모듈 클래스 종속 그래프를 이용한 객체지향 프로그램 슬라이싱에 관한 연구

김 운 용[†] · 정 계 동[††] · 최 영 근[†††]

요 약

본 논문에서는 클래스들간의 종속관계를 효율적으로 표현하기 위한 모듈클래스 종속그래프를 제안한다. 객체 지향언어는 설계시 독립적으로 개발되어지고, 클래스들간의 관계를 형성하여 구성되어진다. 따라서 이러한 독립적인 특성을 고려하고, 클래스 계층구조에 존재하는 클래스들간의 관계를 효율적으로 표현할 필요가 존재한다. 본 논문에서는 어플리케이션에 종속적인 기존의 시스템 종속그래프와 단위 클래스를 표현하는 클래스 종속그래프에서 제시되지 않고 있는 클래스들간의 관계를 표현하기 위해 객체지향시스템의 설계단위인 모듈의 개념을 이용하여 모듈클래스 종속그래프를 제시하고, 객체지향의 특징인 객체의 생성자, 상속관계 및 동적바인딩 효과를 적용시켜 효율성의 검증과 이를 절차간 슬라이싱에 적용시켜 슬라이싱의 관계를 고찰한다. 또한 모듈클래스에 존재하는 클래스의 멤버데이터들간의 구별을 가능하게 하기 위한 파라메터의 표현법을 제시한다. 이러한 모듈클래스 종속그래프를 통해 시스템 설계시 모듈 클래스들간의 관계를 보다 정확하게 분석할 수 있고, 시스템 분석에 필요한 역공학, 테스팅, 시각화와 같은 다른 응용에 폭넓게 적용될 수 있다.

A Study on the Object-Oriented Program Slicing using Module Class Dependency Graph

Woon-Yong Kim[†] · Kye-Dong Jung[††] · Young-Keun Choi[†††]

ABSTRACT

This paper presents the Module Class Dependency Graph for expressing the dependency relations between classes effectively. The object-oriented language is developed independently at design time, and consists of relationship between classes. Therefore we need to consider these characteristics of independence, and to express effectively the relation of classes which is existed in class hierarchy. In the System Dependence Graph and Class Dependence Graph, the relationship of classes is not expressed. To express the class relationship, we propose the Module Class Dependence Graph, and we verify the effectiveness of this method applying to object constructor, inheritance relationship and dynamic binding. Also, we presents the expressing method of parameter to identify the member data of classes. Using this Module Class Dependency Graph, we can analyze the relationship of module class correctly at design time. This method can be applied to reverse engineering, testing, visualization and other various fields to analyze system.

※ 이 논문은 1999년도 광운대학교 교내 학술연구비에 의하여 연구되었음.

† 준 회 원 : 광운대학교 대학원 전자계산학과
†† 정 회 원 : 광운대학교 대학원 전자계산학과
††† 정 회 원 : 광운대학교 전자계산학과 교수

논문접수 : 1998년 11월 14일, 심사완료 : 1999년 5월 31일

1. 서 론

Weiser[19]에 의해 소개된 슬라이싱에 대한 연구는 지속적으로 진행되고 있으며, 특히 절차간 슬라이싱(Interprocedural Slicing)를 수행하기 위해 Susan Horwitz[8]는 시스템 종속 그래프(SDG : System Dependance Graph)와 이를 이용한 2단계 그래프 도달 알고리즘(Two Phase Graph Reachability Algorithm)을 소개하였고, 이를 기반으로 하는 절차간 슬라이싱에 대한 연구가 디버깅, 프로그램 이해, 테스팅, 병렬 처리, 소프트웨어 메트릭스, 역공학[1,2,3,5,7,9,13,16,17]등과 같은 다양한 분야에서 이루어지고 있다. 프로그램 슬라이스[7]는 프로그램의 어느 정점 P에서 정의되거나 사용되어지는 변수 V에 영향을 미치는 프로그램 문장의 집합으로 정의되고, 한 프로시쥬어에 대한 종속그래프를 프로시쥬어 종속그래프(PDG : Procedure Dependence Graph)[4]라 하고, 이 들 프로시쥬어들간의 종속관계를 고려한 그래프를 (SDG : System Dependence Graph)라고 한다.

이들 그래프를 이용한 객체 지향 언어를 표현하는 방법[10][11][14]이 2가지 표현방법으로 연구되고있다. 첫 번째 방법으로 어플리케이션에 종속적인 형태의 SDG를 들 수 있다. 이 방법은 클래스 계층구조를 대상으로 하기보다는 이를 사용하는 메인 프로그램을 대상으로 한 SDG의 표현으로 설계시 독립적으로 개발되어지는 클래스 계층구조에서의 종속관계를 정확히 표현하지 못하고 있는 단점을 가진다. 두 번째 방법은 한 클래스를 대상으로 시스템을 표현하는 클래스 종속그래프(CDG : Class Dependence Graph)를 들 수 있다. 이 클래스 종속그래프는 한 클래스를 대상으로 이 클래스에 존재하는 종속관계를 표현하고 있다. 그러나 이들은 한 클래스의 입장에서 표현하기 때문에 클래스들 사이의 관계를 적절하게 표현하지 못하고 있다. 이러한 문제점을 해결하기 위해 본 논문에서는 객체 지향 언어의 기본설계 단위가 되는 모듈을 기준으로 모듈 클래스 종속 그래프(MCDG : Module Class Dependency Graph)를 표현하기 위해, 모듈 사이의 파라메터들간의 관계와 프레임 종속그래프[6]를 확장한 모듈프레임 제어종속그래프를 제시하고, 객체지향언어의 특징인 생성자, 상속관계 및 동적바인딩효과 들의 표현을 MCDG에 적용시키므로써의 잇점과, 절차간 슬라이싱 알고리즘에서의 슬라이싱 관계를 고찰하고자 한다.

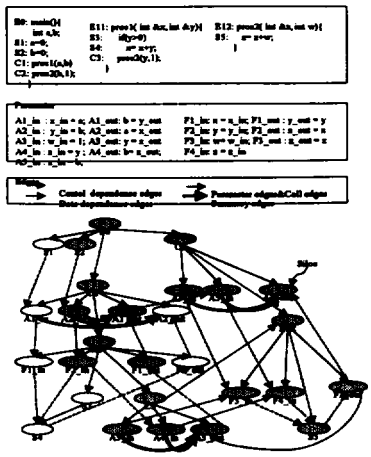
논문의 구성은 먼저 2절에서 슬라이싱과 그래프 표현방법과 관련된 연구와 문제점을 지적하고, 3절에서 MCDG의 정의 및 표현방법과 이 표현에 필요한 파라메터(Parameter) 표기법 및 모듈 클래스를 통해 표현되어지는 객체의 생성자 및 파괴자, 상속관계, 동적 바인딩의 표현을 통해 제시된 MCDG의 잇점과, 절차간 슬라이싱 알고리즘에 적용 시켜 슬라이싱 관계를 고찰한다. 4절에서는 모듈클래스 종속 그래프를 기존의 종속그래프와 비교 분석한다.

