객체지향 프로그램에서 이벤트 추상화 표현

임 근[†] · 이 경 환[††]

요 약

객체지향 언어가 가지고 있는 다양한 개념은 강력한 프로그램 구현을 지원할 수 있다. 그러나 이러한 개념에는 복잡한 이벤트의 관련성에 의해 프로그램의 분석과 이해에 어려움을 가지고 있다. 특히 객체지향 언어의 정적인 면보다는 동적인 측면의 이해를 어렵게 한다. 동적인 면은 클래스간 이벤트 작용을 인식하므로서 이해를 지원한다. 따라서 본 논문에서는 객체지향 프로그램의 이해를 지원할 수 있도록 이벤트 추상화 표현을 제시한다. 또한 클러스터링 개념을 이벤트 추상화에 적용하여, 객체지향 언어의 이해를 용이하게 지원할 수 있도록 이벤트 추상화 표현과 이벤트 추상화에 적용될 클러스터링 개념을 제시한다. 이벤트의 클러스터링에 의해서 사용자는 클래스의 기능성 정보와 클래스 라이브러리 검색시 선택된 클래스와 이벤트 상호작용 관계가 있는 다른 클래스를 파악함으로서 클래스 검색의 효율성을 지원한다.

Abstract Representation of Events on Object-Oriented Programs

Keun Lim[†] · Kyung-Hwan Lee[††]

ABSTRACT

The concepts of class, inheritance and information hiding and so on provide the great strengthes of object-oriented languages, but they also introduce difficulties in porgram analysis and understanding. Particulary, it is more difficult to understand the dynamic aspects than the static ones of object-oriented programs. The dynamic aspects can be understood by recognizing the event's reciprocal action among the classes. In this paper, it will be supplied to the representation of event abstraction which is useful for understanding the object-oriented programs. And the clustering concept with the events will be applied to abstract the events. By clustering the events, user can get the information about the function of the classes and the retrieval of the class library.

1. 서 론

소프트웨어 생산성 향상을 위한 전단계는 대상이 되는 소프트웨어의 전반적인 구조에 관한 정확한 이해라고할 수 있으며, 좀더 효과적인 방법으로 원시코드 분석을 통한 이해과정을 연구해 왔다[1]. 또한 이러한 과정을 자동화하고 여기에서 얻어진 정보를 새로운 시스템에 적용하기 위하여 역공학에 대한 연구가 진행되고 있다. 기존에 절차위주 언어의 개념과는 달리 객체지향 프로그램 환경에서는 상속과 클래스 개념에 의한 정보은닉으로 인하여 복잡한 이벤트의 관계가 발생하게 된다. 즉 객체지향 프로그램에서는 절차위주 언어에 비하여 모듈화가 간편하지만 객체 상태를 표현하는 방법이 다양하고 복잡하므로, 개발 초기의 목적과는 달리 생산성 향상에 더 많은 비용과 시간이 소요되는 비합리적 요소가 나타난다. 따라서

† 종신회원:중앙대학교 컴퓨터공학과 박사과정
†† 비 회 원:중앙대학교 컴퓨터공학과 교수
논문접수:1996년 12월 31일, 심사완료:1997년 5월 15일

본 논문에서는 객체지향 방법에서 생성되는 이벤트들의 추상화 표현을 통해서 객체의 정적인 면보다는 동적인 면을 사용자에게 보이므로서 효과적인 소프트웨어의 이해 지원과 더불어 재사용 및 소프트웨어의 생산성 향상을 도모하고자 한다. 2장에서는 관련 연구를 설명하고, 3장에서는 이벤트의 추상화 표현 방법을 제시하며, 4장에서 알고리즘의 적용 예와 구현 사항을 기술하며, 5장과 6장에서 평가와 결론을 각각 기술한다.

2. 관련 연구

관련 연구로서 시스템의 개괄적인 구조를 이해하기 위해 클러스터링을 적용한 방법과 객체지향 시스템의 이해를 지원하는 연구 내용을 대상으로 하였다. 추상화 관련 연구로는 분산환경에서 각 프로세스간 이벤트의 상호작용을 중심으로 이벤트의 추상화, 절차 언어 프로그램 모듈화와 추상화, 객체지향 언어에서 클래스간 상호작용에 의한 추상화가 있다[2, 3]. 기존의 시스템을 위한 분석과 이해 재사용을 위한 도구들이 개발되었으나, 기존 절차 위주언어와는 달리 객체지향 언어는 분석과 이해가 용이하지 않다. 즉, 객체지향언어가 가지는 상속관계, 정보은닉에 따라 제약이 가해짐으로서, 클래스로 인해 모듈화된 프로그램이 멤버함수에 의해 더 복잡한 관계를 유도하게 된다. 예를 들어 다른 클래스에 정의된 각각의 메소드가 서로 같이 적용하여 하나의 기능을 수행할 경우 이들의 상호작용을 파악하기 어렵다.

