(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) Int. Cl.[6]
H04B 1/59

(45) 공고일자 2004년04월30일
(11) 등록번호 10-0404274
(24) 등록일자 2003년10월22일

(21) 출원번호	10-1996-0702954	(65) 공개번호	10-1996-0706720
(22) 출원일자	1996년05월30일	(43) 공개일자	1996년12월09일
번역문제출일자	1996년05월30일		
(86) 국제출원번호	PCT/IB1995/000759	(87) 국제공개번호	WO 1996/11532
(86) 국제출원일자	1995년09월13일	(87) 국제공개일자	1996년04월18일
(81) 지정국	국내특허 : 일본 대한민국 EP 유럽특허 : 오스트리아 벨기에 스위스 리히텐슈타인 사이프러스 독일 덴마크 스페인 핀랜드 프랑스 영국 그리스 아일랜드 이탈리아 룩셈부르크 모나코 네덜란드 포르투칼 스웨덴		

(30) 우선권주장 94202839.0 1994년09월30일 EP(EP)
94202961.2 1994년10월13일 EP(EP)

(73) 특허권자 코닌클리케 필립스 일렉트로닉스 엔.브이.
네델란드왕국, 아인드호펜, 그로네보드스베그 1
(72) 발명자 요한 헨드릭 안톤 겔리센
네델란드, 베아 아인드호펜 5621, 그로네보드세베그 1
(74) 대리인 이병호

심사관 : 정재우

(54) 멀티플랫폼인터프리터를포함한응용프로그램을포함하는대량데이타의표현을수신하는멀티미디어시스템,상기멀티플랫폼인터프리터와상호작용하는플랫폼서브시스템및,그시스템또는서브시스템용대용량메모리

명세서

기술분야

<1> 본 발명은, 데이타 표현(data presentation)을 수신하는 데이타 처리 시스템으로서, 상기 데이타는 멀티플랫폼 인터프리터를 포함한 응용 프로그램을 포함하고, 더욱이 상기 시스템은 상기 멀티플랫폼 인터프리터와 상호 작용하도록 배열된 플랫폼 서브시스템을 가지며, 상기 응용 프로그램에 의해 구동되고 사용자 입력 수단의 제어 하에서 상기 멀티플랫폼 인터프리터를 실행하기 위한 처리 수단을 포함하며, 상기 멀티플랫폼 인터프리터에 의해 제어 가능한 사용자 출력 수단을 가진다. 다량 데이타는 광디스크와 같은 자체 내장된 대용량 메모리 상에 물리적으로 존재할 수도 있다. 대안으로 다량 데이타는 원격 상호 접속 또는 채널을 통해 대화 구조(dialog structure)로 제공될 수도 있다. 멀티미디어는 서브시스템이 하나보다 많은 신호 카테고리, 예컨대 영문숫자 텍스트(alphanumerical text), 그라픽스, 음성, 음악 등등의 카테고리를 수신, 처리 및 출력할 수 있는 것을 의미한다. 또한, 간단히 응용이라 언급되는 응용 프로그램의 대상은 비디오 게임, 백과사전과 같은 교육 또는 정보 아이템, 개인용 파일링 시스템용 셸(shell) 등등으로 다양하다. 그런 다양한 플랫폼은 CD-I 머신들 및 개인용 컴퓨터들 등으로 상품화되어 있다. 일반적으로 CD-I 표준들 및 컴팩트 디스크 표준들에 대해서는 ACM/Springer Multimedia Systems, Vol. 2, No. 4의 175-171 페이지(1994)에 수록된 Jan Korst 및 Verus Pronk 에 의한 'Compact Disc Standards(컴팩트 디스크 표준들)'이란 논문이 참조되는데, 이 논문은 그런 표준들에 대한 보다 일찍이 공개된 다양한 명세의 개설이다.

<2> 응용 프로그램은 통상적으로 타이틀(title)로 불려지며, 다양한 이런 타이틀들은, 타이틀이 특정 유형 또는 특정 브랜드의 서브시스템에서는 재생되고 다른 서브시스템에는 재생되지 않도록 서브시스템 하드웨어 제조자 등에 의해 제공되어 왔다. 따라서, 이들은 플랫폼 특정적(platform-specific)이다. 다양한 유형의 플랫폼에서 재생할 수 있는, 응용 프로그램들이 잠재적으로 더 많은 소비자에 기반을 둘 수 있도록 허용할 필요성을 느껴왔다.

<3> 발명의 개요

<4> 따라서, 특히 본 발명의 목적은 저장된 정보의 상당한 부분이 일치된 데이타 포맷들(harmonized data formats)을 가지며 다양한 상이한 플랫폼 서브시스템에서 인터프리트 가능하다는 점에서 응용 프로그램을 보다 덜 플랫폼 특정적으로 하는 것이다. 특히, 본 발명의 발명자들은, 특정 환경들 하에서는 송출 플랫폼들(delivery platforms)에 의한 어떤 실시간 변환을 필요로 할 수도 있지만, 대부분의 플랫폼들에 의해 재생될 수 있는 멀티미디어 데이타에 대한 포맷들을 정의함으로써 프로그램들이 상이한 플랫폼들에서 인터프리트 가능해진다는 것을 발견했다. 또한, 대용량의 멀티미디어 데이타가 타이틀에서 한번만 제공된다하더라도, 플랫폼 특정적 포맷들로 몇개의 데이타를 갖는 것이 유리하다는 것이 입증되었다. 본 발명의 다른 목적은 생성 플랫폼과 재생 플랫폼들의 분리(decoupling)이다. 본 발명의 그 측면들 중 하나에 따르면, 본 발명은 상기 응용 프로그램이 상기 실행을 위해 상기 처리 수단에 의해 인터프리트되는 소정의 명령 세트에 기초하며, 그에 따라 상기 플랫폼 서브시스템은 상기 소정의 명령 세트와 소정의 데이

타 유형 세트를 갖고 양적인 최소 필요조건들에 따르는 자원 설비들(resource facilities)을 더 포함하는 미리 특정된 추상적 머신(prespecified abstract machine)의 예(instance)인 것을 특징으로 한다. 특히, 멀티플랫폼 멀티미디어 타이틀 생성은 플랫폼 독립 멀티미디어 타이틀들을 위한 것이다. 위에서, 예는 그런 종류들의 실제 전형 또는 실현이며 각 예는 상기 종류들의 최소 설비들을 가지지만, 여러 예들은 상당히 다를 수도 있다.

<5> 본 발명은 또한 멀티플랫폼 인터프리터를 포함하는, 시스템에 사용하기 위한 자체 내장된 대용량 메모리에 관한 것으로,

<6> - 플랫폼 종속 원시어들(primitives)의 위치 어드레스들을 포함하는 디스패치 테이블(A),

<7> - 실행가능하며 인터프리트 가능한 코드를 저장하기 위해 상기 위치 어드레스에 의해 어드레스가능한 모든 위치들을 포함하는 커널 (kernel) 영역(B) 및,

<8> - 플랫폼 독립적이며 타이틀 독립적인 코드를 포함하는 제 1 영역(C)과 타이틀 종속적인 코드를 포함하는 제 2 영역(D)을 갖는 확장 커널 영역(C+D)을 포함한다. 이런 요소들의 장치는 간단하며, 용이하게 액세스될 수 있으며, 에러들을 거의 발생시키지 않는다는 것이 입증되었다.

<9> 본 발명은 또한 이와 같은 타이틀을 포함하는 다량 데이타와 상호 작용하도록 배열되고 상기 멀티플랫폼 인터프리터를 로딩하기 위한 로드 수단(load means)을 갖는 플랫폼 서브시스템에 관한 것이다. 다양한 추가의 유리한 면들은 종속 청구항들에 기재되어 있다.

도면의 간단한 설명

<10> 상기 및 다른 면들 및 이점들은 이하의 양호한 실시예들의 개시 내용을 참조하여, 특히 첨부된 도면들을 참조하여 더 상세히 설명될 것이다.

<11> 제 1 도는 다양한 프로그램 유형들의 기능적 위치를 나타낸 도면.

