(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2014-0067018
(43) 공개일자 2014년06월03일

(51) 국제특허분류(Int. Cl.)
G06F 9/44 (2006.01)
(21) 출원번호 10-2014-7005375
(22) 출원일자(국제) 2011년10월11일
심사청구일자 없음
(85) 번역문제출일자 2014년02월27일
(86) 국제출원번호 PCT/US2011/055700
(87) 국제공개번호 WO 2013/032505
국제공개일자 2013년03월07일
(30) 우선권주장
13/223,291 2011년08월31일 미국(US)

(71) 출원인
마이크로소프트 코포레이션
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
피어슨 해롤드
미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션
렉터 브렌트
미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션
(뒷면에 계속)
(74) 대리인
제일특허법인

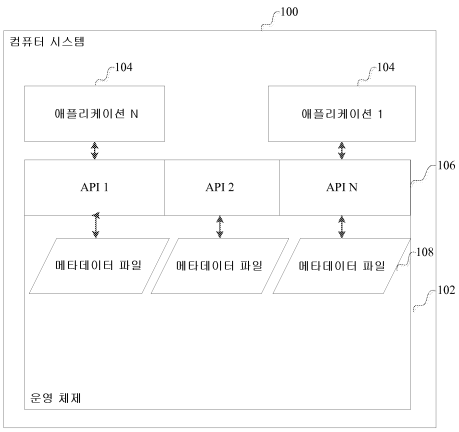
전체 청구항 수 : 총 10 항

(54) 발명의 명칭 **메타데이터를 이용하여 운영 체제의 네이티브 애플리케이션 프로그래밍 인터페이스를 기술하는 기법**

(57) 요 약

네이티브 운영 체제 애플리케이션 프로그래밍 인터페이스(API)는 메타데이터를 사용하여 기술되고 이러한 기술은 알려진 위치에 표준 파일 포맷으로 저장된다. 이러한 메타데이터를 사용하여 API 정의를 저장함으로써, 다른 애플리케이션은 API를 쉽게 식별 및 사용할 수 있다. 이러한 API 표현(representations)을 생성하기 위해, 개발 동안, 개발자는 API에 의해 정의된 클래스, 인터페이스, 메소드, 속성, 이벤트, 파라미터, 구조 및 열거 타입을 포함하는 (그러나 여기에 국한되지는 않는) API의 형태를 기술한다. 이 API 기술은 머신 판독가능 메타데이터 파일을 생성하는 도구에 의해 처리된다. 머신 판독가능 메타데이터 파일은 API 기술과 동일하나, 사람에 의해 작성되기 보다 머신 판독되도록 설계된 포맷을 갖는 정보를 포함한다.

대 표 도 - 도1



(72) 발명자

러벨 마틴

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

프라크리야 마헤쉬

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

로웨 스티븐

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

바수 타사두크

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

울로다크지크 로버트 에이

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

오미야 엘리오트 에이치

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

더니츠 제리

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

홀세크 알레스

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

오스터만 로렌스 더블유

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

젱 웨이

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

와드와 니라즈

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

솔카 샤킬

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

아크시온킨 마이클

미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 엘씨에이 - 인터내셔널 페이턴츠 마이크로소프트 코포레이션

특허청구의 범위

청구항 1

컴퓨팅 머신으로서,

프로세서와,

하나 이상의 컴퓨터 저장 매체와,

상기 컴퓨터 저장 매체에 저장된 컴퓨터 프로그램 명령어들

을 포함하되,

상기 컴퓨터 프로그램 명령어들은 상기 프로세서에 의해 실행되는 경우 상기 프로세서로 하여금 동작들을 수행하도록 하며, 상기 명령어들은 운영 체제를 제공하는 동작을 포함하고, 상기 운영 체제를 통해 애플리케이션이 상기 컴퓨팅 머신의 리소스들을 액세스하고,

상기 운영 체제는, 애플리케이션 프로그램에 의해 액세스가능하여 상기 운영 체제에 의해 구현된 기능에 대한 액세스를 상기 애플리케이션 프로그램에 제공하는 하나 이상의 애플리케이션 프로그래밍 인터페이스를 제공하고,

상기 애플리케이션 프로그래밍 인터페이스 각각은 연관된 메타데이터 파일을 갖고, 상기 메타데이터 파일은 상기 애플리케이션 프로그래밍 인터페이스의 요소들을 머신 판독가능 프로그래밍 언어 독립적인 포맷으로 기술하는,

컴퓨팅 머신.

청구항 2

제1항에 있어서,

상기 메타데이터 파일은 상기 애플리케이션 프로그래밍 인터페이스의 지명된 요소(named elements)에 대해, 이름 공간에 위치한 식별자를 포함하는, 컴퓨팅 머신.

청구항 3

제2항에 있어서,

지명된 요소는 기본 데이터 타입들의 세트 중 하나일 수 있는, 컴퓨팅 머신.

청구항 4

제3항에 있어서,

지명된 요소는 또한 인터페이스, 메소드, 속성 및 이벤트 중 하나일 수 있는, 컴퓨팅 머신.

청구항 5

제3항에 있어서,

지명된 요소는 또한 데이터 구조, 열거 타입 및 어레이 중 하나일 수 있는, 컴퓨팅 머신.

청구항 6

운영 체제에 의해 구현되는 기능에 대한 액세스를 애플리케이션 프로그램에 제공하는 애플리케이션 프로그래밍 인터페이스를 정의하는 데이터 파일을 수신하는 단계와,

상기 애플리케이션 프로그래밍 인터페이스에 대해, 연관된 메타데이터 파일을 생성하는 단계- 상기 메타데이터 파일은 상기 애플리케이션 프로그래밍 인터페이스의 요소들을 머신 판독가능한 프로그래밍 언어 독립적인 포맷으로 기술함 -와,

상기 메타데이터 파일을 상기 운영 체제의 일부로서 저장하는 단계

를 포함하는 컴퓨터로 구현되는 프로세스.

