(19)대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl.
G06F 9/38 (2006.01)
G06F 9/00 (2006.01)
G06F 3/00 (2006.01)
G06F 9/06 (2006.01)

(11) 공개번호 10-2007-0009568
(43) 공개일자 2007년01월18일

(21) 출원번호 10-2006-7017143
(22) 출원일자 2006년08월25일
심사청구일자 2006년08월25일
번역문 제출일자 2006년08월25일
(86) 국제출원번호 PCT/US2005/014557
국제출원일자 2005년04월27일
(87) 국제공개번호 WO 2005/104772
국제공개일자 2005년11월10일

(30) 우선권주장
11/115,403 2005년04월27일 미국(US)
60/565,851 2004년04월28일 미국(US)
60/603,251 2004년08월23일 미국(US)
60/628,557 2004년11월18일 미국(US)
60/639,805 2004년12월29일 미국(US)

(71) 출원인
후지쯔 가부시끼가이샤
일본국 가나가와켄 가와사키시 나카하라꾸 가미고다나카 4초메 1-1

(72) 발명자
라브로우 야니스
미국 메릴랜드주 21201 볼티모어 #410 웨스트 프랫트 스트리트519
마스오우카 류스케
미국 메릴랜드주 20854 포토맥 와일드 올리브 드라이브 8602
후인 두이
미국 노스캐롤라이나주 27606 롤리 필드스프링 레인 101
송 제수안
미국 메릴랜드주 20705 벨트스빌 아파트먼트 201 에반스 트레일11356

(74) 대리인
김태홍
송승필

전체 청구항 수 : 총 60 항

(54) 태스크 컴퓨팅

(57) 요약

시스템을 프레젠테이션 계층, 원격 절차 호의 프로그래밍 인터페이스(API), 프렌젠테이션 계층에서 컴퓨터 구현형 태스크 인터페이스를 실시간에서 동적으로 발생시키기 위해 원격 절차 호 API를 통해 의미론적으로 컴퓨터 시스템상의 서비스로서 설명되는 펑크션의 소스로 인터페이싱하는 미들웨어 계층, 및 미들웨어 계층이 인터페이싱하는 의미론적으로 컴퓨터

시스템상의 서비스로서 설명되는 펑크션의 소스를 제공하는 서비스 계층과 펑크션 소스 실현 계층의 복수개 구현층들로 분할하고, 의미론적으로 설명되는 애플리케이션-, 장치- 및 서비스-리치 컴퓨터에 기초해, 컴퓨터상의 하나 이상 서비스들에 대한 프레젠테이션 계층에서 발생된 태스크 인터페이스를 사용해 하나 이상 서비스들을 포함하는 실행 가능 태스크를 실시간에서 동적으로 구성하는 것에 의한 태스크 컴퓨팅 컴퓨터 시스템.

대표도

도 1a

특허청구의 범위

청구항 1.

펑크션들의 복수개 컴퓨터 시스템 소스들을 컴퓨터 시스템상의 서비스들로서 의미론적으로 설명하는 단계;

컴퓨터 시스템을,

프레젠테이션 클라이언트 프로세싱 계층,

원격 절차 호 API(Application Programming Interface),

의미론적으로 컴퓨터 시스템상의 서비스로서 설명되는 펑크션의 컴퓨터 시스템 소스로의 프레젠테이션 계층에서 컴퓨터 구현형 태스크 인터페이스를 실시간에서 동적으로 발생시키기 위해 상기 원격 절차 호 API를 통해 상기 프레젠테이션 계층이 인터페이싱하는 미들웨어 서버 프로세싱 계층, 및

의미론적으로 상기 미들웨어 프로세싱 계층이 인터페이싱하는 상기 컴퓨터 시스템상의 서비스로서 설명되는 펑크션의 컴퓨터 시스템 소스를 제공하는 서비스 계층과 펑크션 소스 실현 계층을 포함하는 복수개의 프로그램 가능한 컴퓨터 시스템 구현층들로 분할하는 단계; 및

상기 컴퓨터 시스템상의 하나 이상 서비스들로의 상기 프레젠테이션 계층에서 발생된 태스크 인터페이스에 따라, 하나 이상 서비스들을 포함하는 실행 가능 태스크를 실시간에서 동적으로 구성하는 단계를 포함하는 태스크 컴퓨팅 방법.

청구항 2.

제 1 항에 있어서,

상기 서비스 계층은 상기 펑크션 소스 실현 계층으로부터의 서비스 펑크션 및 상기 펑크션 소스 실현 계층의 서비스 펑크션을, 상기 컴퓨터 시스템상의 서비스로서, 의미론적으로 대응되게 설명하는 의미 서비스 설명을 포함하는 것인 태스크 컴퓨팅 방법.

청구항 3.

제 1 항에 있어서,

상기 원격 절차 호 API는 웹 서비스들에 따르는 것인 태스크 컴퓨팅 방법.

청구항 4.

제 1 항에 있어서,

상기 컴퓨터 시스템은 네트워크 통신의 복수개 컴퓨터 시스템들을 포함하고,

상기 원격 절차 호 API는 원격 웹 서비스들의 API를 포함하는 웹 서비스들에 따르며,

상기 방법은,

제 1 컴퓨터 시스템에 의해, 상기 제 1 컴퓨터 시스템의 원격 웹 서비스들의 API를 제 2 컴퓨터 시스템의 네트워크를 통해 등록하는 단계;

상기 제 2 컴퓨터 시스템에 의해, 상기 제 2 컴퓨터 시스템상의 서비스를 발견하는 단계; 및

상기 제 2 컴퓨터 시스템에 의해, 상기 등록된 원격 웹 서비스들의 API를 통해, 상기 제 1 컴퓨터 시스템에 상기 제 2 컴퓨터 시스템의 발견된 서비스들을 통지하고, 그에 의해, 교차-컴퓨터 시스템 서비스 발견을 제공하는 단계를 더 포함하는 것인 태스크 컴퓨팅 방법.

청구항 5.

제 4 항에 있어서,

상기 등록된 원격 웹 서비스들의 API에서 필터를 특정하는 것에 따라 상기 제 2 컴퓨터 시스템에서의 서비스들의 발견을 필터링하는 단계를 더 포함하는 태스크 컴퓨팅 방법.

청구항 6.

제 5 항에 있어서,

상기 필터는 RDQL(Resource Description Framework Data Query Language)에 따른 쿼리인 것인 태스크 컴퓨팅 방법.

청구항 7.

제 1 항에 있어서,

상기 펑크션의 컴퓨터 소스에 대한 의미 설명에서 자연어에 따른 서비스 명칭을 정의함으로써, 자연어 문장에 따른 하나 이상 서비스들을 포함하는 상기 실행 가능 태스크의 구성을 지원하는 단계를 더 포함하는 태스크 컴퓨팅 방법.

청구항 8.

제 1 항에 있어서,

온톨로지(ontoloty)에 따라 상기 펑크션의 컴퓨터 소스에 대한 의미 설명의 펑크션 특징을 상기 서비스로서 정의하는 단계를 더 포함하고,

상기 실행 가능 태스크는 각 서비스의 펑크션 특징에 따라 호환 가능한 하나 이상 서비스들의 구성인 것인 태스크 컴퓨팅 방법

청구항 9.

제 1 항에 있어서,

상기 미들웨어 서버 프로세싱 계층은 서비스 발견 유닛, 서비스 구성 유닛, 서비스 실행 유닛, 서비스 관리 유닛 또는, 상기 원격 절차 호 API를 통해 상기 프레젠테이션 계층에서의 상기 컴퓨터 구현형 태스크 인터페이스의 실시간 동적 발생을 지원하도록 액세스 가능한, 이들 유닛에 관한 임의 조합들의 프로그램된 유닛들을 포함하는 것인 태스크 컴퓨팅 방법.

청구항 10.

제 9 항에 있어서,

상기 컴퓨터 시스템은 네트워크 통신의 복수개 컴퓨터 시스템들을 포함하고,

상기 방법은,

상기 미들웨어 서버 프로세싱 계층의 프로그램된 제 1 유닛을 제 1 컴퓨터 시스템에 원격 미들웨어 서버 프로세싱 계층으로서 설치하는 단계;

상기 프로그램된 제 1 유닛의 제 1 원격 절차 호 API를 상기 제 1 컴퓨터 시스템에 게시하는 단계; 및

제 2 컴퓨터 시스템의 프로그램된 제 2 유닛에 의해, 상기 게시된 제 1 원격 절차 호 API를 통해, 상기 제 1 컴퓨터 시스템의 서비스를 관리하기 위해 상기 제 1 컴퓨터 시스템의 프로그램된 제 1 유닛을 호출하는 단계를 더 포함하는 것인 태스크 컴퓨팅 방법.

청구항 11.

제 10 항에 있어서,

상기 제 1 컴퓨터 시스템의 서비스를 관리하는 단계는 상기 서비스를 위한 펑크션의 컴퓨터 시스템 소스에 대한 의미 설명을 통해 상기 제 1 컴퓨터 시스템의 상기 서비스를 발견하는 단계, 실행하는 단계, 조작하는 단계, 또는 이들 단계에 관한 임의 조합들을 포함하는 것인 태스크 컴퓨팅 방법.

청구항 12.

제 10 항에 있어서,

상기 제 1 컴퓨터 시스템의 서비스를 관리하기 위해 상기 제 1 컴퓨터 시스템의 프로그램된 제 1 유닛을 호출하는 단계는,

파일 시스템, 멀티-캐스트, 커뮤니티 디렉토리, 개방 의미 서비스 디렉토리, 또는 이들에 관한 임의 조합들의 하나 이상 발견 메커니즘들에 따라, 상기 서비스를 위한 펑크션의 컴퓨터 시스템 소스에 대한 의미 설명을 통해 서비스를 발견하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 13.

제 10 항에 있어서,

상기 제 1 컴퓨터 시스템의 서비스를 관리하기 위해 상기 제 1 컴퓨터 시스템의 프로그램된 제 1 유닛을 호출하는 단계는,

상기 서비스를 위한 펑크션의 컴퓨터 시스템 소스에 대한 의미 설명에서, 공백, 전용, 서브넷에 의한 그룹, 관심에 의한 그룹, 또는 이들에 관한 임의 조합들의 범위들로부터 발견 범위를 설정하는 단계; 및

상기 서비스를 위한 상기 펑크션의 컴퓨터 시스템 소스에 대한 상기 의미 설명에서 설정된 상기 발견 범위에 따라, 상기 서비스를 위한 상기 펑크션의 컴퓨터 시스템 소스에 대한 상기 의미 설명을 통해 상기 서비스를 발견하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 14.

제 1 항에 있어서,

상기 펑크션의 상기 컴퓨터 시스템 소스는 OWL-S[OWL(Web Ontology Language) based Web service ontology] 언어에 따라 의미론적으로 설명되는 것인 태스크 컴퓨팅 방법.

청구항 15.

제 1 항에 있어서,

상기 펑크션의 컴퓨터 시스템 소스에 대한 의미 설명은 구어(spoken language)에 따라 컴퓨터 사용자 인터페이스를 제어하는 구어 속성을 포함하는 것인 태스크 컴퓨팅 방법.

청구항 16.

제 15 항에 있어서,

상기 컴퓨터 사용자 인터페이스는 컴퓨터 디스플레이형 그래픽 사용자 인터페이스이고, 상기 구어 속성은 서비스 명칭 및 서비스 설명의 디스플레이된 설명을 상기 구어에 따라 제어하는 것인 태스크 컴퓨팅 방법.

청구항 17.

제 1 항에 있어서,

상기 펑크션의 컴퓨터 시스템 소스에 대한 의미 설명은 완화된 입력 데이터형을 위한 설명들, 서비스로서의 펑크션의 컴퓨터 시스템 소스에 대한 위치, 관리 펑크션 또는 이들에 관한 임의 조합들을 포함하는 것인 태스크 컴퓨팅 방법.

청구항 18.

제 1 항에 있어서,

상기 컴퓨터 시스템상의 서비스로서의 펑크션의 컴퓨터 시스템 소스에 대한 의미 설명은, 입력 데이터의 정확한 도메인을 식별하는 제 1 입력 데이터형 및 상기 제 1 입력 데이터형의 도메인보다 큰 도메인의 완화된 입력 데이터형을 식별하는 제 2 입력 데이터형과 같은, 서비스를 위한 2 이상의 입력 데이터형들을 식별하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 19.

제 1 항에 있어서,

상기 서비스로서의 펑크션의 컴퓨터 시스템 소스에 대한 의미 설명은 상기 서비스를 위한 관리 펑크션을 위한 설명, 상기 서비스를 위한 관리 펑크션을 위한 설명으로의 링크, 또는 양자 모두를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 20.

제 19 항에 있어서,

상기 관리 펑크션은 서비스를 삭제하는 프로세스, 서비스의 핸들링된 의미 오브젝트를 리턴하는 프로세스, 및 상기 서비스에 대한 사용자 인터페이스, 상기 서비스의 상태 정보, 상기 서비스의 위치, 또는 이들에 관한 임의 조합들을 제시하는 프로세스의 하나 이상 서비스 프로세스들을 포함하는 것인 태스크 컴퓨팅 방법.

청구항 21.

제 1 항에 있어서,

상기 펑크션의 의미론적으로 설명된 컴퓨터 시스템 소스로의 상기 컴퓨터 구현형 인터페이스는 컴퓨터 디스플레이형 그래픽 사용자 인터페이스, 음성 명령 사용자 인터페이스, 무선 장치 사용자 인터페이스, 또는 이들에 관한 임의 조합인 것인 태스크 컴퓨팅 방법.

청구항 22.

제 1 항에 있어서,

상기 펑크션의 의미론적으로 설명된 컴퓨터 시스템 소스로의 컴퓨터 구현형 인터페이스는 음성 명령 사용자 인터페이스이고,

상기 방법은,

자연어에 따른 서비스 명칭을 상기 펑크션의 컴퓨터 소스에 대한 의미 설명에서 정의하는 단계,

상기 정의된 자연어 서비스 명칭들에 따라 음성 인식 상태도를 발생시키는 단계,

음성 인식기에 의해 자연 구어 문장을 정의된 구어 서비스 명칭들 중 하나 이상을 포함하는 태스크의 구성으로서 인식하는 단계,

상기 인식된 구어체의 정의된 서비스 명칭들을, 상기 음성 인식 상태 문법도에 따라, 서비스들의 시퀀스에 매칭하는 단계, 및

상기 매칭된 서비스들을 실행하는 것에 의해 상기 태스크를 실행하는 단계를 더 포함하는 것인 태스크 컴퓨팅 방법.

청구항 23.

제 22 항에 있어서,

상기 컴퓨터 디스플레이형 그래픽 사용자 인터페이스는 서비스의 공간적 프레젠테이션인 것인 태스크 컴퓨팅 방법.

청구항 24.

제 23 항에 있어서,

상기 서비스의 공간적 프레젠테이션은,

영역 맵을 디스플레이하는 단계; 및

상기 영역 맵상에, 발견된 서비스들을 공간적으로 디스플레이하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 25.

제 1 항에 있어서,

상기 펑크션들의 복수개 컴퓨터 시스템 소스들을 서비스들로서 의미론적으로 설명하는 단계는,

온톨로지를 사용해 펑크션의 컴퓨터 시스템 소스에 대한 펑크션 특징들을 설명하는 단계, 및

상기 서비스들의 구성 가능성을 자연어 요소들의 자연어 문장으로서의 구성 가능성으로 매핑하는 것을 지원하기 위해, 상기 서비스에 자연어 문장의 요소로서 명칭을 할당하는 단계를 포함하고,

상기 컴퓨터 구현형 태스크 인터페이스의 발생은,

이용 가능한 서비스들을 발견하는 단계, 및

사용자가 자연어 문장에 따른 상기 자연어 서비스 명칭들에 기초해 상기 복수개 호환 가능 서비스들의 실행 가능 태스크를 구성하는 것을 지원하기 위해, 온톨로지를 사용해 펑크션의 컴퓨터 시스템 소스에 대한 펑크션 특징을 설명하는 단계에 기초해, 서비스의 펑크션 특징에 따라 호환 가능 서비스들을 식별하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 26.

제 25 항에 있어서,

상기 컴퓨터 구현형 태스크 인터페이스는 컴퓨터 디스플레이형 스크린의 그래픽 사용자 인터페이스이고,

상기 그래픽 사용자 인터페이스는,

제 1 그래픽 사용자 인터페이스 윈도에 발견된 서비스들의 선택 가능한 그래픽 디스플레이들을 트리 구조에 따라 디스플레이하는 단계;

프로세스에 따라 상기 하나 이상 서비스들의 태스크로의 실시간 동적 구성을 지원하는 제 2 그래픽 사용자 인터페이스를 제시하는 단계로서, 상기 프로세스는,

사용자에 의해 상기 제 1 윈도에서 발견된 서비스를 선택하는 단계;

상기 선택된 발견 서비스와 관련하여 다른 호환 가능 서비스들의 선택 가능한 그래픽 디스플레이를 자동적으로 디스플레이하는 단계;

사용자에 의해 호환 가능 서비스를 선택하는 단계; 및

상기 제 2 그래픽 사용자 인터페이스 윈도에 상기 발견된 서비스 및 상기 호환 가능 서비스들의 상기 사용자 선택에 따른 방향성 서비스 그래프를 상기 태스크로서 실시간에서 동적으로 디스플레이하는 단계

를 포함하는 것인 제 2 그래픽 사용자 인터페이스 제시 단계; 및

상기 태스크를 실행하기 위해 태스크 실행의 선택 가능한 그래픽 디스플레이를 디스플레이하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 27.

제 26 항에 있어서,

상기 디스플레이된 발견 서비스들은 상기 발견된 서비스들의 펑크션 특징들에 따라 상기 제 1 그래픽 사용자 인터페이스 윈도에서 편성되는 것인 태스크 컴퓨팅 방법.

청구항 28.

제 1 항에 있어서,

서비스로의 액세스는 클라이언트에 의해 제공되는 사실들의 서비스 액세스 제어 요소들, 상기 서비스 전용 정책, 공유 정책, 온톨로지들, 또는 그것에 관한 임의 조합들에 따라 제어되는 것인 태스크 컴퓨팅 방법.

청구항 29.

제 1 항에 있어서,

상기 프레젠테이션 클라이언트 프로세싱 계층은 인스턴스 생성기, 인스턴스 복사기, 인스턴스 인터셉터, 인스턴스 저장기, 특성 선택기, 또는 이들에 관한 임의 조합들의 내부 서비스들을 제공하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 30.

제 1 항에 있어서,

의미 인스턴스 직렬화 서비스, 정보 제공 서비스, 센서 서비스, 스냅샷 서비스, OWL(Web Ontology Language) 포매터 서비스, 및 텍스트 포매터 서비스를 의미론적으로 설명되는 펑크션들의 상기 복수개 컴퓨터 시스템 소스들로서 제공하는 단계를 더 포함하는 태스크 컴퓨팅 방법.

청구항 31.

제 30 항에 있어서,

상기 의미 인스턴스 직렬화 서비스는 임의 유형의 의미 인스턴스를 소비하는 단계, 상기 의미 인스턴스를 정보로 직렬화하는 단계, 및 상기 정보를 사용자에게 제공하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 32.

제 30 항에 있어서,

상기 정보 제공 서비스는 펑크션의 컴퓨터 시스템 소스로부터 의미 오브젝트를 출력하는 단계를 포함하고,

상기 펑크션의 컴퓨터 시스템 소스는 펑크션의 정보 제공 소스인 것인 태스크 컴퓨팅 방법.

청구항 33.

제 32 항에 있어서,

상기 펑크션의 정보 제공 서비스는 시간, 날짜, 관련 날씨, 도로 통행량, 또는 이들에 관한 임의 조합들을 제공하는 것인 태스크 컴퓨팅 방법.

청구항 34.

제 30 항에 있어서,

상기 스냅샷 서비스는,

이미지 장치로부터 스틸 이미지를 캡처하는 단계; 및

상기 스틸 이미지를 위한 의미 오브젝트를 출력하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 35.

제 30 항에 있어서,

상기 OWL 포매터 서비스는,

의미 오브젝트를 입력으로서 수용하는 단계;

상기 의미 오브젝트를 인간이 이해 가능한 방식으로 포매팅하는 단계; 및

상기 포매팅된 의미 오브젝트를 출력하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 36.

제 30 항에 있어서,

상기 텍스트 포매터 서비스는,

의미 오브젝트를 수용하는 단계;

상기 의미 오브젝트를 소정의 텍스트 포맷들 중 하나로 포매팅하는 단계; 및

상기 포매팅된 의미 오브젝트를 출력하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 37.

제 1 항에 있어서,

상기 컴퓨터 시스템은 네트워크 통신의 복수개 컴퓨터 시스템들을 포함하고,

상기 서비스들로서의 상기 펑크션들의 컴퓨터 시스템 소스들은 상기 복수개 컴퓨터 시스템들 중 어느 하나상의 펑크션들에 대한 장치, 애플리케이션, 전자 서비스, 내용, 또는 이들에 관한 임의 조합들의 소스들을 포함하는 것인 태스크 컴퓨팅 방법.

청구항 38.

제 25 항 또는 제 37 항에 있어서,

상기 서비스들로서의 상기 펑크션들의 컴퓨터 시스템 소스들은 WSDL(Web Service Description Language), UPnP (Universal Plug and Play), CORBA, RMI, RPC, DCE, DCOM, 또는 이들에 관한 임의 조합들의 원격 절차 호들을 통해 이용 가능한 것인 태스크 컴퓨팅 방법.

청구항 39.

제 1 항에 있어서,

상기 컴퓨터 시스템은 네트워크 통신의 복수개 컴퓨터 시스템들을 포함하고,

상기 원격 절차 호 API는 원격 웹 서비스들의 API를 포함하는 Web 서비스들에 따르며,

상기 방법은,

제 1 컴퓨터 시스템의 상기 원격 웹 서비스들의 API를 게시하는 단계; 및

제 2 컴퓨터 시스템에 의해, 게시된 원격 웹 서비스들의 API를 통해, 상기 제 1 컴퓨터 시스템상의 서비스를 관리하는 단계를 더 포함하는 것인 태스크 컴퓨팅 방법.

청구항 40.

제 39 항에 있어서,

상기 제 2 컴퓨터 시스템상의 서비스를 관리하는 단계는 상기 제 2 컴퓨터 시스템상의 서비스를 생성하는 단계, 게시하는 단계, 발견하는 단계, 실행하는 단계, 또는 이들 단계에 관한 임의 조합들을 포함하는 것인 태스크 컴퓨팅 방법.

청구항 41.

제 1 항에 있어서,

상기 미들웨어 서버 프로세싱 계층은 서비스 관리 프로세스를 포함하고, 상기 프레젠테이션 클라이언트 프로세싱 계층은 펑크션의 컴퓨터 시스템 소스를 위한 의미 오브젝트 및/또는 의미 서비스 설명을 서비스로서 생성, 게시, 및/또는 조작하기 위해 하나 이상의 의미론적으로 설명된 서비스 제어 메커니즘 클라이언트들을 포함하는 것인 태스크 컴퓨팅 방법.

청구항 42.

제 1 항에 있어서,

상기 의미론적으로 설명된 서비스 제어 메커니즘 클라이언트들은 의미화하여 의미 서비스 설명을 생성하고, 상기 의미 설명을 서비스로서 게시하기 위한 클라이언트 프로세스, 실사 오브젝트 의미화기 클라이언트, 데이터베이스 의미화기 클라이언트, 및 미디어 게시자를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 43.

제 1 항에 있어서,

상기 미디어 게시자는,

선택된 데이터 오브젝트를 수신하는 단계;

상기 선택된 데이터 오브젝트를 위한 대응하는 의미 인스턴스를 발생시키는 단계; 및

서비스를 제공하기 위해, 상기 발생된 의미 인스턴스를 위한 의미 서비스 설명을 생성하고 게시하는 단계를 포함하는 것인 태스크 컴퓨팅 방법.

청구항 44.

펑크션의 컴퓨터 시스템 소스들을 의미론적으로 설명된 서비스들로서 추상화하는 서버 프로세싱 시스템; 및

상기 펑크션의 컴퓨터 시스템 소스들의 의미론적으로 설명된 서비스들에 기초해, 컴퓨터 구현형 태스크 인터페이스를 통해 상기 컴퓨터 시스템상의 하나 이상 서비스들을 포함하는 실행 가능 태스크의 실시간 동적 구성을 지원하기 위해 원격 절차 호를 통해 상기 서버 시스템과 인터페이싱하는 클라이언트 프로세싱 시스템을 포함하는 컴퓨터 시스템.

청구항 45.

태스크 컴퓨팅 장치로서,

오브젝트를 인식하는 단계;

상기 인식된 오브젝트에 기초해 의미 인스턴스를 발생시키는 단계; 및

상기 의미 인스턴스를 게시하는 단계를 포함하는 프로세스에 따라 상기 장치를 제어하는 프로그램된 프로세서를 포함하는 태스크 컴퓨팅 장치.

청구항 46.

제 45 항에 있어서,

상기 오브젝트는 물리적 오브젝트, 음성, 비디오, 또는 이들에 관한 임의 조합들인 것인 태스크 컴퓨팅 장치.

청구항 47.

제 45 항에 있어서,

상기 오브젝트를 인식하는 단계는 컴퓨터 판독 가능 코드를 판독하는 단계, RFID(Radio Frequency Identification) 태그를 검출하는 단계, 및 멀티미디어 인식 프로세싱 엔진들에 의해 오브젝트 정보를 검출하는 단계에 따르는 것인 태스크 컴퓨팅 장치.

청구항 48.

데이터베이스 스키마를 온톨로지로 매핑하는 단계; 및

상기 매핑에 기초해, 상기 데이터베이스의 데이터에 대한 의미 오브젝트를 발생시키는 단계를 포함하는 컴퓨터 구현 방법.

청구항 49.

제 48 항에 있어서,

상기 데이터베이스의 데이터 전부 또는 일부로부터 의미 오브젝트를 발생시키기 위해 상기 의미 오브젝트를 발생시키는 단계를 제어하는 단계 및 하나 이상의 의미 오브젝트들을 단일 또는 다수 파일들로 출력하는 단계를 더 포함하는 컴퓨터 구현 방법.

청구항 50.

제 48 항에 있어서,

상기 매핑하는 단계는,

그래픽 사용자 인터페이스를 디스플레이하는 단계; 및

상기 디스플레이된 그래픽 사용자 인터페이스에 기초해, 사용자에 의해 상기 데이터베이스 스키마를 상기 온톨로지로 매핑하는 단계를 더 포함하고,

상기 그래픽 사용자 인터페이스는,

상기 데이터베이스 스키마의 선택 가능한 그래픽 디스플레이들을 디스플레이하는 제 1 윈도; 및

온톨로지의 선택 가능한 그래픽 디스플레이를 디스플레이하는 제 2 윈도를 포함하는 것인 컴퓨터 구현 방법.

청구항 51.

제 50 항에 있어서,

상기 매핑하는 단계는,

프로그램된 프로세서에 의해 상기 데이터베이스 스키마 및 상기 온톨로지의 구문론적 단서들에 따른 가능한 매핑을 제시하는 단계를 더 포함하는 것인 컴퓨터 구현 방법.

청구항 52.

제 48 항에 있어서,

의미 오브젝트의 선택 가능한 그래픽 디스플레이들을 포함하는 그래픽 사용자 인터페이스를 디스플레이하는 단계; 및

선택된 의미 오브젝트의 값을 리턴하는 단계를 더 포함하는 컴퓨터 구현 방법.

청구항 53.

펑크션의 컴퓨터 시스템 소스들을 복수개 서비스들로서 추상화하기 위한 서버 프로세싱 수단; 및

원격 절차 호 메커니즘을 통해, 상기 서비스들 중 2 이상을 포함하는 태스크의 실시간 동적 구성을 위한 사용자 태스크 인터페이스를 제공하기 위한 클라이언트 프로세싱 수단을 포함하는 컴퓨터 시스템.

청구항 54.

프로세스에 따라 컴퓨터 시스템을 제어하는 하나 이상의 프로그램을 저장하는 휴대용 컴퓨터 판독 가능 매체로서,

상기 프로세스는,

펑크션들의 복수개 컴퓨터 시스템 소스들을 서비스들로서 의미론적으로 설명하는 단계;

상기 서비스들로서 의미론적으로 설명된 펑크션들의 복수개 컴퓨터 시스템 소스들을 발견하고 상기 서비스들을 제 1 그래픽 사용자 인터페이스로 제시하는 단계;

사용자에 의해 발견된 서비스를 상기 제 1 윈도에서 선택하는 단계;

상기 선택된 발견 서비스와 관련하여 다른 유효 서비스들의 선택 가능한 그래픽 디스플레이를 자동적으로 디스플레이하는 단계;

사용자에 의해 유효 서비스를 선택하는 단계; 및

상기 발견된 서비스 및 상기 유효 서비스들의 상기 사용자 선택에 따라 상기 제 2 그래픽 사용자 인터페이스 윈도에 방향성 서비스 그래프를 상기 태스크로서 실시간에서 동적으로 디스플레이하는 단계를 포함하는 프로세스에 따라,

상기 하나 이상 서비스들의 태스크로의 실시간 동적 구성을 지원하는 제 2 그래픽 사용자 인터페이스를 제시하는 단계; 및

상기 태스크를 실행하기 위해 태스크 실행의 선택 가능한 그래픽 디스플레이를 디스플레이하는 단계를 포함하는 휴대용 컴퓨터 판독 가능 매체.

청구항 55.

터미널 장치로서,

서버 프로세스로의 원격 절차 호 메커니즘을 통해 의미론적으로 서비스들로서 설명된 펑크션들의 복수개 컴퓨터 시스템 소스들을 발견하는 단계; 및

사용자에 의해 태스크의 명령을 수신하는 단계; 및

상기 서버 프로세스를 통해 하나 이상의 서비스들을 포함하는 상기 태스크의 프로세싱을 제어하는 단계를 포함하는 프로세스에 따라,

상기 하나 이상 서비스들의 상기 태스크로의 실시간 동적 구성을 지원하는 컴퓨터 구현형 사용자 인터페이스를 발생시키는 단계를 포함하는 프로세스에 따라, 터미널 장치를 제어하는 프로그램된 클라이언트 프로세서를 포함하는 터미널 장치.

청구항 56.

제 55 항에 있어서,

상기 컴퓨터 구현형 사용자 인터페이스는 컴퓨터 디스플레이형 그래픽 사용자 인터페이스, 음성 명령 사용자 인터페이스, 라디오 장치 사용자 인터페이스, 또는 이들에 관한 임의 조합들인 것인 터미널 장치.

청구항 57.

제 55 항에 있어서,

상기 클라이언트 프로세스의 프로세스는,

상기 발견된 서비스들을 제 1 그래픽 사용자 인터페이스에 제시하는 단계;

사용자에 의해 상기 제 1 윈도에서 발견된 서비스를 선택하는 단계;

상기 선택된 발견 서비스와 관련하여 다른 유효 서비스들의 선택 가능한 그래픽 디스플레이를 자동적으로 디스플레이하는 단계;

사용자에 의해 유효 서비스를 선택하는 단계; 및

상기 발견된 서비스 및 상기 유효 서비스들의 상기 사용자 선택에 따라 상기 제 2 그래픽 사용자 인터페이스 윈도에 방향성 서비스 그래프를 태스크로서 실시간에서 동적으로 디스플레이하는 단계를 포함하는 프로세스에 따라,

상기 하나 이상 서비스들의 상기 태스크로의 실시간 동적 구성을 지원하는 제 2 그래픽 사용자 인터페이스를 제시하는 단계; 및

상기 서버 프로세스를 통해 상기 태스크를 실행하기 위한 태스크 실행의 선택 가능한 그래픽 디스플레이를 디스플레이하는 단계를 더 포함하는 터미널 장치.

청구항 58.

터미널 장치로서,

의미론적으로 펑크션의 컴퓨터 시스템 소스를 서비스로서 설명하는 OWL-S(Web Ontology Language) 파일을 판독하는 단계, 및

상기 OWL-S 파일의 판독에 따라 상기 서비스를 실행하는 단계를 포함하는 프로세스에 따라,

상기 터미널 장치를 제어하는 프로그램된 클라이언트 프로세서를 포함하는 터미널 장치.

청구항 59.

복수개의 클라이언트 프로세싱 장치들과 통신하는 서버 프로세싱 장치로서,

원격 절차 호 메커니즘에 응답하여, 상기 서버 프로세싱 장치를 제어하는 프로그램된 서버 프로세서를 포함하고,

의미론적으로 서비스들로서 설명된 펑크션들의 복수개 컴퓨터 시스템 소스들을 발견하는 단계; 및

상기 원격 절차 호 메커니즘을 통해, 상기 하나 이상 서비스들의 태스크로의 실시간 동적 구성을 지원하는 클라이언트 컴퓨터 구현형 사용자 인터페이스의 발생을 제어하는 단계를 포함하는 프로세스에 따르는 서버 프로세싱 장치.

청구항 60.

제 59 항에 있어서,

상기 클라이언트 컴퓨터 구현형 사용자 인터페이스를 제어하는 단계는,

펑크션의 컴퓨터 시스템 소스에 대한 의미 설명에서 설명된 서비스의 펑크션 특징에 따라 발견된 복수개 서비스들로부터 호환 가능 서비스들을 식별하는 단계;

선택된 하나 이상의 호환 가능 서비스들에 따라 태스크를 구성하는 단계;

태스크 실행 명령을 수신하는 단계; 및

상기 태스크 실행 명령에 응답하여, 상기 태스크를 실행하기 위해 상기 클라이언트에서 상기 하나 이상의 서비스들을 실행하는 단계를 포함하는 프로세스에 따르는 것인 서버 프로세싱 장치.

명세서

상호-참조 및 관련 출원(들)

본 출원은 그 내용들이 여기에 참고 문헌으로써 포함되어 있는, 2004년 4월 28일, Yannis Labrou, Ryusuke Masuoka, Duy Huynh, 및 Zhexuan Song에 의해 TASK COMPUTING SYSTEMS라는 명칭으로 U.S. PTO(Patent and Trademark Office)에 출원된 미국 가명세서 특허출원 제 60/565,851호(Attorney Docket No. 1634.1007P4)에 관한 것이며, 그에 대해 35 USC 119에 따른 우선권의 이익을 주장한다.

본 출원은, 그 내용들이 여기에 참고 문헌으로써 포함되어 있는, 2004년 8월 23일, Ryusuke Masuoka, Yannis Labrou, 및 Zhexuan Song에 의해 TASK COMPUTING이라는 명칭으로 U.S. PTO에 출원된 미국 가명세서 특허출원 제 60/603,251호(Attorney Docket No. 1634.1007P5)에 관한 것이며, 그에 대해 35 USC 119에 따른 우선권의 이익을 주장한다.

본 출원은, 그 내용들이 여기에 참고 문헌으로써 포함되어 있는, 2004년 11월 18일, Ryusuke Masuoka, Yannis Labrou, 및 Zhexuan Song에 의해 TASK COMPUTING이라는 명칭으로 U.S. PTO에 출원된 미국 가명세서 특허출원 제 60/628,557호(Attorney Docket No. 1634.1007P6)에 관한 것이며, 그에 대해 35 USC 119에 따른 우선권의 이익을 주장한다.

본 출원은, 그 내용들이 여기에 참고 문헌으로써 포함되어 있는, 2004년 12월 29일, Ryusuke Masuoka, Yannis Labrou, 및 Zhexuan Song에 의해 TASK COMPUTING이라는 명칭으로 U.S. PTO에 출원된 미국 가명세서 특허출원 제 60/639,805호(Attorney Docket No. 1634.1007P7)에 관한 것이며, 그에 대해 35 USC 119에 따른 우선권의 이익을 주장한다.

본 출원은, 그 내용들이 여기에 참고 문헌으로써 포함되어 있는, 2003년 12월 12일, Ryusuke Masuoka, Yannis Labrou, 및 Zhexuan Song에 의해 TASK COMPUTING이라는 명칭으로 U.S. PTO에 출원된 미국 특허출원 제 10/733,328호(Attorney Docket No. 1634.1007)에 관한 것이다.

발명의 배경

1. 발명의 분야

본 발명은, 의미론적으로 설명된 애플리케이션-, 장치- 및 서비스-리치(service-rich) 컴퓨팅 환경들에 기초해, 복잡한 태스크들을 실시간에서 동적으로 구성하고 실행하는 것에 관한 것이다.

2. 관련 기술의 설명

퍼스널 컴퓨팅은, 사용자가 단일 장치를 조작하고 그 장치에 상주하는 애플리케이션들을 액세스/사용하는 패러다임으로 언급될 수 있다. 퍼스널 컴퓨팅은, 사용자가 사용자의 컴퓨팅 환경에 대해 그리고 사용자의 컴퓨터에서 이용 가능한 애플리케이션들에 대해 충분한 이해를 가짐으로써, 지식이 있는 사용자로서, 사용자가 이용 가능한 리소들을 적절하게 이용해 복잡한 태스크들을 실행할 수 있을 것을 요한다. 이것은 대다수 사용자들이 날마다 경험하는 컴퓨팅으로서, 애플리케이션들 사이에서 데이터를 수동으로 전달하기(자르기 & 붙여넣기)위해, 각각의 애플리케이션 및 태스트에 관련된 특정 기능을 수동으로 호출하기 위해, 그리고 궁극적으로는, 복잡한 태스크의 실행에 완전한 주의(및 시간)를 기울이기 위해, 사용자의 머신에서 실행 중인 애플리케이션들 각각 및 사용자의 머신이 지원하는 평크션들 각각을 이해해야 하는 사용자에게는 복잡한 태스크들의 실현 방법을 학습해야 하는 부담이 귀속된다. 복잡한 태스크들을 실현하는 것이 한편으로는 태스크에 대한 사용자의 이해에 그리고 또 한편으로는 이용 가능한 리소스들(장치들 및 애플리케이션들)에 대한 사용자의 이해에 의존하므로, 사용자는 이들을 사용자가 실행할 작업 흐름으로 그리고 완결된 태스크일 최종 출력으로 조합할 수 있다.

컴퓨팅 환경에 대한 퍼스널 컴퓨팅으로부터 좀더 태스크-지향적 관점으로의 이동은 다음과 같을 것이다.

