



(19) 대한민국특허청(KR)

(12) 등록특허공보(B1)

(45) 공고일자 2020년02월11일

(11) 등록번호 10-2065563

(24) 등록일자 2020년01월07일

(51) 국제특허분류(Int. Cl.)

G06F 12/02 (2018.01)

(52) CPC특허분류

G06F 12/0253 (2013.01)

G06F 12/0269 (2013.01)

(21) 출원번호 10-2015-7000473

(22) 출원일자(국제) 2013년07월30일

심사청구일자 2018년01월19일

(85) 번역문제출일자 2015년01월08일

(65) 공개번호 10-2015-0039742

(43) 공개일자 2015년04월13일

(86) 국제출원번호 PCT/US2013/052780

(87) 국제공개번호 WO 2014/022431

국제공개일자 2014년02월06일

(30) 우선권주장

61/677,390 2012년07월30일 미국(US)

13/782,433 2013년03월01일 미국(US)

(56) 선행기술조사문헌

US20030221047 A1*

(뒷면에 계속)

전체 청구항 수 : 총 20 항

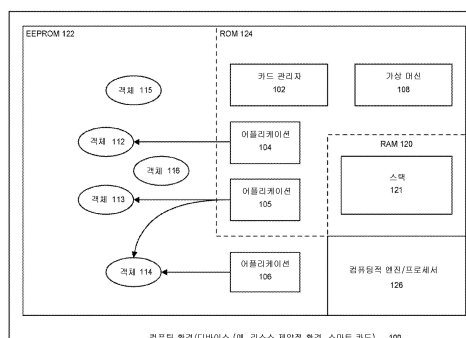
심사관 : 김진권

(54) 발명의 명칭 비트맵 윈도우들을 이용한 지속 메모리 내의 객체 삭제 시스템 및 방법

(57) 요약

메모리 청크들을 나타내는 비트맵 윈도우들을 이용한 지속 메모리 내의 객체 삭제를 위한 시스템 및 방법이 개시된다. 실시예에 따르면, 시스템은 일반적으로, 스마트 카드들, Java 카드들, 및 다른 리소스-제한적 환경들과 같은 지속 메모리를 사용하는 컴퓨팅 환경들(100)과 함께 사용될 수 있다. 실시예에 따르면, 시스템은 프로세서 또는 컴퓨팅적 엔진, 및 소프트웨어 객체들의 저장을 위한 지속 메모리와, 그리고 하나 이상의 메모리 비트맵 윈도우들(144)을 포함할 수 있는 데이터 구조(142)를 포함하며, 상기 메모리 비트맵 윈도우들 각각은 지속 메모리(140) 내의 어드레싱가능한 공간의 청크(146, 148, 150)를 나타내고, 상기 시스템은 지속 메모리로부터 비 도달 가능한 객체들을 삭제 시 하나 이상의 메모리 비트맵 윈도우들을 사용한다.

대표도



(52) CPC특허분류

G06F 2212/177 (2013.01)

G06F 2212/202 (2013.01)

(56) 선행기술조사문헌

US20070174369 A1*

US20090248766 A1*

KR1020050043291 A*

US06453403 B1*

US20080162610 A1*

*는 심사관에 의하여 인용된 문헌

명세서

청구범위

청구항 1

메모리 청크들을 나타내는 비트맵 윈도우들을 이용한 지속 메모리(persistent memory) 내의 객체 삭제(object deletion)를 위한 시스템으로서,

프로세서 또는 컴퓨팅적 엔진(computational engine) 및 복수의 메모리 청크들로 분할된 지속 메모리를 포함하는 컴퓨팅 환경 또는 디바이스 -복수의 메모리 청크들 각각은 하나 이상의 루트 객체 및 그와 연관된 하나 이상의 차일드(child) 객체들을 저장함 -;

상기한 메모리 청크를 각각 나타내는 복수의 메모리 비트맵 윈도우들을 포함하는 데이터 구조를 포함하며,

상기 시스템은 동작들을 수행함으로써 상기 지속 메모리로부터의 삭제를 위해 비 도달가능한(non-reachable) 객체들을 마킹하도록 상기 데이터 구조를 사용하며, 상기 동작들은,

특정 메모리 청크에서 참조되지 않은 루트 객체를 잠재적인 가비지 리스트(potential garbage list)에 배치하는 동작,

상기 루트 객체의 차일드 객체가 상기 특정 메모리 청크와는 상이한 메모리 청크에 존재하는 것으로 결정하는 동작 그리고 제1 동작 및 제2 동작 중 하나를 수행하는 동작 -상기 제1 동작은 상기 차일드 객체가 체크된 이전 메모리 청크에 있으면 상기 잠재적인 가비지 리스트에서 상기 루트 객체를 삭제하는 동작이며, 제2 동작은 상기 차일드 객체가 체크되지 않은 다음 메모리 청크에 있으면 상기 루트 객체를 시작점으로 사용하여 삭제를 위한 객체들에 대한 상기 다음 메모리 청크를 체크하는 동작임 -,

각 메모리 청크가 체크될 때까지 프로세스를 반복하는 동작,

해당 비트맵 윈도우에서 해당 비트를 플래그함으로써 삭제를 위해 잠재적인 가비지 리스트 내의 각 객체를 마킹하는 동작을 포함하는 것을 특징으로 하는 시스템.

청구항 2

제1항에 있어서,

상기 데이터 구조는:

최대 재귀 깊이(maximum recursion depth); 및

연기된 객체 리스트(deferred object list) 중 하나 이상을 포함하는 것을 특징으로 하는 시스템.

청구항 3

제2항에 있어서,

상기 최대 재귀 깊이는 마킹 단계(marking phase)에 대한 재귀 깊이를 제한하는 구성가능한 값이고, 그리고

상기 연기된 객체 리스트는, 일단 구성된 재귀 깊이에 도달되면 아직 프로세스되지 않은 칠드런(children) 객체들을 포함하는 것을 특징으로 하는 시스템.

청구항 4

제3항에 있어서,

상기 해당 비트는 메모리 청크 내의 어드레스를 나타내는 것을 특징으로 하는 시스템.

청구항 5

메모리 청크들을 나타내는 비트맵 윈도우들을 이용한 지속 메모리 내의 객체 삭제 방법으로서,

프로세서 또는 컴퓨팅적 엔진(computational engine) 및 복수의 메모리 청크들로 분할된 지속 메모리를 포함하

는 컴퓨팅 환경 또는 디바이스에서, 상이한 메모리 청크를 각각 나타내는 복수의 메모리 비트맵 윈도우들을 포함하는 데이터 구조를 제공하는 단계 -복수의 메모리 청크들 각각은 하나 이상의 루트 객체들 및 그와 연관된 하나 이상의 차일드(child) 객체들을 저장함 -;

특정 메모리 청크에서 참조되지 않은 루트 객체를 잠재적인 가비지 리스트(potential garbage list)에 배치하는 단계;

상기 루트 객체의 차일드 객체가 상기 특정 메모리 청크와는 상이한 메모리 청크에 존재하는 것으로 결정하고, 그리고 제1 단계 및 제2 단계 중 하나를 수행하는 단계 -상기 제1 단계는 상기 차일드 객체가 체크된 이전 메모리 청크에 있으면 상기 잠재적인 가비지 리스트에서 상기 루트 객체를 삭제하는 단계이며, 제2 단계는 상기 차일드 객체가 체크되지 않은 다음 메모리 청크에 있으면 상기 루트 객체를 시작점으로 사용하여 삭제를 위한 객체들에 대한 상기 다음 메모리 청크를 체크하는 단계임 -;

각 메모리 청크가 체크될 때까지 프로세스를 반복하는 단계;

해당 비트맵 윈도우에서 해당 비트를 플래그함으로써 삭제를 위해 잠재적인 가비지 리스트 내의 각 객체를 마킹하는 단계를 포함하는 것을 특징으로 하는 지속 메모리 내의 객체 삭제 방법.

청구항 6

제5항에 있어서, 상기 데이터 구조는,

최대 재귀 깊이; 및

연기된 객체 리스트 중 하나 이상을 더 포함하는 것을 특징으로 하는 지속 메모리 내의 객체 삭제 방법.

청구항 7

제6항에 있어서,

상기 최대 재귀 깊이는 마킹 단계에 대한 재귀 깊이를 제한하는 구성가능한 값이고, 그리고

상기 연기된 객체 리스트는, 일단 구성된 재귀 깊이에 도달되면 아직 프로세스되지 않은 칠드런 객체들을 포함하는 것을 특징으로 하는 지속 메모리 내의 객체 삭제 방법.

청구항 8

제7항에 있어서,

상기 해당 비트는 메모리 청크 내의 어드레스를 나타내는 것을 특징으로 하는 지속 메모리 내의 객체 삭제 방법.

