(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(51) Int. Cl.[6]
G06F 9/46

(11) 공개번호 특1999-0077379
(43) 공개일자 1999년10월25일

(21) 출원번호 10-1998-0705554
(22) 출원일자 1998년07월16일
번역문제출일자 1998년07월16일
(86) 국제출원번호 PCT/GB1997/00107 (87) 국제공개번호 WO 1997/26596
(86) 국제출원출원일자 1997년01월15일 (87) 국제공개일자 1997년07월24일
(81) 지정국 EP 유럽특허 : 오스트리아 벨기에 스위스 독일 덴마크 스페인 핀랜드 프랑스 영국 그리스 이탈리아 룩셈부르크 모나코 네덜란드 포르투칼 스웨덴

국내특허 : 아일랜드 오스트레일리아 캐나다 중국 일본 대한민국 멕시코 노르웨이 뉴질랜드

(30) 우선권주장 9600823.0 1996년01월16일 영국(GB)
(71) 출원인 브리티쉬 텔리커뮤니케이션즈 파블릭 리미티드 캄퍼니 내쉬 로저 윌리엄

영국 런던(우편번호 이시1에이 7에이제이) 뉴게이트 스트리트 81
(72) 발명자 마틴 폴 앤터니

영국 서포크(우편번호 ip13 6엔와이) 지티. 빌링스 코우치만스카티지
(74) 대리인 김명신, 김원오

심사청구 : 없음

(54) 통신용 분산 시스템 및 그곳에서 사용되는 컴파일러 장치

요약

본 발명은 분산 처리에 관한 것으로, 느슨하게-결합된 다중 프로세서 연산용 소스 코드를 적합하게 하는 새로운 명령문을 부가하기 위해 상기 원시 소스 코드를 수정하는 종래의 객체지향 컴파일러 프로그램(312)으로 연산을하는 전치-컴파일러 프로그램(316)을 제공하는 것을 특징으로 한다.

대표도

도19

명세서

본 발명은 분산 처리방법에 관한 것으로, 특히 전기통신 네트워크의 제어를 위한 분산 처리방법으로 제한되는 것은 아니다. 보다 특별하게는, 본 발명은 분산 프로세서상에서 구현되는 제어 시스템을 개선 및 업데이트 시키는 것에 관한 것으로서, 객체지향 양식으로 구현되는 처리(필요한 것은 아님)에 관한 것이다.

전기통신 네트워크는 높은 대역폭, 낮은 지연 정보 흐름을 지원하는 것이 크게 요구된다. 요구되는 대역폭은 초당 킬로바이트로부터 초당 메가바이트 까지 빠르게 진행되고 있으며, 몇 몇 적용에서는 초당 기가바이트(특히 예를들면, 주문형 비디오; 애니메이트 공유 시뮬레이션, 및 분산 컴퓨팅 등)가 요구된다.

특정 번호로 통화를 재지정하는 것과 같은 '지능망' 설비를 제공하기 위해서, 컴퓨터 프로그램은 교환국과 연결된 다수의 호스트 컴퓨터(예를들어, 100개 이상)상에서 구동된다. 특정 고객(예를들어, 경로설정될 고객을 호출하는 특정 번호)을 위해 제공될 서비스 방법은 상기 호스트 컴퓨터상에서 그 고객에 관련되어 저장된 데이터에 따른다. 따라서, 수십 또는 수백개의 호스트 컴퓨터상의 수백만 가입자가 있을 수 있다.

1995년 4월 BT 테크놀러지 저널 Vol 13 No.2에 P.A.마틴(P.A.Martin) 의 'Twenty-twenty vision - software architectures for intelligence in the 21st centry'에서, 본 발명자는 요구된 분산 처리를 구현하기 위한 객체지향 기술의 사용을 제안하고 있다.

객체지향 기술의 설명은, 예를들어 1993년 7월 E.L.쿠삭(E.L.Cusak)과 E.S.코딩리(E.S.Cordingley)에 의해 편집된 BT 테크놀러지 저널 Vol 11 No.3 의 'Object oriented technology' 등에서 볼 수 있다. 비록 그 항목이 항상 정확히 사용되는 것으 아니지만, 본 명세서에서는 객체지향 컴퓨팅은 호출 프로그램 또는 경로에 의해 직접 액세스 가능하다기 보다는 '엔캡슐레이티드(encapsulated)' 형태로 저장된 데이터로서, 이 데이터는 이를 판독, 기록 및 편집할 수 있는 제한된 프로그램 부분에 의해서 액세스 가능할 뿐인 컴퓨팅 기술을 가리킨다. 데이터의 레코드 및 그과 관련된 컴퓨터 코드는 '객체(object)'로서 언급된다. 객체로의 통신 및 객체로부터의 통신은 일반적으로 '메시지 통과'에 의해: 즉, 객체 통과 데이터값을 호출하고 그 객체 내부에 구성된 프로그램중 하나의 연산을 구한다.

상기 객체지향 어프로우치를 사용하길 바라는 프로그래머들은 여러 언어를 사용할 수 있다. 물론, 현재 가장 대중적인 것은 C++ 이다.

분산 처리는 여러가지 측면에서 단일 프로세서 연산과는 다르다. 먼저, 다른 프로그램 또는 데이터가 호출 프로그램 또는 다른 호스트 컴퓨터로서 동일한 호스트 컴퓨터상에 위치하는지 여부에 따라 차등 액세스 기술이 요구될 수 있다. 프로그램 또는 데이터의 위치는 다른 프로그램으로부터 도달될 수 있는 속도에 영향을 미치게 된다. 또한, 하나 또는 그 이상의 호스트 컴퓨터는 연산중에 따로 남겨질 수 도 있다.

분산 컴퓨팅은 '클라이언트-서버' 배치를 사용하여 종래방법으로 수행되며, 한 컴퓨터 상의 '클라이언트' 프로그램은 다른 컴퓨터 상의 '서버' 프로그램에 질문을 하여 그 기능을 수행하거나 또는 그 클라이언트 프로그램에서 요구하는 데이터를 돌려보낸다.

객체지향 기술은 분산 처리에 광범위하게 적용되어 오지 못했다. 이러한 점에서 당 기술의 상태를 요약을 다음 자료에서 볼 수 있다: 'Object oriented programming systems' Blair G., 피트만 출판사, 런던 1991년 (ISBN 0-273-03132-5) 특히, 9장 223-243페이지; 'Distributed systems and objects' David Hutchson 및 Jonathan Walpole. 앞서의 시도들은 보통 현재의 컴퓨터 언어에 새로운 구문(syntax)을 부가하거나, 또는 새로운 컴퓨터 언어를 생성하여 분산 처리를 다루는 종래의 객체지향 프로그래밍을 확장시키고 있다.

병렬 처리를 위한 도전이라는 1995년 ICPP 워크숍에서의 회보에서, H.Sivaraman 등의 논문 'Parallelizing Sequential Programs to a Cluster of Workstations' 의 38에서 41페이지에는 강하게 결합된 분산 제어 시스템, 달리 표현하면, 공유된 메모리 기술을 사용하는 제어 시스템을 사용하는 다수의 컴퓨터상에서 병렬 실행을 위한 연속 프로그램의 변환이 개시되어 있다.

한 측면에서, 본 발명은 다수의 객체(즉, 자신의 코드에 의해 판독가능하고 기록가능한 데이터 레코드)가 분산 컴퓨팅 시스템의 여러 호스트 컴퓨터에 걸쳐 분산되어 있는 분산 컴퓨팅 시스템을 제공하는데, 각 호스트 컴퓨터에는 상기 시스템의 모든 데이터 레코드가 위치해 있는 컴퓨터를 지정하는 위치 테이블(리스트)이 있다.

따라서, 중앙 제어점은 필요 없으며, 그렇지 않으면 처리에 병목현상이 나타난다. 더욱이, 상기 위치 테이블은 모든 호스트 컴퓨터상에서 동일할 수 있기 때문에, 새로운 호스트 컴퓨터는 현재의 호스트 컴퓨터로부터 그 위치 테이블을 간단히 복사함으로서 상대적으로 쉽게 부가할 수 있다.

적절하게는, 상기 호스트 컴퓨터들은 상기 데이터 레코드가 제1 호스트 컴퓨터에서 제2 호스트 컴퓨터로 이동하게 되도록 배치된다.

보다 적절하게는, 상기 호스트 컴퓨터들은, 이동에 있어서, 레코드가 재위치하게되는 호스트 컴퓨터의 위치가 각 각의 다른 호스트 컴퓨터로 시그널되도록 배치된다.

상기 호스트 컴퓨터들은 각 각의 위치 테이블이 만들어진 데이터 레코드가 있는 호스트 컴퓨터의 위치도 저장하도록 배열되기도 한다.

적절하게는, 데이터 레코드를 액세스하기 위해, 각 호스트 컴퓨터가 그것을 저장하는 상기 위치 테이블을 판독하고, 그리고 필요한 경우에는 데이터 레코드가 만들어진 호스트 컴퓨터의 위치 테이블을 액세스한다.

적절하게는, 상기 데이터 레코드는 관련된 코드를 통해서만 액세스 가능한 객체로 구성되고, 더욱 적절하게는, 그러한 객체는 동일한 클래스(class)의 다수의 객체로 구성되는데, 이 다수의 객체는 상기 관련된 코드를 공유한다.

두번째 측면으로서, 본 발명은 상호연결된 다수의 컴퓨터로 구성된 분산 제어 시스템을 사용하는 컴파일러(compiler)(또는 전치-컴파일러)를 제공하는데, 상기 컴파일러 장치는 하나의 컴퓨터상에서의 컴파일에 적절한 소스 언어내의 원시 소스 프로그램을 수용하도록 배치되고, 상기 분산 제어 시스템은 느슨한-결합을 하는 것을 특징으로 하며; 상기 컴파일러 장치는 상기 원시 소스 코드 프로그램을 수용하고, 상기 원시 소스 프로그램의 내부에서, 상기 원시 프로그램 내부의 다른 지점으로부터 구해지기도 하는 함수들을 위치시키고, 그리고 상기 함수들을 다른 컴퓨터로부터 원격으로 구할 수 있도록 하는 상기 원시 소스 프로그램을 수용하기 위해 배치되는 것을 특징으로 한다.

세번째 측면으로, 본 발명은 상호 연결된 다수의 컴퓨터로 구성된 분산 제어 시스템을 사용하는 컴파일러(또는 전치-컴파일러)를 제공하는데, 상기 컴파일러 장치는 하나의 컴퓨터상에의 컴파일에 적절한 소스 언어내의 원시 소스 프로그램을 수용하기 위해 배치되고, 상기 분산 제어 시스템은 느슨하게-결합된 것을 특징으로 하고; 상기 컴파일러 장치는 하나의 컴파일러로 구성되고 상기 원시 소스 프로그램내부에, 하나의 컴퓨터 메모리 공간내부의 영역으로 포인터로서 상기 원시 컴파일럴에 의해 컴파일될 기준을 위치시키기 위해 배열되고, 다른 상기 컴퓨터의 메모리 공간을 가리킬 수 있도록 하기 위해 상기 원시 소스 프로그램을 수정하도록 배열되는 것을 특징으로 한다.

네번째 측면으로, 본 발명은 상호 연결된 다수의 컴퓨터로 구성된 분산 제어 시스템을 사용하는 컴파일러(또는 전치-컴파일러)를 제공하는데, 상기 컴파일러 장치는 하나의 컴퓨터상에서 컴파일에 적절한 소스 언어내의 원시 소스 프로그램을 수용하기 위해 배치되고, 상기 분산 제어 시스템은 느슨하게-결합되어 있는 것을 특징으로 하며; 상기 컴파일러 장치는 상기 원시 소스 프로그램을 판독하기 위해 그리고 그것으로서 상기 원시 소스 프로그램 내부에 구성된 데이터 및 프로그램을 설명하는 데이터 레코드의 세트를 부가하도록 배치되는 것을 특징으로 한다.

본 발명의 내용중 느슨하게-결합되었다는 말은 제어 시스템이 공유된 메모리 기술에 반대로 메시징 기술을 사용하는 것을 나타낸다.

본 발명의 모든 측면은 다수의 컴퓨터상의 연속 실행을 위한 연속적 프로그램의 변환에 관계되는데, 앞에서의 병렬실행에 크게 관련된 Sivaraman 등의 논문과는 다르다.

편리하게는, 본 발명의 측면은 전치-컴파일러를 구비하고 있어서 보다 오래된 종래의 컴파일러를 가지고 사용될 수 있으나, 본 발명의 함수은 재-기록 컴파일러로 바로 일체화되는 것으로 인식된다.

따라서, 이러한 측면들에서, 본 발명은 프로그래머들에게는 실행가능한 프로그램을 만드는데 요구되는 시간을 감소시키면서 많은 다양한 프로세서상에서의 연산을 위해 그 프로그램을 적용하는 복잡성은 숨겨져 있다.

본 발명은 앞서 언급한 첫번째, 두번째, 세번째 및 네번째 측면의 어느 둘 또는 그 이상의 특성의 조합으로 구성된 컴파일러 장치를 제공하기도 한다.

다섯번째 측면으로서, 본 발명은 상호 연결된 다수의 컴퓨터로 구성된 분산 제어 시스템을 구비하고 상기 컴퓨터용 프로그램의 컴파일용 장치를 컴파일하는 전기통신 시스템을 제공하는데, 상기 장치는 하나의 호스트 컴퓨터상의 실행을 위한 원시 프로그램을 컴파일하도록 배치되어 있는 컴파일러, 그리고 단일 컴퓨터 실행에 적합한 원시 소스 프로그램을 수신하고 추가의 소스 코드를 부가하기위해 상기 원시 소스 프로그램을 수정하여 상기 컴파일러에 의해 컴파일 된 경우에는 상기 분산 제어 시스템내에서 실행될 수정된 소스 프로그램을 만들기위해 배치된 프리-컴파일러를 구비하고 있다.

본 발명의 이 측면으로 만들어진 하나의 실행가능한 프로그램은 동일한 복사물내의 분산 컴퓨팅 시스템의 모든 프로세서(호스트 컴퓨터)로 분산되기도 한다. 이것은 상기 분산 컴퓨팅 시스템에 새로운 호스트 컴퓨터를 더하는 것을 쉽게 만든다.

이러한 실시예에서, 상기 소스 프로그램은 객체지향 언어로 되어있고, 더욱 적절하게는 C++ 또는 그것의 변형으로 된다. C++ 은 분산 환경에서의 컴파일에 있어서 추가의 문제점을 나타내는데, 멀티 프로세서 환경에서는 동작할 수 없는 공유된 메모리를 통해 단일 프로세서 컴파일 및 여러 프로세스 통신을 하는 경향때문이다.

본 발명의 다른 태양들과 실시예는 이후 기술되고 청구되는 바와 같다.

이후 첨부한 도면을 참조하여 적당한 예를 통해서 본 발명의 실시예를 기술한다.

도 1은 본 발명을 구현하는 통신시스템의 구성요소를 나타내는 블럭도,

도 2는 도 1의 시스템의 다른 실시예를 나타내는 블럭도,

도 3은 도 1 및 도 2의 시스템의 일부를 형성하는 호스트 컴퓨터의 구성요소를 나타내는 블럭도,

도 4는 도 1 및 도 2의 시스템들의 일부를 형성하는 컴파일러 장치의 구성요소를 나타내는 블럭도,

도 5는 도 4의 컴파일러 장치의 동작을 나타내는 흐름도,

도 6은 도 5에서 실행된 처리의 단계의 결과를 나타내는 설명도,

도 7a는 도 1의 일부를 형성하는 인텔리전스 도메인내 저장된 데이터의 구조를 나타내는 도면,

도 7b는 도 1의 일부를 형성하는 트랜스포트 도메인내 저장된 데이터의 구조를 나타내는 도면,

도 8은 도 7a의 한 구성요소의 메모리내의 데이터 구조를 나타내는 도면,

도 9는 확장 소스 코드를 발생하기 위해 도 4의 소스 코드 기억장치에 저장된 원시 소스 코드를 확장하는데 있어서 도 4의 장치의 전치-컴파일러의 동작을 나타내는 도면,

도 10은 전치-컴파일러의 동작의 일반적인 시퀀스를 나타내는 흐름도,

도 11(도 11a 및 도 11b 포함)은 형태 모델의 구성을 위한 도 10의 처리의 일부를 보다 상세히 보여주는 흐름도,

도 12는 그 처리에 의해 발생된 부가적인 소스 코드를 나타내는 도면,

도 13은 생성된 형태 모델의 구조를 개략적으로 나타내는 도면,

도 14는 객체의 원격 호출을 위해 제공하는데 있어서 도 10의 처리의 다른 일부의 스텝을 나타내는 흐름도,

도 15는 생성된 대응하는 부가적인 소스 코드를 나타내는 도면,

도 16은 로컬 포인터를 대체하기 위해 도 10의 처리의 다른 단계의 스텝들을 나타내는 흐름도,

도 17은 생성된 대응하는 부가적인 소스 코드를 나타내는 도면,

도 18a는 로컬 포인터를 통한 함수의 호출을 나타내는 설명도,

도 18b는 도 16의 처리의 실행에 이어지는 레퍼런스에 의한 함수의 호출을 나타내는 대응 설명도,

도 19는 도 3의 호스트 컴퓨터의 객체 관리자 프로그램의 일부로서 메모리내에 저장된 객체 위치 테이블의 구조를 나타내는 도면,

도 20은 호스트 컴퓨터의 객체 관리자의 일부로서 도 3의 호스트 컴퓨터의 메모리내에 저장된 호스트 상태 테이블의 구조를 나타내는 도면,

도 21(도 21a 및 도 21b 포함)은 호스트 상태 테이블의 업데이트를 나타내는 흐름도,

도 22(도 22a 및 도 22b 포함)는 새로운 객체를 구성하는 처리를 나타내는 흐름도,

도 23은 한 호스트 컴퓨터로부터 다른 호스트 컴퓨터로 객체를 이동하는 처리를 나타내는 흐름도 및

도 24(도 24a, 도 24b 및 도 24c 포함)는 원격 호스트 컴퓨터의 객체에 메시지를 전송하는 처리를 보여주는 흐름도이다.

도 1에 있어서, 본 발명에 따라 생성된 통신시스템은 트랜스포트 도메인(6)을 구성하는 통신채널(4)(예를 들어 마이크로파 채널, 광파이버 케이블, 동축 구리케이블 또는 상기한 것 중 어느 하나에 수반된 가상회로)에 의해 서로 연결된 복수의 스위칭 센터 또는 교환기(2a, 2b)를 포함한다. 트랜스포트 도메인(6)에는 호스트 컴퓨터(10a, 10b)를 서로 연결하는 시그널링 링크(12a, 12b, 12c)를 통한 스위칭 센터(2a, 2b)와의 시그널링 통신에 있어서 복수의 호스트 컴퓨터(10a, 10b)를 포함하는 인텔리전스 도메인(8)이 연결된다. 예를 들어, 2개의 도메인은 시그널링 시스템(7)(SS7)과 같은 프로토콜을 사용하여 서로 연결될 수 있다.