예를 들어, 오퍼레이팅 시스템의 일 사양으로서, 사용자가 CD 트레이에 뮤직 CD를 삽입할 때, 사용자에게는 그때부터 사용자가 수행할 수 있는 태스크들을 제시하는 윈도(window)가 팝업된다. 이러한 옵션들의 통상적인 리스팅은 다음과 같은 것을 포함할 수 있다:

Play Audio CD

Copy Music from CD

Open folder to view files

Take no action

이들 옵션들 각각은 그 액션을 수행하는데 사용될 애플리케이션도 언급한다. 초점은 태스크를 수행하는데 사용될 애플리케이션들이 아니라 수행될 액션 또는 태스크에 있다.

그러나, 여기에서, 오퍼레이팅 시스템은 특정 이벤트(음악 CD 삽입하기 또는 디지털 카메라 연결하기)의 발생과 연관된 액션들 또는 태스크들의 소정 리스트를 사용하므로, 그 이벤트가 발생할 때, 따라야 할 액션들의 관련 리스팅이 사용자에게 제시된다. 그런 의미에서, 시스템의 응답은 하드와이어링(hardwiring)되어 있고, 이벤트를 트리거링하는 결과로서 수행될 수 있는 액션들에 대해 시스템에 프로그램되어 있는 것을 넘어서는 융통성을 포함하지 않는다. 다시 말해, 시스템들은, 디지털 카메라가 컴퓨터에 접속될 때 발생할 수 있는 액션들의 동일한 세트를 보여 주는데; 오퍼레이팅 시스템의 프로그래머가 특정 이벤트를 위해 액션들의 이러한 특정 리스트를 준비하였다. 애플리케이션들은 리스트의 항목들을 변경할 수 있지만, 최종-사용자들이 그것을 변경하는 것은 쉬운 일이 아니다.

오퍼레이팅 시스템의 다른 일례에서, 사용자에게는 파일 유형에 따른 액션들의 선택이 제시될 수도 있다. 다시 말해, 태스크들의 개별 리스트가 다음의 파일 유형들: 문서들, 픽처들, 포토 앨범, 뮤직, 뮤직 아티스트, 뮤직 앨범, 및 비디오들 각각을 위해 사용자에게 제시된다. 예를 들어, 파일 유형이 픽처라면, "picture tasks"의 리스트가 제시된다.

View (pictures) as a slide show

Order prints online

Print the picture

Set the picture as background

Copy pictures to a CD

태스크들의 이 리스트는 다시 사전-컴파일되고 특정 파일 유형과 연관된다. 최종 사용자들이 리스트를 변경하기 위한 용이한 방법은 존재하지 않는다.

오피스 슈트 소프트웨어(office suite software)의 다른 일례에서는, 스마트 태그들의 사양이 이용될 수도 있다. 스마트 태그 사양은 편집기를 사용하면서 현재 문서의 텍스트를 강조하고 사용자에게, 그 텍스트가 지시하는 오브젝트로써 수행할 수 있는 액션들의 드롭다운 메뉴(drop down menu)를 제공한다. 예를 들어, 텍스트가 이름을 표현하면, 이 사양은 그 이름과 연관된 오브젝트가 사람인 것을 식별할 수 있고, 가능한 액션들의 다음과 같은 리스트를 제공할 수도 있다.

Send mail (to that person)

Schedule a meeting (with that person)

Open Contact (of that person)

Create a Contact (for that person)

문서에서의 캐릭터 스트링이 이름을 표현할 수 있다는 것을 식별하는 것에 의해, 옵션들이 인에이블된다. 시스템은 텍스트의 의미 특징들에 의존해 텍스트의 이러한 특정 부분이 이름을 표현한다는 것을 식별한다. 그러나, 통상적인 미국식 이름을 닮지 않은 캐릭터 스트링(예를 들어, Lusheng Ji)은 사람에 관련된 이름으로 식별되지 않을 수도 있다. 그 이유는, 텍스트 일부를 이름으로 식별하는 시스템 부분이, 용이하게 식별 가능한 패턴들을 텍스트의 의미 형태로 식별하고자 하는 아주 단순한 프로그램(스크립트)이기 때문이다. 일단 텍스트의 "특징"이 (정확하게 또는 부정확하게), 예를 들어, 사람, 주소 등으로 식별되고 나면, 가능한 액션들의 사전-컴파일된 리스트가 사용자에게 제시된다. 애플리케이션 프로그래머들은, 주소들을 식별하는 단계 및 맵 애플리케이션을 호출하는 단계 등과 같은, 다른 도메인들과 애플리케이션들을 위해 스마트 태그들을 생성할 수도 있다.

이하에서는, 사용자에게 컴퓨팅 환경의 좀더 태스크 지향적 뷰를 제시하기 위한 시도의 다른 일례가 논의된다. 사용자가 검색 엔진이 검색 박스에 주소를 타이핑할 때, 서비스는 (통상적인 검색 결과들의 위쪽에), 수반된다면, 주소에 대한 맵을 제공할 매핑 펑크션으로의 링크를 리턴할 것이다.

그러나, 사용자가 타이핑된 주소의 맵을 검색할 수도 있다는 것은 분명하지 않다. 사용자가 이 주소와 연관된 전화번호 리스팅을 원할 수 있거나, 그 주소가 상점이라면, 사용자가 검색된 상점에 대한 상업 개선 협회(BETTER BUSINESS BUREAU) 기록을 보고 싶어하거나 그 근방의 날씨를 점검하고 싶어할 수도 있다는 등의 합리적인 다른 가능성들이 존재한다. 그것의 현재 형태에서, 검색 엔진은 타이핑된 텍스트(이 경우, 주소)가 어떤 유형의 "것(thing)"을 대표하는지를 추측하고 엔트리의 이 유형과 연관된 고유한 태스크를 리턴한다.

따라서, 컴퓨팅 환경의 태스크-지향적 뷰에서, 초점은 태스크를 실행하는데 사용되는 애플리케이션이 아니라 수행될 수 있는 태스크에 맞춰진다. 더 나아가, 사용자는 태스크를 위해 어떤 애플리케이션이 사용될 것인지를 알 필요가 없다. 사용자가 제시된 태스크들 중 하나를 실행하기로 선택하면, 그에 따라 적절한 애플리케이션이 인스턴스화되고 호출(개시)될 것이다.

그러나, 앞서 언급된 컴퓨팅 예들은, 다음과 같이, 실행 가능한 태스크들의 실시간 동적 구성을 허용하지 않는 유사한 사양들을 나타낸다. 소정 방식에서는, 사용자 입력의 유형 또는 특징(텍스트 또는 이벤트)이 추측되고; 사실상, 시스템은, 그것의 구문론적 특징들에 의존하여, 스트링의 의미(의미론)를 추론하고자 한다. 시스템은, 사용자가 그러한 입력이 주어진 상태에서 수행하고자 할 수 있는 그럴 듯한 태스크들을 추측하고; 그 추측은 시스템으로 하드와이어링되므로, 실질적으로, 실시간으로 추측하는 것은, 시스템이 아니라 사용자가 시스템과 상호 작용하기 전에 시스템을 프로그래밍하면서 추측을

수행한 시스템의 프로그래머이다. 적합한 애플리케이션은, (사용자가 제 2 단계에서 무엇을 선택했는지에 상관없이) 사용자의 선택시에 자동적으로 호출되고, (시스템이 제 1 단계에서 추측한 것에 상관없이) 적당한 입력으로써 인스턴스화되는 정적인 인과관계(cause-effect)(또는 트리거-응답) 메커니즘이다.

상기 컴퓨팅 예들이 사용자의 편의를 증대시킬 수는 있지만, 종래 시스템들은 여전히 다음의 퍼스널 컴퓨팅 사양들을 포함한다.

기능은 애플리케이션으로 설계되었고; 애플리케이션의 프로그래머들이 시스템의 응답을 프로그래밍(하드와이어링)하였다. 그에 따라, 가능성들의 범위가 설계 시간 동안에 판정되었기 때문에, 이것은 융통성 있으며 조정 가능한 접근 방법이 아니다.

시스템은 사용자의 액션들 및 희망들을 수용하는 방법들을 제한하여 왔고, 시스템은 입력의 특징(의미론 또는 의미)을 정확하게 "인지"할 수 없다. 일례들 각각에서 사용되는 상이한 기술들에도 불구하고, 시스템은 입력의 구문론적 특징들에 의해 입력의 의미를 정확하게 추측하는 것에 의존한다.

시스템은, 입력의 소정 유형이 단일 액션(애플리케이션 호출)을 초래한다는 점에서, 인과관계(또는 트리거-응답) 메커니즘을 이용한다. 복잡한 사용자 태스크들은, 논의된 이러한 일례들에서 사용되는 단순한 기술들로는 기술적으로 언급하는 것이 불가능할 이벤트들과 액션들의 복잡한 시퀀스들을 갖춘 좀더 복잡한 작업 흐름들을 수반한다.

또한, 퍼스널 컴퓨팅, 즉, 사용자의 애플리케이션들을 실행하고 사용자의 데이터를 "보유하는" 컴퓨터를 사용자가 소유하고 조작한다는 아이디어는 덜 명확한 경계선들의 컴퓨팅 환경들에 길을 내주고 있다. 컴퓨터들이 컴퓨터 네트워크들에 상설적으로 접속됨에 따라, 로컬과 원격 애플리케이션들 및 데이터간의 구별들이 무너지거나, 좀더 심각하게는, 이들이 컴퓨터 사용자들에게 혼란을 주고 있다. 또한, 사용자들은, 퍼스널 컴퓨터들의 관념에서의 컴퓨터들은 아니지만 여전히 상당한 컴퓨터 조작력을 소지하며 사용자들의 목적들에 도움이 될 수 있고 그들이 다양한 태스크들(카메라들, 프린터들, 스마트 어플라이언스들 등)을 실현하는데 도움이 될 수 있는 장치들과 액세스하고 상호 작용할 수 있다. 일례로써, 이용 가능한 리소스들(장치들 및 애플리케이션들)이 계속 변하고 있으므로, 평균적인 사용자는 이러한 컴퓨팅 환경들에서 가능하거나 실현 가능한 것이 무엇인지조차 인지할 수 없을 수도 있다. 다시 말해, 사용자에게 미리 공지되지 않은 장치들 및 애플리케이션들로 가득한 설정에서 퍼스널 컴퓨팅 접근 방법은 실현 불가능하다.

따라서, 흔히 유비쿼터스 편재형 컴퓨팅 환경(ubiquitous pervasive computing environment)이라고 하는, 사용자가 컴퓨팅 환경에서 태스크들을 성취하는 문제에 대해 완전히 상이한 접근 방법을 요하는, 컴퓨팅 환경에서 태스크들을 실시간에서 동적으로 발견, 게시, 구성, 관리, 및 실행하기 위한 필요성이 존재한다.

발명의 요약

여기에서 설명되는 본 발명 실시예의 일 태양은, 의미론적으로 설명된 애플리케이션-, 장치-, 및 서비스-리치 컴퓨터 컴퓨팅 (컴퓨터 시스템) 환경들에 기초해, 복잡한 태스크들을 실시간에서 동적으로 발견하는 단계, 게시하는 단계, 구성하는 단계, 관리하는 단계, 및 실행하는 단계를 제공하는 것이다.

여기에서 설명되는 실시예들의 다른 태양에 따르면, 사용자는 편재형 컴퓨팅 환경과의 상호 작용을 관리하기 위해 그리고 편재형 컴퓨팅 환경과 상호 작용하기 위해, 실시간에서 실질적으로, 효과적으로, 효율적으로, 그리고 동적으로 융통성 있고 통일된 태스크 사용자 인터페이스(발견, 게시, 구성, 서비스 및/또는 태스크 관리, 및 실행 펑크션들)에 의존할 수 있다.

도면의 간단한 설명

다음에서 명백할 다른 태양들 및 이점들과 함께 이들은, 그 일부를 형성하는 첨부 도면들을 참조해야 하는, 다음에서 좀더 상세하게 설명되고 청구되는, 구성 및 동작의 세부 사항들에 속하는데, 여기에서, 참조는, 유사한 참조 번호들이 전체에 걸쳐 유사한 부분들을 언급한다.

도 1a는, 본 발명의 실시예에 따른, 태스크 컴퓨팅(TASK COMPUTING) 환경의 아키텍처에 대한 시스템도이다.

도 1b는, 본 발명의 실시예에 따른, 프레젠테이션 계층에서의 컴퓨터 구현형 태스크 인터페이스로서의 컴퓨터 디스플레이형 그래픽 사용자 인터페이스의 이미지이다.

도 2는, 본 발명의 실시예에 따른, STEER-WS API의 예시적 정의들의 리스트이다.

도 3a 및 도 3b는, 본 발명의 실시예에 따른, STEER-WS API의 사용을 예시하는 예시적인 컴퓨터 소스 코드들이다.

도 4는, 본 발명의 실시예에 따른, 미들웨어 프로세싱 계층(108) 프로그램 모듈들의 기능 블록도이다.

도 5a 및 도 5b는, 본 발명의 실시예에 따른, PIPE-WS API(122)의 예시적 정의들의 리스트이다.

도 6은, 본 발명의 실시예에 따른, 콜백 기반의 교차-환경 서비스 디렉토리의 흐름도이다.

도 7은, 본 발명의 실시예에 따른, 교차-환경 서비스 디렉토리 및 실행의 네트워크도이다.

도 8은, 본 발명의 실시예에 따른, 서비스들을 관리하기 위한 모바일 폰 디스플레이 스크린의 사용자 인터페이스 이미지들의 도면이다.

도 9는, 본 발명의 실시예에 따른, 컴퓨터 디스플레이형 위치 인식 아이콘 그래픽 사용자 인터페이스의 이미지이다.

도 10은, 본 발명의 실시예에 따른, 태스크 컴퓨팅 음성 인식의 상태 흐름도이다.

도 11은, 본 발명의 실시예에 따른, 태스크 컴퓨팅 환경에 대한 음성 인식도를 발생시키기 위한 예시적 의사-코드이다.

도 12a는, 본 발명의 실시예에 따른, 태스크 컴퓨팅 클라이언트 사용자 인터페이스를 임의 언어로 디스플레이하는 흐름도이다.

도 12b는, 본 발명의 실시예에 따른, 프레젠테이션 계층에서의 일본어의 컴퓨터 구현형 태스크 인터페이스로서의 컴퓨터 디스플레이형 그래픽 사용자 인터페이스의 이미지이다.

도 13은, 본 발명의 실시예에 따른, 서비스 액세스 제어의 흐름도이다.

도 14a 내지 도 14h는, 본 발명의 실시예에 따른, 서비스 액세스 제어 프로세스의 단계별 도면들이다.

도 15는, 본 발명의 실시예에 따른, 실사 오브젝트 의미화기(semanticizer)의 아키텍처에 대한 기능 블록도이다.

도 16a 내지 도 16c는, 본 발명의 실시예에 따른, 오브젝트들 및 서비스들을 의미론화하는 단계, 서비스화하는 단계 및 게시하는 단계의 절차들이다.

도 16d 내지 도 16n은, 본 발명의 실시예에 따른, OWL-S의 SSD들(116)에 대한 컴퓨터 해석 가능 소스 코드들의 3가지 일례들이다.

도 17은, 본 발명의 실시예에 따른, 내부 서비스들을 갖춘 STEER-WS TCC의 컴퓨터 디스플레이형 그래픽 사용자 인터페이스의 이미지이다.

바람직한 실시예들의 상세한 설명

도 1a는, 본 발명의 실시예에 따른, TCE(TASK COMPUTING(100) Computer system(s) Environment;100)의 아키텍처에 대한 시스템도이다. 도 1a에서, 컴퓨터 구현 방법은 편재형 TCE(Task Computing computer system environment;100)를 프레젠테이션 프로세싱 계층(104), 원격 절차 호 메커니즘 API(Application Programming Interface)(106), 프레젠테이션 계층(104)에서 컴퓨터의 네트워크형, 비-네트워크형, 또는 양자(110)(컴퓨터 시스템;110)로서의 서비스(112)로서, 펑크션(116)의 의미론적으로 설명된 컴퓨터 시스템 소스로의 컴퓨터 구현형 태스크 인터페이스(130;예를 들어, 소프트웨어/프로그램형 컴퓨팅 하드웨어 인터페이스, 컴퓨터 디스플레이 스크린 GUI(Graphical User Interface), 컴퓨터 음성 사용자 인터페이스)를 실시간에서 동적으로 발생시키기 위해, 프레젠테이션 계층(104)이 원격 절차 호 API(106)를 경유하여 인터페이싱하는 미들웨어(서버) 프로세싱 계층(108), 및 펑크션(116)의 의미론적으로 설명된 컴퓨터 시스템 소스를 미들웨어 프로세싱 계층(108)으로 인터페이싱하는 컴퓨터 시스템 서비스(112)로서 제공하

는 서비스 계층(112)과 펑크션 소스 실현 계층(114)을 포함하는 복수개의 태스크 컴퓨팅 컴퓨터 시스템 구현층들로 분할하는 단계; 및 컴퓨터 시스템(110)상의 하나 이상 서비스들(112)로의 프레젠테이션 계층(104)에서 발생된 태스크 인터페이스(130)에 따라, 하나 이상 서비스들(112)을 포함하는 컴퓨터 시스템 실행 가능 태스크를 실시간에서 동적으로 구성하는 단계를 포함한다.

태스크 컴퓨팅(100)은 애플리케이션-, 장치-, 전자 서비스-, 및 내용-리치 컴퓨터 네트워크 환경들(114)에서 복잡한 태스크들을 실시간에서 동적으로 발견, 게시, 구성, 관리, 및 실행하기 위한(즉, 태스크들을 실현 계층(114)에서 실행하기 위한) 새로운 패러다임이다. 태스크 컴퓨팅(100)은 (예를 들어, SSD들(Semantic Service Descriptions;116a 내지 116n)을 통해) 그들의 의미론에 따라 동작 중에 최종 사용자들에 의해 실행 가능 태스크들로 구성될 수 있는 컴퓨팅 장치들(114a-n)의 서비스들(115a-n)을 의미론적으로 설명하는 것에 기초한다. 따라서, 여기에서 설명되는 실시예들에 따르면, 태스크 컴퓨팅(100) 시스템은 프레젠테이션 클라이언트 프로세싱 계층(104), 클라이언트 계층(104)이 원격 절차 호 메커니즘을 통해 인터페이싱하는 미들웨어 서버 프로세싱 계층(108), 및 복수개 서비스들의 복수개 컴퓨터 시스템 계층으로 이루어진 3 이상의 프로그램된 컴퓨팅 및/또는 컴퓨터 판독 가능 정보 계층들(예를 들어, 의미론 인스턴스들, SSD들(116))의 다중 계층 컴퓨터 시스템 아키텍처를 가진다.

여기에서 설명되는 실시예들에 따르면, "서비스"(112)라는 용어는 컴퓨터 장치들, 컴퓨터 애플리케이션들/소프트웨어, 전자-서비스들 및 컴퓨터(나 머신 또는 양자의) 판독 가능 내용의 펑크션 소스 실현 계층(114) 영역으로부터의 기능에 대한 컴퓨터 조작 실시예들을 언급한다.

"태스크"(126)라는 용어는, 예를 들어, 사용자가 수행하고자 하는 발견된 컴퓨터 시스템 서비스들(112)에 따른 하나 이상 액션들의 구성을 의미한다. 여기에서 설명되는 실시예들에 따르면, "태스크"(126)는 자동적으로, 사용자 주도하에, 또는 그것에 관한 임의 조합으로, 컴퓨터 구현형 태스크 인터페이스(130)를 통해 구성되고 관리된다. 사용자의 경우, 하나 이상 서비스들(112)의 구성으로서의 태스크(126)는 프레젠테이션 계층(104)에서 관리(예를 들어, 발견, 게시, 구성, 실행 등)된다. 비한정적인 일례에서, 서비스들(112)의 구성, "view on projector 112 weather info 112 of business address 112 of my contact 112"는 "view on projector", "weather info", "business address", 및 "my contact"의 4개 서비스들(112)을 포함하는 "태스크"(112)이다. 다시 말해, "태스크"는 하나 이상 서비스들(112)의 구성을 포함한다.

"구성"이라는 용어는, 서비스(112)의 (제한없이) 의미 입력들 및 출력들, 예를 들어, 서비스(112)의 입력(소비)/출력(생산)을 위한 데이터 오브젝트 유형과 같은, 의미론적으로 설명된 서비스들(112)의 제공되는 펑크션 특징(들)에 따라 복수개 서비스들(112)을 다같이 조합하는 것에 의해 형성하는 것을 의미한다. 서비스의 펑크션 특징 일례는 서비스 구성 가능성을 판정하기 위한 서비스(112)의 선행 조건 및 영향일 수 있다. 서비스(112)의 선행 조건 및 영향의 일례는 서비스(112)를 위한 입력 및 출력 데이터의 오브젝트 유형들일 수 있다.

"의미 인스턴스(semantic instance)" 또는 "의미 오브젝트(semantic object)"라는 용어는, 하나 이상의 온톨로지(ontology)에 기초한, 소정 항목에 대한 한 세트의 설명들을 의미한다. SSD(Semantic Service Description;116)는 하나 이상의 서비스 펑크션 온톨로지에 기초해 서비스 펑크션(115)을 설명한다.

"게시"라는 용어는 하나 이상의 서비스 발견 메커니즘들을 통해 이용 가능한 SSD(116)를 형성하는 것을 의미한다.

"발견"이라는 용어는 일반적으로 SSD(들)(116)의 발견을 의미한다.

태스크 컴퓨팅(TASK COMPUTING)은, 자동적으로, 사용자 주도하에, 또는 양자에 의해(그것에 관한 임의 조합으로), 의미론적으로 설명된(116) 애플리케이션-, 장치-, 및 서비스-리치 컴퓨터 컴퓨팅(컴퓨터 시스템) 환경들(110)에 기초해 하나 이상의 서비스들(112)을 포함하는 "태스크"를 실시간에서 동적으로 발견, 게시, 구성, 관리, 및 실행하는 것을 지원하는 컴퓨터 시스템(100) 유형을 지시한다.

STEER(Semantic Task Execution EditoR;의미론적으로 설명된 서비스들(116)을 발견하여 실행 가능 태스크들로 구성하기 위한 소프트웨어) 및 PIPE(Pervasive Instance Provision Environment;의미 인스턴스들 및/또는 의미 서비스들(116)을 게시하고 관리하기 위한 소프트웨어)라고 하는 2가지 태스크 컴퓨팅 클라이언트 실시예들은, 그것의 전체 내용들이 여기에 참고 문헌으로서 포함되어 있는, 본 출원의 FUJITSU LIMITED 양수인에 의해 소유되는, Ryusuke Masuoka, Yannis Labrou, 및 Zhexuan Song에 의해 TASK COMPUTING이라는 명칭으로 2003년 12월 12일에 U.S. PTO에 출원된, 관련된 공동 소유의 미국 특허출원 제 10/733,328호(Attorney Docket No. 1634.1007)에 설명되어 있다. 여기에서

설명되는 실시예들은, 의미론적으로 설명된 서비스들(116)의 실행 가능 태스크들(126)로의 실시간 동적 구성을 위해서 뿐만 아니라 의미론적으로 설명된 서비스들(116)의 관리(예를 들어, 발견, 생성/게시, 조작 등)를 위해서도 사용되는 기술들 및/또는 기술들에서의 향상들에 관한 것이다.

도 1a에서는, 여기에서 설명되는 실시예(들)에 따라, 하나 이상의 TCS들(Task Computing Systems;118a-n)이 원격 절차 호 메커니즘에 기초한 클라이언트-서버 컴퓨터 시스템 아키텍처에 따라 제공된다. TCS(118)는 구현시에 논리적으로, 클라이언트 유형의 프로그램된 프로세스들을 TCC들(Task Computing Clients;119a-n)로서 제공하는 프레젠테이션 프로세싱 계층(104) 및 서버 유형의 프로그램된 프로세스들을 제공하는 미들웨어 프로세싱 계층(108)으로 분할되는데, 분할된 프레젠테이션 및 미들웨어 프로세싱 계층들(104, 108)은, TCE-WS API(Web services (WS) as a Task Computing Environment-Web Service Application Programming Interface;106a-n)와 같은, 임의의 원격 절차 호 메커니즘에 따라 인터페이싱된다. 웹 서비스들의 개념은 널리 공지되어 있다. 따라서, 여기에서 설명되는 실시예들에 따르면, 일반적으로 TCS(118)는 프레젠테이션 계층(104)에서 클라이언트형 프로세스들을 제공하는 TCC(Task Computing Client;119)를 포함하고, TCC(119)는, WS(Web services)와 같은, 원격 절차 호 API를 통해 미들웨어 서버 프로세싱 계층(108)과 인터페이싱하는데, WS를 통해 미들웨어 서버 프로세싱 계층(108)과 인터페이싱할 경우, TCC(119)는 WS TCC(119)로서 언급된다. 원격 절차 호 메커니즘의 일례로서, 웹 서비스들을 사용하는 TCS(118)를 여기에서는 WS TCS(118)라고 한다. 웹 서비스들과 같은, 원격 절차 호 메커니즘을 사용하는 것에 의해, 웹 서비스 호들과 같은, 원격 절차 호를 형성할 수 있는(또는 원격 절차 호출 능력을 포함할 수 있는), 제3자 애플리케이션들(예를 들어, MICROSOFT WORD, EXCEL, OUTLOOK, ADOBE ACROBAT 등)을 포함하는, 임의의 애플리케이션이 TCC(Task Computing Client;119)가 될 수 있다. 여기에서 설명되는 실시예들은 웹 서비스들을 원격 절차 호 메커니즘의 일례로서 사용하지만, 본 발명은 이러한 구성으로 한정되지 않으며 임의의 원격 절차 호 메커니즘이 사용될 수 있다.

따라서, 웹 서비스들을 원격 절차 호 API의 일례로서 사용한다면, STEER-WS TCS(Semantic Task Execution EditoR-Web Services Task Computing System;118a)는, STEER-WS API(120)를 통해 미들웨어 서버 프로세싱 계층(108)과 인터페이싱되는 프레젠테이션 프로세싱 계층(104)에 STEER-WS TCC(STEER-WS Task Computing Client;119a)를 포함하는 WS TCS(118)의 일례이다.

PIPE-WS TCS(Pervasive Instance Provision Environment-Web Services Task Computing System;118b)은 WS TCS(118)의 다른 일례이다. PIPE-WS API(122)는, 태스크 컴퓨팅(100)에서 사용되는 의미 오브젝트 인스턴스들 및/또는 SSD들(116)을 관리(예를 들어, 생성/게시, 제거, 조작 등)하는데 뿐만 아니라 태스크들(126)을 관리하는데도 일반적으로 사용되는 미들웨어 서버 관리 도구들(124)을 노출시킨다. PIPE-WS(122)를 사용하는 애플리케이션 클라이언트(119)를 여기에서는 SDSCM(Semantically Described Service Control Mechanism;119b)이라고 하는데, 그것의 일례들이 다음에서 부연되는 "화이트 홀(White Hole)"(119b-1), "서비스 관리자"(119b-2), "실사 오브젝트 의미화기"(119b-3), 및 데이터베이스 의미화기(119b-4)이다. 예를 들어, PIPE-WS(122)를 사용하는 WS TCS(118b)는, PIPE-WS API(122)를 통해 미들웨어 서버 프로세싱 계층9108)과 인터페이싱하는 프레젠테이션 프로세싱 계층(104)에, "화이트 홀" 태스크 컴퓨팅 클라이언트("화이트 홀")(119b-1)와 같은, 웹 서비스들의 태스크 컴퓨팅 클라이언트(애플리케이션 클라이언트) 또는 SDSCM(119b)을 포함한다.

(제한없이) STEER-WS TCC(119a), 화이트 홀(119b-1), "서비스 관리자"(119b-2), 실사 오브젝트 의미화기(119b-3), 및 데이터베이스 의미화기(119b-4)와 같은, WS TCC들(Web services Task Computing Clients;119)을 프레젠테이션 계층(104)에서의 프로그램 가능한 컴퓨팅 컴포넌트들(예를 들어, 태스크 컴퓨팅 클라이언트 소프트웨어)로서 사용하는 것을 통해, 사용자들은 TCE-WS API(106)를 통해 미들웨어 서버 프로세서들(108)에 의해 이용 가능해진 의미론적으로 설명된 서비스들(116)에 기초해 태스크들을 관리(예를 들어, 발견, 게시, 구성, 실행, 조작)할 수 있다.

도 1a에서는, 오늘날의 컴퓨팅 환경들에 따라, 사용자가, 인터넷을 통해 이용 가능한 전자 서비스들(이-서비스들;e-services), 사용자가 조작하는 컴퓨팅 장치들에서 실행되는 애플리케이션들, 컴퓨터 판독 가능 매체에서 입수 가능한 내용, 또는 단순히 특정 펑크션을 지원하는 장치들과 같은, 장치들 또는 컴퓨터-중재 서비스들을 포함하는 실현 계층(114)이라고 하는 기능에 의해 둘러싸여 있다. 그러한 장치들, 애플리케이션, 이-서비스들, 및 내용의 일례들로는 (제한없이) 전화기들, 컴퓨터 디스플레이들, 카메라들, 엔터테인먼트 장치들/센터들, 텔레비젼들, PDA들(Personal Digital Assistants), 무선 통신 장치들(예를 들어, 모바일 폰들 등), 오디오 플레이어들, 팩스 머신들, 프린터들, 날씨 서비스들, 맵 서비스들, 오피스 슈트 컴퓨팅 소프트웨어(예를 들어, 이메일 애플리케이션, 주소록 등), 멀티미디어 컴퓨터 판독 가능 매체들(예를 들어, 음악 CD, 영화 DVD 등), 인터넷 사이트들, 데이터베이스들 등을 들 수 있다.

도 1a에서, 실현 계층(114)에 의해 제시되는 기능 또는 서비스들(115a-n)은, 예를 들어, (제한없이) (예를 들어, 엔터테인먼트 장치의 경우) 음악을 청취하는 것, 노래들을 다운로드하는 것, 스트리밍 비디오들을 시청하는 것, 라디오들을 청취하

는 것, 연락 정보를 제공하는 것, 맵상의 주소들을 점검하는 것 등을 (제한없이) 포함할 수 있다. 종래에, 실현 계층(114)은 각각의 장치 또는 서비스와 상호 작용하는 (그리고/또는 조작하는) 사용자에 의해 사용자에게 기능을 제공하도록 설계되었는데; 예를 들어, 사용자가 방문 중인 방에 비치된 전화기로 동료에게 전화를 하고 싶고 동료의 전화 번호가 사용자 랩탑상의 사용자의 전자 주소록 애플리케이션에 저장되어 있다면, 사용자는 랩탑 애플리케이션을 시작해야 하고, 궁금한 전화 번호를 찾은 다음, 전화기에서 수동으로 전화 번호를 다이얼링해야 한다. 다시 말해, 사용자가 태스크(126)를 구성할 수는 없다. 심지어 애플리케이션들, 이-서비스들 및 장치들이 물리적으로 서로 통신할 수 있는 경우, 즉, 그들 사이에 통신 링크가 존재하는 경우라 하더라도, 실현 계층(114)의 설계자들이 그러한 특정 태스크를 염두에 두고 펑크션의 컴퓨터 시스템 소스, 예를 들어, 컴퓨팅 장치를 설계하지 않았다면, 그들은 사용자의 태스크를 위해 의미있는 방식으로 데이터를 교환할 수 없다. 펑크션들(114a-n)의 과다한 소스들에 직면할 경우, 사용자는, 펑크션들(114a-n)의 소스들이 그 태스크를 위해 설계되지 않았다면, 이러한 소스들 모두로부터의 기능들을 이용하는 태스크들을 수행할 수 없다. 또한, 부주의한 사용자는 대개, 그러한 태스크들이 가능하다는 것을 알지 못한다.

도 1a에서는, 여기에서 설명되는 실시예에 따라, 서비스 계층(112)이, (네트워킹형, 비-네트워킹형, 또는 양자인) 컴퓨터 시스템(110)의 서비스(112)로서, 펑크션 소스 실현 계층(114)으로부터의 서비스 펑크션(115a) 및 그에 따라 펑크션 소스 실현 계층(114)의 서비스 펑크션(115a)을 의미론적으로 설명하는 의미 서비스 설명(116a)을 포함한다. 여기에서 설명되는 실시예들의 태양에 따르면, 서비스 펑크션(115)과 SSD(116)간의 관계는 특정 펑크션 소스(114)에 대해 다대다(n:m)일 수 있다. 예를 들어, 하나의 SSD(116)가 복수개 서비스 펑크션들(115)로 대응될 수 있는데, 이 경우, SSD는 (구성에 복수개 서비스 펑크션들(115)을 가진) 서비스 펑크션(115) 구성을 SSD(116)로서 저장한다. 그리고, 하나의 서비스 펑크션(115)이 다수의 SSD들(116)로 대응될 수도 있는데, 이 경우, SSD는 단일 서비스 펑크션(115)에 대한 복수개 종류들 또는 유형들의 의미화(semanticization)를 제공한다. 예를 들어, (ISBN 입력에 대해, 저자들, 가격들, 사진들 등을 리턴하는) 북 룩업 서비스 펑크션(115)은, 하나가 저자 연락처를 리턴하고, 다른 SSD(116)가 이미지를 리턴하는 식으로, 의미 서비스들(116)에 의한 근거가 될 수 있다. 좀더 구체적으로, 여기에서 설명되는 실시예들에 따라, 서비스 계층(112)은, 다함께 (네트워킹형, 비-네트워킹형, 또는 양자의) 이용 가능한 컴퓨터 시스템(110) 서비스들(112)을 형성하는, 실현 계층(114a-n)에 의해 이용 가능한 서비스 펑크션들(115a-n) 및 서비스 펑크션들(115a-n)에 대응되는 SSD들(Semantic Service Descriptions;116a-n)을 포함한다. SSD(116)는 컴퓨터 네트워크에서 실현 계층(114)의 서비스 펑크션(115)을 노출시킨다. SSD(116)의 소정 실시예(들)은, 그것의 전체 내용들이 여기에 참고 문헌으로서 포함되어 있는, 본 출원의 FUJITSU LIMITED 양수인에 의해 소유되는, Ryusuke Masuoka, Yannis Labrou, 및 Zhexuan Song에 의해 TASK COMPUTING 이라는 명칭으로 2003년 12월 12일에 U.S. PTO에 출원된, 관련된 공동 소유의 미국 특허출원 제 10/733,328호 (Attorney Docket No. 1634.1007)에 설명되어 있다.

따라서, 태스크 컴퓨팅(100)은, 사용자가 펑크션들(114a-n), 예를 들어, 컴퓨팅 장치(114)의 실현 계층 소스들의 서비스 펑크션들(115a-n)과 상호 작용하는 방법을 위한 새로운 패러다임으로서, 태스크를 실현하는 방법을 위한 특정 수단을 강조하는 것이 아니라, 컴퓨팅 장치(114)를 사용하면서 사용자가 실현하고자 하는 태스크(126)를 강조한다. 태스크 컴퓨팅(100)은 사용자들이 수행하고자 하는 것과 그들의 환경들에서 이용 가능할 수 있는 컴퓨팅 장치(114)의 서비스 펑크션(115)간의 간극을 메운다. 태스크 컴퓨팅(100)은, 현재의 퍼스널 컴퓨팅 패러다임과 같은, 전통적인 접근 방법들에 대한 실질적인 이점들을 제시하는데, 다시 말해, 태스크 컴퓨팅(100)은 비-전문적 컴퓨터 사용자들에게 좀더 적합하고, 모든 유형들의 사용자들에 대해 좀더 시간-절감적이며, 새로운 편재형 컴퓨팅 유형의 컴퓨팅 환경들에 특히 적합하다.

따라서, 도 1a에서는, 여기에서 설명되는 실시예들에 따라, 확장하고 증축하는데 융통성이 있는 (소프트웨어 및/또는 프로그램 가능한 컴퓨팅 하드웨어의) 컴퓨터 시스템 아키텍처를 제공하기 위해, 그것의 기능이 원격 절차 호 API들 (Application Programming Interfaces;106)을 통해 프레젠테이션 프로세싱 계층(104)에 대해 이용 가능해질 수 있는 별도의 모듈화된 미들웨어 서버 프로세싱 계층(108)이 생성되므로; 애플리케이션 개발자들 및 사용자들은 이들을 사용해, 서비스들(112) 및/또는 태스크들(126)의 구성, 저장, 실행, 모니터링, 게시, 관리 등을 포함하는, 서비스(112) 발견 및 실행 가능 태스크들(126)로의 구성과 같은, 태스크 컴퓨팅 펑크션들에 액세스할 수 있다. 예를 들어, 웹 서비스들과 같은, 원격 절차 호 메커니즘은 위치(즉, 상이한 컴퓨터들상의 상이한 프로세싱 계층들), 플랫폼, 및 최종 사용자 애플리케이션 개발을 위해 필요한 프로그래밍 언어 독립성을 제공한다.

앞서 논의된 바와 같이, 유비쿼터스 편재형의 네트워킹형 컴퓨터의 컴퓨팅 환경들은 대체로 본질적으로 과도적인 다수의 장치들 및 다른 기능(이-서비스들, 애플리케이션들, 내용)(114, 115)으로 채워지고; 또한, 최종 사용자들 또는, 심지어, 유비쿼터스 환경을 위한 애플리케이션을 생성 중인 개발자들조차도 어떤 펑크션들(리소스들;114) 및 대응되는 서비스 펑크션들(115)이 소정 시점에서 이용될 수 있는지를 그리고, 좀더 중요하게는, 이들이 무엇을 위해 사용될 수 있는지를 미리 알지 못할 수도 있다. 이러한 동태성(dynamism)의 이점을 취하기 위해서는, 서비스 기능들(114, 115)이 설계시가 아니라 실행시에 발견되고 결합될 수 있어야 한다. 따라서, 여기에서 설명되는 실시예들은, 일례로서, 의미 웹(Semantic Web) 기술들을 사용하는데, 컴퓨터 네트워크 리소스들(114, 115)이 머신-판독 가능 의미들(116)에 의해 충분히 자체적으로 설명된

다면, 리소스들(114, 115)을, 최종 사용자들로 하여금, 리소스들(114, 115)이 무엇을 제공하며 무엇을 위해 사용될 수 있는지에 대한 그들만의 이해를 상기하는 것에 애플리케이션 개발자들이 통상적으로 수행하는 것을 수행할 수 있게 하기 위한 컴퓨터 시스템 서비스들(110)로서 충분히 이해하는 인프라스트럭처(100)를 구축할 수 있기 때문이다. 의미 웹의 개념은 널리 공지되어 있다.

