분산 객체지향 소프트웨어를 위한 수정 영향 분석

김 경 희† · 박 재 년†† · 윤 용 익†††

요 약

기존의 수정 영향 분석은 단일 환경을 기반으로 하기 때문에 분산 환경에 직접 적용하기 어렵다. 본 논문에서는 분산 환경에서 객체지향 소프트웨어의 수정 영향을 분석한다. 객체지향 소프트웨어의 수정을, 자료, 메소드, 클래스의 집합으로 구분하여 수정 영향을 분석하였으며, 분석 결과를 DPDG(Distributed Program Dependency Graph)에 표현하였다. DPDG는 분산 환경에서의 객체지향 소프트웨어들의 관계를 메소드, 자료요소, 클래스, 설계 문서, 서버 등을 사용하여 그래프에 표시한다. DPDG는 소프트웨어에 수정 발생 시, 재시험하여야하는 소프트웨어 요소를 찾기 위한 그래프이다. 따라서, DPDG를 통해 재시험에 드는 노력을 절약할 수 있다. 본 논문에서는, DPDG를 통해 발견된 절약된 재시험 요소를 방화벽 테이블로 나타내었으며, 이를 구현하여 본 논문에서 설계한 시험 지원도구 VIST(Visual Information Structure Tester)에서 사용하였다. VIST는 절약된 방화벽을 사용하여, 분산 객체지향 소프트웨어 시험에 드는 노력과 비용을 절약하는 도구이다.

Change Impact Analysis for Object-Oriented softwares in the distributed environment

Kyung-Hee Kim† · Jai-Nyun Park†† · Yong-Ik Yoon†††

ABSTRACT

Applying the *change impact analysis* to the distributed environment is not straightforward since it is based on the centralized system environment. In this paper, we investigate the change impact analysis of object-oriented softwares in the distributed environment. We, first, categories the types of changes common in object-oriented software into three sets: data, method, and class level changes. We, then, analyze the impact of each set of changes and represent it in the form of a DPDG. A DPDG is a graph showing relationship of object oriented softwares - with data elements. classes, design documents, servers - in the distributed environment. DPDG searches element of software to retest when the software is changed. Thus, DPDG saves effort of software to retest. In this paper, We propose the table of firewall for retest elements that be discovered by DPDG and implement the table of firewall. The table of firewall is used VIST that we design a software testing tool. The VIST utilizes the minimized firewall, then saves efforts and costs of retesting for distributed object-oriented software.

1. 서 론

기존의 소프트웨어 개발은 소프트웨어 개발 인력이 모두 한 곳에 모여 개발 프로젝트를 진행한다는 가정하에 독립적(stand alone) 방식으로 발전되어 왔다. 그러나, 최근 들어 소프트웨어가 복잡해지고, 그 크기(size)가 방대해짐에 따라 소프트웨어를 개발하기 위해 여러 개발자들이 클라이언트-서버 환경에서 소프트웨어를 개발하고 처리하는 필요성이 부각되고 있다[1].

† 준 회 원 : 숙명여자대학교 대학원 전자계산학과
†† 정 회 원 : 숙명여자대학교 전자계산학과 교수
††† 종신회원 : 숙명여자대학교 전자계산학과 교수
논문접수 : 1999년 1월 11일, 심사완료 : 1999년 3월 16일

또한, 네트워크가 고속의 데이터 통신을 지원하고 있고, 컴퓨터의 성능이 향상되고 있음을 고려할 때, 네트워크로 연결된 여러 대의 컴퓨터에 소프트웨어를 분산처리하기가 더욱 용이하게 되었으며, 이는 사용자에게 보다 편리한 환경을 제공하고 전체적인 성능을 향상시킬 수 있는 이점을 가지고 있다[2].

소프트웨어의 유지보수(maintenance)는 소프트웨어 산출물(software production) 전체 비용의 두 배, 또는 세 배 이상의 비용이 소요된다[3]. 소프트웨어의 유지보수 단계에서는 프로그램 변경사항으로 인한 수정(change)이 빈번하게 발생한다. 이 때, 발생한 수정이 프로그램에서 올바른 영향을 미치는가 하는 것과 그 프로그램의 수정으로 인해 발생되는 문제가 다른 부분들에게 미치는 영향을 찾아야한다[4].

회귀 시험(regression test)은 소프트웨어의 유지보수 단계에서 중요한 역할을 담당하고, 시스템에서 수정이 발생한 이후에 소프트웨어를 재시험(retest)하는 것을 포함한다. 수정은 명세에서 발생하기도 하고, 프로그램 소스 코드에서 발생하기도 한다[5]. 회귀 시험의 목적은 수정된 프로그램이 정상적으로 수행되고 있음을 보이는데 있다.

근래의 소프트웨어는 크기가 방대해지고 적용 범위가 확산되며, 미치는 영향이 극대화되고 있다. 이러한 현실을 고려할 때, 회귀 시험에 드는 시간과 노력을 줄이기 위해서 수정에 의해 영향받는 콤포넌트를 찾아내는 일은 필수적이다[6].

본 논문에서는 분산 환경에서 객체지향 소프트웨어의 분석을 통하여 분산환경 소프트웨어의 수정 영향 분석(Change Impact Analysis) 방법을 제안한다. 또한, 이를 위해 분산 환경에서 각 프로그램들과 데이터 그리고 통신 채널(network channel)과의 관계를 표현하는 DPDG(Distributed Program Dependency Graph)를 제안한다. DPDG는 기존의 MDG(Member Dependency Graph)를 분산환경에 적용하여 확장한 그래프로서, 각 소프트웨어 요소들의 관계를 보이고 분산환경에서의 수정 영향 범위를 나타낸다. 따라서, DPDG는 소프트웨어의 변경 사항으로 인한 수정이 발생하였을 때, 재시험하여야 하는 소프트웨어 요소를 찾고, 해당 요소를 그래프에 표시함으로써, 불필요한 소프트웨어 요소의 재시험을 방지한다. 본 논문에서는 DPDG를 통해 발견된 재시험이 요구되는 소프트웨어 요소를 테이블에 표시하고, 이를 구현하였으며, 이를 설계한 시험 지원도구 VIST에서 사용하였다. VIST는 수정 영향 분석 방법을 적용하여, 분산 객체지향 소프트웨어 시험에 드는 노력과 비용을 절약하는 도구이다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어, 제 2장에서는 관련 연구로 기존의 수정 영향 분석 방법을 소개한다. 제 3장에서는 분산 환경의 소프트웨어 분석을 통해 각 관계별로 영향범위를 설정하는 방법을 제안하고 본 논문에서 제안한 DPDG를 소개한다. 제 4장에서는 제안한 수정 영향 분석을 적용하여 설계한 시험 지원 도구인 VIST(Visual Information Structure Tester)를 보이고, 제 5장에서 기존의 수정 영향 분석 방법과 본 논문에서 제안한 방법을 비교한다. 마지막으로 제 6장에서 결론과 향후 연구 과제를 언급한다.