호스트 컴퓨터(10a, 10b, 10c)가 공유 메모리 기술을 사용하여 통신하지는 못하고, 다만 메시지 전송 기술을 사용하여 통신할 수 있다는 점에서 이들은 함께 느슨하게-결합된 분산 제어 시스템을 구성한다.

비디오 플레이어(16)와 같은 광대역 통신장치와 전화(14a, 14b)와 같은 최종 사용자 단말 장치(또한 단말 설비라고도 함)는 공동으로 로컬 루프 접속(20a, 20b, 20c)(예를 들어 광파이버, 셀룰러 무선 또는 꼬인 한 쌍의 구리 케이블 링크)를 통해 트랜스포트 도메인(6)에 연결된 최종 사용자 도메인(18)을 포함한다.

또한, 비디오 플레이어(24)와 같은 서비스(예를 들어 비디오 서비스)를 제공하는 설비와 ISDN 채널 또는 다른 고대역 링크와 같은 로컬 루프 접속(28a, 28b, 28c)을 통해 트랜스포트 도메인(6)과 연결된 컴퓨터 단말(26)로 구성되는 서비스 제공자 도메인(22)이 제공된다.

주지된 방식에 있어서, 최종 사용자 단말 장치(14 또는 16)는 트랜스포트 도메인(6)을 통해 서비스 제공자 도메인(22)으로 요청을 전송하는데 사용된다. 그 결과, 채널은 트랜스포트 도메인(6)을 통해 셋업되고, 서비스 제공자 도메인(22)은 채널을 통해 최종 사용자 도메인(18)으로 서비스를 전송한다(예를 들어 실시간 비디오 필름 또는 전자적인 형태의 데이터 파일을 전송함으로써).

종래의 간단한 구형 전화 서비스(POTS)에 있어서, 트랜스포트 도메인(6)은 단순히 트랜스포트 경로를 셋업하기 위해 최종 사용자 도메인에서 발생된 호출된 번호에 의해 제어된다. 그러나, 현재 트랜스포트 도메인의 '지능망' 제어는 인텔리전스 도메인(8)에 의해 제공된다. 인텔리전스 도메인(8)은 트랜스포트 도메인(6)으로부터 호출된 번호 및/또는 호출되는 번호를 수신하고, 호출된 또는 호출되는 번호 중 어느 하나 또는 이들 모두에 따라 몇가지 호출 처리를 실행한다. 인텔리전스 도메인은 통상 번호 변환 서비스를 제공하고, 여기서 호출된 전화번호는 서비스를 발송하는 호출을 제공하기 위해 다른 번호로 변환된다. 이 경우에 있어서, 호출된 번호는 호스트 컴퓨터(10) 중 하나에 저장된 기록에 대응하고, 이것은 대응하는 새 번호를 발생하기 위해 링크(12) 중 하나의 신호에 응답하여 액세스된다.

일반적으로, 트랜스포트 도메인(6)에서의 이벤트의 발생(최종 사용자 도메인(18)로부터의 호출의 개시와 같은)에 응답하여 인텔리전스 도메인은 제어 정보를 제공하여 트랜스포트 도메인(6)을 제어한다.

다른 데이터가 또한 인텔리전스 도메인내에 저장된다. 이 실시예에 있어서, 각각의 고객에 대한 주기적인 요금청구를 할 수 있도록 각각의 호출에 대한 요금청구 데이터가 인텔리전스 도메인에 저장된다.

도 2에 있어서, 인텔리전스 도메인(8)은 또한 네트워크 서버(11a-11c)를 통해 호스트 컴퓨터(10)에 연결된, 프로그램화된 워크스테이션으로 구성되는 컴파일러 장치와, 컴파일러 장치(30), 호스트(10) 및 서버(11) 사이에서 동작하는 광역네트워크(WAN)를 포함한다.

서버는 또한 인터넷(32)에 포함된 하나 이상의 월드와이드웹(WWW) 서버 컴퓨터에 연결되며, 또한 (예를 들어, 로컬 패킷 스위칭 노드를 통해) 인터넷에 연결된 편집 단말(15a-15d)에 연결된다.

도 3에 있어서, 각각의 호스트 컴퓨터(10)는 WAN을 통해 서버(11)에 연결된 통신 하드웨어(100), 프로세서(102) 및 랜덤 액세스 메모리 형태의 고속 액세스 기억장치와 자기 또는 광디스크 드라이브 형태의 오프라인 기억장치를 포함하는 기억장치(104)를 포함하는 메인프레임 또는 서버를 포함한다.

기억장치(104)내에는 운영체계(106)(예를 들어 UNIX(TM)), 객체 관리자 프로그램(108) 및 클래스 코드(110)와 객체 데이터를 포함하는 객체 모델이 저장되어 있고, 이들 모두는 아래에서 보다 상세히 논의될 것이다.

각각의 편집 단말(15)은 개인용 컴퓨터를 포함하고, 모뎀을 통해 사용자 설비측의 대응하는 전화(14)를 갖는 공통 전화 소켓에 연결된다.

따라서 각각의 편집 단말(15)은 프로세서, 스크린 출력장치, 입력장치(예를 들어 키보드 또는 마우스와 같은 커서 제어장치) 및 그래픽 사용자 환경(예를 들어 WINDOWS(TM)), 통신 프로그램 및 객체 브라우저 프로그램)을 포함하는 기억장치(RAM, ROM 및 하드디스크)를 포함한다.

도 4에 있어서, 컴파일러 장치는 WAN 서버(11)에 연결된 통신 인터페이스 회로기판(300), 프로세서(302) 및 고속 액세스 메모리(RAM)과 고용량 기억장치(예를 들어 하드디스크)를 포함하고, 운영체계(예를 들어 UNIX(TM)), 고유한 C++ 컴파일러 프로그램(312)(예를 들어 선 마이크로시스템의 SunPRO와 같은), 아래 보다 상세히 기술되는 전치-컴파일러(316) 및 새로운 프로그램에 포함되는 표준 함수와 정의(서브프로그램 또는 서브루틴을 나타내는)을 저장하는 라이브러리(314)를 저장하는 기억장치(304)(개별적으로 표시되지 않음)를 포함한다.

C++ 컴파일러는 종래와 마찬가지로 재배치 가능한 2진 코드로 컴파일하는 컴파일러와, 라이브러리(314)에 저장된 2진 코드 루틴과 2진 코드를 연결하고, 실행을 위해 메모리 어드레스 공간에 연결된 코드를 배치시키는 링커 프로그램(312b)를 포함한다. 이 컴파일러는 또한 일반적으로 부가적인 코드를 읽어들이는 'include'문장과 같은 컴파일러 지령을 해석하고, 컴파일 동안 다른 동작을 실행하는 전치 프로세서를 포함한다.

또한 C++ 프로그램을 정의하는 입력 소스 코드(예를 들어 입력장치(320)를 통해 입력되거나 통신회로

(300)를 통해 다운로드되거나 기억장치(304)내에 포함된 디스크 드라이브를 통해 로드된)를 저장하는 기억장치(308)과, C++ 컴퓨터(312)에 의해 발생된 실행 가능한 코드(즉, 컴파일러 프로그램의 제어하에서 동작하는 프로세서(302)에 의해)를 저장하는 기억장치(310)을 포함한다. 또한 분산형 프로세서(10)의 수에 관한 시스템 데이터, 각 프로세서(10)가 이용가능한 기억장치의 용량 및 프로세서(10)의 동작 속도 등을 저장하는 기억장치(318)를 포함한다.

프로세서(302)는 실행가능한 코드 기억장치(310)내에 실행가능한 코드를 생성하기 위해, 소스 기억장치(308)내 소스 코드상의 C++ 컴파일러(312), 또는 소스 기억장치(308)내 소스 코드상의 C++ 컴파일러(312)에 앞서는 전치-컴파일러(316)를 선택적으로 실행하도록 배치된다.

전자의 경우, 생성된 코드는 어떠한 적절한 하나의 프로세서상에서 실행될 것이다. 상기 실시예에서 프로세서(302)는 C++ 컴파일러에 의해 직접적으로 이러한 방법으로 생성된 코드를 실행할 수 있도록 배치되어, 프로그램이 일반적으로 원하는대로 작동하는지 여부를 사용자가 즉시 테스트할 수 있도록 한다.

후자의 경우, 전치-컴파일러(316)는 먼저 소스 기억장치(308)내 소스 코드를 처리하고(코드가 실행될 호스트(10)로 이루어지는 분산 시스템과 관련되는 시스템 데이터를 고려), 컴파일러(312)에 의해 실행가능한 코드 기억장치(310)내에 생성된 실행가능한 코드로 컴파일되는 소스 기억장치(308)에 수정된 소스 코드를 생성한다. 그러나, 이 실행가능한 코드는 다중 분산 호스트(10)상의 실행을 위한 것이기 때문에 반드시 컴파일러 장치(30)에서 실행가능할 필요는 없다.

도 5를 참조하면, 감시 프로그램(307)의 제어하의 컴파일러(30)의 일반적인 동작은 다음과 같다.

단계(202)에서, 소스 코드는 (예를 들어 입력 장치(320)를 통해) 소스 코드 기억장치(308)로 입력된다. 단계(204)에서, 사람인 운영자는 소스 기억장치(308)내 소스 코드를 편집하려고 할 수 있고, 거기서 편집된 텍스트는 (종래의 텍스트 프로세서를 이용하여) 소스 기억장치(308)로 입력된다.

일단 그러한 편집이 완료되면, 단계(206)에서, 사용자는 소스 코드를 국부적으로 테스트할 것인지 여부를 결정할 수 있다. 테스트하는 경우, 단계(208)에서 프로세서는 실행가능한 코드 기억장치(310)내에 실행 가능한 코드를 생산하기 위해 소스 코드 기억장치(308)내 소스 코드상에서 C++ 컴파일러(312)를 실행하고, 단계(210)에서, 프로세서(302)는 실행가능한 코드를 실행한다. 사용자가 일어나고 있는 일을 볼 수 있도록 하기 위해 출력 스크린(322)에 디스플레이하는 것과 같은 대리 연산 및 무의미한 연산을 차단하기 위해 시뮬레이션 프로그램이 제공될 수 있다.

실행중 에러가 발생하는 상황에서, 단계(212)에서 사용자는 소스 코드 기억장치(308)내 소스 코드를 편집하기 위해 단계(204)로 돌아가려고 할 수 있다. 만일 소스 코드가 만족스러운 것으로 나타난다면, 단계(214)에서 전치-컴파일러(316)는 수정된 코드를 생성하기 위해 소스 코드 기억장치(308)내 소스 코드에 적용되고, 그러면 수정된 코드는 단계(216)에서 실행가능한 코드 기억장치(310)에 실행가능한 코드를 생성하기 위해 C++ 컴파일러에 의해 컴파일된다. 그리고 이것은 단계(218)에서 WAN 서버(11)를 통해 분산 호스트 컴퓨터(10)로 전송된다. 이것이 도 6에 그래프로 표시되어 있다.

지금부터 인텔리전스 도메인에서 채택된 데이터 모형을 간단하게 설명할 것이다. 다음에서, (1대 다수의 관계에서, 객체 클래스와 관련된 동일한 코드는 실제로 그 클래스의 모든 객체를 액세스한다는 관점에서) 각각의 '객체'는 다수의 데이터 필드로 이루어지는 데이터 레코드이고, 그 객체에 대응하는 코드에 의해서만 액세스된다.

종래의 경우와 같이, 객체는 클래스로 분류되고, 동일한 클래스의 객체는 동일한 포맷의 다른 데이터를 포함한다. 또한 각각의 객체는 데이터로 동작하고 일반적으로 그렇게 동작하는 수단만으로 구성되는 (일반적으로 '방법' 또는 '함수'이라고 불리는) 하나 또는 그 이상의 서브루틴과 관련된다.

동일한 클래스의 다른 객체와 관련된 서브루틴이 대응하는 데이터를 수신하고 반환할 포맷은 동일하다(즉, 동일한 클래스의 모든 객체는 공통 인터페이스를 갖는다). 사실, C++에서 코드 및 객체가 1대 다수의 관계에 있도록 하기 위해, 서브루틴은 동일한 클래스의 모든 객체에서 한 번만 표시된다(즉, 서브루틴을 위한 코드는 한 번만 저장된다). 따라서 코드는 각각의 객체보다 객체 클래스와 관련된다.

각각의 객체 클래스는 객체 지향 프로그래밍에서 일반적인 것과 같이, 좀더 총체적인 클래스의 부분할이 될 수 있다. 이러한 경우, 코드는 좀더 총체적인 클래스('수퍼클래스')와 관련하여 대신 저장될 수 있다. 객체 관리자(108)는 각각의 객체를 구성하는 데이터의 위치 리스트 포함하고, 각각의 객체 발생(즉, 객체 호출)에서, 객체 관리자는 클래스 코드 저장 영역(110)내 관련된 서브루틴 코드를 액세스하며, 객체 저장 영역(112)내 관련된 객체를 위한 데이터 주소를 코드에 공급한다.

도 7을 참조하면, 이 실시예에서, 인텔리전스 도메인의 호스트(10)내에 제공된 객체는 새로운 고객의 액세스에서 생성되고, 고객이 자발적으로 이탈하거나 또는 네트워크로부터 단절되는 경우 소멸되며, 고객의 요구가 변경되는 경우 편집되는 다수의 고객 객체(500)(하나는 수천만 고객 각각에 관련된 데이터를 보유); 그 각각이 각각의 통화의 아웃세트에서 생성되고 통화 종결후에 소멸되는 다수의 통화 객체(600a-600c); 및 각각이 고객 터미널 장치의 아이템과 관련되는 다수의 통신장치 객체(700a-700c)로 이루어지고, 고객 터미널 장비와 네트워크와의 제 1 연결상에서 생성된다.

도 7b를 참조하면, 이 실시예에서, 전송 영역(6)의 교환국(2a, 2b...)은 교환국내의 스위치 및 스위치의 포터를 각각 나타내는 객체(800a-800b) 및 객체(900a-900f)가 저장된 호스트 컴퓨터로 이루어진다. 따라서, 각각의 스위치 객체는 어떠한 순간에 대응하는 스위치의 상태 레코드를 포함한다; 이러한 객체들은 메모리에 영구적으로 존재하고, 포트 또는 스위치 객체로의 쓰기가 각각의 포트 또는 스위치의 상태를 변경하고, 포트 또는 스위치 객체 읽기가 대응하는 물리적 장치의 실제 상태를 정확하게 반영하도록 하기 위해 교환국(2)에 있는 물리적 장치로의 1대 1 사상을 갖는다.

예시에 의해. 고객 객체내 데이터의 구조가 도 8에 설명된다.

객체(500)에 의해 유지된 속성 데이터는 (고객이 고용인이거나 일부 다른 특수 상태에 있는지, 또는 정상적인 고객인지를 나타낼 수 있는) 고객 타입 필드(502); 사용자 ID 필드(504); 및 (일반적으로

http/TCP/IP 포맷으로) 객체(500)가 생성된 호스트(10)를 나타내는 호스트 필드(506)로 이루어진다.

또한 고객에게 제공된 서비스와 관련된 데이터가 저장된다; 예를 들어, 일반적인 고객의 전화번호(필드(508)); 고객 호출이 특정 날짜의 시간에 재루트되는 전화번호(필드(510)); 및 호출이 재루트되는 동안의 날짜의 시간(필드(512))이 있다.

마지막으로, 각각의 통화에서 호출된 (및/또는 호출한) 전화번호, 통화의 날짜 및 시간, 및 통화 비용을 저장하는 통화 로그 필드(514)의 형태로 고객의 청구서 정보가 저장된다(필드(514)).

이 정보의 다른 부분은 다른 개인에 의해 액세스될 필요가 있다. 예를 들어, 고객에게 제공될 서비스를 정의하는 필드(508-512)는 최종 사용자 터미널(15)을 통해 고객 자신 또는 고객 서비스 직원에 의해 편집될 수 있는 반면, 청구서 데이터(필드(514))는 네트워크를 운영하는 요금청구 및 요금계산에 의해서만 쓰기가능해야 한다. 확실히, 최종 사용자 터미널(15)로부터 그 요금청구 레코드를 재기록할 수 있어야 하는 고객은 없다.

운영중에, (최종 사용자 영역내 '오프 후크(off hook)' 상태를 모니터하는 것과 같은) 트랜스포트 도메인내 상황 발생은 인텔리전스 도메인내 객체와 관련된 코드의 동작을 발생시킨다. 예를 들어, 최종 사용자 도메인에서 전화가 오프 후크되는 경우, 새로운 '호출' 객체(600)를 생성하기 위한 코드가 발생된다. 호출된 번호가 검출되면, 상기 번호는 시그낼링 링크(12)를 통해 인텔리전스 도메인(8)으로 전송되고; 호출한 측의 고객 객체(500)는 그 청구서 작성 레코드 필드를 수정하기 위해 활성화되며; 호출된 측의 고객 객체(500)는 호출이 전송되어야 하는 번호를 결정하기 위해 액세스되며, 그러면, 호출이 전달될 경로를 설정하기 위해 전송 영역내 스위치 객체(800)로 정보가 전송된다. 호출이 전달되는 동안, 인텔리전스 도메인의 역할은 대개 제한된다. 온 후크(on hook) 상황의 검출에서 통화를 클리어하는 경우, 고객 객체(들)(500)와 관련된 청구서 작성 함수 코드는 청구서 작성 데이터 필드를 갱신하고, 통화 객체는 객체 관리자(108)에 의해 삭제된다.

도 9를 참조하면, 이 실시예에서 전치-컴파일러(316)는 각각의 명령문의 어떤 요소가 함수, 변수 등인지를 결정하기 위해 소스 코드 기억장치(308)내 연속적인 C++ 명령문을 검사하고 저장된 C++ 구문 그래프(316b)를 인용하는 파서 프로그램(316a); 및 파서(316a)에 의해 예정된 구문 조건을 검출하는 경우 소스 코드 기억장치(308)내 소스 코드에 추가되는 대응하는 추가 소스 코드를 생성하는 코드 쓰기 프로그램(316c)으로 이루어져 있다.

도 10을 참조하면, 일반적인 아이템의 전치-컴파일러의 동작은 다음과 같다. 단계(1002)에서, 전치-컴파일러(316)는 소스 코드 기억장치(308)의 소스 코드에 있는 모든 객체 클래스를 각각 나타내는 한 세트의 객체로 이루어지는 새로운 클래스로 구성되는 추가 데이터 구조('타입 모형')를 생성한다.