기존 시스템에서 얻어지는 지식은 새로운 시스템의 개발에서 재사용되어 소프트웨어 시스템에 대한 이해가 필수적이며 이를 위해서 가장 많이 참조되는 것은 원시코드라고 할 수 있다. 디자인 명세 문서와 사용자 지침서가 있을지라도 소프트웨어 유지보수자들은 소프트웨어의 실제 기술서와 같은 원시 코드에 많이 의존한다. 그 이유는 완전한 소프트웨어 시스템이 존재할지라도 문서는 구현된 시스템과는 독립적으로 관리되어지는 경우가 많고 그와 동시에 해당 소프트웨어의 구조와 내용에 관한 완전한 문서를 찾기 어렵기 때문이다. 또한 소프트웨어 시스템의 구조와 컴포넌트를 이해하기 위해 많은 원시코드 리스트를 분석하는 것은 대단히 소모적인 작업이다. 역공학에서는 소프트웨어를 이해하기 위해 불필요한 자세한 구현 사항으로부터 추상화하여 상위 단계로 재구성함으로서 유지보수와 재사용을 지원하도록 한다. 다음은 본 논문에 근거하는 관련된 연구를 기술하도록 한다.

- Rigi 시스템[8]

이것은 사용자와의 대화형식으로 시스템의 이해를 지원하는 도구이다. 분석하고자 하는 시스템을 4가지 표현 뷰로 나타내며 시스템의 전반적 구조에 대한 이해를 지원한다. 이 시스템은 응용 프로그램의 초기 호출 그래프를 그래프 데이터 베이스에 저장한다. 합성 오퍼레이션과 서브 시스템 식별을 이용하여 서브시스템과 상속 관계는 초기 호출 그래프상에 구성된다. 이 시스템에 사용된 클러스터링 방법은 프로그램의 중요 엔트리 포인트를 찾는 것이다. 그리고 이 노드로 부터 도달 가능한 모든 루틴의 의존도 트리를 생성하고 중심 컴포넌트를 식별한다. 관련된 노드의 집합은 파일로 구성되고, 그 파일은 서브시스템 형성에 이용된다. 이 과정은 상호간의 의존도와 새로운 서브시스템 생성을 위해 반복된다.

- Wim De Pauw[9]

객체 지향 시스템의 이해와 코드의 재사용을 위해서는 동적인 행위를 이해해야만 한다. Wim은 객체 시스템에서 행위의 동적인 면을 시각화한 방법이다. 이 방법은 시스템의 행위를 시각화하는 것으로 여러 가지 뷰의 집합을 사용한다. 이러한 뷰는 분산구조를 취하며, 종류로는 클래스와 객체들간의 상호작용, 행위의 단계와 클래스 인스탄스를 보여주는 히스토그램과 클래스의 클러스터링 과정을 보여주는 클러스터 뷰를 가지고 있다. 그리고 inter 클래스와 intra 클래스 참조관계의 형태를 식별하고 교차참조를 지원하는 뷰로 구성되며, 이러한 뷰들을 메트릭스의 형태로 보인다.

- Sniff[3]

이 방법은 브라우징, 교차참조, 디자인 시각화, 문서화, 편집기능을 제공하는 개발환경이다. 문자와 시각화된 정보로 많은 소프트웨어 시스템을 브라우즈하고 효과적인 C++ 프로그램을 생성하도록 한다. 정보추출기와 프로그램 환경으로 구성되어 있으며, 원시코드로 부터 선언과 정의에 대한 정보를 추출하며,

원시 코드에서 정의된 심볼들에 대한 개괄적인 정보를 얻도록 한다. 다만 이 방법은 시각화에 기반한 방법으로 추상화의 개념은 고려하고 있지 않다.

– Phillip A. Hausler[10]

함수 추상화의 배경은 데이터 분석, 프로그램 슬라이싱, 패턴 메칭과 같은 함수 이론의 개념을 기본으로 한다. 추상화의 입력은 구조적인 프로그램으로서 수학적인 속성을 이용하여 함수의 상위 단계의 프로그램 논리를 추출할 수 있도록 한다. 함수 추상화를 자동화하는 과정은 프로그램을 구조적인 프로그램으로 재구성하고, 데이터 분석을 통하여 변수들을 지역변수화하고 이를 기반으로 프로그램 슬라이싱, 패턴 메칭의 개념을 적용하는 과정을 거친다. 즉 추상화의 개념에는 충실한 배경을 가지고 있으나, 객체지향 방법에서 제시하는 클래스의 개념과 상호관계성을 표시하는 부분은 미약하다고할 수 있다.