청구항 9

삭제

청구항 10

삭제

청구항 11

명령어들이 저장된 비일시적 컴퓨터 판독가능 저장 매체로서, 상기 명령어들은 컴퓨팅 환경 또는 디바이스에 의해 판독 및 실행될 때, 상기 컴퓨팅 환경 또는 디바이스로 하여금 단계들을 수행하게 하며, 상기 단계들은:

프로세서 또는 컴퓨팅적 엔진(computational engine) 및 복수의 메모리 청크들로 분할된 지속 메모리를 포함하는 컴퓨팅 환경 또는 디바이스에서, 상이한 메모리 청크를 각각 나타내는 복수의 메모리 비트맵 윈도우들을 포함하는 데이터 구조를 제공하는 단계 -복수의 메모리 청크들 각각은 하나 이상의 루트 객체들 및 그와 연관된 하나 이상의 차일드(child) 객체들을 저장함 -;

특정 메모리 청크에서 참조되지 않은 루트 객체를 잠재적인 가비지 리스트(potential garbage list)에 배치하는 단계;

상기 루트 객체의 차일드 객체가 상기 특정 메모리 청크와는 상이한 메모리 청크에 존재하는 것으로 결정하고, 그리고 제1 단계 및 제2 단계 중 하나를 수행하는 단계 -상기 제1 단계는 상기 차일드 객체가 체크된 이전 메모리 청크에 있으면 상기 잠재적인 가비지 리스트에서 상기 루트 객체를 삭제하는 단계이며, 제2 단계는 상기 차일드 객체가 체크되지 않은 다음 메모리 청크에 있으면 상기 루트 객체를 시작점으로 사용하여 삭제를 위한 객체들에 대한 상기 다음 메모리 청크를 체크하는 단계임 -;

각 메모리 청크가 체크될 때까지 프로세스를 반복하는 단계;

해당 비트맵 윈도우에서 해당 비트를 플래그함으로써 삭제를 위해 잠재적인 가비지 리스트 내의 각 객체를 마킹하는 단계를 포함하는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

청구항 12

제11항에 있어서, 상기 데이터 구조는,

최대 재귀 깊이; 및

연기된 객체 리스트 중 하나 이상을 포함하는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

청구항 13

제12항에 있어서,

상기 최대 재귀 깊이는 마킹 단계에 대한 재귀 깊이를 제한하는 구성가능한 값이고, 그리고

상기 연기된 객체 리스트는, 일단 구성된 재귀 깊이에 도달되면 아직 프로세스되지 않은 칠드런 객체들을 포함하는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

청구항 14

제13항에 있어서,

상기 해당 비트는 메모리 청크 내의 어드레스를 나타내는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

청구항 15

제1항에 있어서, 각각의 메모리 청크 내의 루트 객체들 및 차일드 객체들은 재귀적으로 마킹되는 것을 특징으로 하는 시스템.

청구항 16

제1항에 있어서, 상기 마킹하는 동작이 메모리 청크에서 완료될 때, 상기 메모리 청크 내의 임의의 참조되지 않은 객체는 상기 잠재적인 가비지 리스트 상에 배치되는 것을 특징으로 하는 시스템.

청구항 17

제1항에 있어서, 각각의 메모리 청크는, 상기 잠재적인 가비지 리스트가 궁극적으로, 참조되지 않고 가비지로 고려될 수 있는 이 객체들을 포함하도록 프로세싱되는 것을 특징으로 하는 시스템.

청구항 18

제5항에 있어서, 각각의 메모리 청크 내의 루트 객체들 및 차일드 객체들은 재귀적으로 마킹되는 것을 특징으로 하는 지속 메모리 내의 객체 삭제 방법.

청구항 19

제5항에 있어서, 상기 마킹하는 단계가 메모리 청크에서 완료될 때, 상기 메모리 청크 내의 임의의 참조되지 않은 객체는 상기 잠재적인 가비지 리스트 상에 배치되는 것을 특징으로 하는 지속 메모리 내의 객체 삭제 방법.

청구항 20

제5항에 있어서, 각각의 메모리 청크는, 상기 잠재적인 가비지 리스트가 궁극적으로, 참조되지 않고 가비지로

고려될 수 있는 이 객체들을 포함하도록 프로세싱되는 것을 특징으로 하는 지속 메모리 내의 객체 삭제 방법.

청구항 21

제11항에 있어서, 각각의 메모리 청크 내의 루트 객체들 및 차일드 객체들은 재귀적으로 마킹되는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

청구항 22

제11항에 있어서, 상기 마킹하는 단계가 메모리 청크에서 완료될 때, 상기 메모리 청크 내의 임의의 참조되지 않은 객체는 상기 잠재적인 가비지 리스트 상에 배치되는 것을 특징으로 하는 비일시적 컴퓨터 판독가능 저장 매체.

발명의 설명

기술 분야

[0001] 저작권 공지

[0002] 본 명세서에서 개시된 부분은 저작권 보호를 받는 내용을 포함한다. 저작권자는 미국특허상표청의 특허 파일 또는 기록에 나타난 대로 본 특허 문서 또는 특허 개시내용을 어느 누군가가 복사하여 재생하는 것은 반대하지 않지만, 그 밖의 모든 것은 저작권으로 보호된다.

[0003] 기술분야

[0004] 본 발명의 실시예들은 일반적으로, 지속(persistent) 메모리 스토리지에 관한 것이며, 특히 리소스-제약적 환경(resource-constrained environment)들 및 다른 컴퓨팅 환경들, 그리고 메모리 청크(chunk)들을 표시하는 비트맵 윈도우들을 이용하여 지속 메모리로부터 비-도달가능(non-reachable) 객체들을 삭제하는 시스템들 및 방법들에 관한 것이다.

배경 기술

[0005] 스마트 카드 또는 유사한 실행 환경 또는 디바이스와 같은 전형적인 리소스-제약적 환경은 제한된 랜덤 액세스 메모리 리소스들을 가질 수 있으며, 대신 소프트웨어 객체들을 생성 및 저장하기 위한 디폴트 공간으로서 지속 메모리를 사용한다.

[0006] 이러한 환경들은 정의된 루트(root)들의 세트로부터 비 도달가능함에 의해 가비지(garbage)가 된 객체들에 의해 사용중인 지속 메모리를 재활용(reclaiming)할 필요성을 가진다. Java SE와 같은 다른 환경들에서 유용할 수 있는 가비지 수집 알고리즘은 종종, 객체들을 직접적으로 이동 또는 마킹하는 것과 같은 기법들에 의존한다. 그러나, 이 기법들은 리소스-제약적, 스마트 카드, 또는 유사한 실행 환경에서 유용하지 않은 바, 그 이유는 지속 메모리에서 객체들을 이동 또는 마킹하는 것은 시간 소모적이고, 지속 메모리 디바이스가 마모되기 전에 상기 지속 메모리 디바이스에 제한된 수의 데이터 기록들만이 이루어질 수 있음을 고려해 볼 때 잠재적으로는 손상을 주는 동작이기 때문이다.

발명의 내용

[0007] 메모리 청크들을 표시하는 비트맵 윈도우들을 이용한 지속 메모리 내의 객체 삭제를 위한 시스템 및 방법이 본 명세서에 개시된다. 실시예에 따르면, 상기 시스템은 일반적으로, 스마트 카드들, Java 카드들 및 다른 리소스-제약적 환경들과 같이 지속 메모리를 사용하는 컴퓨팅 환경들과 함께 이용될 수 있다. 실시예에 따르면, 상기 시스템은 프로세서 또는 컴퓨팅 엔진, 및 소프트웨어 객체들의 저장을 위한 지속 메모리와, 그리고 하나 이상의 메모리 비트맵 윈도우들을 포함할 수 있는 데이터 구조를 포함하며, 상기 메모리 비트맵 윈도우들의 각각은 지속 메모리 내의 어드레싱가능 공간의 청크를 나타내고, 상기 시스템은 지속 메모리로부터 비 도달가능 객체들을 삭제함에 있어서 상기 하나 이상의 메모리 비트맵 윈도우들을 이용한다.

도면의 간단한 설명

[0008] 도 1은 실시예에 따른 리소스 제약적 환경 또는 스마트 카드와 같이 지속 메모리를 사용하는 컴퓨팅 환경의 예를 도시한다.

도 2는 실시예에 따른 지속 메모리 내의 객체 삭제를 위한 메모리 청크(chunk)들을 표시하는 비트맵 윈도우들의 사용을 도시한다.

도 3은 실시예에 따른 지속 메모리 내의 객체 삭제를 위한 비트맵 윈도우들의 사용을 더 도시한다.

도 4는 실시예에 따른 지속 메모리 내의 객체 삭제를 위한 비트맵 윈도우들의 사용을 더 도시한다.

도 5는 실시예에 따른 지속 메모리 내의 객체 삭제를 위해 비트맵 윈도우들을 사용하는 알고리즘 또는 방법을 도시하는 순서도이다.