따라서, 접촉된 각각의 클래스에서, 새로운 객체가 생성되고, 그 클래스에 보유된 데이터 및 그 클래스에 의해 수행된 함수를 기록하는 데이터의 아이템을 포함한다. 좀더 상세히 후술되는 바와 같이, 이 정보는 실행시간에 각각의 호스트 프로세서(10)가 어떤 주어진 클래스의 새로운 객체를 오픈할 프로세스를 결정할 필요가 있는 정보를 갖도록 하기 위해 코드와 함께 컴파일된다.

따라서, 단계(1002)후에, 이러한 타입 모형(309a)을 설명하는 소스 코드가 소스 코드 기억장치(308)로 쓰여진다.

단계(1004)에서, 전치-컴파일러(316)는 소스 코드 기억장치(308)내 소스 코드를 읽고, 다른 호스트(10)로부터 메시지 및 수신한 경우 지역함수에 대응하는 통화를 수신하는 인보커 함수를 추가한다.

따라서, 소스 코드 기억장치(308)내 소스 코드내 함수('목표 함수')을 각각 검출하는 경우, 전치-컴파일러(316)는 또다른 프로세서로부터 목표 함수를 호출하도록 의도된 메시지를 수신하는 즉시 목표 함수를 호출할 명령문을 인보커 함수으로 삽입한다. 메시지로서 목표 함수으로부터 값을 대응적으로 반환하기 위한 코드도 추가된다.

따라서, 단계(1004)의 실행후에, 소스 코드 기억장치(308)내 소스 코드는 또다른 프로세서로부터 지역 프로시져 호출로 수신된 메시지를 변환하기 위한 인보커 코드(309b)를 포함한다.

전치-컴파일러(316)에 의해 컴파일되는 경우, 소스 코드(308)에서 사용되는 소스 코드내 어떠한 포인터는 컴파일러(312)가 분산 컴퓨팅 환경보다 하나의 프로세서를 예상하기 때문에 하나의, 공용, 메모리 공간내에 주소로 할당될 것이다. 그러나, 분산 컴퓨팅 환경에서, 많은 객체들이 물리적으로 다른 프로세서에 위치될 수 있고, 따라서 하나의 메모리 공간내에서 포인터를 통해 액세스될 수 없다.

따라서, 단계(1006)에서, 전치-컴파일러(316)는 기억장치(308)내 소스 코드를 스캔하고, 각각의 검출된 포인터 발생을 만일 그것이 다른 프로세서상에 있을지라도 관련된 객체의 실제 위치로 액세스하기 위해 사용될 수 있는 참조로 교체한다. 따라서, 단계(1006)의 동작후에, 소스 코드 기억장치(308)내 소스 코드는 또한 참조 코드(309c)를 포함한다.

각각의 경우에 추가된 코드는 한편으로 특수 코드 명령문, 및 다른 한편으로 '#include' 명령문으로 이루어지고, 컴파일러(316)로 하여금 라이브러리(314)에 저장된 라이브러리 루틴을 포함하도록 한다.

타입 모델 생성(Type Model Creation)

지금부터 도 11 내지 도 13을 참조하여, 단계(1002)에서 전치-컴파일러(316)에 의해 수행된 프로세스를 좀더 상세히 설명하도록 한다.

전치-컴파일러(316)는 소스 코드 기억장치(308)에 저장된 코드를 스캔한다. 클래스 선언 명령문을 검출하는 경우(단계(1010)), 전치-컴파일러(316)는 도 12에 도시된 바와 같이 새로운 클래스의 새로운 객체를 생성('실증')하기 위해 코드를 쓴다. 도 12에서, 명령문 '#include 'SysType Model.h''는 다른 것들 가운데에서 새로운 클래스 SysClass를 선언하는 추가 코드를 포함하고, 명령문 'SysClass class 1 ('X')'는 그 클래스의 새로운 객체 Class 1를 선언하며, 객체가 표시하는 클래스의 이름('X')을 저장한다.

따라서, 단계(1012)에서, 후자의 명령문은 소스 코드 기억장치(308)로 쓰여진다.

클래스 선언에서, 전치-컴파일러(316)가 단계(1014)에서 데이터 타입 선언(예를 들어 도 12에서의 명령문 'int X1')을 선언하는 명령문과 만나게 되는 경우, 문자열 데이터와 같은 변수의 이름 및 타입을 저장하는 정적 데이터 선언을 쓴다(단계(1016)).

클래스내 모든 데이터 선언이 스캔된 경우(단계(1018)), 전치-컴파일러(316)는 구성기 함수(즉, 실행 시간중에 관련된 클래스의 새로운 객체를 생성시키는 함수)의 발생을 위해 소스 코드 기억장치(308)내 코드를 스캔한다.

구성기 함수의 선언을 검출하는 경우(단계(1020)), 전치-컴파일러(316)는 구성기 함수의 정적 선언을 소스 코드 기억장치(308)로 쓴다(단계(1022)).

클래스 선언내 그 또는 모든 구성기 함수를 스캔한 후(단계(1024)), 전치-컴파일러(316)는 소스 코드 기억장치(308)내 클래스 선언에서의 함수(즉, 서브루틴, 프로그램, 또는 방법)의 선언 발생을 검출하고(단계(1026)), 함수가 검출되는 각각의 경우에서, 문자열 데이터와 같이 함수의 타입 및 텍스트를 포함하는 변수의 정적 선언을 쓴다.

일단 클래스내 모든 함수가 검출되면(단계(1030)), 전치-컴파일러(316)는 소스 코드 기억장치(308)내 소스 코드에서 선언된 다음 클래스와 동일한 방법으로, 전체 소스 프로그램이 이러한 방법으로 스캔될 때까지 진행한다(단계(1032)).

그러한 때에, 도 13에 도시된 바와 같이, 새로운 클래스 SysClass에서, 원시의 소스 코드로 선언된 각각의 클래스를 위한 객체(CL1,CL2,CL3,CL4...)는 수정된 소스 코드로 선언되었을 것이다.

인보커 생성(Invoker Creation)

지금부터 도 14 및 도 15를 참조하여, 도 10의 단계(1004)를 수행하는 전치-컴파일러(316)의 동작을 좀더 상세히 설명하도록 한다.

먼저, 단계(1040)에서, 전치-컴파일러(316)는 컴파일러(312)가 업무를 수신 및 전송하는 메시지를 수행하는 라이브러리(314)에 저장된 코드를 포함하도록 하는 '#include' 명령문을 삽입한다. 좀더 상세히 후술되는 바와 같이, 이러한 환경에서 '메세지'는 호스트 프로세서(10)의 작동 시스템으로부터 그 객체 관리자로 전송되고, 다른 호스트 프로세서(10)중의 하나 또는 동일한 호스트 컴퓨터(10)상에서 실행하는 또다른 프로세스로부터 발생할 수 있다.

메시지 번역을 형성하는 코드를 포함하는 라이브러리 파일로의 참조를 포함할뿐만 아니라, 전치-컴파일러(316)는 또한 그 실행을 발생시키도록 의도된 메시지의 도달시에 함수로 통화를 실행하고, 반환 메시지를 통해 어떠한 함수의 반환값을 반환하는 다수의 특수 프로시저로 구성되는 인보커 함수의 선언을 쓴다.

단계(1042)에서, 소스 코드 기억장치(308)내 소스 코드를 한번 더 스캔하는 동안, 전치-컴파일러(316)는 구성기 함수의 발생을 검출하고('X::X()'), 인보커 함수의 정의내에서, 수신된 메시지와 인보커 함수로의 호출에 대응하는 텍스트를 비교하며, 메시지가 매치되는 경우(즉, 구성기 함수으로의 호출을 실행하기 위한 시도를 구성하는 경우), 구성기 함수를 호출하는 코드 라인을 삽입한다(단계(1044)).

유사하게, 단계(1046)에서 어떤 다른 종류의 함수를 검출하는 경우(예를 들어, 도 15의 함수 'getX1()'), 단계(1048)에서 전치-컴파일러(316)는 인보커 함수내에서 수신된 메시지와 함수명 호출에 대응하는 텍스트를 비교하고, 메시지가 매치되는 경우 명명된 함수를 호출하는 코드 회선을 삽입하며, 호출된 함수의 반환값을 포함하는 반환 메시지를 생성한다.

따라서, 도 15에 도시된 바와 같이, 도 14의 프로세스 동작후에, 확장된 소스 코드는 메시지를 수신하고 응답으로 메시지를 전송하도록 배치되는 인보커 함수으로의 참조; 및 명명된 지역함수로의 통화 발생을 검출하고, 그 통화를 수행하며, (관련되는 경우) 값을 반환하는 그 함수내에 이루어진 일련의 특수 서브테스트로 이루어진다.

포인터 교체(Pointer Replacement)

도 16 및 도 17을 참고하여,도 10의 수행 단계(1006)에서의 프리-컴파일러(316)의 연산을 보다 더 자세히 설명하도록 하겠다.

단계(1052)에서, 상기 프리-컴파일러(316)는 소스 코드 저장기(308)내의 원시 소스 코드를 스캔하여 다른 것, 이름, 객체(예를들어, 도 17의 'X*otherX')로 포인터의 declaration 의 발생을 검출한다.

그러한 포인터의 다른 객체로의 검출이 있으면, 상기 프리-컴파일러(316)는 코드를 삽입(단계 1054)하여 새로운 객체의 탄생이 있게한다(상기 포인터를 포함하는 객체와 동일한 어드레스 공감을 같음, 즉 '로컬' 객체). 도 17을 참고하면, 이 경우, 상기 새로운 객체는 'XRef' 이다.

상기 새로운 객체의 구성과 함께, 상기 객체로의 셀상에서 수행된 코드가 만들어진다. 도 17을 참고하면, 상기 코드는 포인터 및 관련 연산자로 구성된 메시지를 전송하고, 회신으로 도달한 값을 돌려보내는 명령문을 구비하고 있다.

다음으로, 단계(1056)에서, 상기 원시 소스 코드내의 상기 원시 포인터 declari.., 'X*otherX' 는 'XRef otherX ' 로 수정되어, 로컬 객체 대신 원격 객체로 언급될 수 있게 한다.

따라서, 도 18을 참고하면, 도 18a 에 도 17의 원시 소스 코드의 컴파일에 의해 만들어졌던 효과가 나타나있다. 호스트 프로세서(10) 의 메모리 내부의 객체(X1)은 상기 호스트 프로세서(10)의 메모리 내부의 다른 곳의 객체(X2)로 호출하는 것을 실행하고, 응답 값을 돌려보낸다.

도 18b를 참고하면, 도 16의 처리 연산 후에, 상기 호출되는 객체(X1)은 제1 프로세서(10a)의 메모리내에 유지되고 객체(X2)는 제2 프로세서(10b)의 메모리내에 유지된다. 따라서, 컴파일 동안, 제3 객체(X3)가

상기 호출되는 객체(X1)와 동일한 메모리 공간내에서 생성되며, 그렇지 않으면 포인터 기준은 이제 'X3'를 언급하는 'X2'를 가리킨다. 그러한 기준은 로컬 '프록시' 객체(X3)에 의해 상기 제1 프로세서(10a)의 상기 객체 관리자(108), 운용체계(106), 프로세서(102) 및 통신 장치(100)를 구비하는 하위 레벨 프로세스로 상기 제2 프로세서(10b)로 전송용 메시지 형태로 통과되는데, 이것은 아래에서 좀 더 자세히 설명하도록 하겠다.

상기 메시지는 상기 제2 호스트 프로세서(10b)의 상기 통신 장치(100), 프로세서(102), 운용체계(106) 및 객체 관리자(108)에 의해 수신되는데, (앞서 도 14에서 삽입된 코드에 따라서) 상기 메시지는 상기 객체(X2)로 로컬 함수 호출로 변환되고, 객체(X2)에 의해 되돌려진 그 값은 호스트 프로세서(10a)로 메시지로서 되돌려 전송된다. 이 리턴 메시지는 상기 로컬 프록시 객체(X3)로 다시 통과되고, 그래서 상기 원시 호출 객체(X1)으로 통과된다.

그러므로, 상기 소스 코드 저장기(308)내의 상기 원시 소스 코드내의 그러한 모든 로컬 포인터들은 검출되고(단계 1058), 그렇지 않으면 상기 컴파일러(312)에 의해 발생되어온 모든 로컬 포인터들이 원시적으로 기준인 된 원격 객체로 교대로 발생 및 수신하는 로컬 프록시 객체로의 기준으로 변환된다.

상기 원시적으로 기준이 된 객체는, 도 18b에서 도시되어 있듯이, 물리적으로는 다른 호스트 프로세서상에 위치해 있으며, 또는 원시 호스트 프로세서(10)상에 위치하기도 한다; 메시지 통과 통신의 사용은, 로컬 포인터(일반적으로 C++ 코드에 의해 만들어진)들은 상기 객체가 그의 위치에 상관없이 도달될 수 있는 것을 보장한다.

런타임 동작(Runtime Operation)

앞서, 사전 컴파일 동안의 본 발명의 동작을 설명하였다. 지금부터 런타임에서의 실시예의 동작을 설명하도록 하겠다.

각 호스트 프로세서(10)내부에서, 상기 객체 관리자 프로그램(108)(상기 'daemon')은, 아래에서 보다 더 자세히 설명하겠지만, 새로운 객체 생성을 위한 프로세스; 주어진 객체가 저장되는 위치(즉, 호스트 컴퓨터(10)를 결정하는 프로세스; 원격 호스트 컴퓨터로 메시지를 전송하는 프로세스; 및 그것으로부터 메시지를 수신하는 프로세스를 구비한다.

도 19를 참고하면, 각 호스트 컴퓨터(10)의 객체 관리자(108)는 이름으로 현재 나타나는 객체의 리스트(1510)를 저장하는, 그리고 각 객체를 위한 레코드(1510a-1510z)에 관련된 객체 위치 테이블(1500), 상기 객체가 현재 저장된 호스트 컴퓨터(10)를 저장하는 엔트리(1520a-1520z), 및 상기 객체가 초기에 만들어진 호스트 컴퓨터의 아이덴티티를 저장하는 엔트리(1530a-1530z)를 구비하고 있다.

도 20을 참고하면, 상기 객체 관리자(108) 내부에 또 저장되어 있는 호스트 테이블(1600)은 추출될 수 있는 새로운 객체를 만들기 위해 가장 적합한 호스트 컴퓨터인 각 호스트 컴퓨터상의 상태 데이터를 구비하고 있다. 따라서 이 테이블은 각 호스트 컴퓨터의 아이덴티티를 저장하는 제1 엔트리 세트(1610a-1610z); 그 호스트 컴퓨터사에 사용가능한 여유 메모리의 양을 저장하는 관련된 제2 엔트리 세트(1620a-1620z); 상기 호스트 컴퓨터(10) 각 각을 위한 연결 데이터를 저장하는(예를들어, 접속되는 다른 컴퓨터의 수) 관련된 제3 엔트리 세트(1630a-1630z)를 구비하고 있다.

도 21a를 참고하면, 각 호스트 컴퓨터(10)의 객체 관리자(108)는 주기적으로 그것의 운용체계(106)에 질문을 하여 단계(2002)에서 그 프로세서용 테이블(1600)내에 저장된 데이터중 어느 것에 어느 변호가 있는지 여부(예를들어, 전보다 사용 메모리가 줄어들었는지 여부)를 결정하고, 그러한 정보에서 중요 변화가 검출되면, 단계(2004)에서 상기 객체 관리자(108)는 상기 통신 장치(100)를 통해 방송 메시지 전송을 상기 테이블(600)내에 있는 모든 다른 호스트 컴퓨터들에게 하고, 상기 새로운 정보에 관한 것을 알린다.

도 21b를 참고하면, 어느 호스트 컴퓨터(10)가 단계(2006)에서 그러한 메시지를 수신하게 되면, 그것의 객체 관리자(108)은 도 20의 상태 테이블(1600)을 업데이트 하여 단계(2008)에서 상기 새로운 정보를 반영하도록 한다.

따라서, 어느 때이고, 각 호스트 컴퓨터(10)의 객체 관리자(108)는 도 20의 상기 테이블(16) 내부의 자신의 것 뿐만 아니라 다른 모든 호스트 컴퓨터용 데이터 상태 정보까지 유지한다.

지금부터 도 22를 참고하여 새로운 객체를 구성하는 프로세스를 설명하도록 하겠다. 단계(2010)에서, 상기 지향된 객체를 생성하기 위한 호출을 하는 호스트 컴퓨터의 어드레스 공간내의 객체를 만들기보다는 객체 구성 함수을 호출하는 명령이 소스내에서 마주치지 않게되면, 본 실시예의 객체 관리 프로그램(108)은 단계(2010)에서 상기 호스트 프로세서 상태 테이블(1600)을 판독하고 생성될 객체의 클래스를 설명하는 객체(도 11 내지 도 13에 관련하여 앞서 설명된 프로세스에 의해 만들어진)내에 저장된 데이터는 멤버가 될 것이다.

그러므로, 상기 객체 관리 프로그램(108)은 생성될 객체의 요구(즉, 메모리 저장지의 양을 큰 것을 요구하는 지 작은 것을 요구하는지 여부 및 많은 통신 지점으로부터 또는 통신지점으로 액세스를 요구하는지 여부)를 알고 있으며 각 호스트 컴퓨터의 사용가능한 능력상의 데이터를 가지고 있다.

단계(2014)에서, 상기 객체 관리자(108)는 상기 객체가 생성될 호스트 컴퓨터(10)를 선택한다. 이 결정은 호스트 컴퓨터가 남아있는 사용가능 메모리가 가장 많거나 또는 호출될 새로운 객체로부터의 지점을 가장 최상의 접속으로 액세스 하는지에만 기초하여 이루어지기도 하지만, 적절하게는 웨이티드 함수이 사용된다:

각 프로세서상의 사용가능한 메모리;

각 프로세서의 접속도; 및

현 프로세서용 우선도(즉, 지향된 객체를 생성하기 위한 호출이 있은).

단계(2016)에서, 상기 객체 관리자(108)는 그 호스트 컴퓨터의 어드레스를 구비하는 선택된 호스트 컴퓨

터로 메시지를 전송하고 생성될 객체가 속한 클래스의 구성 함수을 구하기 위한 명령을 생성될 상기 객체의 이름과 함께 전송한다.

단계(2018)에서, 목표가 된 호스트 컴퓨터(10)에서, 상기 메시지는 수신되고 단계(2020)에서 호출 함수(컴파일 시간에서 상기 도14 및 도 15를 참고하여 설명된 것 처럼 만들어진)으로 통과된다. 동시에, 상기 새로운 객체가 생성될 목표가 된 호스트 컴퓨터(10)는 상기 새로운 객체를 포함하도록 그의 객체 테이블(1500)을 업데이트 한다.