2. 연구 배경

수정 영향이란 회귀 시험의 주요한 부분으로, 소프트웨어 요소(elements)의 수정이 다른 부분에 영향을 미치는 것을 일컫는다. 여기에서 소프트웨어 요소는 소프트웨어의 한 단위(a unit of software)를 의미하며, 이는 프로시듀어(procedure), 루틴(routine), 자료(data), 함수(function)나 패키지, 또는 소프트웨어 시스템의 인터페이스 등을 의미한다. 회귀 시험을 수행할 때, 재시험(retest)하는 시간과 노력을 절약하기 위해서 수정된 소프트웨어의 영향범위를 파악하고 분석하는 작업이 필수적이다. 수정 영향 분석은 크게 정적 수정 영향 분석(static impact analysis)과 동적 수정 영향 분석(dynamic impact analysis)의 두 가지로 구분된다. 정적 수정 영향 분석은 프로그램(code)의 수행 없이 정적 코드 구조(static code structure)를 분석하는 방법이고, 동적 수정 영향 분석은 수정을 결정하기 위해 소프트웨어를 수행(executes)하고 분석하는 방법이다[7]. 본 논문에서는 정적 수정 영향 분석 방법을 사용하였다.

수정 영향을 확인하기 위해 시험자는 소프트웨어 요소들 사이의 관계의존도(dependency relationship)에 관한 정보를 보유하여야 한다. 의존은 제어 의존(control dependency)과 자료 의존(data dependency), 문서 의존(document dependency)으로 구분된다. 제어 의존도는 프로그램이 실행될 때, 소프트웨어 요소들 간의 의존 관계를 뜻하고, 자료 의존도는 자료를 사용하거나 정의하는 소프트웨어 요소들 사이의 의존도를 뜻한

다. 또한, 문서의존은 설계문서가 메소드와 객체, 클래스 등을 명세함을 의미한다. 일반적으로 프로그램들은 영향 분석을 위해 그래프를 이용하여 의존도를 표시한다. 따라서, 각 소프트웨어 요소들 사이의 의존도를 자료흐름 의존도(data-flow dependency)를 이용하여 보일 수 있다. 객체지향 소프트웨어의 그래프 표현은 Harrold와 Soffa에 의해 회귀 시험에서 사용되면서 발전되었다[8]. 이 그래프 표현은 단위 회귀 시험(unit regression testing)과 통합 시험(integration regression testing)을 위해 자료 흐름 분석(data-flow analysis)에서 점차적으로 사용되면서 자료 흐름 시험(data-flow testing)과 함께 사용되었다[9]. 또한, Malloy는 제어흐름과 자료흐름을 포함한 프로그램 의존도(Program Dependency Graph)의 개념을 기초로 하여 OPDG (Object Program Dependency Graph)를 제안하였다[10]. 이는 기존의 PDG(Program Dependency Graph)에 객체지향 요소를 포함시킨 개념으로 클래스, 상속, 다형성, 동적 연결과 같은 객체지향 속성을 여러 단계에 포함한 모형이다.

여러 방법이 객체지향 소프트웨어의 수정 영향 분석에 제안되었으며, White와 Leung은 1992년에 모듈 수정(module modification)에 근거한 영향모듈(affected module)의 집합을 방화벽(firewall)이라는 개념으로 소개하였다[9]. 방화벽은 소프트웨어 요소의 수정으로 인해 영향받는 소프트웨어 요소들의 집합을 뜻한다. 1998년에 Jang은 수정 영향을 객체지향 개념에 도입하여 수정된 MDG(Member Dependency Graph)를 제안한다[4]. MDG는 영향받은 부분을 정의하기 위해 메소드들 사이의 의존 관계를 표현하는 메소드 방화벽기법이다. Jang의 방화벽은 시험단위(test unit)를 메소드로 하며, 자료 요소(data member) 수정과 메소드의 수정 그리고 클래스 수정, 상속관계 수정을 반영한다.

본 논문에서는 기존의 MDG를 기본으로, 분산환경 객체지향 소프트웨어의 수정 영향 분석에 적합한 분산 객체지향 소프트웨어 관계 그래프 DPDG를 제안한다. 본 논문에서 채택한 방법은 프로그램의 수행 없이 프로그램의 소스를 분석하여 수정 영향을 파악하는 정적 수정 영향분석 방법이고, 이를 분산환경의 객체지향 소프트웨어에 적용한다.

3. 제안한 수정 영향 분석

기존의 수정 영향 분석은 단일 환경(centralized system environment)에서 수정 영향을 분석하는 것으로 분산환경에 적용이 어렵다. 본 연구에서는 근래의 동향과 필요에 의해 분산환경에서의 수정 영향을 분석한다. 이 장에서는 분산 객체지향 소프트웨어에서 각 요소들의 관계를 경우에 따라 여러 유형별로 분류하고, 이를 DPDG에 나타낸다. DPDG는 콤포넌트와 자료 및 설계문서와 서버간의 관계를 보이며 수정 영향 범위를 표시한다. 본 논문에서는 DPDG가 사이클(cycle)을 갖지 않는 그래프라고 가정한다.