<12> 제 2 도는 인터프리터 접근법을 나타낸 도면.

<13> 제 3 도는 CPF 송출 플랫폼을 나타낸 도면.

<14> 제 4 도는 CPF 런타임 시스템(runtime system)을 나타낸 도면.

<15> 제 5 도는 FORTH 인터프리터에 대한 메모리 배치를 도시하는 도면.

<16> 테이블 1 내지 8 은 다양한 예시적 프로그램 모듈들을 나타낸다.

<17> 양호한 실시예들의 상세한 설명

<18> 제 1 도는 본 발명의 양호한 실시예에 사용되는 다양한 프로그램 유형들의 기능적 위치를 나타낸 도면이다. 다양한 방법으로 실현되는 본 발명의 목적은 소위 공통 공개 포맷(Common Publishing Format)을 획득하는 것이며, 그 결과 상이한 플랫폼 서브시스템들 또는 상이한 하드웨어 머신들에서 재생될 수 있도록 공개된 타이틀들에는 최소의 제한조건들(constraints)만이 붙을 필요가 있게 된다. 소정의 멀티미디어 포맷 및 프로세서 독립 프로그램 코드로 표현되는, 예컨대 화상, 음성, 구조(structure), 문자 폰트(character font) 등등의 다양한 소위 애셋(asset)들은 함께 타이틀을 형성한다. 본 발명에서는 각각의 상이한 해당 플랫폼마다 따로따로 모든 자료들을 단일의 대용량 메모리에 저장할 필요성을 제거한다. 타이틀의 애셋들은 인간인 사용자가 보거나 듣게될 수 있는 요소들이다. 프로그램 코드는 특정 필요조건들에 부합하는 플랫폼 서브 시스템들에서 인터프리트 가능하다. 코드는 각각이 명확한 의미들을 갖는 짧은 포맷의 명령들(short format instructions) 또는 토큰들(tokens)로 구성된다. 그들이 기록되는 언어는 CPF-스크립트라 불려지며, 이것은 OSMOSE 프로젝트, ESPRIT 프로젝트 6788(출판 버전 1.0, Copyright Philips Electronics N.V., The Netherlands, 1994)의 보고서에 개시되어 있다. 이들 토큰 중 몇몇은 특히, 컴퓨터들과 관련 하드웨어의 높은 처리 속도에 비해, 인간의 인지 및 표현이 비교적 느리기 때문에, 처리의 실행에 비해 비교적 많은 시간을 소비하는 음성-시각의 부작용(audio-visual side effect)을 초래할 것이다. 모든 필요조건들을 특정하기 위해, 소위 "추상적 머신(abstract machine)"이 정의되어 왔다. 이와 같은 추상적 머신은 소정의 명령 세트와 소정의 데이타 유형 세트를 가지며, 양적인 최소 필요조건들(quantitative minimum requirements)을 충족시키는 자원 설비들(resource facilities)을 갖는다. 적어도 이들 설비들을 갖는 임의의 실제 머신 또는 플랫폼은 해당 타이틀을 실행시킬 수 있다. 이와 같은 설비들은 동작 속도, 메모리 용량, I/O(디스플레이, 음향), 운영체제 특성들, 코프로세싱(coprocessing) 등등으로 나타낼 수도 있다. 제 1 도에서, 원(20)은 CPF-스크립트로 번역되는 모든 프로그램들을 나타낸다(원의 크기는 이들 프로그램들의 수 및 크기와 전혀 상관없다). 이들 프로그램들의 실제 기록은, 약간의 기능을 희생하여, CPF-스크립트의 프로그램 작성자들의 생활을 더욱 쉽게 만드는 저작 언어(authoring language)의 사용을 통해 행해질 수 있다. 즉, 얻어진 프로그램들은 원(22)으로 표현된다. 원(20)에서의 프로그램의 편집(compilation) 및 일치(conformation)는 추상적 CPF 머신에 의해 실행될 수 있는 모든 가능한 CPF 스크립트 프로그램들을 포함하는 원(26)으로 이들이 표현된다. 그 주위의 원(24)은, 프로그램이 CPF-스크립트로 되어야 한다는 제한조건없이 이러한 머신에 의해 실행될 수 있는 모든 프로그램들을 나타낸다. 원의 쌍들(28/30 및 32/34)은 두개의 임의의 상이한 플랫폼 서브 시스템에 관련된다. 안쪽원들(30, 34)은 원(24)으로부터의 매핑(mapping)을 나타낸다. 바깥쪽원들(28, 32)은 추상적 머신에 의해 실행될 수 없는 안쪽원 외측의 모든 실행가능한 프로그램들의 세트를 나타낸다. 또한, 도시된 다양한 원들의 직경들은 완전히 임의적이다. 또한, 다양한 플랫폼 서브시스템들에 대한 안쪽원들의 내용들은 동일하지만 바깥쪽원들의 내용은 그렇게 할 필요는 없다.

<19> 본 발명의 주목적은 타이틀 개발자(title developer)로 하여금 실제 또는 물리적 플랫폼으로부터 추상화될 수 있도록 하는 것으로, 이것은 추상적 머신의 도입을 통해 행해진다. 이 추상적 머신은 프로그램을 실행할 때 그 플랫폼 서브시스템의 최소 행동(behaviour)뿐만 아니라 최소 자원 필요조건들을 예컨대, 오디오 및 비디오 출력 성능, CD-ROM/XA 인터페이스능력(interfaceability), Korst 등의 참조, 타이

머 기능들, 사용자 입력 메카니즘들, 그 명령 세트, 지원될 데이타 유형으로 정의한다. 실제 플랫폼은 소위 CPF- 스크립트 프로그램을 추상적 머신의 명령 세트의 구현을 통해 재생할 수 있어야 한다. CPF-스크립트에서 정확한 프로그램과 부정확한 프로그램 사이의 차이는 이와 같은 프로그램에 대한 정합 시험(conformance test)을 적용함으로써 이루어진다. 동일한 이유로, 정확한 플랫폼과 부정확한 플랫폼을 판별하기 위해, 하드웨어 개발자에게 제공된 타당성 시험(validation test)이 있다. 즉, 여기서 시험은 플랫폼의 성능들이 충분한지의 여부에 관한 것이다. 그런 타당성 시험의 실용적인 실현은 적합한 프로그램들의 세트의 제공 및 해당 플랫폼에서의 재생가능성(playability) 또는 그 반대의 것을 시험하는 것이다.

<20> 제 2 도는 인터프리터 접근법을 도시하는 도면이다. 본 발명에서 취해진 접근법은 데이타 포맷들을 일치시키고 상이한 플랫폼들에서 인터프리트 가능한 프로그램들을 만드는 것이다. 일반적으로, 송출 플랫폼에 의한 약간의 실시간 변환이 필요할 수도 있지만 대부분의 플랫폼들에 의해 재생될 수 있는 멀티미디어 데이타 포맷들을 정의하는 것이 가능하다. 어떤 데이타는 여전히 두개이상의 데이타 포맷들로 주어져야 할 수 있다. 나아가, 실제 응용 프로그램은 고레벨 표현이고 송출 플랫폼(delivery platform)에 의해 인터프리트되어야 한다. 예를 들면 특정 플랫폼에 특정적인 유틸리티들(이들 유틸리티들에 의해 이용되거나 이용하는)에 관계하는 몇몇 특수 기능들은 해당 플랫폼에 특정적인 방식으로 부호화된다. 제 2 도의 구성에서, 블럭(38)은 CPF 프로그램을 나타낸다. 레벨(40)은 각 송출 플랫폼에 대한 하나의 특정적인 커널(kernel) 인터프리터를 도시한다. 상기 커널들(40)은 CPF 프로그램(각 플랫폼에 대해 동일 프로그램)을 플랫폼-독립(platform-independent) 인터페이스 호출들로 번역한다(translate). 이들의 구현이 CPF 추상적 머신을 구성한다.

