다양한 웹 기반 소프트웨어의 테스트를 위한 효율적인 테스트 케이스의 생성

김 현 수[†] · 최 은 만[††]

요 약

인터넷을 통한 정보의 교류와 비즈니스가 활발해지면서 웹 기반 소프트웨어도 확대되고 있고 그만큼 품질 측면의 노력이 중요하다. 웹 컨텐츠의 링크나 웹 기반 프로그램을 검증하는 연구가 시도되었으나 다양한 웹 기반 구성 방법들을 커버하는 방법은 찾기 어려웠고 또한 적용 실험이 부족하다. 이 논문에서는 다양한 동적인 웹 기반 소프트웨어들을 타입별로 나누어 보고 동일한 방법으로 모델링 한 후 테스트 케이스를 생성하는 방법을 제안한다. 웹 기반 소프트웨어를 구성하는 개체들을 파악한 후 이를 개체 제어흐름 그래프(Object Control Flow Graph)와 개체 관계 다이어그램(Object Relationship Diagram)으로 모델링 한다. 모델에서 테스트 경로를 파악하고 테스트 키 포인트를 구동하는 테스트 케이스를 찾는다. 제안한 방법으로 다섯 가지 타입의 웹 기반 소프트웨어를 적용하여 실용성을 보였으며 이제까지 제안된 방법과 어떤 차이를 보이는지 비교하였다.

키워드 : 웹 기반 소프트웨어, 소프트웨어 테스트, 테스트 성능 비교, 테스트 모델

Effective Test Case Generation for Various Types of Web-based Software

Hyun Soo Kim[†] · Eun Man Choi[††]

ABSTRACT

As information and business communication via Internet are growing up, web-based software is wide spread and more important on the viewpoint of software quality than stand-alone. Research on verification of web content links and web-based program was tried, but has short on covering various types of web based software and making experiments to be applied in real testing practice. This paper suggests a modeling technique to be applied to dynamic and various types of web-based software. First, it identifies each elements consisting of web-based software and then construct a model of Object Control Flow Graph and Object Relationship Diagram. We can generate test cases covering all test paths of ORD or invoking key points test route. Suggested modeling method and test case selection technique are verified by applying five types of web-based software and compared with other web-based test techniques.

Key Words : Web-based Software, Software Test, Software Performance Evaluation, Test Model

1. 서 론

인터넷을 기반으로 하는 전자상거래, 사이버교육 등 다양한 웹 기반 애플리케이션이 등장하면서 비즈니스 세계에서 인터넷은 점차 중추적인 역할을 하고 있다[1]. e-비즈니스의 성장과 함께 웹 기반 애플리케이션의 품질 보증은 필수적이며 점차 높은 비중을 차지하면서 품질과 신뢰성을 보증하기 위한 방법론과 도구에 대한 높은 요구가 발생하고 있다. 특히 상품의 소개와 판매를 목적으로 하는 웹 기반 애플리케이션의 품질이 보증되지 않으면 상품 판매의 차질 뿐만 아니라 기업 브랜드의 가치에 큰 영향을 미칠 수 있다[12]. 인터넷을 매개로 한 전자상거래의 경우 웹은 모든 기업의 경쟁력 향상을 위한 필수 조건이 된다. 그러나 웹 애플리케이션은 점점 더 복잡해지고 있으며, 다양한 웹 기반 기술이 도입되어 웹 애플리케이션의 테스트와 품질 관리가 점점 더 어려워지고 있는 실정이다.

웹 애플리케이션 테스트는 그 중요도에 비하여 연구는 부족한 실정이다. 또한 웹 애플리케이션을 테스트하는 방법은 웹 페이지가 정적인지, 동적인지에 따라 달라질 수 있다. 정적인 경우는 HTML 구문 검사와 링크 검사로 가능하지만, 동적인 경우에는 통합 테스트가 필요하다[2, 6]. 테스트를 하려면 우선 웹을 분석하기 위한 모델이 필요하다. 분석은 테스트 포인트를 결정하기 위한 필요조건이며 테스트 케이스를 자동 생성하는 출발점이 될 수 있기 때문이다[10].

웹 애플리케이션을 위한 테스트 데이터를 생성하는 것은

† 정 회 원 : (주)대우조선 연구소 연구원
†† 정 회 원 : 동국대학교 컴퓨터멀티미디어공학과 교수, 교신저자
논문접수 : 2005년 1월 25일, 심사완료 : 2005년 5월 26일

결코 쉬운 일이 아니다. 그 이유는 여러 가지를 꼽을 수 있다. 웹 애플리케이션을 구성하는 컴포넌트로의 분할이 쉽지 않으며, 빈번히 일어나는 변경으로 테스트 노력이 많이 들고 반복적인 작업이 많다. 또한 정적 분석으로 테스트 드라이버를 작성할 수 있으나 대부분 네비게이션 링크, 철자법, HTML 호환 오류 등을 찾아내기 위한 목적으로만 사용되고 있다. 결정적인 이유는 웹 페이지의 동적인 부분과 정적인 부분을 동시에 테스트하기가 어렵기 때문이다[2]. 즉 웹 사이트를 구성하고 있는 링크의 유형이 웹 사이트의 노드 구조에 의해 결정되는 기본 링크로만 구성되어 있는 것이 아니라 기본 링크가 가지는 항해의 불편함을 해소해 주기 위하여 노드 구조에 기반을 두지 않는 새로운 링크인 부가적인 링크로도 구성되어 있기 때문에 웹 애플리케이션 테스트에 큰 어려움이 존재한다[3, 12].

테스트에 소요되는 비용은 크게 두 부분으로 구분하여 측정할 수가 있는데, 하나는 테스트를 수행하는데 필요한 시간이고 다른 하나는 테스터가 테스트 케이스를 만들고 그 테스트 케이스를 분석하는데 필요한 시간이다[4]. 이 두 가지 측면에서, 웹 애플리케이션이 점점 더 복잡해짐에 따라 테스트에 소요되는 노력과 비용은 점점 증가하고 있다. 결국 테스트 케이스가 복잡해지고 또한 그 수도 많아지게 된다. 웹 기반 애플리케이션 개발은, 개발에서 고객에게 사용되기까지의 시간이 매우 짧으므로 그 기간 동안 올바른 기능을 수행할 수 있는 제품을 개발해야 하는 것은 개발자에게 결코 쉬운 일이 아니다. 테스트 케이스의 복잡성과 그 수를 줄이는 것은 테스트에 소요되는 비용, 즉 노력과 시간을 크게 줄일 수 있을 것이다[4].

현재까지 웹 기반 애플리케이션을 테스트하기 위한 많은 연구가 진행되고 있지만, 웹의 복잡성 때문에 정적인 부분과 동적인 부분에 대한 테스트에 대한 어려움과 테스트 케이스 생성의 어려움, 생성된 테스트 케이스의 복잡함, 이로 인한 전체적인 테스트 비용의 증가 등 아직까지 해결되지 않은 많은 문제가 존재한다[6]. 이를 해결하고자 이 논문에서 제안하고자 하는 내용은 다음과 같다.

첫째, 웹 애플리케이션 분석을 위한 효과적인 모델링 방법은 무엇이며, 정적인 부분과 동적인 부분을 어떻게 테스트 할 수 있는가? 둘째, 웹 기반 애플리케이션에서 테스트 케이스 생성에 대한 비용-효율적인 접근 방법은 무엇인가? 셋째, 효율적인 테스트 케이스 생성을 위한 자동화 방법은 무엇인가이다. 이러한 문제를 해결하기 위하여 이 논문에서는 ORD(Object Relationship Diagram)를 이용하여 웹을 표현하고 있는 오브젝트들 사이의 관계를 알아보고 ORD를 바탕으로 OCFG(Object Control Flow Graph)를 이용하여 테스트 케이스를 생성하기 위해 웹 애플리케이션을 모델화한 뒤, 테스트 케이스를 생성하고자 한다.

위에서 제안한 연구 문제를 해결하기 위해서 2장에서는 기존의 웹 기반 애플리케이션의 연구에 대해서 알아보고, 3장에서는 웹 기반 애플리케이션을 테스트하기 위한 분석 방법을, 4장에서는 사례 연구로 테스트 케이스 생성 방법과 실제 테스트 케이스에 대한 비교에 대해 기술한다. 마지막 5장에서는 분석에 대한 결론과 향후 연구 과제로 이 논문을 마친다.

2. 웹 기반 테스트와 관련된 연구

웹 기반 소프트웨어는 매우 높은 수준의 품질이 요구된다[13]. 웹 애플리케이션은 전통적인 소프트웨어와 많은 차이가 있는데, 웹 애플리케이션은 네트워크 상에서 수행되면 네비게이션과 링크를 제공하고 사용자와 다양한 상호작용이 이루어지며, 대부분의 웹 테스트는 Javascript 검증 도구, 링크 체크 도구, HTML 검증기, 캡쳐/플레이백 도구 등을 이용하여 클라이언트 부분의 검증과 정적인 서버 부분의 검증에 초점을 맞추고 있다[11]. 현재 웹 테스트에 사용하고 있는 기법은 대부분 전통적인 기법, 예를 들면, 기능테스트, DB 테스트, 보안 테스트, 성능 테스트 등을 사용하며 관련된 테스트 도구를 사용하고 있다[17].

