을 의미한다. (그림 6)에 표현된 최상위 레벨에서 인터넷뱅킹의 중요한 핵심 기능은 상태 7이다.



(그림 5) STD : 인터넷뱅킹의 최상위 레벨

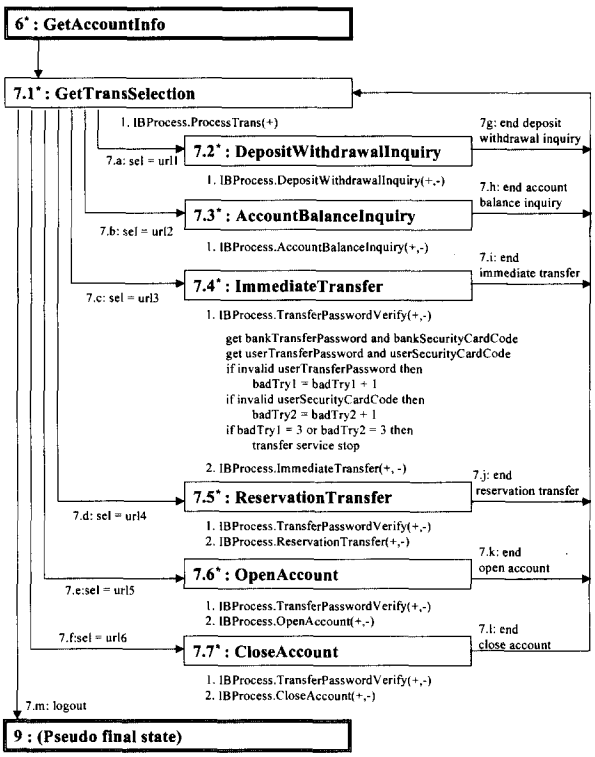
상태 7을 더 상세히 분석하면 (그림 6)과 같이 된다. 입출금명세조회, 즉시이체, 예금신규 등을 위한 트랜젝션 선택이 7.1이며 선택 후 적절한 상태로 들어간다. 즉시이체 과정을 예로 들면 7.4.1 이체비밀번호, 안전카드번호 입력, 7.4.2 출금계좌 및 비밀번호, 입금계좌, 이체금액 입력, 7.4.3 이체 내용 확인, 7.4.4 이체 완료 등으로 STD는 하향식으로 계속 상세화하여 추가할 수 있다.

4.2 테스트 시나리오 추출

테스트 시나리오 추출 과정은 STD를 검증하고 분석하여 단일 경로와 단일 사이클을 추출하는 과정을 포함한다. 시스템의 행위를 모형화한 STD는 방향 그래프로 볼 수 있으며 그래프의 각 정점은 STD에서의 상태를 나타낸다. 테스트 시나리오 추출 과정은 다음과 같다.

① 유효한 그래프인지 점검한다.
② 정점을 축소한 그래프(Reduced Directed Graph)를 작성한다.
③ 정점에 새 명칭을 부여한다.
④ 그래프의 이행적 폐쇄(transitive closure)를 작성한다.
⑤ 단일 실행 경로(기본 경로)를 찾는다.
⑥ 기본 경로들과 단일 사이클들로부터 가능한 시나리오를 모두 추출한다.

Object-Z명세로부터 STD를 작성한 다음 STD를 검증하고 분석한다. 예비 분석이 성공적으로 이루어지면 RDG를 작성한다. 인터넷뱅킹의 최상위 레벨 STD를 예비 분석하면 정점의 수는 감소되고 명칭도 바뀐다. 그 결과는 (그림 7)과 같다.



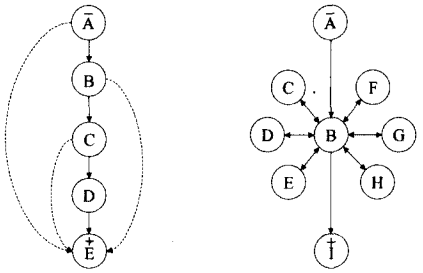
(그림 6) 상태 7에 대한 상세 STD

Reduction of vertices:
(v1 → v2) ⇔ A, v3 ⇔ B
(v4 → v5) ⇔ C,
(v6 → v7) ⇔ D, v9 ⇔ E
Initial node: A; Final node(s) : E The graph is a dag.

