Alpha : 자바 시각화 도구

김철민†

요 약

웹 환경과의 연동, 병렬 프로그래밍, 객체지향 프로그래밍, 안전성(컴파일 시간 에러 검출, 예외 처리 기법)과 이식성, GUI 등을 지원함으로써, 프로그래밍 언어 자바(Java)의 활용도가 점점 증가하고 있다. 객체지향 프로그래밍 언어로서 자바는 클래스, 인스턴스, 정보은닉, 상속, 다형성 등 객체지향 개념들에 기반하고 있다. 그러나 자바 프로그램 수행시 이들과 관련된 제반 현상이 자바가상기계(*Java Virtual Machine*) 내부에 감추어지기 때문에, 자바 언어를 배우거나 활용하는 대부분의 사용자들이 큰 어려움을 겪는다. 이 문제에 대한 효과적 해결책으로서, 본 연구에서는 자바가상기계 내부의 현상을 객체지향 개념과 연계시켜 시각화해 주는 도구 *Alpha*를 개발하였고, 본 논문은 그 설계와 특징들을 기술하고 있다. *Alpha*는 실용성과 확장성을 높이기 위해 MVC(*Model-View-Controller*) 구조로 설계되었고, 이를 통해 자바가상기계 내부의 현상(인스턴스 생성 및 소멸, 메쏘드 호출 및 복귀, 필드 접근, 인스턴스 간의 상호 참조관계 변화, 쓰레드의 수행 흐름 등)을 사용자의 수준이나 활용 목적에 맞게 다양한 방식으로 시각화시켜 준다.

키워드 : 시각화, 객체지향 프로그래밍, *MVC* 패러다임

Alpha : Java Visualization Tool

Cheol-Min Kim†

ABSTRACT

Java provides support for Web, concurrent programming, safety, portability, and GUI, so there is a steady increase in the number of Java users. Java is based on the object-oriented concepts such as classes, instances, encapsulation, inheritance, and polymorphism. However the JVM(*Java Virtual Machine*) hides most of the phenomena related to the concepts. This is why most of Java users have much difficulty in learning and using Java. As a solution to the problem, I have developed a tool ***Alpha*** that visualizes the phenomena occurred in the JVM from the standpoint of the concepts and will describe the design and features of the tool in this paper. For practicality and extendability ***Alpha*** has an MVC(*Model-View-Controller*) architecture and visualizes the phenomena such as object instantiations, method invocations, field accesses, cross-references among objects, and execution flows of threads in the various ways according to the levels and purposes of the users.

Keywords : *visualization, object-oriented programming, MVC paradigm*

1. 서 론

웹 환경과의 연동, 병렬 프로그래밍, 객체지향(*object-oriented*) 프로그래밍, 안전성(컴파일 시간 에러 검출, 예외 처리 기법)과 이식성, GUI 등을 지원함으로써, 프로그래밍 언어 자바(Java)의 활용도가 점점 증가하고 있다. 하지만 자바가 클래스(*class*), 인스턴스(*instance*), 정보은닉(*encapsulation*), 상속

† 정 회 원: 제주대학교 컴퓨터교육과 조교수
논문접수: 2004년 4월 19일, 심사완료: 2004년 5월 13일
* 이 논문은 2001년도 제주대학교발전기금 학술연구비에 의거 연구되었음.

(*inheritance*), 다형성(*polymorphism*) 등 객체 지향 프로그래밍과 관련된 고급 개념들에 기반하고 있어, 자바 언어를 배우거나 활용하는 대부분의 사용자들이 많은 어려움을 겪고 있다.

객체 지향 프로그래밍 언어는 '객체 간 상호작용'을 통해서 문제를 해결하도록 그에 필요한 언어 개념과 요소들을 제공한다. 객체의 특성을 정의할 수 있도록 클래스(*class*)라는 개념을 제공하고, 상호작용에 참여할 객체(*object*)를 원하는 클래스로부터 생성할 수 있게 지원하며, 객체 간의 상호작용 과정을 기술할 수 있도록 메쏘드 호출 등을 포함한 제어 구조를 제공한다. 자바 프로그래밍 과정은 결국 원하는 '객체 간 상호작용'이 프로그램 수행 과정에서 일어나도록, 자바 언어의 요소들을 적절히 활용·조합하여 프로그램으로 표현하는 과정이다. 자바 프로그램에 표현된 '객체 간 상호작용'은 그 프로그램이 수행될 때 실제 시스템(자바가상기계)에서 발생된다. 따라서 자바 학습자나 사용자가 표현된 '객체 간 상호작용(프로그래밍 환경에서 설정되는 객체 간 상호작용)'과 발생한 '객체 간 상호작용(프로그램 수행 환경에서의 객체 간 상호작용)'의 연계성을 명확히 이해하고 있을 때, 자바 언어의 올바른 활용과 수준 높은 프로그래밍이 가능하다. 대부분의 자바 사용자들이 겪는 어려움은 자바가상기계(*Java Virtual Machine*) 내에서 일어나는 '객체 간 상호작용'을 시각적으로 확인하기 어렵다는 데 기인한다.

본 논문은 이러한 어려움을 덜어주기 위해 저자가 개발한 도구 ***Alpha***의 구조와 기능, 그 활용 방법 등을 제시하고 있다. ***Alpha***는 자바 프로그램 수행 중 발생하는 '객체 간 상호작용'을 시각화시켜 주는 도구이며 '대화식 관찰' 도구이다. 여기서 '대화식 관찰'은, '객체 간 상호작용'의 진행상황을 관찰하는 것과, 관련 객체의 필드값 변경이나 메쏘드 호출 등을 통해 '객체 간 상호작용'에 개입하는 것을 모두 포함한 것으로, 자바 사용자들이 자바 학습이나 활용 과정에서 필요로 하는 실질적 도움을 제공한다. '객체 간 상호작용'의 시각화에 있어 ***Alpha*** 설계시 고려한 대표적 요구사항은 다음 두 가지이다.