<21> 플랫폼 독립 CPF-타이틀들에 이어, 각 플랫폼 유형은 그 특정 플랫폼에 특정적인 하나 이상의 타이틀들을 갖는다. 이들 플랫폼 몇몇의 이점은 그 전용 또는 특정적인 하드웨어에 기인한다. 그러므로, 플랫폼 독립을 얻기 위해, 프로그램들은 저레벨 시스템 기능으로부터 추상화한다. 이것은 어느 정도 CPF 기능이 그들 자신에 의해 취해진 각각의 플랫폼들의 것보다 다소 낮음을 의미한다. 인터프리터들(40)의 출력은, 각각 다른 플랫폼, 특히 개인용 컴퓨터(46), 및 CD-I 플레이어(48)를 나타내는 열(columns: 42..48) 중 적당한 하나에 주어진다. 그것의 실현에서, 해당 플랫폼은 라이브러리(CPF.DLL, CPF.LIB), 운영 체제(마이크로소프트 윈도우즈, CD-RTOS: 콤팩트 디스크 실시간 운영 체제, 미국, 아이오와, 데스 모이네스, 마이크로웨어에 의해 유도됨) 및 적당한 하드웨어(멀티미디어 PC 하드웨어, CD-I 하드웨어)를 포함한다. 위에서, DLL 은 Dynamic Link Library under Windows 이고, LIB 는 CD.RTOS Library 이다. 좌측의 플랫폼들(42, 44)에 있어서, 다양한 서브시스템들간의 시퀀스들은 대응하고 있다.

<22> 제 3 도는 로딩되고 사용할 수 있는 응용 프로그램과 함께 CPF 송출 플랫폼을 도시한다. 각 구성은 핸들러(handler) 레벨에서 이용 가능한 현 플랫폼 능력을 다이어그램의 보다 높은 다음 레벨로 건넨다. 레벨(60)은 응용 프로그램 자체를 포함하고, 이는 추상적인 머신의 동일한 예들(instances)간 뿐만아니라, 제 2 도에 관해 설명된 상이한 예들 간에도 이식가능하다. 프로그램 또는 타이틀은 CPF 스크립트로 나타내어지고, 그러한 것으로서 층(60)은 물리적 디스크를 추상한 것이다. 다음 레벨은 플랫폼 특정적인 인터프리터(62) 및 런타임 라이브러리(64)를 포함한다. 즉 다음 하위 레벨에서 이들은 멀티미디어 핸들러 인터페이스를 가진다. 다음의 하위 레벨은 핸들러들(68 내지 74)의 셋과, 해당 레벨내에 존재하는 유효 구조(effective organization)를 다음 상위 레벨로 건네기 위한 작은 배치 기능(configuration function)을 포함한다. 상기 핸들러들의 셋업은 배치 특정적이며, 예로서 도시한 바와 같이, 데이타 및 디바이스 핸들러(68), 디스크 핸들러(72), 오디오 및/또는 비디오 디스플레이와 상호 작동하는 프리젠테이션 핸들러(72), 및 유틸리티 핸들러(74) 등이 있다. 이 상황에서, 이들 유틸리티들은 그래픽 입력 메카니즘들 등의 추가 설비들에 의해 실현될 수 있다. 즉 다음 하위 레벨에서 핸들러들은 디바이스 드라이버 인터페이스를 갖는다. 그 자체로 잘 알려진 것과 같이, 이와 같은 핸들러들은 각각 하나 이상의 소프트웨어 모듈들을 포함하며, 이것은 입력으로 다양한 범용 제어 문(statement)들을 수신하고, 이들을 관련 디바이스의 레벨 상에서 기본 동작들을 정의하는 각각의 표현식의 시퀀스들로 번역한다. 즉 상기 다음 하위 레벨에서, 핸들러들은 디바이스 드라이버에 대한 인터페이스를 가진다. 상기 다음 하위 레벨 자체는 디바이스 드라이버들(78); 및 상기 레벨 내의 구성을 다음 상위 레벨로 건네기 위한 배치 기능(76)을 갖는다. 드라이버들은, 그 자체로 잘 알려진 것과 같이, 각각 다음 상위 레벨에 의해 생성된 표현식들 (expressions)에 응답하고, 이들을 전자기계, 전자, 및 해당 디바이스의 가능한 추가의 치수 (demensioning) 등에 의해 결정된, 디바이스상의 동작(action)들로 번역한다. 즉 상기 다음 하위 레벨에서, 하드웨어 플랫폼은 하드웨어 시스템 버스를 포함한다. 비제한적인 예로서, 다양한 디바이스들이 다음 하위 레벨 상에 표시되어 있다. 즉, 포인팅 장치(pointing device)들과 같은 사용자 인터페이스 장치들과 상호 작용을 하는 아날로그 장치(80), 위의 Korst 등의 참증에 정의되어 있고, 해당 타이틀의 실시간 재생을 허용하기 위해 애셋들(assets)의 로딩, 및 로딩된 CD-ROM으로부터의 가능한 응용을 허용하는 CD-ROM/XA 장치(82), 데이타 베이스나 신호 처리와 같은 다양한 기능들을 실행하는 범용 처리 장치(84), 광디스크와 같은 국부적으로 주어진 단일의 메모리 매체로부터 대신에, 원격 통신 채널을 통해 원격지(remote site)에 공급된 다량 데이타를 수신하는데 이용되는(원격) 통신 장치(86)가 다음 레벨상에 표시되어 있다. 이와 같은 데이타는 이후 독립적으로 또는 플랫폼 서브시스템이나 서브 디바이스에 의한 요구시 주어질 수 있다. 최종 요소들은 오디오 장치, 비디오 디스플레이 또는 이들의 서브시스템 등의 프리젠테이션 장치(88)이다. 도시한 것과 같은 장치들은 직접 물리적 링크들로 상호 접속될 수 있다.

<23> 제 4 도는 CPF 런타임 시스템을 도시하는데, 이것은 실제로 본 발명에 따른 응용 프로그램들을 처리하고 CPF 포맷에 일치시키기 위한 실행 모델이고 오디오, 비디오, 및 제어 레벨상의 상호 작용들을 포함한다. 도시된 것과 같이, 상기 제어 채널들은 제어 신호들(C)을 반송하고, 데이타 채널들은 데이타 신호들(D)을 반송하며, 타이밍 채널들은 타이밍 신호들(T)을 반송한다. 중앙 명령 및 데이타 메모리 박스(100)는 인터프리터 프로그램(102)과 직접 상호 작용하는 응용 프로그램을 나타내고, 후자는 또한 응용 프로그램 자체로서 동일 타이틀의 부분으로서 플랫폼에 주어진다. 박스(100)로 들어가는 외부 C&D 입력들(101)은, 그 하위 레벨 하드웨어로의 인터프리터에 의해 발행된 명령들의 결과로서, 그 중에서도 하위 레벨 플랫폼 하드웨어에서 생긴 멀티미디어 애셋들을 나타내는 관련 패러미터들을 갖는 CPF 스크립팅 구문들(constructs)을 나타낸다. 상기 도면의 중앙 부분은 시스템의 플랫폼-종속 부분을 나타내고, 한편, 바깥 블럭들은 시스템의 플랫폼 독립 부분을 나타낸다. 특히, 상기 도면은 CD/ROM(XA) 소프트웨어 블럭(108), CD/ROM(XA) 하드웨어블럭(116), CD/ROM(XA) 드라이브(122), 오디오 소프트웨어(106), 오디오 하드

웨어(114), 오디오 버퍼들(104), 및 오디오 출력 구조들(120), 비디오 소프트웨어(110), 비디오 하드웨어(118), 비디오 버퍼들(112), 및 비디오 출력 구조들(124)을 포함한다. 명확히 하기 위해, 사용자 제어 동작들은 이 도면에 도시하지 않았다. 특히, 오디오 및 비디오 데이타 스트림들은 박스(100)를 통과하지 않는데, 이는 오버헤드 및 관련 저속 동작(slower operating)을 야기할 수 있기 때문이다. 비실시간 동작 모드에서, 타이틀은 오디오, 비디오 및 제어 화일들을 가지며, 제어 화일은 파일들 중 어느 것이 다른 장치로 전송되는가를 알 수 있도록 충분한 정보를 포함한다. 반면, 실시간 동작 모드에서는, 오디오, 비디오, 및 제어의 인터리브된 파슬들(parcels)을 포함하는 실시간 파일이 있다. 인터프리터 내의 핸들러는 이들 중에서 의도된 위치들로 전송하기 위해 파슬들의 선택 및, 동시에 이들 파슬들에 정확한 오디오 및 비디오 관련 기능들의 할당을 처리한다. 상기 도면에 도시된 타이밍 정보는 기초를 이루는 하드웨어의 기능 레벨들에 의존하여 다양한 동작들을 동기화할 수 있다.

