동적 인스트루먼테이션을 이용한 자바 가상 머신 재구성 모델

손재웅*, 김영필*, 유혁*
*고려대학교 컴퓨터학과
e-mail : jwson@os.korea.ac.kr

Reconfigurable Java Virtual Machine Model Using Dynamic Instrumentation

Jae-Woong Son*, Young-Pill Kim*, Hyuck Yoo*
*Dept. of Computer Science & Engineering, Korea University

요 약

최근 다양한 내장형 시스템에서 이식성, 신뢰성, 재사용성에서 장점을 가지는 자바가 많이 사용되고 있다. 또한, 내장형 시스템 환경에서 서비스들에 대한 사용자의 요구와 하드웨어 플랫폼이 다양해지고 있다. 내장형 시스템의 플랫폼과 사용자 요구가 다양해짐에 따라 이를 효율적으로 반영하기 위하여 자바 가상 머신의 동적 재구성이 필요하다. 그러나 기존의 자바 가상 머신 재구성에 관한 연구는 컴포넌트 기반 재구성 방법이 대부분이고 이 방법은 컴포넌트의 일부분이 교체될 때 오버헤드가 발생한다. 따라서 본 연구에서는 기존 연구의 단점을 해결할 수 있는 동적 재구성이 가능한 자바 가상 머신 모델을 제안한다.

1. 서론

핸드폰 단말기, PDA, 디지털 TV, 웹 스크린 폰, 그리고 스마트카드 등과 같은 내장형 시스템은 다양한 하드웨어 플랫폼이 존재한다. 또한, 내장형 시스템을 기반으로 한 서비스들이 많아짐에 따라서 사용자의 서비스 요구 역시 다양해지고 있다. 이러한 서비스 요구를 만족시키기 위해서는 그에 상응하는 새로운 응용 프로그램이 개발되어야 한다. 응용 프로그램 개발을 위한 플랫폼으로써 하드웨어와 무관하게 일관된 응용 프로그램을 제공할 수 있고 재사용이 용이하고 신뢰성이 보장되는 자바 언어의 중요성이 커지고 있다. 따라서 자바 가상 머신(Java Virtual Machine)[1]에 대한 관심 역시 증대되었는데, 이는 내장형 시스템에서 다양한 요구를 효율적으로 만족 시키고 시스템 환경에 최적화시키는 것을 목표로 한다.

내장형 시스템은 메모리 크기, 입출력 주변 장치, CPU 성능 등에 대한 제약을 갖고 있다. 그래서 내장형 시스템에서 사용되는 소프트웨어 시스템은 이러한 제약 사항을 만족시키기 위해서 커스터마이즈(customize) 해야 한다. 썬마이크로시스템(Sun Microsystems)은 내장형 시스템에서 시스템 개발자가 각 하드웨어 플랫폼에 커스터마이즈된 자바 가상 머신을 제공할 수 있도록 자바 환경 명세서를[2,3] 제공한다. 그러나 자바 가상 머신의 커스터마이즈는 컴파일(compile) 과정에서 진행되기 때문에 자바 가상 머신의 소스코드(source code)를 수정하고 컴파일한 후 요구들을 만족시킬 수 있다. 이 방법은 내장형 시스템에서의 다양한 요구를 효과적으로 만족시키기 어렵다.

최근, 여러 종류의 하드웨어 플랫폼과 사용자 요구의 다양성을 적시에 만족시키기 위해 자바 가상 머신의 동적 재구성에 대한 연구가 진행되고 있다. 이 연구는 사용자의 요구를 적시에 만족시키고, 체계적인 방법으로 자바 가상 머신의 커스터마이즈에 초점을 두고 있다. 자바 가상 머신의 재구성에 관한 연구들은 컴포넌트 기반으로 진행되고 있다. 컴포넌트 기반의 재구성 방법은 컴포넌트를 교체함으로써 자바 가상 머신의 기능을 확장 또는 축소할 수 있다. 그러나 이러한 방법은 컴포넌트의 크기가 큰 경우 재구성하는데 많은 시간이 소요되고 컴포넌트 내부의 일부분을

수정 또는 확장, 삭제할 경우 이를 포함하는 컴포넌트를 교체해야 하는 단점을 갖고 있다.

본 논문에서는 다양한 내장형 시스템을 지원하고 사용자 요구를 반영할 수 있으며 필요한 자바 가상 머신의 기능을 동적 인스트루먼테이션(Instrumentation)을 이용하여 제공할 수 있는 동적 재구성 모델을 제시한다.