3. 이벤트의 추상화 표현

3.1 연구 목적

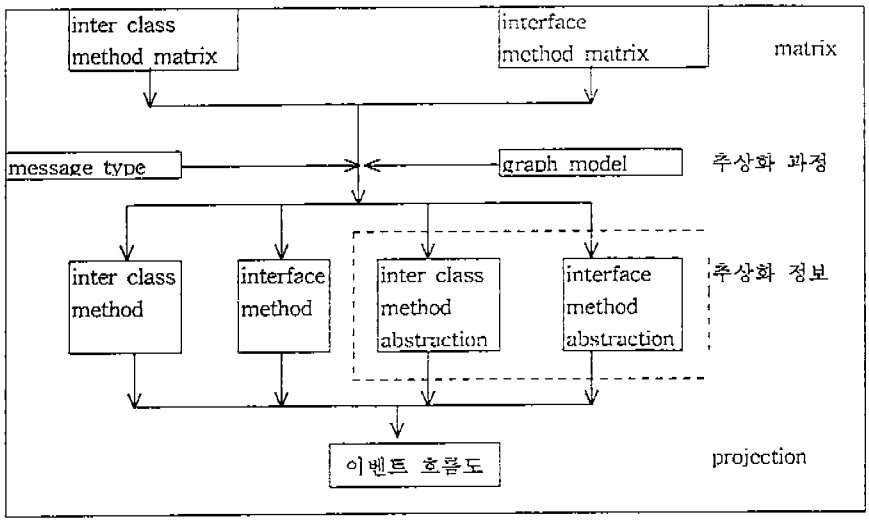
기존의 시스템을 위한 분석과 이해 재사용 도구들이 많이 개발되었으나, 기존 절차 위주언어와는 달리 객체지향언어는 분석과 이해가 용이하지 않다. 즉, 객체지향언어가 지니는 상속관계, 정보은닉에 따라 제약이 가해짐으로서, 클래스로 인해 모듈화된 프로그램이 멤버함수에 의해 더 복잡한 관계를 유도하게 된다. 따라서 객체지향 언어의 이해를 용이하게 지원할 수 있도록 이벤트 추상화 표현과 이벤트 추상화에 적용될 클러스터링 개념을 제시한다. 이벤트의 클러스터링에 의해서 사용자는 클래스의 기능성 정보와 클래스 라이브러리 검색시 선택된 클래스와 이벤트 상호작용 관계가 있는 다른 클래스를 파악함으로서 클래스 검색의 효율성을 지원한다. 또한 객체지향 프로그램의 동적인 면을 시각화하므로서 메시지의 복잡한 관계를 간단, 명료하게 표현하며, 유지보수를 용이하게 진행할 수 있도록 한다.

이러한 목적을 위해서 이벤트의 추상화된 표현방법을 사용하여 객체지향 프로그램의 재사용과 유지보수를 지원하도록 원시코드를 클래스 표현으로 재구성한다. 또한 객체지향 프로그램의 정적인 면보다는 동적인 면의 중요성을 인식하고 이를 이해하기 위한 방법으로 각 객체들간 이벤트의 커뮤니케이션 관계와 객체 내부의 이벤트의 흐름을 보여주는 이벤트 흐름도(Event Folw Diagram)를 재구성한다. 이벤트 흐름도에서 내부 메소드간 이벤트와 각 객체간 이벤트를 추상화 방법을 적용하여 나타낸다. 또한 프로그램의 원시코드 수정시 이와 관련된 이벤트의 전달 관계를 시각화하여 원시코드의 수정시 발생하는 제약성을 사용자가 보다 쉽게 해결하도록 한다.

3.2 이벤트의 추상화

분석과 식별의 대상이 되는 객체지향 시스템의 동적요소를 정리하면 메소드, 내부 메소드, 인터페이스 메소드, 이벤트 및 메시지이다. 클래스에서 정의된 행위 또는 함수를 메소드라고하며 그중 한 클래스에서만 관계가 있는 메소드를 내부 메소드로 식별한다. 또 다른 요소인 인터페이스 메소드는 다른 클래스의 메소드를 호출하거나 호출되는 메소드이며, 이벤트나 메시지는 메소드들간 호출관계와 한 클래스 내에서나 클래스들간 호출관계에 대응하는 요소이다. 시스템의 동적 정보는 본 논문에서 제안하는 그래프 모델의 적합여부에 의해 추출된다. 따라서 추출된 정보는 하나의 클러스터로 클러스터링이 되어 추상화된 형태로 시각화된다. 추상화 기법은 메소드의 호출관계의 그래프 형태에 의한 클러스터링으로서 기능 중심의 클러스터링이 될 수 있고, 이를 직접 사용자에게 표현함으로서 메소드의 클래스들간 기능과 역할을 표현할 수 있다.