마지막으로, 단계(2022)에서, 상기 목표가 된 호스트 컴퓨터는 다른 모든 호스트 컴퓨터(10)들에게 새로운 객체의 아이덴티티 및 위치를 방송하며, 이것은 단계(2024)에서 원시 호스트 컴퓨터(다른 것들 가운데)에서 수신되고 그것에서 객체 태이블(1500)을 업데이트 하는데 사용된다. 따라서, 본 실시예에 따르면, 각 호스크 컴퓨터상의 주기적으로 업데이트된 정보에 기초하여 가장 적합한 호스트 컴퓨터로 객체의 분산을 균형잡기 위해서 객체는 선택된 호스트 컴퓨터에서 런타임내에 생성된다.

도 23을 참고하면, 객체는 주어진 호스트 컴퓨터상에서 영구적으로 있게될 필요가 있으나 한 호스트에서 다른 호스트로 이동되어도 된다(즉, 한 호스트의 저장지로부터 삭제되고 다른 호스트의 저장지 메모리내에 포함됨).

단계(2030)에서, 각 호스트 컴퓨터(10)용 객체 관리자(108)는 그것의 상태 테이블(1500)을 재검토된다. 단계(2032)에서, 상기 객체 관리자 프로그램(108)은 객체가 상기 객체 관리자(108)가 상주하는 호스트 컴퓨터(10)로부터 제거되었는지를 결정하는데; 예를들어 민일 그 호스트 컴퓨터(10)가 사용가능한 메모리 밖에서 동작되는지 여부를 결정한다. 만일 그렇다면, 이동하기 위한 적절한 객체가 선택된다(예를들어, 드믈게 액세스된 객체).

단계(2034)에서, 상기 객체 테이블(1500)은 상기 객체가 생성된 호스트상에 현재 상주하는지를 상기 객체용 해당 필드(1530)내의 엔트리로부터 결정하기 위해 재검토된다. 만일 그렇다면, (즉, 만일 객체가 생성된 이후 이동하지 않았다면), 단계(2036)에서, 새로운 호스트가 선택되고(상기 새로운 객체의 생성에 관련되어 설명된 것과 아주 동일한) 단계(2038)에서 그 객체는 이동된다.

단계(2033)에는 두 성분이 있다: 먼저 호스트에서 상기 객체의 생성을 명령하고 그 객체 내부에 저장된 데이터의 값을 지정하는 새로운 호스트로 전송된 메시지; 및 두번째로, 현재 호스트 컴퓨터(10)상의 메모리로부터 삭제된 객체가 그것이다.

만일 단계(2034)에서 객체가 생성된 호스트 컴퓨터(10)상에 현재 상주하지 않고 있다면(즉, 객체가 이미 생성된 이후 이동했다면), 단계(2040)에서 상기 객체 관리자는 이 새로운 호스트 위치를 상기 객체가 원시 생성되었던 원시 호스트 컴퓨터(10)로 메시지를 전송하고(그 객체용 엔트리(1530) 및 테이블(1500)로부터 결정된 것과 같이), 그 원시 호스트 컴퓨터로부터 확인을 수신한 다음, 단계(2042)에서 상기 설명한 절차를 진행한다(단계 2036).

지금부터 도 24를 참고하여 상기 객체 위치 테이블(1500)의 사용에 대해 좀 더 자세히 설명하도록 하겠다.

객체로 메시지가 전송되는 경우, 먼저 객체 관리자(108)는 객체 위치 테이블(1500) 내부에서, 상기 객체가 위치하는 것으로 레코드된 현재 호스트 컴퓨터(10)를 결정하고, 그 객체를 호출하면서 단계(2050)에서 그 호스트로 메시지를 전송한다.

대부분의 경우, 메시지는 현재 호스트로 도달하게 되지만, 통신 동안의 간섭 또는 다른 원인으로 인하여 상기 객체 위치 테이블(1500)이 완전히 업데이트 되지 않을 가능성도 있어서, 상기 호출된 객체가 지시된 호스트 컴퓨터상에 있지 않을 수 도(또는 더이상 있지 않음)도 있다.

도 24b를 참고하면, 메시지가 어드레스되는 호스트 컴퓨터(10)는 단계(2052)에서 메시지를 수신하고, 객체가 단계(2054)에서 그 자신의 새로운 위치 테이블(1500)내부에 현재있는 레코드인지 아닌지를 결정한다. 만일 존재한다면, 객체는 단계(2056)에서 호출되며 리턴 메시지는 앞서 설명한 바와 같이, 단계(2058)에서 되돌려 전송된다.

그렇지 않다면, 단계(2060)에서, 상기 호스트 컴퓨터는 그 객체 호출 실패를 되돌려 알린다.

단계(2062)내에서 그러한 메시지를 수신하면, 단계(2066)에서 상기 원시 호스트 컴퓨터는 문의 메시지를 그 질문의 객체가 원시 구성되었던 것 처럼 상기 테이블(1500)내에 레코드된 원시 호스트 컴퓨터(10)로 전송한다. 도 23의 분리 위치 시그널링 단계(2040,2042)로 인해, 상기 객체가 구성되었던 원시 호스트 컴퓨터상의 객체 위치 테이블(1500)은 그 객체에 대한 데이터로 완전히 되어야 한다.

도 24c를 참고하면, 단계(2068)에서 상기 원시 호스트 컴퓨터가 위치 질의 신호를 수신하게 되면, 그것의 객체 관리자(108)는 단계(2070)에서 그 내부에 부비된 객체 위치 테이블을 언급하고 단계(2072)에서 관련된 객체의 결정된 위치를 되돌려 보낸다.

제1 호스트 컴퓨터는 상기 원시 호스트 컴퓨터로부터 상기 위치 신호를 수신하고, 단계(2074)에서 그 호스트 컴퓨터가 상기 객체가 이미 단계(2050)에서 지시되었는지를 위한 메시지와는 다른 객체의 위치를 가리키고 있는지를 결정한다. 만일 그렇다면, 상기 객체 관리자(108)는 단계(2076)에서 상기 위치 테이블(1500)을 업데이트 하고, 단계(2050)으로 되돌아가 그 객체를 위해 새롭게 지시된 호스트 컴퓨터(10)로 상기 메시지를 반복하여 전송한다.

만일 수신된 어드레스가 그 메시지가 사전에 지시된 어드레스와 동일하다면, 단계(2078)에서, 상기 객체 관리자 프로그램(108)은 발생된 그 객체를 호출하려도 했던 곳으로부터 그 객체, 프로세스 또는 사용자로 상기 객체에 도달하는 것이 실패하였음을 나타내는 메시지를 되돌려 보낸다.

따라서, 본 실시예에서, 각 호스트 컴퓨터(10)는 시스템 내의 모든 객체의 위치가 레코드된 객체 위치 테이블을 운반하고, 그로부터 각 호스트는 중앙 데이터베이스를 기준할 필요없이 호스트 컴퓨터 상의 다른 객체를 호출하는 것을 자율적으로 할 수 있다. 그럼에도 불구하고, 객체의 위치가 다른 호스트 컴퓨터로

연속적으로 이동하는 경우라 할지라도 그 호스트 컴퓨터상에서 원시 생성되었던 모든 객체의 위치의 트랙을 추가적으로 유지하는 각 호스트 컴퓨터(10)를 제공함으로서 보전성의 상위 레벨이 부가된다.

더욱이, 각 호스트 컴퓨터(10)가 그 자신의 저장 영역 내부에 위치한 이러한 객체들의 정확한 레코드를 유지한다는 것은 명백한 것이다.

다른 실시예 및 수정

다른 호스트 컴퓨터상에 저장된 객체들 뿐만 아니라 각 호스트 컴퓨터 내부에 국부적으로 저장된 객체를 위한 그리고 그 호스트 컴퓨터상에서 원시 생성된 객체를 위한 분리 테이블을 제공할 수 있음은 물론이다. 그러나, 모든 객체 위치를 저장하는 단일 테이블을 제공하는 것은 모든 호스트 컴퓨터(10)의 모든 객체 위치 테이블(1500)이 충분히 동일하기 때문에, 어느 하나의 호스트 컴퓨터로부터의 객체 위치 테이블을 새로운 호스트 프로세서상으로 복사하기 위한 새로운 호스트 프로세서 시스템을 포함하기 위해 분산 컴퓨팅 시스템의 확장이 있게 하여 더 큰 유연성을 보장한다.

이상의 설명으로부터 많은 수정 및 대체가 가능하다는 것은 명백한 것이다. 예를들어, 상기 이유로 인해, 비록 종래의 C++ 컴파일러가 있는 전치-컴파일러로 본 발명을 제공하는 것이 편리하긴 하지만, 본 발명을 종래 컴파일러가 아닌 컴파일러로 통합하는 가능성도 있다(그리고 전치-컴파일러 및 컴파일러의 조합은 바로 이점을 구비하도록 고려되기도 한다).

당연히, 본 발명은 C++ 뿐만 아니라 분산 환경에 제공될 Smalltalk(TM) 등의 다른 객체지향 언어를 사용할 수 있다. 더욱 일반적으로는, 비 객체지향 언어 및 비-분산 시스템에서의 동일한 기술을 사용하는 것도 가능하다.

본 발명을 전기통신 시스템 내부의 컴파일러 장치를 포함하여 설명하긴 했으나, 실제로는 상기 컴파일러는 다른 영역내에 있을 수 도 있고 국제 전기통신망을 통해 호스트 컴퓨터와 링크됨으로서 실현될 수 있다; 또한, 상기 컴파일러 장치는 사용된 전기통신망과의 조합 및 통신망으로부터의 분리 모두에 발명의 보호를 요구한다.

물론, 공유 분산 컴퓨팅과 같은 전기통신이 아닌 다른 애플리케이션도 가능하다.

통상의 지식을 가진 자에게는 많은 다른 선택 및 수정이 명백하다 할 것이다. 또한, 본 발명은 본 명세서에 기재된 어느 사항 및 모든 사항들을 포함하는 것이며 첨부된 특허청구범위만으로 제한되는 것은 아니다.

본 출원인에 의해 본 발명과 같은 날에 동일한 제목으로 출원된 영국 특허출원 제9600854.5호는 본 발명에서 사용가능하며 본 발명의 참고가 된다. 특히, 그곳에서 도출되는 기능적 정보들은 본 명세서에 서술된 타입 모델내에 저장되기도 한다.

(57) 청구의 범위

청구항 1

각 각이 미리 결정된 데이터 레코드(112) 및 상기 데이터 레코드(112)를 액세스하는 프로그램(110)을 저장하는 메모리 공간(104)으로 구성되는 다수의 호스트 컴퓨터(10)를 구비하고, 각 각의 호스트 컴퓨터(10)상에서 모든 호스트 컴퓨터(10)상의 모든 데이터 레코드(112)의 테이블(150)을 공급하는 것을 특징으로 하는 분산 컴퓨팅 시스템.

청구항 2

제 1 항에 있어서,

상기 데이터 레코드(112)는 제1 호스트 컴퓨터(10a)로부터 제2 호스트 컴퓨터(10b)까지 이동하기도 하는 것을 특징으로 하는 분산 컴퓨팅 시스템.

청구항 3

제 2 항에 있어서,

그러한 이동에서, 레코드가 위치하게 될 상기 호스트 컴퓨터의 위치가 각 각 다른 호스트 컴퓨터(10)로 알려지는 것을 특징으로 하는 분산 컴퓨팅 시스템.

청구항 4

제 2 항에 있어서,

상기 각 각의 위치 테이블(1500)은 상기 데이터 레코드가 만들어진 호스트 컴퓨터(10)의 위치도 저장하는 것을 특징으로 하는 분산 컴퓨팅 시스템.

청구항 5

제 4 항에 있어서,

데이터 레코드를 액세스하기 위해, 상기 컴퓨터 각 각이 상기 위치 테이블(1500)를 판독하고, 필요한 경우 상기 데이터 레코드(112)가 만들어진 호스트 컴퓨터(10)의 위치 테이블(1500)를 액세스 하도록 배치되는 것을 특징으로 하는 분산 컴퓨팅 시스템.

청구항 6

제 1 항 내지 제 5 항중 어느 한 항에 있어서,

상기 데이터 레코드(112)는 관련된 코드(110)를 통해 액세스가 가능할 뿐인 각 각의 객체를 구비하는 것을 특징으로 하는 분산 컴퓨팅 시스템.

청구항 7

제 6 항에 있어서,

상기 객체는 동일한 클래스의 다중 객체(112)를 구비하고, 상기 클래스의 다중 객체는 상기 관련된 코드(110)를 공유하는 것을 특징으로 하는 분산 컴퓨팅 시스템.

청구항 8

상호 연결된 다수의 컴퓨터(10)로 구성되는 분산 제어 시스템 및 상기 컴퓨터(10)용 프로그램을 컴파일하는 컴파일링 장치(30)를 구비하고, 상기 장치(30)는 하나의 호스트 컴퓨터(10)상에서의 실행을 위한 소스 프로그램을 컴파일하기 위해 배치된 컴파일러(312), 및 단일-컴퓨터 실행에 적합한 원시 소스 프로그램을 수신하기위해 배치되고, 추가의 소스 코드(309)를 부가하기 위해 상기 원시 소스 프로그램을 수정하여, 상기 컴파일러(312)에 의해 컴파일된 경우, 상기 분산 제어 시스템내에서 실행될 수정된 소스 프로그램을 만들기 위해 배치된 프리-컴파일러(316)를 구비하는 것을 특징으로 하는 전기통신 시스템.

청구항 9

제 8 항에 있어서,

상기 컴파일링 장치(30)는 상기 상호 연결된 컴퓨터(10) 중 하나에서 실행을 위해 각 각의 상기 컴파일된 프로그램의 다중 복사를 발생하기 위해 배치되는 것을 특징으로 하는 전기통신 시스템.

청구항 10

상호 연결된 다수의 컴퓨터(10)로 구성된 분산 제어 시스템 사용을 위한 컴파일러 장치(30)에 있어서, 상기 컴파일러 장치는 하나의 컴퓨터(10)상에서 컴파일에 적합한 소스 언어내의 원시 소스 프로그램을 수용하기 위해 배치되고, 상기 분산 제어 시스템은 느슨하게-결합되어 있고; 상기 컴파일러 장치는 상기 원시 소스 코드 프로그램을 수용하기 위해 그리고, 상기 원시 소스 프로그램 내부에서, 상기 소스 프로그램내의 다른 지점들로부터 구해지기도 하는 함수들을 위치시키기 위해, 및 상기 원시 소스 프로그램을 적합하게 하여 상기 함수들이 다른 컴퓨터(10)로부터 원격으로 구하게 되도록 배치되는 것을 특징으로 하는 컴파일러 장치.

청구항 11

상호 연결된 다수의 컴퓨터(10)로 구성된 분산 제어 시스템을 사용하는 컴파일러 장치(30)에 있어서, 상기 컴파일러 장치는 하나의 컴퓨터(10)상에서의 컴파일에 적합한 소스 언어내의 원시 소스 프로그램을 수용하기 위해 배치되고, 상기 분산 제어 시스템은 느슨하게-결합되고; 상기 컴파일러 장치(30)는 컴파일러(312)를 구비하고, 상기 원시 소스 프로그램 내부에서, 상기 컴파일러(312)에 의해 컴파일 될 포인터를 하나의 컴퓨터 메모리 공간 내부의 영역으로의 포인터로서 위치시키기 위해 배치되고, 그리고 다른 상기 컴퓨터(10)의 메모리 공간내의 객체를 나타낼 수 있도록 하기 위해 상기 원시 소스 프로그램을 수정하기 위해 배치되는 것을 특징으로 하는 컴파일러 장치.

청구항 12

상호 연결된 다수의 컴퓨터(10)로 구성된 분산 제어 시스템을 사용하는 컴파일러 장치(30)에 있어서, 상기 컴파일러 장치는 하나의 컴퓨터(10)상의 컴파일에 적합한 소스 언어내의 원시 소스 프로그램을 수용하기 위해 배치되고, 상기 분산 제어 시스템은 느슨하게-결합되고; 그리고 상기 컴파일러 장치(30)는 상기 원시 소스 프로그램을 판독하고 거기에 더하여 상기 원시 소스 프로그램 내부에 구성된 상기 데이터 및 프로그램 구조를 설명하는 데이터 레코드의 세트를 부가하기 위해 배치되는 것을 특징으로 하는 컴파일러 장치.

청구항 13

제 8 항 내지 제 12 항중 어느 한 항에 있어서,

상기 원시 소스 프로그램은 객체지향 언어인 것을 특징으로 하는 장치 또는 시스템.