2. 관련 연구

이 절에서는 SDG를 이용해 현재 가장 많이 이용되고 있는 절차간 슬라이싱 방법을 소개하고, 현재 객체지향 언어를 위해 연구되고 있는 그래프 표현 방법인 어플리케이션 독립적인 메인 프로그램를 표현하는 SDG와 단위 클래스를 대상으로 표현하는 CDG에 대해 고찰하고 이들의 문제점을 지적한다.

2.1 절차간 슬라이싱

객체 지향 언어의 특성이 클래스 단위 구성을 가지며 이 클래스 단위는 많은 멤버함수와 멤버데이터로 이루어지고 있기 때문에 이들 간의 관계를 표현하기 위해 절차간 슬라이싱의 개념을 도입해야 한다. 이 절차간 슬라이싱을 수행하는 과정을 표현하기 위해 (그림 1)과 같은 구조적 프로그램의 SDG[8]를 구성할 수 있다. 절차간 슬라이싱은 이렇게 구성된 SDG를 이용하여 Susan Horwitz[8]에 의해 제시된 2단계 그래프 도달 알고리즘을 적용하여 슬라이싱이 되어진다.



(그림 1) 절차간 슬라이싱을 위한 SDG

(그림 1)에서 제시한 프로그램을 대상으로 절차간 슬라이싱을 수행하기 위해 SDG는 프로그램에서의 문장과 호출을 표현하기 위한 정점, 그들간의 제어 종속 관계를 표현하는 제어 종속 에지(Control Dependence Edges), 데이터 종속관계를 표현하는 데이터 종속 에지(Data Dependence Edges), 그리고 호출 사이트의 내부에 대한 데이터 종속 관계를 표현하는 이행적 종속성 에지(summary edges)[15]와 호출 및 파라메터 바인딩을 표현하는 호출/파라메터 에지(call & parameter edges)로 구성되어져 있다. 이렇게 구성된 SDG를 기반으로 2단계 그래프 도달 알고리즘을 이용한 절차적 슬라이싱을 수행하면 다음과 같다. 첫번째 단계에서 이 알고리즘은 대상 슬라이스 정점(C2->A4_out)으로부터 파라메터 아웃 에지(Parameter-Out Edges)를 제외한 모든 에지를 이용해 후방향으로 도달된 정점을 표시한다.(C2->A4_out, C2->A3_in, C2->A5_in, C2, E0, C1->A1_out, C1->A2_in, C1, S2). 이 과정이 완료되면, 두번째 단계에서는 첫번째 단계에서 검색된 정점들을 대상으로 파라메터 인 에지(parameter-in edges)와 호출 에지를 제외한 모든 에지를 이용하여 후방향으로 도달된 정점을 표시한다(E12->F3_out, S5, E12->F3_in, E12->F4_in, E12, E11->F1_out, C3->A3_out, C3->A3_in, C3->A4_in, C3, E11->F2_in, E11). 이 표시된 정점들이 바로 슬라이스 정점(C2->A4_out)의 슬라이스가 된다.

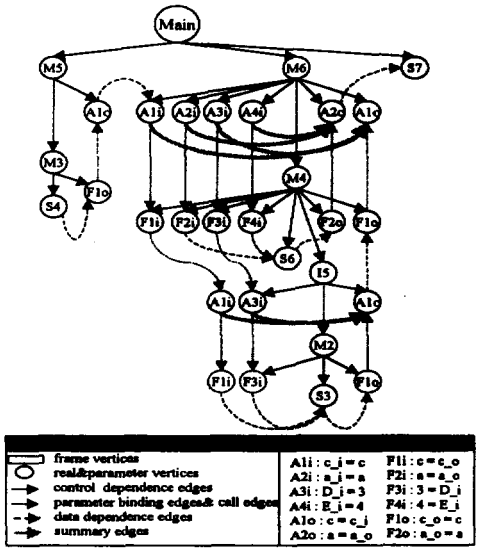
2.2 어플리케이션에 종속적인 시스템 종속 그래프

객체 지향언어를 대상으로 어플리케이션 종속성을 가진 SDG는 Main()에 의해 작성된 프로그램을 기준으로 그래프를 구성하는 방법이다. 이러한 SDG를 구성하기 위해 (그림 2)와 같은 예제 프로그램을 기준으로 작성하면 (그림 3)과 같이 구성된다.

```
C1: Class A{                C2: Class B : public A{          Main()
      private int a,b;            public int c;              {
      public :                    public :                     C2 *c;
M1:   A(){                  M3:   B(){                 M5:    c = new C2();
S1:       a= 0;             S4:       c=0;           M6:    c->setC(3,4);
S2:       b=0;                      }                S7:    cout << c.c;
      }                     M4:   void setC(int D,int E){      }
M2:   void setA(int C){     S6:       c=E;
S3:       a = C;            I5:       setA(D);
      }                               }
};                                  };
```

(그림 2) 예제 프로그램

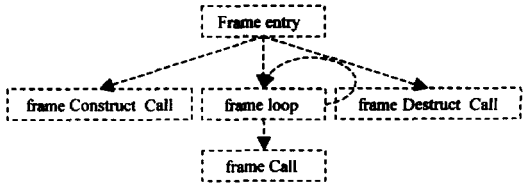
(그림 3)은 (그림 2)의 Main()함수에 대한 SDG로 Main()에 포함된 구성을 중심으로 프로그램을 구성한 형태이다. 이러한 구성상 SDG는 클래스 계층구조 보다는 실제 작성된 어플리케이션에 종속하여 구성함으로써, 이 프로그램에 필요한 클래스 계층구조의 정보를 확인할 수 없는 형태를 가진다. 그러므로 이 어플리케이션에 필요한 클래스 계층구조에 대한 적절한 표현이 필요하다.