2.1 분석 모델을 이용한 테스트

웹 애플리케이션 분석 모델은 HTML 페이지를 통하여 서버와 클라이언트가 상호작용을 한다고 보고 웹 페이지를 물리적인 단위로 나누어 동적인 특징과 그들 간의 상호작용을 분석하기 위한 것이다[17]. 서버와 클라이언트 사이에 행해지는 상호작용을 분석하기 위해서 조합과 전이규칙이 사용된다.

Basis : P
Sequence (p->p1 · P2) : p1에 이어 p2가 온다.
Selection (p->p1|p2) : p1, p2 둘 중 하나를 선택하며, 동시에 둘을 선택할 수는 없다.
Aggregation (p1{p2}) : p2가 p1에 포함된다.

(그림 1) 조합 규칙

P*는 0 또는 더 많은 구역을 조합하는 것을 표현하며, 1 또는 더 많은 구역의 조합은 P+로 나타낼 수 있다. Pn은 정확히 n번의 조합을 의미하며, p(0|1)은 p가 선택적으로 포함될 수 있다는 것을 나타낸다. 예를 들어, P->p1 · (p2|p3)* · p4의 의미는 제일 처음 p1이 선택되며, 다음으로 p2 또는 p3이 0번 이상 선택적으로 선택되며, 그 다음으로 p4가 선택된다는 의미이다.

웹 애플리케이션에서 각 컴포넌트의 결합은 HTML 링크나 액션 링크로 이루어진다. 정적인 웹 페이지의 경우는 HTML 링크가 미리 결정되나 동적으로 생성되는 페이지의 경우는 실행될 때 사용자의 반응 등에 의하여 결정된다. 따라서 전이 규칙은 크게 HTML 링크에 의하여 다른 페이지

Link Transition (p=>q) : 링크를 통한 전이
Composite Transition (s->p) : s를 실행시킴으로써 일어나는 전이
Operation Transition (p→q) : 시스템의 구성으로 야기되는 전이

(그림 2) 전이 규칙

〈표 1〉 GradeServlet의 원자적 요소

p1 =	PrintWriter out = response.getWrite();
	out.println("<HTML>");
	out.println("<HEAD><TITLE>"+title+"</TITLE></HEAD>");
	out.println("<BODY>");
	If (Validate(ID, PASSWD))
	{
	out.println("<B> Grade Report </B>");

p2 =	for (int I=o; I<numberOfCourse; I++)
	out.println("<P><B>"+CourseName(I)+"</B>"+CourseGrade(I) +</P>");
	}
	else

p3 =	{
	out.println("Wrong ID or wrong password");
	out.println("<FROM MEHTOD=\
	"GET\" Action=\" SendEmail\">");
	out.println("<INPUT TYPE=\ "TEXT\" Name=\"SUBJECT\"
	SIZE=\"50\">");
	out.println("<INPUT TYPE=\"TEXTAREA\"
	NAME=\"BODY\" WIDTH+50>");
	out.println("<INPUT TYPE=\"SUBMIT\"
	NAME=\"SUBMIT\" VALUE=\"SUBMIT\">");
	out.println("<INPUT TYPE=\"RESET\"
	VALUE=\"RESET\"></FORM>");
	}

p4 =	out.println("<A HREF=\"index.html\"> Return To Main Page</A>"); out.println("</BODY></HTML>"); out.close

로 전이되는 경우와 사용자의 반응에 의하여 이루어지는 변이로 나눌 수가 있다. (그림 2)에서 p와 q는 컴포지트 섹션을 말하며, s는 HTML 페이지를 물리적 단위로 나눈 구역을 나타내고 s는 서블릿 또는 다른 소프트웨어의 컴포넌트를 의미한다.

<표 1>을 조합과 전이 규칙을 이용하여 분석하면 다음과 같다.

S = {index.html}
C = {GradeServlet = p1 · (p2* | p3) · p4, SendEmail = ...}
T = {S⇒GradeServlet, GradeServlet.p3⇒SendEmail,
GradeServlet.p4⇒S}

분석 모델을 이용해 테스트 케이스를 생성한다. 테스트 케이스는 원하는 페이지에 도달하기 위해 조합과 전이 규칙을 이용해 생성하고, 테스트 케이스의 실행은 서버와 클라이언트의 상호작용의 결과를 포함하게 된다. <표 1>에서 정상적으로 아이디와 패스워드를 입력하여 원하는 페이지에 도달한 경우의 테스트 케이스는 다음과 같다.

$$S \Rightarrow GradeServlet \rightarrow p1 \cdot p2 \cdot p4 \Rightarrow S$$

학생이 잘못된 아이디와 패스워드로 웹 페이지에 로그인을 하게 될 경우에 테스트 케이스는 다음과 같다.

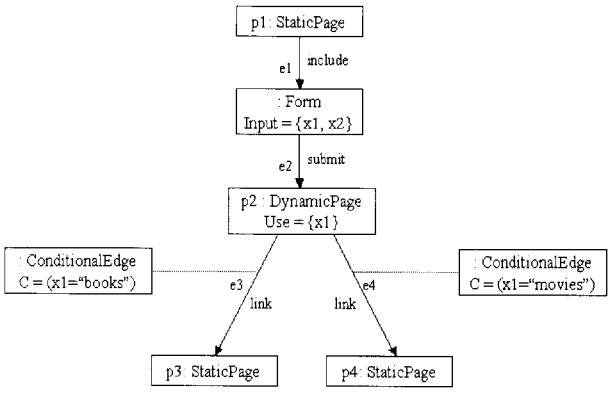
$$S \Rightarrow GradeServlet \rightarrow p1 \cdot p3 \cdot p4 \Rightarrow S$$

$$S \Rightarrow GradeServlet \rightarrow p1 \cdot p3 \cdot p4 \Rightarrow SendMail....$$

$$S \Rightarrow GradeServlet \rightarrow p1 \cdot p3 \cdot p4 \rightarrow Previous\ S \Rightarrow GradeServlet \rightarrow p1 \cdot p2 \cdot p4 \Rightarrow S$$

첫 번째 경우는 로그인이 제대로 이루어지지 않아 다시 시작 페이지로 돌아가는 경우이고, 두 번째는 관리자에게 메일을 보내는 경우, 세 번째는 시작페이지로 돌아가 로그인을 다시 시도하는 경우를 나타내고 있다.

이런 식으로, 웹 페이지를 물리적 단위로 나누어서 서로간의 상호작용을 분석해 분석 모델을 만들고 이를 통해 테스트 케이스를 생성한다. 그러나 이러한 방법은 동적인 웹 페이지 테스트에서 전이규칙을 사용하여 쉽게 표현할 수 있는 장점과 위의 예처럼 간단한 시나리오에 대해서는 간결한 표현식을 이용한 테스트 케이스 생성이 가능하지만, 복잡한 시나리오를 포함한 다양한 컨텐츠를 가진 웹 애플리케이션의 경우에는 테스트 케이스 생성이 어려운 점이 존재할 수 있다. 복잡한 웹 사이트를 테스터가 테스트를 위해 물리적인 단위로 나누기 위해서는 많은 시간과 노력이 필요하기 때문이다. 물리적인 단위로 웹 사이트를 구분하더라도 테스트 케이스를 다시 생성해야 하는 번거로움이 존재하며 테스터의 역량에 따라 물리적인 단위가 틀려질 수 있기 때문에 테스트에 대한 신뢰성을 보장할 수 없다. 또한 웹 사이트에는 웹 사이트의 구조를 고려한 기본적인 링크만 존재하는 것이 아니라 사용자의 편의를 위해 존재하는 부가적인 링크가 존재하므로 이러한 부가적인 링크를 가진 웹 애플리케이션의 경우 물리적인 단위로 구분하기가 어렵기 때문이다.



(그림 3) 입력 폼을 가진 사이트의 UML 모델

2.2 UML 모델 기반 테스트

[10]에서 Ricca와 Tonella는 UML(Unified Modeling Language)을 이용하여 웹 애플리케이션 모델을 만들어 웹 애플리케이션 분석과 테스트 케이스 생성에 대해 연구하였다. 이 연구에서는 가능한 경로를 표현하고 이를 이용해서 임시 테스트 케이스(quasi test case)를 생성한다. 정적 검증을 위해 사용자에게 제공되는 네비게이션 패스나 사용자로부터 모아진 정보의 데이터 흐름에 초점을 맞추고 있다. 도달 불가능한 페이지, 고스트 페이지, 도달한 프레임, 데이터 의존성, 최단 경로 등이 정적 검증의 대상이다. 동적 검증을 위해서는 화이트 박스 테스팅에 초점을 맞추고 있으며, 페이지 테스트, 하이퍼링크 테스트, 정의/사용 테스트, 총 사용 테스트, 총 경로 테스트 등이 검증 대상이다.

테스트 작업의 주된 대상인 입력 폼을 가진 사이트는 다음과 같이 모델링 할 수 있다.

(그림 3)에서 주의 깊게 볼 점은 동적페이지 p2에서 x1의 값을 입력받아 ConditionalEdge에서 조건을 검사하여 정적인 페이지 p3와 p4로 나누어진다는 점이다. (그림 3)에서 테스트를 위해 생성된 경로식(path expression)을 다음이 구한다.

path expressioin : e1e2(e3+e4)

이 방식은 UML을 사용하여 표현함으로 표준적인 테스트 케이스를 생성한다는 장점을 가지고 있으나, 동적인 페이지 뿐만 아니라 정적인 페이지 또한 입력값을 고려하여 경로 표현식 테스트 케이스를 생성하므로 불필요한 경로의 테스트 케이스를 생성할 뿐만 아니라 경로 표현식을 작성할 때, 많은 페이지를 포함하고 있는 웹 사이트에 대해 테스트 케이스를 생성하는 경우는 테스트 케이스에 대한 수가 많아지는 경우가 발생한다. 또한, 테스트 케이스의 표현이 복잡하지며, 테스트를 위해서 name-value 값을 테스터가 직접 만들어야 하기 때문에 테스트 케이스 생성 시간이 많이 걸릴 수 있고, 그 입력 값의 정확도를 예상하기 힘든 점이 있다.