본 논문의 구성은 다음과 같다. 먼저 2 장에서는 자바 가상 머신의 재구성에 관한 기존 연구 및 방법에 대하여 알아보고, 3 장에서는 동적 인스트루먼테이션 기법을 이용한 자바 가상 머신의 재구성 모델을 제안한다. 마지막으로 결론을 기술한다.

2. 관련 연구

현재 자바 가상 머신을 동적 재구성하는 방법으로 컴포넌트 기반 재구성 방법[4,5]이 있다. 이 방법은 자바 가상 머신의 재구성이 동적으로 이루어져 커스터마이즈가 가능하지만 컴포넌트의 크기에 따른 오버헤드(overhead)를 갖고 있다. 본 논문에서는 컴포넌트 기반 재구성 방법을 설명하고 이 방법이 갖고 있는 오버헤드를 해결할 수 있는 동적 인스트루먼테이션 기법[6]에 대하여 소개하겠다.

2.1 컴포넌트 기반 재구성 방법

내장형 시스템에서 컴포넌트 기반 재구성 방법을 간단히 소개하자면, 자바 가상 머신의 구성 요소(Class Loader, Execution Engine, Scheduler, Memory manager, Exception Handler)를 컴포넌트로 만들고 이러한 컴포넌트들을 재구성하여 자바 가상 머신을 커스터마이즈하는 방법이다. 그리고 컴포넌트를 재구성할 경우 교체할 컴포넌트는 다양한 요구에 따라 만들어진 컴포넌트가 저장되어 있는 서버에서 다운로드(download) 받아 재구성한다.(그림 1)

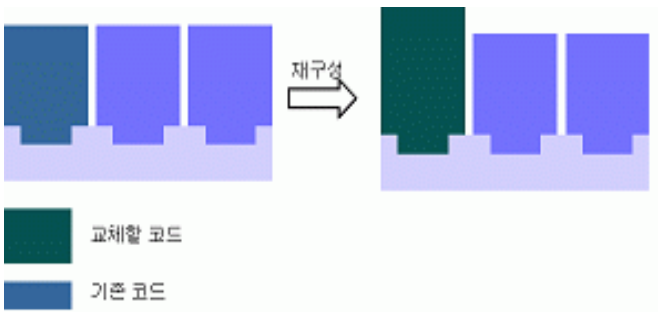

그림 1 컴포넌트 기반 동적 재구성 방법

컴포넌트 기반 재구성 방법은 공통적인 인터페이스를 갖는 컴포넌트를 교체함으로써 동적으로 쉽게 재구성할 수 있는 장점을 갖고 있다. 그러나 동적 재구성할 때 자바 가상 머신의 구성 요소로 컴포넌트를 구성한다면, 컴포넌트의 일부분을 수정, 삭제, 확장할 경우 이 부분을 포함한 컴포넌트를 교체함으로써 불필요한 작업이 수행된다. 또한 이러한 컴포넌트를 다수 교체할 필요가 있을 때, 교체할 컴포넌트를 서버에서 다운로드를 받는 시간이 많이 소요되는 단점을 갖고 있다. 이러한 단점을 보안하기 위해 자바 가상 머신의 구성 요소보다 작게 컴포넌트를 구성할 경우, 자바 가상 머신의 컴포넌트 사이의 의존성이 존재하며 이에 대한 고려가 필요하다.[9] 또한, 작은 크기의 컴포넌트로 구성하면 수 많은 컴포넌트의 인터페이스들로 인하여 자바 가상 머신의 크기가 커지는 단점을 갖고 있다.

2.2 동적 인스트루먼테이션 기법

인스트루먼테이션 기법은 임의의 프로그램 안에 새로운 기능을 기계어로 삽입하여 추가, 수정하거나 프로그램의 특정 부분을 삭제할 수 있다. 기존 인스트루먼테이션에 관한 연구들은 대상 프로그램을 멈추고 인스트루먼테이션을 수행한 후 다시 실행시키는 방법이었지만 최근 연구들은 프로그램의 실행 중에 인스트루먼테이션 기법을 적용하여 프로그램의 추가, 수정, 삭제 과정이 진행되며, 이러한 과정들은 변화된 프로그램을 재컴파일(re-compile), 재링크(re-link), 재실행(re-execution) 단계를 생략한다. 이 기법의 장점은 제어 흐름 그래프를 바탕으로 기존 프로그램의 재컴파일, 재링크, 재실행 과정 없이 기능의 삭제 및 삽입이 가능하다.