(그림 3) 메인함수에 대한 SDG표현

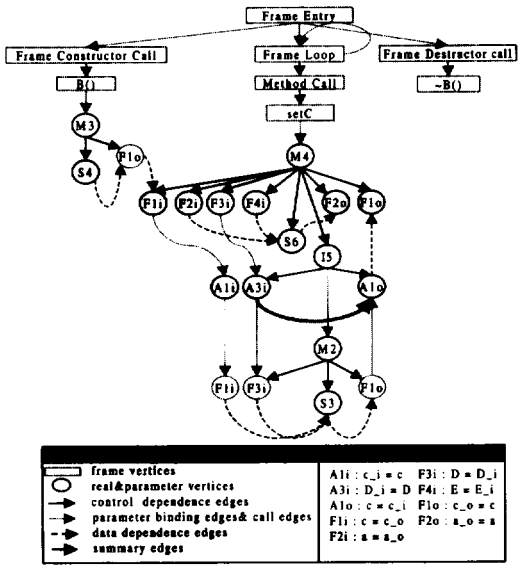
2.3 단위 클래스에 대한 클래스 종속 그래프

클래스 종속 그래프는 단위 클래스내에 존재하는 메소드들간의 종속관계를 독립적으로 표현하는 그래프 형태를 나타낸다. 이러한 표현을 위해 Harrold 와 Rothermel[6]은 객체지향 소프트웨어의 그래픽 표현에 대한 연구에서, 독립적인 하나의 클래스를 기존의 프로그램 분석 도구에 적용시키기 위한 방법으로 공통 멤버함수 호출과정을 묘사하는 제어종속 그래프인 프레임 제어종속그래프[6]를 제시하여 (그림 4)와 같이 표현하였다.



(그림 4) 프레임 제어종속 그래프

이러한 프레임 제어종속그래프를 이용해 단위 클래스에 대한 클래스 종속그래프를 구성할 수 있다. (그림 2)와 같은 예제에서 C2 클래스에 대한 클래스 종속그래프를 구성하면 (그림 5)와 같이 표현된다.



(그림 5) C2 클래스에 대한 CDG

(그림 5)는 (그림 2)의 예제 프로그램에서 C2 클래스를 대상으로 CDG를 구성한 것이다. 이러한 구성은 프레임 형태를 통해 단위 클래스 대상 CDG를 구성할 수 있으나 몇가지 문제점을 가지고 있다. 첫 번째로는 M2를 호출하는 과정에서, M2가 존재하는 클래스에 대한 표기가 존재하지 않고, 두 번째로 이들 단위 클래스에 존재하는 파라메터간의 클래스에 대한 구분이 존재하지 않는다. 즉 F2i의 변수는 C1클래스에 존재하는 변수이다는 것이 명시되어야한다. 세 번째로 M2가 호출되어지기전에 객체 지향언어의 특성상 M2에 해당되는 생성자 함수 M1이 호출되어져야하기 때문에 이러한 생성자의 관계를 표현해야 한다.

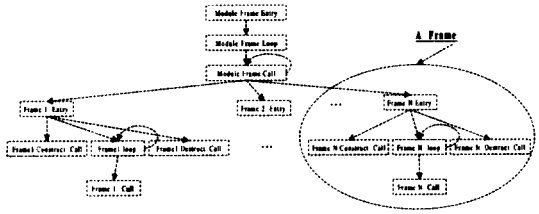
(그림 3)과 (그림 5)에서 제시된 문제점을 해결하기 위해 본 논문에서는 MCDG를 제안하고자한다. 먼저 모듈 클래스를 구성하는 전체 구성형태를 나타내는 모듈 프레임과, MCDG에 필요한 파라메터들간의 연관관계를 제시하고, 이 MCDG를 통해 표현되어지는 특징을 고찰하기 위해 객체지향언어의 특징을 적용시킨다.

3. 모듈 클래스 종속그래프

어플리케이션에 독립적인 MCDG를 표현하기 위해서는 먼저 클래스들간의 관계를 표현할 수 있는 모듈 프레임이 필요하다. 이러한 모듈프레임를 표현하기 위해 본 논문에서는 기존의 단위클래스들간의 모든 멤버 함수들의 호출을 가능하게 하는 프레임 제어종속그래프[6]를 확장하여 모듈클래스에 대한 프레임 제어종속그래프 표현하는 방법을 제시한다.

3.1 모듈 클래스에 대한 프레임 제어종속그래프 표현

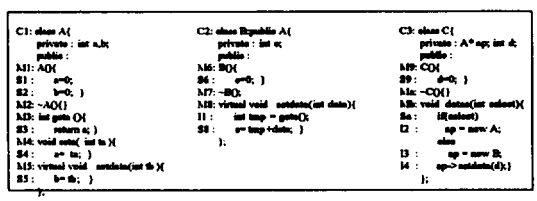
모듈 클래스 내부에 존재하는 모든 클래스들간의 호출경로를 구성하기 위해 (그림 6)과 같은 형태로 모듈 클래스 프레임 제어종속그래프를 구성한다.



(그림 6) 모듈클래스 프레임 제어종속그래프 구성

(그림 6)은 (그림 4)에서 제시한 프레임 제어종속그래프를 확장하여 모듈 클래스 프레임 제어종속그래프를 구성한 것이다. 이러한 모듈클래스 프레임 제어종속그래프는 단위 클래스를 구성요소로 하여 그들간의 관계를 표현함으로써, 모듈클래스에 존재하는 모든 클래스들간의 공통 호출과정을 묘사하고 있다.

이제 이러한 형태의 모듈클래스 프레임 제어종속 그래프를 적용시킨 MCDG를 표현하기위해 (그림 7)의 예제를 이용하여, MCDG를 구성하는데 필요한 파라메터 표현방법 및 객체지향 특징을 적용시키므로써, MCDG를 이용한 표현의 잇점을 제시하고자 한다.



(그림 7) 객체 지향 예제 프로그램

3.2 모듈 클래스 제어종속그래프의 파라메터 표현

모듈 클래스의 구성단위가 클래스이며, 이 클래스는 멤버함수와 멤버데이터로 구성되고, 이들은 데이터 종속과, 제어 종속관계를 가진다. 이러한 종속관계를 MCDG에 적용시키기 위해 멤버 데이터를 파라메터 형태로 구성해야 하며, 구성된 파라메터는 객체 특성을 담고 있어야 한다. 즉 클래스에서 사용되어지는 객체가 동일한 클래스에서 다른 클래스변수이름으로 같은 클래스를 사용할때나, 상속된 클래스에서 사용하는 멤버데이터와 상속할 클래스에 존재하는 멤버데이터는 구별되어지는 형태로 표현되어야한다. 이러한 특징을 적용시키기 위한 파라메터 표현방법으로 (그림 8)과 같은 방식을 제시한다. 이 표기는 (그림 7)의 예제프로그램을 이용하여 표기하였다.