<24> 플랫폼 독립 포스(FORTH) 사전들

<25> 이하, 널리 알려진 포스(FORTH) 언어 (프로그래밍 언어에 대한 다양한 교본 참조)용 인터프리터들의 세트가 입력으로부터 출력으로의 이들의 매핑에 기초하여 설명된다. 간략히 하기 위해, 포스 언어 자체는 특별히 고려되지 않는다. 상기 인터프리터는 명령들을 인터프리트하고, 거기에서 요청된 애셋들을 인출(fetch)하고, 라이브러리로부터 관련 아이템들을 불러낸다. 각각의 상이한 플랫폼에 대해, 애셋들의 렌더링(rendering)으로 끝날 수 있는, 동일한 명령들의 소스 화일 세트가 존재한다. 원칙적으로, 적어도 두개의 이와 같은 인터프리터들이 필요하며, 일반적으로 운영 체제 도스/윈도우 3.1 하의 PC를 위한 것의 하나와, CD-RTOS 하의 CDI 플레이어를 위한 것의 하나가 필요하다. 또 다른 실제 플랫폼은 애플 매킨토시 머신일 수 있다. 제 5 도는 이와 같은 포스(FORTH) 인터프리터용 메모리 레이아웃을 도시한다. 이하에 그 동작을 설명한다. 상기 메모리 레이아웃은 메모리 위치들의 열(column)로 도시된다. 제 5 도의 메모리 맵과 관련해서, 인터프리터는 간접적으로 스레디드(threaded)되는 것으로 가정한다. 상기 후자의 특성은 우선 사전(dictionary)에서 이용 가능한 직접 스레디드 코드에 포인트들을 제공하는 간접 또는 디스패치 테이블이 액세스되고; 이후, 인터프리테이션을 즉시 시작할 수 있다는 것을 의미한다. 메모리에서, 부분(140)은 해당 머신의 실행을 가능케 하는 기본 동작들 또는 머신(플랫폼) 종속 원시어들(primitives)(A) 또는 호출가능 포스 어드레스들(Callable Forth Addresses) 즉 cfa's의 테이블을 포함한다. 부분(A)은 로딩 중 부분들(C+D)의 정의들과 부분(B)의 머신 종속 명령들간의 간접 어드레싱 특징을 통해 바인딩을 제공하는 디스패치 테이블이다. 부분들(C+D)에서의 참조들(referrals)은 부분(B)의 관련된 머신 종속 명령들을 찾기 위해 부분(A)의 상기 테이블을 이용한다. 부분(140)의 각 요소는 메모리 부분(142)의 그 자신의 세그먼트를 가리키고, 이는 다양한 머신 종속 원시어들(B)의 구현들을 포함한다. 이는 인터프리터의 핵심이고 모든 CODE(실행가능한 어셈블리 코드)와; CODE(포스 인터프리터 가능 코드), 어드레스용 인터프리터, 및 초기화 코드를 포함한다. 상기 테이블은 타이틀의 머신-독립 부분의 로딩시 및 타이틀 생성 도중, 사전의 이러한 부분의 저장(기억)시 이용된다. 동시에, 부분들(A 및 B)은 타이틀의 머신 종속 부분을 나타낸다. 그러므로, 적어도 두개의 상이한 하드웨어 플랫폼용 다중 플랫폼 타이틀을 구성하기 위해 타이틀의 비교적 최소한의 실현은 양 부분들(A 및 B)의 두 버전들을 포함한다. 부분(C 및 D)은 타이틀의 머신 독립 코드를 포함한다. 여기서, 부분(C)은 모든 타이틀에 주어지는 타이틀 독립 및 머신 독립 코드를 포함한다. 부분(D)은 타이틀 종속 코드를 포함하고, 타이틀 종속 코드가 생성될 때 총 빌딩 처리(building process)의 부분으로서 생성된다.

<26> 사전을 저장할 때, 부분들(A+B)은 지원되어야 될 각각의 상이한 플랫폼을 위한 하나의 버전이 제1 단일 파일로 함께 제공된다. 이 후 부분들(C+D)은 제 2단일 파일로 기억된다. 상기 언급한 바인딩 메카니즘 및 각 플랫폼을 위한 하나의 섹션, 플랫폼 독립 명령들의 공통 부분 및 플랫폼 종속 명령들의 다중 섹션들을 가지기 때문에, 다양하고 상이한 플랫폼 상에서 타이틀을 실행할 수 있게 하는 이러한 메카니즘이 있다. 이것은 이하에서 더욱 상세히 설명된다.

<27> 부분(A)의 엔트리들이 부분들(C+D)이 기억된 플랫폼상의 디스패치 테이블과 매칭되면, 부분들(C+D)의 기억 체제(storage organization)는 상이한 플랫폼상에서의 로딩을 허용하고, 부분(C+D)의 모든 cfa's 를 부분(B)의 현재 선택된 플랫폼 변형(variant)에 링크한다. 타이틀의 재생 중 부분(A)은 더이상 유효하지 않은 데 그 이유는 그 바인딩 효과가 그것을 불필요하게 만들었기 때문이라는 것을 유의하자. 작은 단점은 포스 인터프리터가 더이상 직접 스레디드될 수 없고, 즉 더이상 모든 사전 엔트리가 머신 명령을 포함하지 않는다는 것이다. 또한, 로딩 및 재배치(relocating)은 약간 더 복잡해진다.

<28> 또한 간접 스레딩(threading)의 이러한 메카니즘은 부분(C 및 D)의 내용들을 플랫폼 운영 체제로부터 독립시키는데에도 이용된다. 이것은 상기 인터프리터에 의해 사용된 시스템 호출들의 일반화(예를 들어, 일어날 수 있는 바이트 교체(swap) 문제들의 해결 및 화일 I/O을 위함) 및 머신 종속 코드의 간접 호출들의 동일 메카니즘을 이용함으로써 실행된다. 커널 부분(B)에서의 시스템 호출들은 커널 부분들(C+D)로 확장된 부분들의 머신 독립 코드로부터 부분(A)의 디스패치 테이블을 통해 야기된다.

<29> 실행은 다음과 같은 세가지 기능들을 사용한다.:

<30> • forth_loader( ), 이 기능은 커널 B 와 확장 커널 C+D 란 이름들로 불리운다. 테이블 A 는 시스템 종속 커널 기능을 확장 커널과 결합하기 위해 사용된다.

<31> • load_context( ), 이 기능은 적당한 루트 문맥(appropriate root context)이란 이름으로 불리운다. 루트 문맥은 애플리케이션의 시작 섹션이고, 초기화 동작 후, 후속 load_context( ) 호출들과 함께 다른 문맥들을 불러낼 것이다.

<32> • forth_execute( ), 이 기능은 실제 인터프리테이션 처리를 시작하고 루트 문맥내에서 첫번째 토큰의 어드레스를 파라미터로 갖는다. 상기 기능은 두 값들을 복귀시킨다. 복귀된 제로 신호는 문맥 내용의 모든 FORTH 워드들이 인터프리트되지 않고 (대신) FORTH 명령 워드 'sleep'이 실행되게 한다. 이 Forth 워드의 실행으로 사용자 영역에 실행 상태를 저장하고, forth_execute( ) 기능의 호출자에게 복귀시킨다. 스레드(thread)는 동일한 파라미터들로 기능 forth_execute( )를 한번 더 호출함으로써 다시 시작할 수 있다. 문맥의 전체 내용이 인터프리트되었을 경우에만 기능 forth_execute( )는 제로가 아닌(non-zero) 값을 돌려줄 것이다. 그 결과, 스택, 복귀 스택, 및 사용자 영역으로 사용된 영역들은 포기되

고 다시 다른 목적으로 사용될 수도 있다.