2.3 사용자 세션을 이용한 테스트

[16]에서는 사용자 세션 데이터를 이용하여 웹 애플리케이션을 테스트하는 방법을 설명하고 테스트 결과를 기존의 방법과 비교 설명하고 있다. 화이트 박스 웹 애플리케이션 테스트 기술의 제한된 요소는 입력 값을 선택하는 것이 느리고 자동화할 수 없는 부분이 많기 때문에 이러한 부분을 보완하기 위해서 사용자 세션을 이용한 테스트 방법을 제안하고 있다.

사용자 세션 데이터를 위해 클라이언트 요구를 캡쳐하고 저장하는데, 모아진 url과 name-value로 테스트 케이스를 만들 수 있는 여러 가지 방법이 있다. 개별적인 사용자 세션을 순차적으로 리플레이하는 방법, 다수의 사용자로부터의 상호작용을 혼합하여 리플레이하는 방법, 일반적인 사용자 요구와 문제가 있는 요구를 혼합하는 방법 등이다. 이 방법은 사용자의 행위에 대해 정밀하지 않을 수도 있지만 사용자의 조작상의 측면에서, 테스트 노력과 관련해서 유용한 방법이다. [16]에서는 두개의 명확한 테크닉에 초점을 맞추고 있는데 하나는 전체 세션을 적용한 방법이고 다른 하나는 혼합한 세션을 리플레이하는 방법을 사용하고 있다. 결과를 보면 전제적으로 큰 차이를 보이지는 않으며, 화이트박스 테스트 기술이 다른 기술보다 조금 더 많은 결함 오류 검출률을 보이고 있지만, 각각의 기술들이 어떤 점에서 뛰어난지에 대해 정확히 비교 결과를 제시하지는 못하고 있다. 또한 어떤 기술이 어떤 부분의 에러에 대한 발견이 뛰어난지에 대한 결론이 없는 것이 이 실험 결과의 단점이다.

결론적으로 이 실험 방법의 장점으로는 테스트 케이스 생성을 위해 사용자 요구를 이용함으로써 비교적 웹 애플리케이션에 의존하지 않고, 사용자 세션 자체가 테스트 케이스가 되므로 테스트에 적은 노력이 든다는 점이다. 그러나 테스트를 위한 웹의 상태는 전혀 고려하지 않은 단점이 있으며, 웹 애플리케이션의 개발이 완료된 후 실질적인 사용자들의 세션을 이용하는 것이므로 개발단계에서의 테스트는 불가능한 단점이 있다.

3. ORD와 OCFG를 이용한 웹 애플리케이션 분석 및 테스트 케이스 생성

다양한 웹 애플리케이션에 대한 테스트 케이스를 만들려면 먼저 웹 애플리케이션을 모델링 할 수 있는 범용적인 방법이 필요하다. 본 연구에서는 개체 제어 흐름도(Object Control Flow Graph)를 도입하여 웹 애플리케이션을 모델링하고 분석을 하고자 한다. 개체 제어 흐름도란 웹 애플리케이션을 기본 문법단락으로 나누고 이를 그래프로 표현하여 웹 페이지의 제어 흐름을 한 눈에 볼 수 있는 그래프이다. 그러나 웹 애플리케이션은 워낙 규모가 커서 단번에 시스템 전체에 대한 개체 제어 흐름도를 그리기가 어렵다. 따라서 웹 애플리케이션을 구성하는 여러 가지 다양한 개체, 예를 들면 클라이언트 페이지, 서블릿, 메뉴 등의 관계를 파악하고 이를 바탕으로 더 세부적인 각 페이지 내부의 제어 흐름을 파악해 나가야 한다.

웹 사이트는 제공하는 상호작용에 따라 크게 5가지로 나

눌 수가 있다. HTML 문서로 표시되는 정적 웹 사이트, 사용자로부터 정보를 모으기 위하여 사용하는 폼 기반 정적 웹 사이트, 데이터베이스를 접근하기 위하여 front-end로 사용하는 웹 사이트, 데이터베이스 컨텐츠에서 수행 쿼리 등을 사용하여 검색하고 검색 결과가 HTML 문서로 동적으로 표시되는 동적 데이터 접근 정적 웹 사이트, 사용자 맞춤 페이지를 제공하기 위한 동적 생성 사이트, 마지막으로 일반적인 비즈니스 프로세스의 일부분으로 간주되는 형식인 클라이언트/서버 애플리케이션 유형의 웹 기반 애플리케이션이다.

이러한 다양한 웹 사이트의 성격을 커버하는 범용적인 모델링 방법과 이를 근거로 웹 애플리케이션의 테스트 케이스를 찾아내는 방법을 3장에 소개한다.

3.1 개체 제어 흐름도

웹 애플리케이션에서 자료 흐름 테스팅 정보를 추출하기 위하여 웹 애플리케이션의 각 컴포넌트를 속성과 오퍼레이션을 포함한 개체 모델로 표현하여야 한다. 개체는 세 종류의 타입이 존재한다[17].

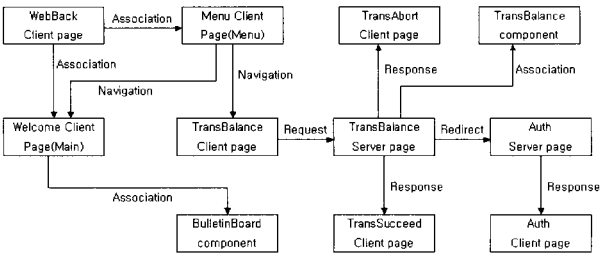
〈표 2〉 개체 타입

개체 타입	설 명
클라이언트 페이지	내장 스크립트를 가진 HTML문서 클라이언트 측의 웹 브라우저로 만들어진다.
서버 페이지	Common Gateway Interface (CGI) Active Server Page (ASP) Java Server Page (JSP)
컴포넌트	Java applet ActiveX Control Java Bean 클라이언트 페이지와 상호 작용하는 다른 프로그램 모듈

오브젝트 사이의 내부 의존성을 표현하기 위해 ORD(Object Relation Diagram)를 사용한다.

ORD = (V,E)
V는 오브젝트로 표현되는 노드들의 집합
E는 <= V * V인 E는 오브젝트 사이의 관계를 표현하는 Edge의 집합

오브젝트 사이의 관계는 6가지 타입으로 표현할 수 있다. O-O프로그램에서 흔히 사용하는 aggregation, association의 2가지 타입과 클라이언트 페이지와 서버 페이지 사이에서의 특별한 association관계인 request, response, navigation, redirect의 4가지 타입으로 표현할 수 있다. 특별한 association관계는 서버 페이지와 클라이언트 페이지의 관계를 구체적으로 규명하기 위하여 사용하였고, association 관계는 흔히 생각하는 오브젝트간의 구조를 표현하기위해 사용하였다. 예를 들어, 동적으로 생성되는 클라이언트 페이지는 서버 페이지에 정적인 출력으로써 이미 포함되어 있고, 이렇게 생성될 페이지들은 각각의 고유한 구조와 행위를 가지고

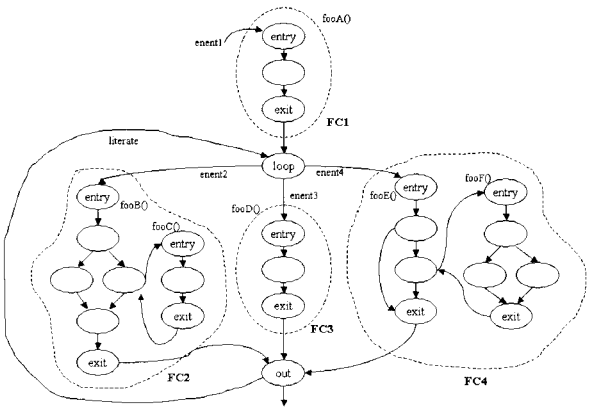


(그림 4) Web Bank에 대한 ORD의 예

있다. 그러므로 이렇게 생성될 페이지들은 서버 페이지들과 분리하여 독립적인 오브젝트로 취급할 수 있게 때문에 서버 페이지와 클라이언트 페이지의 관계를 표현하는 특별한 association관계가 아니라 일반적인 association관계를 사용할 수 있다. 특별한 association인 4가지 타입을 설명하면, 클라이언트 페이지에서 서버 페이지를 요청할 때 사용하는 request, 서버 페이지의 응답에 의해 클라이언트가 생성될 때 사용하는 response, 2개의 클라이언트 페이지에서 하나의 클라이언트 페이지가 다른 하나의 클라이언트 페이지에 대해 하이퍼링크를 가지고 있을 때 사용하는 navigation, 2개의 서버 페이지에서 하나의 서버 페이지가 다른 하나의 서버 페이지에 HTTP 요청을 지시할 때 사용하는 redirect 관계로 설명할 수 있다.

