(19)대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl.
G06F 9/45 (2006.01)

(45) 공고일자 2007년04월30일
(11) 등록번호 10-0712767
(24) 등록일자 2007년04월23일

(21) 출원번호 10-1999-0009754
(22) 출원일자 1999년03월23일
심사청구일자 2003년08월26일

(65) 공개번호 10-1999-0078122
(43) 공개일자 1999년10월25일

(30) 우선권주장 09/046,401 1998년03월23일 미국(US)

(73) 특허권자 썬 마이크로시스템즈, 인코포레이티드
미국 캘리포니아 95054 산타클라라 네트워크 써클 4150

(72) 발명자 라스,배크
미국,캘리포니아94303,팔로알토,코니아웨이3782

슬젼,미트로빅
미국,캘리포니아94065,레드우드쇼어즈,메디터레이니언레인826

(74) 대리인 강명구
강석용

(56) 선행기술조사문헌
EP0945791
US5586328 B
US5692047 B
US5579520 B
US5659751 B
AUSLANDER J ET AL : "FAST, EFFECTIVEE DYNAMIC COMP

* 심사관에 의하여 인용된 문헌

심사관 : 이정호

전체 청구항 수 : 총 36 항

(54) 컴파일된 코드에서 동적 클래스 초기화 검사의 비용을 줄이기 위한 컴퓨터 시스템의 명령어 컴파일 방법 및 그 컴퓨터 판독가능 매체

(57) 요약

본 발명은 컴파일을 마친 코드(compiled code)에서 동적 클래스 적재 검사(dynamic class loading check)와 클래스 초기화 검사(class initialization check) 의 비용을 줄이는 기술에 관한 것이다. 요구되어지는 런타임 실행 정보(required runtime execution information)가 컴파일(compile)을 하는 동안에 이용 가능하지 않더라도, 가상 기계 명령어(virtual machine instruction)가 하나 이상의 원시 기계 명령어(native machine instruction)로 컴파일(compile)된다. 원시 기계 명령어는 요구되어지는 런타임 실행 정보(required runtime execution information)가 있어야 하는 플레이스 홀더 데이

터(placeholder data)를 포함한다. 상기 원시 기계 명령어는, 런타임 실행시, 플레이스 홀더 데이터(placeholder data)를 요구되어지는 런타임 실행 정보(required runtime execution information)로 바꾸는, 그리고 계속해서 실행하는 코드(code)의 섹션, 또는 스터브(stub)의 섹션으로 제어를 이동시키는 한 원시 기계 명령어로 겹쳐 쓰기(overwritten) 된다.

대표도

도 5

특허청구의 범위

청구항 1.

컴퓨터 시스템의 명령어 컴파일 방법으로서, 이 방법은,

- 런타임 실행 정보를 필요로하는 런타임 컴파일(runtime compilation)을 위한 명령어를 수신하고(301),

- 상기 명령어를 하나 이상의 원시 기계 명령어로 제 1 위치에서 컴파일하며, 이때 상기 하나 이상의 원시 기계 명령어는 요구된 런타임 실행 정보에 대한 플레이스홀더 데이터(placeholder data)를 포함하고(303), 그리고

- 상기 제 1 위치에서의 상기 하나 이상의 원시 기계 명령어를 한 섹션의 원시 기계 명령어로 제어를 이동시키는 원시 기계 명령어로 덮어쓰도록 하며, 런타임(runtime)시에 상기 한 섹션의 원시 기계 명령어가 상기 플레이스홀더 데이터를 요구되는 런타임 실행 정보로 바꾸도록 하는 (307)(309)(311)(313),

이상의 단계를 포함하는 것을 특징으로 하는, 컴파일된 코드에서 동적 클래스 초기화 검사의 비용을 줄이기 위한 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 2.

제 1 항에 있어서, 하나 이상의 원시 기계 명령어를 제 2 위치로 복사하는 단계를 추가로 포함하는 것을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 3.

제 2 항에 있어서, 상기 원시 기계 명령어 섹션은 제 2 위치에 놓인 하나 이상의 원시 기계 명령어의 플레이스홀더 데이터를, 요구되는 런타임 실행 정보로 바꾸는 것을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 4.

제 3 항에 있어서, 상기 원시 기계 명령어 섹션이 제 2 위치에서 요구되는 런타임 실행 정보를 가진 상기 한개 이상의 원시 기계 명령어를 제 1 위치로 복사하는 것을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 5.

제 4 항에 있어서, 상기 원시 기계 명령어 섹션이 제 1 위치에서 요구되는 런타임 실행 정보를 갖는 상기 한 개 이상의 원시 기계 명령어로 제어를 이동시키는 것을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 6.

제 2 항에 있어서, 상기 원시 기계 명령어 섹션이 런타임 실행시 상기 한 개 이상의 원시 기계 명령어를 제 1 위치로 복사하는 것을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 7.

제 6 항에 있어서, 상기 원시 기계 명령어 섹션이 런타임 실행시 제 1 위치에 놓인 상기 한 개 이상의 원시 기계 명령어의 플레이스홀더 데이터를, 요구되는 런타임 실행 정보로 바꾸는 것을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 8.

제 7 항에 있어서, 상기 원시 기계 명령어 섹션은, 제 1 위치에서 요구되는 런타임 실행 정보를 가진 한개 이상의 원시 기계 명령어로 제어를 이동시키는 것을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 9.

제 1 항에 있어서, 클래스가 로딩(loading)되지 않았기 때문에 요구되는 런타임 실행 정보가 컴파일 중 이용될 수 없는 것임을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 10.

제 9 항에 있어서, 상기 요구되는 런타임 실행 정보는 로딩되지 않은 클래스의 인스턴스 변수(instance variable)와 관련되는 것임을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 11.

제 9 항에 있어서, 상기 요구되는 런타임 실행 정보가 로딩되지 않은 클래스의 정적 변수(static variable)에 관련되는 것임을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 12.

제 1 항에 있어서, 상기 요구되는 런타임 실행 정보가 클래스가 런타임 실행시 한 정적 변수를 위해 초기화되었는 지 여부를 나타내는 것임을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 13.

제 12 항에 있어서, 클래스가 초기화되었는 지를 결정하기 위해 런타임 실행 검사(runtime execution check)가 실행되는 것임을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 14.

제 1 항에 있어서, 상기 명령어가 가상 기계 명령어(virtual machine instruction)인 것을 특징으로 하는 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 15.

제 14 항에 있어서, 상기 명령어가 자바 가상 기계 명령어(Java virtual machine instruction)인 것을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 16.

제 15 항에 있어서, 상기 명령어가 getfield, putfield, getstatic, 또는 putstatic 중 하나인 것을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 17.

- 런타임 실행 정보를 필요로하는 런타임 컴파일용 명령어를 수신하도록 하는(301) 컴퓨터 코드,

- 상기 명령어를 한개 이상의 원시 기계 명령어로 제 1 위치에서 컴파일하며, 이때 상기 한 개 이상의 원시 기계 명령어는 요구된 런타임 실행 정보에 대한 플레이스홀더 데이터(placeholder data)를 포함하도록 하는(303) 컴퓨터 코드, 그리고

- 상기 제 1 위치에서의 상기 하나 이상의 원시 기계 명령어를 한 섹션의 원시 기계 명령어로 제어를 이동시키는 원시 기계 명령어로 덮어쓰도록 하며, 런타임(runtime)시에 상기 한 섹션의 원시 기계 명령어가 상기 플레이스홀더 데이터를 요구되는 런타임 실행 정보로 바꾸도록 하는(307)(309)(311)(313) 컴퓨터 코드를 포함함을 특징으로 하는 컴파일된 코드에서 동적 클래스 초기화 검사의 비용을 줄이기 위한 컴퓨터 시스템의 명령어 컴파일을 위한 프로그램을 기록한 컴퓨터 판독 가능 매체.

청구항 18.

컴퓨터 시스템의 명령어 컴파일 방법으로서, 이 방법은,

- 런타임 실행 정보를 필요로하는 런타임 컴파일을 위한 명령어를 수신하고(301),

- 상기 명령어를 한개 이상의 원시 기계 명령어로 제 1 위치에서 컴파일하며, 이때 상기 한 개 이상의 원시 기계 명령어는 요구된 런타임 실행 정보에 대한 플레이스홀더 데이터(placeholder data)를 포함하고(303) ,

- 상기 하나 이상의 원시 기계 명령을 제 2 위치로 복사하고(305),

- 한 원시 기계 명령어 섹션을 발생시키고, 런타임시에 , 제 2 위치에서 하나 이상의 원시 명령어 내 플레이스홀더 데이터를 요구되는 런타임 실행 정보로 바꾸고(309), 제 2 위치에서 요구되는 런타임 실행 정보를 갖는 하나 이상의 원시 기계 명령어를 제 1 위치로 복하하며(311), 그리고 제 1 위치에서 상기 요구되는 런타임 실행 정보를 갖는 하나 이상의 원시 기계 명령어로 제어를 이동시키고(313); 그리고

- 원시 기계 명령어 섹션으로 제어를 이동시키는 별도의 원시 기계 명령어로, 제 1 위치에 있는 상기 한개 이상의 원시 기계 명령어를 덮어쓰는,

이상의 단계를 포함하는 것을 특징으로 하는, 컴파일된 코드에서 동적 클래스 초기화 검사의 비용을 줄이기 위한 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 19.

제 18 항에 있어서, 클래스가 로딩(loading)되지 않았기 때문에 런타임 컴파일 중 요구되는 런타임 실행 정보가 사용될 수 없는 것임을 특징으로 하는 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 20.

제 19 항에 있어서, 요구되는 런타임 실행 정보가 로딩되지 않은 클래스의 인스턴스 변수(instance variable)에 관련되는 것임을 특징으로 하는 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 21.

제 19 항에 있어서, 요구되는 런타임 실행 정보가 로딩되지 않은 클래스의 정적 변수(static variable)에 관련되는 것임을 특징으로 하는 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 22.

제 18 항 내지 제 21 항 중 어느 한 항에 있어서, 요구되는 런타임 실행 정보는, 런타임 실행시 클래스가 정적 변수로 초기화되는 지 여부를 나타내는 것임을 특징으로 하는 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 23.

제 22 항에 있어서, 클래스가 초기화되었는 지를 결정하기 위해 런타임 실행 검사(runtime execution check)가 실행되는 것을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 24.

- 런타임 실행 정보를 필요로하는 런타임 컴파일을 위한 명령어를 수신하도록 하는(301) 컴퓨터 코드,

- 상기 명령어를 한개 이상의 원시 기계 명령어로 제 1 위치에서 컴파일하며, 이때 상기 한 개 이상의 원시 기계 명령어는 요구된 런타임 실행 정보에 대한 플레이스홀더 데이터(placeholder data)를 포함하도록 하는(303) 컴퓨터 코드,

- 상기 하나 이상의 원시 기계 명령을 제 2 위치로 복사하는(305) 컴퓨터 코드,

- 한 원시 기계 명령어 섹션을 발생시키고, 런타임시에 , 제 2 위치에서 하나 이상의 원시 명령어 내 플레이스홀더 데이터를 요구되는 런타임 실행 정보로 바꾸고(309), 제 2 위치에서 요구되는 런타임 실행 정보를 갖는 하나 이상의 원시 기계 명령어를 제 1 위치로 복하하며(311), 그리고 제 1 위치에서 상기 요구되는 런타임 실행 정보를 갖는 하나 이상의 원시 기계 명령어로 제어를 이동시키도록 하는(313) 컴퓨터 코드; 그리고

- 원시 기계 명령어 섹션으로 제어를 이동시키는 별도의 원시 기계 명령어로, 제 1 위치에 있는 상기 한개 이상의 원시 기계 명령어를 덮어쓰도록 하는 컴퓨터 코드를 포함함을 특징으로 하는 컴파일된 코드에서 동적 클래스 초기화 검사의 비용을 줄이기 위한 컴퓨터 시스템의 명령어 컴파일을 위한 프로그램을 기록한 컴퓨터 판독 가능 매체.