좀더 구체적으로, 여기에서의 실시예(들)에 따르면, 태스크 컴퓨팅(100)은 널리 공지되어 있는 의미 웹 및 웹 서비스들의 개념들을 이용한다. 그러나, 정말로 동적인 애드-호크 유비쿼터스 컴퓨팅 환경에서 실제로 동작하는 시스템을 전달하기 위해서는, 여기에서 설명되는 태스크 컴퓨팅(100)에 따르면, 다음과 같은 것들이 확립되고 구현된다:

(1) 도 1a에 도시된 바와 같이, 펑크션들(110)의 컴퓨터 시스템 소스들에 태스크 인터페이스(130) 제공하기. 태스크 인터페이스(130)는 (1) TCC(Task Computing Client;119)를 포함하는 프레젠테이션 프로세싱 계층(104) 및 (2) 프레젠테이션 계층(104)에서의 TCC(119)가, TCE(Task Computing Environment) 웹 서비스들의 API(106)(예를 들어, STEER-WS API(120) 및 PIPE-WS API(122))와 같은, 원격 절차 호 메커니즘 API(106)와 인터페이싱하는 미들웨어 서버 프로세싱 계층(108)으로 논리적으로 분할되는 TCS(Task Computing System;118)를 포함한다. API(106)는 미들웨어 서버 프로세싱 계층(108)을 노출시켜 프레젠테이션 프로세싱 계층(104)에 의해 인터페이싱되게 한다. 또한, 태스크 인터페이스(130)는, 서비스 펑크션들(115)을 의미론적으로 설명하는 SSD(Semantic Service Description;116) 계층도 포함한다. SSD(116)는 미들웨어 프로세싱 계층(108)으로 덮여 TCC(119)를 통해 프레젠테이션 계층(104)에 제시될 것이고, 서비스 펑크션(115)은, 예를 들어, TCC(119)를 통해 프레젠테이션 계층(104)에서 제공되는 제어 명령에 따라 그리고 실행될 서비스 펑크션(115)을 위한 SSD(116)에 기초해, 미들웨어 프로세싱 계층(108)에 의해, 예를 들어, 실행될 태스크의 일부로서, 실행된다.

(2) 다같이 서비스 계층(112)을 제공하기 위한 SSD들(116) 및 서비스 구현들(115)의 분리;

(3) (경우에 따라, 서비스 또는 저장된 태스크의) 발견 메커니즘들 및 발견 범위들과, "구(sphere)"의 개념을 컴퓨터들(110)에서 실행 중이며 서비스들(112)을 위한 발견 범위들을 실현하기 위해 원격 TCC들(119)에 의해 액세스 가능한 원격 절차 호 API의 서브세트로서 인지하는 것에 의한 그러한 범위들내에서의 그리고 그러한 범위들 사이에서의 서비스들(112)의 조작 능력간 분리.

(4) 사용자들(및 애플리케이션들)이 다른 사용자들에 의해 이용 가능화될 수 있으며 공유될 수 있는(또는 필요할 경우, 이용 불가능화될 수 있는) 서비스들(112)을 동적으로 생성하고 조작하는 (즉, 서비스 제어 관리를 제공할 수 있는) 능력.

그리고, (5) 태스크들(126)을 흥미롭게 할 수 있으며 정말로 유용하게 할 수 있는 다양한 서비스들(112) 제공하기.

따라서, 도 1a에 도시된 바와 같이, 상술된 계층들의 분리는 논리적(개념적)인 동시에, 구현시에는, 사용자가, (암시적으로도 명시적으로도) 컴퓨터 네트워크 시스템으로 설계되지 않은 복잡한 태스크들을 수행할 수 있는 태스크 컴퓨팅(100)을 구축하는데 유용하므로, 기능(114, 115)(장치들, 애플리케이션들, 내용 및 이-서비스들)의 소스들에 대한 사용들을 배가한다. 본 발명은 의미 웹 및, 데이터가 애플리케이션, 엔터프라이즈 및 여기에서 설명되는 실시예들에 의해 사용될 수 있는 통신 경계들에 걸쳐 공유되고 재사용될 수 있게 하는, 다른 의미형 기술들이나 프레임워크에 한정되지 않는다.

도 1a에서, 펑크션 소스 실현 계층(114)은, 최하위 계층으로서, 사용자에게 이용 가능한 모든 기능이 유래하는, 컴퓨터 장치들, 컴퓨터 애플리케이션들/소프트웨어, 전자-서비스들 및 컴퓨터(나 머신 또는 양자의) 판독 가능 내용 일체를 포함한다. 펑크션 소스(114)의 (다음에서 부연되는) 서비스 펑크션들(115)은 기능의 컴퓨터 조작 실시예들이다. 그러한 서비스 펑그션(115)은 일반적으로 소스들(114)의 3가지 이상의 유형들: 장치들, 애플리케이션들 (소프트웨어) 및 웹을 통한 이-서비스들로부터 유래한다. 이들 3가지 소스들(114)은 대략적으로 정의된 비한정적인 카테고리들인데, 그들 사이의 경계들은 상당히 순응적일 수 있기 때문이다. 일례에서, 장치(114)에서 유래하는 서비스들(115)은, 장치(114)가 전달하도록 설계된 핵심 기능이다. 예를 들어, 전화기(장치)(114)의 핵심 기능은 통화하는 것(서비스)(115)이다. 마찬가지로, 애플리케이션(소프트웨어;114)에서 유래하는 기능들은, 컴퓨팅 장치(114)에서 실행 중인 소프트웨어(114)의 서비스 펑크션들(115)이다. 예를 들어, PIM(personal information management) 애플리케이션의 기능들은 개인들의 연락처 정보를 저장하고 검색하는 것을 포함한다. 마지막으로, 이-서비스들 및/또는 내용 서비스 기능(115)은, 예를 들어, 사용자의 로컬 네트워크 경계들을 넘어서는 웹으로의 액세스를 통해, 서비스 기능(115)을 전달하기 위해 소정 원격 서버에서 실행 중인 서비스 펑크션(115)이다. 기능(114)의 제 4 소스로서의 내용는 아주 유용한, 다시 말해, 서비스 펑크션(115)으로서 이용 가능한 내용일 수 있는데; 서비스 펑크션(115)의 이 유형은 사용자들 사이에서의 정보-공유 메커니즘으로서 아주 편리할 수 있다. 따라서, "서비스들"(112)은 여기에서 컴퓨터 장치들, 컴퓨터 애플리케이션들/소프트웨어, 전자-서비스들 및 컴퓨터(또는 머신이나 양자의) 판독 가능 내용 일체의 펑크션 소스 실현 계층(114)으로부터의 기능에 대한 컴퓨터 조작 실시예들을

의미한다. 따라서, 펑크션 소스 실현 계층(114)으로부터의 기능에 대한 컴퓨터 조작 실시예로서의 "서비스"(112)는, 서비스, 펑크션(들) 등의 이름 및, "서비스"(112)로의 입/출력과 같은, 서비스의 펑크션 특징들을 포함하는, "서비스"의 설명을 포함할 수 있는, "서비스"(112)와 상호 작용하기 위한 인터페이스 특징들을 가진다. 더 나아가, 여기에서 설명되는 실시예들에 따르면, 컴퓨터 시스템 서비스(10)로의 컴퓨터 구현형 사용자 인터페이스는 "서비스"(112)의 입력 데이터 및 출력 데이터 온톨로지에 기초해 의미론적으로 설명된다. 예를 들어, SSD(116)에서 설명되는 디스플레이 프로젝터상에 파일을 디스플레이하기 위한 서비스(112)는 "View on Projector"로 명명될 수 있는데, 이것은 "File"을 입력으로서 수용하고 출력 파라미터는 존재하지 않는다.

도 1a에서, 서비스 계층(112)은 SSD들(116)을 통해 컴퓨터 조작적으로 서비스 펑크션들(115)로서 이용될 수 있는 기능(114)의 소스들이다. SSD들은 서비스 펑크션들(115)의 발견 및 서비스 펑크션들(115)(의 실행)로의 액세스를 허용한다. 각각의 서비스 펑크션(115)은 하나 이상의 SSD(116)와 연관되는데, SSD(116)는, 예를 들어, RDF(Resource Description Framework)/XML(Extensible Markup Language) 교환 구문을 사용하는 OWL(Web Ontology Language)에 기초한 웹 서비스 온톨로지 언어인 OWL-S에 따라 인코딩되고, SSD(116)는, 서비스들(115)이 동적으로 생성될 수 있으므로(이용 가능해질 수 있으므로), 동작 중에 생성될 수 있다. 설명된 SSD 실시예는 OWL-S 구현으로 한정되지 않으며, 웹 서비스들을 포함하는, 컴퓨터 시스템 서비스 펑크션들(115)의 특성들 및 능력들을 설명하기 위한 임의의 컴퓨터 해석 가능 언어 구성이 사용될 수 있다. SSD(116)는 3개 부분들: 프로파일, 프로세스 및 그라운딩(grounding)을 포함하는데, 여기에서, 프로파일 부분은 사용자들로 하여금 의미 계층의 서비스(115)를 조작할 수 있게 하고 그라운딩 부분은 사용자들로 하여금 서비스들(115)을 실제로 호출할 수 있게 한다. 서비스들(115)은 태스크 컴퓨팅 분야(100)에서 이용 가능한 기능을 표현하고, 이러한 서비스들(115)의 SSD들(116)은 서비스 기능(115)의 기초적인 소스들의 복잡함으로부터 사용자를 보호하기 위한 것이고 사용자가 이러한 서비스 소스들(115)을 이용해 흥미롭고 복잡한 태스크들을 실현하는 것을 용이하게 하기 위한 것이다. 의미론적으로 설명된 서비스들(116)의 실시예(들)는, 그것의 전체 내용들이 여기에 참고 문헌으로서 포함되어 있는, 본 출원의 FUJITSU LIMITED 양수인에 의해 소유되는, Ryusuke Masuoka, Yannis Labrou, 및 Zhexuan Song에 의해 TASK COMPUTING이라는 명칭으로 2003년 12월 12일에 U.S. PTO에 출원된, 관련된 공동 소유의 미국 특허출원 제 10/733,328호(Attorney Docket No. 1634.1007)에 설명되어 있다.

도 1a에서, 미들웨어 서버 프로세싱 계층 컴포넌트들(108)은 서비스들(115, 116(또는 112))을 찾아내는 것, 서비스들(115, 116)이 실행 가능 태스크들로 구성될 수 있는 방법을 판정하는 것, 서비스들을 실행하고 서비스 실행을 모니터링하는 것, 그리고, 의미론적으로 설명된 서비스들(116)의 생성 및 게시를 포함하는, 다양한 관리 동작들을 가능하고 용이하게 하는 것을 책임진다. 다시 말해, 미들웨어 프로세싱 계층 컴포넌트들(108)의 목적은 모든 서비스 리소스들(115)을, 그것들을 조작하고자 하는 사용자들 또는 애플리케이션들에 대해 (예를 들어, TCC들(119)을 통해 프레젠테이션 계층(104)에서) 이용 가능해질 수 있는 의미론적으로 설명된 서비스들(116)로서 추상화하는 것이다.

도 1a에서, 프레젠테이션 계층(104)은 미들웨어 프로세싱 계층(108)의 능력들을 이용해 사용자들로 하여금 이용 가능한 모든 서비스 기능(116, 115 (112))을 조합하는 것에 의해 태스크들을 실행할 수 있게 한다. WS TCC들, WS 애플리케이션들, 및/또는 (웹 브라우저로써 액세스 가능한) WS 웹-기반 인터페이스 애플리케이션들이라고 하는, 웹 서비스들(118a-n)을 사용하는 프로그램 가능한 다양한 컴퓨팅 클라이언트들(예를 들어, 소프트웨어 클라이언트들, 프로그램 가능한 컴퓨팅 하드웨어 클라이언트들, 또는 양자 등)이 미들웨어 프로세싱 계층(108)을 통해 이용 가능한 모든 서비스 기능(112)을 조합하는 것에 의해 태스크들을 실행하기 위해 제공된다. 여기에서 설명되는 실시예에 따라, 미들웨어 계층 컴포넌트(108)는 잘 정의된 WS API들(Web Services Application Programming Interfaces;106)을 통해 노출됨으로써, 이러한 API들(106)을 이용하는 WS TCC들(WS Task Computing Clients;119)의 생성을 허용한다.

중간 프로세싱 계층(108)에서, 서비스(112) 발견, 구성, 실행, 저장, 생성, 관리와 같은, 태스크 컴퓨팅의 핵심 기능들로의 무제한적인 액세스들을 위한 WS API들(106)의 태스크 컴퓨팅 환경을 정의하는 것은 가능성들의 전체 어레이를 개방하는 것이다. 예를 들어, WS TCC들(119)은, 사용자가 웹 서비스(106) 호들을 형성할 수 있고, 사용자가 임의의 플랫폼에서 임의의 프로그래밍 언어를 사용해서 작업하여 WS TCC들(119)을 생성할 수 있고 서비스들(112)에 액세스할 수만 있다면, 태스크 컴퓨팅 모듈들의 특정 구현으로 한정되지 않는다.

따라서, 도 1a에서는, 여기에서 설명되는 실시예들에 따른, TCE-WS(Task Computing Environment-Web Services) API(106)가 제공된다. TCE-WS API(106)의 서브세트들은 다양한 태스크 컴퓨팅 목적들을 위해 사용될 수 있는데, 여기에서, STEER-WS TCS(118a)에서 사용될 경우에는 STEER-WS API(120), 하나 이상의 PIPE-WS TCS들(118b)에서 사용될 경우에는 PIPE-WS API(122), 그리고 (다음에서 부연되는 바와 같이) 교차-환경의 태스크 컴퓨팅을 위한 "구(Sphere)"를 제공하는데 사용될 경우에는 SoM(Sphere of Management)-WS API(123)라고 한다. 본 발명의 실시예들에 따라, 여기에서는 다음과 같은 것들이 설명될 것이다:

여기에서는, STEER-WS API(120)에 기초하며 의미론적으로 설명된 서비스들(116)을 찾아내고 의미론적으로 설명된 서비스들(116)의 실행 가능 태스크들로 구성하기 위한 소프트웨어인 STEER-WS TCC(Semantic Task Execution EditoR-Web Services;119a)와 같은, 다양한 WS TCC(Web Services Task Computing Client;119a) 실시예들이 설명될 것이다. WS TCS(118)의 프레젠테이션 계층(104) 컴포넌트로서의 STEER-WS TCC(119a)는 다양한 컴퓨터 구현형 사용자 인터페이스들을 제공한다. 따라서, 여기에서는, XT(STEER-WS-Extended) TCC(119a-1)라고 하는 컴퓨터 디스플레이형 그래픽 사용자 인터페이스, 모바일 폰과 같은, 무선 장치에 구현되며, Mobile-Phone STEER-WS TCC(119a-2), SIS(STEER-WS-Spatial Information System) TCC(119a-3), VoiceSTEER-WS TCC(119a-4), 및 Tasklet-WS TCC(119a-5)라고 하는 컴퓨터 디스플레이형 그래픽 사용자 인터페이스가 설명될 것이다.

또한, 여기에서는, PIPE-WS API(122)에 기초한 애프리케이션 클라이언트들(119)도 설명될 것인데, 이들은, 서비스들(112)을 생성, 제거, 변경하는 것과 같은, 서비스들(112)을 관리하기 위한 SDSCM(Semantically Described Service Control Mechanism;119b)이다. 특히, 여기에서는 "서비스 관리자"(119b-2), "실사 오브젝트 의미화기"(119b-3), 데이터베이스 의미화기(119b-4), 및 미디어 게시자(119b-5)가 설명될 것이다.

또한, 여기에서는, 교차-환경의 서비스(112) 발견 및 실행을 위한 "관리 구(sphere of management)" 개념도 설명될 것이다.

또한, 여기에서는, 통신 언어 응화(communication language accommodation;예를 들어, 구어의 국제화) 및 새로운 파라미터들과 같은, SSD(116)의 확장된 용도(향상된 거동들)가 설명될 것이다. SSD(116)는 서비스(115)의 파라미터들을 서비스(115) 자체로부터 WS TCC들(119)로 전달하는 수단이다. 좀더 구체적으로, 서비스(112) 입력을 위한 완화된 유형, 서비스들(112)의 위치들, 다중-언어 서비스들(112), 및 서비스(112) 관리 펑크션들을 포함하는, 태스크 컴퓨팅(100) 환경을 향상시키기 위한 파라미터들의 새로운 종류들이 구현된다.

또한, 여기에서는, 서비스들(112)로의 액세스 제어도 설명될 것이다.

또한, 여기에서는, 인스턴스 생성기, 인스턴스 복사기, 인스턴스 인터셉터, 인스턴스 저장기 및 특성 선택기와 같은, 태스크 컴퓨팅 클라이언트(119)의 내부 서비스들도 설명될 것이다.

또한, 여기에서는, (1) 의미 인스턴스 직렬화 서비스들, (2) 정보 제공 서비스들, (3) 시간/온도 서비스들, (4) 스냅샷 서비스들, (5) OWL 포매터 서비스, 및 (6) 텍스트 포매터 서비스를 포함하는, 설계되고 구현되는 새로운 서비스들(112)도 설명될 것이다.

사용자 인터페이스 - STEER-WS-XT TCC(119a-1):

도 1b는, 본 발명의 실시예에 따른, 프레젠테이션 계층(104)에서의 STEER-WS-XT TCC(119a-1)에 의한 컴퓨터 디스플레이형 그래픽 사용자 인터페이스의 이미지이다. 도 1b에서, 컴퓨터 디스플레이형 그래픽 사용자 인터페이스 윈도(142)는 발견된 서비스들(112)을 아이콘 트리 구조에 따라 디스플레이하는 발견된 서비스(112) 윈도(또는 발견 윈도)(142)이다. 여기에서 설명되는 실시예들의 태양에 따르면, 서비스들(112)은, (제한없이) 친숙한 이름들, 서비스 유형, 알파벳 순서 등과 같은, 온톨로지 및/또는 다른 범주화들에 기초한 서비스들(112)의 계층 구조 카테고리의 임의 유형에 따라, 발견된 서비스 윈도(142)로 편성된다. 컴퓨터 디스플레이형 그래픽 사용자 인터페이스 윈도(144)는 다수 입/출력들을 위한 서비스들(112)의 비-선형적 구성을 응화하는 방향성 서비스(112) 그래프인 태스크 윈도(또는 태스크(126) 구성 윈도)(144)이다. 도 1b에서, 태스크(126) 윈도(144)는 비한정적인 일례로서 5개 서비스들(112)을 포함하는 태스크(126)를 디스플레이한다. 특히, 태스크(126)는 "view on my projector 112 a route of 112 of home address 112 and business address 112 from my contact 112"이다.

도 1b에서는, 여기에서 설명되는 실시예들의 태양에 따라, SSD(116) 윈도(150)가 서비스 윈도(122)의 선택된 서비스(112)를 위한 SSD(116) 파라미터들/특성들을 디스플레이한다. SSD 윈도(135)는, 예를 들어, 태스크(126)를 구성하는 단계의 일부로서 태스크(126)를 테스트하는데 유용할 수 있다.

도 1b에서, 태스크 윈도(144)는 발견된 서비스들의 윈도(142)에서 선택된 서비스들(112)의 선택 가능한 그래픽 디스플레이들을 제공한다. 태스크 윈도(144)에서는, 발견된 서비스(112)의 선택시에, 온톨로지에 기초한 서비스의 펑크션 특징에 따라 호환 가능 서비스들이 자동적으로 식별되고, 서비스(112)의 그래픽 디스플레이 또한 자동적으로, 선택된 발견 서비스를 위한 이용 가능하거나 유용한 (호환 가능) 서비스들(112)을 표현하는 하나 이상의 선택 가능한 펑크션 특징 버튼들

(145a-n)을 포함한다. 펑크션 특징 버튼의 선택은 선행 서비스(112)의 성과를 소비함으로써, 그에 의해, 서비스들(112)의 그래픽 디스플레이들을 접속하는 디스플레이된 라인들에 의해 지시되는 바와 같이, 하나 이상 서비스들(112)의 구성이 다 같이 태스크(126)를 생성할 수 있는 발견된 다른 서비스들(112)의 선택 가능한 리스트를 디스플레이한다. 좀더 구체적으로, 태스크 윈도(142)에서, 사용자는 지시된 서비스(112) 그래프를 태스크(126)로서 구성한다. 서비스(112)의 입/출력 데이터 오브젝트 유형을 서비스(112)의 펑크션 특징들로서 사용하는 경우, 출력 펑크션 특징 버튼(145a)은 컬러 또는 임의의 공지된 여타 컴퓨터 디스플레이 구별 방법들에 의해 입력 펑크션 특징 버튼(147a)과 구별된다.

다음에서는, STEER-WS TCC(119a)이 아닌, Mobile-PhoneSTEER-WS TCC(119a-2), STEER-WS-SIS TCC(119a-3), VoiceSTEER-WS TCC(119a-4), 및 Tasklet-WS TCC(119a-5)의 다른 컴퓨터 구현형 사용자 인터페이스들을 부연할 것이다.

도 1a 및 도 1b를 참조하면, 태스크 컴퓨팅(100) 시스템은 현재 환경에서 이용 가능한 서비스들을 찾아내고, 이용 가능한 서비스들의 사용자-중심적인 태스크 뷰를 구성하고 조작하며, 다수 서비스들로 이루어진 결과적 태스크들을 실행하기 위한 기초를 제공하는 아키텍처를 가진다. 이것은 심지어, 최종 사용자들도 새로운 서비스들을 필요에 따라 동적으로 용이하게 생성할 수 있게 한다. 태스크 컴퓨팅(100) 시스템의 3가지 특징들/요소들은 다음과 같다:

(1) 모든 펑그션(114, 115)의 서비스들(112)로서의 균일한 추상화. 여기에서 논의되는 바와 같이, 태스크 컴퓨팅(100)에서, 미들웨어 서버 프로세싱 계층(108)은 모든 리소스들을 의미론적으로 설명되는 서비스들(112)로서 추상화하는 기능을 한다. 의미론적으로 설명된 서비스는, 서비스들을 설명하기 위한 언어(예를 들어, OWL-S)의 의미 설명(파일)(116)이 특정된, (제한없이) WSDL(Web Service Description Language), UPnP(Universal Plug and Play), CORBA, RMI, RPC, DCE, DCOM 서비스 펑크션들(115)과 같은, 원격 절차 호들을 통해 이용 가능한 서비스 펑크션(115)이다. 그러한 의미 설명들(116)을 특정할 때, 특정된 온톨로지는, 서비스(116, 115 (112))가 작용하는 도메인을 위해 특정된다. 온톨로지들과 관련하여, 소프트웨어 도구들이 온톨로지들을 생성하는데 사용될 수 있고, 가능하다면, 기존의 또는 이용 가능한 온톨로지들이 사용될 수 있다. OWL-S 서비스 설명들(116)은 의미화되고 있는 서비스 펑크션(115), 예를 들어, 입력 및 출력의 펑크션 특징을, 예를 들어, 의미 오브젝트들 및, 서비스(112)의 소유자, 생성자, 위치 등으로 표현한다. 설명은, 실제의 WSDL 및/또는 UPnP 서비스가 적절하게 실행될 수 있도록 하기 위해, 그라운딩 정보도 포함한다. 이러한 설명들을 제공함에 있어서, 여기에서 설명되고 그리고/또는 언급된, (제한없이) 실사 오브젝트 의미화기(119b-4), 데이터베이스 의미화기(119b-5), 내부 서비스 순간 생성자 등과 같은, 의미화기 도구들은 온톨로지 오브젝트들을 WSDL 파라미터들로 매핑하고 임의의 필요한 그라운딩(그라운딩은 XSLT 스크립트들을 통해 표현됨)을 생성하는데 사용되었다. 웹 서비스 인터페이스들(106)은, 그에 기초해 사용자가 프레젠테이션 클라이언트 프로세싱 계층(104)에서 인터페이싱하는 직관적 태스크(126)가 제공되는 미들웨어 서버 프로세싱 계층(108)을 위해 제공되었다.

태스크 컴퓨팅 미들웨어를 의미 서비스 설명들의 동적인 리포지토리(repository)로도 볼 수 있다. 여기에서 논의되는, 이러한 설명들에 액세스하고 이러한 설명들을 조작하기 위한 API들(106)과는 별도로, 서비스 설명들에 대하여 임의의 RDQL(RDF Query Language) 쿼리를 프로세싱할 API를 구현하는 것에 의해 이러한 리포지토리를 직접적으로 쿼리하기 위한 수단이 제공된다(JENA 2.0은 RDQL 쿼리들의 프로세싱을 위한 일례로서 사용된다). 예를 들어, 개발자는, 그러한 목적을 위한 명시적 API가 제공되지 않는 경우라 하더라도, 태스크 구성을 위해 사용자에게 제시되는 서비스들을, 위치와 같은, 서비스들의 소정 특징에 의해 필터링할 수 있다. 이 능력은 애플리케이션 개발자의 능력을 연장하고, 소정 쿼리들이 좀더 유용해짐에 따라, 이들은 사전-특정된 RDQL 쿼리들을 실행하는 API들(106)로서 미들웨어에 영구적으로 부가될 수 있다.

서비스들(112)과 같은 기능의 추상화는, 기능이 보편적으로 액세스될 수 있게 하고 태스크 컴퓨팅 인프라스트럭처가 그러한 기능과 상호 작용할 수 있게 한다. 태스크 컴퓨팅(100) 시스템은 환경의 장치들 중 사용자 컴퓨팅 장치의 (애플리케이션들 및 OS로부터의) 그리고 인터넷을 통해 이용 가능한 이-서비스들의 기능(114, 115)을 추상화된 서비스들(112)로 변환한다. 이러한 추상화는, 그 자체만으로는 기능들의 태스크들(126)로의 사용자 실시간 조작 및 구성을 제공하기에 불충분할 수도 있지만 더 적은 사전-준비들이 환경에서 이용 가능한 기능들을 처리할 수 있는 방법을 준비하는데, 그에 따라, 여기에서 설명되는 실시예들은 복수개 서비스들(112)을 포함하는 태스크(126)의 실시간 동적 관리를 지원하는 프레젠테이션 계층(104)도 제공한다.

(2) SSD들(116)에 기초해 (사용자 및/또는 시스템에) 추상화된 서비스들(112)의 직관적 조작 제공하기. 서비스들(112)의 직관적 조작은 SSD들(116)의 사용을 통해 가능해지고; 온톨로지들은 그러한 시스템 및/또는 시스템의 직관적 조작을 실현하기 위한 메커니즘이다. SSD(116)의 개념은, 그것의 전체 내용들이 여기에 참고 문헌으로서 포함되어 있는, 본 출원의

FUJITSU LIMITED 양수인에 의해 소유되는, Ryusuke Masuoka, Yannis Labrou, 및 Zhexuan Song에 의해 TASK COMPUTING이라는 명칭으로 2003년 12월 12일에 U.S. PTO에 출원된, 관련된 공동 소유의 미국 특허출원 제 10/733,328호(Attorney Docket No. 1634.1007)에 설명되어 있다.

만약, 예를 들어, SSD(116) 대신에, 펑크션(115)의 WSDL(Web Service Description Language) 소스만이 사용되어 웹 서비스의 펑크션 특징들을 설명한다면, WSDL-설명형 웹 서비스들은, 프로그래머들이 (WSDL 설명들을 넘어서는) 그들의 의미들을 이해하고 서비스들을 제대로 사용하기 위한 코드를 개발할 것을 요한다. 그 결과, 최종 사용자들의 기능들과의 상호 작용은 이러한 프로그램들의 범위에 의해 개발자들에 의해 미리 정의된 방법들로 한정된다. 온톨로지 오브젝트들을, (제한없이) WSDL 파라미터들과 같은, 펑크션(115) 파라미터들의 소스로 매핑하고 필요한 임의의 그라운딩을 생성하는 것에 의한 (SSD(116)에 공급되는) 추가 의미들로 인해, 태스크 컴퓨팅(100) 인프라스트럭처는 사용자들이 이러한 깊은 지식없이도 서비스들을 조작하는 것을 도울 수 있다. 예를 들어, 의미들은 사용자들에 의한 서비스들의 조작을 제한하거나 사용자에게 현재 환경에서 이용 가능한 태스크들(126)을 제시하는데 사용될 수 있다. WSDL만이 서비스들의 의미 입력들 및 출력들에 기초한 서비스 구성을 위해 의존된다면, 구성은, 예를 들어, XSD(XML Schema Definition) 스트링을 발생시키는 서비스와 XSD 스트링을 소비하는 다른 서비스의 임의 구성들로 한정되지 않을 것이므로, 실행 불가능한(또는 무효한) 서비스 구성들을 초래할 수도 있다. 따라서, 여기에서 설명되는 실시예들에 따르면, "구성"은, (제한없이) 서비스(112)의 의미 입력들 및 출력들, 예를 들어, 서비스(112)의 입력(소비)/출력(생산)을 위한 데이터 오브젝트 유형과 같은, 의미론적으로 설명되는 서비스들(112)의 제공되는 펑크션 특징(들)에 따라 복수개 서비스들(112)을 모으는 것에 의해 형성하는 것을 의미한다. 서비스의 펑크션 특징의 일례는 서비스 구성 가능성을 판정하기 위한 서비스(112)의 전제 조건 및 결과일 수 있다. 서비스(112)의 전제 조건 및 결과의 일례는 서비스(112)를 위한 입력 및 출력 데이터의 오브젝트 유형들일 수 있다. 특히, 서비스들(112)의 SSD들(116)은 서비스들의 입력들 및 출력들에 대한 좀더 미세한 세분성(granularity)을 제공함으로써, 예를 들어, "Address" 의미 오브젝트를 발생시키는 서비스는 의미론적으로 호환 가능한 서비스들로써만 구성 가능할 것이다.

서비스들(112)의 사용자 직관적 조작을 제공하는 다른 메커니즘은, "Route from My Home to" 서비스 명칭과 같은, 자연어에 따른 적절한 서비스 명칭들을 제공하는 것에 의한 것인데, 호환 가능 서비스들의 구성된 서비스 명칭들은 자연어 태스크(126) 표현(들)(예를 들어, "View on Projector"(112) + "My File"(112), "Route from Company-1 to"(112), "A City Name Airport"(112))으로서 기능할 수 있다. 온톨로지들 또한, 서브클래스-수퍼클래스 관계들에 기초한 구성들과 같은, 메커니즘들 및 최종 사용자들에게 아주 자연스러운 의미 오브젝트 번역들을 지원할 수 있다. 따라서, 태스크(126)의 구성은 자연어 문장에 기초하는데, 다시 말해, 구성된 태스크(126)는 자연어 문장처럼 읽힌다. 좀더 구체적으로, 여기에서 설명되는 실시예들은 자연어 요소들의 자연어 문장으로서의 구성 가능성으로 매핑하는 서비스들의 구성 가능성을 지원하기 위해 자연어 문장의 요소(예를 들어, 구)로서 서비스에 이름을 할당하는 단계를 제공한다. 따라서, 태스크 컴퓨팅(100) 시스템은, 최종 사용자들이 환경(110)의 서비스들과 상호 작용하기 위한 아주 풍부하고 흥미로운 방법들을 허용한다.

(3) 사용자는, 예를 들어, 도 1b에 도시된 바와 같이, (1) 및 (2)에 기초한 컴퓨터 구현형 사용자 인터페이스를 통해 태스크(126)의 실시간 및/또는 동적 (지연 바인딩형) 구성을 안내할 수 있다.

TCE-WS API(TCE Web Service Application Programming Interface;106):

도 2는, 본 발명의 실시예에 따른, STEER-WS API(120)의 예시적 정의들의 리스트이다. 도 1a 및 도 1b에서, STEER-WS TCC(119a)는, 서비스들(112)을 발견하고 필터링하며, 서비스들(112)을 태스크들(126)로서 구성하고, 실행하며, 저장하는데 편리한 사용자 인터페이스를 제공하는 WS TCC(119)이다. TCE-WS API(106)인 STEER-WS API(120)는 태스크 컴퓨팅 기능들을 독립적인 모듈들로 추출하고 그들을, 예를 들어, STEER-WS TCC(119a)와 같은, 임의의 WS TCC(119)에 의해 액세스 가능한 표준 웹 서비스 인터페이스들로서 노출시킨다.

도 2에 도시된 바와 같이, 태스크 컴퓨팅 미들웨어 서버 프로세싱 계층(108)의 기능들을 웹 서비스들(106)에 의해 노출시키는 것에 의해, 프레젠테이션 프로세싱 계층(104)에서의 WS TCC(119)는 태스크 컴퓨팅 미들웨어 서버 프로세싱 계층(108)의 모듈들에 대한 구현으로부터 자유로울 수 있다. WS TCC(119) 개발자는, 웹 서비스(106) 호들이 형성될 수 있고, 그에 의해 WS TCC(119)를 제공할 수만 있다면, 임의의 프로그래밍 언어를 임의의 오퍼레이팅 시스템에 사용할 수 있다. 웹 서비스 호들을 형성할 수 있는(또는 웹 서비스들의 호출 능력을 포함할 수 있는) 제3자 애플리케이션들(MICROSOFT WORD, EXCEL, OUTLOOK, ADOBE ACROBAT 등)조차도 잠재적 WS TCC(119)일 수 있다.

도 2에서, 발견, 구성, 실행, 모니터링, 저장 등과 같은, 기능들은 STEER-WS API(120)에서 지원된다. 일반적으로, (다음에서 도 5a 및 도 5b를 참조하여 부연하는) STEER-WS API(120) 및 PIPE-WS API(122)와 같은, TCE-WS API(106)는 SSD(116)에서 설명되는 의미론적으로 설명된 서비스 펑크션(115)을 고유하게 식별하는 어떤 것인 SID(Service(112)

identifier) 파라미터에 의존한다. 통상적으로, 여기에서 설명되는 실시예들에 따르면, SID는 SSD(116)에서 설명되는 의미론적으로 설명된 서비스 펑크션(115)으로의 URL(Uniform Resource Locator) 스트링이다. 예를 들어, 도 3a는 발견된 서비스들(112)에 관한 로컬 지식을 동기화하기 위해 STEER-WS API(120)를 사용하는 예시적인 컴퓨터 소스 코드(300)를 나타낸다. 도 3b는 STEER-WS API(120)를 사용해 다수 서비스들(112)을 가진 태스크들(126)을 호출하는 또 하나의 예시적인 컴퓨터 소스 코드(310)를 나타낸다. 비한정적인 일례의 도 3b에서, ServiceList 파라미터는, 예를 들어, 다수 태스크들을 한정하기 위해 "&"를 사용하고 태스크내의 서비스(112) 식별자들을 한정하기 위해 "I"를 사용하는 입력 스트링이고, WS TCC(119)는 태스크 실행을 호출하고 모니터링하기 위해 자신의 코드에 도 3b의 프로그램 루프를 가질 수 있다. 따라서, 본 발명에서, 미들웨어 서버 프로세싱 계층(108)의 원격 절차들을 호출하기 위해 TCE WS API(106)를 이용하는, 도 3a 및 도 3b와 같은, 소스 코드들은, STEER-WS TCC(119a)와 같은, WS TCC들(119)의 실시예 구현들이다.

도 4는, 본 발명의 실시예에 따른, STEER-WS TCC(119a)의 미들웨어 서버 프로세싱 계층(108) 프로그램 모듈들의 기능 블록도이다. 도 1 및 도 4에 도시된 바와 같이, STEER-WS TCC(119a)의 미들웨어 프로세싱 계층(108)은, STEER-WS API(120)를 통해 프레젠테이션 프로세싱 계층(104), 서비스(112) 발견 모듈들(404), 실행 모듈들(406), 및 모니터링 모듈들(408)로부터의 요청들을, 웹 서비스들(106)에 따라, 제어하는 중앙 모듈(402)을 포함한다. 중앙 모듈(402)은 서비스(112) 파싱 및 인덱싱 모듈들(410)과 서비스(112) 구성 및 태스크(126) 실행 플래닝(412)을 포함한다. 서비스(112) 파싱 및 인덱싱 모듈들(410)은, 발견 모듈들(404)이 발견된 서비스들(112)을 등록/등록 취소할 수 있게 하는 등록 인터페이스(422)를 제공한다. 발견 모듈들(404)은, 로컬 발견 모듈(414), UPnP와 같은, 임의의 제3자 서비스 펑크션(115) 발견 모듈(416), 원격 사이트 발견 모듈들(418), 및 각각의 발견 모듈이 상이한 환경(110)에서 사용되어야 하는지의 여부를 판정하는 관리 펑크션을 가진 발견 모듈 관리(420)와 같은, 개개 발견 모듈들의 세트를 포함한다.

여기에서 설명되는 실시예들의 태양에 따르면, 서비스 발견 모듈들(404)은 제3자 발견 모듈(416) 또는 원격의 제3자 발견 모듈(418n)을 사용하는 원격 사이트 발견 모듈을 통해 임의의 서비스 펑크션(115) 발견 메커니즘에 따라 서비스 펑크션들(115)을 발견한다. 제3자 발견 메커니즘들(416)은, 예를 들어, UPNP(Universal Plug and Play) 기술, JINI 기술, BLUETOOTH 등이거나 그것의 임의 조합일 수 있다. 예를 들어, CYBERLINK UPNP 및/또는 INTEL UPNP TOOLKIT 구현이, UPnP에 의해 서브-네트워크내에서 브로드캐스팅되는 서비스 설명들을 발견하기 위한 제3자 발견 모듈(416)에 사용될 수 있다. 또한, 서비스 발견 모듈들(4041)은 로컬 발견 모듈(414) 및 원격 사이트 발견 모듈(418)을 통해 SSD들(116)도 발견할 수 있다.