3.1 관계 분석에 따른 수정 영향 모델

이 장에서는 각 요소들(members) 즉, 객체, 메소드, 자료 및 통신 채널의 관계를 DPDG에 보이는 방법을 설명한다. 분산 객체지향 소프트웨어에서 수정이 발생하는 경우는 다음과 같다.

- 경우 1 : 메소드의 수정
- 경우 2 : 클래스의 수정
- 경우 3 : 자료요소의 수정
- 경우 4 : 설계문서의 수정
- 경우 5 : 통신 채널의 수정

경우 3에서 자료 요소는 데이터 테이블을 의미하고, 경우 4의 설계문서는 소프트웨어를 설계할 때 작성되는 "운용자 지침서"나 "사용자 지침서"와 같은 소프트웨어 개발에 지침이 되는 설계문서를 나타낸다. 경우 5에서, 통신 채널의 수정은 서버가 삭제, 추가되거나 각 요소의 위치가 다른 서버로 바뀌는 경우를 말한다.

본 논문에서는 분산 객체지향 소프트웨어의 객체와 자료요소 및 통신 채널의 관계를 표현하기 위한 DPDG를 제안한다. DPDG는 Jang의 수정된 MDG(Member Dependency Graph)[4]를 분산환경에 적용하여 확장시킨 것으로 메소드들 간의 관계와 자료요소, 그리고 서버와 설계문서들의 관계를 포함한다. 다음은 DPDG를 정의한 것이다.

정의 1. DPDG는 유방향 그래프 G = (V, E)로 표현한다. 여기에서, $V = \{V_1, \ldots, V_n\}$는 노드들의 유한 집합이고, $E = \{E_1, \ldots, E_n\}$는 변(edge)들의 유한 집합이며, $E \subseteq V \times V$ 이다.

정의 2. 분산 객체지향 프로그램 P를 위한 유방향 그래프 DPDG = (V, E)에서 V는 P의 객체 클래스를 나타내는 노드들의 집합으로 $V = V_C \cup V_M \cup V_D \cup V_{ob} \cup V_O$이다. 이 때, V_C는 객체 클래스의 집합이고, V_M은 메소드의 집합이며, V_D는 자료요소의 집합이다. 그리고, V_O는 설계문서의 집합이다.

정의 3. 분산 객체지향 프로그램 P를 위한 유방향 그래프 DPDG = (V, E)에서 E는 노드들의 관계를 보이는 변들의 집합으로 $E = E_I \cup E_C \cup E_D \cup E_O$이다.

정의 3.1 $E_I \subseteq E_{IC} \cup E_{IM} \cup E_{ID} \cup E_{IO}$는 각 노드들 사이의 상속 관계를 나타낸다. 다음은 각 노드들의 정의이다.

- $E_{IC} \subseteq V_C \times V_C$는 클래스간의 상속을 표현한다.
- $E_{IM} \subseteq V_C \times (V_M \cup V_D)$는 메소드와 자료요소를 상속하는 것을 표현한다.
- $E_{ID} \subseteq (V_C \times V_D) \times (V_C \times V_D)$는 클래스와 데이터의 상속을 표현한다.
- $E_{IO} \subseteq (V_C \times V_O) \times (V_M \times V_O)$는 서버에 존재하는 각 설계문서와 클래스의 상속을 표현한다.

여기에서, 설계문서는 서버에 속한 모든 요소들이 상속받기 때문에, DPDG에서 E_{IO}는 표시하지 않는다.

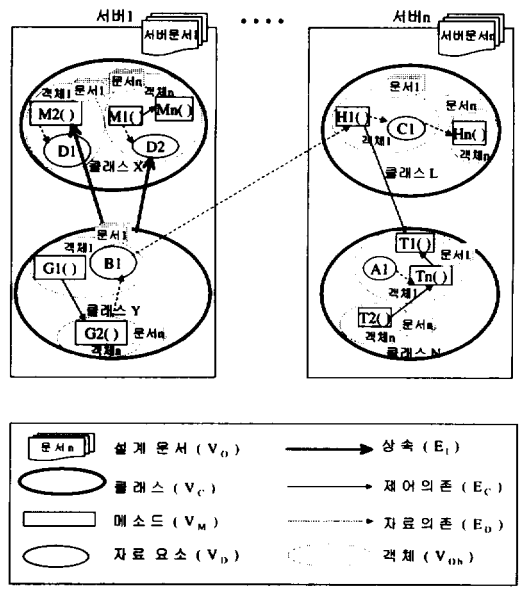
정의 3.2 $E_C \subseteq V_M \times V_M$은 메소드 사이에서 제어의존(control dependency)을 나타낸다.

정의 3.3 $E_D \subseteq (V_M \cup V_D) \times (V_M \cup V_D)$는 메소드와 자료요소들 사이의 관계를 나타낸다.

정의 3.4 $E_O \subseteq (V_O \times (V_M \cup V_D)) \times (V_O \times (V_M \cup V_D))$는 설계문서와 메소드 그리고 설계문서와 자료요소들의 관계를 나타낸다. 각 서버는 자신의 설계문서를 갖고 서버에 존재하는 모든 메소드와 자료요소 및 클래스는 자신의 서버에 위치한 설계문서를 상속받는다. 따라서, DPDG에 E_O의 표시는 생략하였다.

(그림 1)은 앞에서 언급된 DPDG를 보인 것이다.

DPDG는 여러 개의 서버가 존재하고 각 서버는 각기 자신이 소유하는 설계문서와 클래스를 갖는다. (그림 1)에서 서버1은 클래스 X와 클래스 Y를 갖으며, 하위 클래스(subclass)인 클래스 Y가 상위 클래스(superclass)인 클래스 X를 상속받는다. 클래스 X에서 M1과 M2 그리고 Mn은 메소드를 나타내고, D1과 D2는 자료요소를 나타낸다. 메소드 M1은 D2를 사용하는 메소드이고, M2는 D1을 사용하는 메소드이다. 클래스 Y는 클래스 X로부터 M1과 D2를 상속받는다. 클래스 X에서 M1은 Mn을 호출(invoke)하고, 클래스 Y에서 G1은 G2를 호출한다. 서버n의 클래스 L에서 메소드 H1은 서버1의 클래스 Y에 위치한 자료요소 B1을 읽기 수행한다. 또한, 서버1에서 클래스 Y의 메소드 G2는 자료요소 B1에 쓰기 수행한다.