객체지향 시스템의 동적 특성, 특히 객체들간의 이벤트 전송단계의 추출을 위해 객체지향 시스템을 제안하는 추상화 기법과 절차에 따라 분석하게 된다. 추상화 방법을 개념적으로 살펴보면 (그림 1)과 같다. 원시코드를 입력으로 받아들이면 이를 분석한 정보가 메트릭스에 저장된다. 이때 메소드의 호출관계와 메시지의 타입을 분류하고 이를 메트릭스에 저장한다. 이때 내부 메소드는 내부 메소드 메트릭스에 저장되고 클래스간의 인터페이스 메소드는 인터페이스 메트릭스에 저장된다. 객체지향 프로그램의 분석시 메소드간 또는 클래스간 메시지 전달 관계를 이해하고 이들 사이의 복잡도를 줄이기 위하여 클러스터링을 적용하여 이벤트를 추상화시킨다. 따라서 메트릭



(그림 1) 이벤트의 추상화 표현 모델
(Fig. 1) Abstract presentation model of event

스에 저장된 정보를 본 논문에서 정의한 그래프 모델을 방법에 의해서 추출하여 이를 저장한다. 추출하고자 하는 그래프 모델은 평행관계, 연쇄관계 그리고 조건관계, 순차관계이다.

시각화된 정보는 인터페이스 메소드를 추상화하는 단계, 클래스간 인터페이스 메소드와 이벤트와의 관계, 내부 메소드를 추상화하는 단계, 모든 내부 메소드와 인터페이스 메소드의 호출관계 순으로 원시코드의 전역적인 정보에서 상세화 과정을 거쳐 표시한다.

3.3 메트릭스의 정의

원시 코드에서 분석된 정보는 내부 메소드 메트릭스와 인터페이스 메소드 메트릭스로 분류되어 저장된다. 메트릭스는 행과 열을 이용하여 호출하는 메소드와 호출되는 메소드의 관계를 효율적으로 나타낼 수 있으므로 메트릭스를 사용하여 호출관계를 저장한다.

Let matrix M = a [i, j] be n * m matrix
  Matrix Definition
    if a [i, j] ≠ 0 then
      [0, j] is callee of [i, 0]
      [i, 0] is caller of [0, j]

위와 같은 내용은 다음 관계를 유도한다.

i = {1, ..., n}, K = {i, n | a [i, n] = true} 일때
  N(K) is outdegree (callers) of a [i, 0]

j = {1, ..., m}, K = {1, j | a [1, j] = true} 일때
  N(K) is indegree (callees) of a [0, j]

a[i, j] = true ∧ a [j, i] = true 일때
  a [i, j] ≠ a [j, i]

즉 열은 호출된 메소드를 나타내며 행은 호출하는 메소드를 나타낸다. 예를 들면 a→b는 메소드 a를 호출한 다음 b 메소드를 호출하는 관계를 나타낸다고 할 때 〈표 1〉에서 나타난 것과 같이 a→b→c의 호출관계를 나타낸다. 내부 메소드 메트릭스 정보는 내부 메소드의 클러스터링으로 표현하고, 〈표 2〉와 같이 인터페이스 메소드 메트릭스 정보는 클래스의 클러스터링으로 이용된다. 그리고 a→b≠b→a이므로 비대칭적이다. 메트릭스의 가로 열은 호출하는 함수를

나타내며, 세로 열은 자신을 호출하는 메소드를 나타낸다. 메트릭스는 이벤트 타입과 함께 기술되며 시점과 종점이 같을 경우 메시지의 타입이 다를 수 있으므로 다른 cell에 기술된다.

〈표 1〉 내부 메소드
〈Table 1〉 Inter method

	a	b	c	d
a	0	1	0	0
b	0	0	1	0
c	0	0	0	0
d	0	0	0	0

〈표 2〉 인터페이스 메소드
〈Table 2〉 Interface method

	A:a	B:b	C:c	D:d
A:a	0	1	0	0
B:b	0	0	0	0
C:c	0	1	0	0
D:d	0	0	1	0

〈표 3〉은 이벤트 추상화 표현시 사용될 타입을 정의하였다. 이벤트 타입은 메소드의 호출순서와 함께 메트릭스에 저장됨으로서 그래프 모델의 추출시 조건으로 사용된다.

〈표 3〉 타입 정의
〈Table 3〉 Type definition

표시 형식	내 용
i	t :조건문에 의한 분기 메소드 f :조건문에 의한 비분기 메소드
smt	s :이벤트의 시점 메소드 m :이벤트의 중간 경로 메소드 t :이벤트의 종점 메소드
r	t :함수 반환값이 있는 메소드 f :함수 반환값이 없는 메소드

메트릭스의 cell의 구조는 (그림 2)와 같다. 호출순서는 메소드에서 호출되는 순서를 나타내고 이벤트의 타입은 i, r은 true와 false로 나타낸다. 예를 들면 i가 true이면 조건문에 의해 분기되는 메소드이며 false이면 조건문에 분기되지 않는 메소드이다.