도 6은 실시예에 따른 지속 메모리 내의 객체 삭제를 위해 비트맵 윈도우들을 사용하는 알고리즘 또는 방법을 더 도시하는 순서도이다.

도 7은 실시예에 따른 지속 메모리 내의 객체 삭제를 위해 비트맵 윈도우들을 사용하는 알고리즘 또는 방법을 더 도시하는 순서도이다.

발명을 실시하기 위한 구체적인 내용

- [0009] 상기 기술된 바와 같이, 스마트 카드 또는 유사한 실행 환경 또는 디바이스와 같은 전형적인 리소스 제약적 환경은 제한된 랜덤 액세스 메모리 리소스들을 가질 수 있고, 대신 소프트웨어 객체들을 생성 및 저장하기 위한 디폴트 위치(location)로서 지속 메모리를 사용한다. 이러한 환경들은 정의된 루트들의 세트로부터 비 도달가능함에 의해 가비지가 된 객체들에 의해 사용 중인 지속 메모리를 재활용할 필요성을 가진다. 그러나, 다른 환경에서 유용할 수 있지만 객체들을 직접적으로 이동시키거나 마킹하는 것과 같은 기법들에 의존하는 가비지 수집 알고리즘들은 리소스 제약적, 스마트 카드, 또는 유사한 실행 환경에서는 유용하지 않은 바, 그 이유는 지속 메모리 내의 객체들을 이동시키거나 마킹하는 것은 시간 소모적이고, 지속 메모리 디바이스가 마모되기 전에 상기 지속 메모리 디바이스에 제한된 수의 데이터 기록들만이 이루어질 수 있음을 고려해 볼 때 잠재적으로는 손상을 주는 동작이다.
- [0010] 이를 해소하기 위해 사용될 수 있는 접근법의 타입들의 일부 예들은, 각각의 객체가 시스템 내의 어떤 다른 객체로부터 참조되는지 여부를 결정하기 위하여 각각의 객체를 체크하기 위해 브루트 포스 알고리즘(brute force algorithm)을 사용하는 것, 0의 참조 카운트가 객체가 다른 객체에 의해 참조되지 않음을 나타내도록 하는 참조 카운트로 객체들을 갱신하는 참조 카운팅 알고리즘을 사용하는 것, 또는 메모리 내의 객체들을 표시하는 RAM 비트맵을 사용하는 것이다.
- [0011] 이 접근법들 중 처음 두 가지의 문제점은 이들이 심지어 간단한 사이클들에 대해서조차 사이클적 참조들을 고려하지 않는다는 것이다. 첫번째 접근법은 또한 시간 소모적인 바, 그 이유는 최악의 경우의 시나리오에서 잠재적으로 시스템 내의 모든 객체가 일 객체로부터 다른 객체로의 참조들을 결정하기 위해 시스템 내의 다른 모든 객체에 대해 체크되어야 할 것이기 때문이다. 세번째 접근법의 문제점은 이 접근법이 시스템이 실제 어드레스가 아닌 객체 테이블로부터의 객체 참조들을 사용하도록 객체들을 역참조(dereferencing)하는 것에 의존하지만, 많은 동작들이 객체 어드레스들을 요할 것이며 참조로부터 어드레스를 얻기 위해 변환이나 프로세스를 격어야 할 것이기 때문에 이는 전체 실행 환경을 느려지게 한다는 점이다.
- [0012] 실시예에 따르면, 메모리 청크들을 표시하는 비트맵 윈도우들을 사용하는 지속 메모리 내의 객체 삭제를 위한 시스템 및 방법이 본 명세서에 기술된다. 실시예에 따르면, 시스템은 일반적으로, 스마트 카드들, 자바 카드들, 및 다른 리소스 제약적 환경들과 같은 지속 메모리를 사용하는 컴퓨팅 환경들과 함께 사용될 수 있다. 실시예에 따르면, 시스템은 프로세서 또는 컴퓨팅적 엔진(computational engine) 및 소프트웨어 객체들의 저장을 위한 지속 메모리, 그리고 데이터 구조를 포함하며, 상기 데이터 구조는 하나 이상의 메모리 비트맵 윈도우들을 포함할 수 있고, 메모리 비트맵 윈도우들 각각은 지속 메모리 내의 어드레스가능한 공간의 청크를 표시하며, 시스템은 지속 메모리로부터 비 도달가능한 객체들을 삭제할 시 하나 이상의 메모리 비트맵 윈도우들을 사용한다.
- [0013] 다양한 실시예들에 따르면, 본 명세서에 기술된 시스템들 및 방법들의 일부 장점들은 이들이 참조 레퍼런싱 문제를 해소하는 마크-스weep 가비지 수집 기법(mark-sweep garbage collection technique)의 고속 버전을 제공할 수 있다는 점, 객체들을 역참조할 필요성이 없다는 점, 메모리의 실제 재활용까지 지속 메모리에 기록할 필요성이 역시 없다는 점, 그리고 접근법이 전체 타겟 메모리 영역의 증가가 리소스 요건들을 증가시키지 않도록 스케일러블하다는 점을 포함한다.
- [0014] 도 1은 실시예에 따른 리소스 제약적 환경 또는 스마트 카드와 같은 지속 메모리를 사용하는 컴퓨팅 환경의 예

를 도시한다.

- [0015] 도 1에 도시된 바와 같이, 실시예에 따르면, 리소스 제약적 환경, 스마트 카드 또는 다른 환경과 같은 컴퓨팅 환경 또는 디바이스(100)는 미니어처(miniature) 컴퓨팅 디바이스로 고려될 수 있고, 상기 미니어처 컴퓨팅 디바이스는 컴퓨팅적 동작들을 수행하기 위한 회로망을 포함할 수 있는 프로세서 또는 컴퓨팅적 엔진(126), 및 랜덤 액세스 메모리(RAM)(120), 전기적으로 소거가능하고 프로그램가능한 판독 전용 메모리(EEPROM)(122), 및/또는 판독 전용 메모리(ROM)(124)와 같은 하나 이상의 메모리를 포함한다.
- [0016] 실시예에 따르면, RAM은 휘발성 랜덤 액세스 메모리를 포함할 수 있고, EEPROM은 기록가능 비휘발성 메모리를 포함할 수 있고, ROM은 판독 전용 메모리를 포함할 수 있다. RAM은 메모리 스택(121)과 같은 데이터 아이템들 및 데이터 구조들을 저장하기 위해 사용될 수 있다. ROM은, Java와 같은 플랫폼-독립적 프로그래밍 언어로 기록된 어플리케이션들이 스마트 카드에 의해 실행되게 하는 Java 가상 머신과 같은 가상 머신(108)을 포함할 수 있고, 또한 스마트 카드에 액세스하는 클라이언트들을 위해 서비스들을 제공하는 하나 이상의 어플리케이션들(104, 105)을 포함할 수 있다. 실시예에 따르면, 어플리케이션들(106)은 또한, EEPROM에 또는 ROM 및/또는 EEPROM 둘 모두에 위치될 수 있다.
- [0017] 실시예에 따르면, 시스템은 카드 관리자(102) 또는 유사한 로직을 포함할 수 있고, 상기 카드 관리자 또는 유사한 로직은, (예컨대, 클라이언트들과 어플리케이션들 간의 통신을 인에이블하는 것을 포함하여) 컴퓨팅 환경 또는 디바이스 상에서의 어플리케이션들의 실행을 관리하고 메모리로부터 객체들을 삭제하기 위한 코드를 포함한다. 예를 들어, 도 1에 도시된 바와 같이, 디바이스의 동작 동안 EEPROM은 어플리케이션들(104, 105)에 의해 액세스되는 복수의 객체들(112, 113, 114)을 포함할 수 있고(보다 구체적으로는, 어플리케이션(104)은 객체(112)에 액세스하고, 어플리케이션(105)은 객체들(113 및 114)에 액세스하며, 어플리케이션(106)은 객체(114)에 액세스한다), 링크되지 않은 다른 객체들(115, 116)은 더이상 어떤 어플리케이션에 의해 참조되지 않는다. 일반적으로, 메모리 공간을 자유화(free up)하기 위해 이 참조되지 않은 객체들을 삭제하는 것이 바람직하다.
- [0018] 도 1에 도시된 환경은 본 명세서에 기술된 시스템들 및 방법들을 이용할 수 있는 일 타입의 환경의 예의 방식으로 제공된다. 다양한 실시예들에 따르면, 본 발명은 리소스 제약적 환경들 또는 스마트 카드들, 또는 상기에 기술된 특별한 환경 또는 구성에 제한되지 않으며, 일반적으로 기록가능한 지속 또는 비휘발성 메모리에 객체를 저장할 수 있는 다른 타입의 컴퓨팅 환경, 디바이스 또는 시스템과 함께 사용될 수 있다.
- [0019] 실시예에 따르면, 메모리의 청크를 표시하는 비트맵 윈도우들의 사용을 위한 시스템 및 방법이 제공된다. 기법은 많은 양의 어드레싱가능한 공간을 매핑하는 동안 매우 제한된 리소스들로 작업을 할 수 있는 알고리즘 또는 방법을 사용한다. 기법은 또한, 증가된 지속 메모리 사이즈에 대해, 심지어 전체 RAM 사이즈가 상대적으로 변경되지 않은 채로 유지되더라도 알고리즘 또는 방법이 여전히 작업을 하게 되도록 스케일러블하다. 추가적으로, 상기 알고리즘 또는 방법은 브루트 포스, 참조 카운팅 및 역참조 기법들과 함께하는 경우에 있을 수 있는 전체 시스템 성능의 감소를 야기하지 않는다.
- [0020] 도 2는 실시예에 따른 지속 메모리 내의 객체 삭제를 위한 메모리 청크들을 표시하는 비트맵 윈도우들의 사용을 도시한다.
- [0021] 도 2에 도시된 바와 같이, 실시예에 따르면, 시스템은 데이터 구조(142)를 포함할 수 있고, 상기 데이터 구조는 하나 이상의 메모리 비트맵 윈도우들(144)(156, 158, 160)을 포함할 수 있고, 상기 비트맵 윈도우들은 객체들의 저장을 위한 지속 메모리(140) 내의 어드레싱가능 공간의 청크(146, 148, 150)를 표시한다. 데이터 구조는 이미 프로세스된 윈도우들 내의 참조되지 않은 객체들을 리스팅하는 잠재적인 가비지 리스트(162), 이전 윈도우로부터 참조되지만 이전 윈도우 내의 어드레싱가능 공간에 있지 않은 객체들 - 상기 객체들은 다음 윈도우에 대한 루트들이 됨 - 을 리스팅하는 동적 루트 리스트(164), 마킹 단계(marking phase)를 위한 재귀 깊이(recursion depth)를 제한하는 구성가능한 값인 최대 재귀 깊이(166), 및 일단 구성된 재귀 깊이에 도달되면 아직 프로세스되지 않은 칠드런 객체들(children objects)을 포함하는 연기된 객체 리스트(deferred object list)(168)을 더 포함할 수 있다.
- [0022] 도 3 및 4는 실시예에 따른 지속 메모리 내의 객체 삭제를 위한 비트맵 윈도우들의 사용을 더 도시한다.
- [0023] 실시예에 따르면, 마킹 프로세스 동안 EEPROM으로의 불필요한 기록들을 회피하기 위해, 시스템은 RAM에 비트맵을 유지할 수 있다. 일부 환경들, 예컨대, 전형적인 리소스 제약적 환경에서, RAM은 희소 자원(scarce resource)이고, 전형적인 양은 2k일 수 있으며, 이는 비트맵 저장을 위해 아주 작은 공간을 제공한다. 예로서, 512 바이트 비트맵은 4k 비트를 포함하며, 각각의 비트는 4 바이트를 나타내어서, 이 사이즈의 비트맵은 16k의