청구항 7

제6항에 있어서,

상기 메타데이터 파일은 상기 애플리케이션 프로그래밍 인터페이스의 지명된 요소에 대해, 이름 공간에 위치한 식별자를 포함하는, 컴퓨터로 구현되는 프로세스.

청구항 8

제7항에 있어서,

지명된 요소는 기본 데이터 타입들의 세트 중 하나일 수 있는, 컴퓨터로 구현되는 프로세스.

청구항 9

제8항에 있어서,

지명된 요소는 또한 인터페이스, 메소드, 속성 및 이벤트 중 하나일 수 있는, 컴퓨터로 구현되는 프로세스.

청구항 10

제8항에 있어서,

지명된 요소는 또한 데이터 구조일 수 있는, 컴퓨터로 구현되는 프로세스.

명 세 서

배 경 기 술

[0001] 운영 체제는 전형적으로 애플리케이션이 그 운영 체제에 의해 지원되는 기능에 액세스할 수 있도록 해주는 몇몇 애플리케이션 프로그래밍 인터페이스를 구비한다. 이러한 API는 전형적으로 컴퓨터 프로그래밍 언어에서 지명된 파일(named file) 또는 객체를 사용하여 운영 체제에 의해 지정된다. 예를 들어, C 프로그래밍 언어는 "interface.h"와 같은 이름을 가질 수 있는 헤더 파일을 사용한다. 이와 유사하게, C#에서, "P/Invoke" 시그너처로 불리는 메카니즘이 사용되어 운영 체제 API에 액세스한다. 운영 체제 API를 이용할 컴퓨터 프로그램을 작성하는 사람은 지명된 API 파일 또는 객체에 대한 참조를 프로그램 내에 포함시키거나, 또는 프로그래밍 언어에 의해 제공된 또 다른 메카니즘을 사용한다. 그 프로그램은 예를 들어 그 API에 의해 사용되는 신택스(syntax) 내에, 그 API에 의해 정의된 기능에 대한 호출을 포함한다.

[0002] 이러한 방식으로 정의된 API는 그들이 작성된 언어와는 다른 언어로 직접 액세스될 수 없다. 다른 언어로 작성된 프로그램에 액세스할 수 있도록 하기 위해, API는 "래핑된다(wrapped)". 이 래핑은 전형적으로 API마다 또한 프로그램마다 수동으로 수행되어야 하며 타겟 언어와 API 및 운영 체제 모두에 대한 깊은 이해를 필요로 한다. 따라서, 다수의 운영 체제 API는 이용가능하지 않다.

발명의 내용

과제의 해결 수단

[0003] 본 요약은 이하의 상세한 설명에서 보다 자세히 설명될 선택적인 개념들을 단순화된 형태로 소개하고자 제공된다. 본 요약은 청구 대상의 필수 특징 또는 핵심 특징을 나타내고자 하는 것이 아니며, 청구 대상의 범위를 한정하기 위해 사용되는 것도 아니다.

[0004] 네이티브(native) 운영 체제 애플리케이션 프로그래밍 인터페이스(API)는 메타데이터를 사용하여 기술되고 이러한 기술은 알려진 위치에 표준 파일 포맷으로 저장된다. 이러한 메타데이터를 사용하여 API 정의를 저장함으로써, 다른 애플리케이션은 API를 쉽게 식별 및 사용할 수 있다.

[0005] 이러한 API 표현(representations)을 생성하기 위해, 개발 동안, 개발자는 API에 의해 정의된 클래스, 인터페이스, 메소드, 속성, 이벤트, 파라미터, 구조 및 열거 타입을 포함하는 (그러나 여기에 국한되지는 않는) API의 형태를 기술한다. 이 API 기술은 머신 판독가능 메타데이터 파일을 생성하는 도구에 의해 처리된다. 머신 판독가능 메타데이터 파일은 API 기술과 동일한 정보를 포함하지만, 사람에 의해 작성되기보다 머신 판독되도록 설계된 포맷으로 되어 있다. 예를 들어, 이 머신 판독가능 메타데이터 파일은 ECMA-335 CLI 포맷으로 저장될 수 있다.

[0006] 운영 체제가 구축되면, 개개의 API 기술 모두는 개개의 메타데이터 파일로 컴파일링된다. 이들 메타데이터 파일은 함께 결합되어 시스템 내에서 이용가능한 모든 API에 대한 종합적인 정보를 제공한다. 이 결합된 메타데이터는 설치를 위한 운영 체제 이미지(즉, 컴파일링된 이진 파일) 내에 포함된다. 예를 들어, 결합된 시스템 메타데이터는 ECMA-335 CLI 포맷으로 일련의 메타데이터 파일에 저장될 수 있지만, 특정 포맷이 본 발명에 중요한 것은 아니다. 이러한 방식으로, 운영 체제 및 그의 API는 자기-기술적이다(self-describing).

[0007] 프로그래밍 언어 독립적인 메타데이터에 의해 API가 완전히 기술되는 운영 체제를 구비함으로써, 언어 투영(language projection)을 구현할 수 있는데, 이 언어 투영은 메타데이터를 판독하고 API를 또 다른 프로그래밍 언어로 구현하는 애플리케이션이다. 예를 들어, 자바스크립트 해석기는 이러한 언어 투영을 포함할 수 있고 자바스크립트 프로그램에 의한 운영 체제 API로의 액세스를 자동으로 제공할 수 있다. 컴파일링된 언어의 프로그램은 이러한 언어 투영을 포함하는 컴파일러에 의해 유사한 액세스를 제공받을 수 있다. 후속 출시에 있어, 운영 체제가 새로운 API를 추가하는 경우, 그 새로운 출시와 함께 배포된 메타데이터는 애플리케이션들이 해석기 또는 컴파일러에 대한 수정없이 새로운 기능의 이점을 즉각 이용할 수 있게 해준다.