(그림 4)는 Web Bank에 대한 간략한 ORD의 예이다. Web Bank 클라이언트 페이지는 2개의 클라이언트 페이지 Menu 클라이언트 페이지와 Welcome 클라이언트 페이지와 association 관계에 있으며, Menu 클라이언트 페이지는 은행의 잔고를 확인하는 TransBalance 클라이언트 페이지와 Welcome 클라이언트 페이지와 navigation 관계에 있다. Welcome 클라이언트 페이지는 은행의 새로운 소식을 전하는 BulletinBorad 컴포넌트와 association 관계에 있다. TransBalance 클라이언트 페이지는 은행잔고의 확인을 위해 TransBalance 서버 페이지에 요청을 하게 되고, TransBalance 서버 페이지는 고객의 인증을 담당하는 Auth 서버 페이지에 고객의 인증 정보를 요청하게 된다. Auth 서버 페이지는 Auth 클라이언트 페이지에 그 결과를 응답하게 된다. TransBalance 서버 페이지는 은행 잔고 확인 시 TransSucceed 클라이언트 페이지와 TransAbort 클라이언트 페이지에 성공 여부를 응답하게 된다.

웹 애플리케이션에서 각각의 GUI 이벤트는 함수들의 집합에 의해 실행되고 사용자들의 상호작용에 의존한다. 그리고 다양한 함수의 요청 시퀀스가 존재하고 있다. 그러므로 데이터의 정확한 상호작용을 보증하기 위해서는 다른 함수 또는 컴포넌트 등에 의해 실행되는 경로에 대한 정확한 실행이 필요하다. 개체에서 다른 함수들의 요청 시퀀스로부터 실행 경로의 정보를 얻기 위해서 OCFG(Object Control Flow Graph)가 사용된다. OCFG는 개체 안에 함수들의 모든 CFG(Control Flow Graph) 또는 ICFG(Interprocedure Control Flow Graph)를 연결하여 표현될 수 있다.



(그림 5) 웹 페이지 개체의 OCFG

(그림 5)는 웹 페이지를 OCFG로 표현한 예이며, OCFG는 점선으로 표현된 2개의 CFG와 2개의 ICFG로 연결되어 있으며, 각각의 CFG와 ICFG는 entry와 exit가 있고, loop를 통해서 각 CFG는 데이터 값에 의해 다른 CFG로 이동하거나 out을 통해 빠져나오기도 한다. 위의 웹 페이지의 모델은 각각이 포함하고 있는 함수 FC1, FC2, FC3, FC4를 통해 표현할 수 있는 경로식은 다음과 같다.

$$FC1(FC2|FC3|FC4)^{+}$$

3.2 웹 애플리케이션 모델 분석 알고리즘

구조적 테스팅 기술을 사용한 실행 가능한 경로의 실행 없이 웹 애플리케이션의 품질을 보증하기란 어려운 일이며, 웹 애플리케이션 모델이 존재하더라도 이를 분석하고 테스트 케이스 생성을 위해서는 테스터가 수작업을 통하여 일일이 테스트 케이스 생성을 할 수 밖에 없지만, 모델을 바탕으로 테스트 케이스 생성을 자동화하기 위해서는 웹 애플리케이션 모델 분석 알고리즘이 필요하다. 웹 페이지를 구성하는 각각의 오브젝트를 하나의 노드로 간주하고 페이지 또는 오브젝트 간의 이동 경로를 간선으로 생각하여 입력으로 각각의 간선으로 이어진 노드가, 출력으로는 노드의 번호로 표시되는 테스트 케이스 생성을 위한 웹 애플리케이션 모델 분석 알고리즘은 다음과 같다.

이미 방문되었던 노드를 만나거나 분기노드를 만날 때까지 경로를 따라 이동하면서 분기노드를 만나면 큐를 검사하여 분기노드가 큐에 없는 경우 이 분기노드가 최상위 노드가 되고 지금까지의 경로를 저장한다. 한 번 방문되었던 노드는 배열에 저장하고, 각각의 노드 방문 시마다 배열에 저장된 노드와 비교하여 방문되었는지, 방문되지 않았는지에 대한 검사를 실시하며 방문되었으면 그 방문된 노드를 포함한 경로를 저장한다. 자식 노드의 자 노드를 검사하여 자 노드가 하나 이상이면 자식 노드를 분기노드로 간주하여 경로를 추가한다. 자식 노드를 방문하기 위해서 항상 자식 노

```
Path      [][] path; //2차원 배열
Node      [] hNode; //1차원 배열
last = 0;
hNode[last] = 1 // 최상위 노드
cNode = 1; // 현재 노드
visit(cNode){
        if(hNode.has(cNode)){//hNode[]에 cNode가 있으면
                path.addPath(currentPath[])// 현재까지의 경로 저장
                if(cNode = getNextBrother()){
                        visit(cNode);
                else{//다음의 형제 노드가 없으면
                        parentNode = cNode.getParent();
                        cNode = parentNode.getNextBrother();
                        //부모노드의 다음 형제 노드를 현재 노드에 할당
                        parentNode.remove();
                        //현재 노드의 부모노드를 큐에서 삭제
                        visit(cNode);
                }
        }else if(hNode[last] is 분기노드){
                last++;
                hNode[last] = cNode;
                hNode[last].addQueue(new queue()); //현재 최상위 노드에
                                                   큐할당
                hNode.addSon();// 현재 노드가 가지는 모든 자식노드를
                        큐에 저장
                visit(hNode.getSon[0]);
        }
        else{//하나이하의 자식을 갖는 처음 방문한 노드일 경우
                addPath();
                visit(cNode.getNextBrother());
        }
}
```

(그림 6) 웹 애플리케이션 모델 분석 알고리즘

〈표 3〉 SearchEngines 서블릿의 atomic section

P1	public class SearchEngines extends HttpServlet {
	public void doGet(HttpServletRequest request,
	HttpServletResponse response)
	throws ServletException, IOException {
	String searchString = request.getParameter("serchString");
	if ((searchString == null) || (searchString.length()==0) {

P2	reportProblem(response,"Missing search string.");
	return;
	}

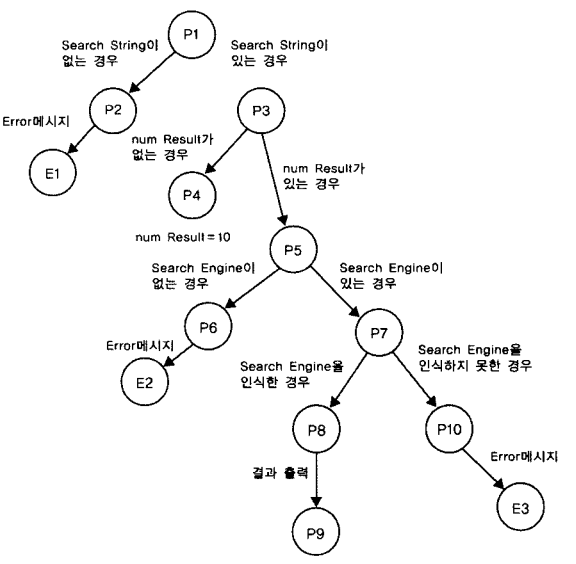
P3	searchString = URLEncoder.encode(searchString);
	String numresults = request.getParameter("numresults");
	if ((numresults == null)||(numresults.equals("0") ||
	(numreults.length() == 0)) {

P4	numResults="10" }
P5	String searchengine = request.getParameter("serchEngine"); if (searchEngine == null) {
P6	reportProblem(response, "Missing search engine name"); return; }
P7	SearchSpec[] commonSpecs =SearchSpec.getCommonSpecs(); for(int I=0; I<commonSpecs.length; I++) {
P8	searchSpec searchSpec = commonSpecs[I]; if (searchSpec.getName().equals(searchEngine)) {
P9	String url = searchSpec.makeURL(searchString, numresults); response.sendRedirect(url); return; } }
P10	reportProblem(response, "Unrecognized search engine"); }

드가 분기노드인지, 방문되었던 노드인지를 검사하고 자식 노드가 분기노드가 아니면 방문되었던 노드, 분기노드를 만날 때까지 그리고 더 이상 자식 노드가 없을 때까지 이동을 계속하고 경로를 추가한다. 자식노드를 방문하는 순서는 노드 A가 BCD라는 자식노드를 가지고 있으며, BCD를 큐에 저장하고 B부터 차례로 pop하여 큐가 널이 될 때까지 자식 노드를 방문한다.

[17]에서 atomic section은 정적인 HTML 페이지 또는 HTML로 출력되는 서버 프로그램의 section을 말하지만, 위의 <표 3>에서는 논리 흐름에 따라 다양한 함수에 의해 호출되는 오브젝트 개념으로써의 atomic section으로 정의하였다. SearchEngines 서블릿은 reportProblem(), doGet(), doPost(), SearchSpec()의 메서드를 가지고 있으며, SearchEngines 서블릿을 일반적인 OCFG로 표현하여 패스 표현에 의한 테스트 케이스를 생성하면 논리 흐름에 따라 많은 테스트 케이스가 생성된다. 예를 들어, 정상적으로 수행되는 경우, Search String이 없는 경우와 있는 경우, numResult가 있는 경우와 없는 경우 그리고 이들을 모두 조합한 경우 등 논리 흐름에 따라 복잡한 테스트 케이스가 생성될 수 있다. 이러한 테스트의 복잡성을 줄이기 위해서 이 논문에서는 일반적인 OCFG에 위의 알고리즘을 적용하여 보다 효율적이고 간단한 테스트 케이스를 생성하고자 한다.