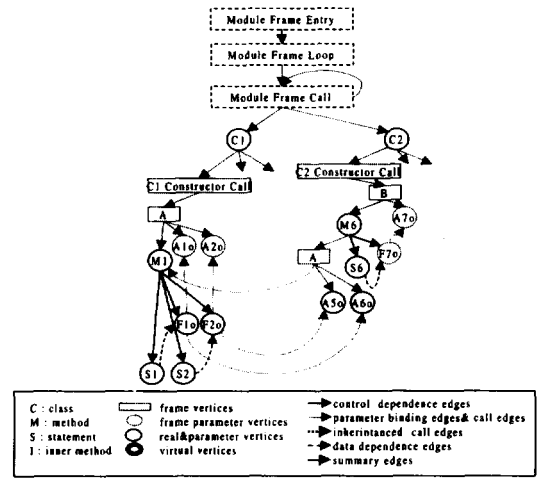
파라메터 변수이름 : 자신의 클래스_[상속클래스_]_클래스변수]*변수이름

예를 들어 B클래스에서 상속된 A클래스의 멤버데이터인 a를 표현하기 위해 B_A_a로 표현하고, A클래스에 존재하는 멤버데이터 a는 A_a로 표현된다. 이러한 구성방식은 모듈 클래스 내부에서 데이터 종속관계를 표현할 때, 클래스간의 멤버데이터를 정확하게 분리할 수 있도록 한다.

이제 제시된 모듈클래스 프레임 제어종속그래프와 파라메터 표기 방법을 이용해 MCDG를 제안하고자한다. 먼저 MCDG를 구성시에 존재하는 잇점들을 객체지향적인 특성을 고려하여 제시하고, 이들의 특징을 포함하여 (그림 7)에서 제시한 예제프로그램을 이용한 전체적인 MCDG를 구성하고자 한다.

3.3 객체 생성자 표현

구조적 프로그램과 달리 객체지향 프로그램은 클래스가 사용되어질 때 해당 클래스의 생성자가 호출되어지고, 상속받은 클래스가 사용되어질 때에는 이 상속받은 클래스는 자신의 생성자를 호출하는 동시에 자신의 부모 클래스의 생성자도 함께 호출하게된다. 이러한 표현방법을 제시하기 위해 (그림 9)에서는 (그림 7)의 B클래스 생성자를 대상으로 한 MCDG를 표현하고 있다.



(그림 9) 객체 생성자 표현

(그림 9)는 (그림 7)의 B 클래스의 생성자 관계를 표현한 것으로, B클래스는 A클래스를 상속받아 생성된 클래스이기 때문에, 자신의 부모클래스인 A 클래스를 호출하는 형태로 구성될 수 있다. 여기에서 C2의 생성자함수인 M6이 호출될 때 C1의 생성자함수 M1이 호출되는 과정을 표현하고 있다. 이러한 표현을 MCDG에 적용시키므로써 모듈클래스 내부에 존재하는 클래스 및 상속된 클래스 생성자들간의 관계를 정확하게 분석하고 검증하는 것이 가능하다.

a_in A_a	b_in A_b	ta_in A_ta	tb_in A_tb	a_in B_A_a		c_in B_c	data_in B_data	d_in C_d	select_in C_select		b_in C_ap_b	c_in C_ap_c			ta_in B_A_ta
a a_in	b b_in	ta ta_in	tb tb_in			c c_in	data data_in	d d_in	select select_in						
A_a a_out	A_b b_out			B_A_a a_out	B_A_b b_out	B_c c_out		C_d d_out		C_ap_a a_out	C_ap_b b_out	C_ap_c c_out	C_ap_A_a a_out	C_ap_A_b b_out	
a_out a	b_out b					c_out c		d_out d							

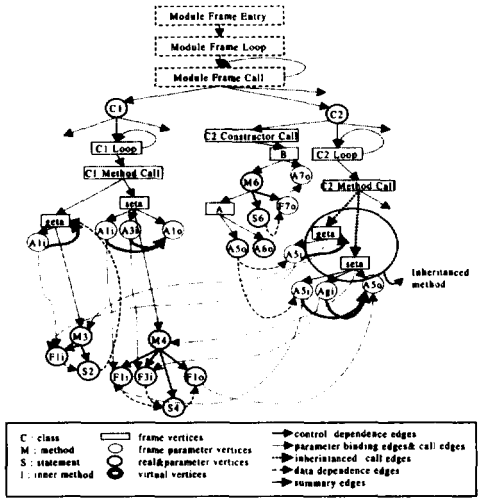
* 변수 표기법 : 자신의클래스_[상속 클래스_] 클래스 변수_]*변수이름

Key for Parameter Vertices

(그림 8) 예제 프로그램 파라메터 표기 방법

3.4 상속된 멤버함수의 표현

객체지향의 특징으로 상속된 클래스는 자신에게 존재하지 않는 멤버함수를 조상 클래스로부터 상속받아 사용할 수 있다. 이러한 특징을 표현하기 위해 상속된 클래스인 B클래스를 이용하여 표현하면, (그림 10)과 같은 형태로 상속된 멤버함수가 표현된다.

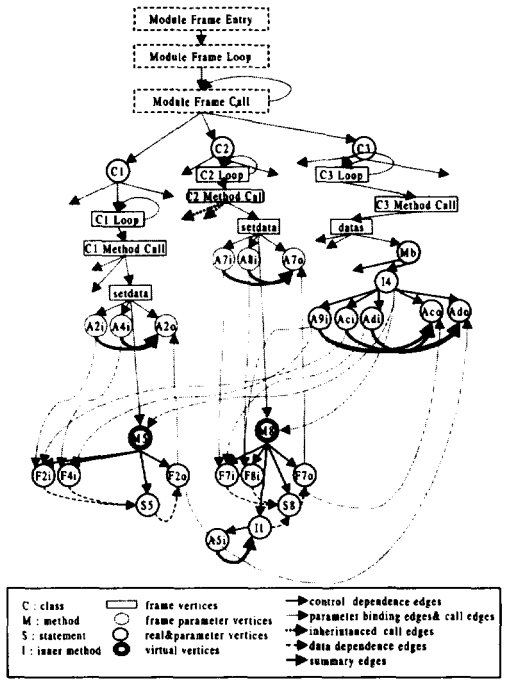


(그림 10) 상속된 메소드 표현

(그림 10)의 B클래스는 A클래스의 geta()와 seta()라는 메소드를 상속받아 사용가능하다. 그러나 이들 B에서 호출되는 geta()와 seta()를 표현하기 위해서는 B클래스가 상속한 A클래스의 멤버변수를 이용하여 표현하여야한다. 이러한 멤버변수는 MCDG안에서 파라메터로 표현되고, A클래스 자신에 포함된 멤버변수와 구별되어 표현되어진다. 예제에서 A클래스에 존재하는 geta()는 A클래스의 멤버변수 A1i이라는 실 파라메터(Actual Parameter)을 가지지만, B클래스에 존재하는 geta()는 B클래스에서는 A클래스의 멤버변수를 표현하는 A5i라는 실 파라메터를 가지고서 geta()가 호출되어지도록 표현하고 있다. 이러한 표현방법은 모듈 클래스에 존재하는 클래스들의 멤버함수가 실제 적용되고있는 멤버데이터를 이용하여 표현하는 것을 가능하게 하여, 모듈 클래스 내부의 멤버데이터의 흐름을 명확하게 파악할 수 있도록 한다.