메모리를 나타내기 위해 사용될 수 있다.

- [0024] 실시예에 따르면, 객체 삭제 프로세스 동안, 시스템은 메모리에서 도달가능한 모든 객체들을 "마킹"하는 마킹 프로세스를 수행할 수 있다. 이는 도달가능한 객체들을 찾기 위해 "루트들"로부터(가령, 애플릿 테이블들 및 정적 변수들로부터) 참조들의 체인들을 따라감으로써(following) 달성될 수 있다. 예를 들어, 애플릿 객체는 다른 객체를 포인팅할 수 있고, 상기 다른 객체는 또다른 객체들을 포인팅하며, 이들 각각은 또한 다른 객체들을 포인팅한다. 유사하게, 정적 변수들은 객체들을 포인팅할 수 있고, 상기 객체들은 또다른 객체들을 포인팅한다.
- [0025] 실시예에 따르면, 마킹 단계 동안 윈도우의 루트 객체들로부터 시작하여 이의 객체 칠드런이 마킹된다. 차일드 객체(child object)가 윈도우 밖에 그리고 다음 윈도우에 상주하면, 그 객체는 다음 윈도우에 대한 루트로서 동적 루트 리스트 내로 삽입될 수 있다. 차일드 객체가 이 윈도우 내에 있으면, 이는 마킹될 수 있고, 시스템은 그 객체의 칠드런에 채귀할 수 있다. 차일드 객체가 이전 윈도우에 있으면, 시스템은 잠재적인 가비지 리스트를 체크하고 그 객체에 채귀하며, 그 다음, 잠재적인 가비지로부터 그 객체를 제거하는 바, 그 이유는 그 객체가 이 윈도우로부터 어드레싱가능하기 때문이다. 윈도우에 대한 마킹 단계 후, 시스템은 마킹되지 않은 객체들을 찾고, 잠재적인 가비지 리스트에 이들을 삽입한다. 이 객체들은 실제로, 가비지로 아직 고려되지 않을 수 있는 바, 그 이유는 이들이 다음 윈도우로부터 어드레싱가능할 수 있기 때문이다. 그 다음, 시스템은 다음 윈도우들을 프로세스한다. 궁극적으로, 윈도우들을 완료한 후, 잠재적인 가비지 리스트는, 참조되지 않고 가비지로 고려될 수 있는 이 객체들을 포함할 것이다.
- [0026] 도 3에 도시된 바와 같이, 실시예에 따르면, 메모리 공간은 윈도우들로 분할된다. 비트맵은 윈도우들에 대해 초기화되고, 비트는 메모리 내의 어드레스를 나타낸다. 예를 들어, 메모리 관리가 4-바이트 정렬(alignment)로 수행되면, 비트는 4-바이트 메모리 셀 내의 제1 바이트를 나타낼 수 있다. 그 다음, 비트맵 윈도우들이 프로세스된다. 각각의 윈도우에 대해, 루트 객체들이 마킹되는 바, 이는 애플릿 테이블(169)에 의해 나타내진 현재 윈도우 내의 애플릿 객체들(170)을 마킹하는 것과, 윈도우 내에 있는 정적 필드들(176)로부터 참조되는 객체들(174)을 마킹하는 것과, 그리고 윈도우 내에 있는 동적 루트 테이블로부터의 객체들(178)을 마킹하는 것을 포함한다.
- [0027] 실시예에 따르면, 그 다음, 칠드런 객체들이 마킹되는 바, 이는 차일드 객체(171, 172)가 현재 윈도우 내에서 발견되면 비트맵 내의 대응 비트를 마킹(184)함으로써 그 객체를 마킹하는 것을 포함한다. 차일드 객체(180)가 잠재적인 가비지 리스트에서 현재 윈도우 밖에서 발견되면, 객체 및 이의 칠드런은 현재 윈도우 및 잠재적인 가비지 리스트에서 채귀적으로 마킹된다. 차일드 객체가 현재 윈도우 밖에서 발견되고 잠재적인 가비지 리스트에 없는 경우, 객체는 메모리의 이전에 프로세스된 영역에 있고, 이 객체는 이미 도달가능한 것으로 마킹되었으며, 그러므로 이는 무시될 수 있다. 대신, 차일드 객체가 여전히 프로세스될 필요가 있는 메모리의 영역에 있으면, 이는 동적 루트 리스트에 삽입된다.
- [0028] 실시예에 따르면, 차일드 객체들을 마킹하는 동안, 최대 채귀 깊이에 도달되면, 계층구조(hierarchy) 내의 다음 객체는 연기된 객체 리스트에 삽입되고, 현재의 계층구조를 프로세싱하는 것의 종료 시 프로세스될 것이다. 연기된 객체 리스트가 풀(full)이고 최대 채귀 깊이에 도달되었으면, 마킹 프로세스는 비트맵을 리셋함이 없이 현재 윈도우에서 다시 시작되며, 그러므로 제2 및 후속적인 사이클들 내의 루트들의 수를 증가시킨다. 일단, 마킹 단계가 윈도우에서 완료되면, 윈도우 내의 어떤 참조되지 않은 객체들이 잠재적인 가비지 리스트에 삽입된다.
- [0029] 도 4에 도시된 바와 같이, 프로세스는 다음 윈도우 및 이의 루트 및 차일드 객체들(185, 186, 187, 188)로 계속된다. 모든 윈도우들이 프로세스된 때, 잠재적인 가비지 리스트는 오직, 참조되지 않고, 그러므로 가비지인 객체들(189)을 포함할 것이다. 실시예에 따르면, 이 객체들은 사용되는 메모리를 재활용하기 위해 삭제될 수 있다.
- [0030] 도 5는 실시예에 따른 지속 메모리 내의 객체 삭제를 위해 비트맵 윈도우들을 사용하는 알고리즘 또는 방법(190)을 도시하는 순서도이다.
- [0031] 도 5에 도시된 바와 같이, 단계(192)에서, 메모리 공간은 윈도우들로 분할된다.
- [0032] 단계(194)에서, 비트맵은 윈도우들에 대해 초기화되고, 비트는 메모리 내의 어드레스를 나타낸다.
- [0033] 단계들(196, 198 및 202)에서, 비트맵 윈도우들이 프로세스된다. 실시예에 따르면, 각각의 윈도우에 대해, 루트 객체들이 마킹되며, 이는 현재 윈도우 내의 애플릿 객체들을 마킹하는 것, 윈도우에 있는 정적 필드들로부터 참조되는 객체들을 마킹하는 것, 그리고 윈도우에 있는 동적 루트 테이블로부터의 객체들을 마킹하는 것을 포함한다. 그 다음, 하기에 더 상세히 기술되는 바와 같이 기준에 따라 칠드런 객체들이 마킹되고 비트맵들이 업데이트

트된다. 현재 윈도우 내의 어떤 참조되지 않은 객체들은 잠재적인 가비지 리스트에 삽입된다.