(그림 7)에 표현한 모델을 이용하여 테스트 케이스를 생성하기 위해 각각의 노드를 입력으로 웹 애플리케이션 모델 분석 알고리즘을 적용하면 출력으로는 각각의 노드 번호로 표현되는 모듈화된 테스트 케이스가 자동으로 생성되게 된다. 테스트 케이스 자동 생성을 위해 테스터는 각각의 노드 번호를 입력해야하는 단점이 있다.



(그림 7) SearchEngines 서블릿의 OCFG

3.4 테스트 케이스 생성

[10]에서 제안하고 있는 표준 경로식 테스트 케이스 생성 방법을 사용하여 테스트 케이스를 생성하면, 표준적인 하나의 테스트 케이스로 웹 애플리케이션의 구조적 테스트 수행할 수 있다는 장점이 있지만, 하나의 테스트 케이스로 웹 애플리케이션을 커버하기에는 테스트 케이스가 복잡해지며, 전체가 아닌 일부분만을 테스트하기에는 어려움이 발생한다. [10]에서 제안한 방법으로 구조적 테스트를 위한 경로식 테스트 케이스를 생성하면 다음과 같다.

$$P1\cdot(P2\cdot E1+P3\cdot P4^{(0|1)}\cdot P5\cdot(P6\cdot E2+P7\cdot(P8\cdot P9+P10\cdot E3)))$$

이러한 어려움을 해결하기 위하여 (그림 7)을 웹 애플리케이션 모델 분석 알고리즘을 적용하여 테스트 케이스를 생성하면 <표 4>와 같다.

위의 <표 4>에서 나타난 것과 같이 생성된 테스트 케이스가 모듈화 된다면, 테스터가 전체 프로그램의 일부분, 예를 들어 검색엔진을 선택하지 않은 경우까지만 테스트하고자 할 때는 전체적인 경로를 고려하지 않고도 1, 2, 3번만으로 간편하게 테스트를 수행할 수가 있으며, 테스트 케이스 생성이 자동화되므로 테스트 케이스를 생성하고 분석하기에 보다 적은 시간이 소요되므로 전체적인 테스트에 소요되는 비용 또한 줄일 수가 있다. 그리고 구조적 분석을 통해 각각의 개체를 모듈화 함으로써, 웹 애플리케이션의 변경이 생기더라도 테스터가 쉽게 그 구조를 파악할 수 있으며, 빠른 시간에 테스트 전략을 세울 수 있다는 장점이 있다.

4. 다양한 웹 사이트에 대한 모델링 및 테스트 케이스 생성 실험과 비교

이 장에서는 웹의 상호작용에 따라 테스트 케이스 생성이 어떻게 달라지는지를 밝히기 위해 3장에서 제안한 ORD를 이용하여 웹을 모델링한 후 웹의 상태를 분석하고 웹 애플리케이션 모델 분석 알고리즘을 적용하여 웹 애플리케이션의 구조적 테스트를 위한 테스트 케이스를 생성한 후, 그 결과를 [10]에서 제안한 방법과 비교하여 설명한다.

웹 사이트가 제공하는 상호작용에 따른 분류에는 크게 5가지로 나눌 수가 있다. 정적 웹 사이트, 폼기반 정적 웹 사이트, 동적 데이터 접근 정적 웹 사이트, 동적 생성 사이트, 마지막으로 웹 기반 애플리케이션이다. 이 중에 정적 사이트는 외부 입력 데이터가 없고 웹 페이지 내부 의존성이 단순하여 ORD 구성과 테스트 케이스 생성이 간단하다. 또한 폼 기반 정적 웹 사이트에서 정적 웹 사이트에 대한 요소들을 다 포함하고 있으며, 고객 맞춤형 페이지를 제공하는 동적 생성 사이트에 대한 요소들은 웹 기반 애플리케이션이 다 포함하고 있으므로, 나머지 세 가지 종류의 웹 사이트에 대하여 모델링과 테스트 케이스를 생성한 실험을 소개한다.

실험 한 세 가지 부류의 웹 사이트의 대한 규모와 상세한 설명은 아래 <표 5>와 같다.

4.1 폼 기반 정적 웹 사이트

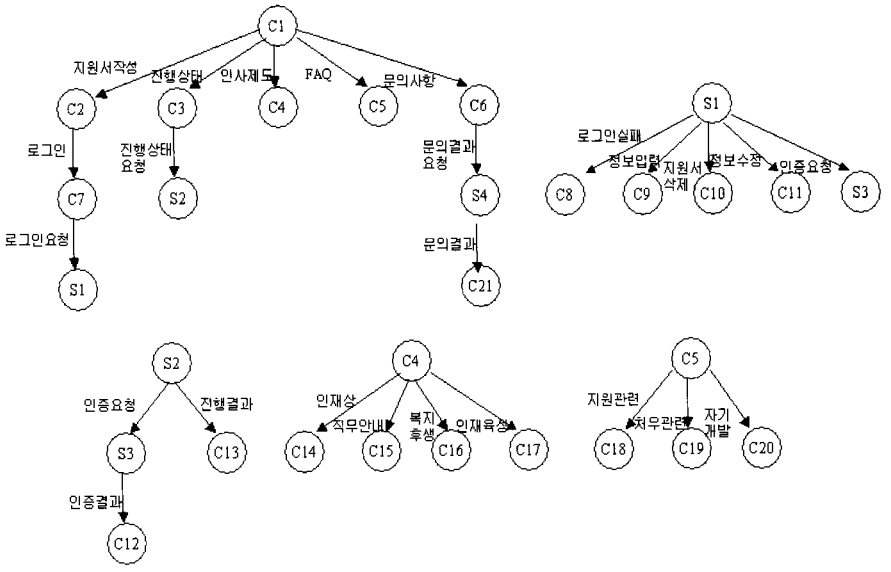
폼 기반 정적 웹 사이트는 폼을 사용하여 사용자로부터 모아진 정보를 사용하는 사이트로써, 일반적으로 문서의 전

〈표 4〉 생성된 테스트 케이스

NO	테스트 케이스	예상 결과	성공/실패	비고
1	P1·(P2·E1+P3) 검색어가 주어진 경우와 검색어가 주어지지 않은 경우로 나누어 테스트	검색어가 주어진 경우는 P3으로 이동하고, 검색어가 주어지지 않은 경우는 E1에서 에러 메시지를 보여줌		
2	P3·P4$^{(0\|1)}$·P5 검색될 수가 입력되는 경우와 입력되지 않는 경우, 입력되지 않은 경우 자동적으로 검색 수는 10이 입력	검색 수가 입력되면 P5로 이동, 검색 수가 입력되지 않은 경우 자동적으로 10이 입력되어 P5로 이동		
3	P5·(P6·E2+P7) 검색엔진을 선택한 경우와 선택하지 않은 경우	검색엔진을 선택하지 않은 경우는 E2에서 에러 메시지를, 선택한 경우는 P7로 이동		
4	P7·(P8·P9+P10·E3) 검색 엔진을 인식한 경우와 인식하지 못한 경우	검색 엔진을 인식한 경우는 P9에서 결과를 출력, 인식하지 못한 경우는 E3에서 에러 메시지를 보여줌		

〈표 5〉 대상 사이트에 대한 구분

웹 사이트 구분	설 명	웹 사이트의 규모	
		Object 수	페이지 수
폼 기반 정적 웹 사이트	폼을 사용하여 사용자로부터 모아진 정보를 사용하며, 일반적으로 문서의 전송을 위한 사이트	25	25
동적 데이터 접근 정적 웹 사이트	웹 사이트를 데이터베이스 접근을 위한 Front-end로써 사용하며, 쿼리를 사용하여 검색이 가능하고 검색 결과는 HTML 문서로 표시되는 사이트	32	42
웹 기반 애플리케이션	클라이언트/서버 애플리케이션으로써의 일반적인 비즈니스 프로세스를 처리하는 웹 사이트	27	45



(그림 8) 폼 기반 정적 웹 사이트의 OCFG

〈표 6〉 폼 기반 정적 웹 사이트의 테스트 케이스

No	생성된 Test Case	
	Ricca & Tonella의 테스트 케이스	웹 애플리케이션 모델 분석 알고리즘을 적용한 테스트 케이스
1	C1·(C2·C7·S1·C3·S2+C4+C5+C6·S4·C21)	C1·(C2·C7·S1·C3·S2+C4+C5+C6·S4·C21)
2	C1·C2·C7·S1·(C8+C9+C10+C11+S3)	S1·(C8+C9+C10+C11+S3)
3	C1·C3·S2·(S3·C12+C13)	S2·(S3·C12+C13)
4	C1·C4(C14+C15+C16+C17)	C4(C14+C15+C16+C17)
5	C1·C5(C18·C19+C20)	C5(C18·C19+C20)

송을 위한 사이트를 말한다. 실험대상은 회사에 지원하기 위한 사이트로써 일정한 폼에 사용자가 정보를 입력하면 그 내용이 회사의 서버로 전송되는 사이트이다. 이 사이트는 사용자를 위한 지원서 작성 페이지와 진행상태 확인 페이지, 인재제도 페이지, FAQ 페이지, 문의사항 페이지로 구성되어 있다. 이 사이트의 정보를 ORD와 웹 애플리케이션 모델 분석 알고리즘을 사용하여 OCFG로 표현하면 (그림 8)과 같다.