(그림 1) DPDG 구성

3.2 분산 환경에서의 수정 영향분석

기존의 수정 영향분석은 단일 환경에서 이루어졌고, 클래스들의 수정에 영향받는 메소드와 자료요소를 살피고 이를 그래프로 표현하는 방법을 사용하였다. 그러나 분산환경에는 여러 개의 서버와 클라이언트가 존재한다. 따라서, 각 클래스와 메소드 및 자료요소의 위치를 분산환경에서 서버와 클라이언트에 의해 나타내고 관리하여야 한다.

분산 객체지향 소프트웨어에서 수정이 발생하는 경우는 메소드, 클래스, 자료요소, 설계문서 그리고 채널

의 경우이다. 이러한 수정들은 각기 다른 수정 영향 범위를 갖는다.

본 논문에서는 분산 객체지향 소프트웨어의 수정 영향을 분석하기 위해 소프트웨어의 요소들 즉, 메소드와 자료, 또는 설계문서 등을 경우에 따라 구분하여 수정 영향을 분석하고 이를 방화벽 F에 적용하며, DPDG에 표현한다. 다음은 방화벽 F를 위한 용어 정의이다.

- 수정이 발생한 하나의 메소드는 하나의 M(i)이다.
- 수정된 메소드 M(i)를 호출하는 메소드는 call(M(i))이다.
- 수정된 메소드 M(i)가 호출하는 메소드는 called(M(i))이다.
- 수정된 메소드 M(i)에 의해 영향받은 설계문서는 docu(M(i))이다.
- 수정된 메소드 M(i)에 의해 영향받은 자료요소는 data(M(i))이다.
- 수정된 자료요소 D(i)에 의해 영향받는 메소드는 metho(D(i))이다.
- 수정된 자료요소 D(i)에 의해 영향받는 설계문서는 docu(D(i))이다.

본 논문에서는 DPDG를 사용하여 분산환경에서의 소프트웨어 방화벽을 표현하며, 이를 위해 수정이 발생한 경우를 구분하여 수정 영향을 분석한다. 다음은 경우에 따른 수정 영향 범위를 분석하고 방화벽을 보인 것이다.

(1) 메소드의 수정

객체지향 소프트웨어에서 메소드의 수정은 메소드의 삽입과 삭제, 메소드 명(name)의 변경, 메소드 인수(argument) 변경 등 여러 경우가 있다. 다음은 경우들을 분류하기 위한 정의이다.

- 메소드의 이름과 인수, 그리고 리턴값(return value)의 형태를 메소드의 "인터페이스(interface)"라 한다.
- 구현된 프로그램에서 메소드의 구현 부분을 메소드의 "구현(implementation)"이라 한다.

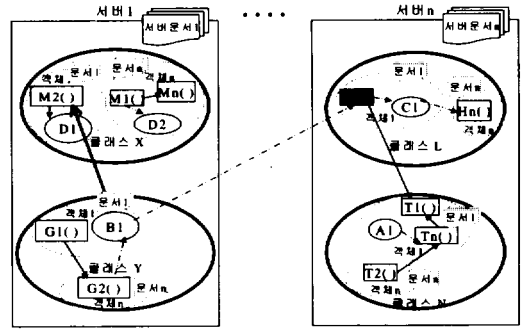
① 인터페이스의 수정

메소드의 인터페이스가 수정된 경우, 수정된 메소드를 호출(call)하는 다른 메소드는 모두 수정되어야 한다. 따라서, 수정이 정확한가를 재시험하고 이를 방화벽에 표현하여야 한다. 이를 식으로 표현하면 다음과 같다.

$$F = F \cup M(i) \cup call(M(i)) \cup docu(M(i)) \quad <식1>$$

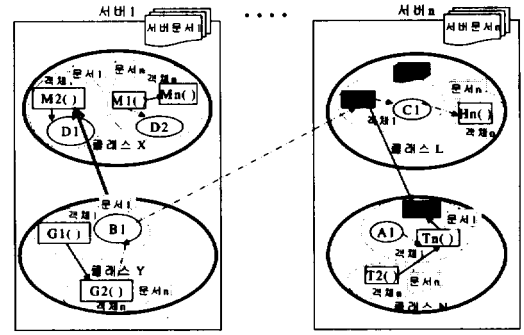
앞의 식은 메소드 M(i)가 수정되었을 때, M(i)를 호출한 메소드 call(M(i))와 M(i)의 수정에 의해 영향받는 설계문서 docu(M(i))를 방화벽 F에 포함하여 보인 것이다.

(그림 2)는 수정으로 인한 방화벽의 변환을 보인 그림이다. DPDG 그래프에서 회색으로 표시된 메소드나 자료요소, 설계문서 등은 재시험이 요구되는 방화벽을 보인 것이다. (그림 2)에서, 서버n의 H1 메소드에 수정이 발생하여 메소드 H1이 방화벽에 포함되었다.



(그림 2) 메소드 H1의 인터페이스 수정 발생

다음으로 (그림 3)은 메소드 H1의 수정으로 다른 요소들이 영향받아 방화벽이 확산되었음을 보인다. (그림 3)에서 회색으로 보이는 소프트웨어 요소는 방화벽에 포함된 요소들을 나타낸 것이다. 메소드 T1은 메소드 H1을 호출하여 사용하는 메소드로서, 메소드 H1의 수정과 더불어 재시험이 요구된다. 또한, 자료요소 C1은 메소드 H1을 호출하는 자료요소로서 H1을 기술하는 "운영자 지침서" 등을 나타낸다. 따라서, H1의 수정으로 인해 C1은 영향받아 재시험이 요구된다.