<33> 프로그램 구현의 설명

<34> 공통 공개 포맷이 어떻게 효과적으로 사용되는지를 설명하고, 이 기술 분야에서 숙련된 사람이 본 발명을 실시할 수 있도록 하기 위해, 부록들이 다양한 예시적인 프로그램 모듈들을 제공한다. 예는 멀티미디어 응용의 짧은 부분인데, 프로세스를 통한 상기 응용의 발전이 최종 사용자(end user) 플랫폼에서 실제 재생을 실현하기 위해 기술된다.

<35> 디스플레이될 상기 응용은 다음과 같다. 즉 우선, 어두운 스크린으로부터 시작해서, 표준 dyuv 색채 포맷으로 정의된 화상이 비디오 효과 와이프클럭(video effect wipeClock)에 의해서 형성된다. 거기에, 미리 정의된 원점에서 시작해서, 스위핑 라인(sweeping line)이 회전되고, 한편 효과적으로 디스플레이되는 필드가 스위프에 의해 커버된 영역에 의해 주어진다. 다음에, 상기 응용은 사용자가 커서를 필드, 핫스폿의 특정 영역으로 움직이고, 마우스와 같은 포인팅 기구상에서 오른쪽 작동 버턴이 눌려 질때까지(button 1 Down) 대기한다. 만약 이것이 발생하면, 상기 응용은 실제 화상으로부터 다른 비디오 효과(wipeBT, 여기서 와이핑 모션(wiping motion)은 화상의 바닥에서 위쪽으로 움직이는 직선이다)를 통해 상이한 표준 포맷(clut7 포맷)으로 코드화되는 다른 화상으로 표시를 변경함으로써 진행된다. 얻어진 화상은 타이머에 의해 제어된 사전 지정된 시간 간격동안 나타난다. 그후, 상기 응용은 종료한다.

<36> 완전한 응용은 CPF-Talk라 불리우는 저작 언어(authoring language)로 프로그램된다. 이 언어로, 그것은 그래픽 디자인들로부터 출력을 표현하기 위해 사용될 수도 있고, 응용을 형성하는 출력은 소위 문맥들로 분할되고, 상기 문맥의 각각은 자신의 특정적인 행동을 나타낼 수도 있다. 다양한 문맥들은 한 문맥으로부터 다른 문맥으로 제어를 전달하는 소위 액션들을 통해 상호 연결되어 있다. 상술된 응용에서, 3개의 다른 문맥들은 구별된다. start_context는 첫번째 화상을 보여주는 것에 관한 것이고, next_context는 다음 화상을 보여주는 것에 관한 것이며, stop_context는 응용의 종료에 관한 것이다. 제어는 사용자가 액션 버튼이 핫스폿에 위치된 작동 버튼을 누를 때 start_context로부터 next-context로 전달된다. 제어는 타이머 출력 신호의 제어하에서 next_context로부터 stop_context로 전달되고, 상기 타이머는 next_context 로의 선행 제어 전달에서 시작한다.

<37> 이들 세 문맥들에는 다른 세 문맥들에 의해 요구되는 다양한 글로벌 오브젝트들을 정의하는 공통 root_context가 부가된다. 상기 root_context는 응용이 시작될 때 첫번째로 호출되는 것이다. 즉 root_context의 모든 명령들이 실행될 때 그것은 제어를 자동적으로 첫번째 non_root context, 이 경우 start_context 로 제어를 넘긴다. 테이블들(1~4)은 상술된 응용에 대한 이들 4 개의 문맥들의 내용들을 나타낸다. 테이블 1에서, 먼저 하나는 시간 간격이고 하나는 효과량(effect quantity)인 두 변수들과 함께, root_context가 나열되었다. 게다가, 두개의 컬러 팔레트들이 정의되었고(그들은 상이할 수 있음) 두개의 표면들이 정의되었다. 상기 표면들은 평면 상에 정의되었고, 적절하다면 그들의 정의된 컬러 코딩, 픽셀 사이즈, 시작 컬러, 및 페이드(fade) 후의 컬러를 얻는다. 최종적으로, root_context는 다음에 행해져야 할 것을 상술한다.

<38> 테이블 2 에서, start_context는 먼저 초기 이미지, 다음의 커서 형상, 핫스폿 위치, 및 핫스폿 위치의 버튼이 적어도 사전 지정된 간격(dur50)동안 작동되었을 때 행해져야 하는 동작을 액세스하는 곳을 상술한다. 최종적으로, 이 문맥으로 들어갈 때 실행되어야하는 동작들이 상술된다. 테이블 3 에서, next_context의 구조는 유사하다. 두드러진 특징은 타이머에 대한 스펙(specification)에 있다. 테이블 4에서, stop_context는 자명하다.

<39> 최종 사용자 플랫폼에서 실제 재생을 향한 프로세스에서의 다음 단계는 이들 4 개의 문맥들을 Forth 언어로 번역하는 것이다. 그 결과는 또한 CPF 스크립트 포맷으로 불리운다. 번역 결과들은 부록들(5~8)에 상술되어 있다. 표준 Forth 사전에서 허용된 것보다 많은 워드들이 이용 가능하기 때문에 어휘가 증가했다. 즉 이러한 확장은 나중에 설명될 것이다. 테이블들(1~4) 및 테이블들(5~8) 각각에서 나타나는 사건간의 일치는 자명하다.

<40> 테이블들(5~8)에 따른 표현(representation)은 인터프리터가 응용을 구축하는 동안 사용하는 포맷과 언어에 있다. 사용자 플랫폼상의 실제 재생 중 더욱 효율적인 포맷이 사용된다. 이러한 간결한 포맷은 파싱 및 어드레스 분해 동작들(parsing and address resolving operations)이 CPF 스크립트 포맷으로 표현되는 문맥상에서 실행된 후에 인터프리터내의 문맥의 내부 표현이다. 이러한 간결한 포맷은 2 진 모듈 포맷으로 불리운다. 거기에, 예를 들어, 기호 참조들(symbolic references)은 사전내의 실제 어드레스 오프셋에 의해 대체된다. 이들 2 진 모듈들은 사람이 읽을 수 없는 포맷으로 되어 있기 때문에 명확하게 하기 위해 부록들에서 생략되었다. 이들 2진 모듈들, 이들이 참조하고 모든 필요한 표준 지원 기능들이 있는 애셋들이 구축 처리를 완성하는 대용량 메모리(mass memory)에 올려진다.

<41> 이하에, 런타임에서의 인터프리터의 동작이 설명된다. 인터프리터는 세부분들로 구성된다. 제 1의 부분 또는 커널은 모든 플랫폼들과 모든 타이틀들에 대해서 동일하다. 종종 확장 커널로 불리우는 제 2 의 부분은 모든 플랫폼에 동일한 특정 응용을 제외하고는 응용 종속적이다. 커널 및 확장 커널은 플랫폼 종속 기능들을 포함하는 제 3 의 부분을 이용하고 참조한다. 암호화된 형식의 이들 기능들은 동일한 물리적 길이들을 가질 필요가 없고, 따라서 사전들내의 오프셋 값들은 플랫폼마다 변할 수 있다. 이로 인해 야기되는 문제를 해결하기 위해, 로딩 중에만 제 4 의 구성요소(component)가 인터프리터에 부가되며, 이것은 실행이 이루어지고 있는 실제 플랫폼에 대해 사전내의 플랫폼 종속 기능들에 대한 오프셋들을 포함한다. 로딩 시, 커널 및 확장 커널로부터 플랫폼 종속 사전으로의 참조들(referrals)은 이러한 테이블로부터 취해진 참조들로 대체된다.