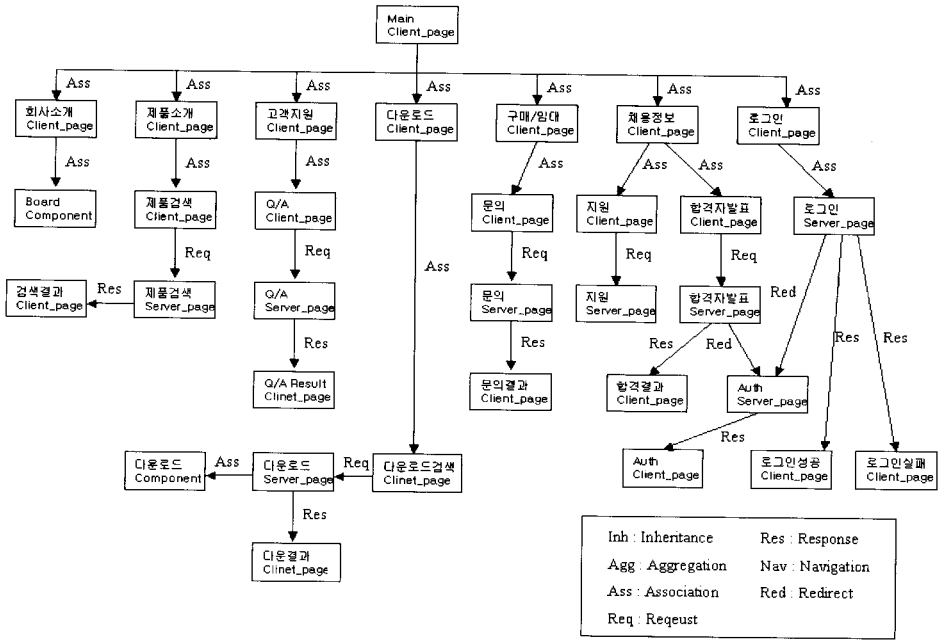
<표 6>는 폼 기반 정적 웹 사이트에 대하여 생성한 테스트 케이스이다.

4.2 동적 데이터 접근 정적 웹 사이트

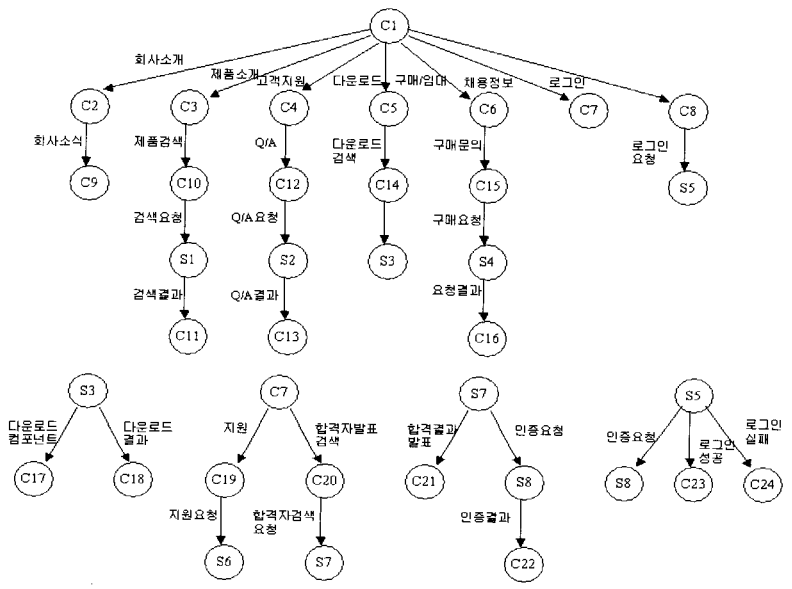
데이터베이스 접근을 위한 front-end로써 사용하는 웹 사이트를 말하며, 웹 페이지 사용자는 데이터베이스 컨텐츠에서 수행 쿼리 등을 사용하여 검색할 수 있다. 검색 결과는 정적인 HTML 문서로 표시되는 사이트이다. 동적 데이터 접근 정적 웹 사이트의 모델링을 위해 선정한 실험 대상 사이트는 사용자에게 회사 또는 상품에 대한 전반적인 정보를 제공하며, 고객 문의 또는 고객 지원 부분에서 검색을 통해 다운로드를 제공하는 사이트이다.

이 사이트의 메인 클라이언트 페이지는 7개의 클라이언트 페이지와 association 관계에 있다. 특히 주목하고 테스트할 동적인 데이터 접근 부분은 채용정보 클라이언트 페이지로 2개의 클라이언트 페이지와 association 관계로 연결되어 있는데, 하나는 지원 클라이언트 페이지이고 다른 하나는 합격자발표 클라이언트 페이지이다. 지원 클라이언트 페이지에서 지원자가 작성한 내용은 지원 요청을 위해 지원 서버 페이지에 request관계로 연결되며 합격자 발표 클라이언트 페이지는 합격자 발표 서버 페이지에 합격 여부를 요청하게 된다. 합격자 발표 서버 페이지는 사용자에 대한 인증을 위해 인증 서버 페이지에 redirect관계로 연결되어 있으며, 인증 서버 페이지는 그 결과를 인증 클라이언트 페이지에 응답하게 된다. 이 사이트의 정보를 ORD와 분석 알고리즘을 바탕으로 OCFG로 표현하면 (그림 10)과 같이 되고 여기에서 테스트 데이터를 산출하면 <표 7>과 같다.

동적 생성 사이트의 특징은 모든 사용자에게 고객 맞춤형 페이지를 제공하기 위한 것이며, 그 결과는 정적 페이지로 표시된다는 점이다. 모델링 방법은 이제까지 소개한 방법이 그대로 적용된다. 다만 4.4에 설명하는 것처럼 Ricca의 방법은 웹 사이트를 표현하고 있는 모든 페이지에 대해 즉, 동적 페이지뿐만 아니라 입력값이 필요 없는 정적 페이지에 대해서도 입력값을 고려하고 있기 때문에 테스트에 필요 없는 경로가 많이 생성되는 단점이 있다. 또한 웹 애플리케이션 모델 분석 알고리즘을 적용한 방법은 동적 페이지에 대해서만 입력값을 고려하기 때문에 적의 수의 테스트 케이스



(그림 9) 동적 데이터 접근 정적 웹 사이트의 ORD



(그림 10) 동적 데이터 접근 정적 웹 사이트의 OCFG

〈표 7〉 동적 데이터 접근 정적 웹 사이트에 대한 테스트 케이스

No	생성된 Test Case	
	Ricca & Tonella의 테스트 케이스	웹 애플리케이션 모델 분석 알고리즘을 적용한 테스트 케이스
1	C1·(C2·C9+C3·C10·S1·C11+C4·C12·S2·C13+C5·C14·S3+C6·C15·S4·C16+C7+C8·S5)	C1·(C2·C9+C3·C10·S1·C11+C4·C12·S2·C13+C5·C14·S3+C6·C15·S4·C16+C7+C8·S5)
2	C1·C2·C9	S3·(C17+C18)
3	C1·C3·C10·S1·C11	C7(C19·S6+C20·S7)
4	C1·C4·C12·S2·C13	S7·(C21+S8·C22)
5	C1·C5·C14·S3·(C17+18)	S5(S8+C23+C24)
6	C1·C6·C15·S4·C16	
7	C1·C7·C19·S6	
8	C1·C7·C20·S7·(C21+S8·C22)	
9	C1·C8·S5·(S8·C22+C23+C24)	

만으로 Ricca의 방법과 같은 성능의 테스트 케이스 생성이 가능하다. 또한, Ricca의 테스트 케이스 생성보다는 표현 방식이 간단하여 구조적 테스트를 위한 노력이 줄어든다는 장점이 있다.

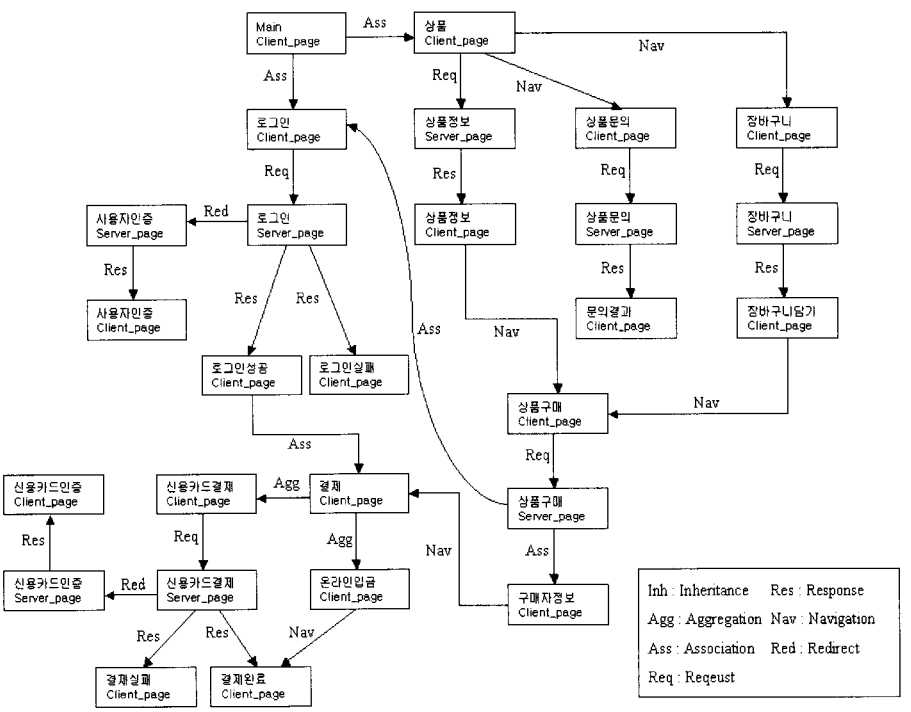
4.3 웹 기반 애플리케이션

웹 기반 애플리케이션은 일반적인 비즈니스 거래의 처리를 위한 사이트이다. 대표적인 예로는 인터넷 뱅킹을 위한 사이트 또는 물건을 구매하기 위한 쇼핑몰을 들 수 있겠다. 웹 기반 애플리케이션에서는 비즈니스 거래를 위해 클라이언트와 서버 페이지의 상호작용이 활발히 이루어진다. 실험 대상으로 선정한 사이트는 고객이 상품을 구매하기 위한 상품분류 체계가 비교적 계층적으로 잘 나타나 있으며, 상품을 구매하기 위한 기본 링크와 고객의 편의를 위해 만들어진 부가적인 링크가 적절히 표현된 인터넷 쇼핑몰 사이트이다.