Reduction of vertices:
v_6 ⇔ A, $v_{7.1}$ ⇔ B
$v_{7.2}$ ⇔ C, $v_{7.3}$ ⇔ D
$v_{7.4}$ ⇔ E, $v_{7.5}$ ⇔ F
$v_{7.6}$ ⇔ G $v_{7.7}$ ⇔ H
v_9 ⇔ I
Initial node : A; Final node(s) : I. The graph contains cycle(s).

(그림 7) 정점 축소 과정

인터넷뱅킹의 최상위 레벨과 상태 7의 RDG는 각각 (그림 8)(a), (그림 8)(b)와 같다. 이 그래프에서 정점 이름 위의 -는 시작 정점을 나타내고 +는 최종 정점을 나타낸다. (그림 8)(b)에서 B는 트렌젝션 선택, C는 입출금명세조회, D는 계좌잔액조회, E는 즉시이체, F는 예약이체, G는 예금신규, H는 예금해약, I는 최종 상태를 나타낸다.



(a) (b)

(그림 8) 인터넷뱅킹의 RDG

<표 2>의 (*A, BEB, I*)는 사용자가 먼저 즉시이체 서비스를 선택하고 로그아웃한 경우의 시나리오이다. 경로는 정점 축소를 역으로 하면 복원할 수 있다. 이렇게 하여 얻은 하위 경로는 다음과 같다.

$V_6V_{7.1}V_{7.4}V_{7.1}V_9$

분해된 하위 경로와 함께 <표 1>의 경로 0($V_1V_2V_3V_4V_5V_6V_7V_9$)을 대체하면 경로는 아래와 같다.

$V_1V_2V_3V_4V_5V_6V_{7.1}V_{7.4}V_{7.1}V_9$

〈표 1〉 최상위 레벨 경로

Num	Length	Path	Original Path
0	4	A, B, C, D, E	$V_1V_2V_3V_4V_5V_6V_7V_9$
1	3	A, B, C, E	$V_1V_2V_3V_4V_5V_9$
2	2	A, B, E	$V_1V_2V_3V_9$
3	1	A, E	$V_1V_2V_9$

〈표 2〉 상태 7의 확장 경로

Path	Len	Basis	Path
0	2	-	*A, B, I.*
1	4	0	*A, BCB, I.*
2	4	0	*A, BDB, I.*
3	4	0	*A, BEB, I.*
4	4	0	*A, BFB, I.*
5	4	0	*A, BGB, I.*
6	4	0	*A, BHB, I.*
7	6	1	*A, BCBCB, I.*
8	6	1	*A, BCBDB, I.*
9	6	1	*A, BCBEB, I.*
10	6	1	*A, BCBFB, I.*
11	6	1	*A, BCBGB, I.*
12	6	1	*A, BCBHB, I.*
13	6	2	*A, BDBCB, I.*
14	6	2	*A, BDBDB, I.*
15	6	2	*A, BDBEB, I.*
16	6	2	*A, BDBFB, I.*

이와 같은 방법으로 추출한 즉시이체 과정의 테스트 시나리오를 <표 3>에 나타내었다.

〈표 3〉 테스트 시나리오의 예 (즉시이체)

Test Scenario : $V_1V_2V_3V_4V_5V_6V_{7.1}V_{7.4.1}V_{7.4.2}V_{7.4.3}V_{7.4.4}V_{7.1}V_9$			
Current State	Function list	Next State	Event
1	{ }	2	click on Internet banking home
2	IBProcess.INIpluginStart(+, -)	3	confirmation process completed
3	IBProcess.BeforeLogin(+, -)	4	click on Log-in
4	IBProcess.CheckCertificate(+, -)	5	valid certificate filename checked
5	IBProcess.CertificatePasswordVerify(+, -)	6	certificate password verified
6	IBProcess.GetAccount(+, -)	7.1	account information received
7.1	IBProcess.ProcessTrans(+)	7.4.1	transaction selection
7.4.1	IBProcess.TransferPasswordVerify(+, -)	7.4.2	transfer password verified
7.4.2	IBProcess.ImmediateTransfer(+, -)	7.4.3	immediate transfer transaction
7.4.3	IBProcess.TransactionConfirm(+, -)	7.4.4	click on confirmation button
7.4.4	IBProcess.EndTransfer(+, -)	7.1	transaction completed
7.1	IBProcess.ProcessTrans(-)	9	click on Log-out
9	{ }		

4.3 테스트 케이스 작성 및 실험

테스트 시나리오를 추출한 후에는 <표 4>의 데이터 프로파일에 명세되어 있는 각 변수의 타입과 범위를 이용하여 테스트 케이스를 작성한다. 구조 데이터 타입은 더 간단한 데이터 타입들에 근거하여 정의된다.