청구항 25.

컴퓨터 시스템의 명령어 컴파일 방법으로서, 이 방법은,

- 런타임 실행 정보를 필요로하는 런타임 컴파일을 위한 명령어를 수신하고(301),

- 상기 명령어를 한 개 이상의 원시 명령어로 제 1 위치에서 컴파일하며, 이때 상기 한 개 이상의 원시 기계 명령어는 요구된 런타임 실행 정보에 대한 플레이스홀더 데이터(placeholder data)를 포함하며(303),

- 상기 한개 이상의 원시 기계 명령어를 제 2 위치로 복사하고(305),

- 원시 기계 명령어 섹션을 발생시키며, 실행 시, 제 2 위치에 있는 한개 이상의 원시 기계 명령어를 제 1 위치로 복사하고(311), 제 1 위치에 있는 상기 한개 이상의 원시 기계 명령어내 플레이스홀더 데이터를 요구되는 런타임 실행 정보로 바꾸며(309), 그리고 제 1 위치에 있는 요구되는 런타임 실행 정보를 가진 상기 한개 이상의 원시 기계 명령어에게로 제어를 이동시키고(313), 그리고

- 원시 기계 명령어 섹션에 제어를 이동시키는 별도의 원시 기계 명령어로, 제 1 위치에 있는 상기 한개 이상의 원시 기계 명령어를 덮어쓰는(307),

이상의 단계를 포함하는 것을 특징으로 하는, 컴파일된 코드에서 동적 클래스 초기화 검사의 비용을 줄이기 위한 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 26.

제 25 항에 있어서, 클래스가 로딩되지 않았기 때문에, 런타임 실행 중 요구되는 런타임 실행 정보가 사용될 수 없는 것임을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 27.

제 26 항에 있어서, 요구되는 런타임 실행 정보가 로딩되지 않은 클래스의 인스턴스 변수(instance variable)에 관련되는 것임을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 28.

제 26 항에 있어서, 요구되는 런타임 실행 정보가 로딩되지 않은 클래스의 정적 변수(static variable)에 관련되는 것임을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 29.

제 25 항 내지 제 28 항 중 어느 한 항에 있어서, 상기 요구되는 런타임 실행 정보가 정적 변수에 대한 런타임 실행시 클래스가 초기화되는 지 여부를 나타내는 것임을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 30.

제 29 항에 있어서, 클래스가 초기화되었는 지를 결정하기 위해 런타임 실행 검사(runtime execution check)가 실행되는 것을 특징으로 하는, 컴퓨터 시스템의 명령어 컴파일 방법.

청구항 31.

- 런타임 실행 정보를 필요로하는 런타임 컴파일을 위한 명령어를 수신하도록 하는(301) 컴퓨터 코드,

- 상기 명령어를 한 개 이상의 원시 명령어로 제 1 위치에서 컴파일하며, 이때 상기 한 개 이상의 원시 기계 명령어는 요구된 런타임 실행 정보에 대한 플레이스홀더 데이터(placeholder data)를 포함하도록 하는(303) 컴퓨터 코드,

- 상기 한개 이상의 원시 기계 명령어를 제 2 위치로 복사하도록 하는(305) 컴퓨터 코드,

- 원시 기계 명령어 섹션을 발생시키며, 실행 시, 제 2 위치에 있는 한개 이상의 원시 기계 명령어를 제 1 위치로 복사하고(311), 제 1 위치에 있는 상기 한개 이상의 원시 기계 명령어내 플레이스홀더 데이터를 요구되는 런타임 실행 정보로 바꾸며(309), 그리고 제 1 위치에 있는 요구되는 런타임 실행 정보를 가진 상기 한개 이상의 원시 기계 명령어에게로 제어를 이동시키도록 하는(313) 컴퓨터 코드, 그리고

- 원시 기계 명령어 섹션에 제어를 이동시키는 별도의 원시 기계 명령어로, 제 1 위치에 있는 상기 한개 이상의 원시 기계 명령어를 덮어쓰도록 하는(307) 컴퓨터 코드를 포함함을 특징으로 하는 컴파일된 코드에서 동적 클래스 초기화 검사의 비용을 줄이기 위한 컴퓨터 시스템의 명령어 컴파일을 위한 프로그램을 기록한 컴퓨터 판독 가능 매체.

청구항 32.

컴퓨터 시스템에서의 원시 기계용 비-원시 명령어의 컴파일 방법으로서, 이 방법은,

- 원시 기계 상에서 실행될 비-원시 명령어를 수신하고(301), 이때, 상기 비-원시 명령어가 원시 기계 상에서 실행될 수 있도록 상기 비-원시 명령어가 런타임 실행 정보를 필요로하며,

- 상기 비-원시 명령어를 제 1 세트의 원시 기계 명령어로 컴파일하고, 이때, 상기 제 1 세트의 원시 기계 명령어는 한개 이상의 원시 기계 명령어를 포함하며(303),

- 제 1 세트의 원시 명령어가 상기 원시 기계 상에서 실행될 수 있도록, 원시 기계 상에서 실행될 때 런타임 실행 정보를 제공할 수 있는 제 2 세트의 원시 기계 명령어를 발생시키고, 이때, 상기 제 2 세트의 원시 기계 명령어는 한개 이상의 원시 기계 명령어를 포함하며, 그리고

- 제 1 세트의 원시 기계 명령어가 원시 기계 상에서 실행될 때, 원시 기계 상의 명령어 실행이 제 1 세트의 원시 기계 명령어로부터 제 2 세트의 원시 기계 명령어로 바뀌도록 하는(307)(309)(311)(313),

이상의 단계를 포함하는 것을 특징으로 하는, 컴파일된 코드에서 동적 클래스 초기화 검사의 비용을 줄이기 위한 컴퓨터 시스템에서의 원시 기계용 비-원시 명령어의 컴파일 방법.

청구항 33.

제 32 항에 있어서, 원시 기계 상의 명령어 실행이 제 1 세트의 원시 기계 명령어로부터 제 2 세트의 원시 기계 명령어로 바뀌도록 하는 상기 단계가,

- 원시 기계 상의 명령어 실행 제어를 바꾸도록 원시 기계 상에서 실행될 수 있는 제어 전환 명령어를 발생시키고, 그리고

- 제 1 세트의 가상 머신 명령어의 한개 이상의 명령어를 제어 전환 명령어로 덮어쓰며, 이때, 상기 제어 전환 명령어는 제 1 세트의 가상 기계 명령어로부터 제 2 세트의 가상 기계 명령어로 가상 기계 상의 실행 제어를 바꾸는, 즉, 전환하는,

이상의 단계를 포함하는 것을 특징으로 하는, 컴퓨터 시스템에서의 원시 기계용 비-원시 명령어의 컴파일 방법.

청구항 34.

제 32 항에 있어서, 상기 제 2 세트의 원시 기계 명령어로부터의 한개 이상의 명령어가 상기 제 1 세트의 원시 기계 명령어들 중 한개 이상의 명령어를 덮어쓰는 것을 특징으로 하는, 컴퓨터 시스템에서의 원시 기계용 비-원시 명령어의 컴파일 방법.

청구항 35.

제 32 항에 있어서, 비-원시 명령어가 가상 기계 명령어인 것을 특징으로 하는, 컴퓨터 시스템에서의 원시 기계용 비-원시 명령어의 컴파일 방법.

청구항 36.

제 32 항에 있어서, 상기 비-원시 명령어가 자바 가상 기계 명령어인 것을 특징으로 하는, 컴퓨터 시스템에서의 원시 기계용 비-원시 명령어의 컴파일 방법.

명세서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 컴파일을 하는 소프트웨어(compiling software)에 관한 것이다. 보다 더 상세하게, 본 발명은 컴파일을 마친 코드(compiled code)에서 (예를 들면, 프로그램이 실행되는 시간 중에 동적 클래스 적재 검사(dynamic class loading check)와 동적 클래스 초기화 검사(dynamic class initialization check) 의 비용을 줄이는 기술에 관한 것이다.

Java™ 프로그래밍 언어는 Sun Microsystems, Inc 에 의하여 개발된 객체-지향 고-레벨 프로그래밍 언어(object-oriented high level programming language)이고 ; Java™ 프로그래밍 언어는 크기가 작은 장치(예를 들면, 페이저, 셀 방식에 의한 전화기, 및 스마트 카드 등)에서 슈퍼컴퓨터까지의 범위에 걸쳐있는 광의의 컴퓨터(computer)에서 실행됨에 있어서 충분하게 이식될 수 있게 설계된 언어이다. Java(그리고, 다른 언어)에 의하여 만들어진 컴퓨터 프로그램(computer program)은, Java 가상 기계에 의하여 실행을 위한 가상 기계 명령어(virtual machine instruction)로 컴파일(compile)될 수 있다. 일반적으로, Java 가상 기계(Java virtual machine)는 가상 기계 명령어를 디코딩(decoding)하고 실행(execution)하는 인터프리터(interpreter)를 의미한다.

자바 가상 기계(Java virtual machine)를 위한 가상 기계 명령어는 바이트 코드(byte code)인데, 이것은 하나 이상의 바이트를 포함한다. 바이트 코드(byte code)는 "클래스 파일(class file)" 이라고 불리는 특별한 파일 형식으로 저장되는데, 여기서 "클래스 파일(class file)" 은 클래스에 있는 방법을 위한 바이트 코드(byte code)를 포함한다. 클래스에 있는 방법을 위한 바이트 코드(byte code)에 추가하여, 클래스 파일은 심벌 테이블(symbol table)뿐 아니라 다른 부속의 정보를 포함한다.

하나 이상의 클래스 파일에서 자바 바이트 코드(Java byte code)로써 구현되는 컴퓨터 프로그램은 플랫폼(platform)에 독립적이다. Java 가상 기계의 구현을 실행할 수 있는 어떤 컴퓨터에서, 컴퓨터 프로그램은 실행될 수 있고, 변경되지 않는다. Java 가상 기계는 "일반(generic)" 컴퓨터를 위한 소프트웨어 에뮬레이터(software emulator)인데 ; 여기서 "일반(generic)" 컴퓨터라는 것은 Java 가상 기계를 위한 컴퓨터 프로그램이 플랫폼(platform)과는 독립적이도록 하는 데 중요한 요인이다.

자바 가상 기계(Java virtual machine)는 소프트웨어 인터프리터(software interpreter)로써 구현될 수 있다. 종래에 있었던 인터프리터(interpreter)는 실행 중 한 번에 하나의 명령씩 인터프릿을 마친 프로그램(interpreted program)을 디코딩(decoding)을 하고 실행(execution)을 하는데 ; 이와는 현저하게 차이가 있는 컴파일러(compiler)로서, 실행 전에 소스 코

드(source code)를 원시 기계 명령어(native machine instruction)로 디코딩(decoding)함으로써, 실행 중에는 디코딩(decoding)을 실행하지 않는 것이다. Java 가상 기계는 런타임에 컴파일(compile)을 위한 인터프리터(interpreter)와 컴파일러(compiler) 모두 포함할 수 있다. 일반적으로, 자바 가상 기계(Java virtual machine)는 자바 프로그래밍 언어를 제외한 다른 프로그래밍 언어(예를 들면, C++ 프로그래밍 언어)에서 이용될 것이다.