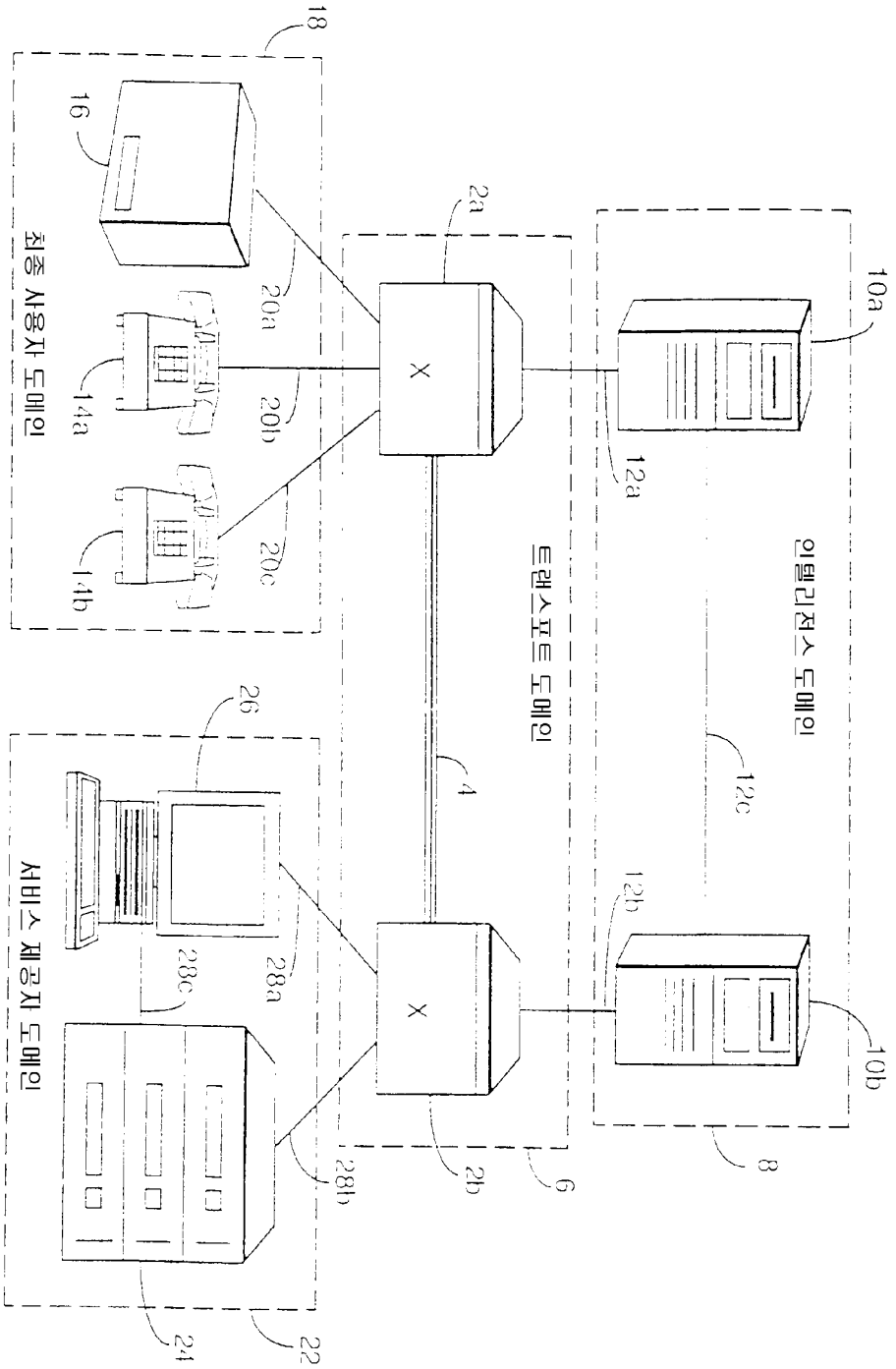
청구항 14

제 13 항에 있어서,

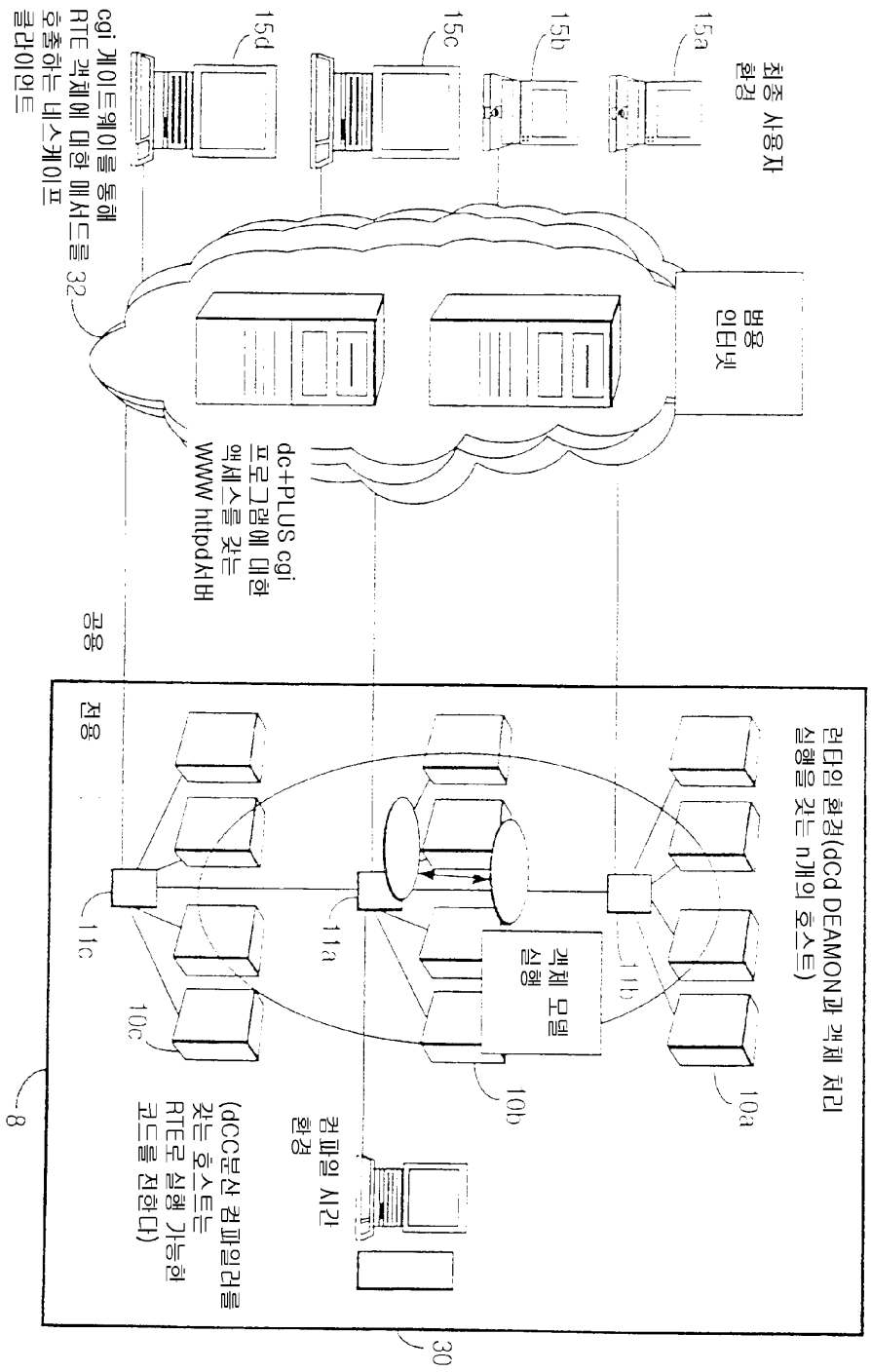
상기 언어는 C++ 또는 그의 변형인 것을 특징으로 하는 장치 또는 시스템.

도면

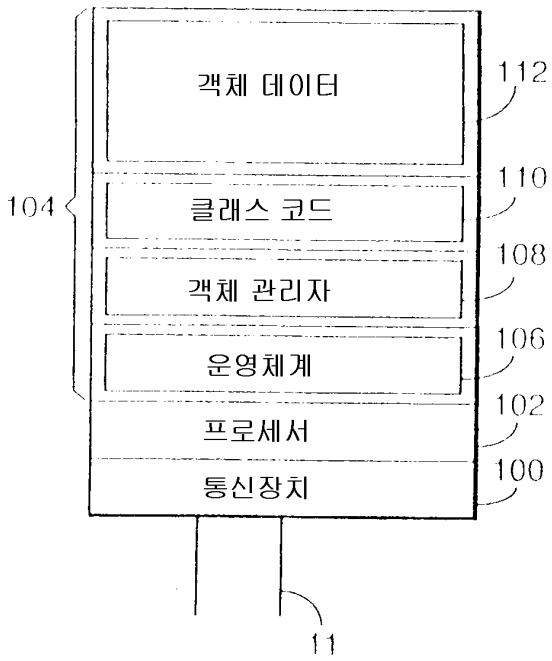
도면1


18
16
최종 사용자 도메인
20a
20b
20c
14a
14b
2a
10a
12a
인텔리전스 도메인
트랜스포트 도메인
4
12c
12b
10b
8
6
2b
28b
26
28a
28c
서비스 제공자 도메인
24
22

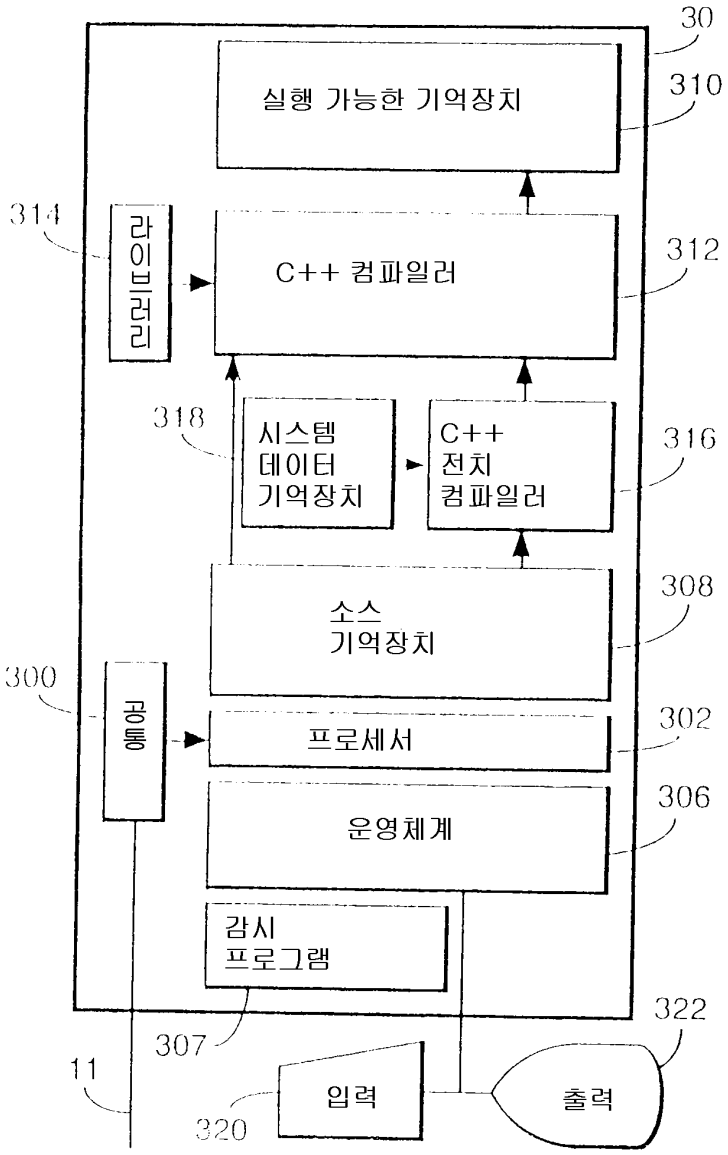
도면2


최종 사용자
환경
15a
15b
15c
15d
범용
인터넷
32
cgi 게이트웨이를 통해
RTE 객체에 대한 메서드를
호출하는 네스케이프
클라이언트
dc+PLUS cgi
프로그램에 대한
액세스를 갖는
WWW httpd서버
공용
전용
8
11c
11a
11b
10c
10b
10a
객체 모델
실행
컴파일 시간
환경
(dCC분산 컴파일러를
갖는 호스트는
RTE로 실행 가능한
코드를 전한다)
런타임 환경(dCd DEAMON과 객체 처리
실행을 갖는 n개의 호스트)
30

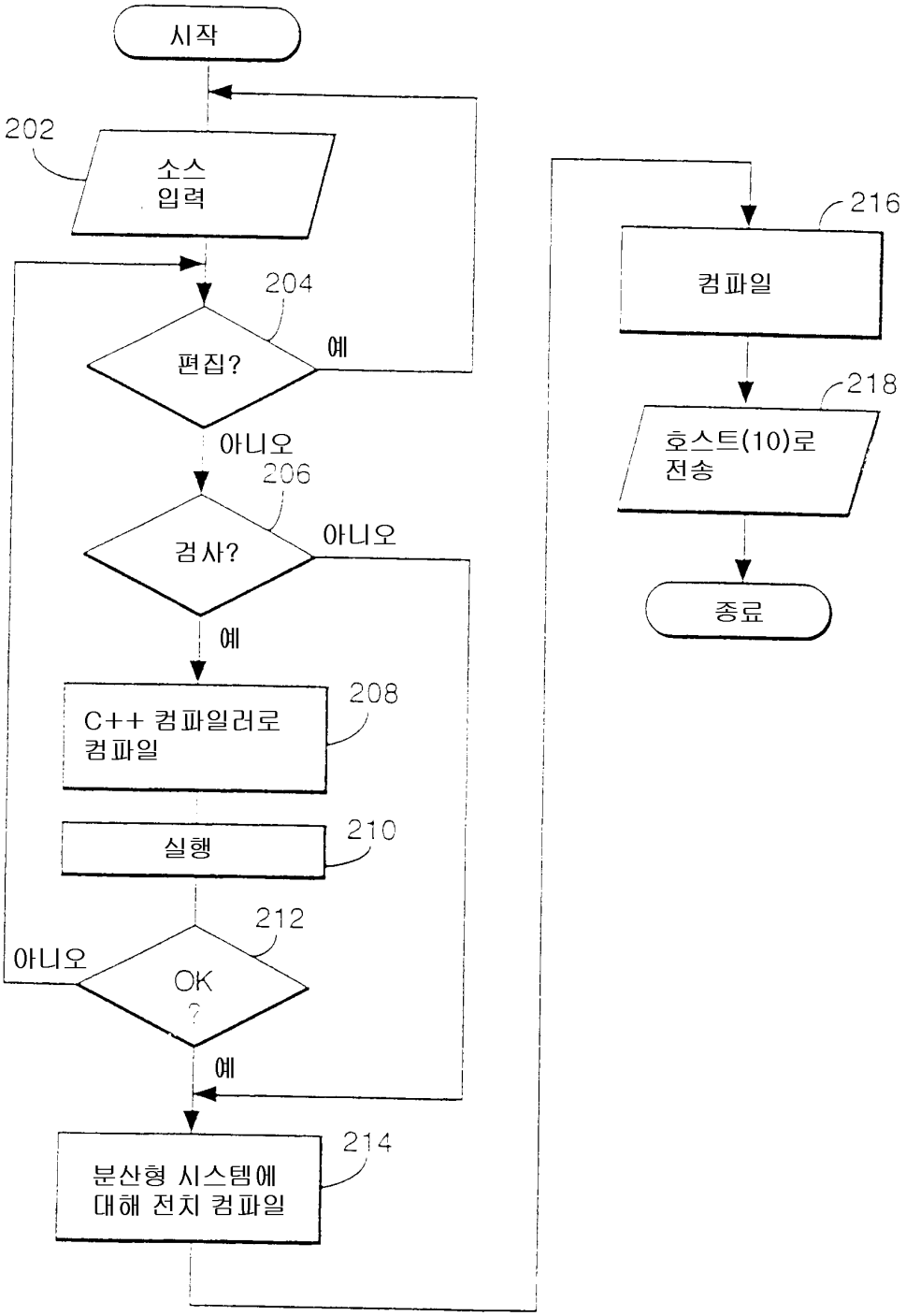
도면3


객체 데이터
112
클래스 코드
110
104
객체 관리자
108
운영체계
106
프로세서
102
통신장치
100
11

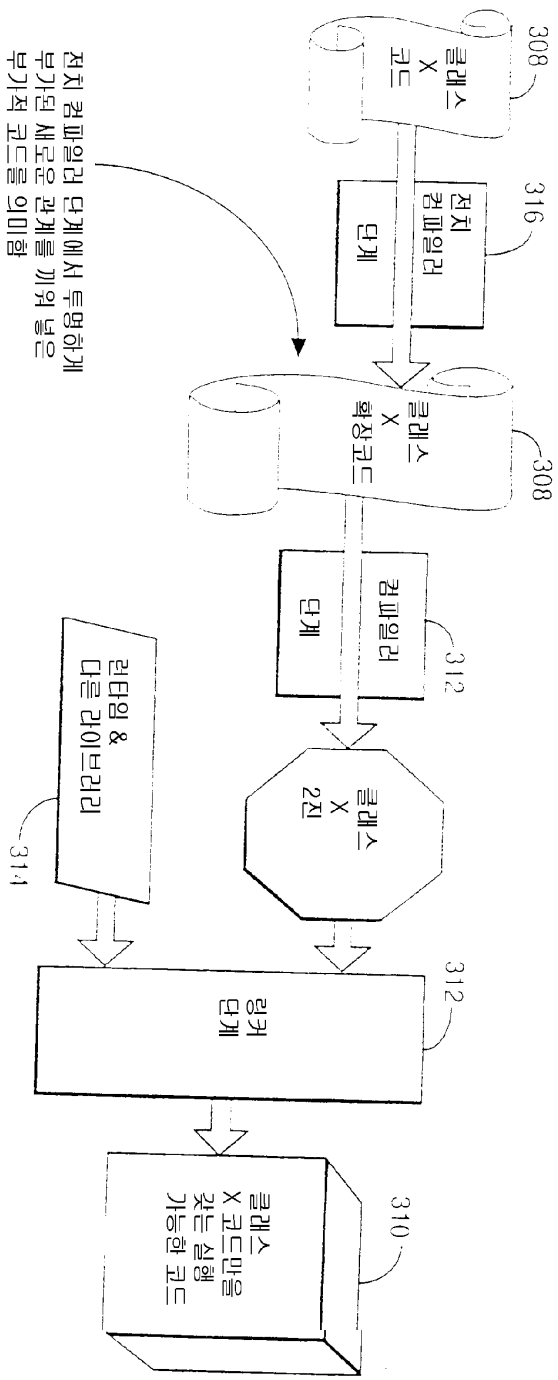
도면4


30
실행 가능한 기억장치
310
314
라이브러리
C++ 컴파일러
312
318
시스템 데이터 기억장치
C++ 전치 컴파일러
316
소스 기억장치
308
300
공통
프로세서
302
운영체계
306
감시 프로그램
307
322
11
입력
320
출력

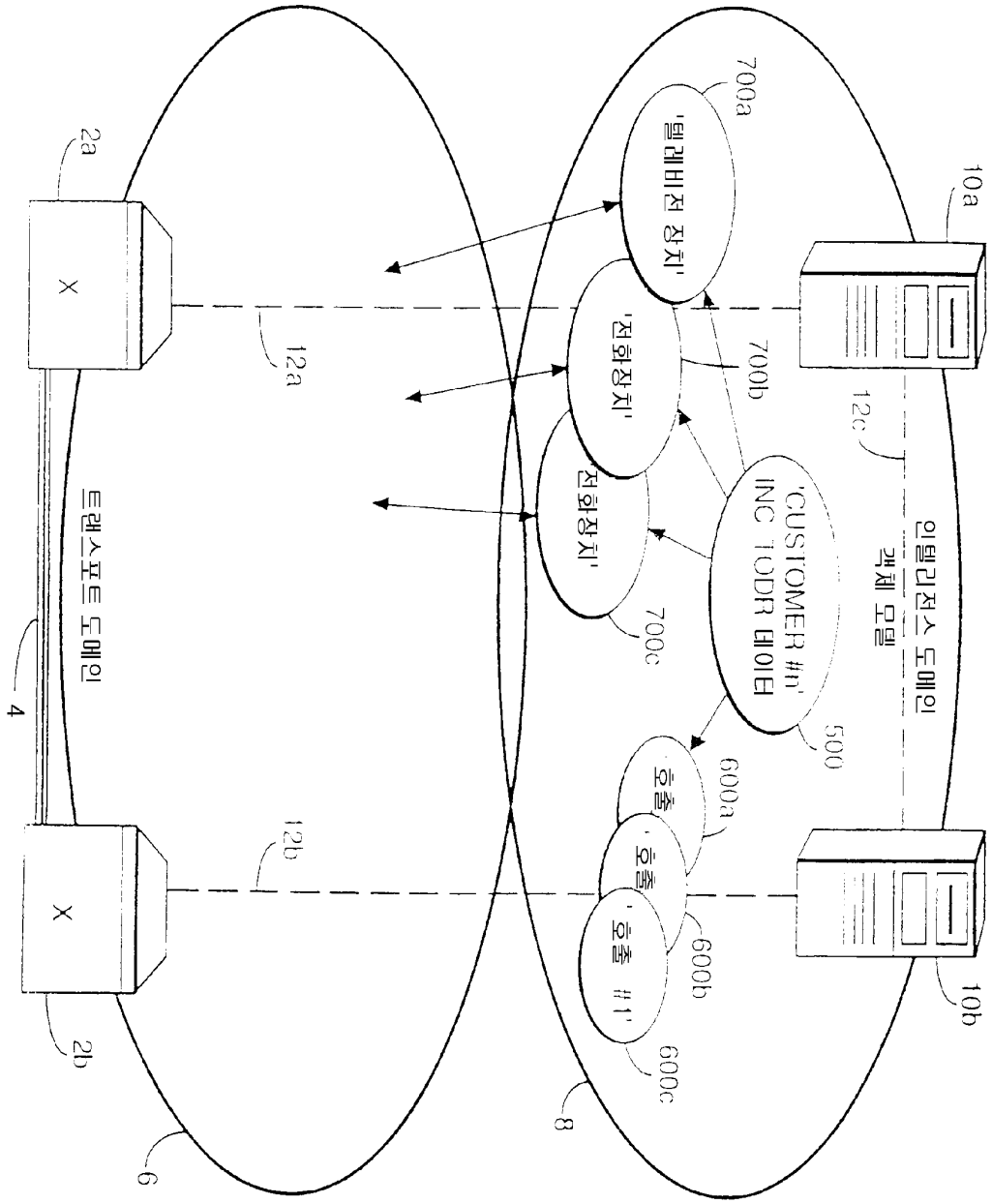
도면5


시작
202
소스
입력
204
편집?
예
아니오
206
검사?
아니오
예
208
C++ 컴파일러로
컴파일
210
실행
212
아니오
OK
?
예
214
분산형 시스템에
대해 전치 컴파일
216
컴파일
218
호스트(10)로
전송
종료