이러한 장점을 갖는 인스트루먼테이션 기법은 프로그램 안에 새로운 코드를 삽입시킬 때 삽입될 코드는 프로그램의 힙(heap) 영역에 적재하고 프로그램의 텍스트 영역에 코드 삽입될 주소공간(address space)으로 분기하는 방법을 사용한다. 분기하는 방법은 삽입한 코드가 있는 주소로 분기할 수 있는 분기 명령어를 프로그램의 텍스트 영역에 덮어쓰고 분기 명령어를 덮어쓸 부분의 코드는 삽입한 코드의 앞에 저장한다. 분기 명령어를 덮어쓰는 프로그램의 텍스트 영역을 인스트루먼테이션 포인트(Instrumentation point)라고 호칭한다.

분기 명령어를 이용한 인스트루먼테이션 기법은 인스트루먼테이션 기법을 사용할 시스템의 마이크로프로세서(microprocessor) 설계 구조마다 차이가 있다. 마이크로프로세서의 설계 구조가 RISC 인 경우 명령어 길이가 동일하기 때문에 인스트루먼테이션을 할 때 인스트루먼테이션 포인터 부분의 다른 코드까지 덮어쓰지 않는다. 그러나 마이크로프로세서 설계 구조가 CISC 인 경우 명령어 길이 다르기 때문에 인스트루먼테이션을 할 경우 다른 코드까지 덮어쓸 수 있다. 따라서 CISC 인 경우 로컬 바운싱(local bouncing) 기법[6]을 사용한다.

본 연구에서는 동적 인스트루먼테이션 기법에 관한 연구 중 분산 컴퓨팅에서 타겟(target) 시스템의 성능 및 병목 현상의 원인을 분석할 목적으로 개발된 DyninstAPI[7]를 이용한다. 이 DyninstAPI 는 마이크로프로세서 설계 구조에 따른 고려사항들을 포함한 라이브러리이다.

3. 제안 모델

컴포넌트 기반 자바 재구성 방법은 동적 재구성이 가능하지만 컴포넌트의 일부분을 수정, 삭제, 추가할 경우 오버헤드를 발생할 수 있다. 따라서 자바 재구성의 방법으로 동적 인스트루먼테이션 기법을 적용하였고 인스트루먼테이션 기법을 제공하는 DyninstAPI 를 이용하여 모델을 설계하였다.

3.1 DyninstAPI

DyninstAPI 의 전체적인 구조는 그림 2 와 같다. 뮤테이터(mutator)와 응용프로그램이라고 호칭하는 두 개의 프로세스(process)가 있다. 그림의 왼쪽은 DyninstAPI 를 포함하는 뮤테이터 프로세스를 위한 코드(code)를 보여준다. 뮤테이터에는 실행 중인 응용 프래그램의 프로세서를 조작하기 위한 유틸리티(utility)와 런타임 컴파일러(runtime compiler)가 포함된다. 그림의 오른쪽 중 상단 부분은 프로그램의 원래 코드를 갖는 응용 프로그램 프로세스를 보여준다. 응용 프로그램의 하단 두 부분은 프로그램 안에 삽입할 snippets 과 DyninstAPI 를 지원하는 런타임 라이브러리(runtime library)이다.