런타임에 컴파일(compile)을 할 때, 자바 방법(Java method)은 적재되지 않았던 클래스의 필드(field)에 접근할 수 있다(판독을 위하여, 또는 기록을 위하여). 런타임 성능을 증가시킬 목적으로 런타임(runtime)에 가상 기계 명령어(virtual machine instruction)를 원시 기계 명령어(native machine instruction)로 컴파일(compile)하는 자바 가상 기계(Java virtual machine)는, 컴파일(compile) 시간에 이용 가능한 모든 정보를 갖지 않는다. 예를 들면, 클래스가 적재되지 않는다면, 클래스의 실체(instance) 내에서 가변적인 실체(instance variable)를 위한 오프셋(offset)을 결정하는 것은 어렵다.

프로그램이 실행되는 시간에서 실행에 대한 이용 가능하지 않은 정보(unavailable runtime execution information)의 문제에 대한 해결 방법은, 클래스의 필드에 접근하기 전에 클래스를 적재하고 초기화하는 것에 대하여 검사하는 원시 기계 명령어를 생성시키는 것이다. 실제로, 이러한 해결 방법은 이롭기보다는 훨씬 이롭지 못할 수 있다. 런타임 검사(runtime check)는, 컴파일(compile)이 시스템의 기능을 향상시키기보다는 시스템의 기능을 떨어뜨릴 수 있다.

프로그램이 실행되는 시간에서 실행에 대한 이용 가능하지 않은 정보(unavailable runtime execution information)의 문제에 대한 또 하나의 해결 방법은, 어떤 클래스를 적재하고 가능하다면 초기화하는 것인데 ; 이러한 클래스에 대한 런타임 실행 정보는 가상 기계 명령어를 컴파일(compile)하는데 필요하다. 이러한 해결 방법은 마치 좋은 해결 방법인 듯 한데, 프로그램이 실행되는 시간의 실행(runtime execution)에서 클래스를 적재하는 것이 필요하지 않다면, 시스템의 속도를 떨어뜨리는 것은 불필요할 수 있다. 예를 들면, 클래스 A 의 실체의 필드를 변경시키는 문장은 X=0 일 때만 실행된다면, (클래스 A 는 다른 이유로 적재되지 않는다고 가정하면) 클래스 A 는 X=0 일 때 뒤늦게 적재될 수 있다. 추가로, 자바 런타임 환경은 클래스가 필요할 때만 런타임에서 클래스를 동적으로 적재하도록 설계되기 때문에, 클래스를 불필요하게 초기화하는 것은 의미에 있어서 정확하지 않다.

발명이 이루고자 하는 기술적 과제

일반적으로, 본 발명의 실시예에 의하여, 컴파일을 마친 코드(compiled code)에서 동적 클래스 적재 검사(dynamic class loading check)와 동적 클래스 초기화 검사(dynamic class initialization check) 등의 비용을 줄이는 혁신적인 기술을 제공한다. (필드에 대한 오프셋과 같은) 프로그램이 실행되는 시간에 요구되어지는 실행 정보(required runtime execution information)가 컴파일(compile)을 위한 시간에서 이용 가능하지 않더라도, 가상 기계 명령어를 하나 이상의 원시 기계 명령어로 컴파일(compile)한다. 원시 기계 명령어는 플레이스 홀더 데이터(예를 들면, 모조 값)만을 가질 수 있는데, 프로그램이 실행되는 시간에서 요구되어지는 실행 정보가 되어야만 한다. 프로그램이 실행되는 시간(runtime)에서 플레이스 홀더 데이터(placeholder data)만을 프로그램이 실행되는 시간에서 요구되어지는 실행 정보(required runtime execution information)로 바꾸는 코드(code)나 스터브(stub)의 섹션으로 제어를 이동시키는 원시 기계 명령어로써 ; 원시 기계 명령어의 모든 부분 또는 일부분을 그 위에다 기재할 수 있다.

따라서, 원시 기계 명령어가 실행되는 첫 번째 시간에, 클래스는 필요하다면 적재될 수 있고 초기화될 수 있다. 하지만, 원시 기계 명령어가 실행되는 첫 번째 시간에, 프로그램이 실행되는 시간에서 요구되어지는 실행 정보(required runtime execution information)는 원시 기계 명령어로 패치(patch)되기 때문에, 원시 기계 명령어의 결과로써 일어나는 실행은 클래스 적재 검사와 클래스 초기화 검사 등에 대한 비용을 일으키지 않는다. 본 발명에 의하여, 프로그램이 미리 실행되는 시간에서의 컴파일(compile)에 대한 런타임 기능을 공급한다. 추가로써, 시스템의 의미(semantic)는 적합한 시간에서만 클래스를 초기화하는 것에 의하여 유지될 수 있다. 본 발명의 몇몇 실시예가 하기에서 기술될 것이다.

하나의 실시예에 있어서, 명령어를 컴파일하는 컴퓨터에 의하여 구현된 방법(computer-implemented method)은 실행되는 동안의 실행 정보(runtime execution information)를 요구하는 런타임 컴파일(runtime compilation)을 위한 명령어를 수신하는 단계를 포함한다. 명령어는 제 1 위치에서 하나 이상의 원시 기계 명령어로 컴파일(compile)되는데, 기계 명령어는 실행되는 동안에 요구되어지는 실행 정보(required runtime execution information)를 위한 플레이스 홀더 데이터(placeholder data)를 포함한다. 실행되는 중에, 플레이스 홀더 데이터(placeholder data)를, 실행되는 동안에 요구되어지는 실행 정보(required runtime execution information)로 바꾸는 원시 기계 명령어의 섹션으로 제어(control)를 이동시키는 원시 기계 명령어로, 제 1 위치에 있는 원시 기계 명령어의 초기 부분에 중복 기재한다. 선호되는 실시예에서, 컴파일(compile)되는 명령어는 자바 가상 기계 명령어이다.

또 하나의 실시예에 있어서, 명령어를 컴파일하는 컴퓨터에 의하여 구현된 방법(computer-implemented method)은 실행되는 동안의 실행 정보(runtime execution information)를 요구하는 런타임 컴파일(runtime compilation)을 위한 명령어를 수신하는 단계를 포함한다. 명령어는 제 1 위치에서 하나 이상의 원시 기계 명령어로 컴파일(compile)되는데, 원시 기계 명령어는 실행되는 동안에 요구되어지는 실행 정보(required runtime execution information)를 위한 플레이스 홀더 데이터(placeholder data)를 포함한다. 하나 이상의 기계 명령어의 적어도 일부분은 제 2 위치로 복사된다. 실행되는 중에, 제 2 위치의 하나 이상의 원시 기계 명령어에 있는 플레이스 홀더 데이터(placeholder data)를, 실행되는 동안에 요구되어지는 실행 정보(required runtime execution information)로 바꾸는, 그리고 제 2 위치의 실행되는 동안에 요구되어지는 실행 정보(required runtime execution information)를 갖춘 하나 이상의 원시 기계 명령어를 제 1 위치로 복사하는, 그리고 제 1 위치의 실행되는 동안에 요구되어지는 실행 정보(required runtime execution information)를 갖춘 하나 이상의 원시 기계 명령어로 제어를 이동시키는 원시 기계 명령어(native machine instruction)의 섹션(section)을 생성시킨다. 원시 기계 명령어의 섹션으로 제어(control)를 이동시키는 원시 기계 명령어로, 제 1 위치에 있는 하나 이상의 원시 기계 명령어를 중복 기재한다.

또 하나의 실시예에 있어서, 명령어를 컴파일하는 컴퓨터에 의하여 구현된 방법(computer-implemented method)은 실행되는 동안의 실행 정보(runtime execution information)를 요구하는 런타임 컴파일(runtime compilation)을 위한 명령어를 수신하는 단계를 포함한다. 명령어는 제 1 위치에서 하나 이상의 원시 기계 명령어로 컴파일(compile)되는데, 기계 명령어는 실행되는 동안에 요구되어지는 실행 정보(required runtime execution information)를 위한 플레이스 홀더 데이터(placeholder data)를 포함한다. 원시 기계 명령어의 적어도 일부분은 제 2 위치로 복사된다. 실행되는 중에, 제 2 위치의 하나 이상의 원시 기계 명령어를 제 1 위치로 복사하는, 그리고 제 1 위치의 하나 이상의 원시 기계 명령어에 있는 플레이스 홀더 데이터(placeholder data)를, 실행되는 동안에 요구되어지는 실행 정보(required runtime execution information)로 바꾸는, 그리고 제 1 위치의 실행되는 동안에 요구되어지는 실행 정보(required runtime execution information)를 갖춘 하나 이상의 원시 기계 명령어로 제어를 이동시키는 원시 기계 명령어(native machine instruction)의 섹션(section)을 생성시킨다. 원시 기계 명령어의 섹션으로 제어(control)를 이동시키는 원시 기계 명령어로, 제 1 위치에 있는 하나 이상의 원시 기계 명령어의 초기 부분은 중복 기재된다.

본 발명의 다른 특징과 다른 이점 등은, 부속된 도면과 관련하여 다음의 상세한 기술의 요약에서 쉽사리 분명해질 것이다.

정의(definition)

기계 명령어(또는 명령어) - 연산 코드(operation code, OP code)에 의하여, 그리고 선택적으로 하나 이상의 피연산자 등에 의하여 분류된 연산(operation)을 실행하도록 컴퓨팅 장치(computing device)를 이끄는 명령어

가상 기계 명령어(virtual machine instruction) - 소프트웨어에 의하여 에뮬레이션(emulation)되는 마이크로프로세서, 또는 컴퓨터 아키텍처 등을 위한 명령어 (또한 가상 코드(virtual code)로 불림)

원시 기계 명령어(native machine instruction) - 특별한 마이크로프로세서, 또는 컴퓨터 아키텍처 등을 위하여 설계된 명령어 (또는 원시 코드(native code)로 불림)

방법(method) - 소프트웨어 경로 (또는, 함수, 서브-경로, 절차, 그리고 멤버 함수 등으로 불림)

필드(field) - 정적 변수와 실체 변수 등을 포함하는 클래스 타입의 변수

실체 변수(instance variable) - 실체마다 한 번 존재하는 클래스의 실체에 있는 변수

정적 변수(static variable) - 클래스마다 한 번 존재하는 클래스의 변수

런타임 컴파일(runtime compilation) - 런타임에 실행되는 코드의 컴파일

프로그램이 실행되는 시간에서 실행에 대한 정보(runtime execution) - 런타임에 실행되는 코드의 실행

발명의 구성

다음에 있는 기술에 있어서, 본 발명은 선호되는 실시 예에 관하여 기술되는데, 이러한 실시 예에서 컴파일(compile)을 마친 자바 가상 기계 명령어(Java virtual machine instruction)를 위하여 동적 클래스 적재 검사(dynamic class loading check)와 동적 클래스 초기화 검사(dynamic class initialization check) 등의 비용을 줄인다. 상세하게, 컴파일(compile)을 마친 자바 가상 기계 명령어(Java virtual machine instruction)를 위하여 런타임 클래스 적재(runtime class loading)와 초기화 검사(initialization check) 등을 실행할 목적으로, IBM 개인 컴퓨터와 그 호환이 있는 컴퓨터 등을 위하여 생성될 수 있는 실제 원시 기계 명령어(actual native machine instruction)를 나타내는 실례를 기술할 것이다. 하지만, 본 발명은 어떤 특별한 언어, 컴퓨터 아키텍처, 또는 특별한 구현 등에 제한되지 아니 한다. 그러므로, 다음에 있을 실시예의 기술은 설명을 위한 것이지 제한이 없다.