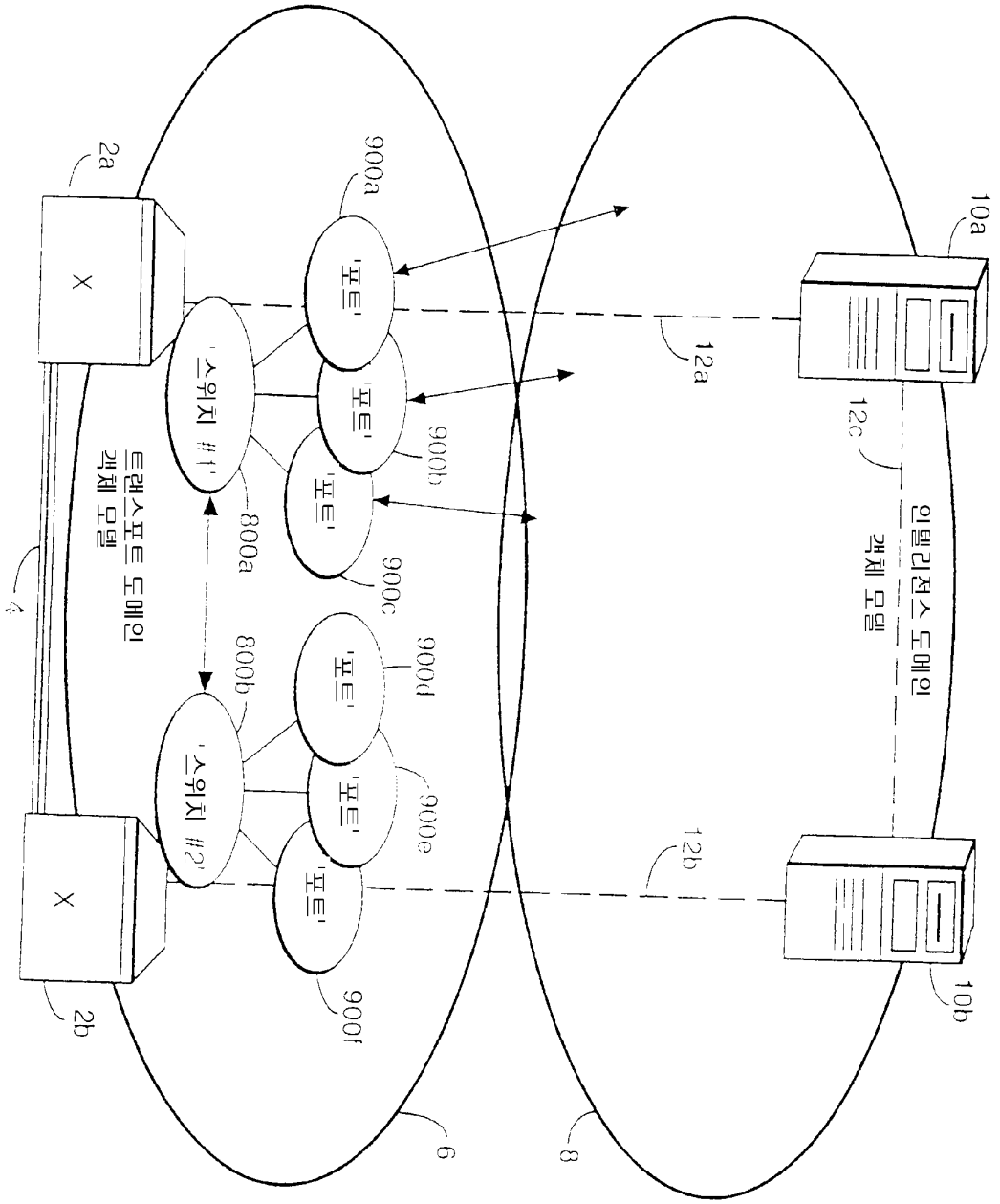
도면6


전치 컴파일러 단계에서 투명하게 부가된 새로운 관계를 끼워 넣은 부가적 코드를 의미함
클래스 X 코드
308
전치 컴파일러 단계
316
클래스 X 확장코드
308
컴파일러 단계
312
클래스 X 2진
런타임 & 다른 라이브러리
314
링커 단계
312
클래스 X 코드만을 갖는 실행 가능한 코드
310

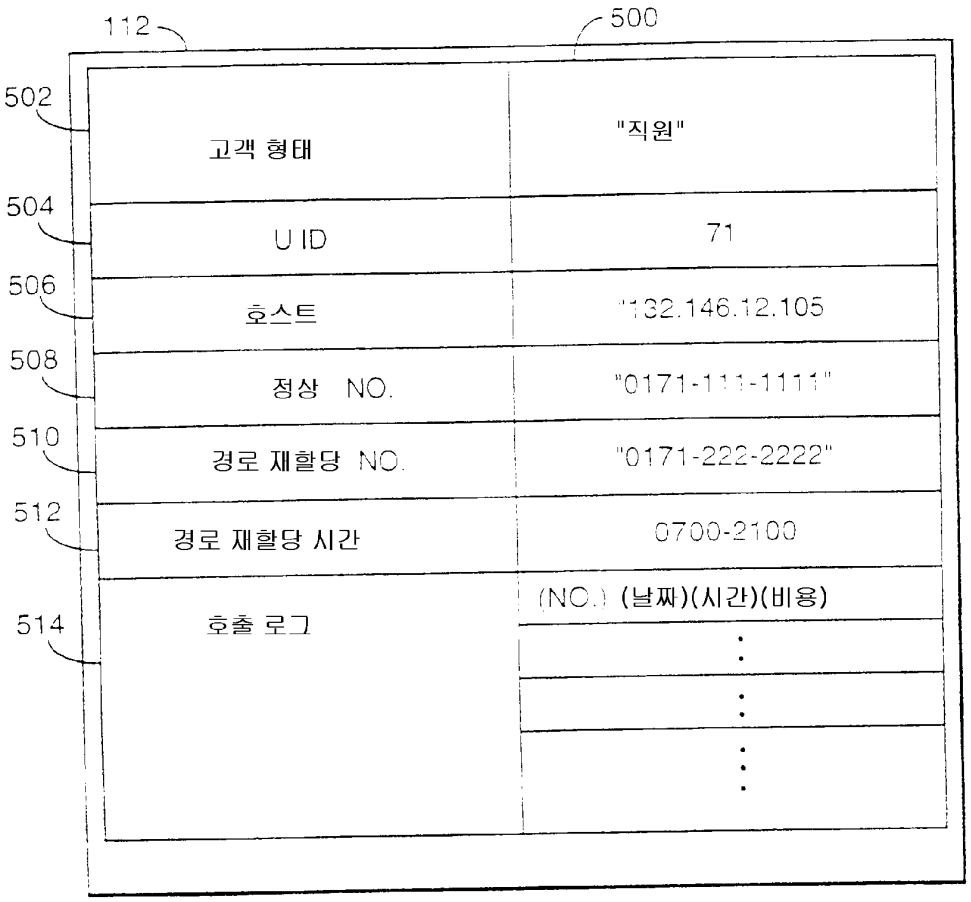
도면7a


2a
X
12a
트랜스포트 도메인
4
12b
X
2b
6
700a
'텔레비전 장치'
700b
'전화장치'
'전화장치'
700c
'CUSTOMER #n'
INC TODR 데이터
500
12c
인텔리전스 도메인
객체 모델
10a
600a
호출
'호출
600b
'호출 #1'
600c
10b
8

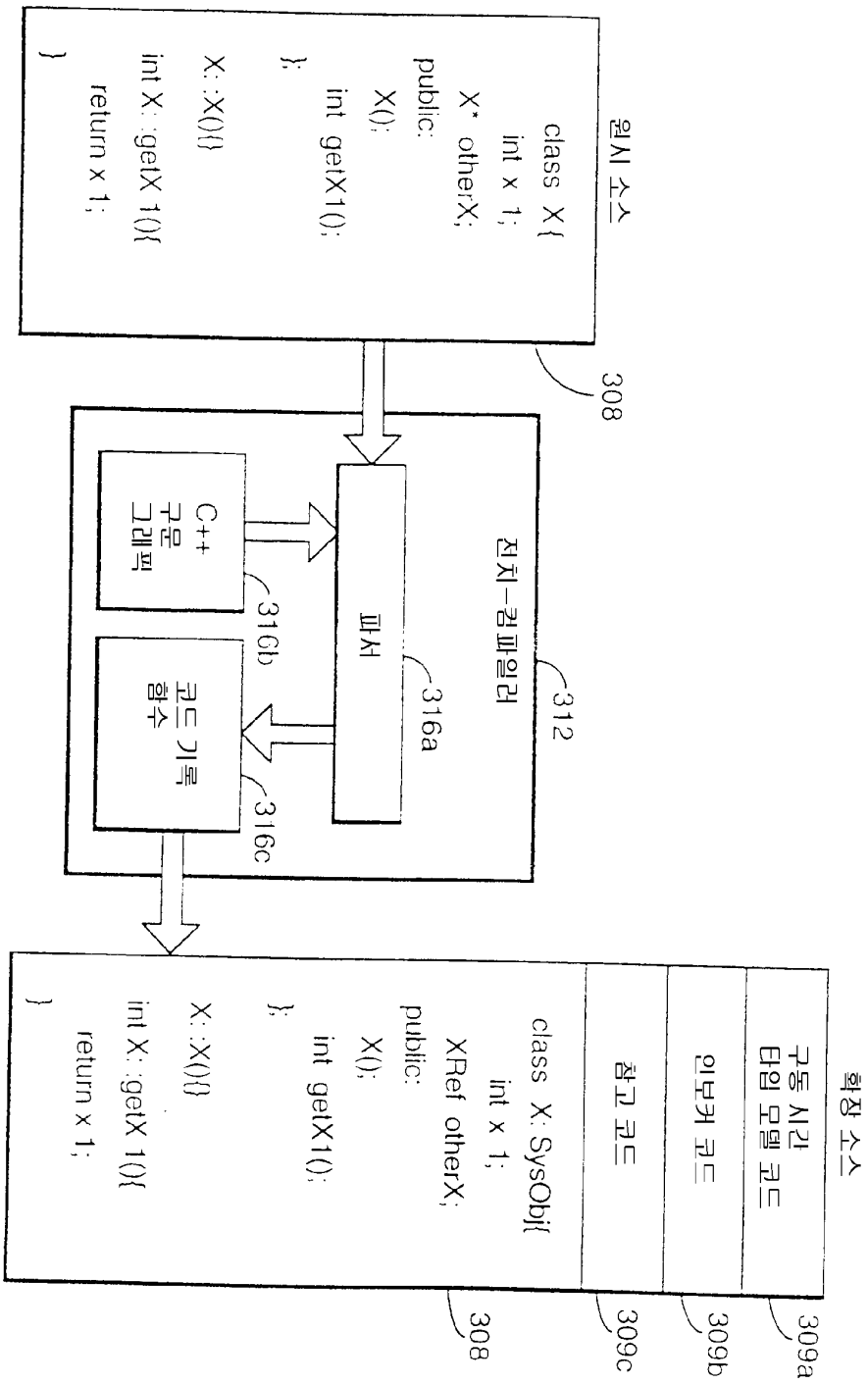
도면7b


2a
X
'스위치 #1'
800a
'포트'
900a
'포트'
900b
'포트'
900c
트랜스포트 도메인
객체 모델
4
800b
'스위치 #2'
X
2b
'포트'
900d
'포트'
900e
'포트'
900f
6
8
10a
12a
12c
인터리소스 도메인
객체 모델
12b
10b

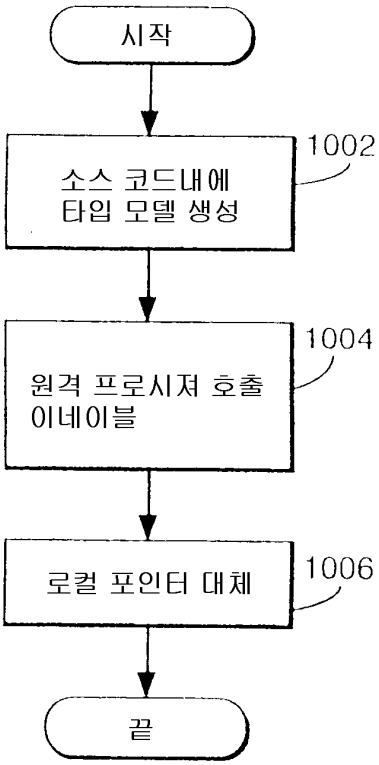
도면8


112
500
502
504
506
508
510
512
514
고객 형태
"직원"
U ID
71
호스트
"132.146.12.105
정상 NO.
"0171-111-1111"
경로 재할당 NO.
"0171-222-2222"
경로 재할당 시간
0700-2100
호출 로그
(NO.) (날짜)(시간)(비용)

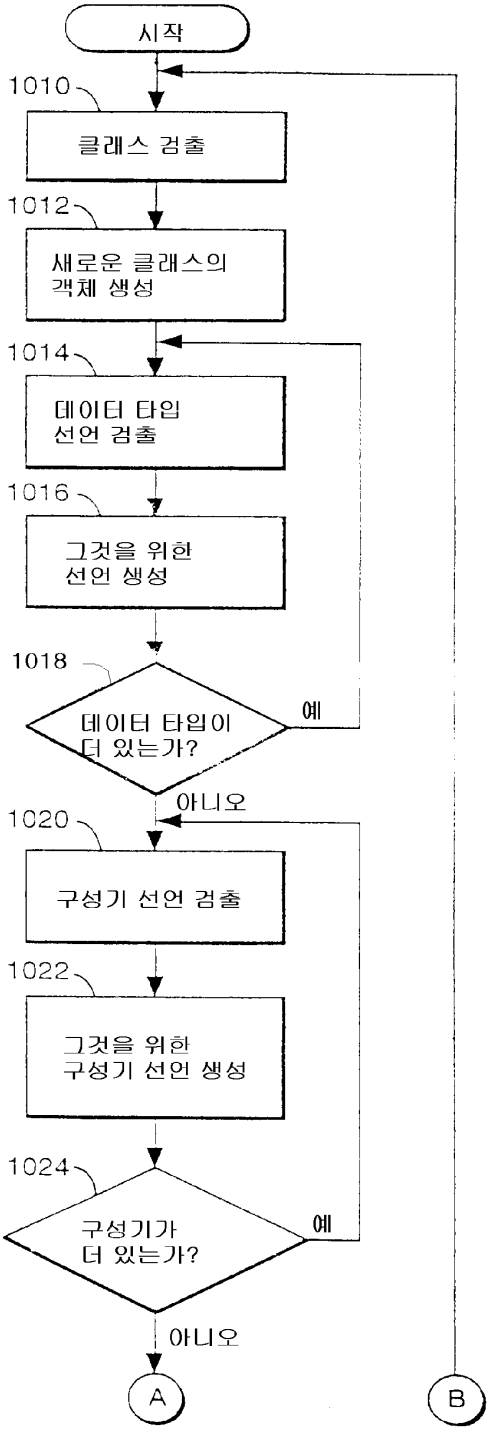
도면9


원시 소스
class X {
int x 1;
X* otherX;
public:
X();
int getX1();
};
X::X(){}
int X::getX 1(){
return x 1;
}
308
전치-컴파일러
312
C++
구문
그래픽
316b
파서
316a
코드 기록
함수
316c
확장 소스
구동 시간
타입 모델 코드
309a
인보커 코드
309b
참고 코드
309c
class X: SysObj{
int x 1;
XRef otherX;
public:
X();
int getX1();
};
X::X(){}
int X::getX 1(){
return x 1;
}
308