여기에서 설명되는 실시예들의 태양에 따르면, HEWLETT-PACKARD DEVELOPMENT COMPANY에 의한 JENA가 SSD들(116)을 저장하는데 사용된다. 파싱 및 인덱싱 모듈들(410)은 SSD들(116)을 파싱하고 분석하기 위한 파싱 및 분석 펑크션들을 포함한다. 예를 들어, 여기에서 설명되는 실시예들의 태양에 따르면, SSD(116)는, MINDLAB, UNIVERSITY OF MARYLAND, USA에 의한 PELLET 및 OWL-S API의 지원하에, HEWLETT-PACKARD DEVELOPMENT COMPANY에 의한 JENA를 사용해 파싱된다. 특히, "a service 112 is discovered"는 "the SSD 116 of a service 112 is found"와 등가이다. 서비스 발견 모듈들(404) 중 하나에 의해 발견 가능한 SSD(116)는, 등록 인터페이스(422)를 통해, SSD(116)가, 예를 들어, PELLET 지원하의 JENA에 의해 처음으로 파싱되는 중앙 모듈(402)로 송신된다. 일단 SSD가 파싱되고 나면, PELLET는 RDQL 쿼리들에 응답할 준비를 갖추게 된다. 서비스(112) 및 인덱싱 모듈(410)로부터 쿼리들을 요청하는 것에 의해 그리고 쿼리 결과들에 기초해, 서비스(112) 구성 및 태스크(126) 실행 플래닝 모듈(412)은 서비스(112) 구성(들)을 태스크(126)로서 완결짓고, TCC(119)로부터의 태스크(126) 실행 명령에 응답하여, 태스크(126)를 위한 실행 계획을 판정한다. 일단 실행 계획이 판정되고 나면, 중앙 모듈(402)은, 실행 모듈들(406)을 통해, 서비스 펑크션(115)을 호출하기 위해 SSD(116)에서 제공되는 그라운딩 호출(424)을 포함하는 관련된 서비스 펑크션(들)을 호출한다. 발견 모듈들(404)은, 서비스 펑크션들(115) 및 SSD들(116)을 포함할 수 있는 서비스들(112)을 발견한다. 서비스(112) 파싱 및 인덱싱(410)의 상기 설명은 그러한 구성으로 한정되지 않으며, JENA 및 PELLET가 아닌, SSD들(116)을 파싱하고 분석하기 위한 임의의 메커니즘이 사용될 수도 있다.

여기에서 설명되는 실시예들의 태양에 따르면, 독립 모듈로서, WS TCC(119)는, 통일된 그리고 높은 레벨로 추상화된 발견 및 실행 메커니즘들이 웹 서비스들의 API(들)(106)에 따라, 예를 들어, 기본적인 BLUETOOTH SDP, IR, RENDEZVOUS, JINI 등(404, 406)을 위한 웹 서비스 인터페이스(106)를 구현하는 것에 의해 구현되기만 한다면, 모든 종류의 기본적인 서비스(112) 발견 메커니즘들(404) 또는 실행 메커니즘들(406)을 사용할 수 있다. 따라서, 예를 들어, 사용자가 특정해야 하는 단 한가지는 서비스 계층(112)(예를 들어, 발견된 서비스들(115, 116))과 인터페이싱하기 위한 STEER-WS API(120)를 위한 WSDL(Web Service Definition Language) 파일들의 URL(Uniform Resource Locator)이다. 웹 서비스 API(106)가 제공되기만 한다면, TCE-WS API(106)에 의한 기본적인 전체적 발견 절차는 프레젠테이션 프로세싱 계층(104)의 WS TCC(119)에서의 사용자에 대해 투명하다. 예를 들어, STEER-WS API(120a) 중 하나는 BLUETOOTH 기반 서비스들(112)을 발견하고 실행하기 위해 BLUETOOTH 발견 모듈들(404)을 사용 중일 수 있다. 다른 STEER-WS API(120n)는 UPnP 발견 모듈들(404)을 사용 중일 수 있다.

도 1a에서, PIPE-WS TCS(118b)는 의미 오브젝트 인스턴스들을 게시하고 관리하기 위한 WS TCS(118)의 다른 일례이다. PIPE-WS API(122)는 태스크 컴퓨팅 관리 기능들(124)을 독립적인 모듈들로 추출하고 독립 모듈들을, "화이트 홀"(119b-1), "서비스 관리자"(119b-2), "실사 오브젝트 의미화기"(119b-3), 및 "데이터베이스 의미화기"(119b-4)와 같은, 임의의 WS TCC(119)에 의해 액세스 가능한 표준 웹 서비스 인터페이스들(106)로서 노출시킨다. 좀더 구체적으로, PIPE-WS API(122)는, 오퍼레이팅 시스템이나 애플리케이션 오브젝트들, 장치 서비스들 등을 게시하는 것과 같이, 서비스들(112)을 관리하기 위한 PIPE-WS TCS들(118b)에 웹 서비스들의 인터페이스(106)를 제공한다.

도 5a 및 도 5b는, 본 발명의 실시예에 따른, PIPE-WS API(122)의 예시적 정의들의 리스트이다. PIPE-WS API(122)는 태스크 컴퓨팅 서비스들(112)의 용이한 생성을 가능하게 한다. 예를 들어, 도 5a에 도시된 바와 같이, OWL 오브젝트로 설정된 파라미터, 데이터로써 "Insert" 웹 서비스들(122) 중 하나를 호출하는 것에 의해, PIPE-WS TCS(118b)는 소정 특성들을 가진 소정 OWL 오브젝트에 따른 서비스(112)를 생성할 수 있다. 스트링의 웹 서비스를 위한 OWL-S로 설정된 파라미터, 데이터로써 "Insert" 웹 서비스들(122) 중 하나를 호출하는 것에 의해, 웹 서비스가 소정 특성들을 가진 태스크 컴퓨팅 서비스(112)로서 이용되게 할 수 있다. "Remove" 웹 서비스들(122)을 호출하는 것에 의해, 태스크 컴퓨팅 서비스(112)를 즉시 이용 불가능하게 할 수도 있다.

교차-환경 발견 및 실행:

여기에서 설명되는 실시예들에 따르면, 웹 서비스 인터페이스들(106)의 지원하에, 태스크 컴퓨팅 미들웨어 서버 프로세싱 계층(108a)은, 웹 서비스들을 통해 다른 태스크 컴퓨팅 미들웨어 서버 프로세싱 계층(108n) 모듈들에 액세스하는 것에 의해, 교차-환경 발견 및 실행 메커니즘들을 위한 "관리 구(sphere of management)"의 새로운 개념을 포함할 수 있다. 교차-환경 발견 및 실행을 위한 한 세트의 TCE-WS API들(106), 예를 들어, STEER-WS API(120) 및 PIPE-WS API(122)의 서브세트를 여기에서는 "SoM(Sphere of Management)-WS API들"(123)이라고 한다. 따라서, SoM-WS API(123)는 원격 서비스들(112)을 관리(제공)하는데 사용된다. 여기에서 "교차-환경"은 일반적으로, 복수개 서브-네트워크들(110a-n)이 상이하거나 다른 네트워크 환경들(예를 들어, 전용 네트워크들, 가상의 전용 네트워크들, 서브넷들, 인트라넷들 등)로서 공지 기술들에 기초해 서로 통신하고 있으며 각각의 컴퓨터 네트워크 환경이 태스크 컴퓨팅 환경(100)의 일부로서 발견 가능하며 실행 가능해야 하는 펑크션들(110)의 컴퓨터 네트워크 서비스들 또는 소스들(즉, 펑크션 소스 실현 계층(114) 및 서비스 계층(112))을 포함할 수도 있는 컴퓨터 시스템 네트워크 특징을 고려하는 것을 의미한다.

여기에서 설명되는 실시예들의 태양에 따르면, 다음의 STEER-WS API(120)(도 2)가 SoM-WS API(123)로서 사용된다.

checkExecutionStatus

executeService

filterServiceByProperties

findAllServices

getServiceDescription

queryByRDQL

여기에서 설명되는 실시예들에 따르면, "구"를 실현하기 위해, SoM-WS API(123n) 미들웨어 서버 프로세싱 프로그램들(108n)이 다른 원격 컴퓨터 시스템(110n)에서 실행된다(다른 원격 컴퓨터 시스템(110n)에 설치된다). 미들웨어 서버 프로세싱 프로그램들(108a)의 인스턴스는 TCE-WS API(106n)(즉, 원격 "구" API(123n))의 인스턴스들 또는 상이한 네트워크들, 상이한 플랫폼들 및/또는 상이한 기술들(예를 들어, BLUETOOTH)에서 실행 중인 서비스들(112)을 통합하고, 서비스들(112)을 발견하며, 서비스들(112)을 태스크들(126)로서 실행하기 위한 원격 API를 구현하는 무엇인가를 통해 내부적으로, 다른 "구"의 미들웨어 서버 프로세싱 계층들(108n)의 인스턴스들에 액세스할 수 있다.

따라서, "구"는, 교차-환경의 서비스(112) 발견, 게시, 실행 및 관리를 위해 일관된 뷰를 제공하는, SoM-WS API(123)와 같은, 원격 API들의 세트에 의해 실현된다. "구"의 개념은 "필터링된 구"로 확장된다. "필터링된 구"는 한 쌍의 교차-환경 서비스(112) 발견의 SoM WS API(123) 및 서비스(112) 필터이다. WS TCC(119)는, 그것이 친숙한 "구" 명칭과 함께 (또는 친숙한 "구" 명칭없이) 원격 구 API(123)로의 링크를 추가할 때, 필터를 적용하고 그것을 "필터링된 구"라고 한다. 일반

적으로, 필터링된 SoM-WS API(123)는 선택적 스트링을 친숙한 "구" 명칭으로서 제공하고, RDF를 위한 쿼리 언어인(임의의 다른 쿼리 언어가 필터로서 사용될 수도 있음) RDQL(RDF Data Query Language)에 따른 (부분적인) 쿼리(예를 들어, 스트링) 및 WSDL에 따른 서비스(112) 발견 SoM-WS API(123)로의 URL을 요한다. 쿼리는 서비스(112)의 SSD(116)에서의 서비스(112) 특성(들)에 기초한다.

예를 들어:

("My Services at Company-1", "http://www.company-1.com/steerws/service.asmx?wsdl", "owner eq 'Bob Smith'")

("Company-1 Conf Room Services", "http://www.company- 1.com/steerws/service.asmx?wsdl", "location eq 'Conference Room'")

상기 2가지 일례들에서, "owner eq 'Bob Smith'" 및 "location eq 'Conference Room'"은 RDQL 쿼리의 서비스(112) 필터들로서의 일례들이다. 첫번째 경우에서, "My Services at Company-1"은, 서비스(112) 특성 "owner"가 "Bob Smith"와 동일하거나 "Bob Smith"에 의해 소유되는 그러한 서비스들(112)에 따라 필터링된다. 특히, 쿼리(들)는 서비스 파라미터(들)의 값(들)에 따른다. 이 경우, 쿼리는, "Bob Smith"가 서비스 특성 "owner"의 값과 동일한지를 판정한다. STEER-WS TCC(119a)가 필터를 사용해 STEER-WS API(120)로부터 서비스들(112)을 찾아낼 때, 필터와 매칭되는 서비스들(112)만이 친숙한 서비스 명칭에 의해 정의되는 카테고리 아래에 표시된다. STEER-WS API(120)가 그것을 이해할 수만 있다면, 필터가 완전한 RDQL 쿼리일 필요는 없다. 따라서, 서비스들(112)을 필터링한다는 개념이 "구"에 적용될 때, 원격 컴퓨터 시스템(110)의 필터링된 서비스들(112)만이 제공되는 "필터링된 구" 또는 "가상 구"가 실현될 수 있다. 다시 말해, 서비스(112) 발견에서 쿼리를 특정하는 것이 사용자의 "가상 구" 또는 "서브-구"를 실현한다.

따라서, 여기에서 설명되는 실시예들에 따르면, 단일 "구"가 다수의 "필터링된 구들"을 제공할 수도 있다. 필터들이 상이하기만 하다면, 상이한 "필터링된 구들"을 위해 발견된 서비스들(112)의 세트들은 WS TCC(119) 사용자에 의해 독립적으로 인지된다. 필터링 메커니즘들은 원격 구들(123)의 API에 대한 서비스(112) 발견 메커니즘 뿐만 아니라 로컬 및 편재형 (예를 들어, UPnP) 서비스(112) 발견 메커니즘들에도 적용 가능하다.

도 6은, 본 발명의 실시예에 따른, 콜백 기반 교차-환경 서비스 디렉토리의 흐름도이다. 도 3a에서 논의된 방법은 서비스(115, 116;112) 디렉토리를 위한 폴 기반 솔루션(poll based solution)이다. 교차-환경 태스크 컴퓨팅을 구현하기 위해, 여기에서는 도 6을 참조하여, 교차-환경 서비스(112) 디렉토리 및 실행의 다른 방법이 설명된다. 도 6에 따르면, 교차-환경 서비스 디렉토리를 위한 콜백 기반 솔루션은, 우선, STEER-WS API(120)에/로 "Inform(스트링 메시지)"(120)이라는 새로운 웹 서비스들의 인터페이스를 추가/제공하는 단계를 포함한다. 동작 600에서, 다른 컴퓨팅 환경(110)내의 서비스들(112)을 발견하기 위해, WS TCC(119)(A)는, STEER-WS API(120a)와 같은, TCE WS API(106a)를 통해, 스스로를 "Inform" STEER-WS(120a)를 통해, 여기에서 SoM WS API 123n(B)라고 하는, 그러한 다른 컴퓨터 환경(110n)의 다른 원격 TCE WS API(106n)에 등록한다. 앞서 논의된 바와 같이, 원격 또는 다른 컴퓨터 환경(110n)에서 실행하는 SoM WS API(123n)는 "구"를 실현한다. 동작 602에서, B가 새로운 서비스들(112)(또는 서비스들(112) 중 일부가 제거된 것)을 나중에 발견할 경우, B는 A의 "Inform" 웹 서비스 인터페이스(106a)를 사용해 A가 그 변화들을 알게 한다. 동작 602에서 새롭게 발견된 서비스들(112)의 식별 정보들을 포함하는 메시지 또는 A는, 동작 604에서, "findAllServicelds" 웹 서비스들(120)을 사용해 마지막 정보를 검색할 수 있다. 콜백 기반 솔루션의 이점들은 (1) A가 B를 반복해서 폴링할 필요가 없다는 것으로 인한 효율성과, (2) 그것이 신속한 발견을 지원한다는 것이다. 또한, 폴 기반 솔루션에서는, B가 어떤 변화를 가진다 하더라도, A는 후속 폴까지 그것을 알 수 없는 반면, 콜백 기반 솔루션에서는, A가 거의 즉시 그것을 알게 될 것이다. 그러나, 콜백 기반 솔루션은, A가 방화벽 뒤쪽에 존재한다면, 단점을 가질 수도 있는데, A의 웹 서비스들(106a)가 B로부터 액세스될 수 없어, 콜백 기반 솔루션이 작용할 수 없을 수도 있기 때문이다. 따라서, 폴 기반 솔루션은, 방화벽들이 구현된 경우의 교차 컴퓨팅 환경들에서 작용하는 이점을 가진다. 또한, 폴 기반 솔루션은 일방 접속을 요하는 이점을 가진다.

도 3b에서의 실행 샘플 소스 코드가 교차-환경 태스크(126) 실행을 위해 사용될 수 있다. 절차가 사용자에게 투명하도록 하기 위해, 교차-환경 서비스(115)의 SSD(116)를 파싱할 때, SSD(116)의 그라운딩 부분은 다른 교차-환경들(110)의 TCE-WS API(TCE Web services API;106)를 포인팅하도록 다시쓰기될 수 있거나, 다른 방법으로는, TCE-WS API(106)를 구현하는 다른 교차-환경의 끝이 SSD(116)를 다시쓰기(즉, OWL-S를 다시쓰기)하여, 실행이 특정 웹 서비스들(106)을 통과하는 방식으로 그것의 TCE-WS API(106)를 통해 SSD(116)를 제공할 수도 있다.

도 7은, 본 발명의 실시예에 따른, 교차-환경 서비스 발견 및 실행의 네트워크 도면이다. 도 7에서, 각각의 WS TCC(118)(예를 들어, STEER-WS TCC 1-4)는, 펑크션들(112)의 서비스들 또는 소스들이 존재하는 서브-네트워크(110)내에 자신

의 고유한 발견 범위를 가진다. 통상적으로 본 발명에서, 서비스(112) 발견 범위는 서비스(112) 발견에 사용되는 기초적인 기술들에 의해 한정된다. 예를 들어, UPnP가 편재성 서비스들(112)을 발견하는데 사용된다면, UPnP가 동일한 서브-네트워크(110)에 존재하지 않는 또는 동일한 서브-네트워크(110) 외부에 존재하는 서비스들(112)을 찾아낼 수는 없다.

따라서, 여기에서는, STEER-WS API(120) 및 PIPE-WS API(122) 기술들에 기초해, 교차-환경 서비스(112) 발견, 게시, 실행 및 관리가 설명된다. 도 7에 도시된 바와 같이, STEER-WS TCC(119a)와 같은, 다수의 WS TCC들(119)은 상이한 네트워크 환경들(110)에서 실행된다. 자신의 고유한 범위(11Oa)내에서, STEER-WS TCC(119a)는 한 세트의 서비스들(112)을 찾아낼 수 있다. 그 다음, STEER-WS TCC(119a)의 웹 서비스들(120)을 사용하는 것에 의해, STEER-WS TCC(119a)는 다른 STEER-WS TCC(119n)의 웹 서비스 인스턴스들(120)과 통신/인터페이스할 수 있고 그들의 서비스(112) 발견 결과들을 취한다.

일례로서 도 7에는, STEER 클라이언트 1 내지 4라고 하는 4개의 STEER-WS TCC(119a1-n)가 존재한다. 각각은, 구름으로서 묘사되어 있는 자신의 고유한 발견 범위(11Oa)를 가진다. STEER-WS TCC(119a)는 웹 서비스들의 API(120)를 가진다. 범위(110n) 밖의 서비스(115, 116;112)를 발견하기 위해, 나머지 WS TCC들(119a1-n) 각각의 웹 서비스들(120)이 호출된다(예를 들어, STEER-WS TCC(119a2)의 웹 서비스들(120)이 호출된다). 예를 들어, 도 7에서, STEER 클라이언트 3은, STEER 1에 의해 발견되는 모든 서비스들을 검색하기 위해, STEER 클라이언트 1의 "findAllServicelds" 웹 서비스(120)(도 2)를 호출한다. 일단 서비스 ID 리스트가 검색되고 나면, STEER 클라이언트 3은 "findServiceProperty" 웹 서비스(120; 도 2)를 사용해 서비스들(115)의 의미 서비스 설명(116)을 검색하고 그들을 자신의 고유한 엔진에 등록한다. 등록하는 동안, 새로운 서비스(112) 식별 정보가 생성된다. 식별 정보는, 서비스(112)가 원격 서비스(112;원격 서비스 식별자)라는 것과 그것이 처음으로 발견된 곳을 지시한다. 이제, STEER 클라이언트 3은 서비스들(115, 116;112)을, 그들이 마치 자신의 고유한 발견 범위(110a)에 존재하는 것처럼, 사용할 수 있다.

원격 서비스(112)를 호출하기 위해, STEER-WS TCC(119)는 다른 STEER-WS TCC들(119)의 웹 서비스 인터페이스(120)도 사용할 것이다. 상기 도 7의 일례에서, STEER 클라이언트 3은 (입력이 존재한다면) 서비스(115)의 입력을 인코딩할 것이고 STEER 클라이언트 1으로 요청을 송신할 것이다. 그 결과, STEER 클라이언트 3은 요청에 관한 참조 식별자를 수신할 것이다. 다음으로, STEER 클라이언트 3은, 기준 식별자를 사용하는 것에 의해 실행 상태에 관해 STEER 클라이언트 1을 폴링할 것이다. 4가지 가능성들이 존재한다:

1. 실행이 완결되면, (DONE) 메시지가 리턴될 것이다. 그 다음, STEER 클라이언트 3은 참조 ID를 제공하는 것에 의해 출력을 검색하기 위해 다른 웹 서비스 인터페이스(120)를 사용할 수 있다.

2. 실행이 실패하면, (ERROR) 메시지가 리턴될 것이다.

3. 실행이 여전히 진행 중이라면, 어떤 메시지도 리턴되지 않을 것이다. 그 다음, STEER 3은 잠시 대기할 것이고 다시 폴링할 것이다.

4. 서비스(115, 116)가 서비스 제어 UI(user interface)를 가지며 사용자로부터 더 많은 입력을 요한다면, 서비스 제어 UI의 URL이 리턴될 것이다. 그 다음, STEER 클라이언트 3은 사용자에게 URL을 나타낼 것이다. 한편, STEER 클라이언트 3은 잠시 대기할 것이고 다시 폴링할 것이다.

여기에서 설명되는 실시예의 태양에 따르면, 교차-환경 서비스 발견 및 실행은 계층 구조로서 또는 임의의 다른 구성으로 디플로이(deploy)될 수 있다. 예를 들어, 도 7에서, STEER 클라이언트 2는 STEER 클라이언트 4의 서비스들(115, 116;112)을 발견하고 실행하기 위해 웹 서비스 인터페이스(120)를 사용한다. STEER 클라이언트 3이 STEER 클라이언트 2와 통신할 때, STEER 클라이언트 3은 STEER 클라이언트 2에 의해 처음에 발견된 서비스들(112) 뿐만 아니라 STEER 클라이언트 4로부터의 서비스들(112)도 취한다. STEER 클라이언트 3은 그들 사이의 차이점을 알지 못한다. STEER 클라이언트 3은, 서비스들(112) 모두가 STEER 클라이언트 2로부터 유래하는 것이라고 생각할 뿐이다. 서비스(112)를 호출하기 위해, STEER 클라이언트 3은 요청을 STEER 클라이언트 2 웹 서비스 인터페이스(120)로 송신할 것이다. 일단 STEER 클라이언트 2가 요청을 수신하고 나면, STEER 클라이언트 2는 자신의 고유한 미들웨어 프로세싱 엔진(108)을 점검하여, 서비스(112)가 실제로는 STEER 클라이언트 4로부터 유래한다는 것을 알게 된다. 그 다음, STEER 클라이언트 2는, 요청을 STEER 클라이언트 4로 전달할 것이다. 일단 STEER 클라이언트 2가 STEER 클라이언트 3에 의해 폴링되고 나면, STEER 클라이언트 2는 STEER 클라이언트 4를 폴링할 것이고 그것이 취하는 모든 것을 STEER 클라이언트 3로 송신할 것이다.

도 7에서, 태스크 컴퓨팅 시스템(118-1-n)(TSC 1-4, 이 경우, STEER-WS TCS(118a1-n))의 인스턴스 각각은 통상적으로, 태스크 컴퓨팅 미들웨어(108)가 실행 중인 (즉, 그 장치에 대해 로컬인) 동일 머신 또는 시스템(110)에서 실행되는 그러한 서비스들, 인터넷 및 장치들에서 일반적으로 이용 가능한 서비스들 및 동일한 서브네트의 서비스들을 발견(및 실행)할 수 있다. Bob을 방문 중인 Alice가 Bob의 집에서 Carol이 있는 곳까지의 맵을 Bob의 TV에 나타내고 싶어하는 상황을 고려한다. 이 경우, Carol의 연락처 정보는 (Alice의 집에 있는) Alice의 TCS(118-1)를 통해 액세스 가능한 Alice의 PIM에서 입수 가능하지만, 디스플레이 서비스는 Bob의 아파트 9 TCS(118-n)에서 이루어진다. 이 시나리오를 태스크(126)의 범위를 확장하는 것 또는 태스크 컴퓨팅(100)의 범위를 확장하는 것이라고 하고, 그에 따라, 다른 태스크 컴퓨팅 환경의 서비스들(112)을 발견하고 실행할 수 있다. 또한, 각각의 개개 TCS(118)와 연관된 서비스들이 방화벽 뒤에서 실행될 수도 있는데, 이것이 문제를 더욱더 복잡하게 한다. 미들웨어 계층(108)의 웹 서비스 인터페이스들은 태스크 컴퓨팅(100) 시스템의 범위를 확장하기 위한 솔루션을 설계하는데 도움이 된다.

도 7을 참조하여, SoM(Sphere of Management)이라고 하는, 좀더 상세한 설명이 이루어진다. 도 7의 4개 TCS(118, 110) 노드들에서, 각각의 고유한 발견 범위는 구름으로 묘사되며, 그것의 미들웨어 계층(108)은 D, C, E, M으로써 표현되는 4개의 프로그램 가능 모듈들 또는 유닛들(발견(404), 중앙(402), 실행(406) 및 관리(124))을 포함한다. 컴퓨팅 환경(110)에 대하여 원격이라 간주되는 미들웨어 계층(108)을 "구"를 실현하기 위한 SoM API(123)라고 한다. 일 노드(118)가 그것의 범위를 벗어난 서비스를 발견하고자 한다면, 그것은 먼저, 서비스(112)가 어떤 TCS에 상주하며 그러한 다른 노드(118)에 의해 노출된 웹 서비스(106)의 WSDL(123)을 알아야 한다. 이 정보는, 예를 들어, 온라인으로 또는 이-메일에 의해 TCS(118) 인간 오퍼레이터들 사이에서 교환될 수 있고; 다른 옵션은, TCS들(118)의 인간 소유자들이 그들의 TCS들을 게시할 수 있는 디렉터리를 실행하는 것일 것이다.

다음으로는, 원격 노드 TCS(118)에서 그 노드의 서비스들 및 서비스(112) 각각의 SSD(116) 리스트를 취하기 위해, 원격 TCS(118)에 대해 로컬인 클라이언트가 발견 웹 서비스들을 호출하는 것과 동일한 방식으로, 발견 웹 서비스들이 호출될 수 있다. 예를 들어, TCS 3은 TCS 1에 의해 발견될 수도 있는 모든 서비스들을 검색한 다음 그들을 그것만의 고유한 발견 모듈(들)에 등록하기 위해 TCS 1의 웹 서비스(106)를 호출한다. 이 서비스들은 TCS 3에 의해, TCS 3에 의해 직접적으로 발견된 다른 서비스들처럼, 취급되지만, TCS 3의 발견 모듈은, 서비스들이 처음에 발견된 장소를 지시하기 위해 이러한 서비스들에 대한 그것의 내부 표현에 플래그를 추가한다. 초기 발견 이후에, TCS 3은 TCS 1 서비스들의 리스트에 대한 변화들을 점검하기 위해 TCS 1을 폴링할 수도 있다.

실행은 웹 서비스 API(106)를 통해서도 지원된다. 사용자는 TCS 3로부터 TCS 1과 TCS 3 모두로부터의 서비스들을 갖춘 새로운 태스크를 생성할 수 있고 그것을 실행할 수 있다. TCS 3에 의해 처음에 발견된 서비스들은 어떠한 특수 처리도 요하지 않는다. 그러나, TCS 1의 서비스들은 특별한 동작을 요할 수도 있는데, 이들은 방화벽 뒤쪽에 존재할 수도 있어 TCS 3이 그들을 직접적으로 호출할 수 없기 때문이다. 이 경우, TCS 3은 실행 중인 TCS 1으로부터 도움을 받는다. 먼저, TCS 3은 웹 서비스 API들(106)을 통해 문맥(입력 파라미터들 등)과 함께 특정 서비스에 관한 실행 요청을 TCS 1으로 송신할 것이고, 참조 ID를 응답으로 취할 것이다. TCS 1이 서비스를 실행하는 동안, TCS 3은 그러한 참조 ID를 사용해 TCS 1을 상태와 관련하여 폴링할 것이다. 마지막으로, TCS 1에서의 실행이 완결된 후, 업데이트된 문맥(출력 등)이 마지막 폴링 결과로서 리턴된다. 이러한 설계를 위한 이론적 근거는, 폴링 동안, TCS 1이 (현재 실행 상태 또는 실행에 관한 추가 정보 등과 같은) 무엇인가를 역 리턴할 수도 있고, 폴링을 통해, TCS 3이 상태를 모니터링하는 것이 좀더 용이하다는 것이다.

일단 TCS(118, 110) 노드가 외부 사용자들에게 노출되고 나면, 누구든 그것의 서비스들을 발견하고 실행할 수 있다(이들은 결국 웹 서비스들이다). 보안 기술은 본 발명의 웹 서비스 API에 사용자 식별 메커니즘을 추가할 것이고 그리고/또는, 다음에서 부연되는 바와 같이, 웹 서비스들의 보안 표준들에 기초하여 구축될 것이다.

SoM은 단 2개의 TCS들(118) 너머로 연장한다. 도 7에서, TCS 2는 웹 서비스 인터페이스(106)를 사용해 TCS 4의 서비스들을 발견하고 실행한다. TCS 3이 TCS 2와 통신할 때, TCS 3은 TCS 2에 의해 처음에 발견된 서비스들 뿐만 아니라 TCS 4로부터의 서비스들도 수신한다. TCS 4로부터 서비스를 호출하기 위해, TCS 3은 요청을 TCS 2로 송신한다. 일단 TCS 2가 요청을 수신하고 나면, TCS 2는 그것의 고유한 엔진을 점검하고, 서비스가 실제로 TCS 4로부터 기인한다고 판정했을 때, TCS 2는 요청을 TCS 4로 전달할 것이다. 일단 TCS 2가 TCS 3에 의해 폴링되고 나면, TCS 2는 TCS 4를 폴링할 것이고 응답을 TCS 3로 전달할 것이다.

서비스(112) 발견을, 사용자의 문맥과 관련하여, 서비스들(112)의 SSD들을 통해 서비스들(112)을 찾아내는 프로세스라고 한다. 앞서 논의된 바와 같이, 서비스 구현(115)과 SSD(116)가 분리되면, 발견은 TCS(118)에 의한 서비스 펑크션들(115)의 SSD들에 대한 획득 및 프로세싱으로 감소된다. 서비스 발견의 구현은 하나 이상의 발견 메커니즘들에 의존하는

데; 미들웨어 프로세싱 계층(108)을 포함하는 TCS(118)는 다수의 기초적인 서비스 발견 메커니즘들을 이용할 수 있고, 서비스는 다수의 발견 메커니즘들을 통해 발견 가능할 수도 있다. 사용자들 또는 서비스들(또는 그들의 제공자들)이 특정 서비스의 발견을 위해 이용되는 발견 메커니즘을 설정할 수도 있다. 서비스를 위한 발견 메커니즘을 변경하는 것은 서비스를 발견할 수 있는 사람에 영향을 미칠 수도 있다. 서비스 발견 메커니즘들이 발견 범위들에 직교이기는 하지만, 일부 발견 메커니즘들이 다른 것들보다 특정 발견 범위에 좀더 적합하다(표 1 참고). 다음에서는, 각각의 발견 범위가 설명될 것이다.

[표 1]

발견 범위	예시적인 발견 메커니즘
1. 공백	N/A
2. 전용	파일 시스템 기반 발견
3. 서브네트에 의한 그룹	멀티-캐스트 기반 발견
4. 관심에 의한 그룹	공동 디렉토리, 게시/가입(회사, 커뮤니티)
5. 공용	개방 의미 서비스 디렉토리

1. 공백 발견 범위의 공백 서비스들은 누구에 의해서도 발견될 수 없는 서비스들이다. 공백은 전적으로 개념적인 범위이고; (심지어 그것의 소유자에 대해서도) 이용 불가능하게 되는 어떤 서비스도 이 범위를 가정할 수 있다. 예를 들어, 프라이버시 고려들로 인해 다른 사람들에 의해 또는, 사용자가 사용하고 싶지 않은 서비스를 항상 발견하는 것은 번거로운 일이기 때문에, 심지어 사용자 자신에 의해 발견된 사용자의 연락처 정보를 제공하는 서비스를 원하지 않는 사용자는 이 서비스를 위해 공백 범위를 선택할 수도 있다. 나중에, 사용자가 있는 공항으로부터 사용자 집까지의 경로를 키오스크에 디스플레이하기 위해 사용자의 연락처를 사용하고 싶을 경우, 사용자는 서비스를 전용 발견 범위로 이동시킬 수도 있다.

2. 전용 발견 범위에서의 전용 서비스들은 그들의 소유자에 의해서만 발견 가능하고 통상적으로, TCS(118)를 실행하는 사용자의 고유한 컴퓨팅 장치에 상주한다. 예를 들어, 사용자로 하여금 사용자의 장치에서 파일을 선택하고 노출시키게 하는, "My File"(112)과 같은, 로컬 리소스 핸들링 서비스들은 (디폴트로) 이러한 발견 범위를 가정한다. 여기에서, TCS(118)는 소켓들을 사용해 이러한 발견 범위를 구현하는 통지들과 조합된 파일 시스템-기반 발견 메커니즘을 사용한다.

3. 서브네트 발견 범위에 의한 그룹은, 그것의 애드-호크 및 자발적인 그룹화 특징으로 인해, 유비쿼터스 환경들과 가장 밀접한 관련이 있다. 회사의 인트라넷 또는 홈 네트워크의 일부와 같은, 사용자의 동일한 서브네트에서 발생할 서비스들이 발견되어, 극도로 국지화된 발견 메커니즘을 가능하게 할 것이다. 예를 들어, UPnP는 이 범위를 구현하기 위한 발견 메커니즘으로 사용될 수 있다. 구체적으로, UPnP의 발견 메커니즘이 (모두가 태스크 컴퓨팅-인에이블형 서비스들은 아닌) 서브네트상의 UPnP 장치들을 찾아내는데 사용되고, 각각의 UPnP 장치를 위해, TCS(118)는, UPnP 장치가 태스크 컴퓨팅-인에이블형 서비스(112)를 표현하는지를 판정하기 위해 하나의 특정 UPnP 액션(getDescriptionURL)을 호출하고, 그렇다면, TCS(118)는 UPnP 장치로부터 SSD(116)를 다운로드하도록 진행한다. JINI와 같은 다른 발견 메커니즘들 또한, UPnP가 이러한 발견 범위를 구현하는 것과 동일한 방법으로 사용될 수 있다.

4. 관심 발견 범위에 의한 그룹은, 회사의 고용자들의 그룹 또는 골프 클럽의 회원들과 같은, 어쩌면 유사한 관심사들 또는 그룹 멤버쉽에 의해 결집되는, 사람들의 자의적인 임의 그룹에 의해 발견되는 서비스들을 의미한다. 이러한 발견 메커니즘은 웹 서비스들을 콜백들과 조합하고 메커니즘들을 폴링하는 것에 의해 제공될 수 있다.

5. 이러한 발견 범위의 공용 서비스들은 누구에 의해서도 발견될 수 있다. 이 범위를 위한 양호한 발견 메커니즘은 개방 의미 서비스 디렉토리로서; 일례들로는 공용적으로 이용 가능한 서비스들의 SSD들(116)로의 링크들을 갖춘 웹 페이지들 또는 UDDI(Universal Description, Discovery and Integration;의미 서비스 검색 엔진 버전)같은 의미 웹 서비스들을 위한 검색 엔진을 들 수 있다. 다른 방법으로, 사용자들은 SSD들을 서로에게 이-메일로 전달하는 것에 의해 또는 SSD들을 P2P 네트워크를 통해 공유하는 것에 의해 SSD들(116)을 공유할 수 있다.

여기에서 설명되는 실시예들의 태양에 따르면, 교차-환경의 PIPE-WS TCS(118b)도 서비스들(112)의 교차-환경 관리를 위해 제공됨으로써, 그에 의해, 태스크들(126)/서비스들(112)을 관리하기 위한 SDSCM들(Semantically Described Service Control Mechanisms;119b)의 교차-환경 애플리케이션 클라이언트들을 제공할 수 있다. 예를 들어, 교차-환경의 "화이트 홀" 태스크 컴퓨팅 클라이언트(119b-1)의 경우, 화이트 홀(119b-1)은, 반드시 화이트 홀(119b-1)과 동일한 머신에서 실행될 필요는 없는 PIPE-WS API들(122n)의 WSDL URL들을 저장한다. 일단 오브젝트가 드래그되어 화이트

홀(119b-1) 상태가 되고 나면, 오브젝트는 (도 5a에 도시된 "Insert" 웹 서비스들(122) 중 하나를 사용해) 게시 요청들을 동일한 장치상의 그것이 아닌 또는 동일한 장치상의 그것과 함께 다른 PIPE-WS API들(122)로 송신한다. 이런 식으로, 서비스 오브젝트(115)의 의미 인스턴스가 다른 네트워크들(110)에서, 그에 따라, "교차-환경적으로" 생성되고 게시될 수 있다. 여기에서 설명되는 실시예들의 태양에 따르면, 다른 방법으로, 화이트 홀(119b-1)은 (다음에서 부연되는 바와 같이) 사용자로 하여금 사용할 PIPE-WS API들(122a-n)을 선택할 수 있게 한다. 예를 들어, 컴퓨터 환경(110a)의 PIPE-WS (122a), 원격 환경(110n)의 서비스(112)를 사용하는, "화이트 홀"(119b-1)과 같은, SDSCM(119b) 태스크 컴퓨팅 클라이언트는, 원격 환경의 PIPE-WS(122n)를 위해 웹 서비스 호(122n)가 형성될 수만 있다면, 게시될 수 있다.

PIPE-WS API(122)를 사용해 서비스들을 게시하기 위한 사용자 인터페이스 도구인 화이트 홀(119b-1)은 2 이상의 교차-환경들(110n)에서 게시 중인 서비스(112)를 다루기 위해 다수의 PIPE-WS API들(122)을 수용하도록 확장될 수 있다. 화이트 홀(119b-1)은 다음과 같은 화이트 홀(119b-1) 파라미터들을 설정하기 위한 스타트업 다이얼로그 박스 또는 옵션 설정 박스를 제공할 수 있다:

PIPE WS API(122) 펑크션 파라미터들:

1. 명칭(선택적)

2. 타깃 PIPE-WS API(122)의 URL(들)

3. (웹 서비스 호들 및 제어 UI들에 사용할) 프럭시 URL들(선택적)

4. 웹 서비스 호에서의 오브젝트 사용을 위한 옵션

화이트 홀(119b-1) GUI 설정 파라미터들:

1. GUI 옵션 나타내기

2. GUI의 컬러

3. GUI를 위한 아이콘 이미지

4. 그룹화된 또는 독립적인 GUI

하나의 화이트 홀(119b-1) GUI는, 사용자가 화이트 홀(119b-1) 상태가 된 의미 오브젝트를 서비스로서 게시하는데 어떤 PIPE-WS API(122)를 사용할 것인지를 선택하기 위한 다이얼로그 박스를 나타내는 것에 의해 다수의 PIPE-WS API들(122a-n)을 수용할 수 있다. 또는, 사용자가 어떤 아이콘이 어떤 PIPE-WS API(122)에 대응되는지를 구별하고 기억하는 것을 용이하게 하기 위해, 각각의 원격 PIPE-WS API(122a-n)에 대해 각각 상이한 컬러 또는 이미지를 가진 화이트 홀(119b-1)의 다수 인스턴스들을 제공할 수도 있다. 좀더 구체적으로, 화이트 홀(119b-1) 사용자 인터페이스 옵션들은 (경우에 따라) 제공되는 또는 이용 가능한 "구들" 사이의 시각적 및/또는 청각적 차이를 수용한다.