[0008] 따라서, 일 측면에서, 컴퓨터의 운영 체제는 컴퓨터 저장 매체에 저장된 컴퓨터 프로그램 명령어들을 포함하고, 이 명령어들은 처리 장치에 의해 처리되는 경우 프로세서로 하여금 운영 체제를 구현하는 동작을 수행하게 한다. 운영 체제는 운영 체제에 의해 구현되는 기능에 대한 액세스를 애플리케이션 프로그램에 제공하기 위해 애플리케이션 프로그램에 의해 액세스가능한 하나 이상의 애플리케이션 프로그래밍 인터페이스를 구비한다. 메타데이터 파일은 애플리케이션 프로그래밍 인터페이스의 지명된 요소들을 머신 판독가능 프로그래밍 언어 독립적인 포맷으로 기술한다. 지명된 요소는 기본 타입, 열거 타입, 구조, 대리자(delegates), 인터페이스, 클래스, 메소드, 속성 및 이벤트와 같은 다양한 데이터 타입 중 임의의 데이터 타입일 수 있거나 그를 포함할 수 있다. 메타데이터 파일은 애플리케이션 프로그래밍 인터페이스의 각 지명된 요소에 대해, 이름 공간에 위치하는 식별자(또는 타입 이름)를 포함할 수 있다.

[0009] 이하의 설명에서, 본 명세서의 일부를 형성하는 첨부한 도면을 참조하며, 이 도면에서는 예시로서 본 발명의 특정 실시예가 도시되어 있다. 다른 실시예들이 이용될 수 있고 본 발명의 범주 내에서 구조적 변경이 행해질 수 있다.

도면의 간단한 설명

[0010] 도 1은 애플리케이션 프로그래밍 인터페이스를 기술하는 메타데이터를 갖는 예시적인 운영 체제의 블록도.

도 2는 애플리케이션 프로그래밍 인터페이스를 기술하는 메타데이터를 이용하여 운영 체제를 구축하는 개발 도구의 예시적인 실시예를 나타내는 데이터 흐름도.

도 3은 메타데이터를 생성하기 위해 API 기술 파일이 어떻게 처리될 수 있는지에 대한 예를 나타내는 흐름도.

도 4는 이러한 시스템이 구현될 수 있는 예시적인 컴퓨팅 장치의 블록도.

발명을 실시하기 위한 구체적인 내용

[0011] 이하의 섹션에서는 이러한 운영 체제가 구현될 수 있는 예시적인 동작 환경이 제공된다.

[0012] 도 1을 참조하면, 컴퓨터 시스템(100)은 다양한 애플리케이션(104)이 실행되는 플랫폼을 컴퓨터 하드웨어(도 4 참조)와 연계하여 제공하는 운영 체제(102)를 포함한다. 애플리케이션들은 운영 체제에 의해 관리되는 프로세스로서 실행되고, 이들 애플리케이션은 파일 등과 같은 운영 체제에 의해 관리되는 컴퓨터 시스템의 리소스를 소비하거나 또는 그 리소스에 대한 액세스를 갖는다.

[0013] 운영 체제는 애플리케이션(104)에 의해 액세스되는 몇몇 애플리케이션 프로그래밍 인터페이스(106)를 제공한다. 이들 API(106)는 하나 이상의 메타데이터 파일(108)에 의해 기술된다. 메타데이터 파일(108)은 운영 체제의 API 또는 API들의 머신 판독가능한, 프로그래밍 언어 독립적인 표현이다. 이하에서 설명되는 바와 같이, 이러한 메타데이터 파일은 API 기술 파일로부터 자동으로 생성될 수 있고, 따라서, 운영 체제의 서피스(surface) 또는 전체 API의 머신 판독가능한, 프로그래밍 언어 독립적인 기술의 자동 생성을 가능하게 한다. 언어 컴파일러 및/또는 해석기(미도시)와 결합된 이러한 메타데이터 파일은 운영 체제 API가 자연스럽고 자동적인 방식으로 프로그래밍 언어에 투영되게 하는 방식으로 애플리케이션(104)이 개발될 수 있도록 해준다.

[0014] 이러한 상황 하에, 이러한 운영 체제의 예시적인 구현이 도 2 및 도 3과 연계하여 보다 자세히 설명될 것이다.

[0015] 도 2에서, 애플리케이션 개발을 위한 예시적인 데이터 흐름이 도시되어 있다. 애플리케이션 프로그래밍 인터페이스 기술 파일(200)은 개발 프로세스, 즉 운영 체제를 구현하는 코드를 작성하는 프로세스 동안 개발자에 의해 정의된다.

[0016] 운영 체제에 대한 코드를 컴퓨터 시스템 상에 설치되는 실행가능물(executables)로 컴파일링하는데 사용되는 빌드 도구(202)는 API 기술 파일을 처리하여 메타데이터 파일(204)을 생성한다. 끝으로, 빌드 도구(202)는 메타데이터 파일들을 하나의 결합된 메타데이터 파일(206)로 결합한다.