사용자는 여러 가지 상품에 대해 상품, 수량, 결제 방법 등을 선택하고 상품에 대한 주문을 할 수 있다. 사용자는 상품의 이미지 링크를 클릭함으로써 상품에 대한 자세한 정보와 상품에 대한 문의, 장바구니 담기, 주문을 할 수 있다. 상품을 주문할 때는 회원구매 또는 비회원구매가 가능하며, 회원구매시에는 반드시 로그인을 해야 하고, 비회원구매시에는 상품 수령지에 대한 자세한 정보를 입력하여야 한다. 상품문의를 통해서 사용자는 상품에 대한 문의를 할 수가 있으며, 장바구니에 상품을 담은 후에 사용자는 쇼핑을 계속 할 수도 있으며, 주문을 할 수도 있다. 결재는 신용카드 또는 온라인 입금이 가능하다.

구조적 테스팅 기법을 적용한 쇼핑몰의 계층적 구조를 분석하기 위해서는 쇼핑몰의 정확한 분석이 필요하다. 동적인 페이지에 대한 표현, 부가적 링크에 대한 표현 등이 정확히 분석되어야지만 분석 모델을 통해 테스팅을 올바르게 수행할 수가 있다. 분석 모델에서 사용자의 선택에 의해 동적으로 생성되는 페이지는 각각의 페이지를 정적인 부분으로 나누어 모델화 하였다. 링크의 정확성을 표현하기 위하여 기본 링크 즉, 쇼핑몰에서 상품을 선택하고 결재하기 위한 계층구조에 기반을 두는 기본 링크 뿐만 아니라 사용자들에게 편의를 제공하기위한 부가적 링크 또한 정확히 표현되어 분석되어야지만 테스팅을 올바르게 수행할 수 있다. 대상 쇼핑몰의 ORD는 다음과 같다.

이 쇼핑몰에서 핵심적인 부분은 상품 클라이언트 페이지는 상품정보 서버 페이지에 상품 정보를 요청하고, 상품정보 서버 페이지는 상품정보 클라이언트 페이지에 그 결과를 응답하는 부분이다. 상품 클라이언트 페이지는 상품문의 클라이언트 페이지와 장바구니 클라이언트 페이지와 navigation관계에 있고, 상품문의 클라이언트 페이지는 상품에 대한 정보를 상품문의 서버 페이지에 요청하게 된다. 상품문의 서버 페이지는 그 정보를 문의결과 클라이언트 페이지에 응답한다. 또한 장바구니 클라이언트 페이지는 장바구니 서버 페이지와 Request 관계에 있고, 장바구니 서버 페이지는 그 결과를 장바구니담기 클라이언트 페이지에 응답하게 된다. 상품정보 클라이언트 페이지와 장바구니담기 클라이언트 페이지는 상품구매 클라이언트 페이지에 연결되어 상품을 구매하게 되는데, 회원일 경우 로그인 클라이언트 페이지로, 비회원일 경우 구매자정보 클라이언트 페이지로 이동하게 된다.



(그림 11) 웹 기반 애플리케이션의 ORD

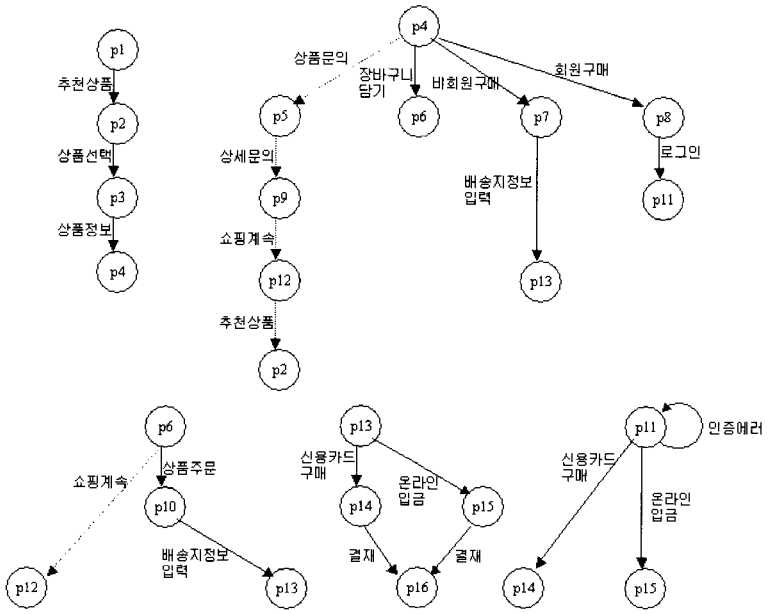
이 사이트의 정보를 ORD와 웹 애플리케이션 모델 분석 알고리즘을 적용하여 OCFG로 표현하면 (그림 12)와 같다.

웹 페이지의 동적인 부분을 표현하기 위해서 사용자의 선택 또는 입력 값에 따라 동적으로 변하는 페이지는 각각의 경우를 정적인 페이지로 생각하여 표현하였다. 예를 들어, 사용자가 상품을 구매하고 결재를 선택할 때 사용자의 선택에 따라 신용카드 또는 온라인 입금을 선택할 수가 있는데, 이와 같은 경우는 사용자의 선택에 따라 각각을 정적인 페이지로 표현하였다. 또한 상품을 선택하고 결재하기 위한 계층적 구조에 기반한 링크가 아닌 상품에 대한 문의 같은 부가적인 링크 또한 일반적인 경우와 같은 링크 표현에 의해 나타내었다.

4.4 테스트 케이스 생성 실험 결과 비교

[10]의 방식으로 생성된 테스트 케이스는 구조적 테스트를 위해 전체적인 경로를 모두 커버할 수 있다는 점에서는 좋은 방식이지만, 동적으로 생성되는 페이지에 대한 입력값만 고려하는 것이 아니라 정적으로 생성되는 페이지에 대한 입력값까지 고려하여 필요 없는 경로의 테스트 케이스가 생성되는 단점과 사용자가 테스트 케이스 생성을 위하여 테스트를 위한 name-value 값을 직접 입력해야 함으로 테스트 케이스가 정확하지 않을 수 있다는 단점이 있다. 또한 테스트 케이스 생성이 쉽지 않고, 테스트 케이스에 중복된 부분이 많아 테스트 케이스 수가 많아지며, 분석하기가 쉽지 않다는 것도 단점이다. 그러나 웹 애플리케이션 모델 분석 알고리즘을 적용한 OCFG를 통해 생성된 테스트 케이스는 각각의 사용사례에 따라 테스트 케이스가 모듈화 되어있어 테스트 케이스 생성이 용이할 뿐만 아니라 동적인 페이지에 대해서는 입력값을 고려하여 테스트 케이스를 생성하지만, 정적인 페이지에 대해서는 입력값을 고려하지 않으므로 필요 없는 경로의 테스트 케이스가 생성되지 않으며, 적은 수의 테스트 케이스만으로 [10]의 방식과 같은 Testability를 보장할 수 있는 장점이 있다. 또한 생성된 테스트 케이스를 분석하기도 쉽기 때문에 구조적 테스트를 위한 테스트에 소요되는 시간이 줄어들 수 있다.

Ricaa & Tonella의 방식과 웹 애플리케이션 모델 분석



(그림 12) 웹 기반 애플리케이션의 OCFG

〈표 8〉 웹 기반 애플리케이션의 테스트 케이스

No	생성된 Test Case	
	Ricca & Tonella의 테스트 케이스	웹 애플리케이션 모델 분석 알고리즘을 적용한 테스트 케이스
1	P1·P2·P3·P4	P1·P2·P3·P4
2	P1·P2·P3·P4·P5·P9·P12·P2	P4·(P5·P9·P12·P2+P6+P7·P13+P8·P11)
3	P1·P2·P3·P4·P6·P12	P6·(P12+P10·P13)
4	P1·P2·P3·P4·P6·P10·P13·(P14+P16)·P16	P13·(P14+P15)·P16
5	P1·P2·P3·P4·P7·P13·(P14+P15)·P16	P11·(P14+P15)
6	P1·P2·P3·P4·P8·P11	
7	P1·P2·P3·P4·P8·P11·(P14+P15)·P16	

알고리즘을 적용한 테스트 케이스는 각각 5가지가 생성되었다. 그러나 [10]의 방식으로 생성된 테스트 케이스에는 <표 6>에서 나타난 것처럼 중복된 부분으로 인해 테스트 케이스가 다소 복잡해지며, 테스트 소요 시간이 길어진다는 단점이 있다.