교차-환경(110) 서비스(112) 관리와 관련하여, PIPE-WS API(122)는 서비스(112) 관리를 위한 새로운 웹 서비스 인터페이스들(122)을 추가하는 것에 의해 확장된다. 따라서, PIPE-WS API(122)를 "서비스 관리자"(119b-2)라고 하는 SDSCM(119b)으로서 사용하는 새로운 WS TCC(Web Service Task Computing Client) 애플리케이션 클라이언트(119)가 생성된다. 서비스 관리자(119b-2)의 한가지 주된 펑크션은 PIPE-WS API들(122)을 사용해 복수개 컴퓨터 시스템 환경들(110)의 서비스들(112)을 관리하는 것이다. 비한정적 일례에서, 서비스 관리자(119b-2)의 관리 액션들은, 서비스(115, 116;112) 명칭 및 설명 변경하기, 서비스(112) 만료 시간 변경하기, 서비스(112) 발견 범위 변경하기, 서비스들(112)의 "구" 변경하기, 서비스(112) 호출 제한 변경하기 등을 포함한다. 특히, 서비스 관리자(119b-2)는, 다음과 같이, 교차-환경 서비스(112) 관리를 수행하는데 사용될 수 있다. 각각의 PIPE-WS API(122)는, 원격 사용자들이 그것을 사용해 그것의 서비스들(112)을 관리할 수 있는지의 여부를 판정하기 위한 옵션을 가진다. 옵션이 참 값으로 설정되면, 원격 사용자는 PIPE-WS API(122)를 사용자의 서비스 관리자(119b-2)에 추가할 수 있다. 서비스 관리자(119b-2) 도구로부터, 사용자는 원격 PIPE-WS API(122)에 의해 관리되는 서비스들(122)에 관한 모든 세부 사항들을 점검할 수 있고, PIPE-WS API(122)가 액세스 가능하기만 하다면, 기본적으로 모든 관리 액션들을 원격적으로 수행할 수 있다.

프레젠테이션 프로세싱 계층(104) 사용자 인터페이스들:

STEER-WS API(120) 및 PIPE-WS(122)의 구현은 WS TCC들(119)에 아주 다양한 태스크 컴퓨팅(100) 사용자 인터페이스들(104)을 제공할 수 있게 하는데, WS TCC(119)의 프레젠테이션 프로세싱 계층(104)이 태스크 컴퓨팅 미들웨어 프로세싱 계층(108)에 대한 모듈들의 구현으로부터 자유로울 수 있기 때문이다. 여기에서는, (1) 라디오 장치 사용자 인터페이스, (2) 위치(장소) 인지 아이콘(예를 들어, 벌룬)의 컴퓨터 디스플레이 스크린 그래픽 사용자 인터페이스, (3) 음성 명령 사용자 인터페이스, (4) 사용자 인터페이스에서의 다수 입/출력들, 및 (5) Tasklet-WS TCC(119a-5)를 위한 WS TCC(119)의 사용자 인터페이스(104) 일례들이 설명된다. 태스크 컴퓨팅(100) 시스템 환경은 상기 사용자 인터페이스들(104)의 임의 조합을 제공할 수 있다.

(1) WS TCC(119)의 모바일(라디오) 폰 사용자 인터페이스:

도 8은, 본 발명의 실시예에 따른, 서비스들(112)을 관리하기 위한 모바일 폰 디스플레이 스크린의 사용자 인터페이스 이미지들의 도면이다. 좀더 구체적으로, 도 8은, 서비스(112) 발견, 구성, 실행, 저장, 생성, 및 다른 서비스(112) 관련 동작들과 같은, 태스크 컴퓨팅의 핵심 기능들을 관리하기 위한 모바일 폰 디스플레이 스크린의 사용자 인터페이스 이미지들의 도면이다. 앞서 논의된 바와 같이, 여기에서 "서비스"는 펑크션(110)의 네트워킹형, 비-네트워킹형, 또는 양자인) 이용 가능한 컴퓨터 시스템의 서비스들 또는 소스들을 의미한다(즉, 여기에서 "서비스들"(112)은 컴퓨터 장치들, 컴퓨터 애플리케이션들/소프트웨어, 전자-서비스들 및 컴퓨터(또는 머신, 또는 양자의) 판독 가능 내용 일체로부터의 기능에 대한 컴퓨터 조작 실시예들을 의미한다). 모바일 폰 사용자 인터페이스의 사용자 네비게이션은 공지의 입력 유닛(들)/장치(들)(예를 들어, 음성 명령/제어를 위한 마이크로폰, 키보드/키패드, 포인팅 장치(예를 들어, 마우스, 포인터, 스틸러스), 터치 스크린 등) 및 출력 유닛(들)/장치(들)(예를 들어, 컴퓨터 디스플레이 스크린, 스피커(들) 등)에 따를 수 있다.

여기에서 설명되는 실시예들에 따르면, 모바일 폰 사용자는 임의의 웹-인에이블형 모바일 또는 라디오 통신 폰(800)에서 태스크 컴퓨팅을 경험할 수 있다. Mobile-PhoneSTEER-WS TCC(119a-2)라고 하는 태스크 컴퓨팅의 모바일 폰 STEER 웹 서비스들의 태스크 컴퓨팅 클라이언트가 모바일 폰의 태스크들(126)을 관리하기 위해 제공된다. 통상적으로 본 발명에 따르면, Mobile-PhoneSTEER-WS TCC(119a-2)는 모바일 폰들을 위한 웹 WS 클라이언트(119)인데, 이는, J2ME(Java 2 Platform, Micro Edition), BREW(Binary Runtime Environment for Wireless), 그 언어로 기입된 애플리케이션들이 모바일 폰에서 실행될 수 있도록 모바일 폰에 설치 가능할 수 있는 임의의 다른 언어, 또는 그것에 관한 임의 조합들과 같은, 모바일 폰에 설치 가능하고 모바일 폰에서 실행 가능한 임의의 컴퓨터 프로그래밍 언어로 구현될 수 있다. 웹 클라이언트들(119a-2)은 웹-인에이블형 폰들(Web-enabled phones)에 사전-설치될 수도 있고 그들이 프로세싱할 수 있는 일종의 HTML(Hyper Text Markup Language) 맥락들에서 광범위하게 달라질 수도 있다. 좀더 구체적으로, 웹 클라이언트들(119a-2)은, 정보를 디스플레이하기 위해 마크업 언어의 문서를 해석하는 임의의 웹 및/또는 Wap(Wireless Application Protocol) 브라우저 소프트웨어를 통해 구현될 수도 있다. 여기에서 설명되는 실시예들의 다른 태양에 따르면, Mobile-PhoneSTEER-WS TCC(119a-2)는 맞춤형 클라이언트 애플리케이션일 수 있다. Mobile-PhoneSTEER-WS TCC(119a-2)의 프레젠테이션 프로세싱 계층(104)은, "TCC II"라고도 하며, 그것의 전체 내용들이 여기에 참고 문헌으로서 포함되어 있는, 본 출원의 FUJITSU LIMITED 양수인에 의해 소유되는, Ryusuke Masuoka, Yannis Labrou, 및 Zhexuan Song에 의해 TASK COMPUTING이라는 명칭으로 2003년 12월 12일에 U.S. PTO에 출원된, 관련된 공동 소유의 미국 특허출원 제 10/733,328호(Attorney Docket No. 1634.1007)에 설명되어 있는, 태스크 컴퓨팅 또는 "Hosted STEER"를 위한 웹-기반 사용자 인터페이스와 유사하게 구현될 수도 있다.

그러나, 여기에서 설명되는 실시예들에 따르면, Mobile-PhoneSTEER-WS TCC(119a-2)는 미들웨어 서버 프로세싱 계층(108)과 인터페이싱하기 위해 WS API(106)에 의존하고, 태스크들(126)을 관리하기 위한 Mobile-PhoneSTEER-WS TCC(119a-2)의 룩앤필(look and feel)이, 훨씬 작은 스크린 사이즈와 같은, 모바일 폰(900)의 특정 요구 사항들을 충족시키기 위해 채택된다. 도 9는, 사용자가 (여러 페이지들의) 발견된 서비스들(112)의 리스트를 요청할 수 있고, 태스크(126)를 생성/구성할 수 있으며, 태스크(126)를 실행할 수 있는, 태스크 컴퓨팅 모바일 폰을 사용하는 사용자 경험의 일례를 예시한다. 모든 동작들은 모바일 폰(900)의 비교적 작은 디스플레이 영역에서 발생한다.

모바일 폰(800)에서 태스크 컴퓨팅을 수행하는 동안, 웹 액세스를 위한 네트워크 접속은 IR, BLUETOOTH, (WLAN-인에이블형 모바일 폰들을 위한) WLAN 또는 모바일 네트워크(GSM, CDMA 등)일 수 있다. 네트워크의 선택은 모바일 폰(800)에서의 Mobile Phone-STEER-WS TCC(119a-2)의 동작에 영향을 미치지 않는데, 태스크 컴퓨팅을 위해 필요한 모든 통신은, (제한없이) HTML 또는 다른 마크업 언어 기반 포맷의 형태로 데이터를 디스플레이할 수 있는 HTTP(Hypertext Transfer Protocol)와 같은, 하이/애플리케이션 레벨의 통신 프로토콜들에 의해 수행되기 때문이다.

다음으로는, 모바일 폰(800) UI가 설명된다. 동작 802에서, 모바일 폰(800)의 사용자가 (그러한 환경(110)을 위한 URL을 입력하는 것에 의해) 모바일 폰 브라우저 소프트웨어를 컴퓨팅 환경(100)으로 유도할 경우, 사용자는 그 환경(110)에서 이

용 가능한 서비스들(112)의 리스팅을 보게 된다. 리스팅은 스크롤링 가능한 단일 페이지로 또는 그들에 도달하기 위해서는 사용자가 "후속 페이지" 링크를 선택할 것이 요구되는 다수 페이지들로 제시될 수도 있다. 동작 804에서, 사용자가 서비스(112)를 선택하면, 동작 806에서, 선택은, 현재의 서비스(112) 구성이라고 하는, 서비스(112) 구성(즉, 태스크 생성)의 요소가 된다(예를 들어, 도 8의 일례에서는 사용자에 의해 News.com이 선택된다). 동작 808에서는, 서비스(112)가 선택된 후, 사용자가 선택된 서비스(들)과의 구성에 등장할 수 있는 서비스들(112;예를 들어, open, print, save, store favorite, view locally)만의 리스팅을 포함하는 페이지로 유도된다. 동작 808에서 디스플레이된 리스트는, 전과 같이, 스크롤링 가능한 단일 페이지로 또는 다수 페이지들로 표시될 수 있다. 동작 808에서, 서비스들(112)의 리스팅을 갖춘 페이지 상부에는, 앞서 선택된 서비스들(예를 들어, News.com)이, 그들이 유효한 서비스(112) 구성에 등장할 수 있는 순서로 디스플레이된다. 동작 808에서, 디스플레이는 현재의 서비스(112) 구성이고, 그것은 배너로서 디스플레이를 가로질러 스크롤링할 수도 있다. 사용자가 서비스(112)를 선택할 때마다(예를 들어, 동작 810, 812, 814), 디스플레이 페이지는, 현재 구성에 사용될 수 있는 추가 서비스들(112)이 존재하지 않을 때까지, 현재의 서비스(112) 구성 및 현재의 서비스(112) 구성에 등장할 수 있는 서비스들(112)의 리스팅을 디스플레이하도록 업데이트되는데, 추가 서비스들(112)이 존재하지 않는 시점이 되면, 동작 816에서, 사용자는 서비스(112) 구성을 실행할 옵션을 가진다. 좀더 구체적으로, 동작들(802 내지 814)은 모바일(라디오) 장치를 통해 태스크(126)를 구성하기 위한 동작들이다. 도 8의 일례의 동작 814에서는, 태스크(126)의 현재 구성에 사용될 수 있는 더 이상의 추가 서비스들(112)이 존재하지 않지만, 추가 서비스들(112)이 이용 가능하다면, 추가 서비스들(112)은 동작들(804 내지 814)과 유사하게 열거될 것이다. 여기에서 설명되는 실시예들의 태양에 따르면, 사용자는, 구성이 비록 완결되지 않았다 하더라도, 구성이 실행 가능하다면, 구성을 실행할 수 있다. 동작 816에서는, 태스크(126)의 실행 상태가 디스플레이된다. 동작 820에서는, 태스크(126)의 실행 완결이 디스플레이된다.

(2) 위치 인지 아이콘(예를 들어, 벌룬) 사용자 인터페이스(UI):

도 9는, 본 발명의 실시예에 따른, 컴퓨터 디스플레이 스크린의 위치-인지 벌룬 그래픽 사용자 인터페이스의 이미지이다. 특히, 도 9는 STEER-WS-SIS TCC(119a-3)라고 하는 STEER-WS TCC(119a)의 예시적 GUI이다. (제한없이) 사용자가 위치하고 있는 주변 영역과 같은, 영역의 공간 이미지를 디스플레이하는 컴퓨터 디스플레이형 그래픽 사용자 인터페이스의 STEER-WS-SIS TCC(119a-3) 및 사용자 영역의 디스플레이된 이미지는 사용자 영역에서 발견된 서비스들(112)의 선택 가능한 그래픽 디스플레이 표현들과 중첩된다.

도 9에서, 위치/장소(902)내의 또는 위치/장소(902)에서 이용 가능한/발견된 서비스들(112)은 위치/장소(902)의 대응되는 영역에 아이콘들(904a-n)로서 (여기에서 설명되는 실시예들에 따르면, 벌룬 아이콘들(904)로서) 표현된다(예를 들어, 도 9에서, 오피스 플로어 맵은, 오피스내의 다양한 영역들에서 서비스들(112)이 발견 가능한 위치/장소(902)이다). (잠재적으로) 대다수 서비스들(112)을 3D 좌표들로써 디스플레이하고 직관적인 사용자 컴퓨터 디스플레이 인터페이스를 실현하기 위해, 다음의 메커니즘들이 제공된다:

1. 서비스들(112)은 동일한 컴퓨터 디스플레이 스크린의 시각적 컴포넌트들(즉, 디스플레이된 아이콘), 동일한 유형이지만 상이한 스타일들(컬러들, 사이즈, 폰트들 등)을 가진 컴포넌트, 또는 (경우에 따라) 각각의 서비스(112)를 위한 상이한 시각적 컴포넌트로써 표현된다. 도 9에서는, 가변 사이즈들 및 컬러들의, "벌룬"이라고 하는, 시각적 컴포넌트가 사용된다.

2. 위치를 가진 서비스(112)는 디스플레이된 맵의 그 위치에 배치된다. 위치를 갖지 않는 서비스(112)는 표 또는 소정의 다른 편성 방식(906)으로 맵의 외부에 배치된다.

3. 벌룬 위치들이 중첩하거나 서로 가깝다면, 벌룬 위치들을 임의 추출한다.

4. 벌룬들을 일반적으로는 작은 상태로 유지하고, 조작되거나 커서가 그러한 벌룬 부근에 위치할 때, 확대된 벌룬(908)을 디스플레이한다.

5. 벌룬의 음영(912)으로써 물리적 높이(Z 좌표)를 표현한다. 서비스(112)의 위치가 물리적으로 높을수록, 음영(912)은 좀더 커지고 그리고/또는 흐려진다. 위치에서의 서비스(112)에 대한 Z 좌표를 강조하기 위해 다른 디스플레이 기술들이 사용될 수도 있다.

6. 벌룬들 중 하나를 드래그하여 그것을 나머지 벌룬상에 드롭하는 것에 의해, (잠재적으로 그들 사이의 번역 서비스들을 갖춘) 2-서비스 구성의 실행을 표현한다.

7. 벌룬이 선택될 때, 구성 가능한 벌룬들(서비스들;112)만이 그들의 컬러들을 유지하거나 강조된다. 나머지들은, 각각, 구성 불가능한 서비스들(112)을 지시하는, 그레이와 같은, 다른 컬러로 변하거나 동일한 상태를 유지한다.

8. 벌룬을 터뜨리는 핀의 은유를 사용해 서비스(112)의 제거를 표현한다.

9. 핀이 선택되면, 삭제 가능한 벌룬들(서비스들;112)만이 그들의 컬러들을 유지하거나 강조된다. 나머지들은, 각각, 삭제 불가능을 지시하는, 그레이와 같은, 다른 컬러로 변하거나 동일한 상태를 유지한다.

10. (통신 언어 선택과 관련하여 상술된 바와 같이) 사용자 언어 선택에 기초해, 서비스(112) 명칭들 및 설명들을 디스플레이하는데 사용되는 언어를 변경한다.

11. 사용자가 이 인터페이스에서 생성될 수 있는 것보다 좀더 복잡한 구성들을 생성하고자 한다면, (좀더 포괄적인) STEER-WS-SIS TCC(119a-3) 인터페이스를 표시하기 위한 버튼을 제공한다.

따라서, 도 9는 위치-인지 아이콘(예를 들어, 벌룬)의 사용자 인터페이스의 일례이다. 도 9의 일례는 디스플레이된 "벌룬"을 이용 가능한 서비스(112) 표현으로서 사용하지만, 본 발명은 그러한 구성으로 한정되지 않으며, 디스플레이된 사용자 영역 이미지상에 중첩된 임의의 디스플레이 표현이 사용될 수 있다. 이것은 이벤트-구동의 객체-지향적 프로그래밍에 의해 구현될 수 있다. STEER-WS-SIS TCC(119a-3)가 시작될 때, 그것은 숫자들과 내부 데이터를 초기화하고, 이벤트-핸들링 코드들을 적절한 이벤트들을 위해 설정한다. 기본적으로, 이벤트들과 이벤트-핸들링 코드들의 그러한 쌍들은 상기 리스트의 항목들에 대응된다. 이벤트-핸들링 코드들에서, STEER-WS API(120) 및 PIPE-WS API(122)와 같은, TCE-WS API(106)로의 적절한 웹 서비스 호들이 형성된다. 다음으로는, 서비스들(112)의 이용 가능성들에 변화들이 있을 때 벌룬들 및 내부 데이터를 업데이트하기 위해, 이벤트 루프가 이벤트들 및 발견을 위한 다른 루프(도 3a)에 주의를 기울일 것이다. 도 9에서는, "업데이트" 버튼(909)의 선택 가능한 그래픽 디스플레이들이 디스플레이된 정보를 업데이트한다. 여기에서 설명되는 실시예들의 태양에 따르면, 사용자 디스플레이들 중 하나로 디스플레이된 정보를 TCC(119)에 의해 업데이트하는 것은 자동적일 수 있다. "Language" 버튼(910)의 선택 가능한 그래픽 디스플레이는, (다중-언어 태스크 컴퓨팅(100) 시스템에 따라 다음에서 부연하는) 일본어와 같은, 선택된 구어에 따라 디스플레이되는 정보를 제공한다.

(3) 음성 사용자 인터페이스(UI):

도 10은, 본 발명의 실시예에 따른, 태스크 컴퓨팅 음성 인식의 상태 흐름도이다. 음성은, 사용자의 클라이언트 장치들이 흔히 사이즈가 제한되어 키보드 또는 키패드와 같은 종래의 입력 기술들이 사용될 수 없거나 불편한 그리고/또는 사용자가, 운행 중일 때와 같이, 시각적 관심을 제공할 여유가 부족할 수 있는 편재형 컴퓨팅 환경을 위한 아주 중요한 사용자 인터페이스일 수 있다. 여기에서 설명되는 실시예들에 따르면, 음성이 태스크 컴퓨팅(100), 특히 서비스(112) 구성들을 태스크(126)로서 직접적으로 실행 중인 태스크 컴퓨팅(100)에 대해 작용하기 위한 지시들을 제공하는데 사용되는 WS TCC (WS Task Computing Client;119)인 VoiceSTEER-WS TCC(119a-4)가 생성된다. 태스크 컴퓨팅(100)에서, 서비스(112) 구성들은 자연어에 따라 서비스(112)의 명칭을 정의하는 것에 의해 문장 구조와 호환 가능하도록 설계되기 때문에, 그것의 구성은 음성 인식기 시스템에 의해 프로세싱될 수 있는 자연어 문장을 생성하는데; 다시 말해, 서비스들(112)의 구성들은, 도 10에 도시된 바와 같은, 음성 인식 상태 문법도로 매핑될 수 있다.

좀더 구체적으로, VoiceSTEER-WS TCC(119a-4)는, 예를 들어, C#으로 개발된 음성-구동의 사용자 인터페이스이다. 사용자는 VoiceSTEER-WS TCC(119a-4)로 발화하는 것에 의해 태스크들을 요청할 수 있고; 구현된 VoiceSTEER-WS TCC(119a-4)는 TCE-WS API(106)로의 호출들과 함께 MICROSOFT의 AGENT AND SPEECH SDK를 사용한다. 각각의 서비스(112)는 구(통상적으로, 서비스(112) 명칭)로 매핑되고, 일단 VoiceSTEER-WS TCC(119a-4)가 "문장을 청취하고 나면"(음성 문장을 인식하고 나면), VoiceSTEER-WS TCC(119a-4)는 문장을 서비스들(112)의 문장으로 매칭하려 할 것이다. 다음으로, 서비스(112) 시퀀스에 기초해 태스크(126)가 구축되고 실행된다. 음성 인터페이스의 한가지 어려움은 인간 음성의 인식율인데; 이 비율을 허용 가능한 레벨로 상승시키는 문법 세트를 정의하는 것이 아주 중요하다. 다른 어려움은, 비록 "Print on Office Printer" 및 "My Video"가 유효한 서비스(112) 명칭들이라 하더라도, 무시되어야 하는 "Print on Office Printer My Video"와 같은, 의미론적으로 무효한 명령들(태스크들)을 식별하고 필터링하는 방법이다. 여기에서 설명되는 실시예들에 따르면, VoiceSTEER-WS TCC(119a-4)는 완전한 문장을, 미들웨어 프로세싱 계층(108)의 중앙 모듈(402)에 의해 생성되고 STEER-WS API(120)를 통해 이용 가능화되는 "문법 체계도"(도 10)에 따라, 유효한(실행 가능한) 태스크들(126)과 매칭한다.

도 10은, 본 발명의 실시예에 따른, 음성 태스크 컴퓨팅의 상태 흐름도인, 문법 체계도의 일부이다. 도 10에서, 서클은 의미론적으로 설명된 서비스(112)의 펑크션 특징을 문법 상태로서 표현한다(예를 들어, 서비스 "open"(112)은 "File" 데이터 오브젝트 유형을 입력으로서 소비하고 - 일례로서의 "File"은, 그러한 파일로의 URL 링크를 포함하는, 컴퓨터의 파일을 설명하기 위한 의미 유형이다). 시작 상태(1002)는 "S"로써 마킹되고 종료 상태(1004)는 이중 서클로써 지시된다. 서비스

들(112)은 문법 상태들을 접속하는 엣지들(라인들)로써 표현된다. 예를 들어, 비지니스 사무실 환경을 위한 완전한 문법 체계도는 50가지 이상의 서비스들(112), 100개 이상의 엣지들 및 12개 이상의 상태들을 가질 수도 있다. 시작 상태(1002)로부터 종료 상태(1004)까지의 경로들은 의미론적으로 유효한 태스크들(126)을 표현하고; 예시적인 비지니스 오피스 환경에서, 완전한 문법 체계도는 100개 이상의 이러한 경로들, 즉, 사용자가 실행할 수 있는 1000개 이상의 태스크들(126)을 가질 수도 있다. "문법 체계도"는 다음의 규칙들에 기초해 발생된다:

1. 서비스(112)가 출력을 갖지 않으면, 서비스(112)는, 도 11의 "Open", "View on Projector", 및 "Add to Outlook"과 같은, 시작 상태에 수반되는 엣지이다.

2. 서비스(1120가 입력을 갖지 않으면, 서비스(112)는, 도 11의 "My File" 및 "My Contact"와 같은, 종료 상태를 포인팅하는 엣지이다.

3. 2가지 서비스들(112 A 및 B)이 구성될 수 있다면, 도 10의 "Business Address of", "Map of", 및 "Weather Info of"와 같은, A를 입력으로 그리고 B를 출력으로 갖는 상태가 존재해야 한다.

"문법 체계도"의 중요한 일 속성은, 그것이 음성 인식 엔진의 문법 규칙 세트와 일대일 매핑을 갖는다는 것이다. 좀더 구체적으로, 문장은, 문장이 엣지들의 명칭들에 대한 결합이도록, 문법 체계도에서 시작 상태로부터 종료 상태까지의 경로가 발견될 수 있을 경우에만, 의미론적으로 의미있는 명령(즉, 유효한 태스크(126))이다. 예를 들어, "Open My File" 또는 "View on Projector Weather Info of Business Address of My Contact"은 유효한 명령들 또는 태스크들(126)이다. "문법 체계도"는 온톨로지 및 서비스들의 의미 (다시 말해, SSD(116)에 기초해 판정되는) 설명들에 의해서만 판정된다. SSD(116)의 사용은 인식율을 상당히 증가시키고, 그에 따라, 의미론적으로 무효한 명령들 또는 무효한 태스크들(126)은 완전히 방지될 것이다. 음성 UI는 태스크 컴퓨팅(100)의 ITS(Intelligent Transportation System) 애플리케이션들을 위해 바람직할 것이다.

도 11은, 본 발명의 실시예에 따른, 태스크 컴퓨팅을 위한 음성 인식도("문법 체계도")를 발생시키기 위한 예시적 의사-코드이다. 먼저, 동작 1102에서, 출력을 갖지 않는 서비스들(112)이 발견된다. 동작 1104에서는, 구성될 수 있는 다른 모든 서비스들(112)이 회귀적으로 발견된다. 특히, 발견된 서비스(112) 각각을 위한 동작(1106)에서, 서비스(1120가 입력을 갖지 않으면, 서비스(112)는 종료 상태로의 엣지 링크로서 지시된다.

VoiceSTEER-WS TCC(119a-4)의 예시적 어휘는 다음에 따른 것일 수 있다:

1. 서비스(112)의 명칭

2. ("show services" 또는 "leave Task Computing"과 같은) VoiceSTEER-WS TCC(119a-4)로의 태스크 컴퓨팅 명령

3. ("Move up", "Click XXX"(여기에서, XXX는 버튼의 명칭이다)와 같은) 클라이언트 동작 명령

4. (웹 페이지의 제 4 제어를 클릭하기 위한 "Click 4"와 같은) 웹 페이지 명령

5. (사용자가 정보를 입력하기 위한 "a" 또는 "9"와 같은) 문자 및 숫자들

작은 어휘 세트를 갖는 것의 2가지 장점들이 존재한다. 첫번째, 음성 인식 시스템을 트레이닝하기가 아주 용이하다. 두번째, 인식율이 높은데, 서로 구별해야 할 단어들 및 문장들이 좀더 적기 때문이다.

태스크 컴퓨팅 서비스(112) 구성은 문법 구조를 가질 수 있기 때문에, 음성 인식 시스템에 의한 인식율은, 다음과 같이, 훨씬 더 향상될 수도 있다. 어떤 것은 문장의 각 컴포넌트를 한번에 하나씩 인식하는 경우보다 전체 문장을 한번에 인식하는 경우에 대해 증가된 인식율을 가질 수 있다. 이것은 기본적으로, 문장의 일 부분에서는 인식이 실패한다 하더라도, 다른 부분이 인식된다면, 여전히 그것을 인식할 수 있기 때문이다(즉, 결합 확률을 고려해야 한다). 예를 들어, "Open My File"의 발화된 문장을 살펴본다. "Open"을 위한 인식율은 a이고 "My File"을 위한 인식율은 b라고 가정한다. 인식이 "Open" 및 "My File"을 위해 개별적으로 수행된다면, "Open"을 먼저 인식해야 하므로, 인식율은 절대 a를 초과할 수 없다. 그러나, 시스템이 전체 문장, "Open My File"을 인식하고자 한다면, 인식율은 1-(1-a)(1-b)= a + b - ab = a + b(1-a)일 것이다. a는 1 미만이고 b는 양수이므로, 인식율은 항상 a보다 크다. 따라서, 음성 인식율은, 문장이 길어질수록 더 높아질 수 있다.

따라서, 여기에서 설명되는 실시예들에 따르면, VoiceSTEER-WS TCC(119a-4)는 서비스 경로들 사이에 "a" 및 "the"를 위한 추가 경로들을 추가한다. 이에 의해, 사용자는 좀더 자연스러운 문장들을 가질 수 있다. 예를 들면, 다음과 같다:

Computer, View on Projector (the) Web Page of (the) Manager of (the) Task Computing Project

명령 문장에서 하나 이상의 번역기 서비스들(112)을 생략할 수도 있는데, 애매한 명령 문장도 여전히 인식 가능할 수 있기 때문이다. 예를 들어, 도 10에서는, "Weather Info of" 및 "Map of" 서비스들(112)과 "File" 노드로부터 "Contact" 노드로의 직접적인 접속들을 갖는 것에 의해, 명령 문장의 "Business Address of" 서비스(112)는, 다음과 같이, 사용자에 의해 생략될 수도 있다.

Computer, View on Projector (the) Weather Info of My Contact

그리고, VoiceSTEER-WS TCC(119a-4)는 여전히 문장을 인식할 수 있다. 이 경우, 사용자가 원하는 것에 불명료함이 존재한다면, VoiceSTEER-WS TCC(119a-4)는 사용자에게 질문하는 것에 의해 명료하게 할 수 있다. 이 경우, "Home Address of" 서비스(112)와 "Address" 노드로부터 "Contact"으로의 다른 링크가 존재하고 사용자가 상기 문장을 질문할 경우, VoiceSTEER(119a-4)는 사용자에게 유효한 태스크(126)를 구성하기 위한 명료화 질문을 질의할 수도 있다:

Do you want to (1) "View on Projector Weather Info of Business Address of My Contact" or (2) "View on Projector Weather Info of Home Address of My Contact?"

접속들의 방향을 반전하고 인식 도면의 시작 노드 및 종료 노드를 대체하는 것에 의해, VoiceSTEER-WS-TCC(119a-4)는, 일본어와 같은, 명사 + 동사 순서의 다른 언어들을 인식할 수 있다.

(4) 사용자 인터페이스의 다중 입/출력:

"Fax" 서비스(112)는 팩스 번호 및 파일을 입력들로 취하는 것과 같이, 일부 서비스들은 다수 입력들을 가진다. 본 발명의 태양에 따르면, STEER-WS TCC(119)는 태스크(126) 실행 동안 회귀적인 방법으로 사용자로부터 누락 입력들을 요청한다. 예를 들어, 태스크 컴퓨팅(100) 시스템의 컴퓨터와 사용자 사이의 다음 대화들을 살펴 본다:

```
User: Computer, "Fax" (the) "Fax number of" "Ryusuke Masuoka"
Computer: What is the "File" for "Fax" service?
User: Use "My File"
Computer:  Will execute "Fax" "My File" with "Fax number" of
"Ryusuke Masuoka" as "File" for "Fax" Service
Computer: Start execution
```

```
User: Computer, "Fax" "My File"
Computer: What is the "Fax Number" for "Fax" service?
```

```
User: Use "Fax Number of" "Zhexuan Song"
Computer: Start execution
```

제 1 대화에서, 사용자는 3가지 서비스들: "Fax", "Fax number of", 및 "Ryusuke Masuoka"(서비스(112)를 제공하는 연락처)을 포함하는 태스크(126)를 호출하고자 한다. 태스크(126)(즉, 서비스(112) 구성)를 점검하는 것에 의해, 컴퓨터는, 서비스 "Fax"가 하나는 "Fax Number"이고 다른 하나는 "File"인 2가지 입력들을 취한다고 판정한다. "Fax Number"는 시퀀스에 의해 제공되므로, 컴퓨터는 이제 사용자에게 나머지 입력 "File"을 요청한다. 그 다음, 사용자는 서비스(112) "My File"로부터 입력을 취할 것을 컴퓨터에 지시한다. 모든 입력들이 특정된 후, 컴퓨터는 태스크(126)의 실행을 시작한다.

제 2 대화는, 처음에 "Fax Number"가 특정되지 않은 상황에 관한 것이다. 이 경우, 사용자에게는 부가적으로, "Fax Number"를 제시하는 서비스(112) 구성을 제공할 것이 요청된다.

상기 2가지 일례들에서는, VoiceSTEER-WS TCC(119a-4)의 인터페이스를 위해 사용되는 언어 특징들로 인해, 사용자가 한 문장의 시작에서부터 모든 입력들을 제공하는 것이 대개는 용이하거나 자연스럽지 않다. 따라서, 그러한 상황들이

검출될 때, 실행 엔진(406)은 VoiceSTEER-WC TCC(119a-4) 컴퓨터를 제어하여 사용자에게 실행 이전 및/또는 이후에 더 많은 입력들을 프롬프팅(prompting)한다. 이것은 복잡한 서비스(112) 구성도를 공간적으로, 사용자와 컴퓨터 사이에서 시간적으로 확장하는 사용자 상호 작용으로 매핑하는 것으로 간주될 수 있다.

다수 출력들을 다루는 것도 본질적으로는 동일하다. 예를 들어, "Bioinformatics Talk"라고 하며, "Contact" 데이터 오브젝트를 그것의 대화자(speaker)로서, "Schedule" 데이터 오브젝트를 그것의 스케줄로서, 그리고 "File" 데이터 오브젝트를 그것의 프레젠테이션 소재로서 발생시키는 서비스가 존재한다고 가정한다. VoiceSTEER-WS TCC(119a-4)를 사용한다면, 다음과 같다:

```
User: Computer, "Add (Schedule) into PIM" (the) "Bioinformatics
Talk"
Computer: What do you want to do with the "Contact" from
"Bioinformatics Talk"?
User: Use "Tell Me"
Computer: What do you want to do with the "File" from
"Bioinformatics Talk"?
User: Use "View on Projector"
Computer: Start execution
```

상기 태스크(126) 구성 시나리오에서, 컴퓨터는, 태스크(126)를 추가적으로 정의하기 위한 사용자 프롬프트들을 통해, 3개의 별도 서비스들 "Add (Schedule) into PIM", "Tell Me", 및 "View on Projector"를 사용해, PIM에 스케줄을 추가할 것이고, 대화자 연락처를 판독해 낼 것이며, 프로젝터상에 프레젠테이션 소재를 표시할 것이다.

2 이상의 입/출력들을 가진 서비스들(112)의 경우 또는 다수 입/출력들을 가진 서비스(112)가 하나 이상일 경우, 그 절차는 유사하다. 이 기술은 태스크(126)가 잘 정의될 때까지 회귀적으로 사용될 수 있다. 다른 방법으로, 컴퓨터는 서비스(112) 구성들의 실행 가능 부분들은 무엇이든 실행할 수 있고, 필요한 추가적 서비스(112) 구성들을 특정하지 않으면서, 실행 가능한 다른 부분들이 존재하지 않는지를 사용자에게 질문한다.

상기 다수 입/출력들의 설명은 음성 인식의 맥락에서 설명되지만, 본 발명은 그러한 구성으로 한정되지 않으며 그 기술들이 그래픽 및 다른 사용자 인터페이스들을 사용하는 다른 태스크 컴퓨팅 클라이언트들(119)에도 적용될 수 있다. 예를 들어, 태스크 컴퓨팅 클라이언트들은 누락 서비스들(112) 및/또는, (제한없이) 서비스(112)의 데이터 오브젝트 입력들 및 출력들과 같은, 서비스들(112)의 추가적 펑크션 특징들을 요청하는 팝업창들을 띄울 수도 있다. 예를 들어, 다수 입/출력들을 다루기 위한 태스크로서 구성된 서비스들(112)의 방향성 그래프가 디스플레이되어 있는 태스크(126) 구성 GUI 구획(144;task 126 construction GUI pane)을 나타내는 도 1b를 참고한다.

(5) 태스크렛 TCC(119a-5)

태스크렛 TCC(119a-5)는, 서비스(들) 또는 서비스 구성(들)(태스크(들);126)의 OWL-S 파일들을 실행하는 초경량 가중치의 TCC(Task Computing Client;119)이다. 태스크렛 TCC가 명령 라인으로부터의 파일들을 포함하는 OWL-S 파일들을 실행하게 하는 다른 방법들 중에서, 선호되는 방법은 실행될 OWL-S 파일들을 더블 클릭하는 것(또는 소정의 적절한 다른 OS 동작들)에 의해 태스크렛 TCC를 호출하는 것이다. 태스크렛 TCC가 OWL-S 파일들을 판독할 때, 태스크렛 TCC는 STEER-WS API들(120)을 사용하는 것에 의해 서비스들 또는 서비스 구성들을 실행할 것이다. 태스크렛 TCC는 자신의 윈도내에 서비스 펑크션(115)의 제어 UI들을 나타낼 수도 있다. 특히, 도 2를 참조하면, 태스크렛 TCC(119a-5)는 OWL-S 설명을 실행하기 위해 "executeOWLS" API(120)를 호출한다.

Advanced SSD(Advanced Semantic Service Description):

서비스들(112)의 다양성을 지원하기 위한 SSD(Semantic Service Description;116)로, (1) 서비스(112) 입력을 위한 완화형(Relaxed Type), (2) 서비스들(112)의 위치들, (3) 다수-언어 서비스들(112), 및 (4) 서비스(112) 관리 펑크션의 향상된 사양들이 다음과 같이 제공된다:

(1) 완화형:

일부 서비스들(112)은 작은 서브세트를 제외한 광범위한 입력을 수용한다. 예를 들어, 서비스 "View on Projector"(112)는 Audio File 및 Video File을 제외한 File을 펑크션 특징으로 수용하는데, 여기에서, Audio File 및 Video File은 File의 서브세트이다. "View on Project"의 입력이 (File - Audio File - Video File)로서 표현된다면, 또 하나의 새로운 문제점, 즉, "My File"과 같은 다른 서비스(112)가 File을 출력으로서 발생시킬 때, 추론 엔진은 구성될 수 있는 2가지 서비스들(112)이 존재한다는 것을 알지 못한다는 문제점에 직면할 수 있다.

그러한 문제점의 원인은 현재의 서비스 설명 언어의 설명력이 제한적이며 구성 조건들이 지나치게 엄격할 수 있다는 것이다. 이 문제점을 위한 우리의 해결책은 서비스(112)를 위해 입력의 2가지 유형들을 지원하는 것에 의해 현재의 서비스 설명 언어를 확장하는 것이다. 일 입력은 입력의 정확한 도메인인 파라미터형 T_p라고 하고, 나머지 제 2 입력은, 이 또한 입력이 해당될 수 있는 좀더 큰 도메인인 완화형 T_r이라고 한다. 예를 들어, 입력 유형 T_i은 다음과 같은 경우라면 수용 가능하다:

1. T_i가 완화형 T의 서브세트이다.

2. T_i와 파라미터형 T_p간의 상호 작용이 널(null)이 아니다.

예를 들어, 서비스(112) "View on Projector"에서, T_r은 "File"이고 T_p는 ("File" - "Audio File" - "Video File")이다. 입력 유형 "File"은 수용 가능하다. "File"의 서브클래스인 "WebPage" 또한 수용 가능하다. 그러나, "Audio File"은 거부된다. "File"의 수퍼 클래스인 "Thing" 또한 거부된다.