[0017] 빌드 도구(202)의 임의의 구현은 API 기술 파일(200)에 대한 임의의 사양(specification) 및 프로그래밍 언어에 의존한다. 이러한 API 기술 파일(200)은 함수 또는 객체 클래스 또는 데이터 구조인 인터페이스를 정의할 수 있고, 이러한 인터페이스에 대해 하나 이상이 메소드, 이벤트, 속성, 파라미터, 데이터 타입, 파라미터 순서, 예외 등을 정의할 수 있다. 빌드 도구는 이러한 API 기술 파일을 파싱하고, 인터페이스의 특징들을 식별하며, 이러한 특징들을 나타내는 데이터를 머신 판독가능한 프로그래밍 언어 독립적인 포맷으로 메타데이터 파일에 저장한다.

[0018] 도 3은 빌드 도구에 의해 수행되는 일반적인 예시적 프로세스를 나타낸다. 빌드 도구는 API 기술 파일을 액세스한다(300). 빌드 도구는 API 기술 파일을 처리하여 클래스, 메소드, 데이터 타입, 속성, 이벤트, 예외 등과 같은 API의 지명된 요소를 식별한다(302). 식별된 지명된 요소는 그의 메타데이터 표현에 맵핑된다(304). 이 메타데이터 표현은 API 기술 파일에 대응하는 메타데이터 파일에 저장된다(306). 단계(308)에서 결정되는 바와 같이 API 기술 파일이 완전히 처리될 때까지, 단계(302) 내지 단계(308)가 반복된다. API 기술 파일을 처리한 후, 또 다른 API 기술 파일(단계(310)에서 결정된 바와 같이 있다면)이 액세스되고(300), 그 파일에 대해 단계(302) 내지 단계(308)가 반복된다. 모든 API 기술 파일에 대한 처리가 완료된 경우, 혼합 메타데이터 파일이 생성된다(312). 예를 들어, 결합된 시스템 메타데이터는 ECMA-335 CLI 포맷으로 일련의 메타데이터 파일에 저장될 수 있지만, 특정 포맷이 본 발명에서 중요한 것은 아니다. 따라서, 이 결합된 메타데이터 파일은 운영 체제의 서피스, 즉 이용가능한 인터페이스의 완전한, 자동적인, 프로그래밍 언어 독립적인 기술을 제공한다.

[0019] 단계(304)에서 사용될 수 있는 예시적인 맵핑이 이제 설명될 것이다. API 내의 지명된 요소를 나타내기 위해 사용되는 메타데이터의 형태와 구조 측면에서 다른 맵핑이 가능함을 이해해야 한다. 이 예시적인 실시예에서, API 기술 파일을 메타데이터에 맵핑하는 것은 먼저 API 기술 파일 내의 지명된 요소들을 식별하는 것을 포함한다. 지명된 요소는 기본 타입, 열거 타입, 구조, 대리자, 인터페이스, 클래스, 메소드, 속성 및 이벤트와 같은 다양한 데이터 타입 중 임의의 데이터 타입일 수 있거나 그를 포함할 수 있다. 메타데이터 파일은 애플리케이션 프로그래밍 인터페이스의 각 지명된 요소에 대해, 이름 공간에 위치하는 식별자(또는 타입 이름)를 포함할 수 있다.

[0020] 이 구현예에서, 모든 지명된 요소는 다양한 데이터 타입 중 임의의 데이터 타입일 수 있거나 또는 그를 포함할 수 있고, 이름공간에 위치한 식별자를 갖는다. 두 개의 지명된 요소는 별도의 이름공간에 있으면 동일한 식별자를 가질 수 있다. API를 지정하는 API 파일이 지명된 요소를 예를 들어 파라미터 또는 구조 필드에 사용하는 경우, 파일의 저자는 완전히 이름공간 검증된 이름(fuly namespace-qualified name) 또는 짧은 식별자를 사용할 수 있다. 짧은 식별자가 사용되는 경우, 그 짧은 식별자를 현재의 이름공간 영역에 첨부함으로써 메타데이터 내에 완전히 이름공간 검증된 이름이 사용된다. 이 메카니즘을 사용하면, 이름이 메타데이터 내에서 모호해지는 경우가 없다. 이름공간 블록이 메타데이터 내에 사용될 수 있고, 이들 블록은 네스트형(nested)이 되어, 블록 내의 모든 지명된 요소에 대해 이름공간을 명시적으로 선언하는 것을 피할 수 있다. 지명된 요소에는 속성이 덧붙여질 수 있다. 예시적인 속성은 버전 번호, 간단한 플래그를 포함하나 여기에 국한되지 않고 또는 파라미터 내에 추가의 정보를 포함할 수 있다.

[0021] 이제 예시적인 표현 세부사항을 참조하면, 기본 타입의 지명된 요소들은 API 기술 파일에 사용되는 식별자가 뒤에 붙는 Boolean, byte, double, float, int, long, short, character, string, guid, handle, error status 등과 같은 기본 타입과 일치하는 키워드에 의해 표현될 수 있다. 예를 들어, "answer"로 불리는 값을 나타내는 Boolean은 "Boolean Answer"로 표현될 수 있다.