- 시각화 대상 선정과 관련된 요구사항 : 시각화 대상(객체 간 상호작용) 설정 작업이 용이해야 한다. 개별 객체의 상태 변화 뿐만 아니라 다수 객체 간의 상호작용을 관찰할 수 있도록, 상호작용 대상으로서의 객체 그룹과 상호작용 주체로서의 쓰레드(*thread*) 그룹을 설정할 수 있게 지원해야 한다.
- 시각화 방법과 관련된 요구사항 : 시각화 방법에 확장성이 있어야 한다. 자바 프로그램 수행시 발생하는 현상을 사용자 수준이나 사용자의 활용 목적에 맞게 관찰할 수 있도록 지원해야 한다. 동일 현상을 서로 다른 시각에서 동시에 관찰할 수 있어야 하며, 새로운 시각화 방법을 구현하고 추가하는 작업이 용이해야 한다.

시각화 대상 선정과 관련된 요구사항을 만족시키기 위해, ***Alpha***는 자바가상기계 내부에 어떤 객체들이 존재하는지 사용자에게 보여주는 한편, 객체(혹은 쓰레드)들의 그룹이나 개별 객체들을 조합하여 새로운 그룹을 구성할 수 있게 지원해 준다. 또한, ***Alpha***를 MVC(*Model-View-Controller*) 구조로 설계함으로써 시각화 방법에 대한 요구사항을 수용하고 있다. MVC 구조[14]는 모델(*model* : 관찰 대상)과 뷰(*view* : 시각화 작업 수행 모듈)를 분리시켜 새로운 뷰의 개발을 용이하게 해 주는 동시에, 사용자가 모델과 뷰를 동적으로 연결하여 자신이 원하는 모델을 다양한 관점에서 관찰할 수 있게 지원한다.

2. 관련 연구

모드'는 자바 응용(*Java application*)을 수행시켜 놓고 그 수행 과정을 추적(*trace*)하면서

<표 1> ***Alpha***, BlueJ, JSwat의 특징 비교

비교 항목		*Alpha*	BlueJ	JSwat
지원 모드		응용모드와 비응용 모드	응용 모드와 비응용 모드	응용 모드
쓰레드(*thread*) 시각화		스택, 지역변수, 수행 위치 시각화	스택, 지역변수, 수행 위치 시각화	스택, 지역변수, 수행 위치 시각화
수행제어 이벤트 (수행 흐름을 정지시키는 이벤트)		정지점, 쓰레드 스텝 감시점(*watchpoint*) 참조 메쏘드 호출/복귀 쓰레드 시작/종료	정지점, 쓰레드 스텝	정지점, 쓰레드 스텝 감시점 참조 메쏘드 호출/복귀 쓰레드 시작/종료
비응용 모드	인스턴스 생성	지원함	지원함	지원하지 않음
	인스턴스 참조 범위	가상기계 내의 모든 인스턴스	동일 그룹(패키지) 내에 생성된 인스턴스들	지원하지 않음
	인위적 메쏘드 호출	지원	지원	지원하지 않음
	관찰 대상	사용자가 임의로 조합한 클래스와 인스턴스들의 그룹들	패키지 단위	지원하지 않음
응용 모드	인위적 메쏘드 호출	지원	지원	지원
	관찰 대상	사용자가 임의로 조합한 클래스와 인스턴스들의 그룹들	선택된 쓰레드의 문맥에서 참조 가능한 클래스와 인스턴스들	선택된 쓰레드의 문맥에서 참조 가능한 클래스와 인스턴스들

전통적으로 프로그래밍, 알고리즘, 자료구조 등의 교육이나 학습에는 큰 어려움이 따른다. 이는 이들 모두가 눈에 보이지 않는 (관념 속이나 시스템 내부에서 발생하는) 작업의 진행 상황을 명확히 이해하도록 요구하고 있기 때문이다. 기존의 많은 연구들[1, 2, 3, 4, 5, 9, 10]이 보이지 않는 현상에 대한 시각화 필요성과 그 효과를 기술함으로써, 시각화의 중요성을 강조하고 있다. 최근 들어 객체지향 개념의 소프트웨어 공학적 가치와 자바 활용도 제고 등으로 객체지향 프로그래밍 교육이 일반화되어 가면서, 객체 지향 프로그래밍 관련 개념이나 자바 프로그래밍 교육 분야에 시각화 도구를 활용하려는 연구들[8, 11, 12, 13]이 활발하다. 절차적(*procedural*) 프로그래밍에 비해 객체지향 프로그래밍 개념들의 추상화 정도가 훨씬 크다는 점에서, 시각화 연구의 필요성과 효과는 매우 크다. 자바 교육을 위한 시각화 도구로서 BlueJ[7]와 JSwat[6]의 기능은 주목할 만한데, <표 1>은 본 연구에서 개발한 ***Alpha***와 두 도구 간의 특징을 비교하여 보여 주고 있다. <표 1>에서 '응용 모드'는 자바 응용(*Java application*)을 수행시켜 놓고 그 수행 과정을 추적(*trace*)하면서 객체 간의 상호작용을 관찰하는 모드를 말하며, '비응용 모드'는 인위적으로 다양한 인스턴스들을 만들고 특정 인스턴스들의 메쏘드를 호출해 가면서 그들 간의 상호작용을 관찰하는 모드를 말한다. <표 1>에서 보는 바와 같이 JSwat은 자바 응용 프로그램을 디버깅하듯 제어하면서 그 수행 과정을 시각적으로 관찰하도록 지원하는 도구일 뿐, 객체 간 상호작용의 상황을 인위적으로 설정하여 관찰할 수 있게 지원하지 않는다. BlueJ는 '비응용 모드'를 지원하지만, 서로 다른 그룹(패키지) 내에 생성된 인스턴스들 간의 상호참조 관계나 상호작용을 관찰하도록 지원하지 않는다. 또한 BlueJ를 이용할 경우 수행제어 이벤트의 종류가 제한되어 있어, 특정 필드가 참조된 시점이나 쓰레드가 시작된 시점 등에 자바 프로그램의 수행을 정지시켜 두고 '객체 간 상호작용'의 세부 상태를 관찰할 수 없다. 이러한 제약점들은 BlueJ나 JSwat을 활용할 경우 '객체 간 상호작용'에 대한 포괄적 내용을 수준별로 교육하거나 학습하기 어렵다는 것을 의미한다.