<42> 인터프리테이션 처리의 상기 부분을 설명하기 위해, 명령 'Copy START_image to surface1'이 상세히 설명될 것이다. 우선, 판독을 위해, 포맷이 테이블들(5~8)에 따라 이미 나타내어 졌음을 알아야 한다. 인터프리터 커널은 다음에 따라 명령을 얻는다:

<43> 'START_image~surface1~000 copy~obj' 그리고 확장 커널로부터 스택상의 모든 관련 인자들(arguments)(000 surface1~ and START_image~)과 함께 모듈 copy_obj를 불러낸다. 여기서 틸데(tilde;

~)는 변수를 나타낸다. 본 경우에 있어서, 확장 커널은 단지 두개의 인자, 즉 objects surface1~ 및 START_image 만을 필요로 한다. 확장 커널은 화상을 복사하기 위한 메모리 공간을 가져야만 하고, 그 목적을 위해 운영 체제로부터 이러한 메모리를 요구하기 위해 프로그램 모듈을 불러낸다. 이 단계에서 인터프리터의 로딩 중 상술한 바와 같이 확장 커널에 링크되는 운영 체제 종속 시스템 호출이 일어날 것이다. 메모리의 예약 후에, 다른 운영 체제 종속 모듈들은 대용량 메모리상의 파일을 오픈하기 위해, 파일의 내용을 예약된 메모리 공간에 복사하기 위해, 그리고 대용량 메모리상의 파일을 닫기 위해 불려진다. 후자의 시스템 호출들은 메모리 핸들러에 대해 기술된 것과 동일한 방식으로 확장 커널에 링크된다.

<44> 이러한 방식으로 'copy START_image to surface1'과 같은 문맥에 존재하는 명령은 각기 다른 플랫폼 서브시스템에 대해서 많은 다른 플랫폼 종속 기능(functionality) 뿐만아니라 예를 들면 메모리 핸들링 및 화일 I/O를 위한 각각 다른 기능을 불러낼 수 있다. 다른 명령들은 유사한 방식으로 이와 같은 다른 기능들로 나타날 수 있다.

<45> [표 1]

```
context root
context START_context in "START.cxt"
context next_context in "next.cxt"
context stop_context in "stop.cxt"
   variable dur = 50
   variable effect = 17
   palette palette1
     size    : 128
     offset  : 0
   end palette
   palette palette2
     size    : 128
     offset  : 0
   end palette
   surface surface1
     plane  : A
     coding : clut7
     size   : 768, 560
     color  : 0
     fadeLevel  : 0
  end surface
  surface surface2
     plane  : B
     coding : clut7
     size   : 768, 560
     color  : 0
  end surface
  on entry
     change context to START_context
  end on
end context
```

<47> [표 2]

```
context START_context
   image START_image
      source : "/h0/STRUCTED/VIDEO/DYUV.d"
   end image
   cursor cursor0
      color   : 0, 255, 255
      pattern : 0x7FFE,0xD557,0xAAAF,0x9FFD,\
                0x900D,0x900B,0x900B,0x900D,\
                0x900D,0x900B,0x900B,0x900D,\
                0x9FFD,0xA227,0xC88B,0x7FFE
   end cursor
   hotspot START0_hs
     location  : 329, 233
     size      : 102, 90
     on button1Down
       dur = 50
       change context to next_context
     end on
   end hotspot
  on entry
     show cursor
     set coding of surface1 to dyuv
     copy START_image to surface1
     attach START0_hs to surface1
     attach cursor0 to START0_hs
     surface order surface2, surface1
     display pal
     wipeClock dur surface2 to surface1
     cursorPosition 384, 280
  end on
end context
```

<49> [표 3]

```
context next_context
   image next_image
      source : "/h0/STRUCTED/VIDEO/CLUT7.cl7"
   end image
   timer next0_timer
    interval  : 100
    count   : 0
    on count 1
     effect = 6
     dur = 50
     change context to stop_context
    end on
   end timer
  on entry
    hide cursor
    set coding of surface2 to clut7
    copy next_image to surface2, palette2
    attach palette2 to surface2
    surface order surface1, surface2
    display pal
    wipeBT dur surface1 to surface2
    cursorPosition 384, 280
    start next0_timer
  end on
end context
```

<51> [표 4]

```
context stop_context
  on entry
    exit
  end on
end context
```

<53> [표 5]

```
"" /users/user/home_directory pc-setenv
root-ctxt
context: START_context- START.cxt
context: next_context- next.cxt
context: stop_context- stop.cxt
4 n 50 => dur-
4 n 17 => effect-
palette: palette1-
  128 palette-size!
  0 palette-offset!
palette;
palette: palette2-
  128 palette-size!
  0 palette-offset!
palette;
surface: surface1-
  plane-a plane!
  d-clut7 coding!
  768 560 surface-size!
  0 surface-color!
  0 fadelevel!
surface;
surface: surface2-
  plane-b plane!
  d-clut7 coding!
  768 560 surface-size!
  0 surface-color!
surface;
entry: on-entry
  START_context- 2ctxt
entry;
```

<55> [표 6]

```
local-context: START_context~
  image: START_image~
  r" /h0/STRUCTED/VIDEO/DYUV.d" image-source!
image;
  cursor: cursor0~
  0 255 255 cursor-color!
  bitmap( 32766 54615 43695 40957 36877 36875 36875 36877 36877
36875
  36875 36877 40957 41511 51339 32766 ) pattern!
cursor;
 hotspot: START0_hs~
  329 233 hotspot-location!
  102 90 hotspot-size!
  [[ ev-b0d ]]event: 'z1
    n 50 dur~ v!
    next_context~ 2ctxt
  event;
hotspot;
  entry: on-entry
  show-cursor
  d-dyuv surface1~ coding!
  START_image~ surface1~ 0 0 0 copy-obj
  START0_hs~ surface1~ attach-obj
  cursor0~ START0_hs~ attach-obj
  0 surface2~ surface1~ surf-order
  display_625 display
  dur~ vn@ time-check surface2~ surface1~ $wait wipe-clock
  384 280 cursor-position
entry;
```

<57> [표 7]

```
local-context: next_context~
  image: next_image~
  r" /h0/STRUCTED/VIDEO/CLUT7.cl7" image-source!
image;
  timer: next0_timer~
  100 interval!
  0 count!
  [[ 1 ]]count: 'h]
    n 6 effect~ v!
    n 50 dur~ v!
    stop_context~ 2ctxt
  count;
timer;
  entry: on-entry
  hide-cursor
  d-clut7 surface2~ coding!
  next_image~ surface2~ palette2~ 0 0 copy-obj
  palette2~ surface2~ attach-obj
  0 surface1~ surface2~ surf-order
  display_625 display
  dur~ vn@ time-check surface1~ surface2~ $wait wipe-bt
  384 280 cursor-position
  next0_timer~ start-obj
entry;
```

<59> [표 8]

```
local-context: stop_context~
 entry: on-entry
 cdiforth-exit
entry;
```

(57) 청구의 범위

청구항 1

데이타 표현(presentation)을 수신하는 데이타 처리 시스템으로서, 상기 데이타는 응용 프로그램과 멀티플랫폼 인터프리터 중 적어도 하나를 포함하며, 상기 시스템은,

상기 멀티플랫폼 인터프리터와 상호 작용하도록 배열되며, 상기 응용 프로그램에 의해 구동되는 상기 멀티플랫폼 인터프리터를 실행하기 위한 처리 수단을 포함하는 플랫폼 서브시스템을 더 포함하고, 상기 응용 프로그램은 실행중 상기 처리 수단에 의해 인터프리트되는 소정의 명령 세트에 기초하며,

그에 따라 상기 플랫폼 서브시스템은 소정의 데이타 세트와 상기 소정의 명령 세트를 가진 미리

특정된 추상적 머신(abstract machine)의 예(instance)이며 , 상기 서브시스템은 양적인 최소 필요조건들(quantitative minimum requirements)에 따른 자원 설비들(resource facilities)을 더 포함하는, 데이타 처리 시스템.

청구항 2

제 1 항에 있어서,

상기 데이타를 수신하는 물리적 인터페이스 수단을 갖는, 데이타 처리 시스템.