구조 데이터 타입이 작성되면, 종단에 이르기까지 분석을

계속한다. 종단은 정수형이나 실수형과 같은 프리미티브 타입이거나 제약조건이 있는 파생된 프리미티브 타입이다. 예를 들면, 인증서 암호는 *CertificatePassword*로 정의되는데 첫자리 영문자 1자를 포함한 6에서 8자리의 영문자와 숫자의 조합이다. 계좌번호 *AccountNo*는 지점번호 *BranchNo*와 과목번호 *AccTypeNo*, 그리고 일련번호 *SerialNo*와 검증번호 *CheckDigit*의 조합으로 이루어지며 각각 3 *digits*, 2 *digits*, 5 *digits*, 1 *digit*으로 구성된다. 이체 금액은 *TransferAmount*로 정의되며 1에서 10000000 사이의 범위를 만족하는 값이어야 한다. 이와 같이 EBNF에 정의된 자료사전은 데이터 분포뿐만 아니라 경계 값이나 예외 값 등 다른 테스트 데이터 선택의 기준이 된다.

〈표 4〉 EBNF 자료사전

digit	::=	<*char*><[0..9]> ;
letter	::=	<*char*><[*a..z, A..Z*]> ;
UserID	::=	[*digit*] *letter*(1..8)](6..8) ;
CertificatePassword	::=	*letter*(1) + [*digit* | *letter*](5..7) ;
TransferPassword	::=	[*digit* | *letter*(1..8)](6..8) ;
SecurityCardCode	::=	*digit*(4) ;
PIN	::=	*digit*(4) ;
AccountNo	::=	*BranchNo* + *AccTypeNo* + *SerialNo* + *CheckDigit* ;
BranchNo	::=	*digit*(3) ;
AccTypeNo	::=	*digit*(2) ;
SerialNo	::=	*digit*(5) ;
CheckDigit	::=	*digit*(1) ;
BankCode	::=	*digit*(2) ;
CustomerName	::=	*Name* ;
Name	::=	<*char*> ;
Remitter	::=	*Name* ;
RequestTypeInfo	::=	[*DepositWithdrawalInfo* | *AccountBalanceInfo* | *ImmediateTransferInfo* | *TransferReservationInfo* | *OpenAccountInfo*] ;
DepositWithdrawalInfo	::=	*AccountNo* + *InquiryPeriod* + *AccountBalance* + *TransactionDate* + *Withdrawal* + *Deposit* + *TransactionType* + *Remarks* + *Branch* + *TotalDeposit* + *TotalWithdrawal* ;
AccountBalanceInfo	::=	*AccountNo* + *AccountBalance* ;
ImmediateTransferInfo	::=	*WithdrawingAccountNo* + *ReceivingAccountNo* *TransferAmount* + *CustomerName* ;
TransferReservationInfo	::=	*WithdrawingAccountNo* + *ReceivingAccountNo* *TransferAmount* + *CustomerName* + *ReservationDate* + *ReservationTime* ;
OpenAccountInfo	::=	*AccountNo* + *AccountType* + *OpeningDeposit* + *Branch* + *WithdrawingAccountNo* ;
Remarks	::=	[*CD* | *ATM* | *PC* | *CC* | *tele*] ;
TransferAmount	::=	<*real*><[1..10000000]> ;
ServiceFee	::=	<*integer*><[0 | 300]> ;

〈표 5〉 즉시이체 테스트 케이스

입력 데이터	테스트 케이스	예상되는 결과	테스트 결과
IBProcess			
Start up			
	없음		
INIpluginStart			
	없음		
CheckCertificate			
	없음		
VerifyCertificatePassword			
CertificatePassword?	a. Abc123	정 상	정 상
	b. A2X?/	비정상	비정상
	c. A	비정상	비정상
	d. abcdefgh123	비정상	비정상
	e. 123Abc	비정상	비정상
GetAccountInfo			
VerifyTransferPassword			
TransferPassword?	a. 123abc	정 상	정 상
	b. A2X?/	비정상	비정상
	c. A	비정상	비정상
	d. abcdefgh123	비정상	비정상
	e. 123456	비정상	비정상
SecurityCardCode?	a. 7241	정 상	정 상
	b. abcd	비정상	비정상
	c. 12	비정상	비정상
	d. 123456	비정상	비정상

<표 4>의 자료사전을 참고하면 인증서 암호는 첫자리 영문자 1자를 포함한 6에서 8자리의 영문자와 숫자의 조합이다. 인증서 암호가 영숫자이어야 한다는 사실로부터 다음 두 가지 동치 클래스를 만들어 낼 수가 있다.