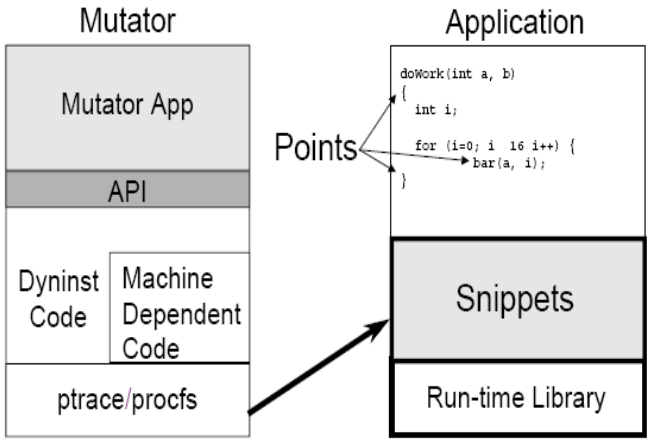

그림 2 DyninstAPI 구조

응용 프로그램에서 뮤테이터 프로세스에 의한 기본적인 오퍼레이션(operation)은 디버거(debugger)에 의해 사용되는 운영 체제 서비스들(ptrace, /proc filesystem 등)을 사용한다. 이러한 서비스들은 프로세스 실행을 조절하고 응용 프로그램의 주소 공간을 읽고 쓸 수 있는 방법을 제공한다. 그리고 유틸리티 함수들과 두 개의 배열을 포함하는 동적 링크 라이브러리는 인스트루먼테이션하는 응용 프로그램 안에 적재된다. 두 개의 배열은 동적으로 메모리의 작은 영역에 할당하기 위해 사용된다. 한 개의 배열은 인스트루먼테이션 변수들을 위해 사용되고, 다른 배열은 인스트루먼테이션 코드를 갖고 있다. 응용 프로그램에 안에 동적으로 삽입할 코드들이 저장될 이 배열들은 힙에 생성되고 배열들은 뮤테이터 프로세스에 의해 관리된다.

삽입될 코드를 생성하기 위해 뮤테이터 프로세스의 메모리 안에서 snippet 을 기계어로 변환한다. 그런 다음, 응용 프로그램의 주소 공간 안에 변환된 기계어를 복사한다. 복사한 후 삽입될 코드로 분기 할 수 있도록 응용 프로그램의 코드를 수정한다. 이를 위해 Dyninst 는 trampoline 이라고 불리는 코드의 작은 부분을 사용한다. 그림 3 은 trampoline 의 구조와 인스트루먼테이션 포인트와의 관계를 보여둔다. Trampoline 은 삽입할 코드를 위해 인스트루먼테이션 코드를 삽입할 부분에 대한 정보를 얻을 수 있는 방법을 제공한다. 이를 위해 인스트루먼테이션 포인트에 있는 명령어들을 base trampoline 의 시작 지점으로 분기하는 몇 개의 명령어로 교체한다. 그 후 base trampoline 코드는 mini-trampoline 으로 분기한다. Mini-trampoline 은 레지스터와 조건 코드(condition code)와 같은 머신 상태를 저장하고 snippet 코드를 포함한다. Snippet 의 마지막 부분에는 머신 상태를 되돌리고, base trampoline 으로 분기할 수 있는 코드가 있다. 이 코드를 실행한 후 Base Trampline 은 본래 인스트루먼테이션 포인트에 있던 코드를 실행한다.

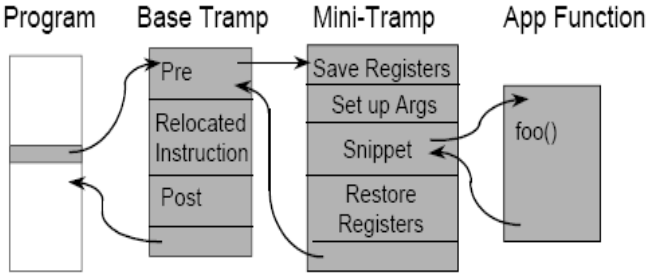

그림 3 수행 중인 프로그램으로 코드 삽입

3.2 동적 재구성 자바 가상 머신 모델

앞에서 설명한 DyninstAPI 구조와 실행 과정은 다양한 하드웨어 플랫폼을 고려했기 때문에 라이브러리의 사이즈는 크다. 따라서 내장형 시스템에서 자바 가상 머신의 재구성 방법으로 이용할 경우 뮤테이터와 Run-time Library, Snippets 에 의한 추가적인 메모리를 요구하는 단점을 갖고 있다.

본 연구에서는 DyninstAPI 를 자바 가상 머신의 재구성 방법으로 이용하기 위해 기존 DyninstAPI 구조를 서버(Server)-클라이언트(Client) 모델로 변화시켰다.

서버에는 동적 인스트루먼테이션을 수행하는 인스트루먼테이션 도구(mutator)가 있다. 이 도구는 DyninstAPI 와 자바 가상 머신 안에 삽입될 Snippet 으로 구성된다. 클라이언트는 내장형 시스템을 위해 개발된 자바 가상 머신(wabaVM[10])과 Run-time Library 로 구성된다.