청구항 3

제 1 항에 있어서,

무선 방식으로 상기 데이타를 수신하는 물리적 부착 수단을 갖는, 데이타 처리 시스템.

청구항 4

제 1 항에 있어서,

상기 플랫폼 서브시스템은 개인용 컴퓨터를 포함하는, 데이타 처리 시스템.

청구항 5

데이타 표현을 수신하는 데이타 처리 시스템에 사용하기 위한 저장 매체로서, 상기 데이타는 응용 프로그램과 멀티플랫폼 인터프리터 중 적어도 하나를 포함하며, 상기 시스템은,

상기 멀티플랫폼 인터프리터와 상호 작용하도록 배열되며, 상기 응용 프로그램에 의해 구동되는 상기 멀티플랫폼 인터프리터를 실행하기 위한 처리 수단을 포함하는 플랫폼 서브시스템을 더 포함하고, 상기 응용 프로그램은 상기 실행하기 위한 상기 처리 수단에 의해 인터프리트되는 소정의 명령 세트에 기초하며,

그에 따라 상기 플랫폼 서브시스템은 소정의 데이타 세트와 상기 소정의 명령 세트를 가진 미리 특정된 추상적 머신의 예이며, 상기 서브시스템은 양적인 최소 필요조건들에 따른 자원 설비들을 더 구비하고,

상기 저장 매체는 멀티플랫폼 인터프리터를 저장하고,

플랫폼 종속 원시어들(platform dependent primitives)의 위치 어드레스들을 포함하는 디스패치 테이블,

실행가능하고 인터프리트가능한 코드를 저장하기 위해 상기 위치 어드레스들에 의해 어드레스가능한 모든 위치들을 포함하는 커널 영역(kernel area), 및

플랫폼 독립적이며 타이틀 독립적인 코드를 포함하는 제 1 영역과 타이틀 종속적인 코드를 포함하는 제 2 영역을 가진 확장 커널 영역을 포함하는, 저장 매체.

청구항 6

멀티플랫폼 인터프리터를 이용하여 제 1 데이타 처리 플랫폼과 상기 제 1 데이타 처리 플랫폼과는 상이한 제 2 데이타 처리 플랫폼 상에서 동일 응용 프로그램의 실행을 가능하게 하는 방법에 있어서,

상기 제 1 및 제 2 데이타 처리 플랫폼들 각각은 소정의 명령 세트와 소정의 데이타 세트를 가진 미리 특정된 추상적인 머신의 소정의 양적인 최소 필요조건들을 만족시키는 자원 설비들과 각각의 처리 수단을 포함하고,

제 1 및 제 2 데이타 처리 플랫폼들은 상기 미리 특정된 추상적인 머신의 각각의 예들이며,

상기 응용 프로그램은 상기 소정의 명령 세트 및 상기 소정의 데이타 세트에 기초하고,

상기 방법은,

상기 제 1 또는 제 2 데이타 처리 플랫폼에 원격 통신들 채널을 통해 상기 응용 프로그램과 상기 멀티플랫폼 인터프리터 중 적어도 하나를 공급하는 단계와,

상기 공급된 데이타 처리 플랫폼 상에서 실행하기 위한 상기 응용 프로그램을 인터프리트하기 위해 상기 공급된 데이타 처리 플랫폼으로 하여금 상기 멀티플랫폼 인터프리터를 사용 가능하게 하는 단계를 포함하는, 방법.

청구항 7

제 6 항에 있어서, 상기 제 1 및 제 2 데이타 처리 플랫폼들은 상이한 하드웨어 배치들(configurations)을 가지는, 방법.

청구항 8

제 6 항에 있어서, 상기 제 1 데이타 처리 플랫폼은 제 1 개인용 컴퓨터를 포함하고 상기 제 2 데이타 처리 플랫폼은 제 2 개인용 컴퓨터를 포함하는, 방법.

청구항 9

제 6 항에 있어서, 상기 제 1 및 제 2 데이타 처리 플랫폼들은 상이한 운영 시스템들을 가지며 상기 멀티플랫폼 인터프리터는 상기 공급된 데이타 처리 플랫폼에서의 라이브러리(library)를 상기 응용

프로그램과 링크하기 위해 상기 공급된 데이타 처리 플랫폼에 의해 이용되는, 방법.

요약

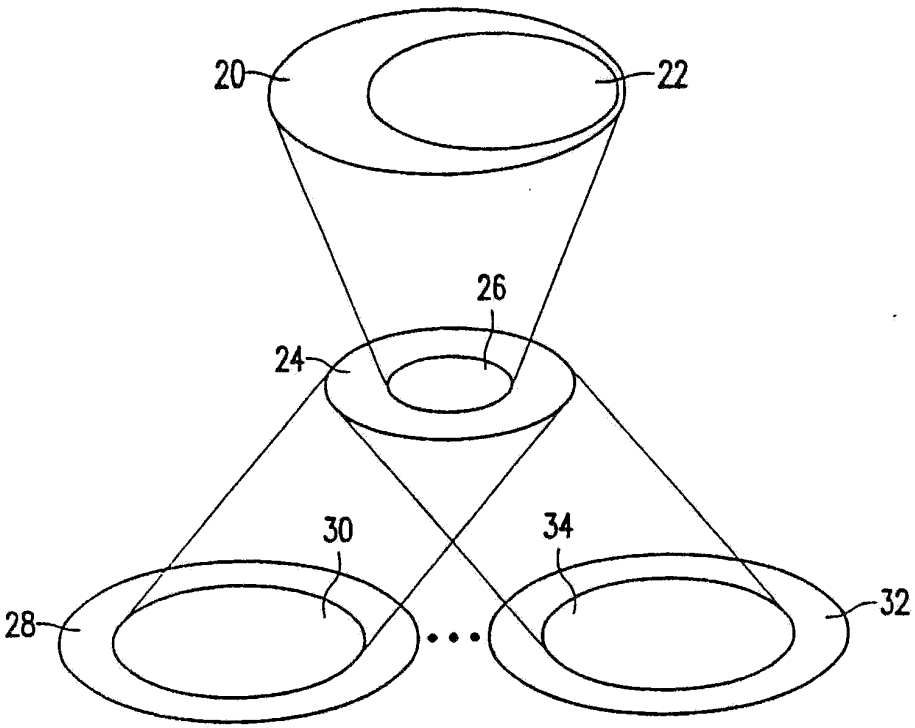
멀티미디어 시스템은 멀티플랫폼 인터프리터를 포함하는 응용 프로그램 타이틀을 구비한다. 응용 프로그램은 멀티미디어 데이타와 스크립팅 언어로 기재된 응용 모두를 포함한다. 플랫폼 서브시스템은 응용 프로그램 타이틀과 상호작용하고 응용 프로그램과 인터프리터를 액세스한다. 그것은 사용자 입력 수단의 제어하에서 응용 프로그램을 실행한다. 또한, 사용자 디스플레이 및 오디오 서브시스템은 응용 프로그램에 의해 제어된다. 응용 프로그램은 멀티플랫폼 인터프리터를 사용하여 처리 수단에 의해 인터프리트되는 명령 세트에 기초한다. 상기 플랫폼은 사전규정된 명령 세트, 데이터 유형 세트, 양적인 최소 필요조건들에 따른 자원 설비들을 갖는 미리 특정된 추상적 머신의 예가 된다.