① 영숫자 암호(정상) : Abc123

② 영숫자가 아닌 암호(비정상) : A2X?/

인증서 암호는 6자리에서 8자리 영숫자이어야 하므로 다음과 같은 동치 클래스가 나온다.

③ 길이가 6미만인 영숫자(비정상) : A

④ 6이상 8이하의 영숫자(정상) : Abc123

⑤ 길이가 8보다 큰 영숫자(비정상) : abcdefgh123

첫 자리는 영문자이어야 한다는 조건으로부터 다음의 동치 클래스를 얻을 수 있다.

⑥ 첫 자리 영문자(정상) : Abc123

⑦ 첫 자리 영문자가 아닌 문자나 숫자(비정상) : 123Abc

제안한 방법에 따라 적용해 본 시나리오는 로그인하여 즉시이체 거래를 하고 로그 아웃한 경우이다. <표 3>의 테스트 시나리오와 <표 4>의 자료사전을 이용하여 테스트 케이스를 작성하면 <표 5>와 같다.

5. 결 론

명세는 소프트웨어 시스템의 요구에 대한 정확한 정의를 나타내는 것뿐만 아니라 테스트의 기초가 된다. 상태 모델은 오퍼레이션의 정보와 함께 시스템의 동적 행위를 나타낸다. 상태 모델은 다이어그램과 정형적 문법 형식 모두 가능하여 정확한 요구를 나타낸 것이므로 이를 기초로 한 테스트는 무엇보다 정확한 검증이 가능하다. 정형적 명세를 사용하면 요구 명세가 모호하거나 불완전하던 부분을 없앨 수 있다. 또한 정확한 명세에 의하여 오류를 찾아내므로 테스트 후의 결과의 신뢰도를 크게 높일 수 있다.

이 연구에서는 웹 사이트가 정형적인 방법으로 표현 가능하며 정형적 명세에 표현된 오퍼레이션 정보로부터 테스트 데이터를 얻을 수 있음을 보였다. Object-Z를 이용하면 기능별로 시험할 수 있고 예상되는 결과가 표현되어 쉽게 비교할 수 있으며 테스트 데이터 찾기가 용이해진다.

향후 과제로 Object-Z로부터 STD를 자동 생성하는 도구의 개발이 요청된다. 도구는 우선 Object-Z 명세를 분석하고 자료흐름 분석이 수행하여야 한다. 자료흐름 분석의 결과는 명세에서 제공되는 계층의 각 레벨에서 상태 다이어그램을 작성하는데 사용된다. 또한 STD의 자동 생성이 가능하도록 Object-Z의 적당한 부분집합을 정의하는 것도 중요하다.

참 고 문 헌

[1] T. A. Powell et al., *Web Site Engineering : Beyond Web Page Design*, Prentice-Hall, 1998

[2] 최은만, "웹 기반 소프트웨어의 시험 및 검증 기술", 정보과학회지, pp.19-26, Nov., 2001.

[3] 최은만, "OCL로 기술된 객체지향 설계 명세의 테스트 케이스 생성", 정보처리학회논문지D, 제8-D권 제6호, pp.843-852, 2001.

[4] B. Beizer, Software Testing Techniques, International Thomson Computer Press, 2nd Edition, 1990.

[5] B. Potter et al., An Introduction to Formal Specification and Z, Prentice Hall, 2nd edition, 1996.

[6] C. Liu, D. Kung and P. Hsia, "An object-oriented web test model for testing Web applications," Proceedings of the First Asia-Pacific Conference on Quality Software, pp.111-120, 2000.

[7] D. Harel, "Statecharts : A Visual Formalism for Complex Systems," Science Computer Programming, Vol.8, pp.231-274, 1987.

[8] D. J. Richardson, S. L. Ahs, T. O. O'Malley, "Specification-based test oracles for reactive system," Proc. of 14th ICSE, pp.105-118, 1992.