호출순서		이벤트 타입		
seq#	if_else	i	r	smt

(그림 2) cell의 구성
(Fig. 2) Structure of cell

3.4 추상화 정보의 정의

메트릭스에 저장된 정보에서 클러스터링을 하기 위해서 그래프 모델을 추출한다. 이때 정점은 내부 메소드이거나 클래스이며 간선은 이벤트를 나타낸다. 간선의 방향은 호출관계를 나타낸다. (그림 3)은 추출되는 그래프 모델을 나타내며 (그림 4)는 추출된 그래프 모델이 클러스터링되어 이벤트가 추상화된 것을 나타낸다. 생성된 정보를 내부 메소드 메트릭스로 구성하고 이를 기반으로 인터페이스 메소드와 내부 메소드의 추상화 정보, 인터페이스 메소드의 추상화 정보를 생성할 경우 메트릭스를 구성하는 규칙을 정형화하여 표현한다. 내부 메소드에서는 평행관계, 연쇄관계 그리고 순차관계 3가지의 그래프 모델을 적용하여 클러스터링을 하며 인터페이스 메소드로 객체들을 클러스터링할 때는 연쇄관계와 순차관계만이 적용된다. 클러스터링시 충돌이 발생할 경우 포함 관계와 교집합 관계 2가지로 분류할 수 있다.

－포함 관계

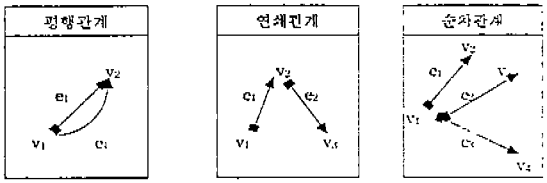
그래프 모델에서 평행관계, 연쇄관계 그리고 순차관계 중에서 2가지 이상의 관계를 만족하는 경우에 해당한다. 2가지 이상의 관계가 만족될 경우 다음의 우선 순위에 의하여 클러스터링이 되며 우선 순위는 (평행관계＜연쇄관계＜순차관계) 이다.

－교집합 관계

해당 그래프 모델에 속하는 어떤 정점이 또 다른 그래프 모델을 만족하게 되는 경우이다. 이때 우선 순위가 높은 관계가 먼저 클러스터링이 되며 클러스터링의 교집합 여부를 식별할 수 있도록 교집합 표시를 한다.

추출된 그래프 모델로 클러스트링을 한 정보를 〈표

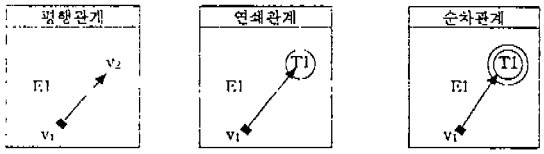
4)와 같은 형태로 저장한다. 내부 메소드에서는 평행관계, 연쇄관계, 순차관계로 클러스터링될 수 있다. 시작점은 호출하는 메소드의 이름이며 종점은 호출되는 메소드의 이름이다. 그리고 first는 순차관계와 조건관계에서 첫번째로 호출하는 메소드이며 last는 마지막으로 호출되는 메소드이다. 인터페이스 메소드 정보는 평행관계와 연쇄관계, 순차관계로 클러스트링 되므로 이에 관련된 정보를 갖는다. 인터페이스 메소드에서는 호출하는 메소드의 이름과 그 메소드를 사용하는 클래스의 이름도 함께 저장된다. 그리고 내부 메소드, 인터페이스 메소드, 클러스터링 정보는 모두 자신이 클러스터링되는 그래프 모델의 이름도 함께 저장된다.



(그림 3) 그래프 모델
(Fig. 3) Graph model

〈표 4〉 추상화 정보의 정의
〈Table 4〉 Definition of abstract information

	내부 메소드	인터페이스 메소드
평행관계	source, destination, parallel	source class name :: method destination class name :: method, parallel
연쇄관계	source, destination, chain	start class name :: method, stop class name :: method, chain
순차관계	first, last, serial	first class name :: method. last class name :: method, serial



(그림 4) 그래프 모델의 클러스터링
(Fig. 4) Clustering of graph model

평행관계 즉, 정점 v_1과 정점 v_2 사이에 두개 이상의 간선이 존재한다면

	정 의
e_1	v_1 -> v_2
e_2	v_1' -> v_2'
E1	e_1 + e_2

으로 간선 e_1과 e_2는 추상화되어 E1으로 표현된다. 연쇄관계, 정점 v_1과 v_2, v_3가 연쇄관계에 있다면 v_1 -> v_3로의 간선을 e_2라 하고 T1을 v_2 + v_3로 추상화 하면 e_1과 e_2는 E1으로 추상화 된다.

	정 의
e_1	v_1 -> v_2
e_2	v_1 -> v_3
T1	v_2 + v_3
E1	v_1 -> T1
E1	e_1 + e_2

순차관계, 즉 간선 e_1, e_2, e_3가 순차관계에 있을 때 v_2, v_3, v_4를 T1으로 추상화하면 e_1, e_2, e_3는 E1으로 추상화 된다.