[0022] 어레이의 예시적인 표현은 아래와 같다. 이 표현은 "arrary"와 같은 키워드와 그 뒤에 식별자가 붙는 일반적 형태를 갖는다. 이 뒤에는 포인터 및 어레이 내의 요소들의 수인 값들의 쌍이 붙는다. 예를 들어 다음과 같다.

```
array [identifier]
{
[pointer]
[number of elements]
```

[0023] }

[0024] 열거 타입("Enum")의 예시적인 표현은 다음과 같다. 먼저, 이 표현은 열거 타입을 식별하는 키워드 "enum"와 그 뒤에 식별자가 붙는 일반적인 형태를 갖는다. 식별자 뒤에 enum 값들의 모음이 붙는다. 모든 타입과 마찬가지로, enum 식별자는 이들이 포함되어 있는 이름공간 내에서 유일하다. 그러나, enum 값 식별자는 enum 그 자체 내에서만 유일할 수 있다.

```
enum [identifier]
{
 [value 1 identifier] [optional: = value], . . .
 [value n identifier] [optional: = value]
```

[0025] }

[0026] 일 예로서, 플레잉 카드(playing card)의 순위를 사용하면 다음과 같다.

```
enum CardRank
{
 Ace = 1,
 Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King
```

[0027] }

[0028] 간단한 데이터 구조 타입("struct")의 예시적인 표현은 다음과 같다. 먼저, 이 표현은 구조 타입 지정자(specifier)와 그 뒤에 필드들의 모음이 붙는 일반적인 형태를 가지며, 각 필드는 타입 및 식별자를 갖는다. 모든 타입과 마찬가지로, struct 식별자는 이들이 포함되어 있는 이름공간 내에서 유일하다. 그러나, struct 필드 식별자는 struct 그 자체 내에서만 유일할 수 있다.

```
[version([version number])]
struct [struct.identifier]
{
  [field[1] type] [field[1] identifier];
  [field[n] type] [field[n] identifier];
```

[0029] };

[0030] 마우스 이벤트에 대한 증가(arguments)를 위한 strct의 특정 예는 다음과 같다.

```
{
  MouseButtons Buttons;
  unsigned int Clicks;
  unsigned int Delta;
  Point Location;
}
```

[0031]

[0032] 인터페이스의 예시적인 표현은 메소드, 속성 또는 이벤트의 집합이다. 메소드의 구현은 인터페이스를 구현하는 클래스에서 이루어질 수 있다. 공통 속성에 더해, 인터페이스는 지정될 UUID 속성을 사용한다. 인터페이스는 또 다른 인터페이스를 "요구"할 수 있다. 이것은 컴포넌트가 주어진 인터페이스를 구현하는 경우, 모든 "요구된" 인터페이스는 또한 동일한 컴포넌트에 의해 구현됨을 의미한다. 인터페이스를 표현하는 예시적인 그래머(grammar)는 다음과 같다.

```
interface:
 | attributes "interface" IDENTIFIER ':' IDENTIFER requires '{'interface_member_list'}'
requires:
 | <empty>
 | "requires" interface_list
interface_list:
 | IDENTIFIER
 | IDENTIFIER "," interface_list
interface_member_list:
 | <empty>
 | interface_member
 | interface_member interface_member_list
interface_member:
 | method
 | property
 | event
```

[0033]

[0034] 이 예시적인 구현에서, 인터페이스의 표현의 끝에는 메소드, 속성 및/또는 이벤트가 위치한다. 이 예에서, 인터페이스는 제로 또는 보다 많은 파라미터를 이용하고 단일 타입을 반환하는 메소드를 포함할 수 있다. 메소드의 반환 타입은 HRESULT이다. 파라미터는 이름 및 타입을 갖는다. 파라미터는 [in] 또는 [out] 속성으로 마크된다. 임의의 수의 입력 및 출력 파라미터가 존재할 수 있다. 파라미터는 모든 RIDL-지원 포인터 타입에 대해 입력과 출력으로서 마크될 수 있다. 단일 출력 파라미터는 HRESULT 반환 값을 예외로 맵핑하는 언어에 대해 [retval]로 선택적으로 마크될 수 있다. 인터페이스 내의 메소드 이름은 고유하다.

[0035] 또한 이 예시적인 구현에서, 속성은 필드와 유사하게 나타날 수 있지만, 이들을 액세스하는데 사용되는 풋 앤 겟 동작(put and get operations)의 입력 및 출력 파라미터와 연관된다.

[0036] 인터페이스는 이벤트들을 지원하는데, 즉 관심 사건이 일어나는 경우 인터페이스가 이해 당사자에게 통지하는 메카니즘을 지원한다. 표현은 추가 메소드의 사양(specification) 및 제거 메소드의 사양을 포함한다. 추가 메소드는 이벤트 대리자 타입의 입력 파라미터인 제1 파라미터와, EventRegistrationToken 타입의 출력 파라미터인 제2 파라미터를 갖는다. 제거 메소드는 EventRegistrationToken 타입의 입력 파라미터인 제1 파라미터를 갖는다. 이벤트에 대한 이벤트 대리자 타입 그 자체는 이벤트 소스, 즉 이 이벤트를 전송하는 객체에 대한 인터페이스 포인터인 제1 파라미터를 갖는다. 이하에서는 대리자 타입의 MouseEventHandler가 어떻게 사용되어 인터페이스 이벤트를 선언하는지를 보여주는 예가 제공된다.

```
[eventadd] HRESULT MouseMove(
[in] MouseEventHandler *pMouse,
[out] EventRegistrationToken* token);
[eventremove] HRESULT MouseMove(
[in] EventRegistrationToken token).
```

[0037]

[0038] 대리자는 대리자 사양의 시그너처에 일치하는 시그너처를 갖는 단일 메소드 Invoke를 갖는 인터페이스로서 표현될 수 있다. 이 메소드는 단일 반환 타입 및 제로 또는 그보다 많은 파라미터를 갖는다. 대리자의 반환 타입은 HRESULT이다. 파라미터들은 이름 및 타입을 갖는다. 파라미터들은 입력 또는 출력으로서 마킹된다. 임의의 수의 입력 및 출력 파라미터가 존재할 수 있다. 단일 출력 파라미터는 HRESULT 반환 값을 예외로 맵핑하는 언어에 대해 반환 값으로 선택적으로 마크될 수 있다. 대리자를 나타내는 예시적인 그래머는 다음과 같다.

delegate:

[0039] | delegate_attributes "delegate" TYPE_IDENTIFIER IDENTIFIER '(' parameter_list ')";'

[0040] 앞선 설명은 API의 요소들이 프로그래밍 언어 독립적인 메타데이터 내에서 어떻게 표현될 수 있는지에 대한 예시일 뿐임을 이해해야 한다. 다양한 메타데이터 표현들이 사용될 수 있다.