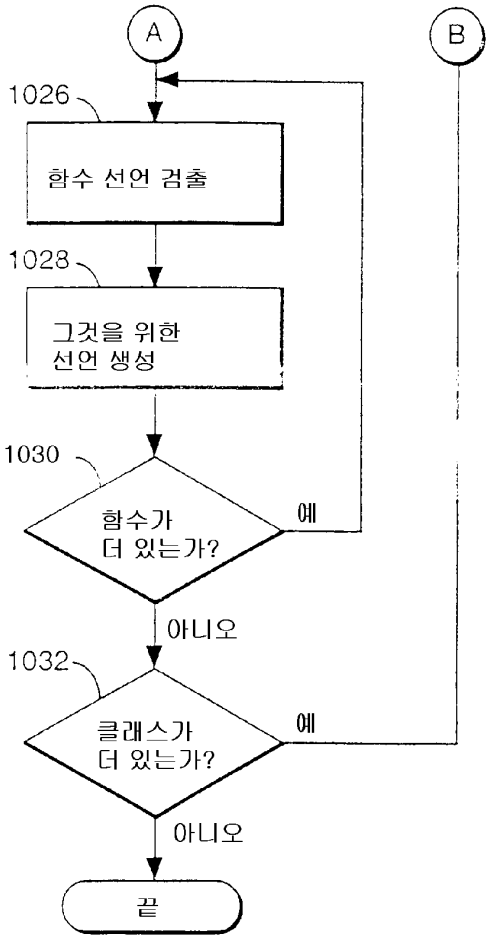
도면10



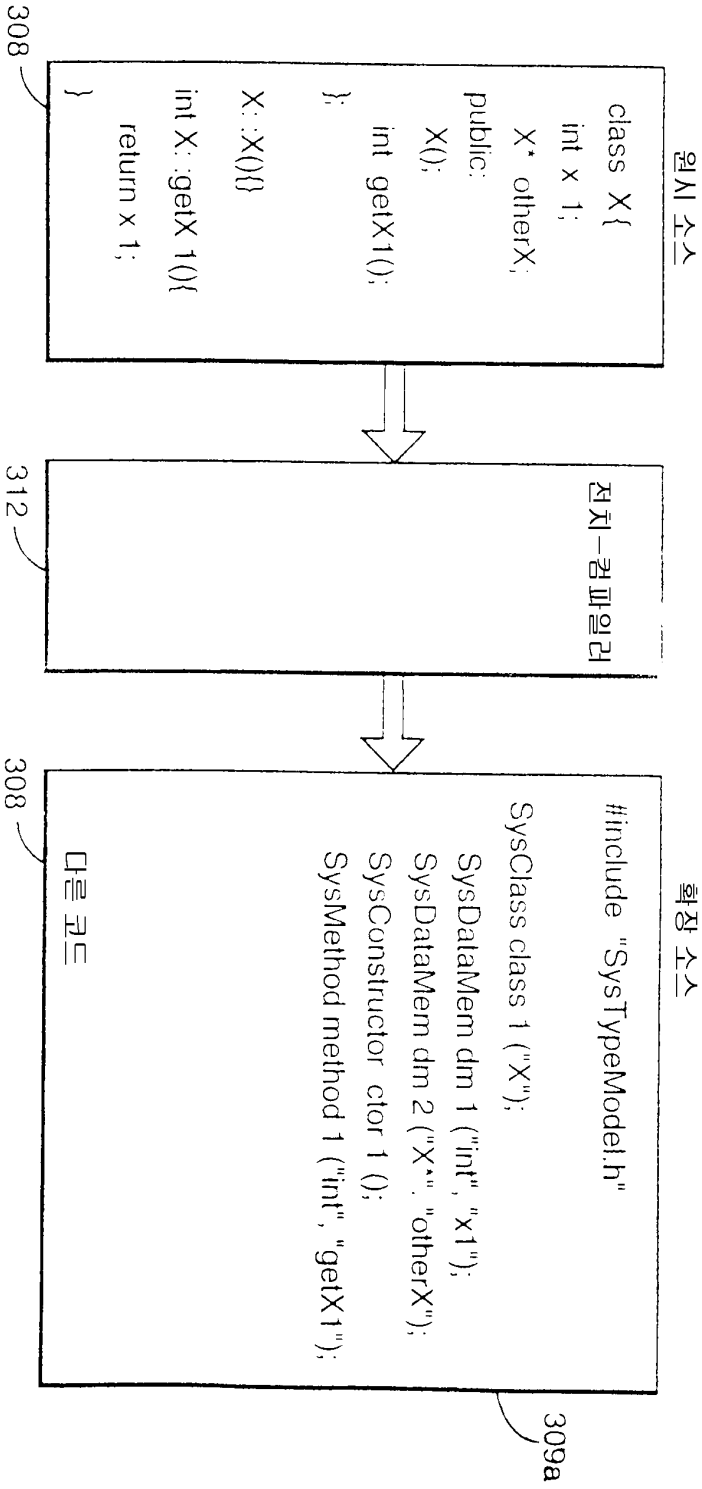
도면11a


시작
1010
클래스 검출
1012
새로운 클래스의 객체 생성
1014
데이터 타입 선언 검출
1016
그것을 위한 선언 생성
1018
데이터 타입이 더 있는가?
예
아니오
1020
구성기 선언 검출
1022
그것을 위한 구성기 선언 생성
1024
구성기가 더 있는가?
예
아니오
A
B

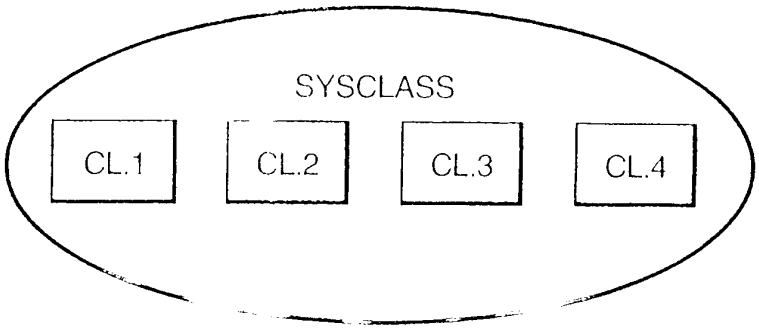
도면11b


A
B
1026
함수 선언 검출
1028
그것을 위한
선언 생성
1030
함수가
더 있는가?
예
아니오
1032
클래스가
더 있는가?
예
아니오
끝

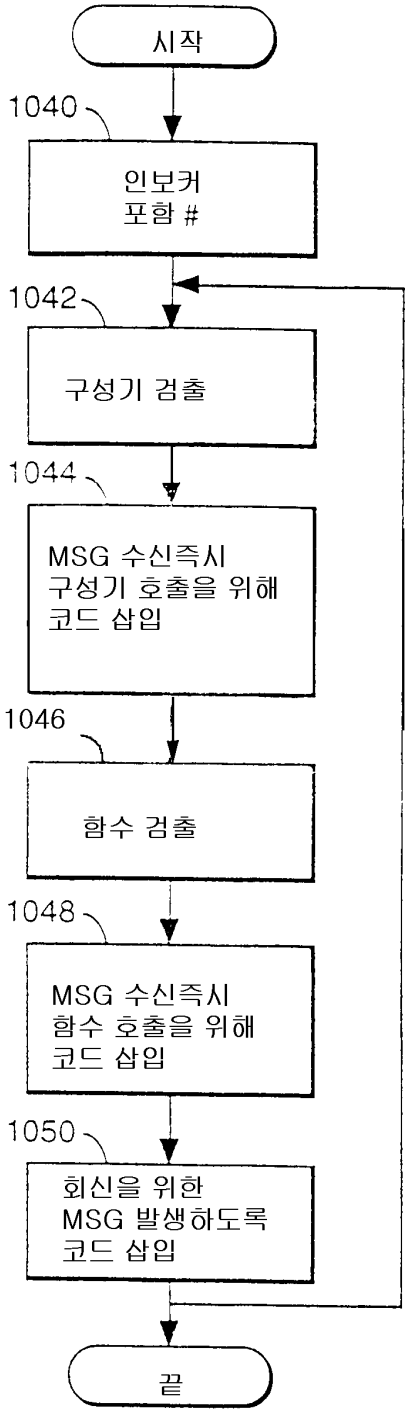
도면12


원시 소스
class X {
int x 1;
X* otherX;
public:
X();
int getX1();
};
X::X(){}
int X::getX 1(){
return x 1;
}
308
전치-컴파일러
312
확장 소스
#include "SysTypeModel.h"
SysClass class 1 ("X");
SysDataMem dm 1 ("int", "x1");
SysDataMem dm 2 ("X*", "otherX");
SysConstructor ctor 1 ();
SysMethod method 1 ("int", "getX1");
다른 코드
308
309a

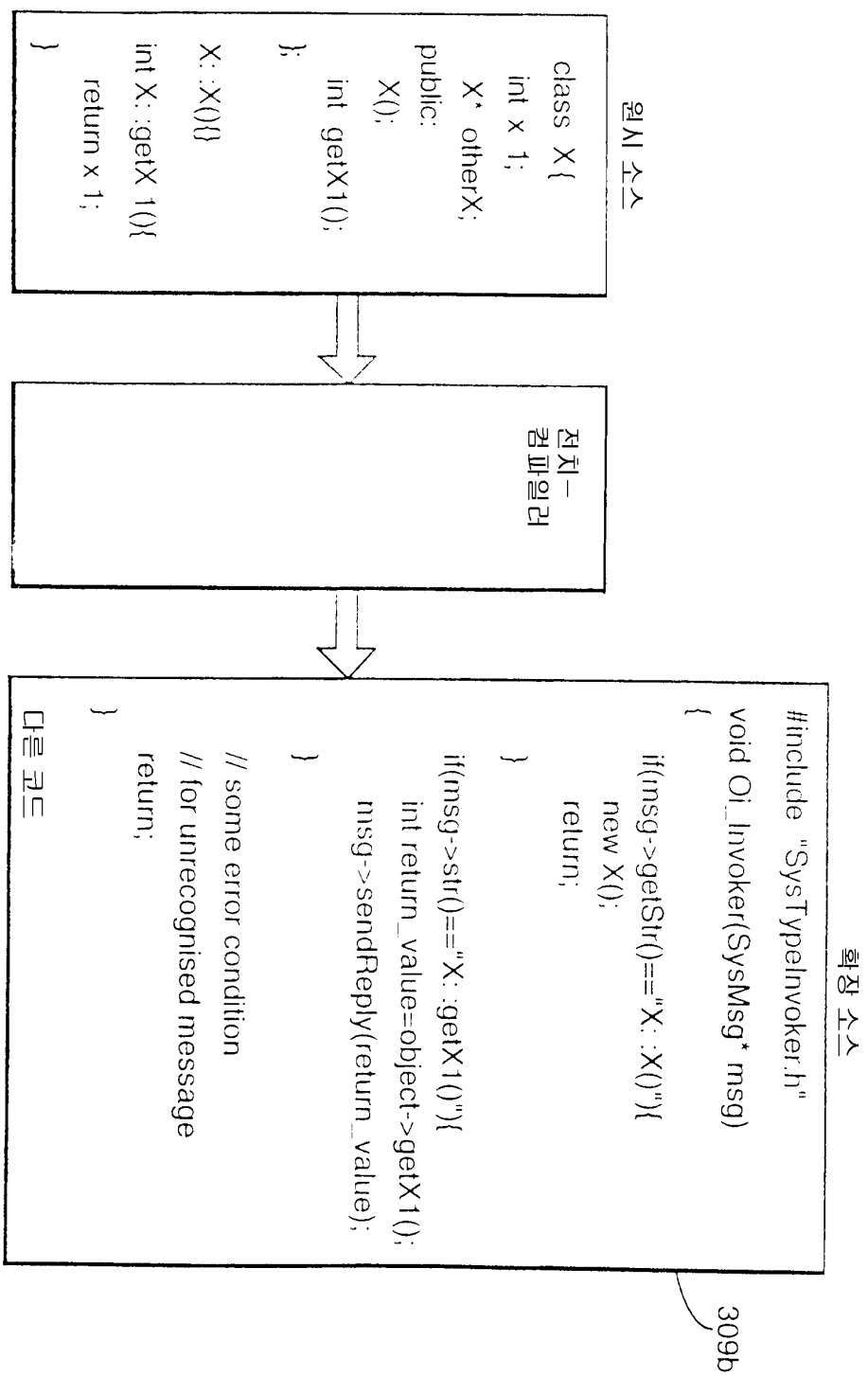
도면13



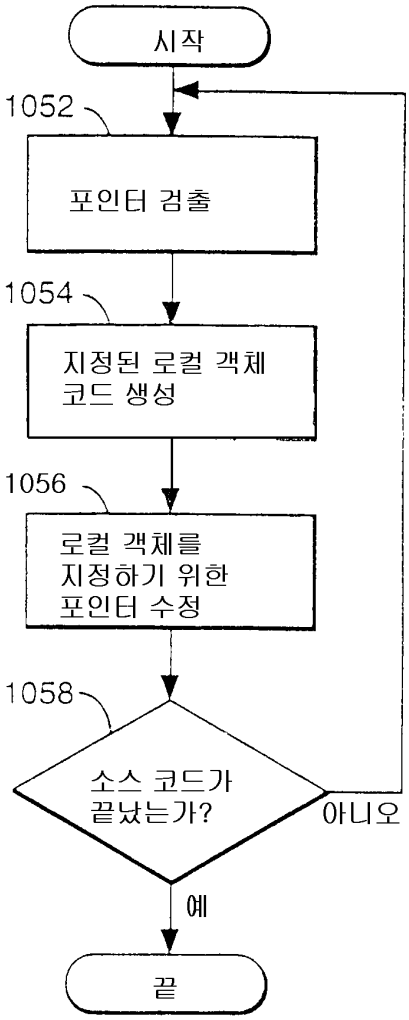
도면14


시작
1040
인보커
포함 #
1042
구성기 검출
1044
MSG 수신즉시
구성기 호출을 위해
코드 삽입
1046
함수 검출
1048
MSG 수신즉시
함수 호출을 위해
코드 삽입
1050
회신을 위한
MSG 발생하도록
코드 삽입
끝

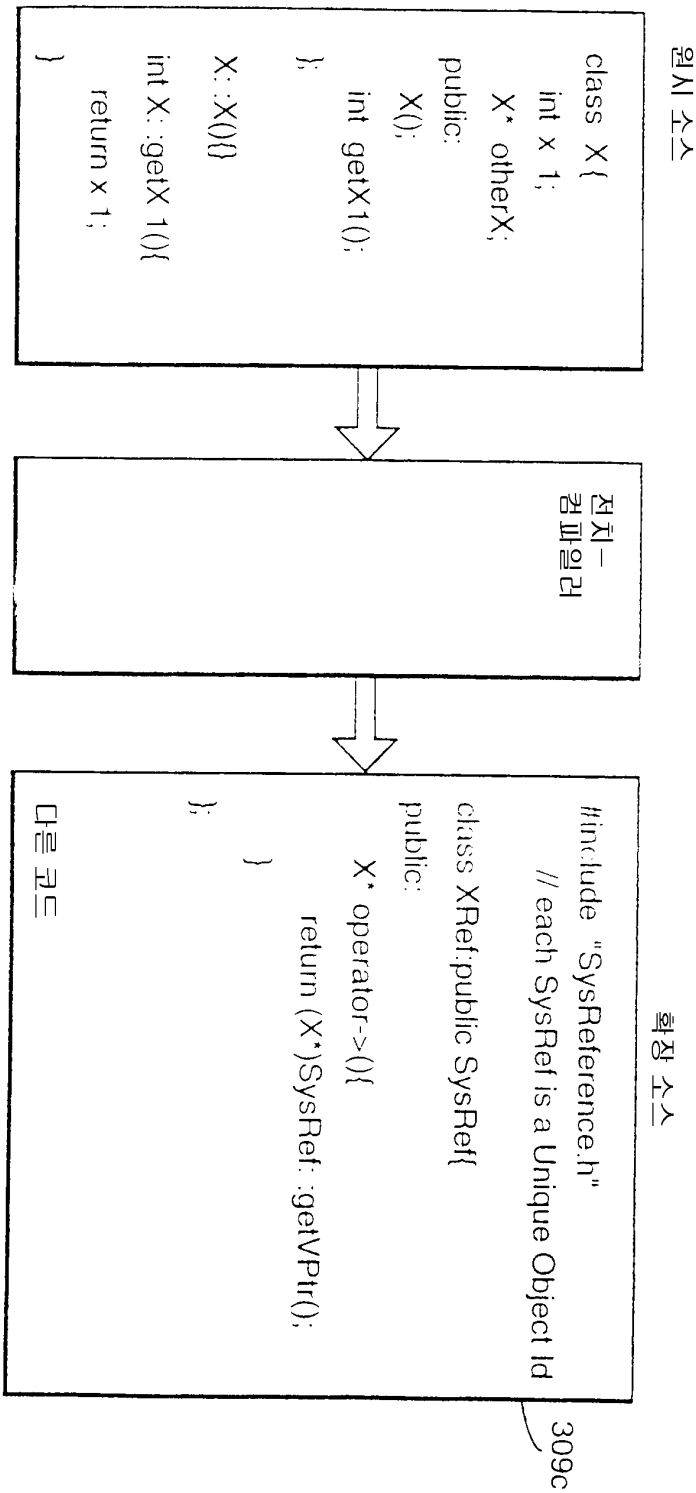
도면15


원시 소스
class X {
int x 1;
X* otherX;
public:
X();
int getX1();
};
X::X(){}
int X::getX 1(){
return x 1;
}
전처리
컴파일러
확장 소스
#include "SysTypeInvoker.h"
void Oi_Invoker(SysMsg* msg)
{
if(msg->getStr()=="X::X()"){
new X();
return;
}
if(msg->str()=="X::getX1()"){
int return_value=object->getX1();
msg->sendReply(return_value);
}
// some error condition
// for unrecognised message
return;
}
다른 코드
309b

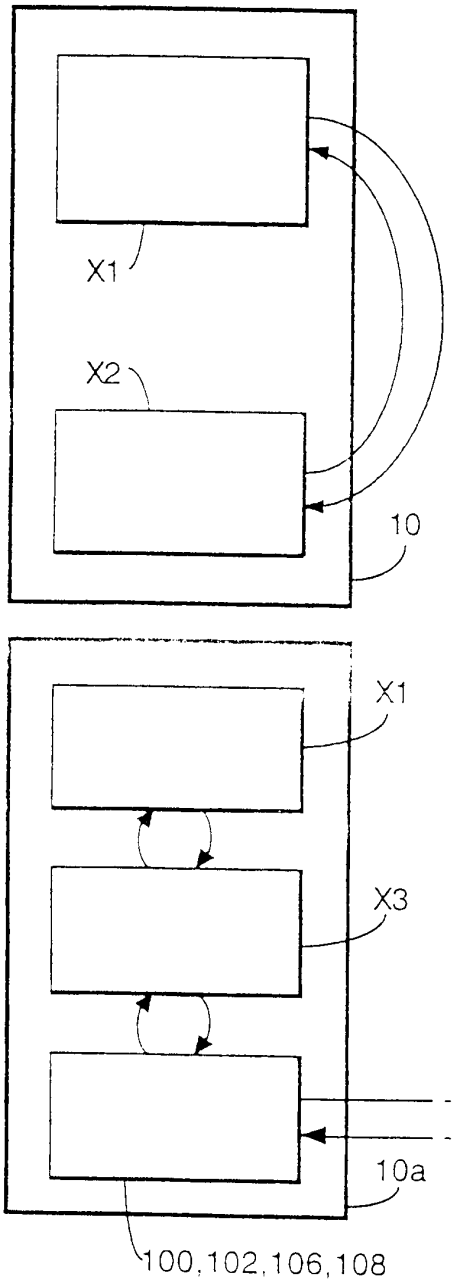
도면16


시작
1052
포인터 검출
1054
지정된 로컬 객체 코드 생성
1056
로컬 객체를 지정하기 위한 포인터 수정
1058
소스 코드가 끝났는가?
아니오
예
끝