	정 의
e_1	v_1 -> v_2
e_2	v_1 -> v_3
e_3	v_1 -> v_4
T1	v_2 + v_3 + v_4
E1	v_1 -> T1
E1	e_1 + e_2 + e_3

내부 클래스 메소드를 기반으로 하여 인터페이스 메소드 메트릭스를 생성한다. 내부 클래스 메소드 메트릭스의 정보중에서 이벤트의 시작점을 찾는다. 왼쪽 행에 있는 메소드가 호출하는 메소드를 알기 위해 열을 검색한다. 수평방향의 메소드는 가장 왼쪽의 메소드가 호출하는 메소드이며 수직 방향의 메소드는

가장 상위의 메소드를 호출하는 메소드이다. 각 cell을 검사하여 정보가 들어 있다면 해당 정보를 호출 순서에 따라 연결 리스트에 저장하고, 동시에 정보의 caller와 callee에 관한 정보도 함께 저장한다. 연결 리스트에 저장된 정보의 순서대로 추적하면서 이벤트 타입과 caller, callee를 참조하여 그래프 모델에서 추출한 정의의 만족 여부를 판단하고 이를 기반으로 추상화 정보를 생성한다. 생성된 추상화 정보는 미리 정의한 형식에 의해 저장되고 이를 표현할 경우 추상화 정보는 연결 리스트를 참조하면서 화면에 시각화한다. 또한 추상화되어 표현된 메소드나 클래스의 이름은 내용에 관계없이 순차적으로 정해지며 도큐멘트 뷰에서 추상화 되어진 정보에 대한 내용을 표현한다. 도큐멘트 뷰에서는 추상화된 메소드나 클래스 이름과 추상화 형태를 표현하게 된다. 이와 같이 추상화를 통하여 얻을 수 있는 장점은 사용자가 시스템에서 동시에 발생되는 다양한 동작(연산)을 볼 수 있으며, 이벤트를 추상화함으로서 사용자가 시스템의 동적인 면의 복잡도를 시각화함으로서 모호성을 감소할 수 있다. 또한 객체간 메시지의 교환관계를 시각화함으로서 객체들의 역할을 분명하게 알 수 있고, 객체들간 역할을 파악하게 됨으로서 시스템을 객체 단위의 재사용뿐만 아니라 서브 모듈로의 재사용도 가능하다. 상속관계에 있어서는 객체들의 상속된 메소드를 평이한 구조로 처리하고, 실제로 메소드를 가지고 있는 객체와 그 메소드를 상속받는 메소드로 구분할 수 있다.

4. 알고리즘 및 구현

4.1 알고리즘

내부 메소드 메트릭스를 기반으로 인터페이스 메소드 메트릭스를 생성한다. 다음 이들 두가지 정보를 추상화의 입력으로 사용하며, 이후에 추상화 규칙을 적용하여 정보를 생성한다. 이때 주로 사용되는 데이타의 구조는 메소드, 이벤트 타입, 호출 순서, callee의 포인터를 저장할 포인터 배열을 갖는 메소드 리스트와 추상화 정보를 저장하는 추상화 정보 리스트 등이 있다. 이때 메소드 리스트와 추상화 정보 리스트는 연결 리스트이다.

다음은 추상화 기법을 적용하여 내부 메소드를 클러스터링시킨 다음 이벤트를 추상화시키는 과정이다. (그림 5)의 입력된 원시코드 리스트를 분석한다. 분석된 정보를 (그림 6)와 같이 내부 메소드 메트릭스와 인터페이스 메소드 메트릭스에 저장한다. 저장된 정보에서 그래프 모델을 추출하여 클러스터링 한다. 클러스터링한 저장 정보 타입은 (그림 7)과 같다.

```
class FileAction : Virtual public StrAction
virtual public Border{
        LineAction *Crt ;
        FileAction *next ;
        int cx, cy ;
public :
        void Dpl (LineAction *p, int) ;
        void Dpl_Ln( ) ;
        void Dpl_Sc( ) ;
}
void FileAct::Delact
{
int length1, length2 ;
int i ;
  if(currentcol < currentline → length)
        save = 0 ;
        Crt→del(currentcol) ;
        Dpl(currentline, cy) ;
        Dpl_Ln( ) ;
  else
        Dpl_Sc( ) ;
}
void Dpl_Sc( ) ;
{
  Dpl(currentline, cy) ;
  Dpl_Ln( ) ;
  Space( ) ;
  Move To( );
}
```

(그림 5) 원시 코드 리스트
(Fig. 5) Source code list

	DelAct		Crt→del		Dpl		Dpl_Ln		Dpl_Sc		Space		Moveto	
DplAct			111	tft	112	tft	111	tft	2E1	tfm				
Dpl_Sc					1	tft	2	tft			3	fft	4	fft

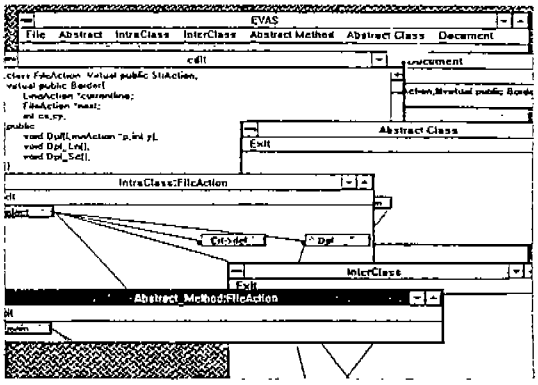
(그림 6) 클래스 FileAction에 대한 내부 메소드 메트릭스
(Fig. 6) Inter method matrix to class FileAction