[0041] API가 프로그래밍 언어 독립적인 메타데이터에 의해 완전히 기술되는 운영 체제를 구비함으로써, 메타데이터를 판독하고 API를 또 다른 프로그래밍 언어로 구현하는 애플리케이션인 언어 투영을 구축할 수 있다. 예를 들어, 자바스크립트 해석기는 이러한 언어 투영을 포함할 수 있고 운영 체제 시스템 API로의 자바스크립트 프로그램에 의한 액세스를 자동으로 제공할 수 있다. 컴파일링된 언어의 프로그램은 이러한 언어 투영을 포함하는 컴파일러에 의해 유사한 액세스를 제공받을 수 있다.

[0042] 예시적인 실시예가 설명되었으며, 이러한 시스템이 동작하도록 설계된 예시적인 컴퓨팅 환경이 이제 설명될 것이다. 이하에서는 이 시스템이 구현될 수 있는 적절한 컴퓨팅 환경에 대한 간단하고 일반적인 설명을 제공하려 한다. 시스템은 다수의 범용 또는 전용 컴퓨팅 하드웨어구성으로 구현될 수 있다. 적절할 수 있는 잘 알려져 있는 컴퓨팅 장치들의 예는 개인용 컴퓨터, 서버 컴퓨터, 핸드 헬드 또는 랩탑 장치(예를 들어, 미디어 플레이어, 노트북 컴퓨터, 셀룰러 전화기, PDA, 음성 녹음기), 멀티프로세서 시스템, 마이크로프로세서 기반 시스템, 셋탑 박스, 게임 콘솔, 프로그램가능 소비자 전자기기, 네트워크 PC, 미니컴퓨터, 메인프레임 컴퓨터, 전술한 시스템 또는 장치 중 임의의 것을 포함하는 분산형 컴퓨팅 환경 등을 포함하는 여기에 국한되지 않는다.

[0043] 도 4는 적절한 컴퓨팅 시스템 환경의 예를 나타낸다. 컴퓨팅 시스템 환경은 적절한 컴퓨팅 환경의 하나의 예시일 뿐이며 이러한 컴퓨팅 환경의 사용 또는 기능의 범위에 대해 어떠한 제한도 두려하지 않는다. 또한 컴퓨팅 환경은 예시적인 동작 환경에 도시되어 있는 컴포넌트들 중 임의의 하나 또는 조합에 관해 어떠한 종속성 또는 요구사항도 가지지 않는다.

[0044] 도 4를 참조하면, 예시적인 컴퓨팅 환경은 컴퓨팅 머신(400)과 같은 컴퓨팅 머신을 포함한다. 이의 가장 기본적인 구성에서, 컴퓨팅 환경(400)은 전형적으로 적어도 하나의 처리 장치(402) 및 메모리(404)를 포함한다. 컴퓨팅 장치는 다수의 처리 장치 및 그래픽 처리 장치(420)와 같은 추가의 공동 처리 장치를 포함할 수 있다. 컴퓨팅 장치의 정확한 구성 및 타입에 따라, 메모리(40)는 휘발성(예를 들어, RAM), 비휘발성(예를 들어, ROM, 플래시 메모리 등), 또는 이 둘의 몇몇 조합일 수 있다. 이러한 가장 기본적인 구성은 도 4에서 점선(406)으로 도시되어 있다. 또한, 컴퓨팅 머신(400)은 또한 추가의 특징/기능을 가질 수 있다. 예를 들어, 컴퓨팅 머신(400)은 자기 또는 광학 디스크 또는 테이프를 포함하나 여기에 국한되지 않는 (착탈가능 및/또는 고정된) 추가의 저장부를 포함할 수 있다. 이러한 추가적인 저장부는 도 4에서 착탈가능 저장부(408) 및 고정된 저장소(410)로 도시되어 있다. 컴퓨터 저장 매체는 컴퓨터 프로그램 명령어들, 데이터 구조, 프로그램 모듈 또는 다른 데이터와 같은 정보의 저장을 위한 임의의 방법 또는 기술로 구현되는 휘발성 및 비휘발성, 착탈가능 및 고정된 매체를 포함한다. 메모리(404), 착탈가능 저장부(408) 및 고정된 저장부(410) 모두는 컴퓨터 저장 매체의 예이다. 컴퓨터 저장 매체는 RAM, ROM, EEPROM, 플래시 메모리 또는 다른 메모리 기술, CD-ROM, DVD, 또는 다른 광학 저장부, 자기 카세트, 자기 테이프, 자기 디스크 저장부 또는 다른 자기 저장 장치, 또는 원하는 정보를 저장하는데 사용될 수 있고 컴퓨팅 머신(400)에 의해 액세스될 수 있는 임의의 다른 매체를 포함하나 여기에 국한되지 않는다. 임의의 이러한 컴퓨터 저장 매체는 컴퓨팅 머신(400)의 일부일 수 있다.