3.5 동적 바인딩을 가지는 모듈클래스 종속그래프의 표현

동적 바인딩은 데이터 타입이 컴파일시가 아닌 실행시에 결정된다는 특징 때문에, 이러한 동적 바인딩 효과를 표현하기 위해서는 바인딩이 이루어지는 모든 호출 가능한 경로를 표현함으로써 구성할 수 있다. (그림 11)은 (그림 7)의 C 클래스의 멤버함수인 datas()에 존재하는 ap->setdata(d)를 표현한 것이다.



(그림 11) 동적 바인딩 표현

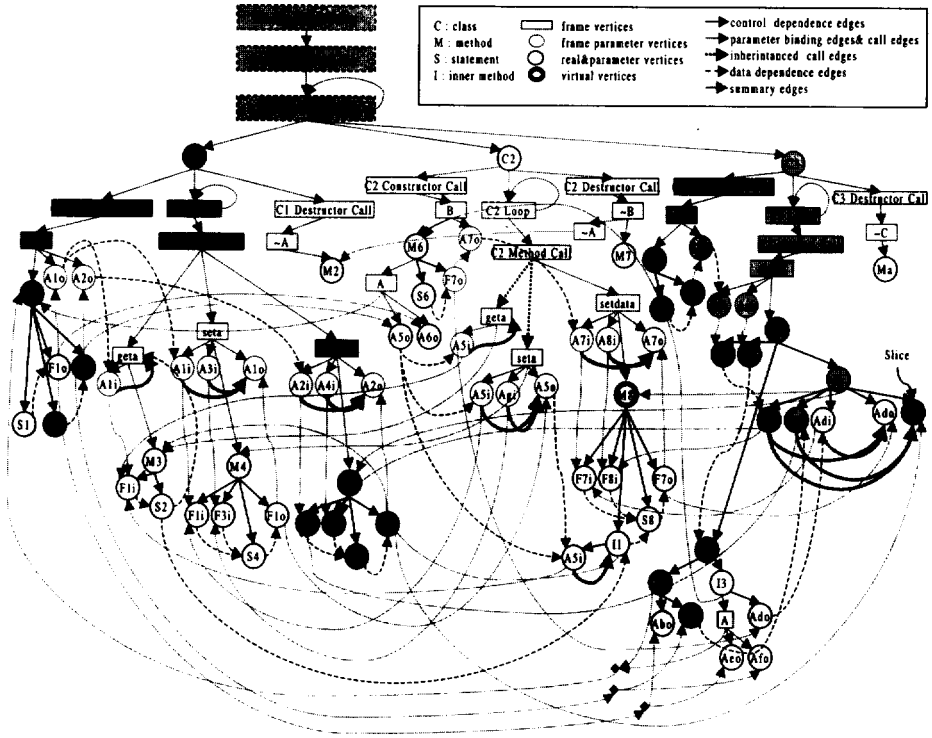
이 문장 I4는 실행시 M5와 M8의 두 개의 메소드 중 하나를 수행할 것이다. 이러한 동적 바인딩을 표현하기 위해 도달 가능한 모든 가상함수의 호출 관계를 표시하고, 이들간의 데이터흐름을 파라메터를 이용하여 구성한다. (그림 11)에서 I4는 M5와 M8을 호출하는 두 개의 호출 에지를 가지고서 동적바인딩를 표현하고 있다. 여기에서 동적바인딩 표현을 모듈클래스 제어종속그래프에 적용시키므로써, M5와 M8이 속한 클래스의 식별이 가능해지고, 포함된 클래스에서의 각 멤버함수의 역할과 영향을 모듈클래스단위로 분석할 수 있게하여 호출되는 멤버함수의 관계를 명확하게 판단할 수 있도록 하였다. 이것은 객체지향 개발이 하나의 클래스를 대상으로 이루어지지 않고, 클래스들간의 관계를 고려하여 개발되어지기 때문에 그들간의 관계를 고려하여 구성한 형태이다.

3.6 모듈 클래스 종속그래프 표현

지금까지 파라메터 표현과 객체 지향적인 특징을 MCDG에 적용시켜 표현되는 특징들을 제시하였다. 이러한 특징을 고려하여 MCDG를 전체적으로 표현하면 (그림 12)와 같이 표현된다. 이 MCDG는 (그림 7)의 예제프로그램을 대상으로 하여 어플리케이션 독립적인 환경에서, 모듈 클래스에 존재하는 클래스들간의 관계를 표현하고 있고, 모듈 클래스 내에서 발생하는 동적 바인딩, 생성자, 상속된 메소드의 처리의 특징들을 포함하여 구성되었다. 이러한 MCDG는 어플리케이션에 독립적인 슬라이싱을 수행할 수 있으며, 모듈 클래스 설계시 특정 어플리케이션에 종속하지 않고, 모듈 클래스 내부에 존재하는 모든 클래스들간의 관계에 대한 정보에 대한 효율적인 분석이 가능하다.

3.7 모듈클래스 종속그래프를 이용한 슬라이싱

이렇게 구성된 MCDG를 이용하여 절차간 슬라이싱 알고리즘을 적용시켜 표현 방법을 고찰하고자 한다. 1단계에서 파라메터 아웃 에지를 제외한 모든 에지를 이용하여 후방향 검색을 통해하면서 도달 가능한 정점를 찾는다. 이러한 슬라이싱 진행과정을 표현하기 위해 (그림 13)에서 제시한 MCDG에서 한 슬라이싱 대상 S=<p,v>를 <I4,Aco>로 설정하고 슬라이싱을 적용시킨다. 이때 <I4,Aco>는 I4번째 줄에서, 변수 C클래스에서 사용되는 A클래스 멤버데이터 b를 대상으로 한 슬라이싱이다. 1단계를 수행한 후 얻어진 정점은 I4->Aco, I4->A9i, I4->Aci, I2->Aco, I2, Mb->Sa, I4, Mb->F9i, Mb->Fai, Mb, datas->A9i, datas->Aai, C->A9o, data, C, C3 Method Call, C3 Loop, C3 Construct Call, C3, Module Frame Call, Module Frame Loop, Module Frame Entry라는 정점를 찾는다. 이렇게 찾아진 정점을 이용하여 2단계를 수행한다. 2단계에서는 표시된 정점들로부터, 파라메터 인 에지와 호출 에지를 제외한 모든 에지를 이용하여 후방향으로 검색하여 도달되어지는 정점을 찾는다. 2단계 완료 후 찾은 정점은 M5->F2o, M5->S5, M5->F4i, M5->M2i, M5, setdata, C1 Method Call, C1 Loop, C1, M1->F2o, M1->S2, M1, A, C1 Construct Call,