T1	Dpl	Moveto	serial

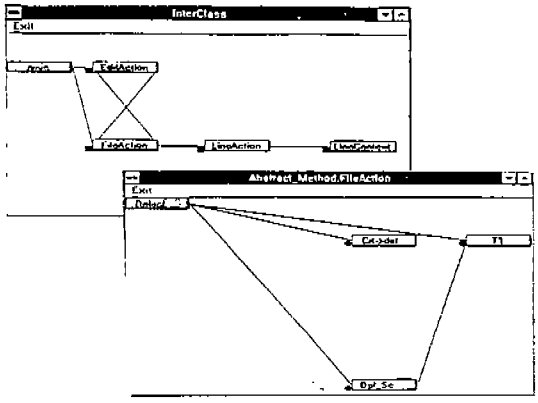
(그림 7) 클러스터링 저장 정보 타입
(Fig. 7) Clustering stored information type

4.2 구 현

이벤트를 추상화한 결과를 시각화 하였다. 최초 화면은 이벤트 추상화 도구의 화면이고, 이어서 내부

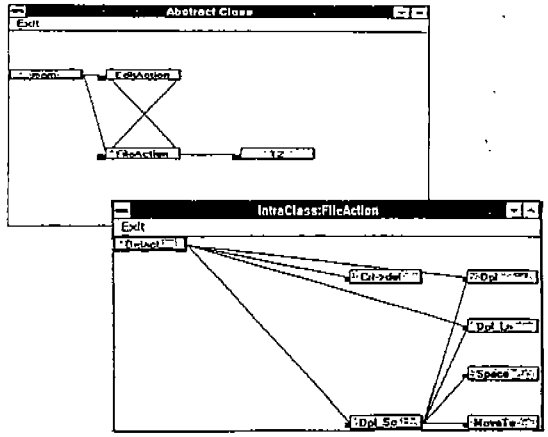


(그림 8) 이벤트 추상화 도구의 화면
(Fig. 8) Window of event abstract tool



(그림 9) 내부 클래스와 추상화 메소드
(Fig. 9) Inter class and abstract method

메소드의 이벤트 관계를 표시한 화면이며, 내부 이벤트의 추상화와 클래스간 이벤트 관계 및 클래스간 이벤트 추상화의 화면이다.



(그림 10) 추상화 클래스와 내부 클래스(내부 메소드의 이벤트 관계)
(Fig. 10) Abstract class and intra class (event relation of intra class)

5. 평 가

본 논문에서는 Sniff 시스템, Rigi 시스템과 Wim, Phillip이 제안한 연구를 다음의 관점에서 비교평가 하였다.

- 정적인 면의 이해 측면
- 동적인 면의 이해 측면
- 시각화 방법의 적용
- 추상화 방법의 적용
- 객체지향 프로그램 지원

정적인 면의 이해의 측면은 본 연구실에서 개발한

〈표 5〉 시스템별 평가표
〈Table 5〉 Evaluation table of each system

	Sniff	Rigi	Wim	Phillip	본 논문
정적인 면	Class hierarchy			프로그램 함수의 수학적 논리 이용	
동적인 면		함수 호출관계의 다이아그램이용	함수 호출관계 메트릭스 이용		메소드와 클래스간 이벤트간 관계를 표현
시각화 방법	symbol browser inheritance rel. browser, class browser	text window, editor, abstract window	instance histogram, cluster matrix		inter method event, interface method event, inter method event, inter method abstraction, editor, documentation view
객체지향 지원			객체지향프로그램지원		객체지향프로그램지원
추상화 방법		컴포넌트 클러스터링	클래스 클러스터링	program slicing, pattern matching	클래스간 이벤트, 클래스내의 이벤트 관계

재사용 시스템 CARS 2.1[11]에서 지원하고 있다. 다만 위의 〈표 5〉와 같이 추상화 방법과 시각화 방법은 대분분의 연구에서 적용하고 있다. 이것은 추상화 방법이 프로그램의 이해 측면을 지원하고 있는 요소임을 확인할 수 있으며, 추상화 방법에 있어서 Wim의 방법은 서로 상호작용이 많은 클래스들을 추상화 시킴으로서 객체지향 프로그램을 몇 개의 클래스 모듈로 분할은 가능하지만 시스템의 동적인 면의 이해도를 저하시킬 수 있다. 따라서 본 논문은 추상화 방법에서 모듈화의 기능은 Wim의 방법에 비하여 미흡하다고 볼 수 있으나, 추상화시 이벤트의 흐름을 파악할 수 있으며, 이로 인하여 클래스 내부에서의 이벤트의 흐름과 시스템을 이루고 있는 클래스들의 상호작용을 이해할 수 있다. 또한 시각화 방법에서 Wim은 메트릭스이며, 본 논문에서는 이벤트를 나타내고 이때 시각화된 이벤트 흐름도가 메트릭스보다 사용자 이해 측면은 우수한 것으로 판단된다.