(그림 3) 메소드의 인터페이스 수정 영향으로 인한 방화벽 변화

② 구현의 수정

메소드의 프로그램 구현부분이 수정되면, 수정된 메소드가 사용하는 자료요소와 메소드 그리고 해당 부분의 설계문서에 수정이 발생할 수 있다. 구현부분의 수정은 경우에 따라 영향이 다르다. 다음은 서버n의 메소드 H1에 수정이 발생하였을 때, 경우에 따른 방화벽을 위한 식이다.

- **경우 1** : 자료요소를 사용하는 메소드에서의 수정

  $F = F \cup M(i) \cup docu(M(i)) \cup data(M(i))$ <식2>

- **경우 2** : 다른 메소드와 상호 작용하는 메소드의 수정

  경우 2의 방화벽 F는 <식1>과 같다.

- **경우 3** : 경우 1과 경우 2를 제외한 메소드의 수정

  $F = F \cup M(i) \cup docu(M(i))$ <식3>

③ 메소드의 삽입

메소드의 수정에서 메소드 M(i)가 삽입되는 경우를 고려할 수 있다. 메소드의 삽입은 다른 메소드에서 삽입되는 메소드를 사용하거나, 삽입되는 메소드가 다른 메소드를 호출하는 경우 등을 고려할 수 있다. 다음은 삽입되는 메소드 M(i)가 경우에 따라 다른 수정 영향을 갖음을 보이고, 방화벽 F를 보인 것인다.

- **경우 1** : 삽입되는 메소드가 다른 메소드를 호출

  $F = F \cup M(i) \cup called(M(i)) \cup docu(M(i)) \cup docu(called(M(i)))$ <식4>

경우 1에서는 삽입된 메소드 M(i)와 메소드 M(i)가 호출하는 메소드 M(j)를 방화벽에 포함한다.

- **경우 2** : 삽입되는 메소드가 자료요소를 사용

  이 경우는 삽입된 메소드 M(i)가 자료요소를 사용하는 경우이다. 경우 2를 방화벽을 이용하여 표현하면 <식2>와 같다.

④ 메소드의 삭제

메소드가 삭제되는 경우는 설계문서에서 삭제된 메소드를 수정하고, 삭제된 메소드를 호출한 메소드를 수정하여야 한다. 이를 반영하는 방화벽 F는 다음과 같다.

$F = F \cup call(M(i)) \cup docu(M(i))$ <식5>

(2) 클래스의 수정

객체지향 소프트웨어에서 클래스의 수정은 클래스의 삽입과 삭제, 클래스 명(name)의 변경 그리고 클래스들 간의 관계(relationship) 변경 등이 있다. 그러나, 본 논문에서는 클래스 삽입과 삭제만을 다룬다. 다음은 클래스의 삽입과 삭제로 인한 영향을 DPDG에 나타내고 방화벽을 보인 것이다.

① 클래스의 삽입

클래스가 삽입되는 경우는 기존의 서버에 새로운 메소드와 자료요소가 추가되는 것을 의미한다. 이 때, 추가되는 메소드와 자료요소가 기존에 존재하는 요소들인 경우, 삽입된 클래스에서 기존의 메소드와 자료요소를 재정의(redefine)하여야 한다. 삽입되는 클래스에 속한 메소드와 자료요소가 갖는 관계에 따라 수정 영향 범위가 달라진다.

다음은 삽입되는 클래스에 속한 메소드와 자료요소의 관계에 따른 수정 영향과 방화벽을 식으로 보인 것이다.

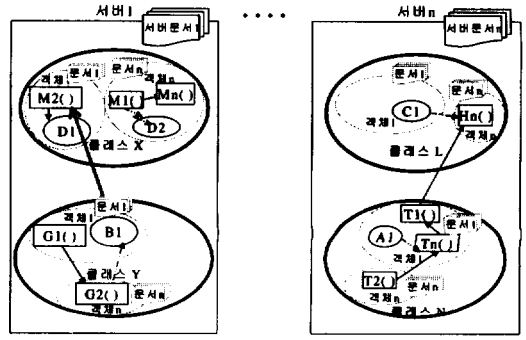
- **경우 1** : 삽입된 클래스에 속한 메소드가 다른 클래스의 메소드를 호출

  경우 1의 수정에 의해 영향받은 범위를 방화벽으로 표현하면 <식1>과 같다.

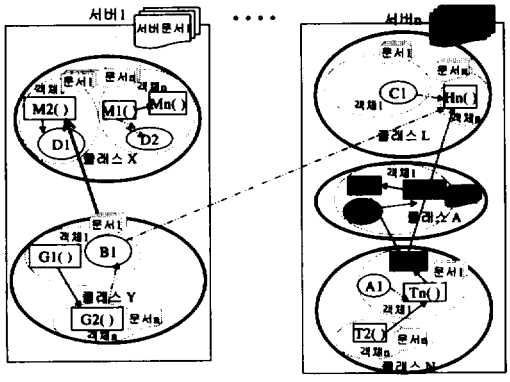
- **경우 2** : 삽입된 클래스에 속한 메소드가 다른 클래스의 자료요소에 쓰기 수행

  이 경우에는 <식2>의 방화벽을 적용하여 수정 영향을 표현 할 수 있다.

다음의 (그림 4)와 (그림 5)는 클래스 삽입으로 인한 방화벽의 변화를 DPDG에 보인 것이다. (그림 4)는 클래스 삽입 전의 DPDG를 보인 것이고, (그림 5)는 서버n에 "클래스 A"가 삽입된 후의 DPDG를 보인다.



(그림 4) 클래스 삽입 전의 DPDG



(그림 5) "클래스 A"의 삽입으로 인한 DPDG의 수정

② 클래스의 삭제

클래스의 삭제는, 클래스의 삽입에서와 같이, 삭제되는 클래스에 속한 메소드와 자료요소의 삭제를 의미한다. 따라서, 삭제된 클래스에 속한 메소드와 자료요소와 관련된 모든 메소드, 자료요소를 수정하여야 한다. 또한, 삭제된 클래스와 상속관계에 있었던 다른 클래스에 속한 메소드와 자료요소도 수정되어야 한다. 이를 경우에 따라 구분하고 방화벽을 이용하여 표현하면 다음과 같다.