[0045] 컴퓨팅 머신(400)은 장치가 다른 장치와 통신할 수 있도록 해주는 통신 연결부(들)(412)를 포함할 수 있다. 통신 연결부(들)(412)는 통신 매체의 일 예이다. 통신 매체는 전형적으로 컴퓨터 프로그램 명령어들, 데이터 구조, 프로그램 모듈 또는 다른 데이터를, 반송파와 같은 변조 데이터 신호 또는 다른 전송 메카니즘으로 전달하고 임의의 정보 전달 매체를 포함한다. "변조된 데이터 신호"란 신호 내의 정보가 인코딩되도록 자신의 특성들 중 하나 이상이 설정되거나 변경되게 하여, 그 신호의 수신 장치의 구성 또는 상태를 변경하는 신호를 의미한다. 예를 들어, 통신 매체는 유선 네트워크 또는 직도 연결과 같은 유선 매체, 및 음향, RF, 적외선 및 다른 무선 매체와 같은 무선 매체를 포함한다.

[0046] 컴퓨팅 머신(400)은 디스플레이, 키보드, 마우스, 펜, 카메라, 터치 입력 장치 등과 같은 다양한 입력 장치(들)(414) 구비할 수 있다. 스피커, 프린터 등과 같은 출력 장치(들)(416)가 또한 포함될 수 있다. 이들 장치 모두는 당업계에 잘 알려져 있어 더 이상 설명할 필요는 없다.

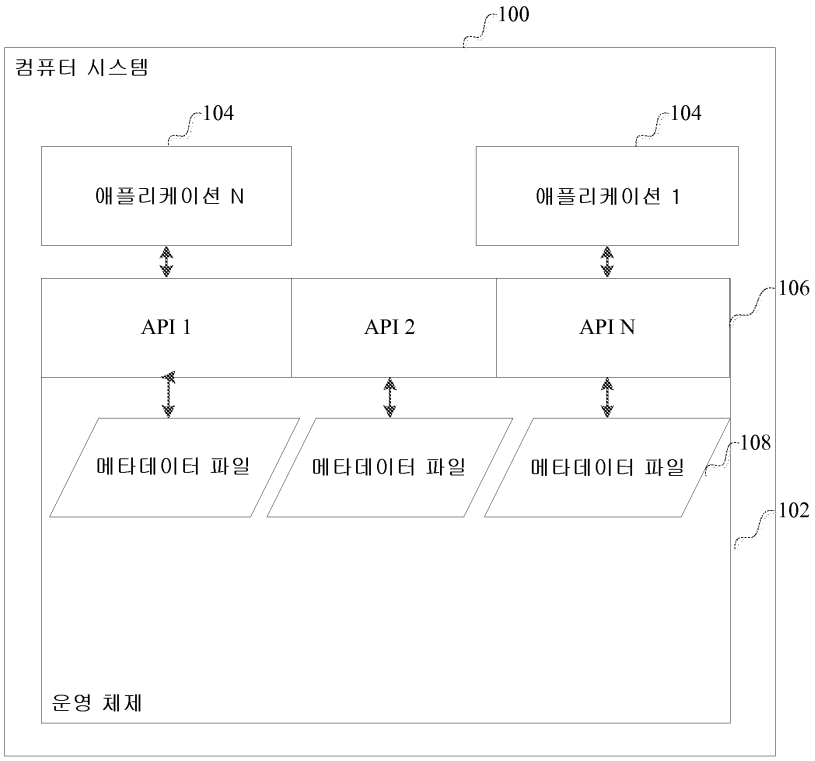
[0047] 애플리케이션 프로그래밍 인터페이스를 기술하는 메타데이터를 갖는 이러한 운영 체제는 컴퓨팅 머신에 의해 처리되는 프로그램 모듈과 같은 컴퓨터 실행가능 명령어들 및/또는 컴퓨터 해석되는 명령어들을 포함하는 소프트웨어의 일반적인 문맥에서 구현될 수 있다. 일반적으로, 프로그램 모듈은 처리 장치에 의해 처리되는 경우 처리 장치로 하여금 특정 작업을 수행하거나 또는 특정 추상 데이터 타입을 구현하도록 하는 루틴, 프로그램, 객체, 컴포넌트, 데이터 구조 등을 포함한다. 이 시스템은 작업들이 통신 네트워크를 통해 링크되는 원격 처리 장치에 의해 수행되는 분산형 컴퓨팅 환경에서 실시될 수 있다. 분산형 컴퓨팅 환경에서, 프로그램 모듈은 메모리 저장 장치를 포함한 로컬 및 원격 컴퓨터 저장 매체 모두에 위치할 수 있다.

[0048] 첨부된 청구항의 서두에 기재된 "제조 물품", "프로세스", "머신" 및 "물질의 구성"이라는 용어는 35 U.S.C. § 101에서 이들 용어를 사용함으로써 정의된 특허가능 청구대상의 범주 내에 속하는 것으로 간주되는 청구대상으로 청구항을 한정하려 한다.

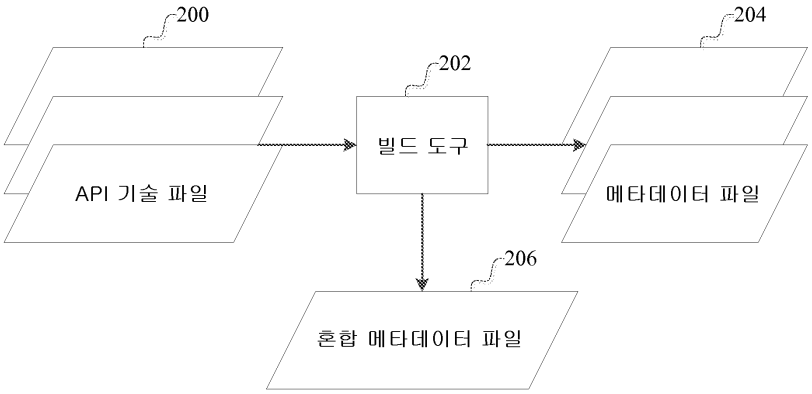
[0049] 본 명세서에서 기술된 전술한 교번적인 실시예들 중 임의의 또는 모든 실시예는 추가의 혼합 실시예를 형성하도록 요구되는 임의의 조합으로 사용될 수 있다. 첨부한 청구항에 정의된 청구대상은 전술한 특정 실시예에 반드시 국한될 필요는 없음을 이해해야 한다. 전술한 특정 실시예는 단지 예로서 개시되었다.