완화형을 사용하는 "View on Projector" 서비스(115)의 일부 SSD(116) 일례는 다음과 같다:

```
<process:Input rdf:ID="URLInput">
  <process:parameterType      rdf:resource="http://www.company-
1.com/tce/ontologies/2004/03/object.owl#ViewableFile"/>
  <Company-1:relaxedType      rdf:resource="http://www.Company-
1.com/tce/ontologies/2004/03/object.owl#File"/>
</process:Input>
```

STEER 추론 엔진에서는, 완화형이 지원된다. 이러한 완화형은 다음과 같이 구현될 수 있는데: STEER가 서비스들을 구성할 때, STEER는 완화형이 존재하는지를 점검한다. 완화형 파라미터가 존재하지 않으면, STEER는 보통의 알고리즘을 사용해 서비스들과 실행을 매칭시킨다. STEER가 서비스 A, 서비스 A를 선행하는 서비스 B를 위한 완화형을 찾아내면, 서비스 B는, B의 출력이 A의 입력 완화형의 서브클래스이고 B의 출력이 A의 입력 파라미터형과 비-공백 중첩(non-empty overlapping)을 가질 경우에만, 서비스 A와 매칭된다. STEER가 구성을 실행할 때, STEER는, B의 출력이 실제로 A의 입력 파라미터형에 해당되는지를 알아보기 위해, B의 출력으로 A를 호출하기 전에, B의 출력을 점검한다.

(2) 위치

SSD(116)에 위치 정보를 포함시키는 것이 또 하나의 새로운 사양이다. 위치 정보는 2D-, 3D-유클리드 좌표계들 또는 다른 임의 좌표계의 좌표들, 좌표계를 위한 기준, 및/또는 위치에 대한 텍스트 설명을 포함할 수 있다. 위치에 관련된 "View on Projector"의 일부 SSD(116)는 다음과 같다:

```
<Company-1:locatedAt>
  <Company-1:Location>
    <profile:sParameter>
      <geoF:Point rdf:ID="ViewServicePosition">
        <rdfs:label>On the table of Conference Room, Company-
1</rdfs:label>
      <geoF:xyzCoordinates>15, 98, 98</geoF:xyzCoordinates>
<geoC:hasCoordinateSystem rdf:resource="http://www.company-
1.com/tce/ontologies/2004/03/geo.owl#MyCoordinateSystem" />
      </geoF:Point>
    </profile:sParameter>
    </Company-1:Location>
  </Company-1:locatedAt>
```

서비스(112) 발견시에, TCC(119)는, TCE-WS API(106)(예를 들어, findAllServices 및 getServiceProperty(120);도 2)를 통해, SSD(116)를 벗어난 위치 정보를 추출하고 사용자를 위한 서비스들(112)의 공간 기반 필터링 또는 (일례가 상술된 위치-인지(벌룬 UI)인) 표현을 지원한다.

서비스들(112)이 그들의 위치를 변경함에 따라, 서비스들(112)은 UPnP(및 다른 발견 메커니즘)를 사용해 새로운 위치로써 그들의 서비스(112) 설명들을 업데이트할 수 있다. 이것은, 위치 변경이 그렇게 자주 발생하지 않는다면, 실행 가능한 옵션이다. 위치 변경이 빈번하다면, 다음에서 논의되는 서비스(112) 관리 펑크션을 사용하는 것이 좀더 효율적이다.

(3) SSD(Semantic Service Descriptions;116) - 통신 언어들(다수-언어들):

여기에서 설명되는 태스크 컴퓨팅(100) 실시예들은, (제한없이) 영어, (약식) 중국어, (전통적인) 중국어, 그리스어, 인도어, 일본어, 한국어, 스페인어 및 터키어의 구어들(spoken languages)과 같은, 임의의 통신 언어를 지원하고, 그에 의해, 언어 독립적인 태스크 컴퓨팅(100)을 제공한다. 언어 독립적인 절차는 2가지 동작들: 1. 서비스(112) 명칭/설명과 같은, 서비스(112;115, 116) 프로파일들 및 2. 사용자 인터페이스를 포함한다.

1. 서비스 명칭/설명과 같은, 서비스(112) 프로파일들과 관련하여, SSD(116)에서, xml:lang 속성은 상이한 언어들로 서비스 명칭들 및 서비스 설명들을 설명하는데 사용된다. 예를 들어, 다음은 영어로 "open" 및 중국어로 "打开"이라는 서비스 명칭을 설명하는 XML로 기입된 SSD(116) 파일의 예시적 부분이다:

<serviceName xml:lang="en">Open</serviceName>

<serviceName xml:lang="zh">打开</serviceName>

동일한 방법이 서비스를 설명하는데도 적용될 수 있다. 도 16d 및 도 16f도 참고한다.

2. 사용자 인터페이스와 관련하여, 도 12a는, 본 발명의 실시예에 따라, 임의 언어로, STEER-WS TCC(119a)와 같은, WS TCC(119) 사용자 인터페이스를 디스플레이하는 흐름도이다. 일례로서, 여기에서 설명되는 실시예들의 태양에 따르면, 다양한 STEER-WS TCC(119a) 사용자 인터페이스들에서, 표(1200)는 사용되는 모든 컴퓨터 디스플레이 사용자 인터페이스 스트링들을 위해 보유된다. 표(1200)의 스트링 각각을 위해, 상이한 언어들의 다수 버전들이 보유된다. 스트링 표(1200)는 XML로 설명되고, STEER-WS TCC(119a)가 개시될 때, 로딩된다. STEER-WS TCC(119a) 컴퓨터 사용자 인터페이스는, 예를 들어, 사용자의 선택에 기초한 언어를 사용해 스트링들을 디스플레이한다.

도 12a의 동작 1202에서, 사용자에게는 컴퓨터 사용자 인터페이스(예를 들어, 컴퓨터 디스플레이 스크린의 GUI (graphical user interface), 음성 인터페이스 등)를 통해 언어를 선택할 것이 프롬프팅된다. 동작 1204에서, 그 언어가 지원되지 않는다고 판정되면, 디폴트로 영어가 선택된다. 동작 1206에서는, 언어 코드 및 문장 순서가 판정된다(예를 들어, 컴퓨터 판독 가능 매체들로부터 검색되거나, 소프트웨어에 의해 판정되는 식이다). 동작 1208에서는, 선택된 언어의 컴퓨터 사용자 인터페이스를 위해 필요한 스트링들이 표(1200)로부터 검색되는데, 이들은 이 일례에서의 GUI를 위한 스트링들이다. 이러한 일례의 동작 1210에서는, 선택된 언어의 서비스 명칭 및 서비스 설명이 SSD(116)로부터 판정되고, 판정된 서비스 명칭, 서비스 설명들 및 검색된 GUI 스트링들이 선택된 언어의 정확한 문장 순서로 컴퓨터 디스플레이 스크린의 사용자 인터페이스에 디스플레이된다. 따라서, 동작 1210에서는, 한번에, STEER-WS TCC(118a)가 선택된 언어의 서비

스 명칭들, 서비스 설명들, 및 컴퓨터 사용자 인터페이스 스트링들을 디스플레이한다. 동작 1210에서, 서비스(115)의 SSD(116)가 선택된 언어를 지원하지 않으면, 디폴트 언어(예를 들어, 영어)가 인출되고 디스플레이될 것이다. 한편, 문장 순서는 다수 언어들을 위해 고려된다. 여기에서 설명되는 바와 같이, 사용자는, 서비스(115)의 서비스 명칭 및 서비스 설명(116)의 어떤 언어 버전을 STEER-WS TCC(119a)를 사용해 검색할 것인지를 특정할 수 있다. 물론, 태스크 컴퓨팅(100)의 상술된 언어 독립적인 동작들은, (제한없이) SDSCM들(119b)과 같은, 임의의 WS TCC(119a-n)로 제공될 수 있다.

문장 순서와 관련하여, 예를 들어, 영어에서, 문장 순서는 VO(동사 + 목적어)이지만, 일본어에서 문장 순서는 OV(목적어 + 동사)이다. 그러한 언어 순서 정보는 STEER-WS TCC(119a)에도 유지/보유된다(또는, 동작 1202에서, 사용자가 STEER-WS TCC(119a)의 시동시에 그것을 설정하게 될 수도 있다). 동작 1206에서, 구성들을 디스플레이할 때, STEER-WS TCC(119a)는 선택된 언어, 선택된 문장 순서, 또는 양자에 기초해 정확한 문장 순서를 선택할 것이다.

도 12b는, 본 발명의 실시예에 따른, 프레젠테이션 계층에서의 일본어의 컴퓨터 구현형 태스크 인터페이스로서의 컴퓨터 디스플레이형 그래픽 사용자 인터페이스의 이미지이다. 특히, 도 12b는 도 12a의 흐름도에 따라 도 1b의 GUI에 대응되는 일본어로 발생된 STEER-WS-XT TCC(119a-1)를 위한 GUI의 이미지이다. 마찬가지로, 도 9에서는, "language" 버튼(910)의 선택 가능한 그래픽 디스플레이를 선택하는 것에 의해, STEER-WS SIS TCC(119a-3)의 GUI가 선택된 통신 언어의 텍스트로써 디스플레이될 수 있다.

(4) 서비스 관리 펑크션:

SMF들(Service Management Function(s))은 서비스들(112)을 위한 메타-서비스들로서 보여질 수도 있다(도 16g 내지 도 16j 참고). SMF들은 서비스들(112) 때문에 존재하지만, 서비스들(112)이 SMF들 때문에 존재하지는 않는다. 서비스(112)가 진행되면, 서비스(112)의 SMF 또한 진행되어야 한다. 각각의 SMF는 OWL-S로 기입된 고유한 설명을 가진다. 따라서, SMF를 서비스(112)와 링크하기 위해, 서비스의 SSD(116)는 SMF 설명들을 포함하거나 SMF 설명들로의 링크들을 가진다. 후자의 경우, SMF 설명들은 어디에든 상주할 수 있다. 마찬가지로, SMF 구현은 어디에든 디플로이될 수 있고 서비스 자체와 동일한 장치에 머물 필요는 없다.

SMF의 일례들로는 다음을 들 수 있다:

1. Destroy, 일단 호출되고 나면, 서비스가 파괴된다.

2. Handled object of, 일단 호출되고 나면, 서비스에 의해 현재적으로 핸들링되고 있는 의미 오브젝트를 리턴한다.

3. Control Ul of, 일단 호출되고 나면, 서비스 제어 UI로의 링크를 리턴한다.

4. Is Alive, 서비스가 여전히 활동 상태인지의 여부를 테스트하기 위한 펑크션.

5. Location of, 일단 호출되고 나면, 서비스의 현재 위치를 리턴한다. 이것은, 위치가 달라지는 서비스들이 자신의 위치를 제공하는 효율적인 방법이다.

6. 기타, 예를 들어, 서비스가 소정 주기에서 이용 가능한지의 여부에 대한 테스트들 또는 격자 서비스 관련 펑크션들 등.

사용자 인터페이스의 관점에서, SMF는 마치 다른 서비스들(112)처럼 취급된다. 이는 VoiceSTEER-WS TCC(119a-4)에서 특히 유용하다. 예를 들어, 다음의 SMF들은 태스크들(126)로서 수행될 수 있다:

1. Destroy Tablet-PC.ppt

2. Handled Object of View on Projector

3. View Locally Control Ul of Play (Audio)

4. Is Alive Bank?

5. Location of Play (Audio)

또한, 사용자는, 다음과 같은, 다른 서비스들(112)로써 SMF들의 결과를 구성할 수도 있다:

1. "View on Kiosk" "Handled Object of View on Projector"

2. "View Locally" "Control UI of View on Projector"

3. "View on Kiosk" "L-Note of Location of Play (Audio)"

서비스 액세스 제어:

도 13은, 본 발명의 실시예에 따른, 서비스(112) 액세스 제어의 흐름도이다. 편재형 컴퓨팅 환경에는, 모든 사용자들에게 개방되거나 개방되지 말아야 하는 일부 서비스들(112)이 존재한다. 서비스들(112)을 위해 일종의 액세스 제어 메커니즘을 채택하는 것이 중요하다. 여기에서 설명되는 실시예들에 따르면, (1) 공유되는 정책과 위임 및 (2) SSD들(116)을 통한 태스크 컴퓨팅 클라이언트들(119)에 의한 서비스 인증을 포함하는 서비스들(112)로의 액세스 제어가 설명될 것이다.

여기에서 설명되는 실시예들에 따르면, REI 정책 언어는 서비스들(112)로의 액세스 정책들을 확립하는데 사용된다. REI는 정책 특정 언어이다. REI의 개념은 공지되어 있다. 도 13에서, REI 정책 엔진(1300)은, 정책들(액세스/보안 규칙들의 세트), 사실들(사용자 및/또는 클라이언트측에 의해 제공되는 정보), 및 온톨로지들에 기초해 누가 어떤 종류의 액세스 권한들을 갖는지를 판정한다. REI 엔진(1330)에는, 그것이 중앙 집중형이거나 분산형일 수 있도록 하기 위해, 웹 서비스들의 인터페이스(106)가 제공된다. REI 엔진(1300) 및/또는 REI 엔진(1300)으로의 웹 서비스들의 인터페이스(106)는 이 프레임워크를 위해 대체될 수도 있다. 원격 절차 호 인터페이스를 갖춘 임의의 정책 엔진 또는 심지어 서비스(112)내의 정책 엔진을 위한 소프트웨어 모듈로도 충분하다.

도 13에서, 워크플로우는 사용자가 사무실을 방문 중이라고 가정하는 6개 동작들을 포함한다:

1. 동작 1302에서는, 새로운 사용자가 카운터에서 등록할 때, 자격 인증서가 사용자에게 발행된다. 자격 인증서는, 이름, 상태 및 위치와 같은, 사용자에 관한 정보 뿐만 아니라, 그것의 생성 시간, 만료 시간 및 그것의 진정성을 보장하는 디지털 서명과 같은, 신용 증명서에 관한 메타-데이터도 포함한다.

2. 사용자가 사무실을 위한 네트워크를 취하는 동작 1304에서, 사용자의 태스크 컴퓨팅 클라이언트(클라이언트;119), 예를 들어, STEER-WS TCC(119a)는 그 환경(110)에서 그리고/또는 발견 시점에 접속되어 있는 "구"에서 현재적으로 이용가능한 모든 서비스들(112)을 발견한다. 서비스들(112) 중 일부는 공용이고 그들 중 일부는 액세스 제어를 가진다. 서비스들(112)의 정보는 그들의 의미 서비스 설명들(116)에서 설명된다. 사용자의 태스크 컴퓨팅 클라이언트(119)는 그것의 의미 서비스 설명(116)을 살펴보는 것에 의해 서비스(112)의 상태를 점검할 수 있다. 의미 서비스 설명(116)은 태스크 컴퓨팅 클라이언트(119)에, 그것이 액세스 제어를 요하는지의 여부 및 그렇다면 어떤 종류의 자격 인증서(들)가 필요한지를 통지한다.

3. 동작 1306에서는, 사용자가 자신의 태스크 컴퓨팅 클라이언트(119)를 통해 서비스(112)를 호출하고자 할 때, 클라이언트(119)가 서비스(112)의 상태를 점검할 것이다. 서비스가 공용적이라면, 클라이언트는 그것을 평소처럼 호출할 것이다. 서비스가 액세스-제어형이라면, 클라이언트(119)는, 서비스(112)를 실행하기 위한 통상적인 TCE 웹 서비스들(106)의 파라미터들과 함께, 서비스(112)로 추가 파라미터, 즉, 사용자의 자격 인증서(들)를 송신할 것이다. 여기에서 설명되는 실시예들의 다른 태양에 따르면, 클라이언트(119)는, SSL상의 HTTP 등과 같은, 보안 접속을 통해 서비스(112)로 추가 파라미터를 송신할 수 있다.

4. 서비스(112)가 요청을 수신하는 동작 1308에서, 서비스는 먼저, 자격 인증서의 디지털 서명을 점검하는 것에 의해, 자격 인증서의 진정성을 확인할 것이다. 서명이 유효하지 않다면, 요청은 즉시 거부될 것이다. 다음으로는, 자격 인증서의 만료 시점이 점검된다. 시간이 만료된 경우에도, 요청은 거부될 것이다. 자격 인증서가 유효한 것으로 입증되면, 자격 인증서의 사실(들)은 추출되어 REI 엔진(1300)에 삽입될 것이다. 그 다음, 서비스(112)는, 서비스(112)의 정책들에 기초해 사용자가 서비스(112)를 호출하도록 인증되었는지의 여부를 REI 엔진(1300)에 질문할 것이다.

5. 동작 1310에서, REI 엔진(1300)은 온톨로지, 서비스의 정책들 및 사용자에 관한 사실들에 기초해 쿼리에 응답할 것이다.

6. 동작 1312에서는, REI(1300)로부터의 응답에 기초해, 서비스(112)가 요청을 이행하거나 거부할 것이다.

REI 엔진(1300)이 중앙 집중식이거나 서비스(112)와는 상이한 위치에서 실행될 필요는 없다. 예를 들어, 전체적인 통합 캠퍼스를 위해 하나의 단일 REI 엔진을 설비할 수 있거나 (프린터와 같은) 각각의 장치가 자신만의 REI 엔진을 가질 수도 있다. 사실상, 편재형 환경에서, 모든 서비스들이 액세스할 수 있는 REI 엔진을 갖는 것이 흔하지는 않다. 본 발명의 설계에서는, REI 엔진의 인스턴스가 정책들, 사실들, 및 온톨로지들에 관한 충분한 정보를 갖기만 한다면, 응답은 주어질 것이다.

(1) 공유되는 정책과 위임:

상기 논의는 주로 (자격 인증서들을 발행하는 인증 기관에 의해 발행된 디지털 서명을 통해 인증될 수 있는) 클라이언트들에 의해 제공되는 사실들, 그것의 정책, 및 온톨로지들을 통해 클라이언트(119)의 액세스 권한들을 판정하는 서비스들에 관한 것이다.

가끔은 고유한 개별 정책 뿐만 아니라 소정 커뮤니티들에 의해 공유되는 정책들도 사용하는 서비스를 원한다. 이것은, 서비스 자체에 액세스하지 않으면서, 권한들의 일 사용자로부터 다른 사용자로의 위임을 실현하고자 할 경우에 특히 유용하다. (유비쿼터스 환경들에서, 서비스는 대개 한정된 컴퓨팅 리소스들을 가진 장치에 의해 호스팅된다. 이런 장치들이 그러한 정책들로의 안전한 실제 액세스를 지원하고 그러한 정책들을 관리하는 것은 상당한 부담일 수 있다.)

공유되는 정책들을 위한 다수 사이트들은 조직적 계층 구조, 지리적 구조 등에 대응될 수도 있다. 서비스가 빌딩 Y의 호스팅되는 부서 X에 속한다면, 서비스는 공유되는 정책 사이트들을 X 및 Y를 위해 사용하고자 할 수도 있다.

처음에 한번, 서비스 담당자는 액세스 제어 계산을 위해 사용될 공유되는 정책들을 위해 점검될 하나 이상의 사이트들을 서비스를 위해 설정한다. 공유되는 정책 사이트들로의 액세스들은 보안될 수도 있다(예를 들어, SSL상의 HTTP). 서비스가 액세스 제어를 계산할 필요가 있을 때, 서비스는 가능한 업데이트들을 위해 특정된 사이트들을 점검할 것이다. 사이트들 중 어떤 것을 위해서도 업데이트가 존재하지 않는다면, 서비스는, 클라이언트에 의해 제공되는 사실들, 그것의 정책, 온톨로지들 및 다른 정보와 함께 캐싱된 정책들에 기초해 액세스 제어를 계산하도록 진행한다. 사이트들 중 하나를 위해 어떤 업데이트가 존재한다면, 업데이트된 정책이 다운로드되고, 캐시가 업데이트되며, 공유되는 최신 정책들로써 계산이 수행될 것이다.

권한들의 위임과 관련하여, 그것은 공유되는 정책 사이트들을 통해 수행될 수 있다. 사용자가 공유되는 정책 사이트로의 안전할 수도 있는 접속을 통해 소정 사람(또는 그룹 등)에게 권한을 위임한다는 스테이트먼트로써 공유되는 정책을 업데이트하는 것에 의해, (사용자가 가진, 예를 들어, 소정 프린터에서 인쇄하기 위해서는 위임해야 할) 권한을 위임할 수 있다. 서비스가 액세스 제어를 계산하는 다음 시점에서, 서비스는 업데이트된 공유 정책을 사용하고 위임된 권한을 가진 사람이 서비스를 사용하게 된다.

호출을 위해, 원래의 사용자는 공유 정책을 업데이트하여, 사용자가 그 사람에게로 권한을 호출할 것을 언급하는 호출 스테이먼트를 추가한다. 또는, 원래의 사용자는 공유 정책으로부터 원래의 위임 스테이트먼트를 제거할 수 있다.

(2) SSD(116)를 통한 태스크 컴퓨팅 클라이언트들(119)에 의한 서비스 인증:

서비스가 클라이언트를 항상 인증하고자 하지는 않는다. 때때로, 클라이언트는 서비스를 인증하고자 하거나 클라이언트가 서비스를 실행하기 위한 권한을 갖는지를 미리 판정하고자 한다. 그것이 미리 판정될 수 있다면, 클라이언트는 사용자에게 그것이 액세스 불가능하다고 경고하거나 액세스 불가능한 서비스들을 사용자로부터 은닉하기로 결정할 수도 있다.

태스크 컴퓨팅에서, 서비스는 클라이언트에 의해 그것의 SSD(Semantic Service Description)를 통해 식별된다. SSD는 서비스가 무엇인지, 그것의 내부 프로세스들이 무엇인지, 그것이 어떻게 실행될 수 있는지 등을 언급한다. 따라서, SSD 자체로 또는 다른 메커니즘들을 통해 개별적으로 디지털 서명을 제시하는 것에 의해, 서비스는 클라이언트에 의해 인증될 수 있다. 디지털 서명은, 클라이언트가 신뢰하는 기관들 중 하나에 의해 서명될 필요가 있다. 디지털 서명은 SSD의 부분들 또는 전체 SSD를 위한 것일 수 있다. 디지털 서명은 SSD의 중요한 부분들을 위해서만 부분적으로 디지털 서명될 수도 있다.

클라이언트가 서비스를 실행할 권한을 갖는지를 판정하기 위해, SSD는 서비스의 정책 정보를 위한 벡터로서 사용될 수 있다. SSD는 SSD에 정책 자체를 또는 그것이 사용하는 정책들로의 포인터들(예를 들어, URL들)을 포함할 수 있다. 정책들

은 공유되는 정책과 위임 섹션에서 상술된 공유 정책을 포함할 수도 있다. 클라이언트가 SSD의 정책 정보를 획득할 때, 클라이언트는, 클라이언트들에 관한 사실들, 온톨로지들, 및 다른 정보와 함께 SSD의 정보로써 클라이언트가 서비스를 실행할 권한을 갖는지를 판정할 수 있다.

서비스는 그것의 정책들 모두를 SSD로 노출시킬 필요는 없지만, 부분적인 정보라 하더라도 사용자가 서비스를 의미없이 실행하는 기회들을 감소시킬 수 있으므로, 그것은 여전히 의미가 있다.

액세스 제어를 위한 태스크 컴퓨팅 클라이언트(119)의 메모리 장치 디플로이먼트:

앞서, 사용자들은 태스크 컴퓨팅 클라이언트를 사용하기 전에 소프트웨어를 설치해야 한다. 그것은 시간 소모적이고 때로는 태스크 컴퓨팅의 사용자 채택을 위한 장애가 된다. 이 문제점을 위한 해결책은, 사용자가 어떤 설치없이도 태스크 컴퓨팅을 사용해 시작할 수 있도록 하기 위해, 태스크 컴퓨팅 클라이언트(119) 뿐만 아니라— Java 런타임과 같은, 실행 환경을 포함하는 (CD 또는 UBS 플래시 메모리와 같은) 휴대용 또는 분리형 매체들 또는 장치를 생성하는 것이다.

휴대용 또는 모바일 TCC(119)는, 사용자 편의를 위한 액세스-제어형 서비스들을 위해 아주 중요한 자격 인증서 발행과도 조합될 수 있다. 사용자가 카운터에서 등록할 때, 자격 인증서가 발생되어 휴대용 매체들 또는 장치에 추가될 수 있다. 그 다음, 사용자는, 사용자에게 할당된 권한들에 기초해, 사용자의 고유 머신상의 매체들 또는 장치를 사용해 서비스들을 액세스할 수 있다. 자격 인증서는, 추가적인 변경들이 수행될 수 없도록, 판독 전용으로 설정될 수도 있다. 매체들 또는 장치상의 태스크 컴퓨팅 클라이언트가 매체들 또는 장치의 고정된 경로로부터 자격 인증서를 판독하게 되어 있다면, 사용자가 자격 인증서를 오용하는 것이 좀더 어려워진다. 메모리 장치가 유일한 선택이 아니며, CD, DVD와 같은, 다른 매체들도 사용될 수 있다는 것에 주목한다.

도 14a 내지 도 14g는, 본 발명의 일 실시예에 따른, 한 장소에서의 서비스 액세스 제어 사용을 도시하는 시나리오의 도면들이다. 특히, 도 14a 내지 도 14g는 예시적 장소로서 사무실에서의 서비스 액세스 제어의 도면들이다.

1. 도 14a에서, University-1의 대학원생인 Bob은, Company-1 또는 site-1의 인턴으로서, Company-1을 방문한다.

2. 도 14a에서, Company-1의 사무실 관리자인 Wendy가 Bob을 맞이한다.

3. 도 14a에서, Wendy는 Bob을 위한 자격 인증서로써 STEER-TCC-Stick(1400)을 생성한다. STEER-TCC-Stick (1400)은, 예를 들어, STEER TCC(119), Java 런타임을 포함하는 태스크 컴퓨팅 클라이언트(119)를 실행시키는데 필요한 모든 것들을 갖춘 USB 메모리 장치일 수 있다.

4. 도 14a에서, 소프트웨어를 사용해, 자격 인증서 생성자인 Wendy는 자격 인증서를 생성하여 STEER-TCC-Stick의 자격 인증서 폴더에 저장한다. 자격 인증서는 그의 이름, 입사 일자, 상태("인턴") 및 자격 인증서의 메타데이터(그것의 생성 날짜, 만료 날짜/시점, 위임 정보 등)와 Company-1의 전용 키로써 서명된 디지털 서명을 포함한다. 도 14b는, 본 발명의 실시예에 따른, 서비스 액세스 제어의 아키텍처 도면이다.

5. 도 14a에서, Bob은 그의 랩탑(1402)에서 STEER-TCC-Stick(1400)으로부터의 STEER TCC(119)를 실행한다. 도 14b는, 본 발명의 실시예에 따른, 일반적인 서비스 액세스 제어 시스템/플로우 아키텍처이다. 도 14b에서, TCC(119)는 서비스(112)를 발견한다(즉, 서비스(112)의 OWL-S로 SSD(116)를 발견한다). SSD는 서비스(112)의 (부분적인) 정책 및 그것의 속성들에 대한 값들을 포함하는 사실들을, 서명된 자격 인증서에서 설명한다. TCC(119)는, 자격 인증서에 기초해 서비스(112)를 신뢰할 수 있는지를 판정할 수 있고, 사용자와의 상호 작용을 통해, 그 서비스(112)가 사용자가 잠재적으로 사용하고자 하는 것인지를 판정할 수도 있다. 예를 들어, 서비스의 자격 인증서는, 사용자가 Company-1 사무실로의 체크-인 시점에서 신뢰하기로 결정한 Company-1에 의해 서명되고, 사용자는 서비스(112)를 신뢰할 수 있는지의 여부를 결정할 수도 있다. 다음으로는, 사용 가능성을 위해, 사용자와 TCC가 서비스의 정책을 충족시키는지를 점검할 수 있다. 사용자가 서비스를 호출하기로 결정하고 TCC(119)에 서비스를 호출할 것을 지시한다면, TCC는 웹 서비스 호출의 다른 파라미터들과 함께 속성 값들을 포함하는 사실들을 송신할 것이다. 서비스(112)는, 송신된 사실들이 유효한지를 판정하기 위해, 자격 인증서의 디지털 서명, 만료 시점, 및 다른 것들을 점검한다. 다음으로는, 사실들, 온톨로지들, 및 공유 정책, 예를 들어, Company-1 정책 사이트(1404)에서의 공유 정책을 사용해, 서비스(112)는, 사용자가 서비스(112)를 호출하고 그에 따라 응답할 권한을 갖는지를 판정한다.

6. 도 14c 및 도 14d에서, Bob은 키 아이콘을 갖춘 "Secure Print" 서비스(112)를 발견한다. "Secure Print" OWL-S 파일에서, 그것은, 그것이 Company-1 자격 인증서를 요한다고 서술한다. (그것은, 그것이 다수의 자격 인증서들 중 하나를 요한다고 서술할 수도 있다.) STEER TCC(119)가 요구 사항 스테이트먼트를 찾아내면, 그것은 서비스(112)를 위한 키 아이콘(이 경우에는, "Secure Print")을 나타낸다.

7. 도 14c 및 도 14d에서, Bob은 "Secure Print"의 사용을 시도하지만, "Intern"에게는 그 서비스(112)를 사용하는 것이 허용되지 않으므로, 실패한다. "Secure Print" OWL-S 파일에 기초해, STEER TCC(119)는 Company-1의 자격 인증서를 그것의 "credential" 폴더에서 찾는다. 그것을 발견했을 때, STEER TCC(119)는 웹 서비스 호(106)의 서비스 호출 파라미터들과 함께 자격 인증서를 송신한다. "Secure Print"는, 그것이 유효하다는 것을 보장하기 위해 자격 인증서의 디지털 서명을 점검한다. (그에 따라, 자격 인증서의 사실들은 변경되지 않는다.) 먼저, 서비스는, 자격 인증서가 만료되지 않았는지를 확인한다. 그렇지 않다면, 서비스는 자격 인증서의 이러한 사실들을 사용해, 호출자가 웹 서비스 API를 통해 호출되는 REI 정책 엔진에 의해 서비스를 사용할 권한을 갖는지를 판정한다. 정책 엔진으로부터의 결과가 오케이라면, "Secure Print"는 파일을 인쇄한다. 그렇지 않다면, "Secure Print"는, 요청이 거절되었다고 서술하는 메시지를 역송신한다. (이 경우, 인턴으로서의 Bob은 인쇄할 권한을 갖지 않으므로, 그는 거절된다.)

8. Bob은 Company-1의 선배 고용자인 John에게 인쇄를 위한 권한을 위임할 것을 요청한다.

9. John은 소프트웨어, 위임 관리자(1406)를 사용해 John에 의한 Bob으로의 권한 위임을 Company-1 정책 사이트에 대해 안전하게 주장한다. Company-1의 정책 사이트에는 선배 고용자가 인턴들에게 권한을 위임할 권한을 가진다는 스테이트먼트가 존재한다.

10. Bob은 "Secure Print" 사용을 다시 시도하고 이번에는 성공한다.

11. 그후, John은 위임 관리자(1406)를 사용해 위임을 철회한다(앞서 생성된 위임 주장은 Company-1의 정책 사이트로부터 제거된다).

도 14e는 도 14a 내지 도 14e의 시나리오에 대한 그래픽 흐름도이다. 도 14f는, 본 발명의 실시예에 따른, 한 장소에서의 서비스 액세스 제어 사용을 도시하는 도 14a 내지 도 14e의 시나리오에 대한 흐름도이다. 도 14a 내지 도 14h의 번호들은 상기 시나리오 항목들(1-11)에 대응된다.

따라서, 액세스 제어는 다음의 요소들: (1) (디지털 서명에 의해 인증된) 태스크 컴퓨팅 클라이언트(119)에 의해 제공되는 사실들; (2) 서비스(112) 사용 정책(private policy); (3) 공유 정책; 및 (4) 온톨로지들에 기초해 판정된다. 서비스(112)는 그것의 구성에 따라 다수의 공유 정책들을 사용할 수 있다. 매번, 열거된 이들 서비스 액세스 제어 요소들은 혼합되어 액세스 제어를 판정한다. 도 14g는, 본 발명의 실시예에 따른, 서비스 액세스 제어 소자들의 행렬(1410)이다. 좀더 구체적으로, 도 14g는 도 14a 내지 도 14f를 참조하여 상술된 서비스(112) 액세스 제어이다. 도 14g의 동작 1420에서, 클라이언트(119)는 클라이언트 정책 및 서비스 공용 속성들($C\text{-}P$, $S\text{-}A_{pub}$)에 기초해 서비스(구성)의 수용 가능성을 계산한다. 서비스 속성들은, (제한없이) 서비스를 사용하기 위한 비용, 임의의 자격 인증서 정보, 동작 정보 등과 같은, 서비스들에 관한 사실들이다. 동작 1420이 서비스의 전용 속성들을 사용하지는 않지만, 그들이 제공되지 않았다면, 동작 1420은, 서비스가 클라이언트에 수용될 수 있는 가능성을 증가시킨다. 동작 1422에서, 클라이언트는 클라이언트 속성들 및 서비스 공용 정책들($C\text{-}A$, $S\text{-}P_{pub}$)에 기초해 서비스(구성)를 사용 중인 클라이언트의 서비스에 대한 실현 가능성을 계산한다. 다시, 동작 1422는, 클라이언트가 서비스의 전용 정책으로의 액세스를 갖지는 않는다 하더라도, 서비스가 실현 가능할 수 있는 가능성을 증가시킨다. 동작 1424에서, 서비스(112)는 클라이언트 속성의 모든 서비스 액세스 가능 팩터들과 서비스의 공용 및 전용 정책($C\text{-}A$, $S\text{-}P_{pub}$, $S\text{-}P_{pri}$ 및/또는 (경우에 따라) 서비스 $S\text{-}A_{pri}$)에 기초해 클라이언트의 수용 가능성을 계산한다(또는 클라이언트를 인증한다). 도 14h는 사실들, Secure Print 서비스(112)를 위한 전용 정책, 및 상기 시나리오에서 사용되는 Company-1을 위한 공유 정책의 예시적 나열(1412)이다. 좀더 구체적으로, 태스크 컴퓨팅 (100) 시스템에서, 서비스 액세스 제어 핸들링(424;도 4)은 서비스(112)에 대한 액세스를 여기에서 설명되는 바와 같이 핸들링할 수 있다.

다음에서는, 태스크 컴퓨팅(100)에서 사용될 의미 오브젝트들을 제공하는 SDSCM들(119b)로서의 4개의 다른 의미화기 클라이언트 애플리케이션들, 즉, (1) 실사 오브젝트 의미화기 클라이언트(119b-3), (2) 데이터베이스 의미화기 클라이언트(119b-4), (3) 미디어 게시자(119b-5), 및 (4) "화이트 홀"(119b-1)이 설명될 것이다.

(1) 실사 오브젝트 의미화기 클라이언트(119b-3):

도 15는, 본 발명의 실시예에 따른, 실사 오브젝트 의미화기 클라이언트(119b-3)의 아키텍처 펑크션 블록도이다. 실사 오브젝트 의미화기 클라이언트(119b-3)는, 책과 같은, 실사 오브젝트들에 의미 오브젝트들을 제공한다. 실사 오브젝트 의미화기 클라이언트(119b-3)는 하나 이상의 실사 오브젝트들로부터 하나 이상의 의미 오브젝트들을 또는 하나 이상의 실사 오브젝트들로부터 다수의 의미 오브젝트들을 발생시킬 수도 있다. 화이트 홀(119b-1) 및 PIPE-WS API(122)와 관련하여 상술된 바와 같이, 일단 의미 오브젝트가 발생/생성되고 나면, 관리 도구(124)로의 PIPE-WS API(122)가 사용되어 발생된 의미 오브젝트를 위한 SSD를 발생시킬 수 있고, 그에 의해, 의미 오브젝트가 발견 및 구성을 위한 서비스(112)가 될 수 있게 한다. 실사 오브젝트 의미화기를 위한 잠재적 사용들의 기술들은 (제한없이) 수동 및/또는 능동의 RFID(Radio Frequency Identification) 태그들, 바 코드, QR 코드 <http://www.qrcode.com>, 일본어에서 주로 사용되는 2차원 코드, 한정되거나 한정되지 않은 어휘 및 문법을 갖춘 음성 인식, 시각 및/또는 동작 인식들, 또는 그들의 임의 조합들을 포함한다.

도 15에서, 실사 오브젝트 의미화기 클라이언트(119b-3)는 1. 인식 프로세싱 엔진(1502), 2. 의미화기 프로세스(1504), 및 3. 게시자(1506)의 프로그램된 프로세스들을 포함한다.

1. 인식 엔진:

인식 엔진(1502)은 태그들, 코드들, 음성, 비디오, 동작 등을 인식한다. 인식 프로세스는 (항상 온 상태이고 자력으로 오브젝트를 인식하는) 능동이거나 (사용자들 또는 프로그램들에 의해 트리거되는) 수동일 수 있다. 태그들 및 코드들의 경우, 적절한 판독기들이 인식 엔진(1502)으로서 사용되고; 음성/시각/동작의 경우, 대응되는 멀티미디어 입력의 인식 엔진들(1502)이 사용된다. 일부 인식 엔진들은 대부분 오류-경향이 있다. 그러나, 데이터 패턴들에 목적 특징적인 소정 제한 사항들을 부여하는 것에 의해, 인식 속도를 향상시킬 수 있다. 예를 들어, 음성 인식의 경우, 사용되는 어휘 및 문법을 제한할 수 있다. 인식 속도가 낮다면, 인식 엔진에 의한 추가적인 확인 프로세스들을 사용하는 것이 시스템의 전반적인 인식 속도를 향상시키는데 도움이 된다. 예를 들어, 사용자는 실사 오브젝트 의미화기 클라이언트(119b-3)에 기초해 음성 인식에 다음의 태스크(126)를 요구할 수도 있다.

(User) Computer, give me the book with the ISBN: 0-7356-1918-2

(Computer) Sure.

다른 태스크(126) 대화에서는, 다음과 같을 수 있다.