도 1 은, 본 발명의 실시 예로써 소프트웨어를 실행시키는데 이용될 수 있는 컴퓨터 시스템(computer system)의 실례를 나타낸다. 도 1 은 디스플레이(3), 스크린(5), 본체(7), 키보드(9), 및 마우스(11) 등을 포함하는 컴퓨터 시스템(1)(computer system)을 나타낸다. 마우스(11)(mouse)는 그래픽 환경에 의한 사용자 인터페이스(graphical user interface)와 상호 작용하는 하나 이상의 버튼을 포함할 수 있다. 본체(7)에 의하여, 본 발명을 구현하는 컴퓨터 코드(computer code), 본 발명에서 이용을 위한 데이터(data), 그리고 유사한 것 등을 합체하는 소프트웨어 프로그램(software program)을 저장하고 검색하는데 이용될 수 있는 CD-ROM 드라이브(13), 시스템 메모리, 및 하드드라이브 등을 수용한다(도 2 를 참조). CD-ROM(15)은 실례로써 컴퓨터에 의하여 판독 가능한 저장 매체로써 나타나 있는데, 플로피 디스크, 테이프, 플래쉬 메모리, 시스템 메모리, 및 하드드라이브 등을 포함하는 컴퓨터에 의하여 판독 가능한 다른 저장 매체가 이용될 수 있다. 추가로, (예를 들면, 인터넷(Internet)을 포함하는 네트워크에서) 반송파에서 구현되는 데이터 신호가, 컴퓨터에 의하여 판독 가능한 저장 매체로 될 수 있다.

도 2 는, 본 발명의 실시 예로써 소프트웨어를 실행하는데 이용되는 컴퓨터 시스템(1)(computer system)의 시스템 블록 다이어그램을 나타낸다. 도 1 에서 나타난 것처럼, 컴퓨터 시스템(1)은 모니터(3), 키보드(9), 마우스(11) 등을 포함한다. 중앙 프로세서(51), 시스템 메모리(53), (하드드라이브와 같은) 고정 스토리지(기억 장치)(55), (CD-ROM 드라이브와 같은) 착탈 가능한 스토리지(기억장치)(57), 디스플레이 어댑터(59), 사운드 카드(61), 스피커(63), 및 네트워크 인터페이스(65) 등과 같은 서브-시스템(subsystem)을 덧붙여서 컴퓨터 시스템(1)은 포함한다. 본 발명의 이용에 적합한 다른 컴퓨터 시스템은 추가의 서브-시스템이나 보다 더 적은 서브-시스템 등을 포함할 수 있다. 예를 들면, 또 하나의 컴퓨터 시스템은 (멀티-프로세서 시스템과 같은) 하나 이상의 프로세서(51)나 캐시 메모리 등을 포함할 수 있다.

컴퓨터 시스템(1)의 시스템 버스 아키텍처는 화살표(67)에 의하여 나타난다. 하지만, 이러한 화살표는 서브-시스템을 연결하는데 이용되는 상호 연결 설계의 실례를 나타낸다. 예를 들면, 로컬 버스는 중앙 프로세서를 시스템 메모리와 디스플레이 어댑터 등에 연결하는데 이용될 수 있다. 도 2 에서 나타난 컴퓨터 시스템(1)은 본 발명의 이용에 적합한 컴퓨터 시스템의 실례로써만 이용된다. 서브-시스템에서 서로 다른 구성을 가지는 다른 컴퓨터 아키텍처는 또한 이용될 수 있다.

일반적으로, 자바 프로그래밍 언어에서 기록되는 컴퓨터 프로그램은, 자바 가상 기계에 의하여 실행되는 바이트 코드나 자바 가상 기계 명령어 등으로 컴파일(compile)된다. 인터프리테이션(interpretation)을 위한 자바 가상 기계로의 입력이 되는 클래스 파일에, 바이트 코드(byte code)는 저장된다. 도 3 은, 인터프리터(interpreter)인 자바 가상 기계에 의한 실행을 통하여, 자바 소스 코드의 단순한 일부분의 진행을 나타낸다.

자바 소스 코드(101)(Java source code)는 Java 에 의한 기록된 전통적으로 이용되는 Hello World 프로그램을 포함한다. 그 다음에, 소스 코드를 바이트 코드로 컴파일(compile)하는 바이트 코드 컴파일러(103)(byte code compiler)로 소스 코드는 입력된다. 소프트웨어에 의하여 에뮬레이션되는 컴퓨터(software emulated computer)에 의하여 바이트 코드는 실행되는데, 여기서 바이트 코드는 가상 기계 명령어가 된다. 일반적으로, 가상 기계 명령어는 일반적(예를 들면, 어떤 특별한 마이크로프로세서나 컴퓨터 아키텍처를 위하여 설계되지 않는다)이지만, 요구되지는 않는다. 바이트 코드 컴파일러(byte code compiler)는 자바 프로그램을 위한 바이트 코드를 포함하는 자바 클래스 파일(105)(Java class file)을 출력한다.

자바 클래스 파일은 자바 가상 기계(107)(Java virtual machine)로 입력된다. 자바 가상 기계는 자바 클래스 파일에서 바이트 코드를 디코딩(decoding)하고 실행시키는 인터프리터(interpreter)를 의미한다. 자바 가상 기계는 인터프리터이지만, 마이크로프로세서나 컴퓨터 아키텍처를 소프트웨어(가령 하드웨어로는 존재하지 않는 마이크로프로세서나 컴퓨터 아키텍처)로 에뮬레이션(emulation)하기 때문에 자바 가상 기계는 가상 기계를 의미한다.

자바 클래스(그리고 인터페이스)가 동적으로 적재되고, 연결되며, 초기화된다. 적재(loading)는 클래스의 이진 형식(가령 클레스 파일)을 찾는, 그리고 상기 이진 형식으로부터 상기 클래스를 대표할 Class 객체를 구성하는 시스템의 처리이다.

Class 라는 클래스는 클래스의 구조를 저장하거나 대표하는 클래스이다. 연결(linking)은 클래스의 이진 형식을 얻는, 그리고 클래스의 이진 형식을 실행할 수 있도록 시스템의 런타임 상태로 클래스의 이진 형식을 결합하는 처리이다. 클래스의 초기화(initialization)는, 클래스의 정적 초기화 장치(initializer)와 클래스에서 선언된 정적 필드를 위한 초기화 수단(initializer) 을 실행하는 것을 포함한다.

각각의 자바 클래스는 이와 관련이 있는 상수 풀(constant pool)을 가진다. 상수 풀(constant pool)은 자바 클래스 파일에 저장되고, 심벌 테이블과 비슷한 작용으로써 이용된다. 일반적으로, 상수 풀(constant pool)에 있는 각각의 엔트리는, 1 에서 시작하여 상수 풀에 있는 엔트리의 숫자로 올라가는 숫자에 의하여 색인 된다. 한 클래스를 위한 방법이 상기 색인(index)에 의하여 상수 풀에 있는 엔트리에 접근하며, 그리고 한 클래스를 위한 방법은 다른 클래스를 위한 상수 풀에는 접근할 수 없다.

리터럴 상수(literal constant)를 저장하는 상수 풀(constant pool)에 추가하여, 상수 풀은 클래스, 방법, 필드, 및 인터페이스를 기호로서 저장한다. 이러한 엔트리를 기호로써 저장하는 것에 의하여, 엔트리를 확인하는 이름이 저장되고, 물리적 주소는 저장되지 않는다는 것을 의미한다. 달리 말하면, 클래스 A 가 필드 F 를 가진다면, (F 에 대한 타입 서명(type signature)과 함께) A 와 F 에 대한 이름 둘 다 상수 풀(constant pool)에 저장될 수 있다. 이름(name)을 저장하고 주소(address)를 저장하지 않는 것에 의하여, 자바 런타임 시스템(Java runtime system)은 프로그램이 실행되는 시간에 동적으로 물리적 주소(physical address)에 대한 기호 참조(symbolic reference)를 해결한다.

도 4 는 자바 런타임 시스템(Java runtime system)의 구현에 대한 요소(component)를 나타낸다. 자바 가상 기계에 대한 구현은 자바 런타임 시스템으로써 공지되어 있다. 자바 프로그램을 실행할 목적으로, 자바 런타임 시스템(201)(Java runtime system)은 자바 클래스 파일(203)(Java class file), 표준적으로 내장된 자바 클래스(205)(standard build-in Java class), 및 원시 방법(207)(native method) 의 입력을 수신할 수 있다. 표준적으로 내장된 자바 클래스(205)(standard build-in Java class)는 스레드(thread), 스트링, 및 그 밖의 것 등과 같은 객체를 위한 클래스가 될 수 있다. 원시 방법(native method)은 자바 프로그래밍 언어를 제외하고 다른 언어로 기록될 수 있다. 원시 방법(native method)은 일반적으로 동적 링킹 라이브러리(dynamic linking libraries, DLLs) 또는 공유된 라이브러리(shared libraries) 내에 저장된다.

자바 런타임 시스템은 운영 체제(209)(operating system)와 인터페이스로 연결되어 있다. 예를 들면, 입력/출력 기능은 운영 체제에 의하여 다루어질 수 있는데, 자바 클래스 파일(203)(Java class file), 표준적으로 내장된 자바 클래스(205)(standard build-in Java class), 및 원시 방법(207)(native method) 등과의 인터페이스(interface)를 자바 런타임 시스템에 공급하는 것을 포함한다.

동적 클래스 적재기와 검증기(211)(dynamic class loader and verifier)는 운영 체제를 통하여 메모리(213)에 자바 클래스 파일(203), 표준적으로 내장된 자바 클래스(205)를 적재한다. 추가로, 동적 클래스 적재기와 검증기(211)는 자바 클래스 파일에서 바이트 코드의 정확을 검증할 수 있는데, 탐지되는 어떤 오류도 보고한다.

원시 방법 연결기(215)(native method linker)는 운영 체제(209)(operating system)를 통하여 원시 방법(207)(native method)내 링크를 자바 런타임 시스템으로 연결시키고, 메모리(213)(memory)내에 상기 원시 방법을 저장한다. 도 4 에서 나타난 것처럼, 메모리(213)는 자바 클래스를 위한 클래스와 방법 영역(class and method area) 그리고 원시 방법을 위한 원시 방법 영역(native method area) 을 포함할 수 있다. 메모리(213) 내 상기 클래스 및 방법 영역은 한 가베지-수집 히프(garbage- collected heap)에 저장될 수 있다. 새로운 객체가 발생됨에 따라 이들 객체는 가베지-수집 히프(garbage- collected heap)에 저장된다. 공간을 더 이상 사용되고 있지 않은 때, 애플리케이션(application)이 아닌 자바 런타임 시스템이 상기 가베지-수집 히브 내 메모리를 리클레임하게 된다.

도 4 에서 자바 런타임 시스템의 중심은, 실행 엔진(execution engine)(217)이다. 상기 실행 엔진(execution engine)(217)은 메모리(213)에서 저장된 명령어를 실행하고, 실행 엔진(execution engine)은 소프트웨어, 또는 하드웨어, 또는 이들 두 개의 조합에 의해 구현될 수 있다. 상기 실행 엔진(execution engine)은 객체-지향 애플리케이션을 지원하고, 개념적으로 자바 스레드(Java thread) 각각에 대하여 하나씩인, 동시에 실행되는 다수의 실행 엔진은 있다. 실행 엔진(217)은 또한 보조 코드(221)(support code)를 이용할 수 있다. 상기 보조 코드는 예외(exception), 스레드(thread), 보완(security), 및 그 밖의 것 등과 관련이 있는 기능을 공급할 수 있다.