도면

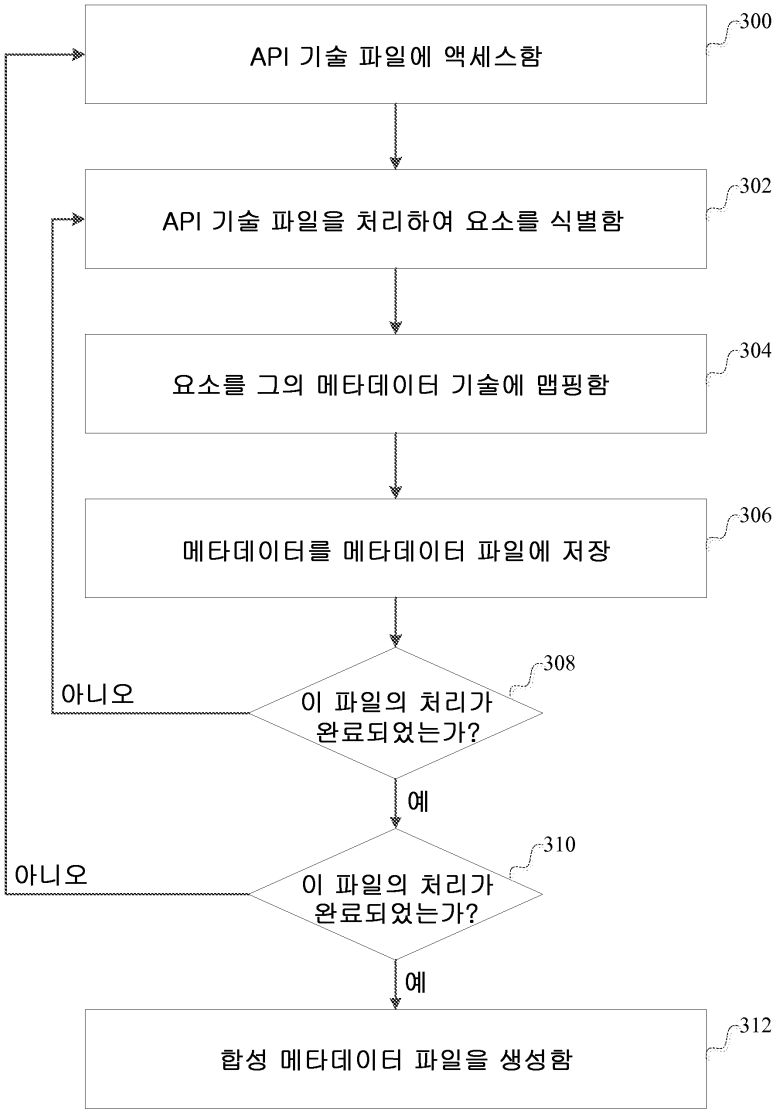
도면1



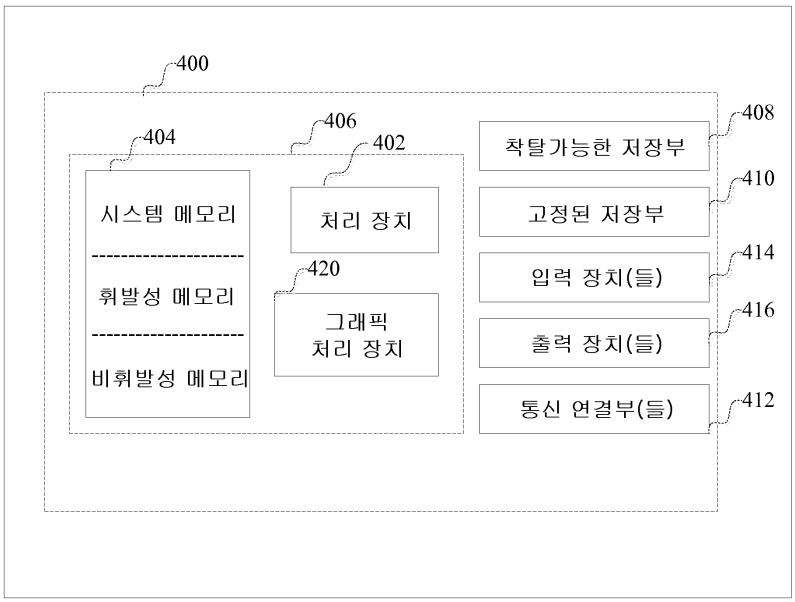
도면2


200
202
204
API 기술 파일
빌드 도구
메타데이터 파일
206
혼합 메타데이터 파일

도면3


API 기술 파일에 액세스함
300
API 기술 파일을 처리하여 요소를 식별함
302
요소를 그의 메타데이터 기술에 맵핑함
304
메타데이터를 메타데이터 파일에 저장
306
이 파일의 처리가 완료되었는가?
308
아니오
예
이 파일의 처리가 완료되었는가?
310
아니오
예
합성 메타데이터 파일을 생성함
312

도면4


400
404
406
402
408
410
414
416
412
420
시스템 메모리
휘발성 메모리
비휘발성 메모리
처리 장치
그래픽 처리 장치
착탈가능한 저장부
고정된 저장부
입력 장치(들)
출력 장치(들)
통신 연결부(들)