3. Alpha의 구조 및 기능

3.1 JPDA

JPDA(*Java Platform Debugger Architecture*)는 J2SE(*Java 2 Platform, Standard Edition*) 환경에서 디버거 응용(*debugger application*)을 개발하는데 필요한 환경을 제공한다. JPDA는 JVMDI, JDWP, JDI라는 3 계층의 API로 구성되는데, JVMDI(*Java Virtual Machine Debug Interface*)는 디버깅을 위해 자바가상기계가 제공해야 할 서비스를 정의하고, JDWP(*Java Debug Wire Protocol*)는 디버깅되는 프로세스와 디버거(*debugger*) 간에 주고받는 정보와 요청의 형식을 정의하며, JDI(*Java Debug Interface*)는 자바 언어로 디버거를 개발할 때 활용할 수 있는 고급의 자바 API이다. 이들 중 ***Alpha*** 구현 과정에서 이용된 패키지(*package*)는 JDI인데, 그 활용 사례 한 가지를 들면 다음과 같다. 사용자에게 특정 필드의 값이 변화되는 과정을 보여 주려 할 경우, ***Alpha***는 JDI를 이용하여 해당 필드에 대해 **ModificationWatchpointRequest**를 설정한다. 이후 자바가상기계는 해당 필드의 값이 변경되려 할 때마다 관련 정보(**ModificationWatchpointEvent**의 인스턴스)를 이벤트 큐(*event queue*)에 넣어 주며, ***Alpha***는 이벤트 큐에서 해당 정보를 읽어 들여 그 내용을 사용자에게 시각화시켜 준다. JDI는 필드변경 이외에도 필드접근, 클래스적재, 메쏘드호출/복귀, 정지점(*breakpoint*) 등에 대한 이벤트통지 요청을 지원하며, ***Alpha***가 이들을 활용하는 방식은 거의 유사하다. ***Alpha***는 필요할 경우 자바가상기계의 수행을 보류(*suspend*)시켜, 사용자가 자바가상기계 내부의 세부 상황을 확인해 보거나 변경시킬 수 있게 지원한다.

JDI는 4개의 서브패키지(*subpackage*)로 구성되어 있는데, 이들 패키지는 대부분 자바의 인터페이스(**Interface**)들로 구성된다. JDI 인터페이스들을 그들이 속한 패키지에 따라 구분하면 다음과 같다.

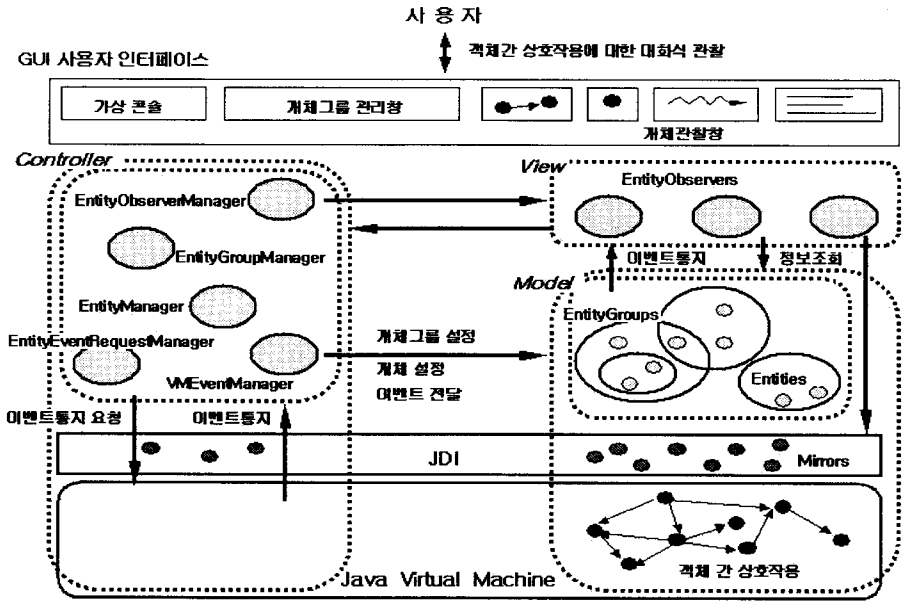
- **com.sun.jdi** : 다양한 미러(*mirror*)들에 대한 인터페이스들
- **com.sun.jdi.connect** : 디버거와 자바가상기계 간의 연결을 설정하는 인터페이스들
- **com.sun.jdi.event** : 자바가상기계가 생성하는 JDI 이벤트들에 대한 인터페이스들
- **com.sun.jdi.request** : 자바가상기계에 대한 이벤트통지 요청 관련 인터페이스들

자바가상기계는 내부적으로 인스턴스, 클래스, 쓰레드, 메쏘드, 필드, 지역변수 등을 관리한다. JDI는 그들과 일대일로 대응된 객체인 미러(*mirror*)를 설정하여 JDI 사용자에게 제공한다. ***Alpha***는 JDI의 미러를 이용하여 자바가상기계 내부에 존재하는 객체나 쓰레드의 상태를 확인하거나 변경하며, 클래스, 메쏘드 등에 대한 세부 정보도 추출해낸다.

3.2 MVC(*Model-View-Controller*)

Alpha는 <그림 1>과 같이 MVC 구조[14]에 기반하여 동작한다. MVC 구조는 모델(*mode*), 뷰(*view*), 컨트롤러(*controller*)라는 서로 독립된 요소들로 구성되는데, 각 요소의 역할을 요약하면 다음과 같다.

- 모델 : 시스템 작동이나 프로그램 수행 등과 관련된 내부 요소들로서, 사용자가 관찰할 수 있는 대상이다.
- 뷰 : 모델에서 발생하는 변화를 사용자에게 보여주는 역할을 하며, 하나의 모델에 대해 다수의 뷰를 설정할 수 있다.
- 컨트롤러 : 동적으로 각 모델과 뷰 간의 연결을 설정해 주고, 모델과 뷰



<그림 1> *Alpha*의 MVC 구조

를 동기화시키기 위해 그들 간의 통신을 지원한다.