- [0034] 단계(204)에서, 모든 윈도우들이 프로세스된 때, 잠재적인 가비지 리스트는 오직, 참조되지 않고 그러므로 가비지인 객체들을 포함할 것이다. 실시예에 따르면, 이 객체들은 사용되는 메모리를 재활용하기 위해 삭제될 수 있다.
- [0035] 도 6은 실시예에 따른 지속 메모리 내의 객체 삭제를 위해 비트맵 윈도우들을 이용하는 알고리즘 또는 방법 (210)을 더 도시하는 순서도이다. 특히, 도 6은 실시예에 따라 어떻게 루트 및 칠드런 객체들의 마킹이 수행될 수 있는지를 더 도시한다.
- [0036] 도 6에 도시된 바와 같이, 단계(212)에서, 각각의 윈도우에 대해 루트 객체들이 마킹되고, 이는 현재 윈도우 내의 애플릿 객체들을 마킹하는 것, 윈도우에 있는 정적 필드들로부터 참조되는 객체들을 마킹하는 것, 그리고 윈도우에 있는 동적 루트 테이블로부터의 객체들을 마킹하는 것을 포함한다.
- [0037] 단계(214)에서, 칠드런 객체들이 마킹되고, 이는 차일드 객체가 현재 윈도우 내에서 발견되면 비트맵 내의 대응 비트를 마킹함으로써 그 객체를 마킹하는 것을 포함한다. 차일드 객체가 잠재적인 가비지 리스트에서 현재 윈도우 밖에서 발견되면, 객체 및 이의 칠드런은 현재 윈도우 및 잠재적인 가비지 리스트에 재귀적으로 마킹된다. 차일드 객체가 현재 윈도우 밖에서 발견되고 잠재적인 가비지 리스트에 있지 않은 경우, 객체가 이미 메모리의 프로세스된 영역에 있으면 이 객체는 이미 도달가능한 것으로 마킹되었고 그러므로 이는 무시될 수 있다. 대신 차일드 객체가 여전히 프로세스될 필요가 있는 메모리의 영역에 있으면, 이는 동적 루트 리스트에 삽입된다.
- [0038] 단계(216)에서, 차일드 객체들을 마킹하는 동안, 최대 재귀 깊이에 도달되면, 계층구조 내의 다음 객체는 연기된 객체 리스트에 삽입되고, 현재의 계층구조를 프로세스하는 것의 종료 시 프로세스될 것이다.
- [0039] 단계(220)에서, 연기된 객체 리스트가 풀(full)이고, 최대 재귀 깊이에 도달되었으면, 마킹 프로세스는 비트맵을 리셋함이 없이 현재 윈도우에서 다시 시작되고, 그러므로 제2 및 후속적인 사이클들에서 루트들의 수를 증가시킨다.
- [0040] 단계(222)에서, 일단, 마킹 단계가 윈도우에서 완료되면, 윈도우 내의 어떤 참조되지 않은 객체들이 잠재적인 가비지 리스트에 삽입된다. 상기에 기술된 바와 같이, 모든 윈도우들이 프로세스된 때, 잠재적인 가비지 리스트는 오직, 참조되지 않고 그러므로 가비지인 객체들을 포함할 것이다. 실시예에 따르면, 이 객체들은 사용되는 메모리를 재활용하기 위해 삭제될 수 있다.
- [0041] 도 7은 실시예에 따른 지속 메모리 내의 객체 삭제를 위해 비트맵 윈도우들을 사용하는 알고리즘 또는 방법 (230)을 더 도시하는 순서도이다. 특히, 도 7은 실시예에 따른 도 5 및 6에서 이전에 기술된 단계들 각각의 조합을 도시한다.
- [0042] 일반적으로, 본 발명은 메모리 청크들을 나타내는 비트맵 윈도우들을 이용한 지속 메모리 내의 객체 삭제를 위한 시스템에 관한 것이며, 상기 시스템은 하나 이상의 마이크로프로세서들에서 동작하고, 프로세서 또는 컴퓨팅적 엔진 및 소프트웨어 객체들의 저장을 위한 지속 메모리를 포함하는 리소스 제약적 환경, 스마트 카드 또는 다른 환경과 같은 컴퓨팅 환경 또는 디바이스에서 데이터 구조를 제공하기 위한 수단 - 상기 데이터 구조는 하나 이상의 메모리 비트맵 윈도우들을 포함할 수 있고, 상기 메모리 비트맵 윈도우들의 각각은 지속 메모리 내의 어드레싱가능한 공간의 청크를 나타냄 - 과, 그리고 지속 메모리로부터 비 도달가능 객체들을 삭제할 시 하나 이상의 메모리 비트맵 윈도우들을 사용하기 위한 수단을 포함한다.
- [0043] 상기 시스템은 잠재적인 가비지 리스트, 동적 루트 리스트, 최대 재귀 깊이 및 연기된 객체 리스트 중 하나 이상을 데이터 구조 내에 제공하기 위한 수단을 더 포함한다.
- [0044] 또한 상기 시스템에서, 잠재적인 가비지 리스트는 이미 프로세스된 윈도우들 내의 참조되지 않은 객체들을 포함한다. 동적 루트 리스트는 이전 윈도우로부터 참조되지만 이전 윈도우 내의 어드레싱가능 공간에 있지 않은 객체들 - 상기 객체들은 다음 윈도우에 대한 루트들이 됨 - 을 포함한다. 최대 재귀 깊이는 마킹 단계에 대한 재귀 깊이를 제한하는 구성가능한 값이다. 연기된 객체 리스트는, 일단 구성된 재귀 깊이에 도달되면 아직 프로세스되지 않은 칠드런 객체들을 포함한다.
- [0045] 상기 시스템은, 윈도우들 내의 객체의 저장을 위해 사용되는 지속 메모리 공간을 분할하기 위한 수단과, 윈도우들에 대한 비트맵을 초기화하기 위한 수단 - 비트는 메모리 내의 어드레스를 나타냄 - 과, 각각의 윈도우에 대해 루트들을 마킹하고 칠드런을 재귀적으로 마킹하기 위한 수단과, 마킹 단계가 윈도우에서 완료될 때, 이 윈도우 내의 어떤 참조되지 않은 객체들을 잠재적인 가비지 리스트 상에 삽입하기 위한 수단과, 그리고 잠재적인 가

비지 리스트가 궁극적으로, 참조되지 않고 가비지로 고려될 수 있는 이 객체들을 포함하도록 윈도우들의 각각을 프로세싱하기 위한 수단을 더 포함한다.