일반적으로, 실행 엔진(217)은 인터프리터와 컴파일러 모두를 포함한다. 상기 실행 엔진은 시스템의 기능을 증가시킬 목적으로 방법과 그 일부분을 컴파일(compile)할 수 있다. 그러므로, "런타임(runtime)"은 컴파일(compilation)과 실행(execution) 모두를 포함할 수 있다. 런타임 컴파일은 자바 바이트 코드가 컴파일될 수 있는 시간 프레임이며, 런타임 실행

은 자바 바이트코드(컴파일 되었든 그렇지 않았든)가 실행되는(자바 바이트 코드 내 결정이 계산되고 수행되는) 시간 프레임이다. 각각의 스레드 내에서 런타임 컴파일(runtime compilation)과 런타임 실행(runtime execution)으로부터의 되풀이되는 이동이 있을 수 있으며 이 같은 이동은 스레드와는 독립적일 수 있다.

도 5 는, 본 발명의 실시 예의 고-레벨 순서 도를 나타낸다. 런타임 실행 정보(runtime execution information)를 요구하는 단일 명령어가 실행 런타임 검사를 효율적으로 실행하도록 어떻게 컴파일(compile)될 수 있는지를 대표하는 흐름도를 나타낸다. 추가로, 본 발명의 범위에서 벗어나지 않으면서 단계는 추가되고, 없어지고, 결합될 수 있다.

일반적으로, 명령어를 컴파일(compile)하는 시스템은 명령어를 검색하고 컴파일하는 처리를 통하여 반복된다. 도 5 의 흐름도는 단일 명령어를 위한 처리를 나타낸다. 단계(301)에서, 시스템은 런타임에 정보를 요구하는 런타임 컴파일을 위한 명령어를 수신한다. 실례로써, 명령어는 적재되지 않았던 클래스의 필드에 접근할 수 있다. 클래스는 컴파일을 보조하기 위하여 적재되지만, 런타임 실행에서 이러한 적재는 필요하지 않기 때문에 이러한 적재는 성능의 속도를 떨어뜨릴 수 있다.

시스템은 단계(303)에서 명령어를 하나 이상의 원시 명령어로 컴파일한다. 필요로 하는 런타임 실행 정보가 존재하기 때문에, 플레이스 홀더 데이터가 원시 명령어에 위치하여 질 수 있다. 플레이스 홀더 데이터(placeholder data)는, 랜덤 데이터, 또는 런타임 실행 중에 어떤 정보를 표시하도록 선택되는 데이터가 될 수 있다. 어떤 실시 예에 있어서, 플레이스 홀더 데이터는 더미(dummy) 값이다.

단계(305)에서, 상기 시스템은 제 1 위치에서 제 2 위치로 원시 명령어를 복사한다. 제 1 위치는 일반적으로 원시 명령어를 실행할 수 있는 위치이고, 제 2 위치는 런타임 중에 명령어를 조정할 수 있는 버퍼(buffer)이다. 한 코드의 섹션으로 제어를 이동시키도록, 상기 시스템이 제 1 위치에서 원시 명령어를 겹쳐 쓰기 한다. 제어를 이동시키는 메커니즘은 jump, goto, 또는 call 을 포함하는 기능을 실행시키는 원시 명령어의 하나일 수 있다. 일반적으로, 제어를 이동시키는 원시 명령어는 모든 원시 명령어를 겹쳐 쓰기 하지 않는 짧은 명령어(short instruction)이다. 따라서, 일정 실시 예에 있어서, 겹쳐 쓰기 된 원시 머신 명령어의 일부분만이 제 2 위치로 복사된다.

단계(309)에 있어서, 상기 시스템은 상기 원시 명령에 있는 플레이스 홀더 데이터를 런타임 실행시에 필요한 런타임 실행 정보로 대체시킬 코드 섹션 내에 코드를 위치시킨다. 예를 들면, 필요한 런타임 정보가 실체 변수로의 오프셋이라면, 상기 섹션에 있는 코드는 런타임 실행에서 실체 변수로의 오프셋을 결정할 수 있다. 그 다음에, 실체 변수로의 오프셋이 원시 명령어를 완성하는데 이용될 수 있는 원시 명령어로 패치(patch)될 수 있다.

단계(311)에 있어서, 상기 시스템은 실행시간 실행시 제 1 위치로 제 2 위치에 있는 원시 명령어를 복사할 수 있는 코드의 섹션 내에 코드를 위치시킨다. 따라서, 상기 완성을 마친 원시 명령어는 컴파일된 코드로 이들의 본래 위치로 다시 복사될 수 있다. 원시 명령어를 다시 복사하는 것은, 단계(307)에서 생성되었던 제어를 이동시키는 원시 명령어를 겹쳐 쓰기 할 것이다. 어떤 실시 예에 있어서, 제 2 위치의 원시 기계 명령어는 런타임 실행에서 제 1 위치로 되 복사되도록 하여 요구되어지는 런타임 실행 정보는 삽입될 수 있도록 한다.

단계(313)에 있어서, 상기 시스템은 프로그램 런타임 실행시, 제 1 위치의 원시 명령어로 제어를 이동시키는 코드의 섹션에 코드를 위치시킨다. 요약하면, 원시 명령어를 처음 실행시키는 시간에, 상기 실시 예가 미완성인 원시 명령어를 패치(patch)하지만, 원시 명령어의 뒤이은 실행은 올바르게 컴파일된 형식으로 실행된다. 이러한 실행에 의하여, 본 발명의 실시 예는 런타임 클래스 적재 검사(runtime class loading check)와 런타임 클래스 초기와 검사(runtime class initialization check), 또는 어느 하나에 대한 신속한 실행을 달성한다.

도 5 가 본 발명 실시 예의 흐름도를 설명하였으나, 상기 흐름도는 원시 명령어가 상기 실시 예에서 어떻게 나타날 수 있는지를 분석하는 데 도움이 될 수 있다. 도 6A 는 원시 명령어(401)의 순서를 나타낸다. 동일한 목적을 위하여, 각각의 기계 명령어는 순서 적으로 꼬리표를 가진다. (꼬리표 7 와 8 등을 가진) 원시 명령어(403)는 한 명령어를 컴파일하여 생성되었다. 이제, 원시 명령어를 생성시켰던 명령어는 런타임 실행 정보를 요구하였다고 가정한다.

도 6B 는, 런타임 클래스 적재 검사(runtime class loading check)와 런타임 클래스 초기와 검사(runtime class initialization check)를 포함하는 도 6A 의 원시 명령어의 순서를 나타낸다. 도 6A 와 도 6B 등과 같은 참조 번호는 비슷한 구조를 나타낸다. 원시 명령어(403)는 원시 명령어(405)의 섹션으로 제어를 이동시키는 점프(jump)를 포함한다. 이러한 섹션은 원래의 원시 명령어(403)의 복사, 원시 명령어의 복사를 고정시키는 코드, 복사를 그 원래 위치로 다시 패치하는 코드, 및 원시 명령어(403)로 다시 제어를 이동시키는 점프를 포함한다. 원시 명령어의 섹션은 도 5 에 있는 단계(309), 단계(311), 및 단계(313) 에 의하여 생성될 수 있다.

원시 명령어(405)의 섹션은 상기 방법이 컴파일된 후에 생성될 수 있다. 상기 M 과 B 라는 꼬리표가 제공되어 제어가 어는 곳으로 이동되는 가를 나타내도록 한다. 그러므로, 요구되어지는 런타임 실행 정보로 상기 원시 명령어를 완성시키기 위해 처음 원시 명령어(403)가 실행되는 때 상기 원시 명령어 섹션이 실행될 뿐이다.

도 7A 와 도 7B 는, 자바 가상 기계에서 방법을 컴파일하는 흐름도를 나타낸다. 단계(501)에서, 시스템은 컴파일할 가상 기계 명령어를 검색한다. 일단 상기 가상 기계 명령어가 검색되면, 상기 시스템은 단계(503)에서 가상 기계 명령어가 필드에 접근하는지를 결정한다. 접근이라는 것(accessing)은 가상 기계 명령어가 런타임 실행 정보를 요구할 수 있는 한 가지 방식이다. 하지만, 본 발명은 다른 타입의 런타임 실행 정보에 적용될 수 있기도 한 것이다.

가상 기계 명령어가 클래스 필드에 접근하지 않는다면, 가상 기계 명령어는 단계(505)에서 적재된다. 그렇지 않다면, 시스템은 필드의 클래스가 단계(507)에서 적재되는지에 대한 결정을 한다. 클래스가 적재되었다면, 시스템은 어떤 필요한 오프셋도 계산할 수 있으며, 단계(505)로 진행된다. 추가로, 시스템은 클래스가 적재는 되었지만 정적 변수를 위해 초기화되지는 않았는지를 검사한다. 상기 클래스가 초기화되지 않았다면, 런타임 실행 검사(runtime execution check)는 클래스가 초기화되었는 가를 결정하도록 실행될 수 있다. 상기 클레스가 클래스의 실체(한 경우)를 생성시키도록 클래스를 초기화하지 않을 것이기 때문에, 상기와 같은 검사는 상기 클레스의 객체 변화에 대해서는 필요하지 않다.

클래스가 적재되지 않았다면, 상기 시스템은 단계(509)에서 요구되어지는 런타임 실행 정보의 위치에 플레이스 홀더 데이터를 사용하여 가상 기계 명령어를 컴파일한다. 상기 시스템은 또한 단계(511)에서 생성된 원시 기계 명령어의 위치를 표시하여, 상기 방법(컴파일되었다면) 후에 원시 기계 명령어를 완성시키기 위한 코드가 생성될 수 있도록 한다(예를 들면, 도 6B 의 코드(405)의 섹션). 마커(marker)가 도 6B 에 있는 M 에 의하여 지정된 위치를 기록할 수 있다.

단계(513)에서, 상기 시스템은 컴파일하기 위한 더 이상의 가상 기계 명령어가 존재하는지에 대한 결정을 한다. 존재한다면, 시스템은 단계(501)를 실행한다. 컴파일할 가상 기계 명령어가 그 이상 존재하지 않는다면, 단계(511)에서 표시되는 미완성의 원시 기계 명령어를 제외하고, 상기 방법이 컴파일된다.

이제 도 7B 에 관하여, 단계(511)에서 남아 있는 어떤 마커(marker)가 존재하는지에 대한 첫 번째 검사에 의하여, 시스템은 마커 각각을 처리한다. 남아 있는 마커가 존재한다면(컴파일을 마친 방법에 대하여 하나 이상의 마커가 존재한다면), 단계(553)에서 상기 시스템이 상기 마커된 원시 기계 명령어를 버퍼로 복사한다. 상기 버퍼는 상기 방법이 컴파일되었을 때 생성된 원시 기계 명령어의 끝의 위치에 있을 수 있다.

단계(555)에 있어서, 상기 시스템은 상기 마커된 원시 기계 명령어를 스터브로의 한 점프로 겹쳐 쓰기하며, 이때 상기 스터브(stub)란 한 코드 섹션을 나타내는 것으로 사용된다. 상기 시스템은 단계(557)에 있는 버퍼에서 원시 기계 명령어를 고정시키거나 완성시키는 스터브(stub)에 원시 기계 명령어를 위치시킨다. 단계(559)에 있어서, 상기 시스템은 고정된 원시 기계 명령어를 다시 그 원래의 위치(예를 들면, 마커에 의하여 표시된 위치)로 복사하기 위해 상기 스터브내에 원시 기계 명령어를 위치시킨다.

단계(561)에 있어서, 고정된 원시 기계 명령어로 점프시키도록, 상기 시스템은 스터브에 원시 기게 명령어를 위치시킨다. 점프 명령어의 경우에서와 같이, 다시 제어를 이동시키는데 단일 원시 기계 명령어만이 필요하다. 그러므로, 기술에서 이용되는 용어에 대하여 복수나 단수 등의 설계는, 특별한 실시 예에 의존하면서 다양해질 수 있다.