컨트롤러의 역할로 인해 모델과 뷰를 분리·구현할 수 있게 되며, 결과적으로 새로운 뷰의 개발 부담이 줄어든다. 또한 컨트롤러에 의해 하나의 모델에 대해 다수의 동기화된 뷰를 설정할 수 있어 모델의 변화를 다양한 관점에서 동시에 관찰할 수 있게 된다. MVC 구조를 기반으로 ***Alpha***는 확장성(새로운 뷰를 쉽게 개발·추가)과 유연성(사용자의 수준이나 목적에 따라 뷰를 동적으로 조합)을 제공한다. 다음 절들은 ***Alpha***의 모델과 관련된 개체(*entity*) 개념, ***Alpha***에 있어 뷰의 역할을 수행하는 개체관찰자(*entity observer*), 모델과 뷰를 동기화시키기 위한 이벤트 전달(*event notification*) 기법에 대해 설명하고 있다.

3.3 개체(*Entity*)

Alpha의 개체는 기본개체(*basic entity*)와 개체그룹(*entity group*)으로 나뉘는데, 클래스 적재시 설정되는 클래스객체(*class object*)나 클래스의 개별 인스턴스(쓰레드 인스턴스 포함)들이 기본개체에 해당되고, 개체들의 모임이 개체그룹이다. 각 기본개체(클래스개체, 인스턴스개체, 쓰레드개체)는 JDI가 제공하는 미러(*mirror*)와 일대일로 대응되어 있어, 해당 미러를 통해 개체 관련 정보를 얻을 수 있다. ***Alpha***는 임의 개의 개체그룹을 다른 개체그룹의 멤버로 등록할 수 있게 지원하며, 개체그룹 B가 개체그룹 A의 멤버일 때 B의 멤버로 등록된 모든 기본개체를 A의 멤버로 간주한다. 사용자는 개체그룹의 계층구조를 이용하여 보다 효과적으로 개체들의 모임을 설정할 수 있다.

개념적으로 볼 때, ***Alpha***의 모델(관찰 대상)은 자바 프로그램 수행 중에 발생하는 '객체 간 상호작용'이다. 자바 언어에서 '객체 간 상호작용'은 특정 '쓰레드들'이 특정 '객체들'을 대상으로 작업(메쏘드 수행)하는 과정에서 발생한다. 다시 말해 '쓰레드들'이

작업주체가 되고 '객체들'이 작업대상이 되어 발생하는 현상이다. 여기서 고려해야 할 점은, 작업주체와 작업대상에 포함된 쓰레드나 객체의 수가 많을 경우 '객체 간 상호작용'의 범위가 커져 그 중 일부분만을 관찰하는데 비효율적일 수 있다는 것이다. ***Alpha***의 MVC 요소 중 컨트롤러는 사용자가 자신의 목적이나 수준에 따라 '객체 간 상호작용'의 범위를 한정하도록 지원함으로써 이 문제를 해결하고 있다. 사용자는 작업주체와 작업대상에 해당하는 두 개체를 지정하여 '객체 간 상호작용'의 범위를 한정할 수 있는데, 이들 두 개체를 '작업개체(*working entity*)'라 한다. 작업주체로서의 작업개체는 '쓰레드들'을 포함한 개체그룹이고, 작업대상으로서의 작업개체는 한 객체에 대응된 기본개체이거나 다수의 객체들을 포함한 개체그룹이다. ***Alpha***의 컨트롤러는 사용자가 작업개체를 효과적으로 설정할 수 있도록 세 가지 유형(*p*-group, *i*-group, *t*-group)의 개체그룹을 만들어 제공한다. 이들은 사용자가 만드는 사용자 개체그룹과 구별하여 시스템 개체그룹이라 불리며, 시스템 내의 모든 개체를 그룹 단위로 계층화하여 사용자에게 제공한다. *p*-group은 패키지 계층구조에 따라, *i*-group은 상속 계층구조에 따라 클래스개체와 그 인스턴스개체들을 그룹핑하는데 사용되며, *t*-group은 쓰레드개체들을 자바의 쓰레드그룹 계층구조에 따라 그룹핑해 준다.

3.4 개체관찰자(*Entity Observer*)

개체관찰자는 ***Alpha***의 MVC 구조에서 뷰의 역할을 수행하는 요소이다. 개체관찰자는 그 타입에 따라 서로 다른 시각화 방식을 구현하기 때문에 동일 작업개체라 할지라도 어떤 타입의 개체관찰자를 설정하느냐에 따라 시각화 방식이 달라진다. 따라서 작업개체를 선택하고, 그것을 시각화할 개체관찰자의 타입을 지정함으로써 사용자는 선택한 대상을 원하는 방식으로 관찰할 수 있게 된다. 또한 동일 작업개체에 대해 복수 개의 개체관찰자들을 설정함으로써, 하나의 대상을 다양한 방식으로 시각화시킬 수 있다.