- **경우 1** : 삭제된 클래스의 메소드가 삭제

  경우 1의 방화벽은 앞의 메소드의 삭제에서 언급된 <식5>와 같다.

- **경우 2** : 삭제된 클래스의 자료요소가 삭제

$$F = F \cup metho(D(i)) \cup docu(D(i)) \quad <식6>$$

경우 2에서는 삭제된 자료요소를 읽거(read)나 쓰기(write)한 모든 메소드를 수정하여야 한다.

(3) 자료요소의 수정

자료요소의 수정은 자료요소의 삽입과 삭제, 수정이 있다. 자료요소의 수정은 자료의 종류(type), 길이(size), 기본키(primary key)의 수정 등이 있다. 다음은 경우에 따른 자료요소의 수정을 보이고 방화벽을 보인 것이다.

① 자료요소의 삽입

자료요소의 삽입은 새로 생성된 클래스의 자료요소로 삽입되는 경우와 기존의 메소드에서 사용하기 위해 자료를 요청하는 경우가 있다. 다음은 자료요소 D(i)가 삽입되었을 때, 경우별로 구분하여 방화벽을 보인 것이다.

$$F = F \cup data(D(i)) \cup docu(D(i)) \cup metho(D(i)) \quad <식7>$$

② 자료요소의 삭제

자료요소가 삭제된 경우는, 그 자료요소를 사용하던 메소드와 설계문서에 영향을 준다. 이를 방화벽으로 나타내면 <식7>과 동일하다.

③ 자료요소의 수정

자료요소의 수정은 앞에서 언급한 바와 같이 종류, 크기, 기본키의 수정 등이 있다. 이는 수정된 자료요소를 사용하는 메소드에 영향을 끼치며 따라서 메소드를 수정하여야 한다. 이를 함수 식으로 표현하면 <식7>과 같다.

(4) 방화벽 구성 알고리즘

소프트웨어 수정으로 인한 수정 영향 범위에 포함된 방화벽을 통하여, 재시험할 소프트웨어 요소를 찾아 시험을 수행하는 것은, 모든 범위를 재시험 범위로 하여 시험하는 기존의 시험 방법과 비교할 때, 재시험에 드는 노력과 비용이 절약된다. 다음은 소프트웨어 요소에 수정이 발생하였을 때, 수정이 발생한 프로그램을 위한 방화벽을 찾고, 방화벽에 포함된 소프트웨어 요소를 재시험하기 위한 알고리즘이다.

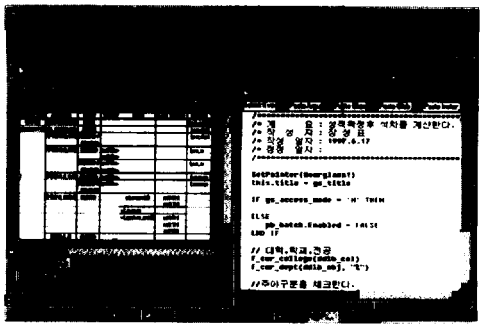
1. 영향받지 않은 소프트웨어 요소들로 구성된 DPDG를 작성한다.
2. 방화벽 F를 초기화한다.
3. 수정이 발생한 소프트웨어 요소가 발생하면, 영향 범위를 DPDG에 표시하고 방화벽을 구한다.
4. 구한 방화벽을 방화벽 F에 더한다.
5. 얻어진 방화벽에 포함된 소프트웨어 요소들의 수정 영향 범위를 DPDG에 표시하고, 방화벽을 구한다.
6. 5에서 구한 방화벽이 공집합(empty set)이 아니면, 구한 방화벽을 방화벽 F에 더한다.
7. 구한 방화벽이 공집합일 때까지, 과정 5와 6을 반복한다.
8. 최종적으로 구한 방화벽 F에 포함된 소프트웨어 요소들을 재시험한다.

4. VIST의 사용자 인터페이스 설계

본 논문에서는 제안한 수정 영향 분석을 적용한 시험 지원 도구를 설계하였다. 설계한 도구 VIST는 클라

이언트-서버환경에서 소프트웨어를 유지 보수할 때, 프로그램 수정으로 인해 재시험이 요구되는 소프트웨어 요소를 보이고, 프로그램 개발자가 직접 시험을 수행하면서 소프트웨어를 수정할 수 있는 환경을 지원한다.

(그림 6)은 VIST를 통하여 시험을 수행하는 화면이다. VIST는 수정이 발생한 프로그램 명을 입력받고 해당 프로그램의 방화벽 테이블을 찾아 화면에 보인다. (그림 6)의 좌측 화면은, 입력된 프로그램의 방화벽 테이블을 보인 것이다. 좌측의 방화벽 테이블에서 재시험할 소프트웨어 요소를 선택하여 클릭하면, 우측 화면에 해당 소프트웨어 요소의 내용이 보인다. 따라서, 시험자는 VIST를 통해 재시험할 요소를 직접 확인하여 수정할 수 있다.



(그림 6) VIST의 시험 작업 화면

VIST는 클라이언트-서버 환경을 기반으로 한 소프트웨어 시험 도구이며, 클라이언트와 서버로 구분된다. VIST는 프로그램 개발자가 직접 시험을 시행할 수 있도록 하였다. 따라서, 프로그램 개발자는 소프트웨어를 시험하기 위해 클라이언트 VIST에 접근한다. 서버 VIST는 클라이언트 VIST로부터 시험하고자 하는 소프트웨어를 요청 받고 응답하며, 시험자는 클라이언트 VIST를 통하여 (그림 6)과 같은 화면을 제공받는다. 화면은 시험하고자하는 소프트웨어의 프로그램 구현 내용(source)과 그 영향범위에 속한 방화벽을 <표 1>과 같은 형식으로 보인다. 시험자는 소프트웨어를 직접 시험하고 수정할 수 있으며, 수정이 발생하였을 때, <표 1>에 제공된 방화벽을 통하여 순서대로 원하는 소프트웨어 요소를 재시험한다.