도 8 은, 도 7A 와 도 7B 등에서 나타난 흐름도에 의하여 생성될 수 있는 컴파일된 방법에 대하여, 원시 기계 명령어의 순서를 나타낸다. 원시 기계 명령어의 순서(601)가 도시되며, 상기 방법에 대한 원시 기계 명령어를 나타낸다. 원시 기게 명령어(603)에 의하여 대표되는 자바 가상 기계 명령어는 putstatic이다. putstatic 가상 기계 명령어는 값을 클래스의 정적 변수내로 한 값을 기록한다.

IBM 호환 기계를 위한 원시 기계 명령어는 다음과 같다 :

MOVL EAX, CLASS

MOVL [EAX+ OFFSET], EBX

여기서, CLASS 는 메모리에 있는 클래스 정의의 위치를 대표하고, OFFSET 는 정적 변수의 클래스 정의 내에 있는 오프셋을 나타낸다. 첫 번째 move 문장은 메모리에 있는 클래스 정의의 위치를 레지스토 EAX 로 이동시킨다. 두 번째 move

문장은 레지스터 EBX 의 내용(정적 변수에 저장될 값을 포함하는)을, 레지스터 EAX 와 OFFSET 을 더함으로써 얻어지는 위치로 이동시키는데, 이는 메모리 내 정적 변수의 위치이다. 상기 클래스가 적재되지 않는다면, CLASS 와 OFFSET 모두에 대한 값들이 런타임 실행이 있을 때까지 해결되지 않을 것이다. 따라서, 더미 값들이 프로그램 실행시간 컴파일 중에 자신들의 위치에 머무르게 된다.

삭제

원시 기계 명령어의 섹션(605)은 원시 기계 명령어(603)의 복사로써 도시된다(상기에서 도시된 바와 같이). 어떤 실시 예에 있어서, 원시 기계 명령어의 섹션은 세밀한 조정을 위한 원시 기계 명령어를 저장하는 버퍼(buffer)이고, 상기 원시 기계 명령어의 섹션은 또한 버퍼를 세밀하게 조정하는 원시 기계 명령어를 포함하는 스터브(stub)이다. 다른 실시 예에 있어서, 버퍼(buffer)라는 것은 겹쳐 쓰기 되는 원시 기계 명령어의 일부분에 대한 임시 기억장치로써 작동하며, 상기 원시 기계 명령어는 그 원래의 위치에서 조정된다.

원시 기계 명령어의 섹션(605)은, 상기 색인을 상기 런타임 스택으로의 필드를 위해 상수 풀로 밀어 넣기 위한 push 문장을 포함한다. 일반적으로, 상기 상수 풀(constant pool)은 클래스, 필드 이름, 및 타입 서명을 포함한다. 다음으로, 해결 경로(resolution routine)가 원시 기계 명령어의 섹션에서 호출될 수 있다.

상기 해결 경로는 원시 기계 명령어를 완성할 수 있으며, 상기 경로의 실시 예가 도 9에 도시된다. 단계(701)에서, 상기 경로가 필요하다면 클래스를 적재한다. 상기 클래스는 상기 해결 경로를 호출하기 전에, 스택으로 넣어진 상수 풀로의 색인에 의하여 분류된다. 경로는 단계(703)에서 필요하다면 클래스를 초기화한다. 상기 클래스는 적재되거나 초기화될 필요가 없는데, 이는 이들 단계가 자바 프로그램의 각기 다른 부분에서 런타임 실행중 이미 실행되었기 때문이다. 추가로, 상기 해결 루틴은 클래스를 적재하고 초기화하는 검사하는 것으로 도시된다. 상기 양 단계 모두는 선호되는 실시 예에서 실행되었지만, 다른 런타임 실행 검사가 본 발명과 관련하여 실행될 수 있다.

단계(705)에서, 상기 경로는 클래스 필드를 해결한다. 상기 경로는 메모리에 있는 필드의 위치, 또는 필드와 필드가 속해있는 클래스의 위치만을 해결한다. 단계(707)에서, 플레이스 홀더 데이터를 해결된 클래스 필드로 바꾸는 것에 의하여, 상기 경로는 상기 원시 기계 명령어를 고정한다. 단계(709)에서, 상기 경로는 상기 고정된 기계 명령어로 되돌아 가서 실행을 계속하도록 한다.

도 8 과 도 9 등에 의하여 지적된 것처럼, 상기 원시 기계 명령 섹션은 원시 기계 명령어를 완성하도록 다른 경로를 호출하여서, 상기 섹션이 모든 실시에서 모든 코드를 포함할 것을 필요로 하지 않도록 한다. 선호되는 실시 예에 있어서, 본 발명에 의하여 생성된 원시 기계 명령어의 섹션 각각에 의하여, 상기 해결 경로가 호출된다.

발명의 효과

상기 발명의 상세한 설명은 본 발명의 선호되는 실시 예의 완전한 기술이지만, 다른 변형, 변경, 및 이에 상응하는 것 등을 이용할 수 있다. 상기에서 기술된 실시 예에 대한 적합한 변경에 의하여, 본 발명은 동일하게 적용 가능하다는 사실은 극히 당연하다. 예를 들면, 상기에서 기술된 실시 예는 특정 런타임 실행 정보(specific runtime execution information)에 관한 것이었지만, 본 발명의 원리는 런타임 컴파일이 진행되는 동안에 이용 가능하지 않은 정보에 쉽게 적용 가능할 수 있다. 그러므로, 상기의 기술은, 본 발명의 특허청구범위에 의하여서만 한정되는 것이다.

도면의 간단한 설명

도 1 은, 본 발명의 실시 예로써 소프트웨어를 실행하는데 이용될 수 있는 컴퓨터 시스템(computer system)의 실례를 설명한다 ;

도 2 는, 도 1 에 있는 컴퓨터 시스템(computer system)의 시스템 블록 다이어그램을 나타낸다 ;

도 3 은, 자바 소스 코드 프로그램(Java source code program)을 어떻게 실행하는지를 나타낸다 ;

도 4 는, 자바 런타임 시스템(Java runtime system)을 구현하는 요소(component)를 나타낸다 ;

도 5 는, 본 발명의 실시예로써 고-레벨 흐름도를 설명한다 ;

도 6A 는, 컴파일을 마친 방법(compiled method)을 위한 원시 명령어(native instruction)의 순서를 나타내고 ;

도 6B 는, 런타임 클래스 적재 검사(runtime class loading check)와 런타임 클래스 초기화 검사(runtime class initialization check) 를 실행시킬 목적으로, 본 발명의 실시 예를 이용하는 원시 명령어(native instruction)의 순서를 나타낸다 ;

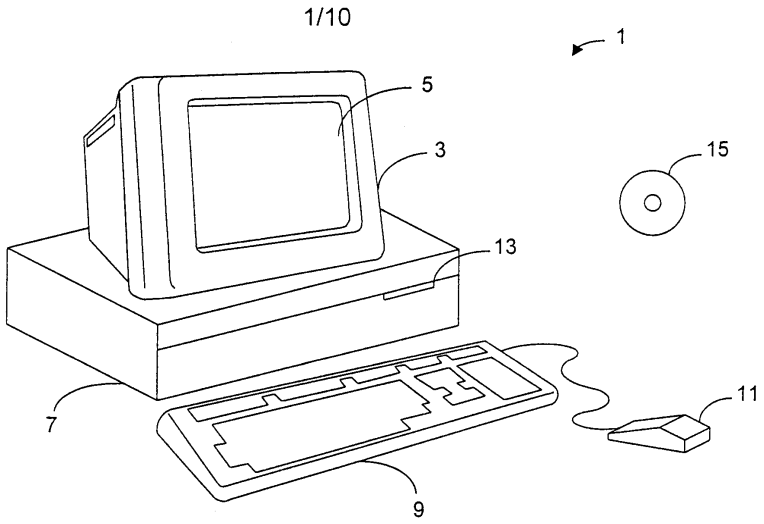
도 7A 와 도 7B 등은, 본 발명의 실시 예에 따라서 방법을 컴파일(compile)하는 흐름도를 나타낸다 ;

도 8 은, 도출 경로(resolution routine)를 이용하는 본 발명의 또 하나의 실시 예에서, 컴파일을 마친 방법(compiled method)을 위한 원시 기계 명령어(native machine instruction)의 순서를 나타낸다 ; 그리고

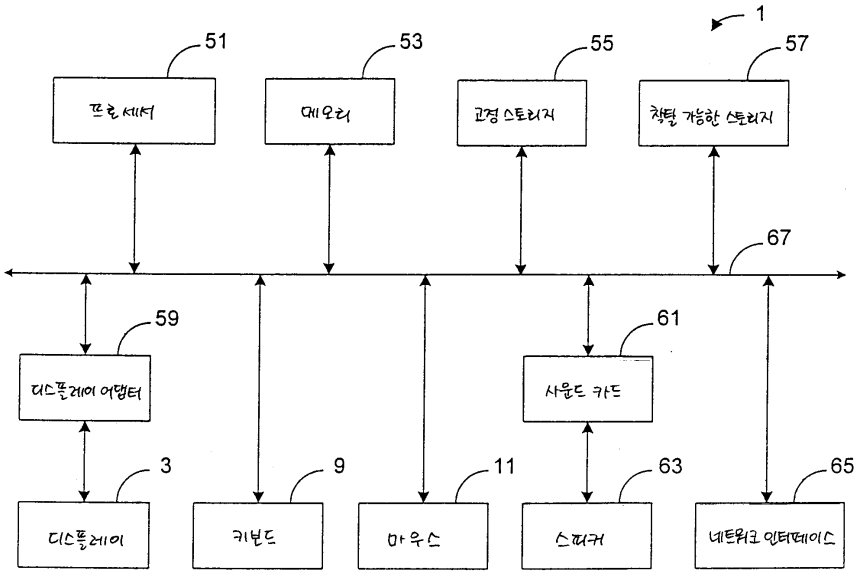
도 9 은, 도 8 에 있는 원시 기계 명령어(native machine instruction)에서 호출될 수 있는 도출 경로(resolution routine)의 흐름도를 나타낸다.