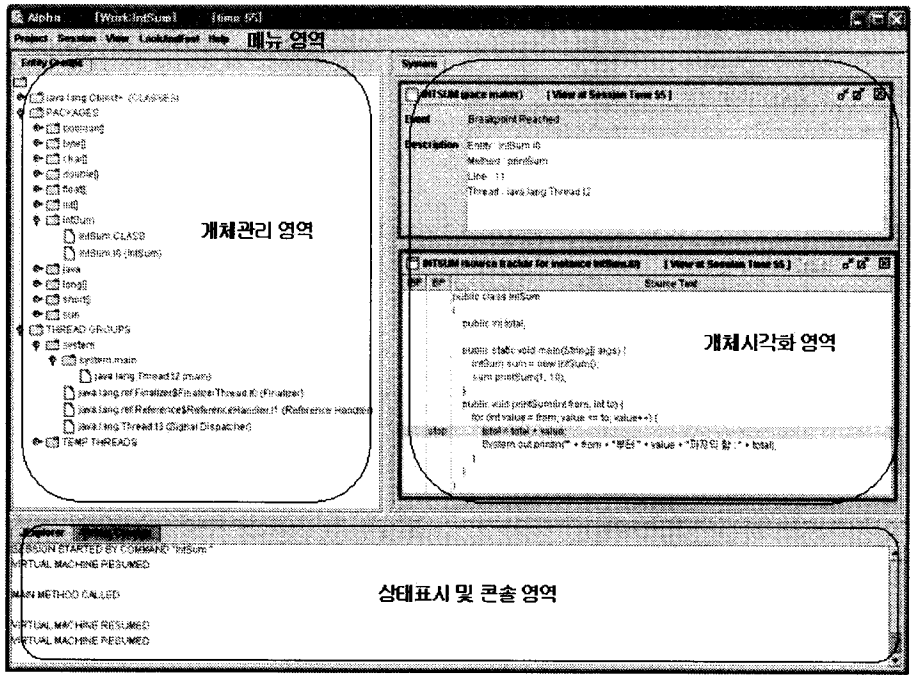
3.5 이벤트 전달

Alpha의 컨트롤러는 기본개체와 개체그룹을 관리하면서, 사용자가 작업개체와 개체관찰자를 연결할 수 있도록 지원한다. 또한, 작업개체와 관련된 이벤트를 개체관찰자에게 통지함으로써, 특정 작업개체와 연결된 모든 개체관찰자가 작업개체의 변화를 발생 즉시 동기적으로 시각화할 수 있도록 지원한다. 개체관찰자에게 통지되는 이벤트는 크게 다음 두 가지 타입으로 구분된다.

- 미러(*mirror*) 수준의 이벤트 : 자바가상기계 내부에서 JDI의 미러와 관련하여 발생하는 이벤트이다. ***Alpha***는 3.1절에 언급했던 바와 같이 JDI 이벤트가 발생하면 이벤트 큐에 해당 이벤트 정보(JDI에 정의된 클래스 **com.sun.jdi.event.Event**의 인스턴스)가 연결된다. 이로부터 이벤트와 관련된 미러를 알아낸 후, 이 미러와 대응된 작업개체를 찾아 그 개체관찰자에게 이벤트 정보를 통지한다.
- 개체 수준의 이벤트 : 개체 관리 작업과 연관된 이벤트이다. 사용자가 개체의 이름을 변경하거나 이벤트통지를 요청할 경우, 개체그룹에 멤버가 등록되거나 삭제될 경우 등 개체 수준의 이벤트가 발생할 때, 해당 개체관찰자에게 이벤트 정보(***Alpha***가 자체적으로 정의한 클래스 **Entity Event**의 한 인스턴스)를 통지한다.

4. Alpha의 구현 및 활용

4.1 사용자 인터페이스



<그림 2> ***Alpha***의 사용자 인터페이스 구성

Alpha의 사용자 인터페이스는 <그림 2>와 같고, 그 위치에 따라 다음 다섯 영역으로 구분될 수 있다.

- 개체관리 영역 : 개체그룹을 생성·관리·제거하거나 각 개체에 대한 이벤트통지를 요청하는 영역으로, 시스템 개체그룹과 사용자 개체그룹이 트리 형태로 계층화되어 제시된다. 사용자는 이 영역에 제시된 개체나 개체그룹을 작업개체로 선택하여 그들에 대한 개체관찰자를 설정할 수 있다.
- 개체시각화 영역 : '관찰창'과 '관찰판'이 설정되는 영역이다. 개체관찰자는 관찰창을 통해 작업개체의 상태나 활동을 시각화하는데, 하나의 관찰판에 다수의 관찰창이 설정될 수 있다. 사용자는 필요할 때 관찰판을 만들고 각 관찰판에 자신이 원하는 관찰창들을 설정함으로써 그룹 단위로 관찰창들을 관리할 수 있다.
- 타이틀 영역 : 윈도우의 최상단에 위치하며 현 프로젝트 이름과 메인 클래스 이름, 세션시각(*session time*) 등이 표시된다. 세션시각은 세션(메인 클래스가 수행되는 기간)이 시작될 때 0으로 초기화된 후 JDI 이벤트가 발생할 때마다 1씩 증가되는 시간이다. 사용자들은 관찰창에 표시된 관찰시각(개체관찰자들이 관찰대상을 시각화시킨 세션시각)과 타이틀 영역에 표시된 세션시각을 비교함으로써, 관찰시각의 현재성을 알 수 있다.
- 메뉴 영역 : 환경 설정(프로젝트 설정, 메인 클래스 지정 등), 세션 제어(세션 시작과 종료, 일시정지와 재개 등), 화면 제어 등을 위한 메뉴들의 영역이다.

■ 상태표시 및 콘솔 영역 : ***Alpha***의 작업 상태(세션 시작, 가상기계 *resume/suspend*, 메인 메쏘드 호출 등)를 표시하거나, 현재 수행되고 있는 자바 프로그램의 입출력을 위해 가상콘솔(*virtual console*)이 설정되는 영역이다. 가상콘솔은 자바 프로그램의 출력 내용 표시와 사용자의 키보드 입력 전달 등, 수행 중인 자바 프로그램과 사용자 사이에서 입출력 매체 역할을 수행한다.

4.2 개체관찰자 PaceMaker

PaceMaker는 기능은 단순하지만 그 역할은 매우 중요한 개체관찰자이다. PaceMaker는 작업개체와 관련된 이벤트가 통지(*notify*)될 때마다 이벤트에 대한 개괄적 정보를 사용자에게 보여준 후, 사용자가 '객체 간 상호작용'의 세부 내용을 조사하거나 변경할 수 있도록 세션을 일시정지(*suspend*) 상태로 유지시켜 준다. PaceMaker는 ***Alpha***가 서론에 기술한 '대화식 관찰' 도구로서의 역할을 온전히 수행할 수 있도록 ***Alpha***의 동작을 보완한다. '대화식 관찰'은 '객체 간 상호작용'의 진행과 이에 대한 사용자의 관찰/개입 작업이 서로 동기화되도록 요구하는데, 사용자의 동기적 관찰은 ***Alpha***의 컨트롤러가, 사용자의 동기적 개입은 PaceMaker가 가능케 한다.