<표 1>에서 "프로그램명"은 수정이 발생한 소프트웨어이고 "Object ID", "Event", "Table" 등은 수정 영향에 포함된 방화벽을 보인 것이다. 이 때, 방화벽의 소프트웨어 요소들은 자신이 속한 서버와 함께 명시하였고, 수정이 발생한 "프로그램명"의 소프트웨어가 속한 서버와 같은 서버에 속한 요소들의 서버 명은 명시하지 않았다. 또한, 재시험이 수행된 소프트웨어 요소를 표시하여, 재시험이 수행되지 않은 소프트웨어 요소와 구분하였다. 따라서, 수정 영향 범위에 포함된 소프트웨어 요소들을 중복 없이 용이하게 재시험 할 수 있다.

〈표 1〉 DPDG의 방화벽 테이블

프로그램명	OBJECT ID	EVENT	TABLE명	함수명	함수의 TABLE명	공통함수
석차계산 (SUH100PB) (server5)	M-WINDOWS (W_SUH100PB)	Open				f_cur_college f_cur_dept
		Close				
	대학 (DDLB_COL)	Selection - Changed				f_cur_dept
	학과 (DDLB_SBJ)	Modified	sys012vw(server1) sys017vw			f_cur_major
		Selection -Changed	sys012vw(server1) sys017vw			f_cur_major
	전공 (DDLB_MJR)	Modified	sys017vw			f_message
		Selection -Changed	sys017vw			f_message
	실행 (PB_BATCH)	Clicked		wf_smt_rank()	suh030tl sud010tl(server2)	
				wf_summary_rank() wf_graduate_rank()	suh030tl sud010tl(server2) sui010tl	
	종료 (PB_3)	Clicked				

5. 비교 및 분석

본 논문에서는 분산 객체지향 소프트웨어를 위한 수정 영향을 분석하였다. 이 장에서는 Jang의 수정된 MDG[4]와 본 논문에서 제시한 DPDG를 비교하였고, <표 1>에서 보인 실 사례를 적용하여 방화벽을 적용하지 않았을 경우와 제안한 방화벽을 적용하였을 경우를 비교하였다.

5.1 MDG와 비교

다음은 MDG와 DPDG를 비교한 표이다.

〈표 2〉 수정된 MDG와 DPDG의 비교

방법 / 비교 대상	수정된 MDG	DPDG
구 성 요 소	클래스(class) 메소드(method) 자료(data)	클래스(class) 메소드(method) 자료(data) 서버(server) 설계문서(document)
기 반 환 경	단일 환경(centralized system environment)	분산환경(distributed system environment)

DPDG는 분산 환경을 기반으로 하여 객체지향 소프트웨어의 관계를 보인 그래프이다. 따라서, 그래프에 서버를 포함하였다. 또한, 시스템 설계 단계에서 필수적으로 작성되고 소프트웨어 유지보수 단계에서 수정의 지침이 되는 설계문서를 포함하여, 소프트웨어 수정 발생 시 필수적으로 요구되는 설계문서의 수정을 고려하였다.

5.2 사례 연구를 통한 분석

제안한 수정 영향 분석 방법의 효율성을 평가하기 위해, S학원에서 사용하고 있는 학생 관리 시스템을 사용하여 분석하였다. 학생 관리 시스템은 PowerBuilder 5.0으로 개발되었다. 이 시스템에서, 서버의 DBMS (Data Base Management System)는 Oracle 7.3.2.2.0이고, 클라이언트용 게이트웨이는 SQL*Net 2.3.2.1.0이다. 앞에서 보인 <표 1>은 학생관리 시스템에서, '서버 5'에 위치한 석차계산 프로그램(SUH 100PB)을 수정하였을 때, DPDG에서 발견된 방화벽의 범위를 보인 것이다. 본 장에서는 <표 1>의 경우를 사례로 하여, DPDG를 적용한 방화벽과 수정 영향 분석을 수행하지 않은 방화벽을 비교한다. 다음의 <표 3>은 서버 5에 속한 석차계산 프로그램에 수정이 발생하였을 때, DPDG를 적용하여 찾은 방화벽에 포함된 객체와 메소드를 보인다. <표 4>는 수정영향 분석을 수행하지 않은 방화벽에 포함된 객체와 메소드를 보인다.

〈표 3〉 DPDG의 방화벽 테이블

프로그램명	객체명	메소드명
석 차 계 산 (SUH100PB) (server5)	M-WINDOWS (W_SUH100PB)	Open Close
	대학 (DDLB_COL)	Selection - Changed
	학과 (DDLB_SBJ)	Modified Selection -Changed
	전공 (DDLB_MJR)	Modified Selection -Changed
	실행 (PB_BATCH)	Clicked
	종료 (PB_3)	Clicked

〈표 4〉 수정 영향 분석을 적용하지 않은 방화벽

프로그램명	객체명	메소드명
석 차 계 산 (SUH100PB) (server5)	M-WINDOWS (W_SUH100PB)	Open Close
	주간(st_4)	Constructor Clicked외 8개
	학기석차(rb_smt)	Clicked외 9개
	누계석차(rb_summary)	Clicked외 9개
	졸업석차(rb_grauate)	Clicked외 9개
	야간(st_5)	Clicked외 9개
	년도(em-year)	Clicked외 9개
	학기(ddlb_smt)	Clicked외 9개
	대학 (ddlb_col)	Selection - Changed
	학과 (ddlb_sbj)	Modified, Selection -Changed
	전공 (ddlb_mjr)	Modified, Selection -Changed
	실행(pb_batch)	Clicked
	종료(pb_3)	Clicked

다음은 <표 3>과 <표 4>에서 보인 소프트웨어 요소의 개수를 비교하여 보인 것이다.