(User) Computer, give me the book with the title, "INTRODUCING MICROSOFT .NET THIRD EDITION"

(Computer) Let me confirm. Is it the book with the title, "INTRODUCING MICROSOFT .NET THIRD EDITION"?

(User) Yes.

"book"(을 다른 의미 오브젝트 명칭들로), "ISBN"(을 의미 오브젝트의 특성 명칭들로), 및 값(특히, ISBN 또는 숫자와 같은 값들이 잘 한정될 수 있음)을 변경하는 것에 의해, 다른 경우들에서는 상기 문장들의 변형들이 사용될 수도 있다.

간혹, (부분적/전체적) 의미 오브젝트들 자체가 태그들 및 코드들(또는 음성 명령으로) 인코딩될 수도 있다. 특히, 대용량 메모리를 갖춘 RFID 태그들 및 QR 태그들은 의미 오브젝트들을 평문 또는 인코딩된 포맷으로 보유할 수 있다. 다른 방법으로, 다운로드될 수 있으며 그것의 SSD(116)가 생성되어 게시될 수 있는 의미 인스턴스로의 포인터를 갖춘 RFID 태그를 가질 수도 있다.

인식 엔진(1502)이 오브젝트 또는 오브젝트들을 인식하자마자, 인식 엔진(1502)은 그 정보를 의미화기(1504)로 전달할 것이다.

2. 의미화기(1504):

인식 엔진(1502)에 의해 전달된 정보로부터, 의미화기 프로세스(1504)는 먼저 대응되는 오브젝트에 관한 정보를 발견하려 한다. 예를 들어, RFID의 경우, 의미화기 프로세스(1504)는 RFID의 t를 갖춘 오브젝트들을 위해 로컬 또는 원격 데이터베이스(1508)를 조회할 수도 있다. 다음으로, 의미화기 프로세스(1504)는 의미 오브젝트를 발생시킨다. 예를 들어,

"Book" 의미 오브젝트는 ISBN 번호에 기초해 로컬 또는 원격 데이터베이스(1508)를 조회하는 것에 의해 인식 엔진(1502)으로부터 획득된다. 도 15에서, 원형 점선은 "수동" 모드 경우를 표현하는데, 여기에서, 실사 오브젝트의 인식은 사용자에 의해 트리거된다. 다른 방법으로, 인식 엔진(1502) 및 의미화기 프로세스(1504)는 물리 오브젝트의 의미 오브젝트(들)를 획득하기 위한 외부 모듈(1510)용 API를 통해 외부 모듈(510)에 의해 호출될 수도 있다.

의미 오브젝트들은 로컬 파일 시스템에 개별 파일들로서 저장될 수도 있고 시스템은 단순히, 정보와 매칭되는 파일을 통해 의미 오브젝트를 획득할 수도 있다. 예를 들어, 의미화기는 단순히 RFID 데이터와 동일한 명칭을 가진 파일을 선택하고 파일의 의미 오브젝트를 리턴한다.

전체의 의미 오브젝트들이 의미화기로 전달될 경우, 그것은 아무 소용이 없지만, 부분적인 의미 오브젝트들의 경우라면, 의미화기는 오브젝트에 관한 추가 정보를 첨부할 수도 그렇지 않을 수도 있다.

의미화기가 하나의 또는 고정된 수의 의미 오브젝트들을 리턴할 것으로 예상되고 그들을 판정할 수 없는 경우라면, 의미화기는 사용자에게 가능한 의미 오브젝트들로부터 적절한 것들을 선택할 것을 또는 인식 프로세스가 다시 발생하도록 조정할 것을 요청할 수도 있다.

종료시에, 의미화기는 그러한 의미 오브젝트들을 게시자에게로 전달하거나 그들을 프로그램에 따른 API 모듈로 리턴할 것이다.

3. 게시자(1506):

게시자는 그러한 의미 오브젝트들을 의미 오브젝트 제공 서비스들로서 제공한다. 의미화기에 의해 하나의 의미 오브젝트가 제공된다면, 게시자는 하나의 의미 오브젝트 제공 서비스를 게시할 것이다. 또는, 다수 오브젝트들이 주어질 경우, 게시자는 다수의 의미 오브젝트 제공 서비스들을 게시할 것이다. 또는, 일부 경우들에서는, 사용자로 하여금 그것의 사용자 인터페이스로부터 다수 오브젝트들 중 하나 이상의 의미 오브젝트들을 선택하게 하는 단일 서비스를 게시할 것이다. 또는, 이러한 방법들의 혼합이 사용될 것이다. 게시 메커니즘들로서, PIPE WS API(122)가 사용될 수도 있다.

여기에서 설명되는 실시예들의 태양에 따르면, 인식은, 예를 들어, 버튼을 클릭하는 사용자에 의해 트리거될 수 있다. 또는, 프로그램에 따른 API 모듈로부터의 펑크션 호에 의해 개시될 수도 있다. 펑크션 호가 인식된 의미 오브젝트들의 리턴 값을 요한다면, 의미화기는 의미 오브젝트들을 프로그램에 따른 API 모듈로 리턴할 것이다. 이 경우, 의미화기는 의미 오브젝트들을 게시자에게로 송신하지 않을 수도 있다. 프로그램에 따른 API 모듈로부터의 펑크션 호들은, 웹 서비스들의 호들을 사용하는 것과 같이, 원격적으로 구현될 수도 있다.

(2) 데이터베이스 의미화기 클라이언트(119b-4):

대부분의 포매팅된 데이터는 오늘날 관계 데이터베이스들에 저장된다. 데이터베이스 의미화기는 데이터베이스들로부터의 데이터가, RDF의 또는 좀더 구체적으로는 OWL의 의미 오브젝트들과 같은, 의미 오브젝트들로서 이용될 수 있게 한다. 좀더 구체적으로, 데이터베이스 의미화기(119b-4)는 부분적인 구조적 텍스트 데이터를 프로세싱한다. 통상적으로 본 발명에 따르면, 데이터베이스 의미화기는 2개의 주요 모듈들: 1. 데이터베이스 스키마와 온톨로지 사이의 매핑을 생성하기 위한 사용자 인터페이스; 및 2. 앞서 주어진 매핑에 기초해 데이터베이스로부터 의미 오브젝트들을 제공하기 위한 의미 서비스 프로세스를 포함한다.

선택적으로, 데이터베이스 의미화기는, 프로세스 1에서 생성된 매핑에 기초해, 하나 또는 다수 파일들로부터의 데이터 전부 또는 일부로부터 의미 오브젝트들을 생성할 수 있다.

매핑을 생성하기 위한 사용자 인터페이스는 그래픽일 수 있다. 좀더 구체적으로, 그것은 GUI 윈도의 한쪽에는 데이터베이스 스키마를 그리고, 사용자가 데이터베이스를 매핑하고자 하는 다른 GUI 윈도의 다른 쪽에는 온톨로지를 나타낼 수 있다. 사용자는 데이터베이스 스키마와 온톨로지 사이의 매핑을 수동으로 특정할 수도 있다. 통상적으로, 여기에서 설명되는 실시예들에 따르면, 사용자는 한쪽의 온톨로지에서 의미 오브젝트를 그리고 다른 쪽의 데이터베이스 스키마에서 항목을 선택하고 (예를 들어, "Map" 버튼을 클릭하는 것에 의해) 이들이 매핑될 시스템을 특정한다. 사용자는, 사용자가 소정 매핑들 모두를 특정할 때까지, 프로세스를 반복한다. 앞서 생성된 매핑 스펙을 사용해, 의미 서비스 프로세스는 데이터베이스로부터의 데이터(실제 값들)를 매핑하고 그에 따라 매핑된 값들로써 의미 인스턴스들을 생성한다. 그러나, 시스템은, 스키마 및 온톨로지의 구문론 단서들에 기초해, 가능한 매핑에 대한 제안들도 제공할 수 있다. 시스템은 매핑 일관성에 대한 즉석 점

검을 제공할 수도 있다. 매핑이 수행될 때, 시스템은 매핑을, 예를 들어, 장차의 사용을 위한 파일로서 저장할 것이다. 데이터베이스 의미화기 클라이언트(119b-4)는, 여기에서 설명되는 바와 같이, PIPE-WS API(122)에 기초해 SSD(116)를 생성하는 것에 의해, 생성된 의미 오브젝트를 사용해 서비스(112)를 생성할 수도 있다.

의미 서비스 프로세스에는 매핑에 기초해 의미 오브젝트를 발생시키기 위한 프로그램에 따른 API들 뿐만 아니라 발생된 하나 이상의 의미 오브젝트들로부터 사용자가 선택하기 위한 사용자 인터페이스도 수반된다. 의미 서비스가 실행될 때, 의미 서비스는 사용자가 하나 이상의 의미 오브젝트들을 선택하기 위한 사용자 인터페이스를 제공하고, 그 다음, 의미 서비스는 선택된 의미 오브젝트들을 그것의 리턴 값으로서 리턴한다. 효율성을 위해, 특히 데이터베이스가 대다수 데이터를 보유할 경우, 의미 서비스는 매번 데이터베이스에 접속해 사용자 인터페이스를 제공하고 매핑에 기초해 의미 오브젝트들에 데이터를 매핑한다. 그러나, 데이터베이스 의미화기가 주어진 매핑에 기초해 데이터베이스로부터 의미 오브젝트들이 생성되게 하고 그러한 의미 오브젝트들을 통해 그것의 펑크션들을 제공할 수도 있다.

(3) 미디어 게시자 클라이언트(119b-5)

디렉토리 게시 서비스처럼, 미디어 게시 서비스는 사용자로 하여금 장치로부터 파일(오디오, 비디오, 또는 이미지)을 선택할 수 있게 하고 대응되는 의미 인스턴스를 취할 수 있게 한다. 그러나, 서비스가 개시되는 방법은 상이하다. 사용자가 (메모리 장치, 디지털 카메라, CD-ROM, DVD-ROM, 또는 외부 하드 드라이브와 같은) 장치를 컴퓨팅 장치에 플러그-인할 때, 프로그램이 개시되어 장치에 파일들(오디오 파일들, 비디오 파일들, 이미지 파일들 등)이 존재하는지의 여부를 점검한다. 그렇다면, 사용자가 그들을 게시하기를 원하는지의 여부를 질문하는 다이얼로그 박스가 팝업될 것이다. 사용자가 그렇게 하기로 결정하면, 새로운 서비스(들)가 발생된다.

본 서비스는, 사용자가 사용자의 파일들(오디오, 비디오, 픽처들 등)을 편안하게 공유하기를 원할 때 특히 유용하다. 사용자는 단순히 장치를 연결하고 OK를 클릭하기만 하면 된다. 다음에는, 모든 것이 사용자를 위해 설정된다. 사용자는 다른 서비스들과 조합된 새로운 게시 서비스들을 사용해 사용자의 태스크들을 실현할 수 있다. 사용자가가 원한다면, 일 옵션 세트로써, 심지어 OK 클릭조차 생략될 수 있어 전체적인 서비스 게시 프로세스가 완전 자동화될 수도 있다. 미디어 게시자 클라이언트(119b-5)는, 여기에서 설명되는 바와 같이, PIPE-WS API(122)에 기초해 SSD(116)를 생성하는 것에 의해, 생성된 의미 오브젝트를 사용해 서비스(112)를 생성할 수도 있다.

(4) "화이트 홀"(119b-1)

도 16a는, 본 발명의 실시예에 따른, 오브젝트들 및 서비스들의 의미화 단계, 서비스화 단계, 및 게시 단계의 절차이다. 도 16a를 참조하면, 화이트 홀(119b-1)은 PIPE-WS API(122)(관리 도구들;124)를 사용해 서비스들의 동적 생성(서비스 게시) 및 그들의 배포(공유)를 지원한다. 이 도구는 오브젝트들 및 서비스들(정보)을 의미화, 서비스화, 및 게시하는데 사용된다. 화이트 홀 클라이언트(119b-1)는 (OS로부터의 파일들, PIM 애플리케이션의 연락처들 등과 같은) 오퍼레이팅 시스템 또는 애플리케이션 오브젝트들, OWL 포맷의 의미 오브젝트들(또는 OWL 파일로의 URL), 및 OWL-S 포맷의 의미 서비스 설명들(또는 OWL-S 파일로의 URL)을 위한 편리한 드래그-앤-드롭 인터페이스를 가진다.

무엇인가가 화이트 홀로 드롭(입력)될 때, 도구는 먼저 그것의 유형을 다음과 같이 결정한다: (a) 그것이 OWL 또는 OWL-S 오브젝트라면, 화이트 홀은 단지 그것을 PIPE-WS API(122)로 전달하고(다음에서 논의됨)(도 16d 내지 도 16f); (b) 그것이 OWL 또는 OWL-S 파일로의 URL이라면, 화이트 홀은 URL의 내용을 다운로드하여 그것을 PIPE-WS API(122)로 전달하고; (c) 그것이 (의미론적으로 말해) 공지의 OS/애플리케이션 오브젝트(도 16g 내지 도 16j) 또는 의미 오브젝트(도 16k 내지 도 16n)라면, 화이트 홀은 오브젝트를 의미화한다(도 16a, 도 16b, 표 1550 참고). 의미화는 (예를 들어, 시맨티 오브젝트들을 생성하기 위한 실사 오브젝트 의미화기(119b-3) 및 데이터베이스 의미화기(119b-4)의 비한정적인 일례들에서 설명된 바와 같이) OS/애플리케이션 오브젝트로부터 의미 오브젝트를 생성하는 프로세스이다. TCS(118)는 (OS로부터의 파일과 URL, PIM 애플리케이션으로부터의 연락처와 스케줄 등과 같은) OS/애플리케이션 오브젝트들의 10가지 유형들을 지원할 수 있다. 화이트 홀은 오브젝트들의 명칭, 확장자, 및 내용에 의해 오브젝트들의 의미 유형을 판정한다. 일단 유형이 판정되고 나면, 유형을 위한 OWL 템플릿이 검색되고 원래의 오브젝트로부터 추출된 값들로써 채워진다. 다음으로는, 오브젝트의 OWL 설명이 발생되어 PIPE-WS API(122)로 전달된다. 예를 들어, 사용자가 PIM 애플리케이션으로부터 연락처 항목을 드롭하면, 화이트 홀은 먼저 연락처 유형을 위한 OWL 템플릿을 로드한 다음, 연락처 항목으로부터 이름, 회사, 이메일, 전화번호 등을 검색하고, 그것들을 템플릿에 채운다. 마지막으로, 완전한 OWL 오브젝트가 PIPE-WS API(122)로 전달된다.

미들웨어 서버 프로세싱 계층(108)의 관리 도구들(124)의 일부인 PIPE-WS API(122)는 오브젝트들을 서비스화하고 그들을 게시하기 위한 도구이고(도 16c 참고); 화이트 홀의 가능한 출력들(OWL의 의미 오브젝트 또는 OWL-S의 의미 서비

스 설명)은 게시 단계 이전에 서비스화되어야 한다. 따라서, 호출될 경우, 의미 오브젝트 자체를 리턴할 (의미 설명과 연관된) 서비스(112)가 생성된다. 구체적으로, PIPE-WS API(122)는 먼저, 호출될 때, 의미 오브젝트를 그것의 출력으로서 리턴하는 웹 서비스를 동적으로 생성하고; 다음으로는, 새롭게 생성된 서비스를 위한 의미 서비스 설명이 발생된다(도 16, 표 1555 참고). 이 프로세스 동안, 서비스의 명칭, 설명, 출력 유형, 및 그라운딩 세부 사항들이 판정되고 (OWL-S의) 하이 레벨에서 설명된다. 따라서, 태스크 컴퓨팅(100) 시스템은 태스크 컴퓨팅(100) 시스템을 통해 정의된 오브젝트들 및/또는 임의의 OWL 오브젝트도 지원한다.

서비스화의 성과는, 사용자가 화이트 홀로 드롭한 원래의 것이거나 새롭게 생성된 웹 서비스를 설명하기 위해 생성된 하나의 PIPE-WS API(122)인 SSD(Semantic Service Description;116)이다. PIPE-WS API(122)는, (상술된 바와 같이) 사용자가 선택하는 발견 범위에 따라 SSD(116)를 게시하는데 사용될 수도 있다. 예를 들어, 사용자가 그것을 서브넷 서비스에 의한 그룹으로서 게시하고자 한다면, PIPE-WS API(122)는 OWL-S 파일을 포인팅하는 getDescriptionURL 액션을 갖춘 UPnP 장치를 생성할 것이다.

비록 PIPE-WS API(122)가 화이트 홀 클라이언트(119b-1)와 관련하여 설명되기는 하지만, PIPE-WS API(122)는 웹 서비스들의 인터페이스를 갖춘 완전히 독립적인 도구이므로, 그것은 TCS(118)의 다른 임의 컴포넌트들에 의해 호출될 수 있고, 오브젝트들 또는 서비스들을 게시하는데 사용될 수 있다. 반대로, PIPE-WS API(122)는, STEER-WS API(120)와 같은, 다른 TCE-WS API(106)를 호출할 수도 있다. PIPE-WS API(120)의 중요한 일 용도는, 의미 오브젝트들의 영속적 리포지토리인 소위 의미 오브젝트 뱅크 서비스를 실현하는 것이다. 뱅크 서비스는 일 환경의 사용자들에 의해 파일들, 연락처들, 스케줄 등과 같은 것들을 그 환경의 의미 오브젝트 제공 서비스들로서 남기는데 사용될 수 있으므로, 사람들은 (어쩌면 나중에라도) 그 서비스들을 태스크들(126)을 실현하는데 사용할 수도 있다.

PIPE-WS API(120)는, 사용자들이, 사용자가 PIPE-WS API(120)를 통해 게시한 의미 오브젝트들 또는 서비스들을 편성하는데 도움이 되는 관리 사용자 인터페이스도 포함한다. 펑크션들은 다음과 같은 것들을 포함한다:

1. 발견 범위 전환하기: 사용자는, 예를 들어, 서비스들을 일시적으로 "보류"하기 위해 PIPE-WS API(122)를 통해 게시된 서비스들을 위한 발견 범위를 전환할 수 있다(공백 발견 범위).

2. 만료 시간: 사용자는 서비스들을 위한 만료 시간을 설정할 수 있으므로, 만료 시간 이후에는 서비스가 발견될 수 없게 된다.

3. 호출 제한: 사용자는 가능한 호출들의 수에 대한 제한을 설정할 수 있으므로, 서비스가 그러한 호출 횟수 이후에는 발견될 수 없게 된다.

4. 명칭/설명: 사용자는 서비스의 명칭 및 텍스트 설명을 설정하거나 변경할 수 있다.

도 16d 내지 도 16n은, 본 발명의 실시예에 따른, OWL-S의 SSD들(116)에 대한 컴퓨터 해석 가능 소스 코드들의 3가지 일례들이다. 특히, 도 16d 내지 도 16f는 Company-1에 의해 생성된 document-1(1570)을 위한 SSD(116)이다. 도 16g 내지 도 16j는 (이 일례에서는, MS OUTLOOK과 같은, 주소록으로부터의 'Bob Smith'의 연락처인) OS/애플리케이션 오브젝트(1572)에 기초해 생성되는 SSD(116)이다. 그리고, 도 16k 내지 도 16n은 (이 일례에서는, "XYZ Project" 의미 오브젝트인) 의미 오브젝트(1574)에 기초해 생성되는 SSD(116)이다.

새로운 서비스들:

(1) 의미 인스턴스 직렬화 서비스들, (2) 정보 제공 서비스들, (3) (제한없이) 시간, 관련 날씨, 온도, 및/또는 감지될 수 있는 어떤 것과 같은, 감지기 서비스들, (4) 스냅샷 서비스들, (5) OWL 포매터 서비스, 및 (6) 텍스트 포매터 서비스를 포함하는 새로운 서비스들(112)이 설계되고 구현된다.

(1) 의미 인스턴스 직렬화 서비스들:

이 카테고리의 서비스들은 의미 인스턴스들의 임의 유형들을 소비하고, 그들을 직렬화하며 사용자들에게 정보를 전달한다. 일례는 "Tell Me" 서비스이다. 그것은 의미 인스턴스를 취하고, 그것을 인간이 이해 가능한 스트링으로 직렬화하며, 그것을 판독해 낸다. "Tell Me" 서비스의 세부 사항들은 다음과 같다.

일단 의미 인스턴스가 도달하면, 의미 인스턴스는 먼저 "Tell Me" 서비스에 의해 파싱된다. 그 다음, "Tell Me" 서비스는, 인스턴스의 클래스 또는 인스턴스의 오브젝트 특성들의 임의 클래스들을 위해 이용 가능한 임의의 직렬화 변환이 존재하는지를 알아보기 위해 그것의 변환 스크립트 리포지토리를 점검할 것이다. 변환 스크립트는 XSLT(Extensible Stylesheet Language) 스크립트일 수 있지만, 그에 한정되는 것은 아니다. 그러한 스크립트가 발견되면, 그것은 먼저 인스턴스에 적용되고 인스턴스(또는 인스턴스의 일부)를 스트링으로 변환한다. 이러한 변환 프로세스는, 인스턴스가 다른 인스턴스들을 그것의 오브젝트 특성들로서 포함할 경우에 회귀적으로 적용된다. (예를 들어, "Contact" 인스턴스는, 그것의 "hasBusinessAddress"와 "hasHomeAddress" 오브젝트 특성들 및 대응되는 스크립트들이 인스턴스에 적용되므로, "Address" 인스턴스들을 포함할 수도 있다.

예를 들어, 서비스가 다음의 "Address" 인스턴스를 수신한다고 가정한다:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:co="http://www.company-1.com/tce/ontologies/2004/03/object.owl#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>
<co:Address>
  <rdfs:label>Company-1, City Name</rdfs:label>
  <co:streetAddress>1000 Example Ave</co:streetAddress>
  <co:city>City Name</co:city>
  <co:state>State Name</co:state>
  <co:zipCode>Zip Code Number</co:zipCode>
   <co:country>Country Name</co:country>
</co:Address>
</rdf:RDF>
```

스크립트를 적용하는 것에 의해, 출력은 다음과 같다:

The "Address" of "Company-1, City Name" is "1000 Example Avenue, City Name, State, Zip Code Number, Country Name"

다음으로는, 변환의 결과 또는, 그러한 스크립트가 발견되지 않으면, 인스턴스 자체가 일반적인 직렬화 모듈로 송신된다. 모듈의 목적은 디폴트 로직을 사용해 임의의 의미 인스턴스들을 직렬화하는 것이다. 상기 일례에서, 어드레스 인스턴스를 위한 스크립트가 존재하지 않으면, 인스턴스는 다음과 같이 직렬화될 것이다:

The "Address", "Company-1, City Name" has "1000 Example Avenue" as the "Street Address", "City Name" as the "City", "State Name" as the "State", "Zip Code Number" as the "Zip Code", and "Country Name" as the "Country"

"Tell Me" 서비스의 마지막 단계는 직렬화된 스트링을 판독해 내는 것이다. "Tell Me" 서비스의 직렬화 모듈은, 스트링을 티커 장치(ticker device)에 디스플레이하기 위한 "Show Me" 서비스와 같은, 다수의 유사한 다른 서비스들에 의해서도 사용될 수 있다.

"Tell Me" 서비스는 VoiceSTEER와 함께 사용될 경우에 특히 유용할 수 있다. 이것은 (의미론적으로) 하나 이상의 입력 및 비 출력을 가진 서비스이다. "Tell Me" 서비스는, 그것이 수신하는 의미 오브젝트(들)를 입력으로서 판독해 낸다. 의미 오브젝트가 이 서비스에 대한 공지 유형일 경우, 그것은 각각의 공지 유형을 위한 XSLT 스크립트와 같은 내부 메커니즘들을 사용해 오브젝트 판독 방법을 판정한다. 오브젝트가 서비스에 대해 비공지라면, 그것은 먼저, 그것의 판독 방법에 대한 정보가 존재하는지를 알아보기 위해 오브젝트 온톨로지를 찾는다. 그것 또한 실패라면, 그것은 디폴트 방법으로 오브젝트가 참조하는 온톨로지를 사용해 그것을 판독해 낸다.

예를 들어, 서비스가 다음의 "Address" 오브젝트를 수신한다고 가정한다:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:co="http://www.company-1.com/tce/ontologies/2004/03/object.owl#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>
<co:Address>
  <rdfs:label>Company-1, City Name</rdfs:label>
  <co:streetAddress>1000 Example Ave</co:streetAddress>
  <co:city>City Name</co:city>
  <co:state>State Name</co:state>
  <co:zipCode>Zip Code Number</co:zipCode>
   <co:country>Country Name</co:country>
</co:Address>
</rdf:RDF>
```

서비스가 "Address" 오브젝트에 관해 알지 못할 경우, 그것은 다음과 같이 판독해 낼 수도 있다:

The "Address", "Company-1, City Name" has "1000 Example Avenue" as its "Street Address", "City Name" as its "City", "State Name" as its "State", "Zip Code Number" as its "Zip Code", and "Country Name" as its "Country"

"co:streetAddress"를 위한 "Street Address"와 같은 특성들을 위한 레이블들은, "Address" 오브젝트가 참조하는 온톨로지로 정의된다. 의미 오브젝트는 (XSLT 코드 피스(code piece)와 같은) 변환 펑크션을 제시하는 것에 의해 스트링으로 직렬화될 수 있다. 온톨로지 파일내에서, 직렬화 펑크션이 정의되거나 "Tell Me" 서비스가 자신의 고유한 지식 기초로 "Address" 오브젝트의 직렬화 방법을 알고 있다면, 그것은 펑크션을 적용하여 이 오브젝트를 다음과 같이 판독해 낼 수도 있다:

The "Address" of "Company-1, City Name" is "1000 ExampleAvenue, City Name, State Name, Zip Code Number, Country Name"

이러한 "Tell Me" 서비스는 임의의 오브젝트들을 그것의 입력으로서 수용한다. 그것은 "Tell", "What is"와 같은 다수 명칭들을 가질 수도 있다(그러한 명칭들은 동일한 OWL-S 파일로 또는 별도의 OWL-S 파일들로 제공될 수도 있다.) 그것은, 사용자가 의미 오브젝트를 조사할 자연스러운 방법을 제공할 것이다. 예를 들어, "Temperature"라고 하는 서비스가 현재의 방 온도를 리턴한다면, 사용자는 VoiceSTEER에 다음과 같이 질문할 수 있다:

Computer, "Tell Me" (the) "Temperature"?

Computer, "What is" (the) "Temperature"?

서비스는 다른 이름들을 가질 수도 있고, 서비스가 태스크 컴퓨팅 클라이언트들의 자동 번역 서비스들의 삽입 메커니즘과 조합될 수 있도록, 오브젝트의 특정 종류들을 수용하게 될 수도 있다. 예를 들어, "Where is"는 "Location" 오브젝트들을 수용하는 상이한 이름을 갖춘 "Tell Me" 서비스라고 하자. 그의 위치를 포함하는 Commander Data의 "Contact" 정보를 제공하는 "Commander Data" 서비스 및 "Contact"로부터 "Location"을 추출하는 "Location of" 서비스가 존재한다고 가정한다. 그 다음, VoiceSTEER에 다음과 같이 말할 수 있다:

Computer, "Where is" "Commander Data"?

그 다음, "Location of" 서비스는 자동적으로 삽입될 것이고, 서비스 구성, "Commander Data", "Location of', "Where is" 가 실행될 것이며, Commander Data의 위치가 판독될 것이다. 물론, 다음과 같이 말할 수도 있다:

Computer, "Tell Me" (the) "Location of' "Commander Data"?

Computer, "What is" (the) "Location of' "Commander Data"?

서비스가 오브젝트의 특정 종류들을 수용할 수 있게 하는 것에 의해, 서비스는 의미 서비스들의 서비스 관리 펑크션과 조합될 수도 있다. 예를 들어, "Where is"가 "Tell Me" 서비스라고 하자. "View on projector" 서비스는 그것의 SSD에서 위치 정보를 제공한다고 가정한다. 그 다음, VoiceSTEER에 다음과 같이 말할 수도 있다:

Computer, "Where is" "View on Projector"?

그 다음, "Location of" 서비스 관리 펑크션(서비스 관리 펑크션의 세부 사항들은 다음에서 좀더 상세하게 논의된다)은 "Where is"와 "View on Projector" 사이에 자동적으로 추가된다.

"Where is"가 실행될 것이고 "View on Projector"의 위치가 판독될 것이다. 물론, 다음과 같이 말할 수도 있다:

Computer, "Tell me" (the) "Location of' "View on Projector"?

Computer, "What is" (the) "Locationof' "View on Projector"?

"Tell Me" 서비스가 그것이 가장 유용할 VoiceSTEER와의 이용 맥락에서 설명되기는 하지만, 그것은 임의의 태스크 컴퓨팅 클라이언트들과 함께 사용될 수도 있다.

(2) 정보 제공 서비스들:

정보 제공 서비스들은 입력을 취하지 않고, 일단 호출되고 나면, 의미 인스턴스들을 출력으로 생성한다. 정보 제공 서비스들과 인스턴스 제공 서비스들간의 차이는 정보 제공 서비스들에 의해 발생된 인스턴스들은 시간에 따라 상이하지만 인스턴스 제공 서비스들에 의해 발생된 것들은 항상 동일하다는 점이다. 정보 제공 서비스들의 일부 일례들로는 온도 서비스, 시간 및 날짜 서비스를 들 수 있다.

"Temperature Service"를 예로 들자면, 일단 호출되고 나면, 그것은 센서에 의해 현재 온도를 점검하고 최종 값으로 의미 인스턴스를 생성한다. 그것은, 앞서 언급된 "Tell Me" 서비스와 조합될 경우에 아주 유용한데, 사용자가 음성-기반 태스크 컴퓨팅 클라이언트를 통해, 다음과 같이: "Computer, Tell me (the) temperature of the conference room" 명령할 때, 사용자는 다음과 같은 대답:"The temperature is 75 degree"을 듣게 될 것이다.

(3) (제한없이) 시간, 관련 날씨, 온도, 및/또는 감지될 수 있는 어떤 것과 같은, 센서 서비스들이 정보 제공 서비스의 일 유형이다.

Time/Temperature 서비스들과 같은, 센서 서비스들은 의미 오브젝트 제공자 서비스들이다. 일단 호출되고 나면, 서비스들은 장치들, 웹 서비스들, 웹 페이지들, 및/또는 다른 정보 소스들을 조회할 수 있고 현재의 시간/온도를 의미 오브젝트들로서 리턴한다.

(4) 스냅샷 서비스

스냅샷 서비스는 디지털 카메라, 디지털 비디오 카메라, 스캐너 등과 같은 이미지 장치들로부터 스틸 이미지를 캡처할 것이고, 그것이 호출될 때, "image" 의미 오브젝트를 리턴한다.

(5) OWL 포매터 서비스

OWL 포매터 서비스는 의미 오브젝트들을 그것의 입력으로서 수용하고, 그들을 인간이 이해할 수 있는 방식으로 포매팅하여 그것을 출력으로서 리턴한다. 현재 기술들의 구현들 중 하나에서, 그것은 그러한 오브젝트들의 설명들을 위해 사용되는 온톨로지들을 사용해 HTML의 Table 포맷으로 의미 오브젝트들을 포매팅하고 HTML의 의미 오브젝트들 자체를 또는 그들로의 URL을 리턴한다.

(6) 텍스트 포매터 서비스

텍스트 포매터 서비스는 의미 오브젝트들을 그것의 입력으로 수용하고, 그들을 소정의 텍스트 포맷들 중 하나로 포매팅하며, 그것을 텍스트 파일들로서 저장하고, 그것을 소정 파일에 첨부하는 식이다. 예를 들어, 텍스트 포매터 서비스는 "Book"

의미 오브젝트를 수용할 수 있고 그것을 BibTeX 포맷으로 포매팅할 수 있으며 그것을 사용자의 고유한 BibTeX 파일에 첨부할 수 있다. 또는, "Protein"과 같은 생물 정보학 오브젝트가, 다른 텍스트 포매터 서비스에 의한 Blast 애플리케이션에 의해 사용되는 포맷으로 포매팅될 수도 있다.

그것은 XSLT 및 다른 스크립트들을 사용해 구현될 수도 있다. 예를 들어, 텍스트 포매터 서비스는 의미 오브젝트와 대응되는 포매팅 XSLT 스크립트 쌍들의 표를 보유할 수도 있다. 그것이 의미 오브젝트를 수신할 때, 텍스트 포매터는, 표에 의해 어떤 XSLT 스크립트를 사용해 그것을 포매팅할 것인지를 판정하고 사용자가 그것을 저장하거나 그것을 첨부할 파일을 선택하기 위한 다이얼로그 박스를 팝업한다.

태스크 컴퓨팅 클라이언트(119) 내부 서비스들:

도 17은, 본 발명의 실시예에 따른, 내부 서비스들을 갖춘 STEER-WS TCC(119a)의 컴퓨터 디스플레이 스크린 그래픽 사용자 인터페이스의 이미지이다. 내부 서비스들은, 태스크 컴퓨팅 클라이언트들(119;STEER TCC, 화이트 홀, 서비스 관리자 등)과 밀접하게 번들링된 서비스들이다. 그것은 TCC들(119)내에서 특수한 방식으로 처리되므로, 내부 서비스들이 임의의 소정 발견 메커니즘들에 통해 발견되지 않는다 하더라도, 그들은 클라이언트들의 GUI 어디에든 등장할 수 있다.

일반적으로, 그러한 내부 서비스들은 지나치게 일반적이다(임의 "Thing"을 제공 및/또는 소비한다). 따라서, 그러한 내부 서비스들은 태스크 컴퓨팅 클라이언트들에 의해 다른 로컬 및 편재성 서비스들(112)과는 상이한 방식들로 제공된다. 내부 서비스가 다른 서비스들(112)과는 전혀 상이하게 취급될 수도 있지만, 그것의 의미 서비스 설명은 외관상 상이하지 않다. 이로 인해, 내부 서비스들을 복합 서비스로 저장하고 복합 서비스를 다른 서비스들과 공유하는 것이 가능하다.

내부 서비스들과의 조합 실행은 다음과 같다. 실행 엔진이 내부 서비스에 직면할 때, (실행 엔진은, 모든 내부 서비스들이 공지의 WSDL URL을 갖춘 통상적인 WSDL 웹 서비스로서 설명되기 때문에, 그것을 알고 있다) 엔진은 WSDL 동작명을 점검할 것이고 어떤 내부 서비스가 그것인지를 결정할 것이다. 일단 그것이 결정되고 나면, WSDL 웹 서비스를 직접적으로 호출하는 대신, 엔진은 내부 서비스를 핸들링하기 위한 특수 모듈을 개시한다.

여기에서 설명되는 실시예들의 태양에 따르면, 내부 서비스를 위한 고정 URL에서 실제 웹 서비스를 구현할 수 있는데, 이는 내부 서비스와 동일한 목적을 가진다. 이것은, 구현된 내부 서비스 메커니즘들이 존재하지 않는 일부 클라이언트들이 여전히 URL의 서비스를 호출할 수 있을 때의 일부 경우들에서 유용하다. 분명히, 본 발명의 접근 방법에서는, 그것을 내부 서비스로서 호출하는 것이 좀더 효율적이다.

여기에서는, 다음과 같은 구현된 내부 서비스들이 설명된다: (1) 인스턴스 생성 서비스, (2) 인스턴스 복사기, (3) 인터셉팅 서비스, (4) 인스턴스 저장 서비스, 및 (5) 특성 선택 서비스. 인스턴스 생성기, 인스턴스 복사기, 인터셉터, 및 인스턴스 저장기는 태스크(126) 실행 흐름 제어에 관련된 4개의 내부 클라이언트(119) 서비스들이다. 이들은, 의미 오브젝트들을 다룰 때 일부의 공통 모듈들을 공유한다. 공통 모듈들은, 사용자가 의미 인스턴스를 파일로서 동적으로 저장하거나 로컬 또는 편재형 인스턴스 제공 서비스를 게시할 수 있게 한다. (그에 따라, 그것은 의미 인스턴스(또는 그것의 오브젝트 특성)를 의미 서비스 구성으로 공급할 것이다.) 사용자는 로컬 저장 공간 또는 전체 인스턴스나 인스턴스의 오브젝트 특성들을 위한 웹 사이트의 파일로부터 의미 인스턴스를 로드할 수도 있다. 그에 따라, 그것은 의미 인스턴스를 의미 서비스 구성 실행으로부터의 결과로서 로드하게 될 수도 있다. 공통 모듈들 또한 ("Integer", "Time" 등과 같은) 온톨로지에 기초해 데이터에 대한 유효성 점검을 수행한다.

(1) 인스턴스 생성기:

도 17에는, "Instance Creator"(1602)를 위한 선택 가능한 그래픽 디스플레이가 표시되어 있다. 인스턴스 생성 서비스는, 온톨로지에 기초해 임의의 의미 유형을 위한 양방향 인터페이스를 발생시키고 사용자로 하여금 인터페이스로부터 그 유형의 의미 인스턴스를 생성할 수 있게 하는 서비스이다. 그것은, 사용자가 입력으로 서비스를 테스트하고 싶어하지만, 그 유형의 입력을 제공하는 서비스들을 전혀 가지고 있지 않을 경우에 유용하다. 인스턴스 생성 서비스는 입력을 취하는 임의 서비스들 이전에 놓여질 수 있다. 인스턴스 생성 서비스의 출력 유형은 그 이후의 서비스 입력 유형과 동일하다.

(2) 인스턴스 복사기

복사기 서비스는 실행 시퀀스의 2개 서비스들 사이에 배치된다. 실행 흐름의 그 포인트에서, 그것은 단지 그것의 입력을 그것의 출력으로 복사할 뿐 다른 것은 수행하지 않는다. 그것은 주로, 다수 서비스들로 복사된 구성으로의 입력들 중 일부를 갖춘 구성들을 저장하는데 사용된다. 복사기가 없다면, 저장된 구성은 정확하게 동일해야 하는 다수 입력들을 가져야

한다. 복사기의 경우, 저장된 구성은 단 하나의 입력만을 가질 수도 있다. 저장된 구성내에서 내부적으로는, 제 1 서비스로서의 인스턴스 복사기가 입력을 수용한 다음, 입력은 구성내의 다수 서비스들로 복사된다. 복사기의 입력 및 출력 유형은 선행 서비스의 출력 유형과 동일하다.