동적 데이터 접근 정적 웹 사이트에 대한 테스트 케이스 <표 7>에서 Ricaa & Tonella의 방식으로 9개의 표준 경로식 테스트 케이스가 생성된 반면, 웹 애플리케이션 모델 분석 알고리즘을 통해서는 5개의 테스트 케이스가 생성되었다. 테스트 케이스의 수가 적고, 표현 방식이 [10]의 방식보다 간단하므로 구조적 테스트를 위한 테스트에 소요되는 시간이 노력이 줄어든다는 장점이 있다.

웹 기반 애플리케이션의 생성된 테스트 케이스 역시 웹 애플리케이션 모델 분석 알고리즘을 적용하여 생성한 테스트 케이스의 수가 [10]에서 제안한 방식으로 생성된 테스트 케이스의 수보다 적으며, 간결함을 알 수 있다. 테스트에 소요되는 시간을 테스트 케이스를 생성하는 시간과 분석하는 시간으로 생각할 때, 웹 애플리케이션 모델 분석 알고리즘을 적용한 테스트 방식이 시간이 적게 소요됨을 알 수 있다.

<표 9>는 웹 사이트의 분류에 따라서 Ricaa & Tonella의 방식과 웹 애플리케이션 모델 분석 알고리즘을 적용하여 생성한 테스트 케이스의 수와 복잡성을 비교하고 있다. 표에서 나타난 것처럼 테스트 케이스의 수는 웹 애플리케이션 모델 분석 알고리즘을 적용하여 생성한 방식이 적음을 알 수 있다. 그리고 테스트 케이스의 복잡성을 비교하였는데, 복잡성은 다음과 같은 식으로 계산하였다.

$$\text{복잡성} = (V \times T) / 100$$

여기서, V는 테스트를 위한 모델에서 노드 수의 합
T는 생성된 테스트 케이스에서의 노드 수의 합

<표 9>의 결과처럼 웹 페이지의 종류에 따른 복잡성 역시 Ricca & Tonella의 방식보다 웹 애플리케이션 모델 분석 알고리즘을 적용한 방식이 낮음을 알 수 있다. 테스트 케이스의 복잡성이 낮으므로 테스트 케이스를 생성한 뒤, 테스트를 위해 테스트 케이스의 분석에 필요한 시간 또한 단축됨을 보여준다.

5. 결론 및 향후 연구과제

인터넷을 이용한 비즈니스가 성장함에 따라 웹 관련 기술의 발전과 함께 웹 애플리케이션은 점점 더 복잡하게 되었다. 이와 함께 웹 애플리케이션의 품질 보증과 신뢰성에 대한 요구 또한 높아졌다. 그러나 웹의 복잡성으로 웹 애플리케이션을 테스트하기는 쉽지 않은 일이며, 현재까지 웹 애플리케이션의 테스트 연구가 여러 분야에서 진행되고 있다. 현재 웹 테스팅에 사용하고 있는 기법은 기능, DB, 보안, 성능 등의 전통적인 기법을 사용하고 있는 실정이다. 웹 기반 애플리케이션의 품질을 보증하기 위해서 구조적 테스팅 기술을 사용한 실행 가능한 경로를 모두 테스트하는 방법이 필요하다.

이 논문에서는 구조적 테스팅을 위해서 웹 애플리케이션을 ORD를 이용하여 오브젝트 사이의 의존관계를 표현함으로써 웹 애플리케이션의 전체적인 상태를 나타내었다. 또한 웹 애플리케이션 모델 분석 알고리즘을 사용하여 테스트 케이스를 생성하고 그 결과를 기존의 표준 경로식 테스트 케이스 생성 방법인 Ricca & Tonella의 방식과 비교 설명하였다. 그 결과 생성된 테스트 케이스의 수가 [10]의 방식에 비해 적고, 표현이 간결하며, 그 복잡성 역시 낮은 결과를 보였다. 동적인 페이지의 테스트를 위해서 사용자의 입력 또는 선택에 따라 동적으로 생성되는 페이지 각각을 정적인 페이지로 분할하여 모델화 하였고, 각각의 이동 경로를 테스트 할 수 있는 방법을 연구하였다. 기존의 표준 경로식 테스트 케이스 생성 방법은 표준적인 테스트 케이스를 생성한다는 장점이 있으나, name-value 값을 테스터가 직접 입력함으로써 테스트 케이스의 정확성을 확인할 수 없으며, 테스트 케이스가 복잡하고 그 수가 많아진다는 단점이 있다. 이 논문에서 제안한 방법은 구조적 테스트를 위해 모든 이동경로를 커버할 수 있는 테스트 케이스의 생성으로 테스트 케이스의 표현이 간결하고, 테스트 케이스의 수가 적으며, 테스트 케이스의 복잡성이 낮다는 장점이 있다. 테스트 케이스의 복잡성이 낮으므로 테스트 케이스를 분석하는 시간이 적게 걸리므로 구조적 테스트에 소요되는 시간과 노력을 줄어들 수 있었다.

향후 연구 과제로는 생성된 테스트 케이스에 대해 테스트 비용에 대한 부분을 정확히 추정하여 비교할 수 있는 메트릭에 대한 연구와 다양한 오브젝트를 포함한 웹 애플리케이션에 대한 테스트 그리고 다양한 부가적 링크에 대한 모델링과 테스트가 진행되어야 한다.

참 고 문 헌

[1] 강제성, 윤광식, 오승욱, 권용래, "웹의 상태 기반 기능 시험

〈표 9〉 웹 사이트 분류에 따른 테스트 케이스의 수

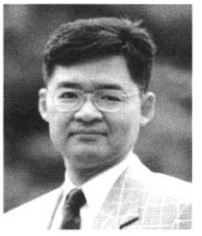
웹 사이트의 종류	Ricca & Tonella의 테스트 케이스		웹 애플리케이션 모델 분석 알고리즘을 적용한 테스트 케이스	
	테스트 케이스 수	복잡도	테스트 케이스 수	복잡도
정적 웹 사이트	1	1.0	1	1.0
폼기반 정적 웹 사이트	5	9.25	5	7.5
동적 데이터 접근 정적 웹 사이트	9	17.36	5	11.47
동적 생성 사이트	6	10.32	4	6.72
웹 기반 애플리케이션	7	8.32	5	4.0

기법", 한국정보과학회 봄 학술발표논문집 Vol.27, No.1, pp. 501-503, 2000.
[2] 권영호, 최은만, "웹 기반 소프트웨어의 테스트 모델에 관한 연구", 정보처리학회 춘계 학술발표논문집 제8권, 제1호, pp. 197-200, 2001.
[3] 김현정 외, "사이버 쇼핑몰의 노드 및 링크 구조에 대한 탐색적 연구", 한국 HCI 98 학술대회, pp.166-172, 1998.
[4] 박은영, 테스트 케이스 작성 방법, Sten Journal, 2003, http://www.sten.or.kr/journal
[5] 이춘우 외, "항해 구조를 이용한 웹 응용의 테스트 방법", 한국정보과학회 가을 학술발표논문집 Vol.30, No.2, pp.361-363, 2003.
[6] 정선미, 최은만, "웹 테스팅 자동화를 위한 테스트 스크립트 생성 방법", 한국정보처리학회 춘계학술발표논문집, Vol.9, No.1, pp.473-476, 2002.
[7] Chien-Hung Liu et al., "Object-based Data flow testing of web applications", Proceedings of First Asia-Pacific Conference on Quality Software, pp.7-16, Oct., 2000.
[8] David C.Kung et al. "An Object-Oriented Web Test Model for Testing Web Applications", Proceedings of the 24th Annual International Computer Software and Applications Conference:, pp.25-27, 2000.
[9] Filippo Ricca and Paolo Tonella, "Understanding and restructuring web sites ith reWeb", Multimedia Magazine, IEEE, Vol.8, No.2, pp.40-51, April-June 2001.
[10] Filippo Ricca, Paolo Tonella, "Analysis and Testing of Web Applications", Proceedings of the 23rd International Conference on Software Engineering, pp. 25-34, 2001.
[11] Giuseppe Antonio Di Lucca et al, "Testing Web Applications", Software Maintenance, Proceedings of International Conference on Software Engineering, pp.3-6, Oct., 2002.
[12] Hung Q. Nguyen, Testing Application on the Web, John Wiley & Sons, 2001.
[13] Jeff Offutt, "Quality attributes of web software applications", IEEE Software: Special Issue on Software Engineering of Internet Software, Vol.19, No.2, pp.25-32, 2002.
[14] Louise Tamres, Testing Web Applications, Addison Wesley, 2002.
[15] Jesper Ryden, Par Svensson, Web application Testing, Department of Transportation and Logistics Chalmers, Master Thesis, 2001.
[16] Sebastian Elbaum et al, "Improving web application testing with user session data", Proceedings of International Conference on Software Engineering, pp.49-59, 2003.
[17] Ye Wu, Jeff Offutt, "Modeling and Testing Web-based Applications", Department of Information and Software Engineering, George Mason University, Technical ISE-TR-02-08, 2002.
[18] Y. Zuev, "A set-covering problem: The combinatorial-local approach and the branch and bound method", U.S.S.R Computational Mathematics and Mathematical Physis, Vol. 19, No.6, 1979.

김 현 수

e-mail : tetis5@korea.com
2003년 동국대학교 컴퓨터공학과(학사)
2005년 동국대학교 대학원 컴퓨터공학전공(석사)
2005년~현재 (주)대우조선 연구소 연구원
관심분야 : 소프트웨어 테스트, 웹 테스트, 설계 방법론

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 컴퓨터멀티미디어공학과 교수
관심분야 : 객체지향 설계, 소프트웨어 테스트, 프로세스와 메트릭, Program Comprehension