비 교	영향받은 객체 개수	영향받은 메소드 개수
DPDG를 적용한 방화벽	6	9
수정영향분석이 안된 방화벽	13	14

DPDG를 적용하였을 때, 재시험하여야 하는 소프트웨어 요소가 절약되었음을 알 수 있다.

6. 결과 및 향후 연구과제

최근 들어, 네트워크와 분산 시스템이 시스템 개발에 주류를 이루고 있고, 소프트웨어의 크기가 방대해지며, 소프트웨어의 유지 보수가 개발비용에서 차지하는 비중이 증가되고 있다. 따라서, 분산 환경에서 객체지향 소프트웨어 수정이 미치는 영향을 분석하고 영향범위에 속한 요소만을 재시험하는 일과, 이를 적용한 시험 도구가 요구된다.

본 논문은 분산환경에서 수행되는 객체지향 소프트웨어의 구성 요소에 수정이 발생하였을 때, 그 수정이 다른 메소드나 자료요소, 설계문서 등에 미치는 영향을 분석하여 재시험 되어야 하는 메소드나 자료요소, 설계문서 등을 찾는 방법을 제안하고, 이를 방화벽으로 정의하였으며, 방화벽을 그래프로 나타내기 위해 DPDG를 제안하였다. 또한, DPDG를 방화벽 테이블로 구현하였고, 이를 적용한 시험 지원 도구 VIST를 설계하였다.

제안한 DPDG는 소프트웨어의 수정이 다른 소프트웨어 요소에 미치는 영향을 그래프를 통해 보이고, 재시험되어야 하는 소프트웨어 요소들을 찾는다. 또한, DPDG는 본 논문에서 설계한 시험 지원 도구 VIST에서 방화벽을 보이는 테이블로 구현되어, 재시험되어야 하는 최소화된 소프트웨어 요소들 즉, 메소드, 자료요소, 클래스, 설계문서 등을 보이고, 시험자가 영향 범위에 속한 최소화된 소프트웨어 요소들을 수정하도록 연결시키는 연결 테이블(mapping table)로 작성된다. VIST는, 유지 보수과정에서 소프트웨어 개발자가 분산환경 객체지향 소프트웨어를 시험하고, 수정할 수 있는 환경을 제공하는 시험 지원 도구이다. 기존의 시험 도구는 방화벽을 적용하지 않았고, 관련된 모든 소프트웨어를 보이고, 시험자가 원하는 소프트웨어를 선택하여 수정하는 방법을 사용하였다. VIST는 기존의 시험 도구에서 적용하지 않은 방화벽을 적용함으로서 재시험하는 소프트웨어를 최소화하였으며, 시험자의 프로그램 시험에 소요되는 시간과 노력을 절약하였다.

참 고 문 헌

[1] 양해술, "클라이언트/서버 개발의 품질관리 방법", 정보처리학회지, Vol.4, No.6, pp.114-123, 1997.

[2] 김태훈, 신우창, 김태균, 신영길, 우치수, "분산 객체 지향 소프트웨어 개발 환경의 설계 및 구현", 정보과학회 논문지(C), 제3권, 제2호, pp.139-151, 1997, 4.

[3] G. Rothermel and M. J. Harrold, "A Safe, Efficient Regression Test Selection Technique," *ACM Transactions on Software Engineering and Methodology*, pp.173-210, Vol.6, No.2, April 1997.

[4] Yoonkyu Jang and Ongrae Kwon, "Change Impact analysis of Object-Oriented Software," *Korea Advanced Institute of Science and Technology, MS Thesis*, 1998.

[5] David C. Kung, Jerry Gao, Pei Hsia, "Class Firewall, Test Order, and Regression Testing of Object-Oriented Programs," *Journal of Object-Oriented Programming*, pp.51-65, May 1995.

[6] Jungwon Bang and Doohwan Bae, "Change Analysis for Regression Test of Object-Oriented Software," *Korea Advanced Institute of Science and Technology, MS Thesis*, 1997.

[7] S. A. Bohner and R. S. Arnold., *'Software Change Impact Analysis,'* Computer Society Press, 1996.

[8] M. J. Harrold and John D. McGregor, "Incremental Testing of Object-Oriented Class Structures," *Proceedings of the 14th International Conference on Software Engineering*, pp.68-80, 1992.

[9] Lee J. White, Hareton K. N. Leung, "A Firewall concept for both Control-Flow and Data-Flow

in Regression Integration Testing," Proceeding of the International Conference on Software Maintenance, pp.262-271, 1992.

[10] Brian A. Malloy, John D. McGregor and Anand Krishnaswamy, Murali Medikonda, An Extensible Program Representation for Object-oriented Software, Technical Report TR94-109, Department of Computer Science, Clemson University, 1994.

김 경 희

e-mail : kkhee@cs.sookmyung.ac.kr

1990년 숙명여자대학교 전자계산학과 졸업

1993년 숙명여자대학교 전자계산학과 대학원(이학석사)

1996년~현재 숙명여자대학교 전자계산학과 박사과정 수료

1999년~현재 천안외국어 대학 산업전산학과 전임강사

관심분야 : 소프트웨어 시험, 수정 영향 분석, 멀티미디어, 분산처리 시스템 등

박 재 년

e-mail : parkjn@cs.sookmyung.ac.kr

1966년 고려대학교 졸업

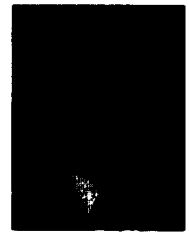
1969년 고려대학교 석사

1970년 함브르크 대학 수료

1981년 고려대학교 박사

1983년~현재 숙명여자대학교 전산학과 교수

관심분야 : 객체지향 분석 방법론, 소프트웨어 시험 등



윤 용 익

e-mail : yiyoon@cs.sookmyung.ac.kr

1983년 동국 대학교, 통계학과

1985년 한국과학기술원 전산학과 (석사)

1994년 한국과학기술원 전산학과 (박사)

1985년~1997년 한국전자통신연구원 책임연구원

1997년~현재 숙명여자대학교 전산학과 교수

관심분야 : 분산 시스템, 실시간 시스템, 컴퓨터 통신과 네트워크 등