'대화식 관찰'이 요구하는 동기화를 위해 ***Alpha***가 지원하는 기능은 '객체 간 상호작용'을 동기적으로 통지(*notify*)하는 일과 일시정지(*suspend*)시키는 일이다. 동기적 통지는 '객체 간 상호작용' 과정에서 이벤트가 발생할 경우 '객체 간 상호작용'의 진행을 일시정지시킨 상태에서 개체관찰자들에게 이벤트를 통지하여 처리되게 하는 것을 말하며, 동기적 정지는 '객체 간 상호작용'이 특정 조건을 만족하는 시점(특정 메쏘드가 호출되거나 특정 필드가 수정되는 등의 이벤트 발생 시점)에 그 진행을 일시정지시키는 것을 말한다. ***Alpha***는 이벤트가 발생하면 '객체 간 상호작용'을 일시정지시킨 채 개체관찰자에게 통지하여 시각화시키게 하고, 이 과정에서 개체관찰자가 동기적 정지를 요청하면 사용자가 재개(*resume*)시킬 때까지 '객체 간 상호작용'의 일시정지 상태를 유지시켜 준다.

결과적으로 ***Alpha***에서 동기적 정지 시점은 개체관찰자들이 어떤 동기적 정지 정책을 가지고 있느냐에 따라 정해진다. 이 경우 가장 큰 문제는 개체관찰자가 동기적 정지 요청을 하지 않으면 이벤트 처리 후 '객체 간 상호작용'이 재개되어 사용자가 동기적으로 개입할 수 있는 기회를 얻지 못한다는 것인데, 이는 사용자가 '객체 간 상호작용'에 대한 개입 시점을 스스로 선택할 수 없다는 것을 의미한다. 이에 대한 해결책이 바로 개체관찰자 PaceMaker의 역할이다. PaceMaker는 통지된 모든 이벤트에 대해 동기적 정지를 요청함으로써 '객체 간 상호작용'의 일시정지 상태를 유지시켜 준다. PaceMaker의 동기적 정지 정책을 통해 동기적 정지 시점에 대한 선택권은 사용자에게 위임된다. 특정 개체에 대해 이벤트통지 요청을 하고 개체관찰자 PaceMaker를 연결해 두면, 해당 이벤트 발생시 PaceMaker가 사용자를 대신해 동기적 정지 요청을 하기 때문이다.

4.3 개체관찰자 SourceTracker

자바 프로그램 수행 중에 발생하는 '객체 간 상호작용'은 쓰레드가 자바 프로그램 코드에 명시된 작업을 수행하는 과정에서 일어난다. 따라서 수행 중인 자바 코드의 내용과 그로 인해 발생하는 '객체 간 상호작용'을 대비시켜 관찰할 경우, 프로그램에 표현된 '객체 간 상호작용'과 프로그램 수행 중 발생하는 '객체 간 상호작용'의 관계를 보다 구체적으로 이해할 수 있게 된다. 이

러한 필요성을 충족시키기 위해 구현된 개체관찰자가 SourceTracker이다. Source Tracker는 특정 객체와 관련된 소스(source) 내용을 <그림 2>와 같이 테이블 형태로 제시하고, 현재 쓰레드에 의해 수행되고 있는 코드의 위치를 표시해 줌으로써 사용자가 자바 코드의 수행 흐름을 파악할 수 있게 지원한다. 또한 현재 설정되어 있는 정지점(*breakpoint*)들을 표시해 사용자가 해당 이벤트가 발생될 것을 미리 예측할 수 있도록 지원한다.

4.4 소스 구성

*Alpha*는 <표 2>에 보인 바와 같이 네 가지 패키지로 구성되어 있다. 패키지 **model**, **view**, **control**은 *Alpha*의 MVC 구조와 관련하여 각각 모델 관련 클래스, 뷰 관련 클래스, 컨트롤러 관련 클래스들을 정의한다. 현재 패키지 **view**에는 앞서 기술한 개체관찰자 **PaceMaker**와 **SourceTracker**가 정의되어 있으며, 이후 개발될 개체관찰자들 역시 패키지 **view**에 포함될 것이다. 패키지 **util**에는 개체관찰자 구현 과정에서 사용되는 자바 인터페이스들이 정의되어 있다. 자바 인터페이스 **EntityObserver**는 개체관찰자 개발시 구현해야 할 메쏘드들을 정의하고 있고, 그 외의 자바 인터페이스들은 *Alpha*의 모델과 컨트롤러에 대한 인터페이스를 정의한다. 자바 인터페이스 **EntityEvent**, **Entity**, **EntityGroup**은 개체나 개체 수준의 이벤트와 관련된 정보를 참조할 때 사용되고, 자바 인터페이스 **SystemUtilities**는 *Alpha*의 기타 자원(혹은 정보)을 이용할 수 있도록 제공되는 인터페이스이다.

<표 2> *Alpha*의 패키지 구성

packages	소 속 클 래 스
model	**InternalEntity** **InternalClassEntity** **InternalEntityGroup**
view	**PaceMaker, SourceTracker**
control	**MainGUI, VMEventManager** **EntityManager, EntityGroupManager** **VirtualMachineConsole, Session** **EntityObserverManager** **EntityObserverAdapter** **InternalEntityEvent** **SystemUtilitiesImpl** **EntityEventRequestManager**
util	**EntityObserver, Entity** **EntityGroup, EntityEvent** **SystemUtilities**

4.5 활용 사례

이 절에서는 <그림 3>에 보인 자바 프로그램 **IntSum.java**가 수행될 때 발생하는 '객체 간 상호작용'을 어떻게 관찰할 수 있는지 설명한다. 우선 **IntSum.class**가 정의된 패키지(폴더 혹은 디렉토리)를 프로젝트 폴더로 설정하고, **IntSum.class**를 메인 클래스(*main class*)로 선택한 후, 클래스 **IntSum**에 대한 세션을 시작시킨다. 그리고 세션이 시작되어 클래스 **IntSum**의 메쏘드 **main**이 호출된 후 **IntSum** 클래스개체가 속한 시스템 개체그룹 IntSum에 대해 개체