대표도

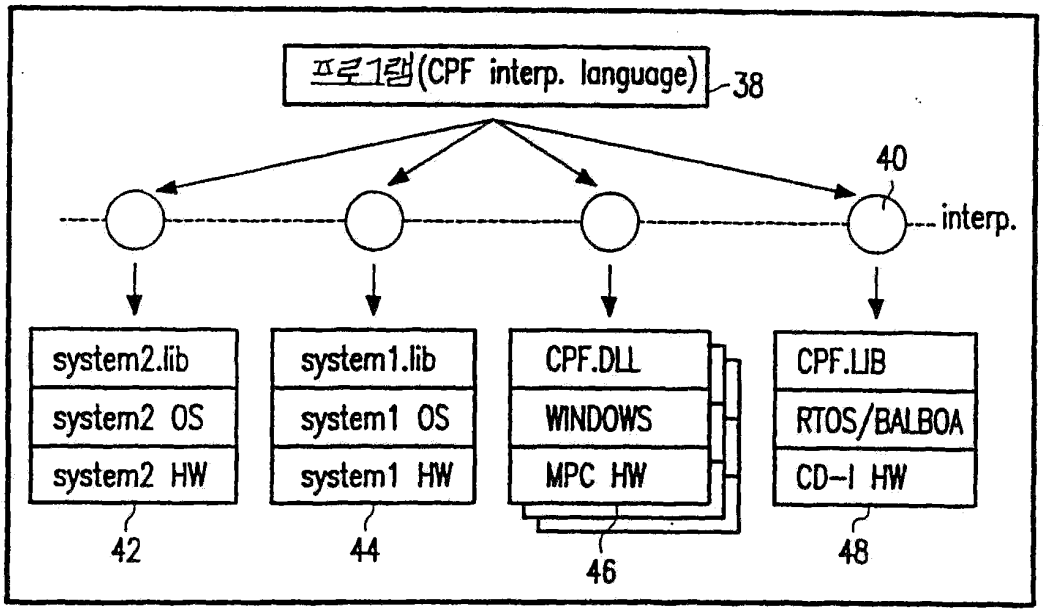
도1

도면

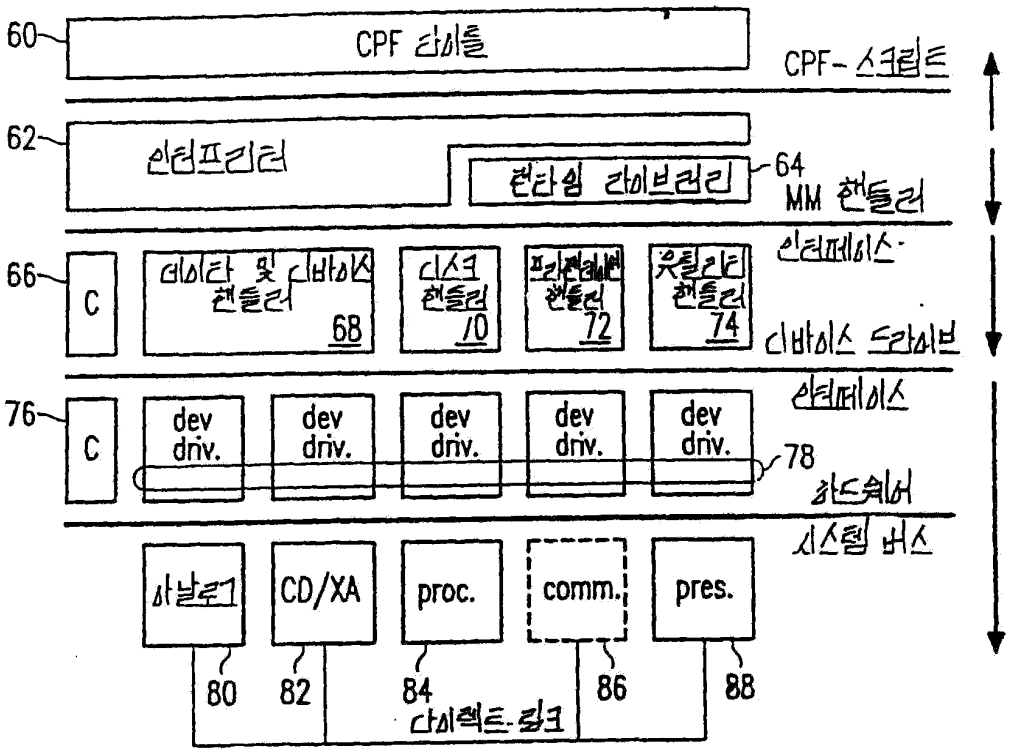
도면1



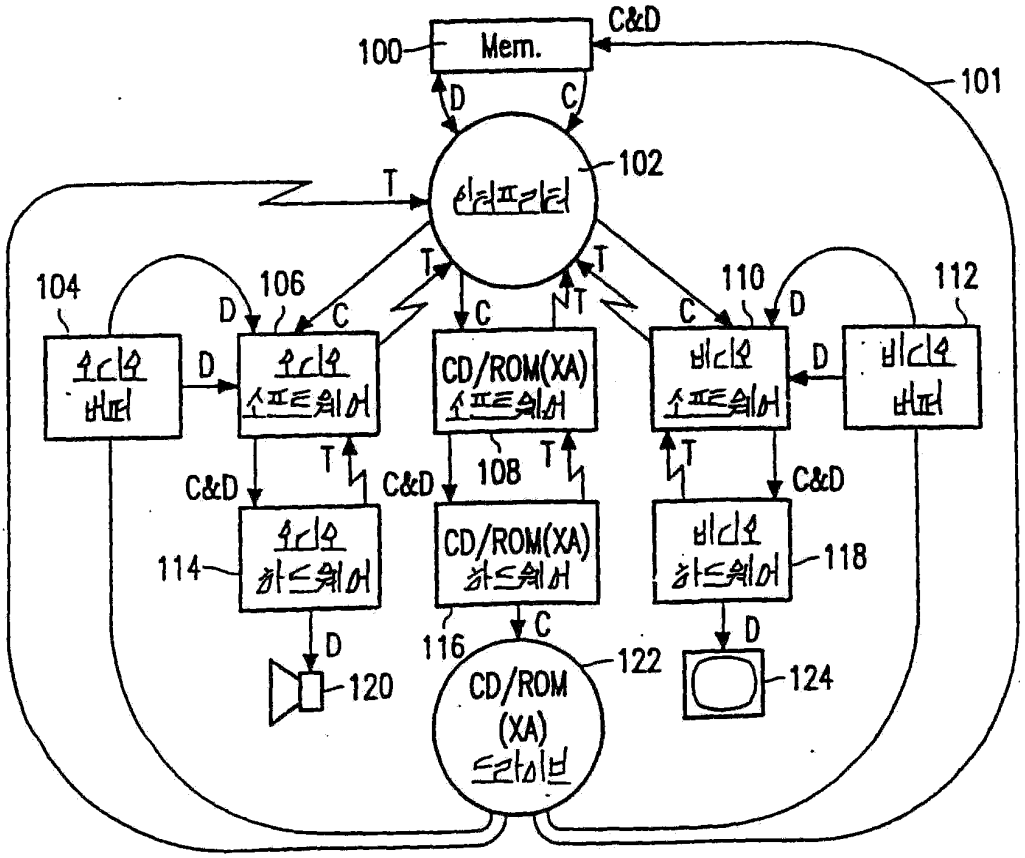
도면2


프로그램(CPF interp. language)
38
40
interp.
system2.lib
system2 OS
system2 HW
42
system1.lib
system1 OS
system1 HW
44
CPF.DLL
WINDOWS
MPC HW
46
CPF.LIB
RTOS/BALBOA
CD-I HW
48

도면3


60
CPF 라이틀
CPF-스크립트
62
인터프리터
런타임 라이브러리
64
MM 핸들러
인터페이스
66
C
데이타 및 디바이스 핸들러
68
디스크 핸들러
70
프리젠테이션 핸들러
72
유틸리티 핸들러
74
디바이스 드라이브
인터페이스
76
C
dev driv.
78
하드웨어
시스템 버스
사날로그
CD/XA
proc.
comm.
pres.
80
82
84
86
88
다이렉트 링크

도면4


100
Mem.
C&D
101
D
C
102
인터프리터
T
104
106
110
112
오디오 버퍼
오디오 소프트웨어
CD/ROM(XA) 소프트웨어
비디오 소프트웨어
비디오 버퍼
108
C&D
114
오디오 하드웨어
CD/ROM(XA) 하드웨어
비디오 하드웨어
118
116
120
122
CD/ROM (XA) 드라이브
124

도면5