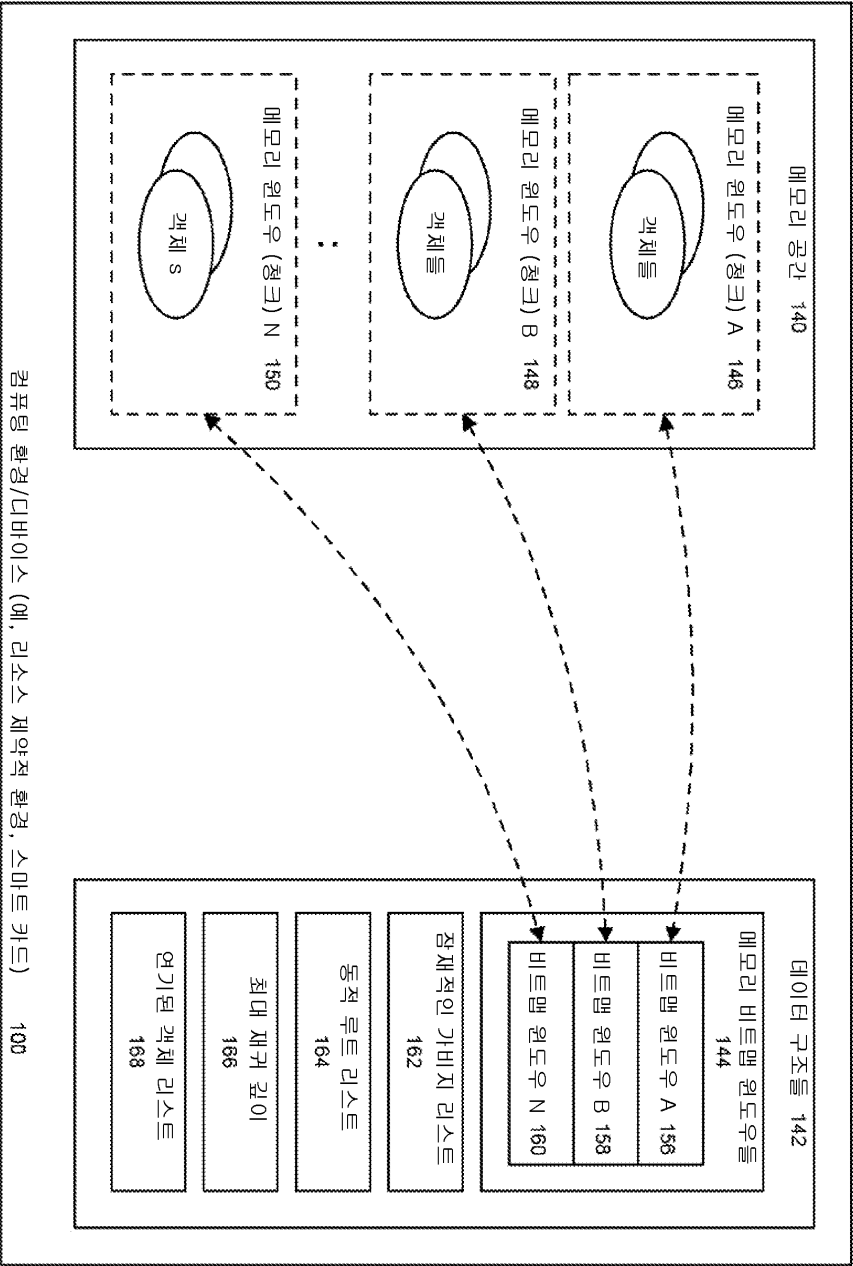
[0046] 본 발명은 통상적으로, 본 발명의 교시들에 따라 프로그램된 하나 이상의 프로세서들, 메모리 및/또는 컴퓨터 판독가능 스토리지 매체를 포함하여 하나 이상의 종래의 범용 또는 특수용 디지털 컴퓨터, 컴퓨팅 디바이스, 머신 또는 마이크로프로세서를 이용하여 구현될 수 있다. 적절한 소프트웨어 코딩은 소프트웨어 기술 분야의 숙련자들에게 분명할 바와 같이, 본 발명의 교시들에 근거하여 숙련된 프로그래머들에 의해 쉽게 준비될 수 있다.

[0047] 일부 실시예들에서, 본 발명은 본 발명의 프로세스들 중 어느 것을 수행하도록 컴퓨터를 프로그램하는 데 사용될 수 있는 명령어들이 저장된 비일시적 스토리지 매체 또는 컴퓨터 판독가능 매체(들)인 컴퓨터 프로그램 제품을 포함한다. 스토리지 매체는 이들로만 한정되는 것은 아니지만, 플로피 디스크(disk)들, 광학 디스크(disc)들, DVD, CD-ROM들, 마이크로드라이브 및 자기-광학 디스크(disk)들을 포함하는 어떤 타입의 디스크, ROM들, RAM들, EPROM들, EEPROM들, DRAM들, VRAM들, 플래시 메모리 디바이스들, 자기 또는 광학 카드들, (분자 메모리 IC들을 포함하는)나노시스템들 또는, 명령어들 및/또는 데이터를 저장하기에 적절한 어떤 타입의 매체 또는 디바이스를 포함할 수 있다.

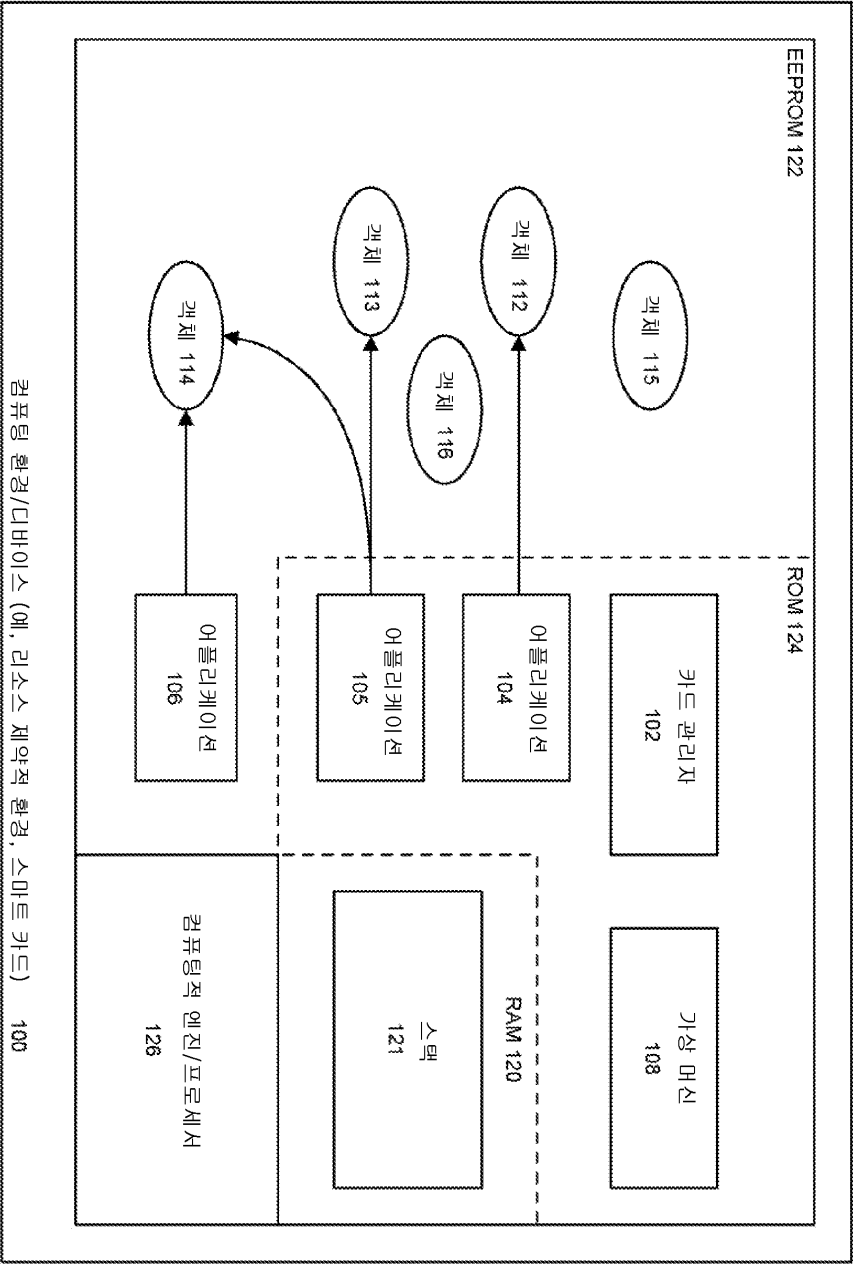
[0048] 본 발명의 상기 상세한 설명은 예시 및 설명을 위해 제공되었다. 본 설명은 완전한 것이거나 또는 정확히 개시된 형태들로만 본 발명을 제한하고자 의도된 것이 아니다. 많은 수정들 및 변형들이 이 기술분야의 숙련자에게 분명할 것이다. 위 실시예들은 본 발명의 원리 및 이의 실용적 응용을 가장 잘 설명하기 위해 선택 및 기술되었으며, 그럼으로써 이 기술분야의 숙련자들은 본 발명에 대한 다양한 실시예들 및 고려되는 특별한 사용에 적합한 다양한 수정들을 이해할 수 있다. 본 발명의 범위는 다음의 특허 청구 범위 및 이의 균등물에 의해 한정되어야 함이 의도된다.