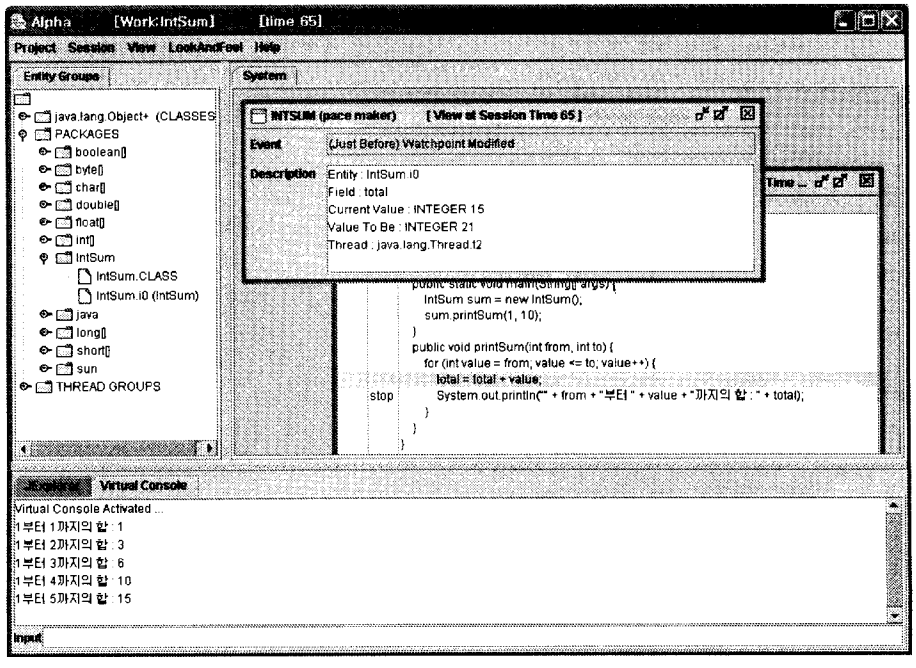
```
public class IntSum
{
    public int total;

    public static void main(String[] args) {
        IntSum sum = new IntSum();
        sum.printSum(1, 10);
    }
    public void printSum(int from, int to) {
        for (int value = from; value <= to; value++) {
            total = total + value;
            System.out.println("" + from + "부터 " + value + "까지의 합 : " + total);
        }
    }
}
```

<그림 3> **IntSum.java**

관찰자 PaceMaker를, 인스턴스 intSum.i0에 대해 개체관찰자 SourceTracker를, 인스턴스 변수 **total**에 대해 **ModificationWatchpointRequest**를, 클래스 **IntSum**의 라인 12에 **BreakpointRequest**를 설정한다. <그림 4>는 15(= 1 + 2 + 3 + 4 + 5)가 저장된 인스턴스 변수 **total**에 21(= 15 + 6)이 치환되기 직전 ***Alpha***가 동기적으로 일시정지되어 있는 모습이다. PaceMaker는 인스턴스 변수 **total**의 현재 값과 치환될 값, 어떤 쓰레드가 그 일을 수행하고 있는지 등을 보여 주고, SourceTracker는 현재 해당 쓰레드가 수행 중인 라인 위치와 정지점(*breakpoint*) 설정 상태를 보여 준다. <그림 5>는 인스턴스 intSum.i0가 두 개체그룹 IntSum과 selected에 등록된 상태에서 개체관찰자들에 의해 시각화되고 있는 모습을 보여 준다. 이는 ***Alpha***를 이용할 경우 기존의 객체들을 임의로 조합하여 그 참조 관계나 상호작용을 효과적으로 관찰할 수 있음을 보여주는 것으로, 앞서 비교했던 BlueJ나 JSwat은 이러한 기능을 지원하지 않는다.

5. 결론 및 향후 계획

프로그래밍의 목적은 프로그램 수행을 통한 문제해결이지만, 프로그램 수행 과정을 바라보는 시각은 프로그래밍 언어(혹은 패러다임)에 따라 다르다. 객체지향 프로그래밍에 있어 프로그램 수행에 대한 올바른 시각은 '객체 간 상호작용'이며, 따라서 객체지향 프로그래밍 언어인 자바를 올바로 이해하고 활용하려면, 자바 프로그램의 수행 과정을 '객체 간 상호작용'의 관점에서 바라볼 수 있어야 한다. 본 논문이 제시하고 있는 시각화 도구 ***Alpha***는, 개발과정 전반에 걸쳐 '자바 프로그램 수행 과정에서 발생하는 객체 간 상호작용을 효과적으로 관찰할 수 있게 지원한다'는 기본 목적 하에 개발된 도구이다. 사용자 작업의 효율성과 다양성을 효과적으로 지원하기 위해 MVC (*Model-View-Controller*) 구조로 설계된 ***Alpha***



<그림 4> **IntSum** 세션 기간 중 동기적으로 일시정지되어 있는 ***Alpha***의 모습

의 특징은 다음과 같다. 첫째, 개체와 개체 그룹 개념을 도입하여 객체나 쓰레드 혹은 그들의 모임을 작업개체(모델)로 설정하기 쉽다. 둘째, 작업개체(모델)와 개체관찰자(뷰)의 일대다 관계를 동적으로 설정할 수 있다. 셋째, 새로운 개체관찰자(뷰)를 쉽게 개발·추가함으로써 활용분야를 쉽게 넓힐 수 있다.