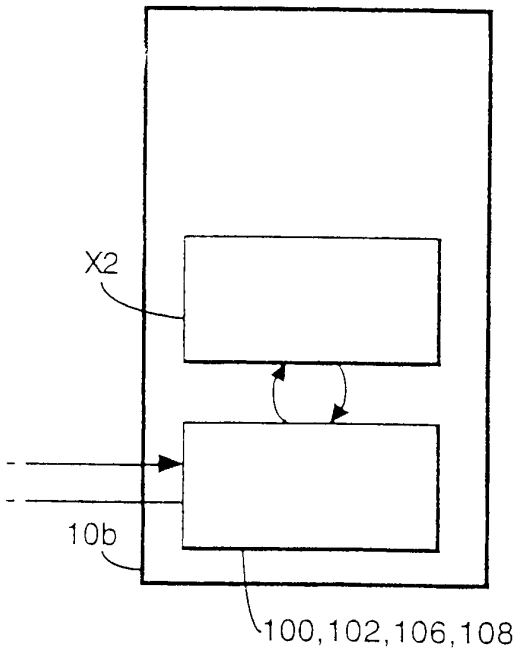
도면17


원시 소스
class X {
int x 1;
X* otherX;
public:
X();
int getX1();
};
X: :X(){}
int X: :getX 1(){
return x 1;
}
전처-
컴파일러
확장 소스
#include "SysReference.h"
// each SysRef is a Unique Object Id
class XRef:public SysRef{
public:
X* operator->(){
return (X*)SysRef: :getVPtr();
}
};
다른 코드
309c

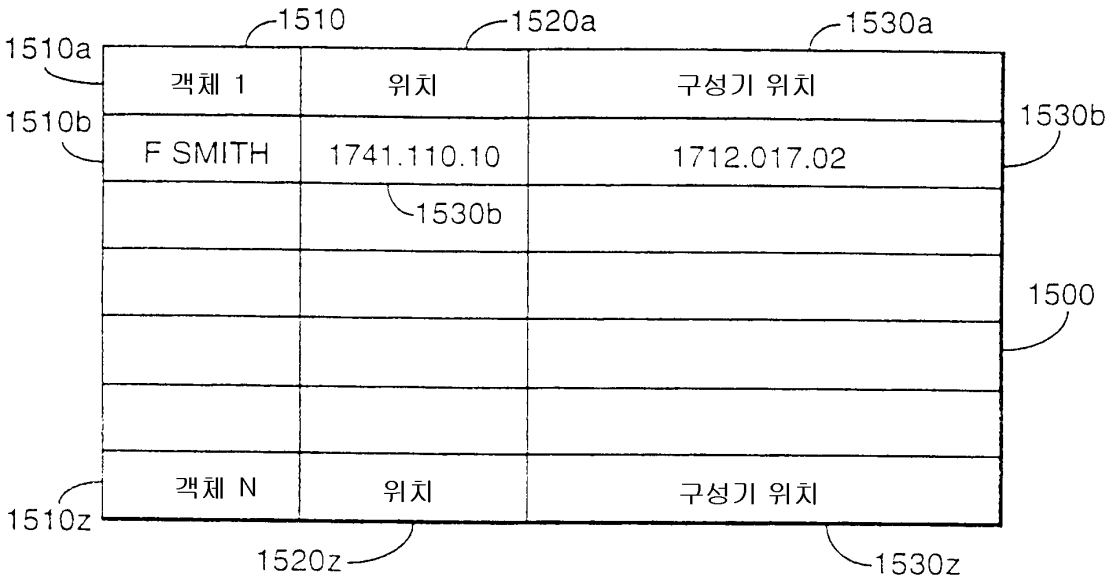
도면18a



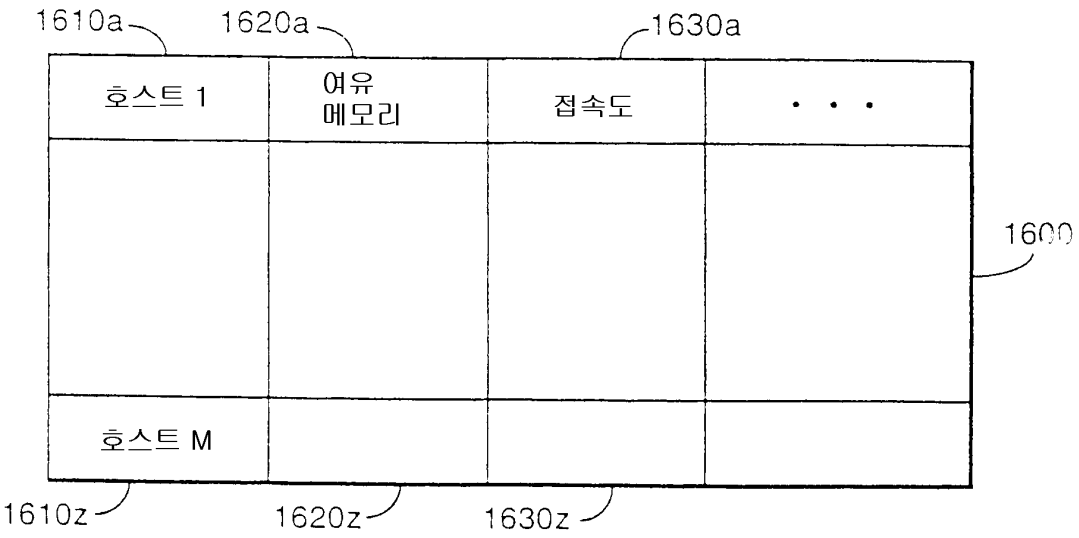
도면18b


X2
10b
100,102,106,108

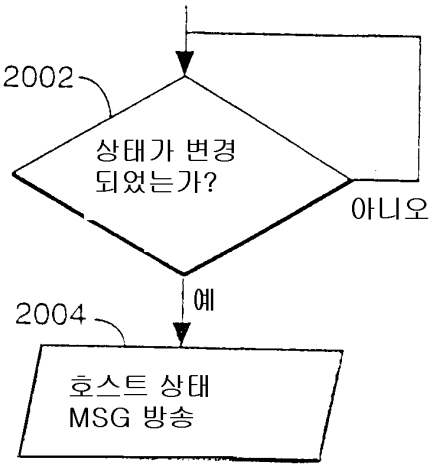
도면19


1510
1520a
1530a
1510a
객체 1
위치
구성기 위치
1510b
F SMITH
1741.110.10
1712.017.02
1530b
1530b
1500
객체 N
위치
구성기 위치
1510z
1520z
1530z

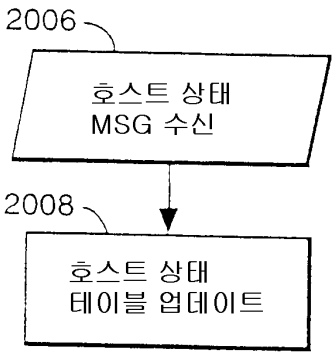
도면20



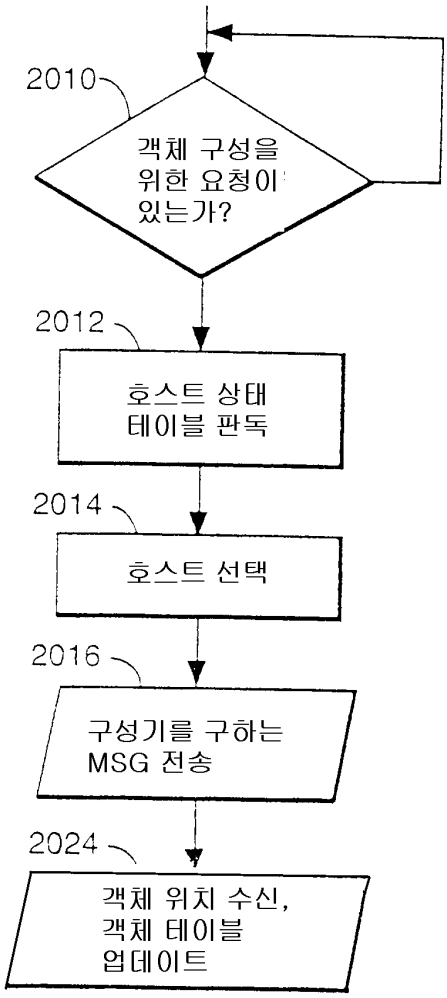
도면21a



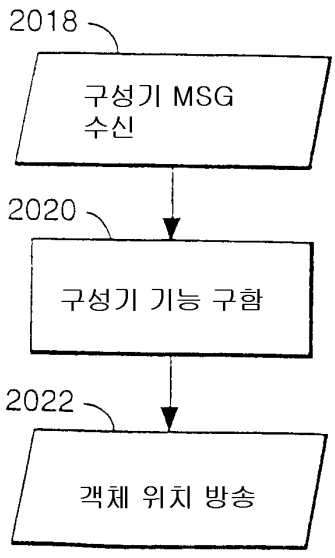
도면21b



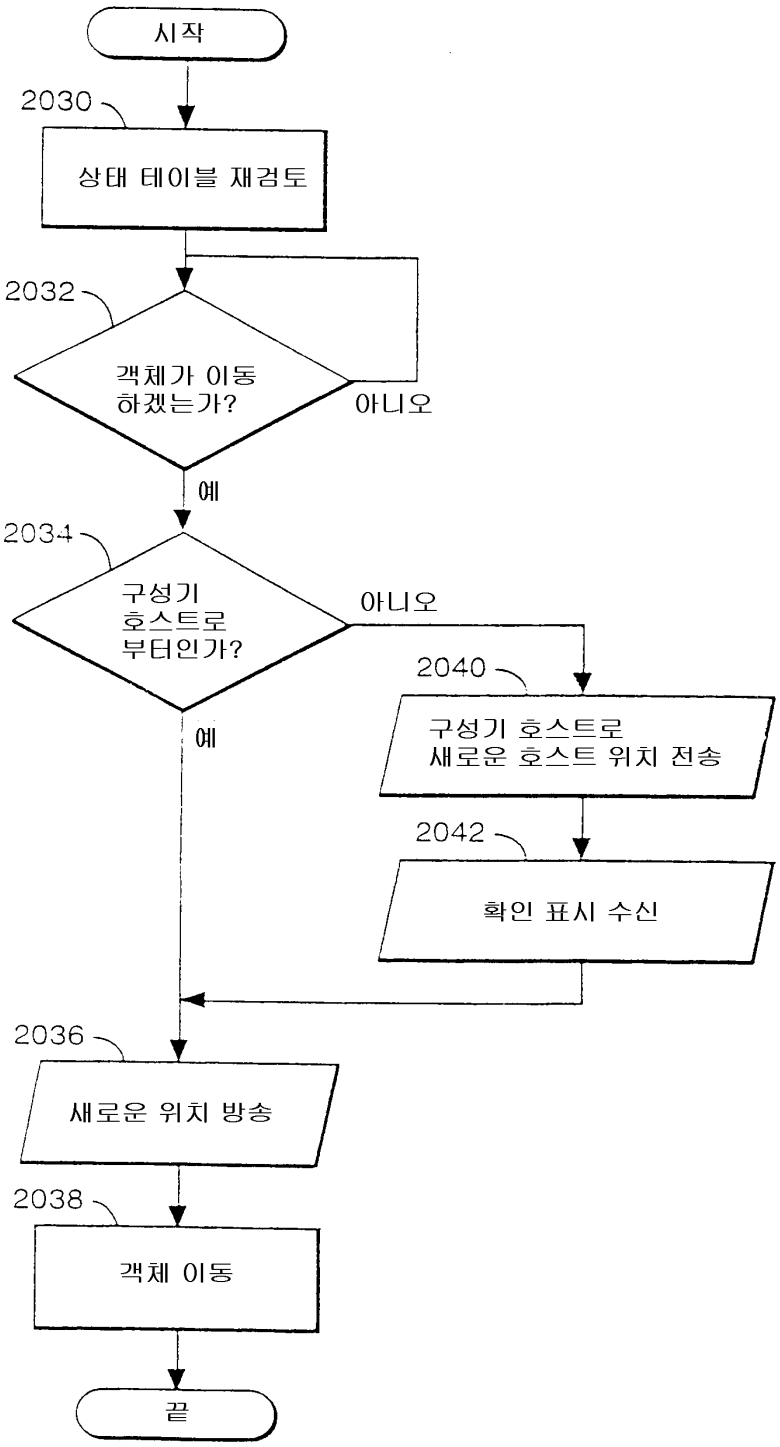
도면22a


2010
객체 구성을
위한 요청이
있는가?
2012
호스트 상태
테이블 판독
2014
호스트 선택
2016
구성기를 구하는
MSG 전송
2024
객체 위치 수신,
객체 테이블
업데이트

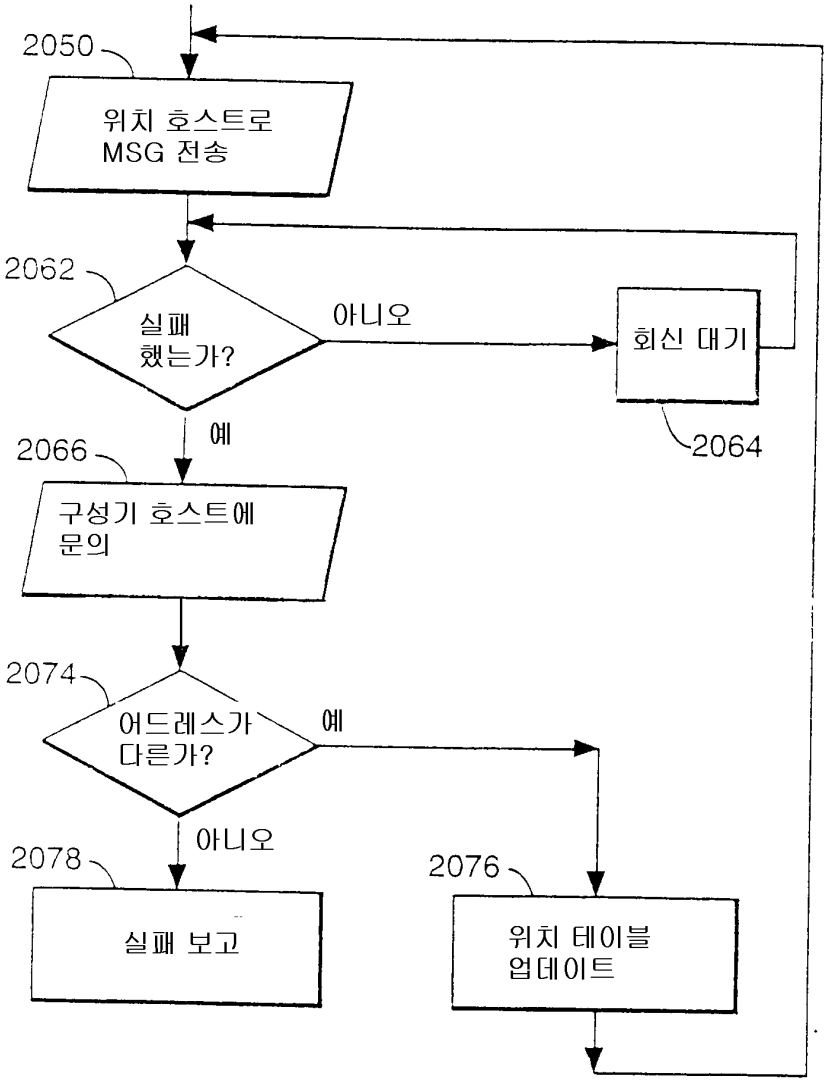
도면22b


2018
구성기 MSG
수신
2020
구성기 기능 구함
2022
객체 위치 방송

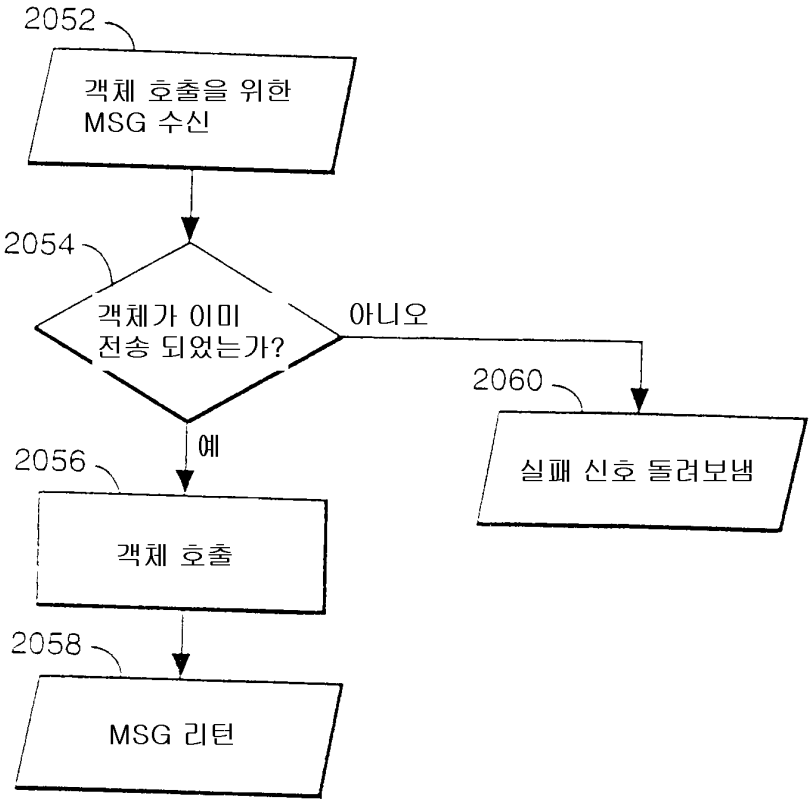
도면23


시작
2030
상태 테이블 재검토
2032
객체가 이동
하겠는가?
아니오
예
2034
구성기
호스트로
부터인가?
아니오
예
2040
구성기 호스트로
새로운 호스트 위치 전송
2042
확인 표시 수신
2036
새로운 위치 방송
2038
객체 이동
끝

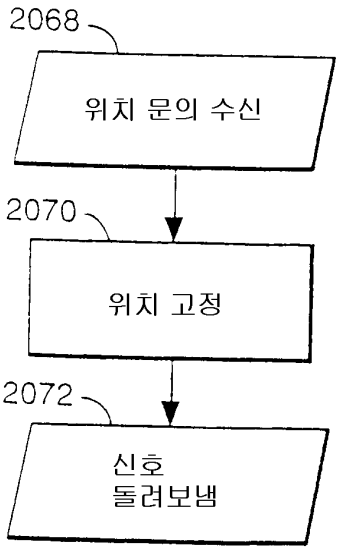
도면24a


2050
위치 호스트로
MSG 전송
2062
실패
했는가?
아니오
회신 대기
2064
예
2066
구성기 호스트에
문의
2074
어드레스가
다른가?
예
아니오
2078
실패 보고
2076
위치 테이블
업데이트

도면24b


2052
객체 호출을 위한
MSG 수신
2054
객체가 이미
전송 되었는가?
아니오
2060
실패 신호 돌려보냄
예
2056
객체 호출
2058
MSG 리턴

도면24c


2068
위치 문의 수신
2070
위치 고정
2072
신호
돌려보냄