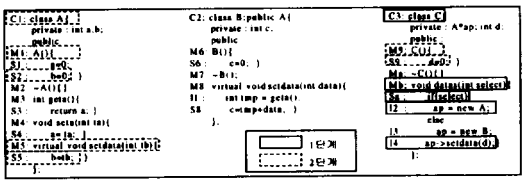
(그림 12) 모듈 클래스 종속 그래프

M9->F9o, M9->S9, M9가 된다. 이렇게 찾은 정점들에서, 모듈프레임구성을 위해 만든 정점을를 제거하고, 실제 프로그램에 적용되어지는 정점로 구성을 하면 다음과 같다.

1단계 : I4->Aco, I4->A9i, I4->Aci, I2->Aco, I2, Mb->Sa, I4, Mb->F9i, Mb->Fai, Mb, C->A9o, C3

2단계 : M5->F2o, M5->S5, M5->F4i, M5->M2i, M5, C1, M1->F2o, M1->S2, M1, M9->F9o, M9->S9, M9

이 정보를 이용하여 실제 예제 프로그램에 적용시켜 표현하면 (그림 13)과 같은 슬라이스로 표현된다.



(그림 13) I4에서 C클래스에서 사용된 A클래스의 멤버데이터 b에 대한 슬라이싱

(그림 13)에서 보여지듯이 C클래스에서 사용된 A클래스의 멤버데이터 b는 C1,과 C3클래스와 연관되어지고, M1,S2,M5,S5,M9,S9,Mb,Sa,I2,I4와 연관되어지는 것을 확인할 수 있다.

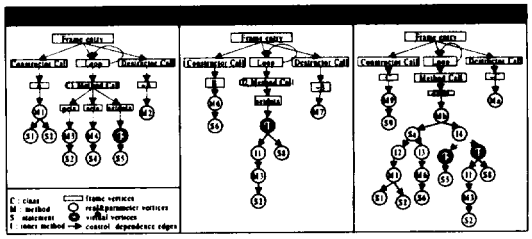
본 논문에서 제시된 MCDG를 이용한 슬라이싱은 하나의 클래스나 Main()를 이용한 어플리케이션 종속적인 방법이 아닌 객체지향 시스템 설계와 재사용의 기본단위인 모듈클래스를 대상으로한 독립적인 슬라이싱을 수행할 수 있고, (그림 12)의 MCDG의 형태에서 보여지듯이 슬라이싱된 각 정점의 관계를 클래스들의 관계와 연관하여 분석할 수 있기 때문에 시스템 분석 및 설계에 필요한 정보를 효과적으로 분석하고 이용할 수 있다.

4. 기존의 그래프표현과의 평가 및 비교분석

제안된 MCDG는 기존 그래프에 포함되지 않는 생성자 및 상속관계등의 객체지향적인 특징을 추가하고, 이들간의 관계을 모듈단위로 표현하고 있다는 점에서 기존의 그래프와 다르다. 이러한 그래프를 기존의 그래프와의 성능 및 효율성을 고찰하기위해 제시된 그래프의 특징과 비슷한 CDG를 통해 분석한다. 구조적 형태의 SDG나 객체지향 형태의 SDG는 구성상 main을 기반으로 구성되기 때문에 클래스에서 호출가능한 모든 클래스 와 멤버함수를 표현하는 MCDG와는 구별되기 때문에 비교대상에서 제외한다. 성능평가방법은 그래프의 표현의 모듈화와 표현력를 대상으로 수행한다.

4.1 그래프 표현의 모듈화

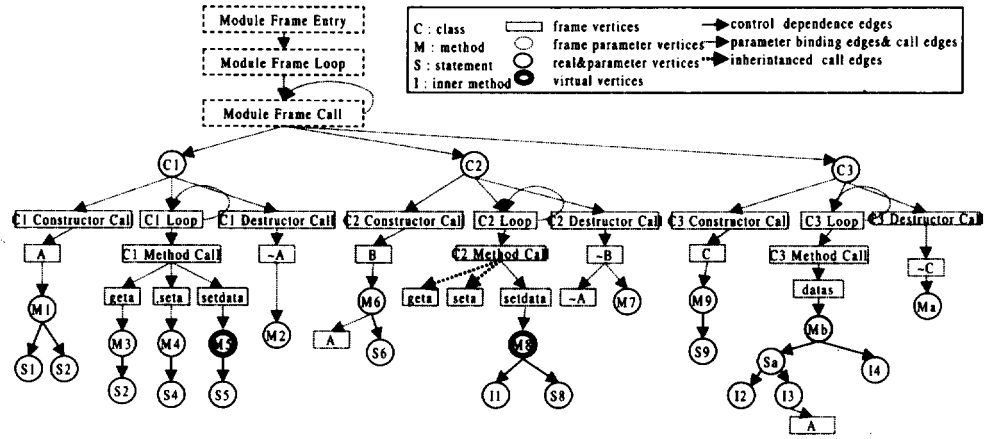
CDG는 단위 클래스 단위로 그래프가 구성되기 때문에 모듈화 되지 않고 각 클래스 단위로 표현되기 때문에 (그림 7)의 예제 프로그램 대상으로 표현하면 (그림 14)와 같은 형태로 표현된다.



(그림 14) CDG표현

이러한 구성형태는 몇 가지 문제점을 가지고 있다. 첫 번째로, 사용되는 문장은 한 클래스 대상으로만 표현되기 때문에 설계단위가 되는 모듈에서의 클래스들간의 관계가 적절하게 표현되지 못하고 있다. 두 번째로 문장의 호출에 의해 사용되어지는 다른 클래스내의 문장들이 중복해서 포함됨으로 노드의 중복효과를 가져온다. 실 예로 이 클래스들을 표현하기 위해 사용되는 노드의 수를 계산하면 62개로 표현되고 있다. 이러한 문제점을 해결하는방법인 MCDG를 표현하면 (그림 15)와 같이 표현될 수 있다.