도면

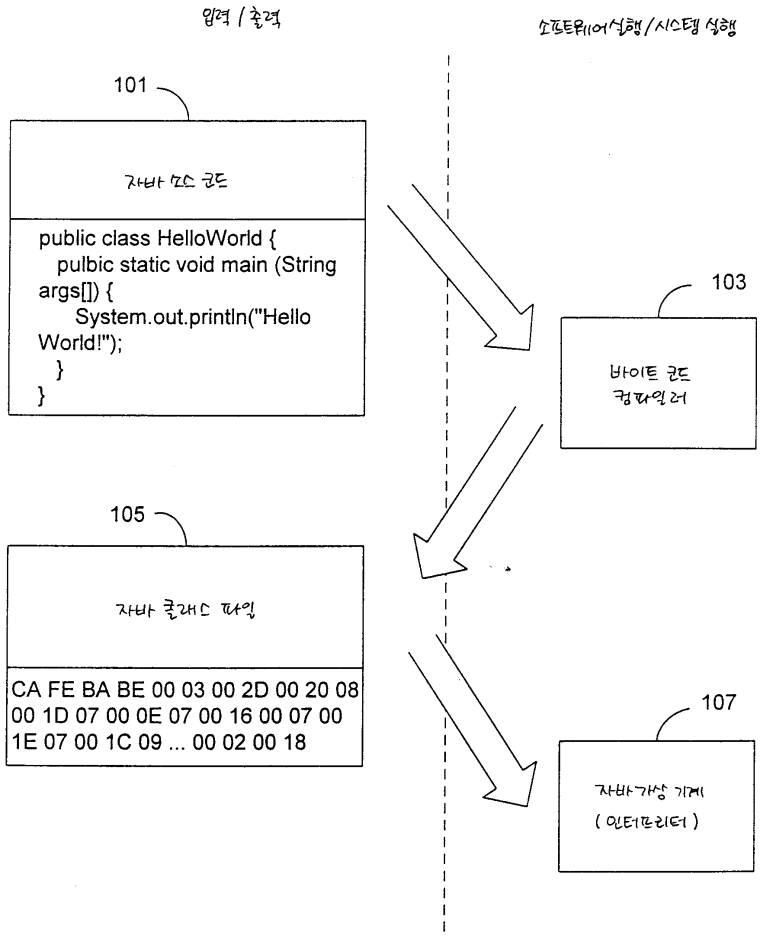
도면1



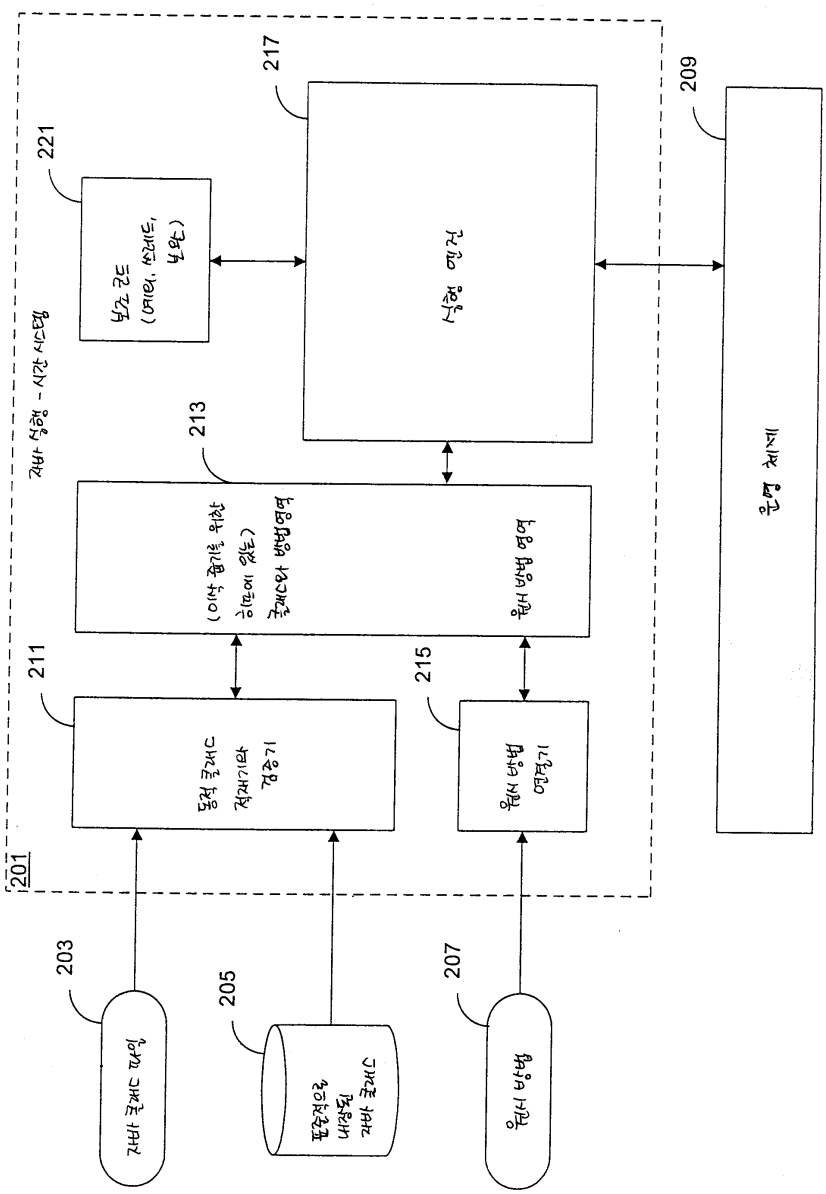
도면2


1
51
53
55
57
프로세서
메모리
고정 스토리지
착탈 가능한 스토리지
67
59
61
디스플레이 어댑터
사운드 카드
3
9
11
63
65
디스플레이
키보드
마우스
스피커
네트워크 인터페이스

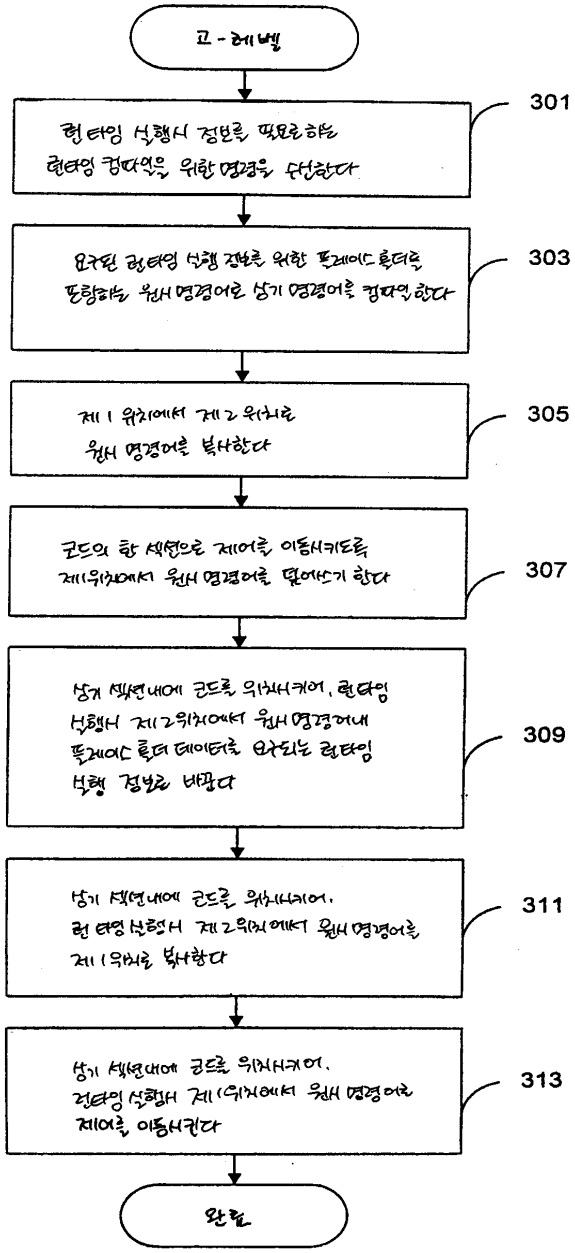
도면3


입력 / 출력
소프트웨어실행 / 시스템 실행
101
자바 소스 코드
public class HelloWorld {
pulbic static void main (String
args[]) {
System.out.println("Hello
World!");
}
}
103
바이트 코드
컴파일러
105
자바 클래스 파일
CA FE BA BE 00 03 00 2D 00 20 08
00 1D 07 00 0E 07 00 16 00 07 00
1E 07 00 1C 09 ... 00 02 00 18
107
자바가상 기계
(인터프리터)

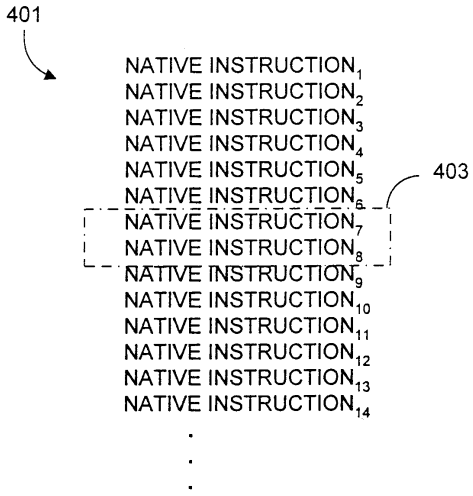
도면4



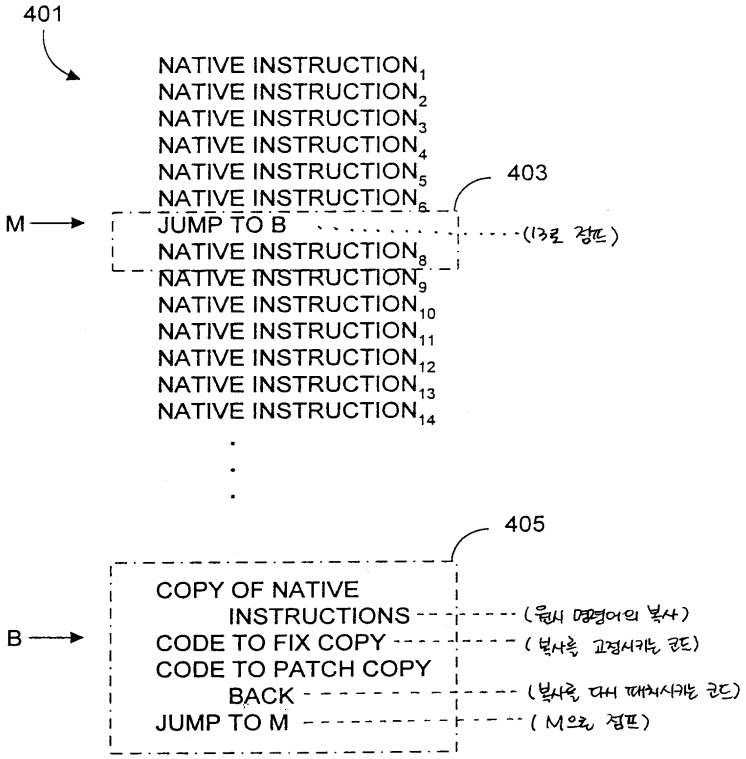
도면5


고-레벨
301
리타임 실행시 정보를 필요로하는
리타임 컴파일을 위한 명령을 선언한다
303
요구된 리타임 실행 정보를 위한 프레이드 룸터를
포함하는 원시 명령어로 상기 명령어를 컴파일한다
305
제1 위치에서 제2 위치로
원시 명령어를 복사한다
307
코드의 한 세션으로 제어를 이동시키도록
제1위치에서 원시 명령어를 덮어쓰기 한다
309
상기 세션내에 코드로 워처시키어, 리타임
실행시 제2위치에서 원시 명령어내
프레이드 룸터 데이터를 요구되는 리타임
실행 정보로 바꾼다
311
상기 세션내에 코드를 워처시키어,
리타임 실행시 제2위치에서 원시 명령어를
제1위치로 복사한다
313
상기 세션내에 코드를 워처시키어,
리타임 실행시 제1위치에서 원시 명령어로
제어를 이동시킨다
완료