6. 결 론

소프트웨어 생산성 향상을 위하여 소프트웨어 재사용 방법이 연구되고 있다. 재사용에 앞서 재사용 대상이 되는 소프트웨어의 정확한 이해를 지원할 수 있어야 하며, 이에 대한 많은 도구가 개발되었다. 다만 객체지향 프로그램의 기본 개념인 상속관계, 정보은닉 등의 개념을 이용하여 프로그램 구현시 많은 성능 향상의 효과를 가져왔다. 다만 구현된 객체지향 프로그램 재사용시 상속관계, 정보은닉, 다형성 문제등에서 제시되는 다양한 이벤트 관계를 분석하고 이해하는데 많은 어려움을 내포하고 있다. 따라서 본 논문에서는 객체지향 프로그램의 특징인 객체간 상호작용을 표현하고 객체내부의 이벤트의 흐름을 나타내는 이벤트 흐름도를 재구성하였으며, 이때 이벤트의 추상화를 적용하였다. 이것은 객체들간 상태 변환을 가져오는 이벤트들의 복잡한 구조에서 기인되므로 이를 감소시키기 위하여 각 클래스들의 기능과 클래스들의 상호작용을 파악하고자 하였다. 또한 객체간 메시지의 교환관계를 시각화하여 각 객체의 역할을 분명하게 알수 있으며, 이벤트를 추상화함으로서 사용자가 시스템의 동적인 면의 복잡도를 시각화시킬 경우 나타날 수 있는 모호성을 감소할 수 있다. 추상화된 정보를 적용한 이벤트 흐름도를 통해서 프로그램에 대한 전반적인 이해도를 증진시키고 클래스 라이브러리의 검색시 선택된 클래스와 이벤트 상호관계에 있는 다른 클래스들을 사용자가 쉽게 파악하므로서 클래스의 검색 효율성을 높이게 하였다.

재사용 측면에서는 객체들간 역할을 파악하게 됨으로서 시스템을 객체 단위의 재사용뿐만 아니라 서브 모듈로의 재사용도 가능하며, 프로그램의 유지보수시 수정하고자 하는 메소드에 영향을 받는 메소드를 쉽게 인지함으로서 프로그램의 수정시에 발생하는 모호성을 감소할 수 있다.

참 고 문 헌

[1] 이 경환, 소프트웨어 공학 개론, 희중당, 1994.

[2] Thomas Kunz, "Reverse_Engineering Distributed Applications to Understand their Behavior", Technical Report TI-3/94, Institute fur theortische Informationtik Fachbereich Informatik Technische Hochschude Darmstadit, April, 1994.

[3] G. caldiera, V. R. Basili, "Identifying and qualifying Reusable Software Components", IEEE Computer, Vol. 24, No. 2, Feb., pp. 61-70, 1991.

[4] Scott R. Tilley, Hausi A. Muller, Mehmet A. Organ, "Docdumenting Software System with Views", Proceedings of 10th International Conference on Systems Documentation pp. 211-219, 1992.

[5] Bent Bruun Kristenesen, "Complex Associations: Abstractions in Object-Oriented Modeling", OOPSLA 94, pp. 272-283, 1994.

[6] Norman Wilde, "Maintaing Object-Oriented Software", IEEE Software, January, pp. 75-80, 1993.

[7] 이 경환, 소프트웨어 재사용을 위한 객체 모델링 기법, 교학사, 1993.

[8] Teisseire, "Dynamic Modeling with Events", OOIS94, pp. 166-199, 1994.

[9] Wim De Pauw, "Visualizing the Behavior of Object-Oriented Systems", OOPSLA 93, pp. 326-337, 1993.

[10] Phillip A. Hausler, "Using Function Abstraction to Understand Program Behavior", IEEE software, pp. 50-63, 1990.

[11] 과기처, 중앙대학교, "소프트웨어 재이용에 관한 연구", 1990.

임 근

1989년 2월 중앙대학교 컴퓨터공학과 졸업(공학사)

1991년 2월 중앙대학교 컴퓨터공학과 졸업(공학석사)

1992년 3월~현재 중앙대학교 컴퓨터공학과 박사과정

관심분야:S/W 공학, 객체지향 방법론, 재사용 방법론, 정보검색등

이 경 환

1980년 중앙대학교 대학원 응용수학 전공(이학박사)

1982년~1983년 미국 Aurban대학 객원교수

1986년 서독 Bonn대학 객원교수

1971년~현재 중앙대학교 컴퓨터공학과 교수

관심분야:소프트웨어 공학, 객체지향 모델링, 소프트웨어 재사용