이러한 표현을 CDG와 비교하면, 첫번째로 문장의 노드의 관계를 모듈단위에서 표현되어지기 때문에 문장의 포함관계가 클래스에 명확하게 표현될수있다는 장점을 가지고, 둘째로 모듈내부에서 사용되는 노드는 중복을 피하여 구성할 수 있으며 사용된 노드에 해당하는 클래스의 구성까지 명확하게 분석할 수 있다. 실 예로 예제를 MCDG로 구성할 때 표현되는 노드수는 58개로 여기에서는 생성자 및 상속관계까지 포함하여 구성하였기 때문에 이를 고려한다면 더 적은 노드로 모듈 클래스에 존재하는 관계표현이 가능하고, 세 번

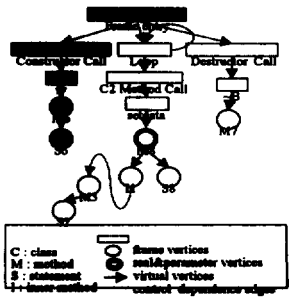


(그림 15) MCDG의 표현

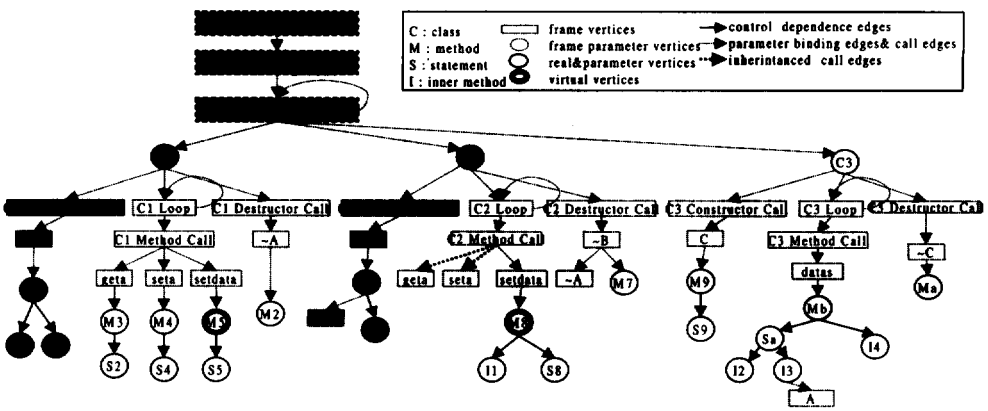
째로 호출과정의 표현시, 같은 이름의 객체가 존재할 때 이들간의 구분이 애매모호한 CDG의 형태를 모듈클래스 내부에 존재하는 클래스와 멤버함수관계를 그래프에 표현함으로써 이러한 문제점을 보안하는 특징을 가진다.

4.2 표현력 분석

구성된 MCDG의 표현력을 분석하기위해 예로 S6에 해당되는 노드를 이용하여 슬라이싱을 CDG와 MCDG에 적용하여 분석하면, CDG에 포함되는 슬라이싱대상 노드는 5개이고, MCDG에 적용되는 슬라이싱 대상 노드는 15개이다. 이것은 적용되는 슬라이싱의 범위의 확장을 의미한다. MCDG에서는 생성자관계 및 상속관계에 대한 노드의 포함과 모듈 클래스내에 존재하는 클래스들간의 종속관계를 표현하는 노드를 포함함으로써 MCDG를 통해 얻어진 정보는 객체 지향적인 특징과 클래스들간의 종속성 정보 표현을 가능하게 한다. (그림 16)과 (그림 17)은 S6 노드를 대상으로하는 슬라이싱 적용시 분석되는 노드들을 보여준다.



(그림 16) CDG에 대한 S6대상 슬라이싱

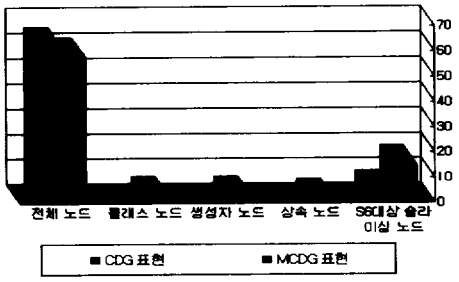


(그림 17) MCDG에 대한 S6대상 슬라이싱

이와 같은 분석을 종합하면, MCDG는 재사용의 기본단위인 모듈을 기반으로 클래스들간의 관계와 이들간의 종속관계를 보다 효율적으로 얻을 수 있을 것이다.

4.3 CDG 와 MCDG 비교

위에서 분석내용을 기반으로 MCDG와 CDG의 표현방법을 대상으로 (그림 7)의 예제프로그램 대상 분석을 수행하면 (그림 18)과 같은 결과를 얻을 수 있다.



(그림 18) CDG와 MCDG의 분석 그래프

(그림 18)의 분석 그래프는 CDG와 MCDG의 효율성을 분석하고 있다. 이 그래프를 통해 CDG에 비해 MCDG의 그래프 표현을 위한 노드수는 감소되지만, 새로운 클래스 표현을 위한 노드 및 생성자, 상속관계 노드가 표현됨으로써 슬라이싱에 필요한 더많은 정보를 MCDG에서 표현할 수 있다. 실제로 S6 대상 슬라이싱을 수행한 결과 MCDG를 이용한 슬라이싱은 CDG의 슬라이싱에 비해 더 많은 슬라이싱 정보를 얻는 것을 확인할 수 있다. 이러한 특징들에 추가해 지금까지 표현되고 있는 그래프들간의 방법을 비교하면 <표 1>과 같이 표현할 수 있다.

〈표 1〉 기존의 방법의 비교

비교대상	구조적 SDG	객체지향 SDG	CDG	본연구(MCDG)
표현대상 언어	구조적 언어	객체지향 언어	객체지향 언어	객체지향 언어
표현 단위	메인과 관련된 프로시져	메인과 관련된 클래스의 멤버함수	클래스 단위의 멤버함수	모듈 클래스 (클래스와 클래스간의 멤버함수
종속성	메인에 종속적	메인에 종속적	독립적	독립적
데이터 표현	변수	변수, 멤버변수	변수, 멤버변수	변수, 멤버변수
프로시져나 클래스 사이의 데이터 구별성	프로시져간 데이터 구별을 고려함	고려없음	고려없음	클래스간 데이터의 구별과, 클래스 사이의 생성자 관계 표현을 고려함

4.4 기존의 방법의 비교

<표 1>의 비교대상은 구조적SDG, 객체지향 SDG, CDG와 제시된 MCDG를 대상으로 비교분석한 표이다. 이 비교표를 통해 MCDG는 SDG와 CDG의 장점들을 고려하여 모듈클래스에 존재하는 클래스들을 대상으로 그래프를 구성하고 있다는 것을 알 수 있다. 이러한 구성은 모듈클래스 단위에서 어플리케이션에 독립적이고, 클래스들간의 관계를 명확하게 표현할 수 있기 때문에, 효율적인 슬라이싱 정보를 관리할 수 있다.