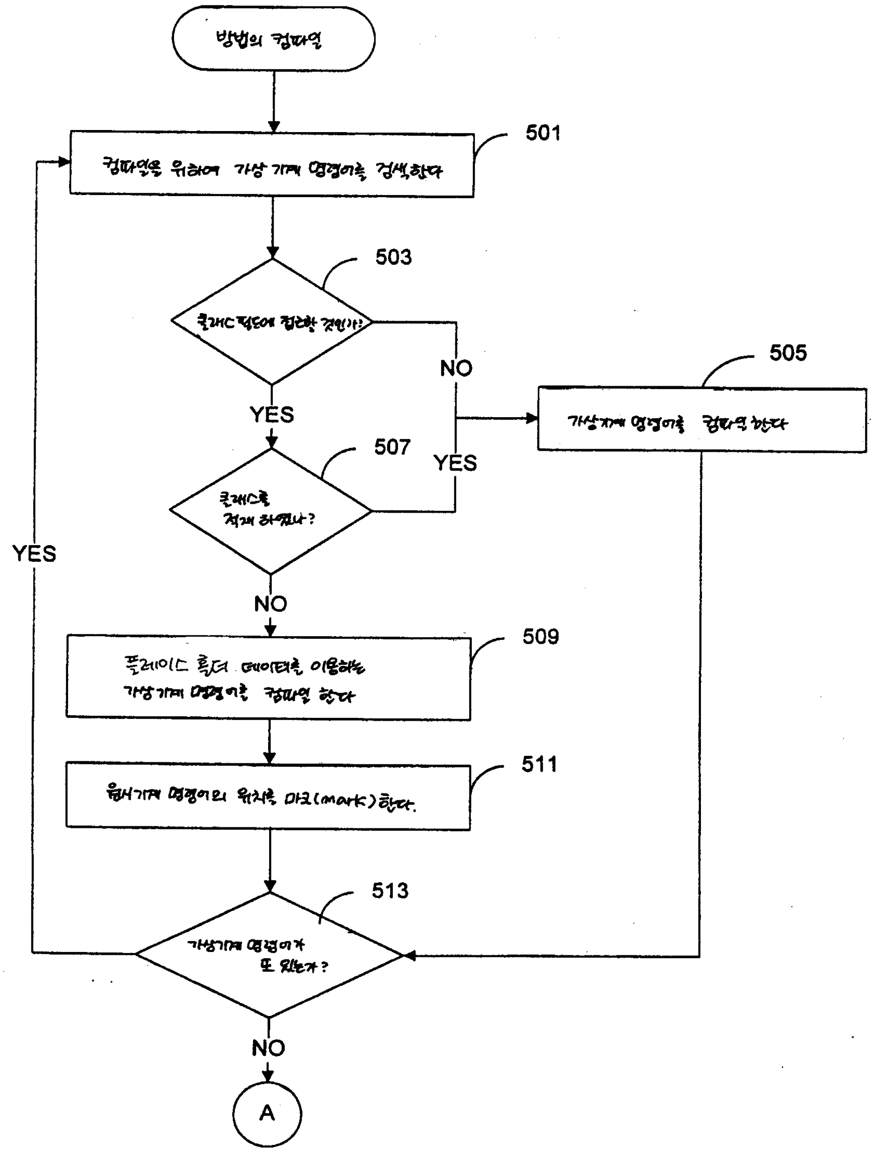
도면6a


401
NATIVE INSTRUCTION1
NATIVE INSTRUCTION2
NATIVE INSTRUCTION3
NATIVE INSTRUCTION4
NATIVE INSTRUCTION5
NATIVE INSTRUCTION6
NATIVE INSTRUCTION7
NATIVE INSTRUCTION8
NATIVE INSTRUCTION9
NATIVE INSTRUCTION10
NATIVE INSTRUCTION11
NATIVE INSTRUCTION12
NATIVE INSTRUCTION13
NATIVE INSTRUCTION14
403

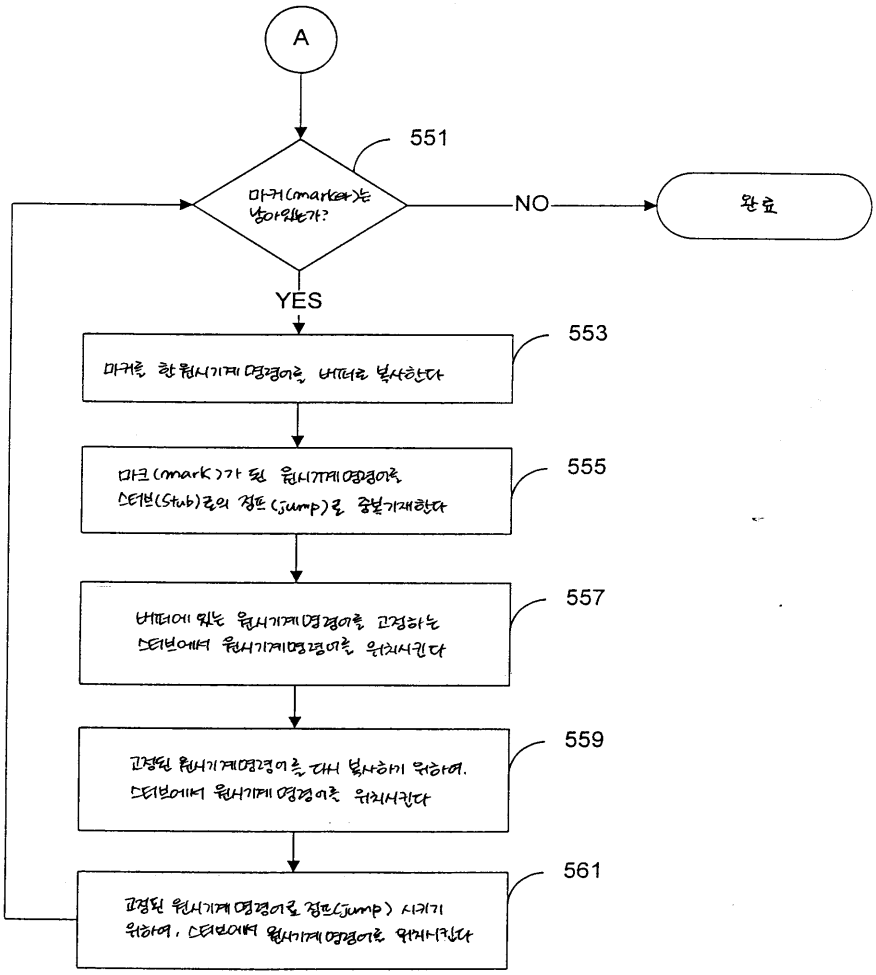
도면6b


401
NATIVE INSTRUCTION1
NATIVE INSTRUCTION2
NATIVE INSTRUCTION3
NATIVE INSTRUCTION4
NATIVE INSTRUCTION5
NATIVE INSTRUCTION6
403
M
JUMP TO B
(13로 점프)
NATIVE INSTRUCTION8
NATIVE INSTRUCTION9
NATIVE INSTRUCTION10
NATIVE INSTRUCTION11
NATIVE INSTRUCTION12
NATIVE INSTRUCTION13
NATIVE INSTRUCTION14
405
B
COPY OF NATIVE INSTRUCTIONS
(원시 명령어의 복사)
CODE TO FIX COPY
(복사를 고정시키는 코드)
CODE TO PATCH COPY BACK
(복사를 다시 패치시키는 코드)
JUMP TO M
(M으로 점프)

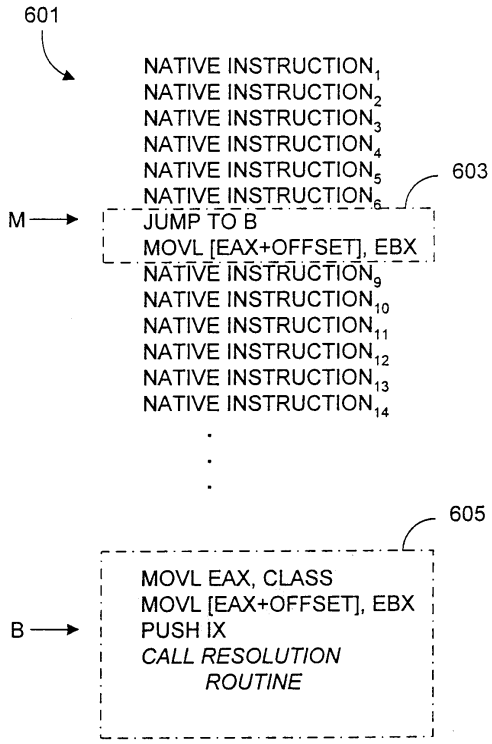
도면7a


방법의 컴파일
컴파일을 위하여 가상 기계 명령어를 검색한다
501
클래스 필드에 접근한 것인가?
503
NO
YES
가상기계 명령어를 컴파일 한다
505
클래스를 적재 하였나?
507
YES
NO
YES
플레이스 홀더 데이터를 이용하는 가상기계 명령어를 컴파일 한다
509
원시기계 명령어의 위치를 마크(mark)한다.
511
가상기계 명령어가 또 있는가?
513
NO
A

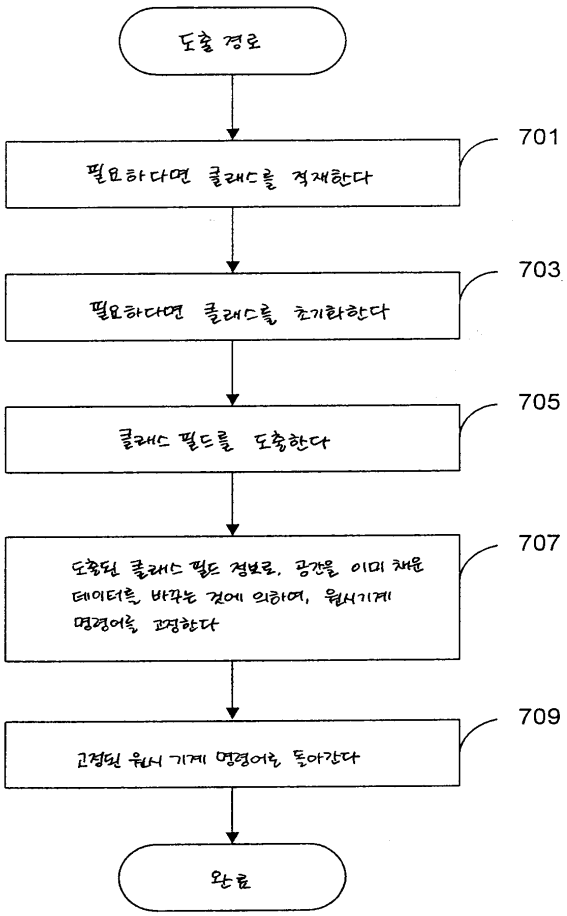
도면7b


A
551
마커(marker)는 남아있는가?
NO
완료
YES
553
마커를 만난 원시기계명령을 버퍼로 복사한다
555
마크(mark)가 된 원시기계명령을 스터브(Stub)로의 점프(jump)로 중복기재한다
557
버퍼에 있는 원시기계명령을 고정하는 스터브에서 원시기계명령을 위치시킨다
559
고정된 원시기계명령을 다시 복사하기 위하여, 스터브에서 원시기계명령을 위치시킨다
561
고정된 원시기계명령으로 점프(jump) 시키기 위하여, 스터브에서 원시기계명령을 위치시킨다

도면8


601
NATIVE INSTRUCTION1
NATIVE INSTRUCTION2
NATIVE INSTRUCTION3
NATIVE INSTRUCTION4
NATIVE INSTRUCTION5
NATIVE INSTRUCTION6
603
M
JUMP TO B
MOVL [EAX+OFFSET], EBX
NATIVE INSTRUCTION9
NATIVE INSTRUCTION10
NATIVE INSTRUCTION11
NATIVE INSTRUCTION12
NATIVE INSTRUCTION13
NATIVE INSTRUCTION14
605
B
MOVL EAX, CLASS
MOVL [EAX+OFFSET], EBX
PUSH IX
CALL RESOLUTION ROUTINE

도면9


도출 경로
701
필요하다면 클래스를 적재한다
703
필요하다면 클래스를 초기화한다
705
클래스 필드를 도출한다
707
도출된 클래스 필드 정보로, 공간을 이미 채운 데이터를 바꾸는 것에 의하여, 원시기계 명령어를 고정한다
709
고정된 원시 기계 명령어를 돌아간다
완료