도면

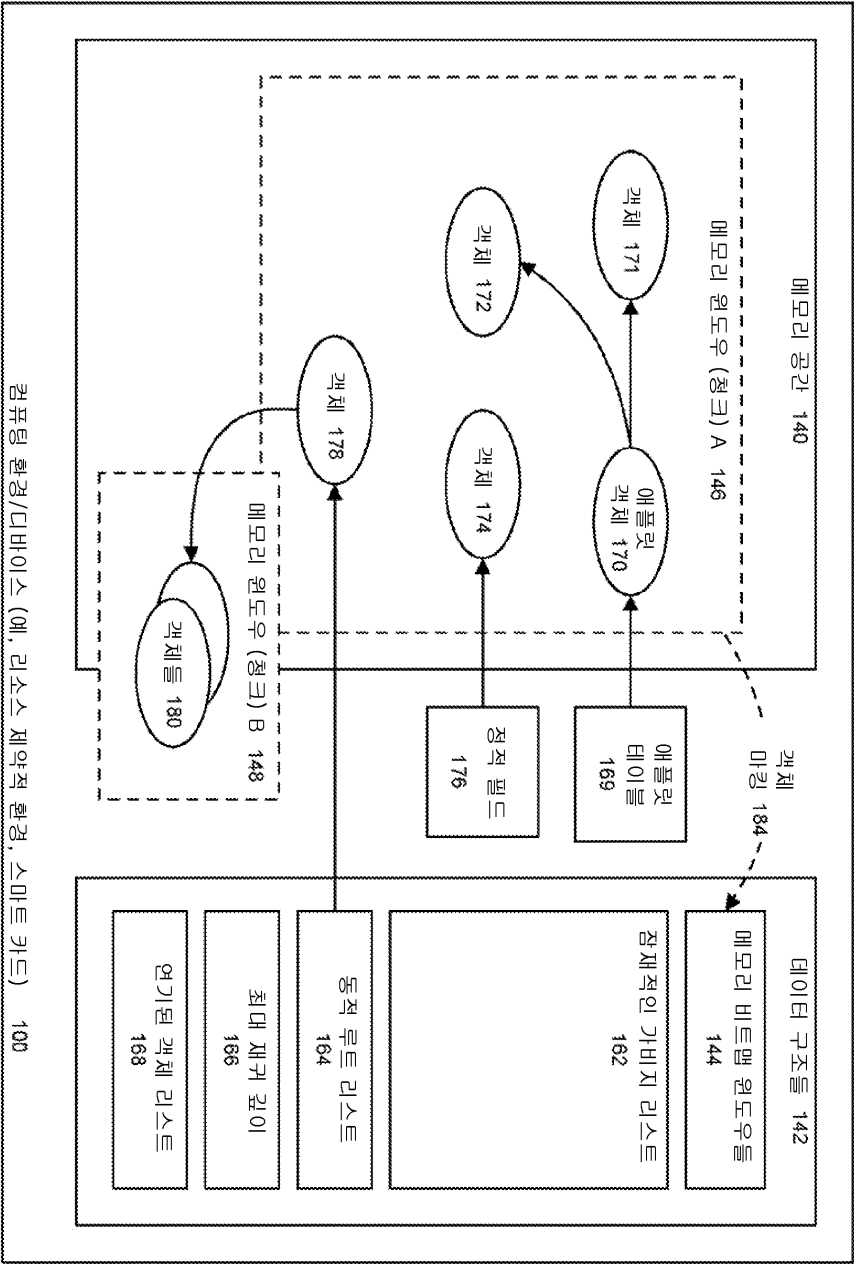
도면1



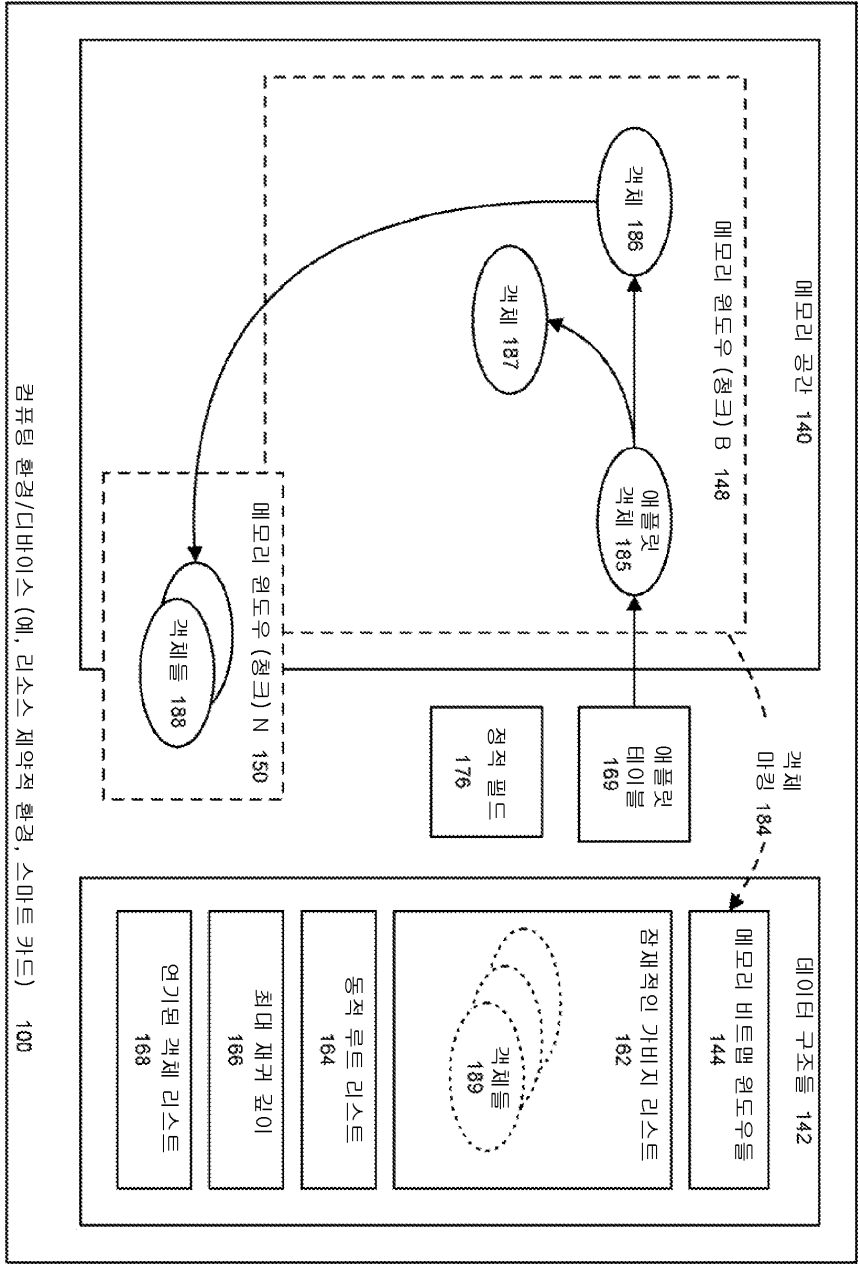
도면2



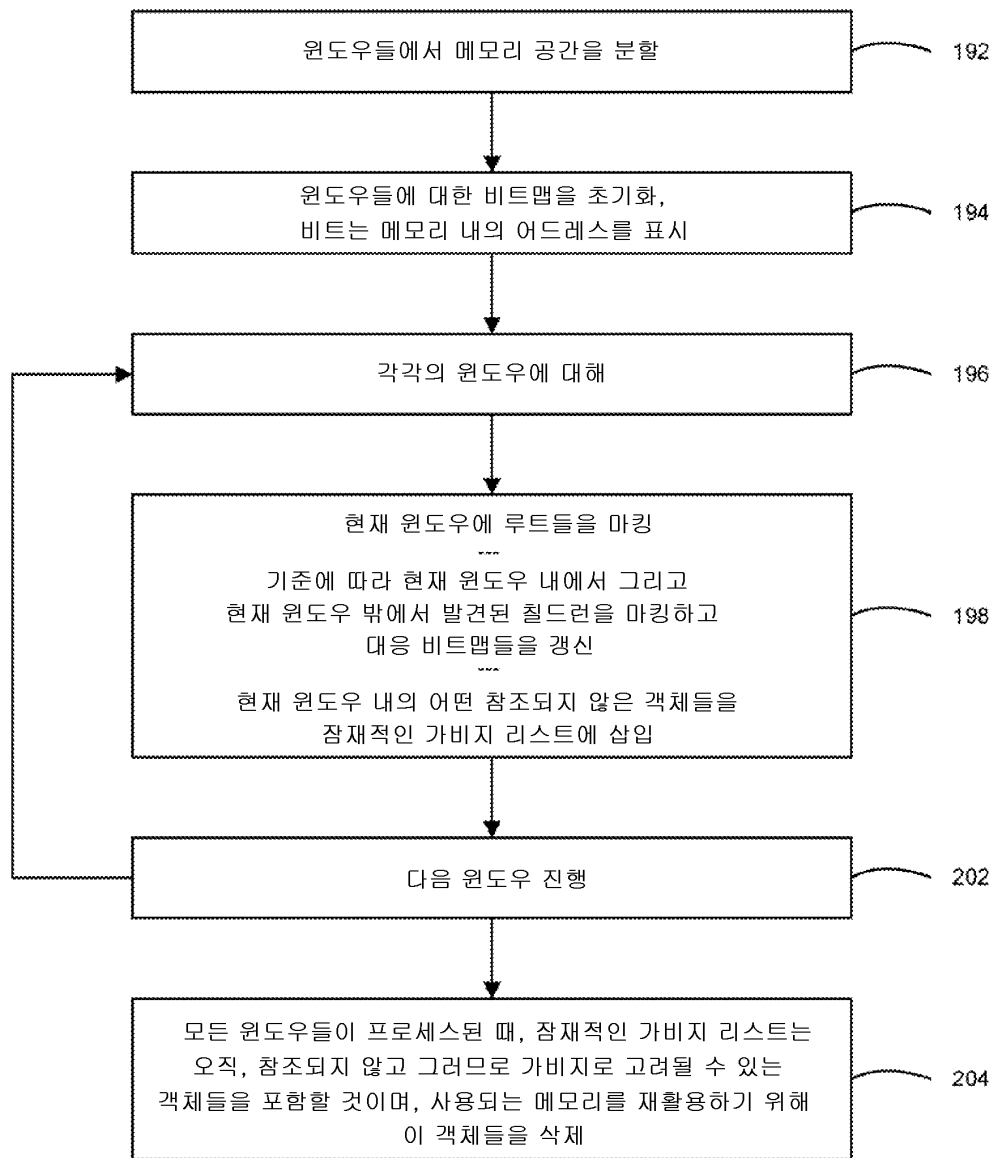
도면3



도면4

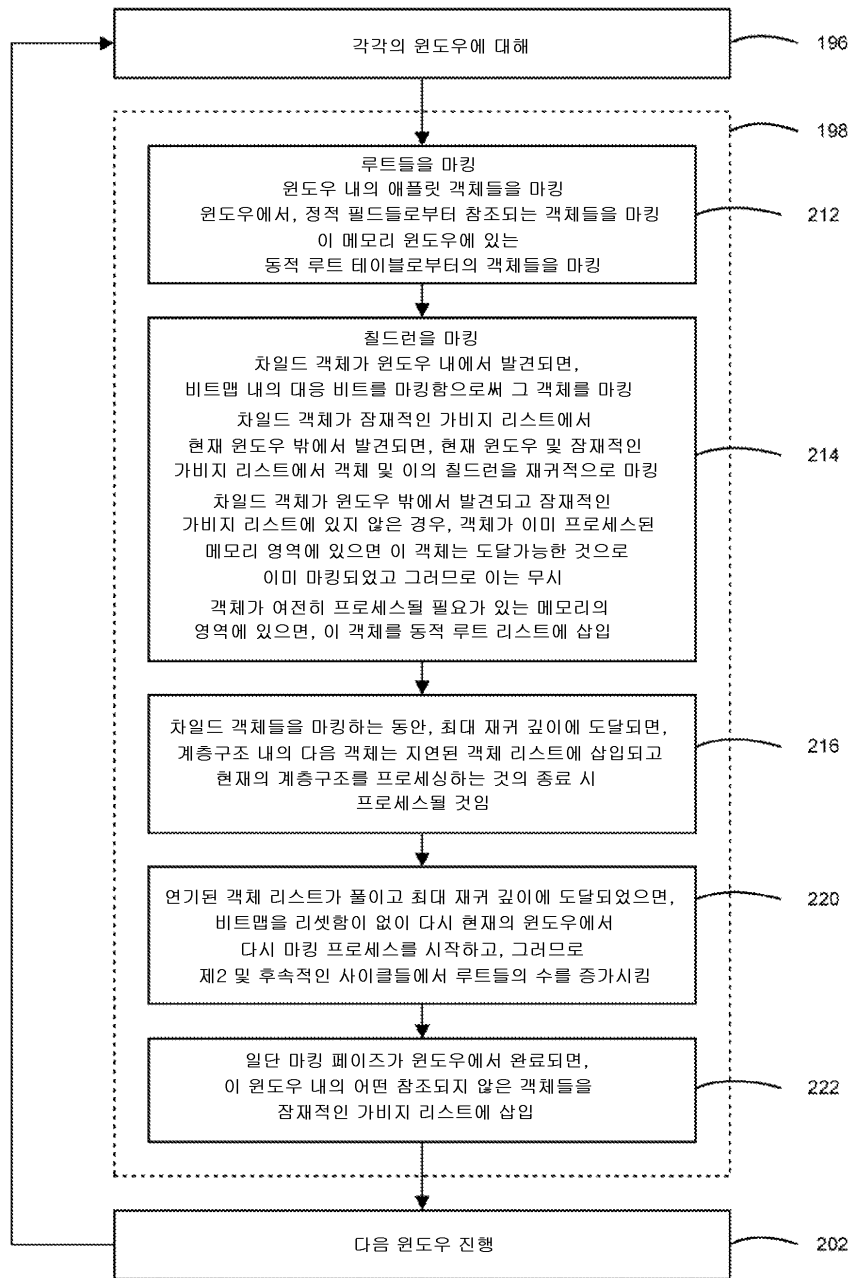


도면5



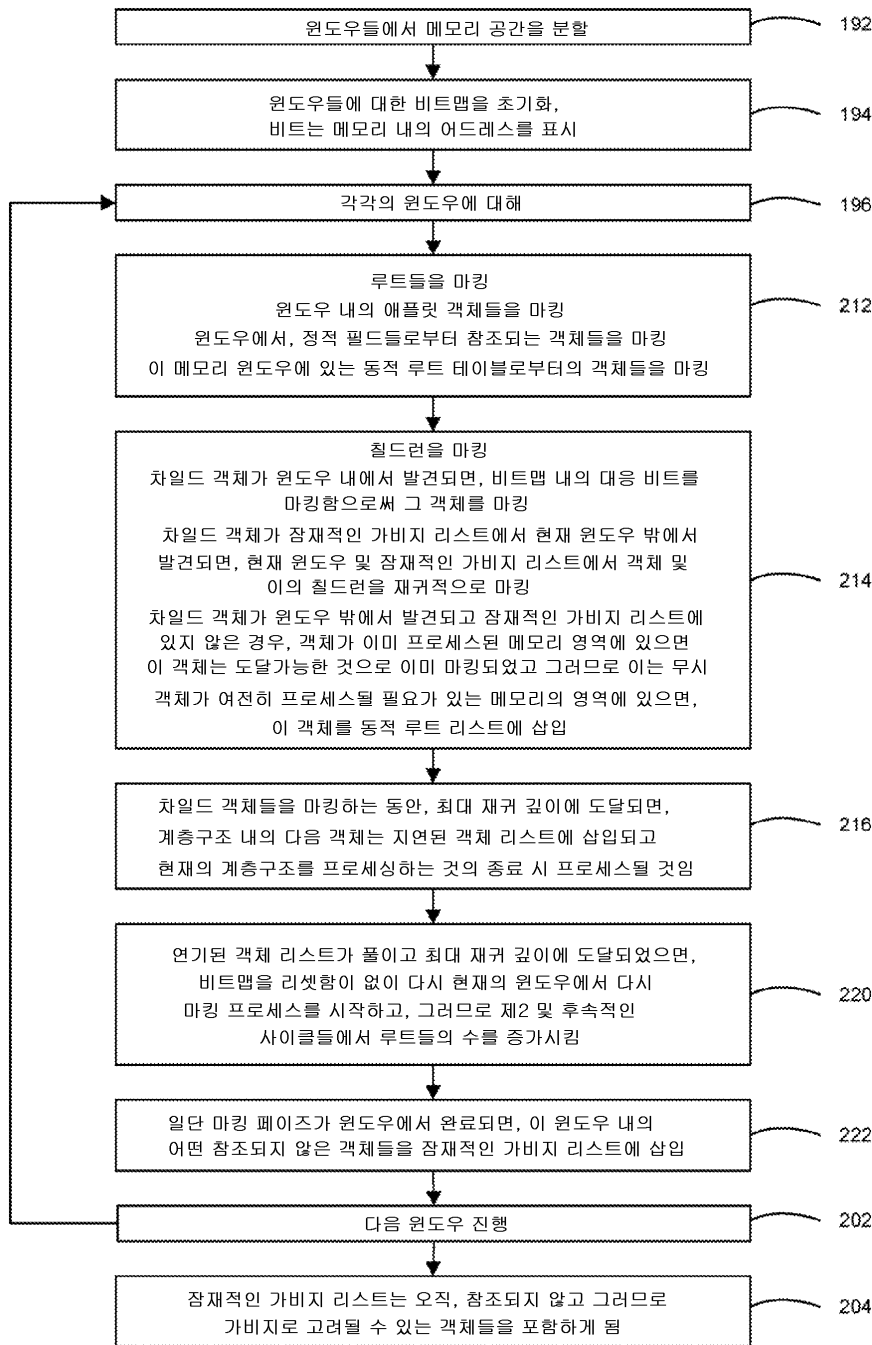
190

도면6



210

도면7



230

【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 1항

【변경전】

차일드 개체

【변경후】

차일드 객체

【직권보정 2】

【보정항목】 청구범위

【보정세부항목】 11항

【변경전】

차일드 개체

【변경후】

차일드 객체

【직권보정 3】

【보정항목】 청구범위

【보정세부항목】 5항

【변경전】

차일드 개체

【변경후】

차일드 객체