그림 4 은 재구성의 모델을 보여준다. 인스트루먼테이션 도구는 자바 가상 머신의 제어 흐름 그래프를 생성하는데 필요한 정보를 ptrace 와 /proc 파일 시스템을 이용하여 가져온다. 제어 흐름 그래프를 생성한 후, 관리자는 클라이언트의 요구에 따라 자바 가상 머신의 기능들 중 어느 기능을 추가, 삭제, 수정할 지를 결정한다. 관리자의 결정에 따라 인스트루먼테이션 도구는 RPC(Remote Procedure Call)로 추가, 삭제, 또는 수정에 관한 명령어와 snippet 을 클라이언트의 Run-time Instrumentation Library 로 전송한다. Run-time

Instrumentation Library 는 전송 받은 데이터를 가지고 자바 가상 머신을 재구성한다.

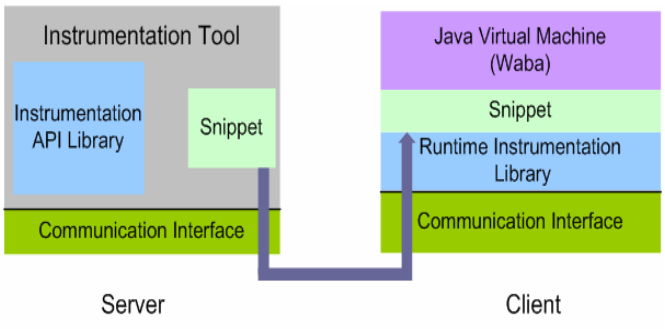

그림 4 동적 재구성 자바 가상 머신 모델

4. 결론 및 향후 계획

본 논문에서는 기존에 응용 프로그램의 병목 현상의 원인이나 성능을 측정하기 위해 사용되었던 동적 인스트루먼테이션 기법을 이용한 자바 가상 머신 재구성 모델을 제안하였다. 이러한 방법은 컴포넌트 기반 자바 가상 머신의 재구성 방법에서 컴포넌트의 일부분을 추가, 삭제, 수정할 때 컴포넌트 전체를 교체함으로써 발생하는 오버헤드를 동적 인스트루먼테이션 기법을 적용함으로써 재구성에 따른 오버헤드를 줄일 수 있다.

향후에는 악의적인 사용자가 응용프로그램에 접근하여 문제를 유발할 수 있으므로 보안에 관한 연구가 필요할 것이다. 또한, 자바 가상 머신을 재구성할 때 대상이 되는 기능이 의존성을 가지는 경우에 대한 연구가 필요하다. 그리고 향후 개발 예정인 동적 재구성 가능한 자바 가상 머신에 본 논문에서 제시한 동적 인스트루먼테이션을 이용한 자바 가상 머신 재구성 모델을 응용하여 구현할 것이다.

참고문헌

[1] T. Lindholm and F. Yellin, “The Java Virtual Machine Specification”, Addison-Wesley, 1999.

[2] Java Community Process, “Connected Limited Device Configuration”.

[3] Java Community Process, “Connected Device Configuration”.

[4] Hiroo Ishikawa, Tatsuo Nakajima, “EarlGray: A Component-Based Java Virtual Machine for Embedded Systems”, Proceedings of the Eighth IEEE International Sysmposium on Object-Oriented Real-time Distributed Computing, 2005.

[5] 이석진, 이승룡, “협력 시스템 지원 자바 가상 머신 재구성 도구 설계 및 구현”, 한국정보처리학회 추계학술대회 논문집 제 11 권 제 2 호, 2004.

[6] D. Pearce, P. Kelly, U. Harder, and T. Field, “GILK: A dynamic instrumentation tool for the Linux Kernel.”, In Proceedings of the 12th International Conference on Modeling Tools and Techniques for Computer and Communication System Performance Evaluation (TOOLS '02), pages 220--226, London, United Kingdom, 2002.

[7] Bryan Buck, Jeffrey K. Hollingsworth, “An API for Runtime Code Patching”, http://www.dyninst.org/papers/apiPreprint.pdf.

[8] B. Venners, “Inside The Java 2 Virtual Machine”, MacGraw Hill, 2000.

[9] 권정호, 김재훈, “컴포넌트기반 미들웨어에서 효율적인 컴포넌트 재구성”, 한국정보과학회 한국컴퓨터종합학술대회 Korea Computer Congress (KCC), 2005.

[10] Waba – The Embedded VM from WABASOFT. http://www.waba.com