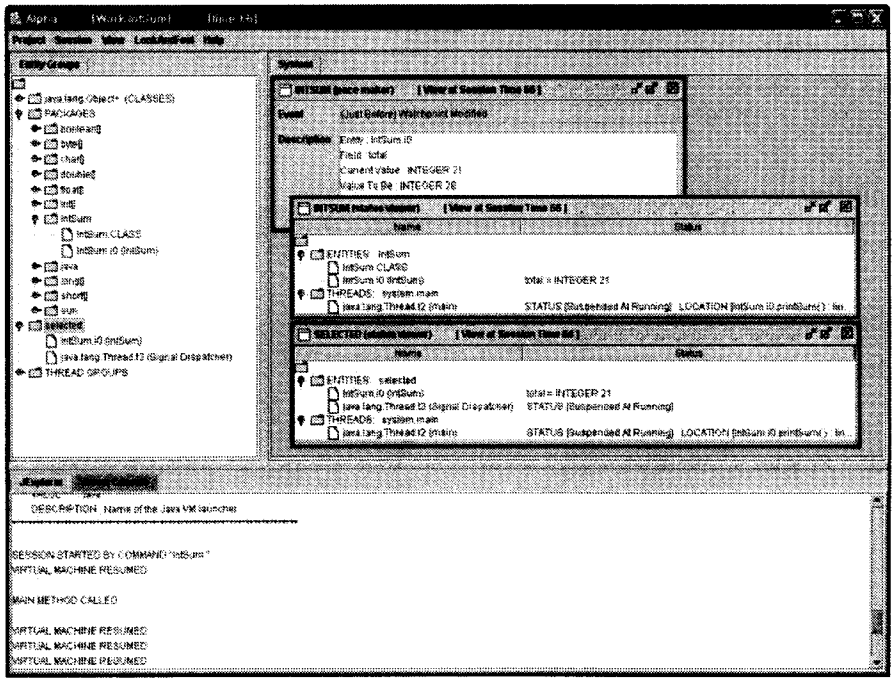
앞서 기술한 개체관찰자들을 구현하면서 개체관찰자 개발 및 추가 작업의 효율성을 검증하였고, 이들이 ***Alpha***의 MVC 구조 상에서 자바가상기계 내부의 '객체 간 상호작용'을 효과적으로 시각화해 준다는 점도 확인하였다. 현재 개체관찰자의 다양성을 확대하면서 시각화 효과를 극대화하기 위해, 객체 간의 상호참조 관계를 보여 주는 개체관찰자와, 쓰레드의 수행흐름을 시각화시켜 주는 개체관찰자, 인위적으로 인스턴스들 간의 참조관계를 설정하고 그들 간에 발생하는 상호작용을 관찰할 수 있게 지원하는 개체관찰자 등을 자바의 GUI 컴포넌트를 활용하여 개발하고 있다. 향후 의사코드(혹은 다른 언어로 작성된 프로그램)로 표현된 알고리즘(혹은 자료구조 연산)의 수행 과정을 시각화시켜 줌으로써, ***Alpha***의 활용도를 알고리즘이나 자료구조 분야로 넓혀갈 계획이다.

참 고 문 헌

[1] Ashley George Hamilton-Taylor, and Eileen Kraemer, SKA: supporting algorithm and data structure discussion, Proceedings of the 33rd SIGCSE technical symposium on Computer science education, 2002, Pages: 58-62.

[2] Carlisle E. George, Using Visualization to Aid Program Construction Tasks, Proceedings of the 33rd SIGCSE technical symposium on Computer science education, 2002, Pages: 191-195.

[3] Guido Rößling and Thomas L. Naps,



<그림 5> 두 개체그룹에 등록된 동일 인스턴스의 관찰

A Testbed for Pedagogical Requirements in Algorithm Visualizations, Proceedings of the 7th annual conference on Innovation and technology in computer science education, 2002, Pages: 96-100.

[4] Henrik Bærbak Christensen and Michael E. Casperson, Frameworks in CS1 - a Different Way of Introducing Event-driven Programming, Proceedings of the 7th annual conference on Innovation and technology in computer science education, 2002, Pages: 75-79.

[5] Herbert L. Dershem, Ryan L. McFall, and Ngozi Uti, Animation of Java Linked Lists, Proceedings of the 33rd SIGCSE technical symposium on Computer science education, 2002, Pages: 53-57.

[6] Nathan Fiedler, `http://www.bluemarsh.com/java/jswat`.

[7] Michael Kölling, The BlueJ Tutorial, `http://www.bluej.org`.

[8] Richard Rasala, Jeff Raab, and Viera K. Proulx, Java Power Tools: Model Software for Teaching Object- Oriented Design, Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, 2001, Pages: 297-301.

[9] Ryan S. Baker, Michael Boilen, Michael T. Goodrich, Roberto Tamassia, and B. Aaron Stibel, Testers and Visualizers for Teaching Data Structures, Proceedings of the thirtieth SIGCSE technical symposium on Computer Science Education, 1999, Pages: 261-265.

[10] T. Dean Hendrix, James H. Cross II, and Saeed Maghsoodloo, Do Visualzations Improve Program Comprehensibility? Experiments With Control Structure Diagrams For Java, Proceedings of the 31st SIGCSE technical symposium on Computer science education, 2000, Pages: 382-386.

[11] Viera K. Proulx, Jeff Raab, and Richard Rasala, Objects from the Beginning - With GUIs, Proceedings of the 7th annual conference on Innovation and technology in computer science education, 2002, Pages: 65-69.

[12] Wim De Pauw, Richard Helm, Doug Kimelman, and John Vlissides, Visualizing the behavior of object-oriented systems, Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications, 1993, Pages: 326-337.

[13] Wim De Pauw, David Lorenz, John Vlissides, and Mark Wegman, Execution Patterns in Object-Oriented Visualization, Proceedings of the Fourth USENIX Conference on Object-Oriented Technologies and Systems, 1998, Pages: 219-234.

[14] Yen-Ping Shan, An Event-Driven Model-View-Controller Framework for Smalltalk, Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications, October 1989, Pages: 347-352.

김 철 민

1988 서울대학교 컴퓨터공학과(공학사)
1990 서울대학교 컴퓨터공학과(공학석사)
1996 서울대학교 컴퓨터공학과(공학박사)
1997~현재 제주대학교 컴퓨터교육과 조교수
관심분야 : 컴퓨터교육, 운영체제, 프로그래밍
E-Mail: cmkim@cheju.ac.kr