5. 결 론

정보를 표현하고 관리하기 위한 그래프 표현은 시스템 분석과 개발 측면에서 매우 유용하다. 본 논문에서는 객체 지향 설계의 기본단위인 모듈을 대상으로 MCDG를 제안하기 위해, 모듈 클래스의 독립적인 표현방법인 모듈프레임 종속그래프와, 파라메터 표현 방법을 제시하였다. 또한 객체 지향 언어의 특징인 상속관계, 동적 바인딩, 객체의 생성자 표현을 제시한 MCDG에 적용시키고, 이를 이용한 절차간 슬라이싱를 고찰함으로써 발생되는 기대효과를 분석하였다. 객체 지향 언어를 이용한 프로그램은 하나의 클래스보다는 모듈 클래스 단위를 고려한 시스템 구축이 필요하며, 어플리케이션에 독립적인 설계와 객체 특성이 충분히 고려되어야 한다. 이 MCDG는 이러한 시스템 설계에 필요한 효율적인 정보를 제공할 수 있을 것이고, 클래스 정보 표현 및 역공학과 같은 다른 많은 응용에 적용될 수 있을 것이다.

참 고 문 헌

[1] S. Bates, and S. Horwitz, "Incremental program testing using program dependence graph," Proceedings of the Sixth ACM Symp. on Principles of Programming Languages, pp.383-396, January, 1993.

[2] J. M. Bieman, and L. M. Ottenstein, "Measuring function cohesion," IEEE Transactions on Software Engineering," Vol.20, No.8, pp.644-657, August, 1994.

[3] D. Binkley, "Using semantic differencing to reduce the cost of regressing testing," Proceedings of Conf. on Software Maintenance,

pp.41-50, November, 1992.

[4] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," ACM Transactions on Progrm. Lang. Syst., Vol.9, No.3, pp.319-349, July, 1987.

[5] K. B. Gallagher, and J. R. Lyle, "Using Program Slicing in Software Maintenance," IEEE Transactions on Software Eng., Vol.17, No.8, pp.751-761, August, 1991.

[6] M. J. Harrold, and G. Rothermel, "Performing dataflow testing on classes," Second ACM SIGSOFT Symp. on the Foundations of Software Eng., pp.154-163. Dec., 1994.

[7] S. Horwitz, J. Prins, and T. Reps, "Integrating noninterfering versions of programs," ACM Transactions on Progrm. Lang. Syst., Vol.11, No.3, pp.345-387, July, 1989.

[8] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," ACM Transactions on Progrm. Lang. Syst., Vol.12, No.1, pp.26-60, January, 1990.

[9] D. Jackson, and E. J. Rollins, "A new model of program dependence for reverse engineering," Proceedings of the Second ACM SIGSOFT Conf. on Foundations of Software Eng., pp.2-10, December, 1994.

[10] A. Krishnaswamy, "Program slicing : An application of object-oriented program dependency graphs," Technical Report, Dept of Computer Science, Clemson University, pp.94-108, July, 1994.

[11] L. Larsen, and M. J. Harrold, "Slicing object-oriented software," In 18th International Conf., on Software Eng., pp.495-505, Mar., 1996.

[12] N. Nunt, "Performance testing C++ code," Journal of Object Oriented Programming, January, 1996.

[13] L. M. Ottenstein, and J. J. Thuss, "Slice based metrices for estimating cohesion," Proceedings of IEEE-CS International Software Metrics Symp., pp.71-81, 1993.

[14] K. J. Ottenstein, and L. M. Ottenstein, "The program dependence graph in a software development environment," Proceedings of the ACM SOGSOFT/SIGPLAN Software Engineering Symp. on Practical Software Development Environments, pp.177-184, April, 1984.

[15] T. Reps, S. Horwitz, M. Sagiv, and G. Rosay, "Speeding up slicing," Proc. of Second ACM Conf. on Foundations of Software Eng., pp.11-20, Dec. 1995.

[16] G. Rothermel, and M. J. Harrold, "Selecting tests and identifying test coverage requirements for modified software," Proceedings of the ACM International Symp. on Software Testing and Analysis, pp.169-184. August, 1994.

[17] F. Tip, "A survey of program slicing techniques," Journal of Programming Language, Vol.3, No.3, pp.121-189, Sept., 1995.

[18] P. Tomella, G. Antoniol, R. Fiutem, and E. Merlo, "Flow insensitive C++ pointers and polymorphism analysis and its application to slicing," In 19th International Conf. on Software Eng., pp.433-443, May, 1997.

[19] M. Weiser, "Program slicing," IEEE Transactions on Software Eng., Vol.10, No.4, pp.352-257, July, 1984.

[20] J. Zhao, J. Cheng, and K. Ushijima, "Static Slicing of Concurrent Object-Oriented Programs," Proceedings of '96 COMSAC, 1996.

[21] 김희천, "객체지향 프로그램의 종속성 모델과 프로그램 슬라이싱을 이용한 클래스 테스팅 방법", 서울대학교, 박사학위 논문, 1997.

[22] 류희열, 박중양, 박재흥, "객체지향 프로그램 표현을 위한 객체지향 프로그램 종속성 그래프", 한국정보처리학회 논문지 제5권 제10호, pp.2567-2574, Oct., 1998.

김 운 용

e-mail : wykim@explore.kwangwoon.ac.kr

1996년 독학사 전자계산학과(이학사)

1999년 광운대학교 전산대학원 전자계산학과(이학석사)

관심분야 : 객체지향 프로그래밍 언어, 객체지향 분산 컴퓨팅 객체지향 분석 및 설계

정 계 동

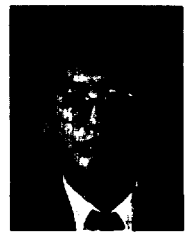
e-mail : kdjung@cs.kwangwoon.ac.kr

1985년 광운대학교 전자계산학과 (이학사)

1992년 광운대학교 산업대학원 전자계산학과(이학석사)

1992년~현재 광운대학교 전자계산학과 박사과정

관심분야 : 객체지향 프로그래밍 언어 및 데이터베이스, 병렬처리 프로그래밍 언어, 객체지향 분산 컴퓨팅



최 영 근

e-mail : ygchoi@daisy.kwangwoon.ac.kr

1980년 서울대학교 사범대학 수학교육과(이학사)

1982년 서울대학교 계산통계학과 (이학석사)

1989년 서울대학교 계산통계학과 (이학박사)

1983년~현재 광운대학교 전자계산학과 교수

1997년~현재 광운대학교 전산정보원 원장

관심분야 : 병렬 컴파일러, 병렬 프로그래밍 언어, 객체지향 프로그래밍언어, 객체지향 분산 컴퓨팅