[9] D. Kung, J. Gao, P. Hsia, J. Lin and Y. Toyoshima, "Design Recovery for Software Testing of Object-Oriented Programs," Proc. of the Working Conference on Reverse Engineering, IEEE Computer Society Press, Baltimore Mayland, pp.202-211, May, 1993.

[10] D. Kung, J. Gao, P. Hsia, Y. Toyoshima and C. Chen, "A Test Strategy for Object-Oriented Systems," Proc. of Computer Software and Applications Conference, IEEE Computer Society, Dallas Texas, pp.239-244, August, 1995.

[11] D. Kung, N. Suchak, P. Hsia, Y. Toyoshima and C. Chen, "On Object State Testing," Proc. of Computer Software and Applications Conference, IEEE Computer Society Press, pp.222-227, 1994.

[12] E. Miller, "WebSite Quality Challenge," White Paper, http://www.soft.com, 2000.

[13] E. Miller, "WebSite Testing," White Paper, http://www.soft.com, 2000.

[14] G. Smith, "State-Based Formal Methods for Distributed Processing : From Z to Object-Z," Technical Report 01-34, Software Verification Research Centre, University of Queensland, 2001.

[15] G. Smith, The Object-Z Specification Language, Kluwer Academic Publishers, 2000.

[16] H. Nguyen, Testing Applications on the Web, Wiley, 2001.

[17] H. S. Hong, Y. R. Kwon and S. D. Cha., "Testing of object-oriented programs based on finite state machines," In Proceedings of the Second Asia-Pacific Software Engineering Conference (Brisbane, Australia), pp.234-241, December, 1995.

[18] H. Zhu et al., "Software requirements validation via task analysis," Journal of Systems and Software, 61, pp.145-169, 2002.

[19] J. Conallen, "Modeling Web Application Architectures with UML," Communications of the ACM, Vol.42, No.10, pp. 63-70, October, 1999.

[20] J. Jacky, The Way of Z : Practical Programming with Formal Methods, Cambridge University Press, 1997.

[21] K. Chang et al., "Testing object-oriented programs : from formal specification to test scenario generation," Journal of Systems and Software, 42, pp.141-151, 1998.

[22] K. Chang, S. Liao, R. Chapman and C. Chen. "Test scenario generation based on formal specification and usage profile," International Journal of Software Engineering and Knowledge Engineering, Vol.10, No.2, 2000.

[23] M. Cartwright, Empirical Perspectives on Maintaining Web Systems : A Short Review, IEEE Trans. on Software Engineering, Vol.26-8, pp.786-796, Aug., 2000.

[24] P. A. Stocks, D. Carrington, "Test templates : A specification-based testing framework," Proceedings of the 15th International Conference on Software engineering, pp.405- 414, 1993.

[25] R. Binder, Testing Object-Oriented Systems : Models, Patterns, and Tools, Addison-Wesley, 1999.

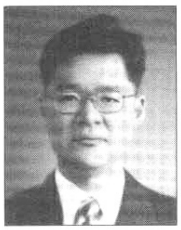
[26] R. Duke, P. King, G. Rose and G. Smith, "The Object-Z specification language, Version 1," Technical Report 91-1, Software Verification Research Centre, Department of Computer Science, University of Queensland, May, 1991.

[27] S. Liao, K. Chang, S. Seidman and C. Chen, "Testing object-oriented programs Based on Usage Profiles and Formal Specifications," In Proc. 8th International Conference on Software Engineering and Knowledge Engineering, pp.9-16.

[28] T. A. Powell et al. Web Site Engineering : Beyond Web Page Design, Prentice-Hall, 1998.

안 영 희

e-mail : yhahn@dongguk.edu
2001년 수원대학교 정보통신공학과(학사)
2003년 동국대학교 대학원 컴퓨터공학과(공학석사)
관심분야 : 소프트웨어 테스트, 정형적 모델, 요구 명세



최 은 만

e-mail : emchoi@dgu.ac.kr
1982년 동국대학교 전산학과(학사)
1985년 한국과학기술원 전산학과(공학석사)
1993년 일리노이 공대 전산학과(공학박사)
1985년~1988년 한국표준연구소 연구원
1988년~1989년 데이콤 주임연구원
2000년~2001년 콜로라도 주립대 전산학과 방문교수
1993년~현재 동국대학교 컴퓨터공학과 부교수
관심분야 : 소프트웨어 테스트, 프로세스와 메트릭, OO 및 컴포넌트 설계, Program Comprehension