(3) 인스턴스 인터셉터:

인터셉터 서비스는 실행 시퀀스의 2개 서비스들 사이에 배치된다. 일단 삽입되고 나면, 그것은 실행 흐름의 그 포인트에서 중단하여, 선행 서비스의 출력을 파싱하고 디스플레이한다. 사용자는 계속하기 전에 그 값을 검토하고 업데이트할 기회들을 가진다. 결과에 만족하지 않으면, 실행을 중단할 것을 선택할 수도 있다. 한편, 중간 결과들은 파일로 저장되거나 의미 인스턴스로서 게시될 수도 있다. 인터셉터 서비스는 임의의 2개 서비스들 사이에 배치될 수 있다. 인터셉터의 입력 및 출력 유형은 선행 서비스의 출력 유형과 동일하다.

(4) 인스턴스 저장기:

도 17에는 "Instance Saver"(1604)를 위한 선택 가능한 그래픽 디스플레이가 도시되어 있다. 인스턴스 저장 내부 서비스(1604)는 임의의 의미 인스턴스들을 분석하고 저장한다. 일부 시나리오들에서는, 서비스에 의해 발생된 출력이 환경내의 다른 임의 서비스들에 의해 소비될 수 없다. 인스턴스 저장 서비스가 없다면, 결과는 손실될 것이다. 그에 의해, 사용자는 결과를 장차의 사용을 위해 또는 서비스가 그것을 소비하는데 이용될 수 있는 다른 환경에서 그것을 사용하기 위해 결과를 저장하는 추가 옵션을 가진다. 인스턴스 저장 서비스는 출력들을 발생시키는 임의 서비스들 이후에 배치될 수 있다. 인스턴스 저장 서비스의 출력 유형은 선행 서비스의 출력 유형과 동일하다.

(5) 특성 선택기

특성 선택기는 출력 일부를 추출하여 후속 서비스로 송신하는 서비스이다. 그것은, 서비스가 다른 서비스에 의해 발생되는 출력의 일부에만 관심이 있을 경우에 유용하다. 예를 들어, "My Contact" 서비스는 사용자로 하여금 사용자의 Outlook으로부터 연락처 항목을 선택하고 "Contact" 인스턴스를 발생시킬 수 있게 한다. "Map of" 서비스는 "Address" 인스턴스를 수용하고 그 주소의 맵을 디스플레이한다. 이들 2가지 서비스들은 함께 링크될 수 없는데, "Contact"와 "Address" 사이에는 수퍼-서브-클래스 관계가 존재하지 않기 때문이다. 그러나, "Contact" 인스턴스는 "Address" 유형의 "hasBusinessAddress"라고 하는 특성을 가진다. 특성 선택기 서비스는 여기에서, 사용자가 가능한 구성의 이 유형을 발견하도록 돕는데 사용된다.

특성 선택기 서비스는, 제 2 서비스의 입력이 제 1 서비스 출력의 특성인(또는 회귀적으로 그러한) 2개 서비스들 사이에 배치될 수 있다. 특성 선택기의 입력 유형은 특성 선택기 이전의 서비스 출력 유형과 동일하고, 특성 선택기의 출력 유형은 특성 선택기 이후의 서비스 입력 유형과 동일하다.

여기에서는, 태스크 컴퓨팅(100) 환경을 프레젠테이션 클라이언트 프로세싱 계층, 원격 절차 호 API(application programming interface), 의미론적으로 컴퓨터 시스템상의 서비스로서 설명되는 펑크션의 컴퓨터 시스템 소스로의 프레젠테이션 계층에서 컴퓨터 구현형 태스크 인터페이스를 실시간에서 동적으로 발생시키기 위한 원격 절차 호 API를 통해 프레젠테이션 계층이 인터페이싱하는 미들웨어 서버 프로세싱 계층; 및 미들웨어 프로세싱 계층이 인터페이싱하는 의미론적으로 컴퓨터 시스템상의 서비스로서 설명되는 펑크션의 컴퓨터 시스템 소스를 제공하는 서비스 계층 및 펑크션 소스 실현 계층의 복수개 컴퓨터 시스템 구현층들로 분할하고; 컴퓨터 시스템상의 하나 이상 서비스들을 위한 프레젠테이션 계층에서 발생된 태스크 인터페이스에 따라, 하나 이상 서비스들을 포함하는 실행 가능 태스크를 실시간에서 동적으로 구성하는 것에 의한 태스크 컴퓨팅 컴퓨터 시스템의 구현이 설명된다. 컴퓨터 서비스는, 의미론적으로 설명되는 애플리케이션-, 장치- 및 서비스-리치 컴퓨터에 기초해, 컴퓨터상의 서비스에 대해 발생된 인터페이스를 사용해 실행 가능 태스크로 실시간에서 동적으로 구성된다. 여기에서 설명되는 실시예들의 태양에 따르면, 사용자는 실질적으로, 효과적으로, 효율적으로, 동적으로, 그리고 실시간으로, 구성 가능한 통합 사용자 인터페이스(구성 및 실행 펑크션들)에 의존해 상호 작용을 관리하고 편재성 컴퓨팅 환경과 상호 작용한다.

태스크 컴퓨팅은 (a) SemanticWeb 기술들을 이용해, 더 많은 리소스들의 (의미) 웹이 유비쿼터스 컴퓨팅 애플리케이션들에 즉각적으로 이용될 수 있게 하는 것을 추구하고, (b) 리소스들이 발견되고, 액세스되며, 태스크 컴퓨팅(100) 시스템에 의해 그들이 사용될 수 있게 하는데 이용될 수 있는 서비스 추상화(116)에 접속되거나 서비스 추상화(116)와 통신되는 방법에 상관없이, 리소스들의 특징에 관해 상당히 관용적인 접근 방법이다. 태스크 컴퓨팅은 의미론적으로 모든 기능의 세계

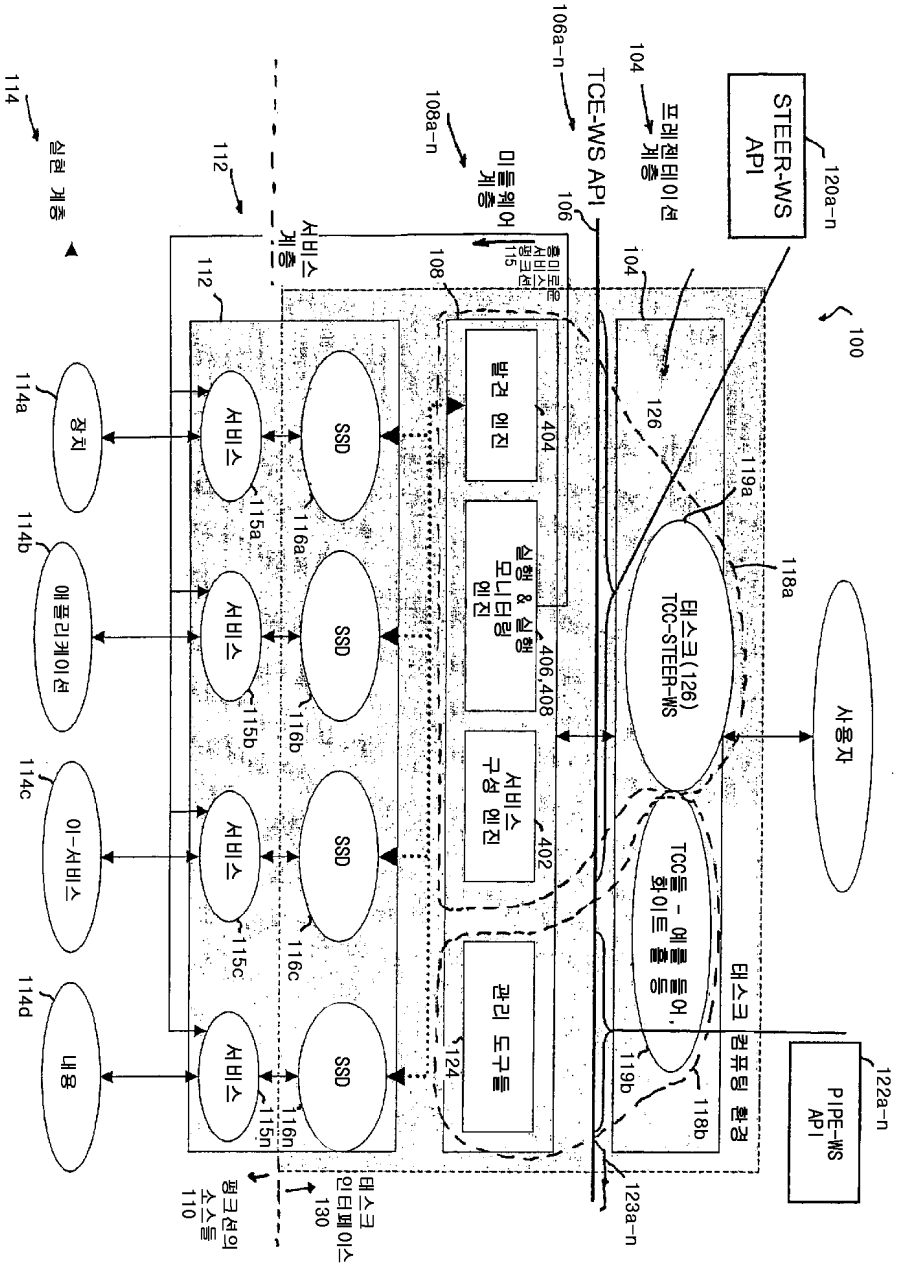
적 추상화로서 설명되는 서비스들(116)에 의존하고; 또한, 태스크 컴퓨팅은, 구성 가능 태스크들(126)이 많은 서비스들(112)을 수반할 수 있기 때문에, 장치-대-서비스 상호 운용성보다 더 큰 범위를 가진다. 예를 들어, 통상적인 태스크 컴퓨팅(100) 시스템 태스크(126)는 실시간에서 동적으로 5-6개 서비스들(112)을 이용할 수도 있다.

본 발명의 상술된 바람직한 실시예들은 (공지의 임의 컴퓨터 판독 가능 매체들에 저장되어 있는) 소프트웨어 및/또는 프로그램 가능한 장치/컴퓨팅 장치(예를 들어, 데이터를 저장, 검색, 제시(예를 들어, 디스플레이) 및 프로세싱할 수 있는 프로그램 가능한 전자 장치) - (제한없이) 퍼스널 컴퓨터, 클라이언트-서버 네트워크 아키텍처 경우의 서버 및/또는 클라이언트 컴퓨터, 분산 네트워크 아키텍처의 네트워킹된 컴퓨터들, 터미널 장치, PDA(Personal Digital Assistant), 모바일 장치와 같은, 임의 유형의 프로그램 가능한 전자 컴퓨팅 장치를 제어하는 프로그램 가능한 컴퓨팅 장치/하드웨어로 구현된다.

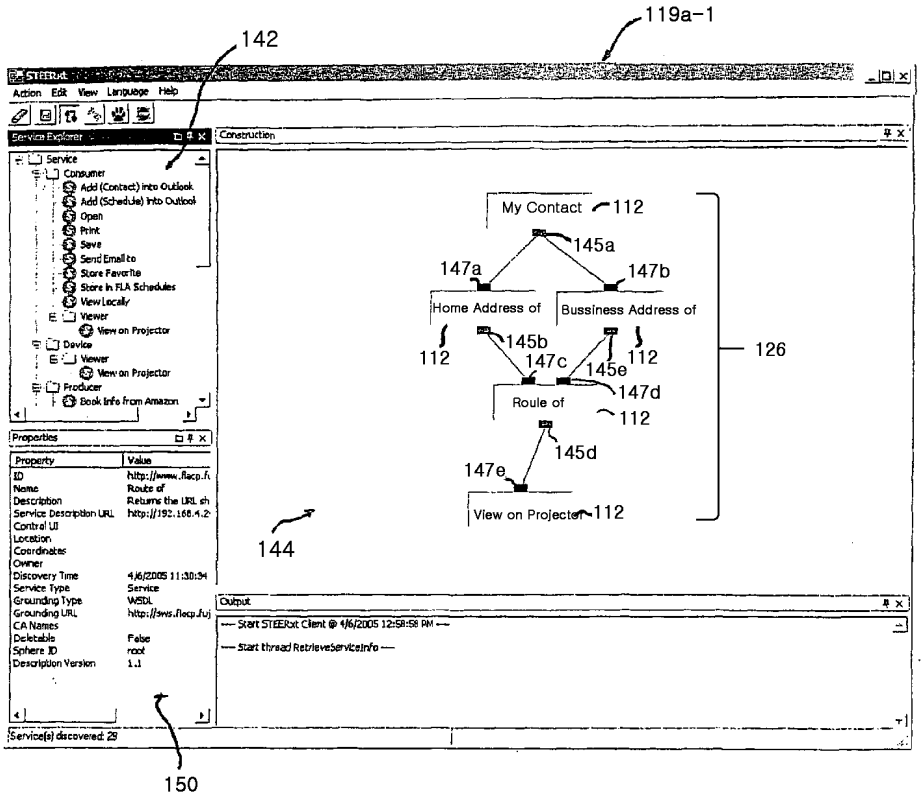
본 발명의 많은 사양들 및 이점들은 상세한 설명으로부터 명백하고, 그에 따라, 첨부된 청구항들에 의해, 본 발명의 정신 및 범위내에 해당되는 본 발명의 그러한 사양들 및 이점들 모두를 커버하고자 한다. 또한, 당업자들에게는 다수의 변경들 및 변형들이 용이할 것이므로, 본 발명은 예시되고 설명된 정확한 구성 및 동작으로 한정되지 않으며, 따라서, 적당한 모든 변경들 및 등가물들은 본 발명의 범위내에 해당하는 것으로 분류될 수 있다.

도면

도면1a


STEER-WS API
120a-n
프레젠테이션 계층
104
TCE-WS API
106a-n
106
미들웨어 계층
108a-n
108
서비스 계층
112
실행 계층
114
100
126
119a
118a
태스크(126) TCC-STEER-WS
사용자
TCC들 - 에를 들어, 화이트 홀등
태스크 컴퓨팅 환경
119b
118b
122a-n
PIPE-WS API
123a-n
124
관리 도구들
402
서비스 구성 엔진
406,408
실행 & 실행 모니터링 엔진
404
발견 엔진
115
미들웨어 서브계층
112
서비스
115a
115b
115c
115n
SSD
116a
116b
116c
116n
114a
장치
114b
에플리케이션
114c
이-서비스
114d
내용
태스크 인터페이스
130
공유 섹션의 소스들
110

도면1b


119a-1
142
My Contact
112
145a
147a
147b
Home Address of
Bussiness Address of
112
145b
147c
145e
112
126
147d
Roule of
112
145d
147e
View on Projector
112
144
150

도면2

STEER-WS API (120)

STEER-WS API(120)는 웹 서비스이다. 시작하기 위해, 예를 들어, HostedSTEER-WS TCC(119) 또는 다른 애플리케이션 클라이언트(들)(119)를 개시한다. 웹 서비스는 http://localhost/steerws/service.asmx. 에서 이용 가능하다. 이것은 다음의 연산들을 가진다:

addRomoteSite 원격 발견 모듈을 STEER에 추가하기
checkExecutionStaus 실행 참조 ID가 주어진 상태에서, 실행의 현재 상태 찾아내기. 출력은 code\pid\extraInfo 의 포맷을 갖는데, 여기에서, code는 실행 상태 코드 (COMPELTE, ERROR, CANCEL, DONE, PREPARE, CONTROLUI) 이고, pid는 현재적으로 실행 중인 프로세스의 ID이며, extraInfo는 제어 UI, 출력 결과, 또는 오류 메시지임.
deleteService 서비스 ID를 제공하는 것에 의해, 소정 서비스 삭제하기
excuteService 단일 서비스 실행하기
excuteServiceList2 서비스의 시퀀스 실행하기. 서비스 리스트의 경우, 입력은 s1/s2/s3...와 같고, 파라미터 쌍의 경우, 입력 리스트를 의미하는 공백이거나, 시퀀스의 소정 번호에서의 서비스 파라미터를 의미하는 output@number=input@numb34 일 수 있음(이 경우, output 및 input 모두는 파라미터 ID이고 number는 0에서 서비스-1의 수까지임).
executeOWLS OWL-S 설명을, 서비스 지식 베이스에 추가하지 않으면서, 실행하기
filterServicesByProperties 기존 서비스들을 조건들에 의해 필터링하기
filterServicesByProperties2 서비스들의 리스트를 조건들에 의해 필터링하기
findAllCompositions 가능한 모든 구성들을 찾아내기
findAllMatchedPairs 구성 윈도을 위해 설계된 모든 생산자, 소비자 쌍들을 찾아내기.
findAllServices 모든 서비스들을 열거하기. 각 항목은 서비스 ID, 서비스 유형 및 (모든 언어들의) 이름들을 가짐.
findAllServicesInSphere 구 ID가 주어진 상태에서, 구에 의해 발견되는 모든 서비스들을 찾아내기
findCompositionsByTerms 키워드들의 리스트가 주어진 상태에서, 그러한 키워드들과의 모든 구성들 찾아내기
findMatchedParameters 구성될 수 있는 2가지 서비스들이 주어진 상태에서, 입력 및 출력 파라미터들 찾아내기
findRelatedServices 소정 서비스들 전후에 직접적으로 배치될 수 있는 서비스들 찾아내기
findServiceCandidates 서비스들의 리스트가 주어진 상태에서, 그들에 의해 구성될 수 있는 모든 서비스들 찾아내기
getClassProperties 소정 온톨로지 클래스의 모든 특성들(데이터 및 오브젝트) 찾아내기
getDirectSubclass 소정 온톨로지 클래스의 직접적인 하부 클래스 찾아내기
getDirectSuperclass 소정 온톨로지 클래스의 직접적인 수퍼 클래스 찾아내기
getNames 임의의 OWL 엔티티가 주어진 상태에서, (모든 언어들의) 그것의 명칭들 찾아내기
getServiceDescription 서비스의 OWL-S 설명 찾아내기
getServiceProperty 서비스의 모든 특성들 찾아내기
holdDiscoveryModule 실행 중인 발견 모듈 보류하기
isRunning STEER가 실행 중인지 검출하기
lastUpdateTime 서비스가 추가/삭제된 최종 시간 찾아내기. 이를 이용해 STEER에서의 모든 업데이트들 찾아내기
listDiscoveryModules 실행 중인 모든 발견 모듈들 열거하기
matchByParameter 파라미터의 URI가 주어진 상태에서, 매칭되는 모든 파라미터들(입력을 위한출력들 및 출력을 위한 입력들) 찾아내기
matchedServiceByParameter 파라미터의 URI가 주어진 상태에서, 매칭되는 모든 서비스들 찾아내기.
queryByRDQL 기존의 서비스 지식 베이스에 대해 RDQL 쿼리 실행하기
removeRemoteSite 발견 리스트로부터 원격 사이트 제거하기
restartDiscoveryModule 기존의 발견 모듈 재시작하기
restoreDiscoveryModule 대기 중인 발견 모듈 시작하기
saveComposition 실행 계획을 새로운 서비스로 저장하고 서비스 설명 리턴하기
sortServices 서비스들의 리스트가 주어진 상태에서, 그들을 실행 가능한 순서로 정렬하기
stopExecution 실행 추적 ID를 제공하는 것에 의해, 실행 중인 실행 중단하기

주: STEER TCC 웹 서비스의 WSDL

도면3a

300

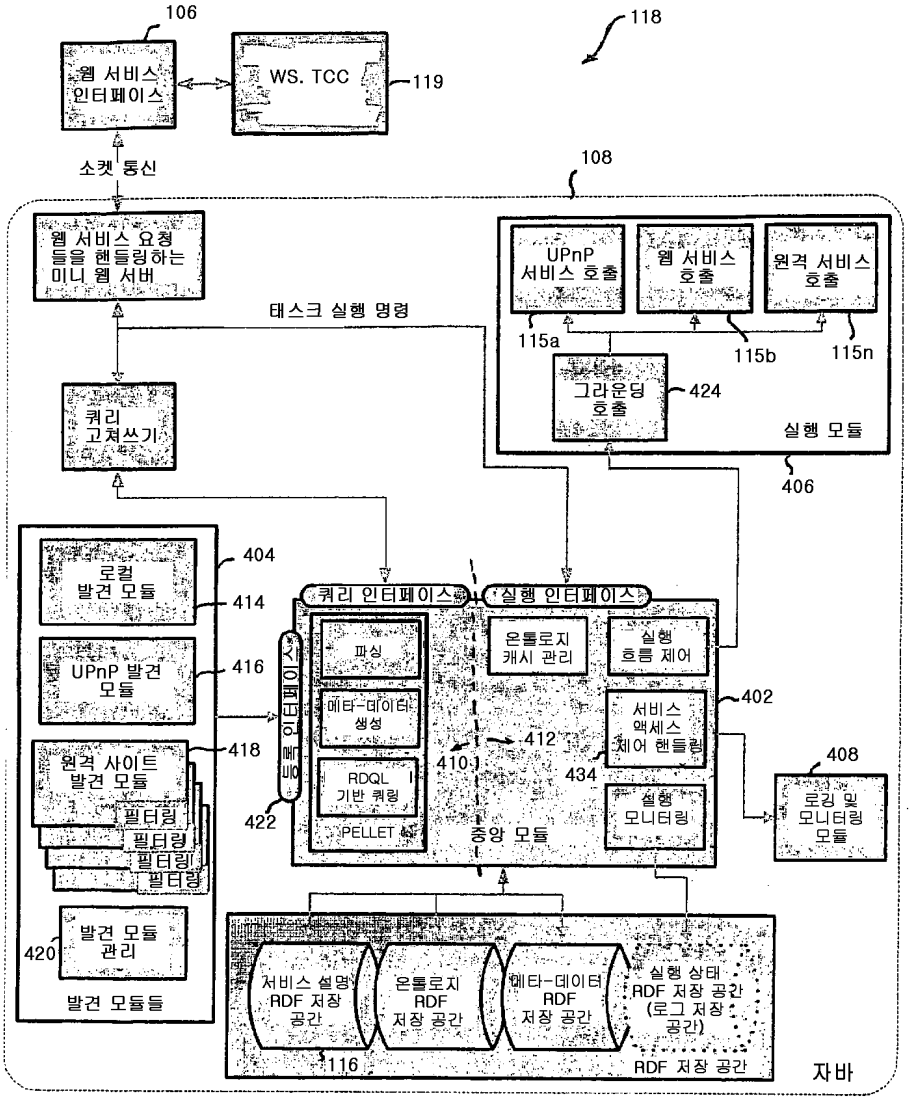
```
string newUpdateTime = lastUpdateTime ();
if (newUpdateTime != oldUpdateTime) {
      // retrieve service list, update your local knowledge
      oldUpdateTime = newUpdateTime;
} else {
      // The old service list is still valid, no need to do //
      any changes.
}
```

도면3b

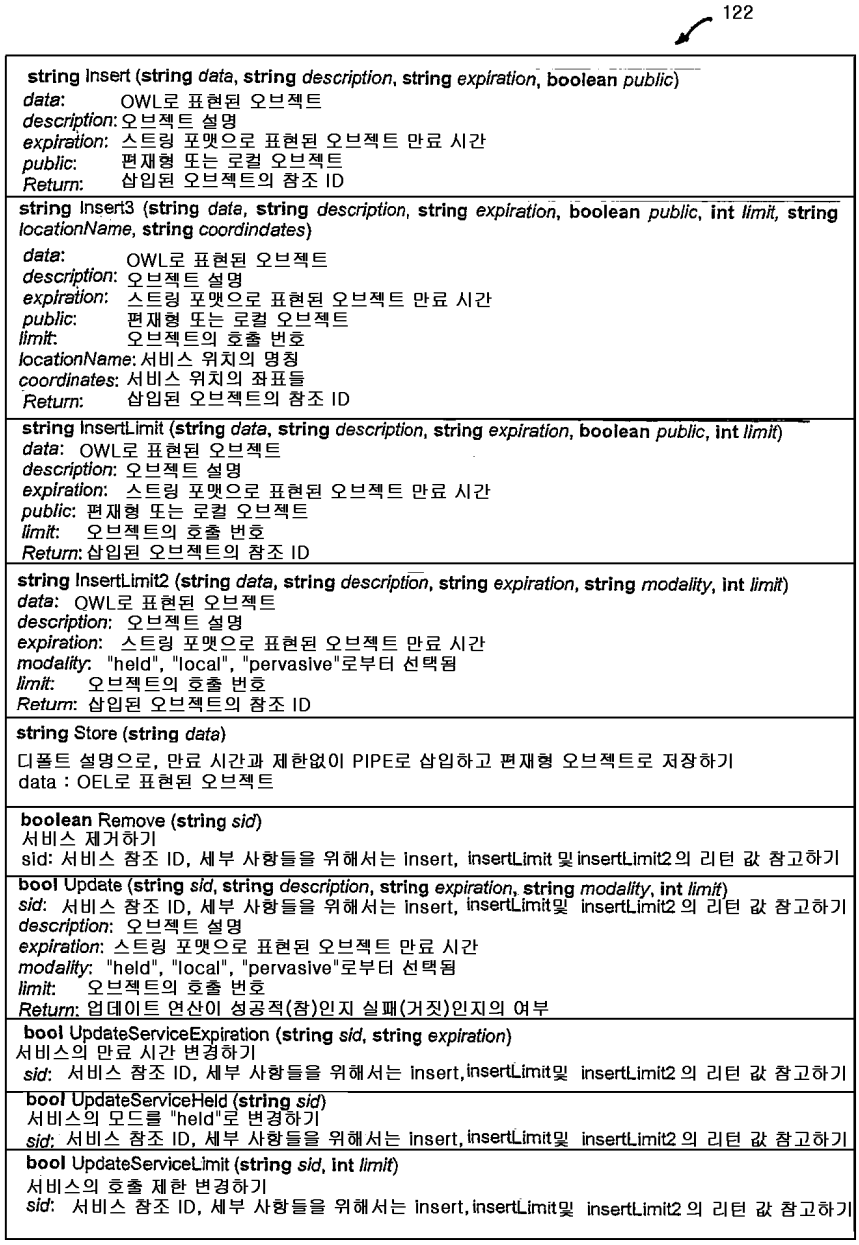
310

```
int executionID = startExecuteServiceList(serviceList);
bool InProcess = true;
string previousStatus = "";
while (InProcess) {
  string status = checkExecutionStatus(executionID);
  If (status == "DONE" or status == "ERROR") {
      // Execution is ended or stopped because of errors
      InProcess = false;
  else if (status != "" and status != previousStatus) {
     // Open status in a Web Browser for Control UI
  }
  previousStatus = status
}
```

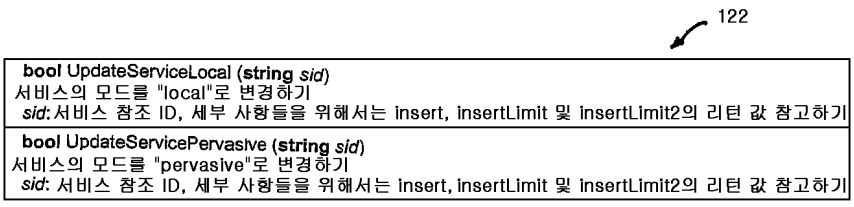
도면4



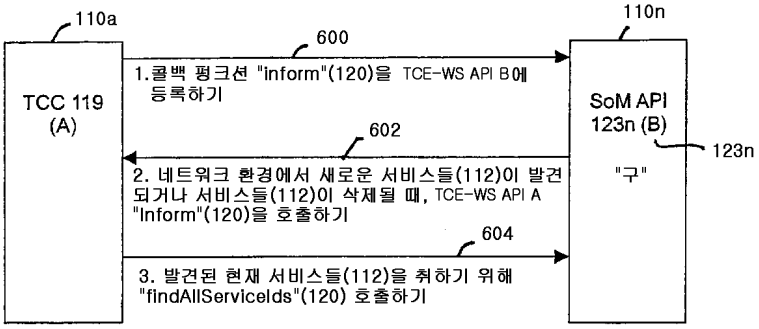
도면5a


122
string Insert (string data, string description, string expiration, boolean public)
data: OWL로 표현된 오브젝트
description: 오브젝트 설명
expiration: 스트링 포맷으로 표현된 오브젝트 만료 시간
public: 편재형 또는 로컬 오브젝트
Return: 삽입된 오브젝트의 참조 ID
string Insert3 (string data, string description, string expiration, boolean public, int limit, string locationName, string coordindates)
data: OWL로 표현된 오브젝트
description: 오브젝트 설명
expiration: 스트링 포맷으로 표현된 오브젝트 만료 시간
public: 편재형 또는 로컬 오브젝트
limit: 오브젝트의 호출 번호
locationName: 서비스 위치의 명칭
coordinates: 서비스 위치의 좌표들
Return: 삽입된 오브젝트의 참조 ID
string InsertLimit (string data, string description, string expiration, boolean public, int limit)
data: OWL로 표현된 오브젝트
description: 오브젝트 설명
expiration: 스트링 포맷으로 표현된 오브젝트 만료 시간
public: 편재형 또는 로컬 오브젝트
limit: 오브젝트의 호출 번호
Return: 삽입된 오브젝트의 참조 ID
string InsertLimit2 (string data, string description, string expiration, string modality, int limit)
data: OWL로 표현된 오브젝트
description: 오브젝트 설명
expiration: 스트링 포맷으로 표현된 오브젝트 만료 시간
modality: "held", "local", "pervasive"로부터 선택됨
limit: 오브젝트의 호출 번호
Return: 삽입된 오브젝트의 참조 ID
string Store (string data)
디폴트 설명으로, 만료 시간과 제한없이 PIPE로 삽입하고 편재형 오브젝트로 저장하기
data : OEL로 표현된 오브젝트
boolean Remove (string sid)
서비스 제거하기
sid: 서비스 참조 ID, 세부 사항들을 위해서는 insert, insertLimit 및 insertLimit2의 리턴 값 참고하기
bool Update (string sid, string description, string expiration, string modality, int limit)
sid: 서비스 참조 ID, 세부 사항들을 위해서는 insert, insertLimit및 insertLimit2 의 리턴 값 참고하기
description: 오브젝트 설명
expiration: 스트링 포맷으로 표현된 오브젝트 만료 시간
modality: "held", "local", "pervasive"로부터 선택됨
limit: 오브젝트의 호출 번호
Return: 업데이트 연산이 성공적(참)인지 실패(거짓)인지의 여부
bool UpdateServiceExpiration (string sid, string expiration)
서비스의 만료 시간 변경하기
sid: 서비스 참조 ID, 세부 사항들을 위해서는 insert, insertLimit및 insertLimit2 의 리턴 값 참고하기
bool UpdateServiceHeld (string sid)
서비스의 모드를 "held"로 변경하기
sid: 서비스 참조 ID, 세부 사항들을 위해서는 insert, insertLimit및 insertLimit2 의 리턴 값 참고하기
bool UpdateServiceLimit (string sid, int limit)
서비스의 호출 제한 변경하기
sid: 서비스 참조 ID, 세부 사항들을 위해서는 insert, insertLimit및 insertLimit2 의 리턴 값 참고하기

도면5b

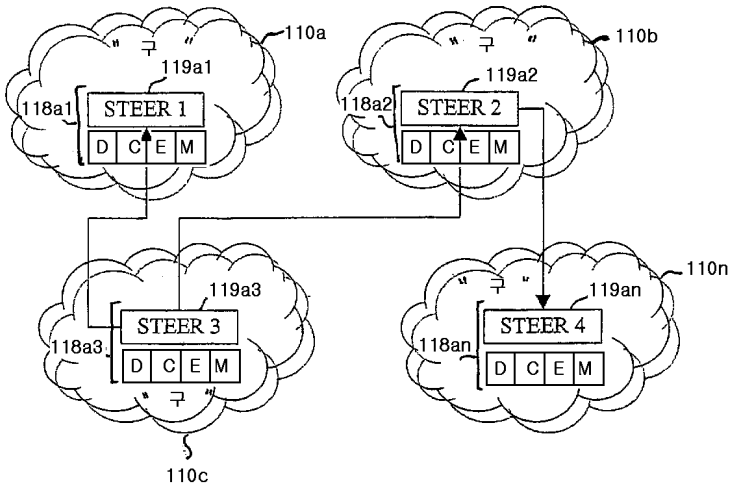

122
bool UpdateServiceLocal (string sid)
서비스의 모드를 "local"로 변경하기
sid: 서비스 참조 ID, 세부 사항들을 위해서는 insert, insertLimit 및 insertLimit2의 리턴 값 참고하기
bool UpdateServicePervasive (string sid)
서비스의 모드를 "pervasive"로 변경하기
sid: 서비스 참조 ID, 세부 사항들을 위해서는 insert, insertLimit 및 insertLimit2의 리턴 값 참고하기

도면6

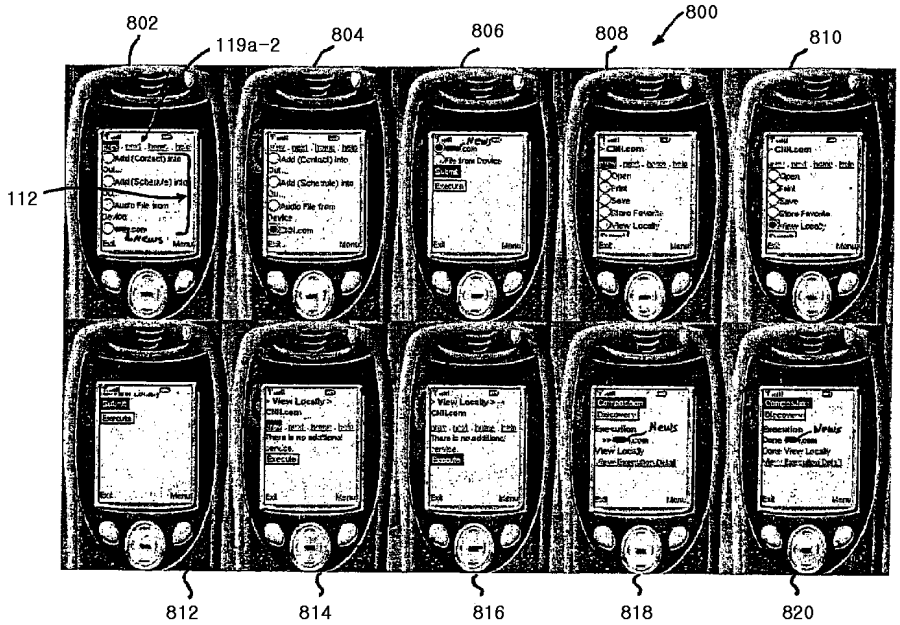


콜백 기반의 교차-환경 서비스 발견

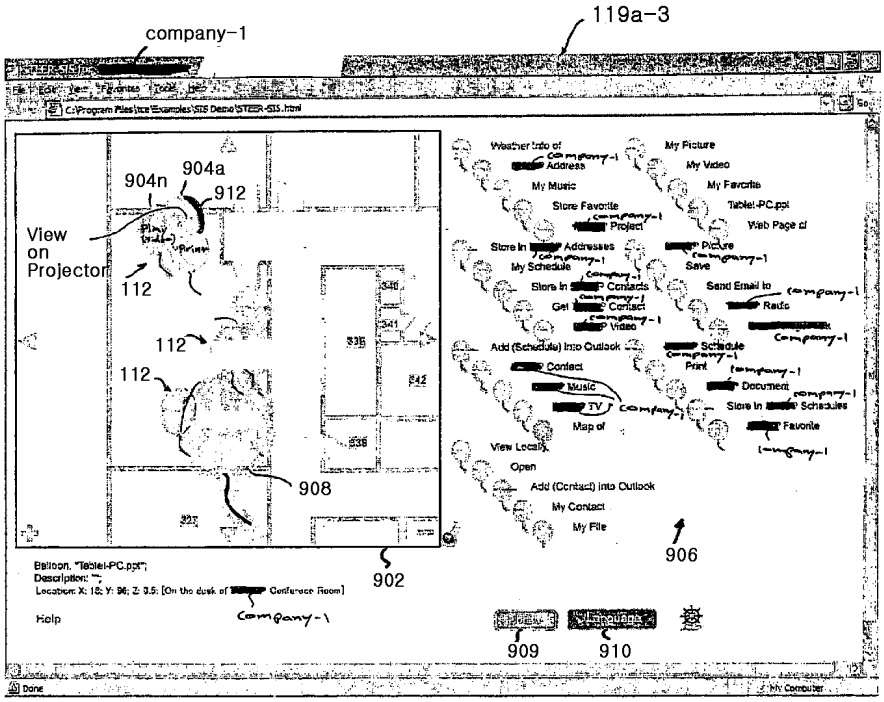
도면7



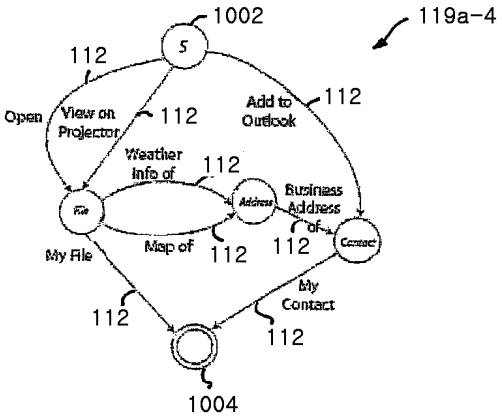
도면8



도면9


company-1
119a-3
904n
904a
912
View on Projector
112
112
112
908
902
906
909
910

도면10

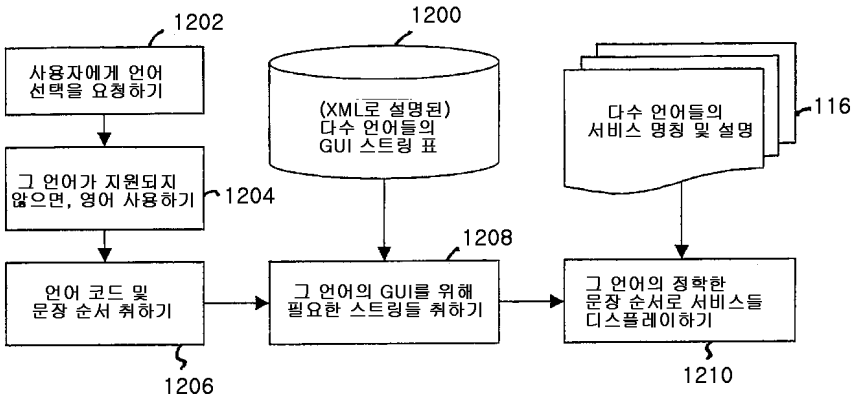

1002
119a-4
S
112
Open
View on Projector
112
Add to Outlook
112
Weather Info of
112
File
Address
Business Address of
Contact
My File
Map of
112
112
112
My Contact
112
112
1004

도면11

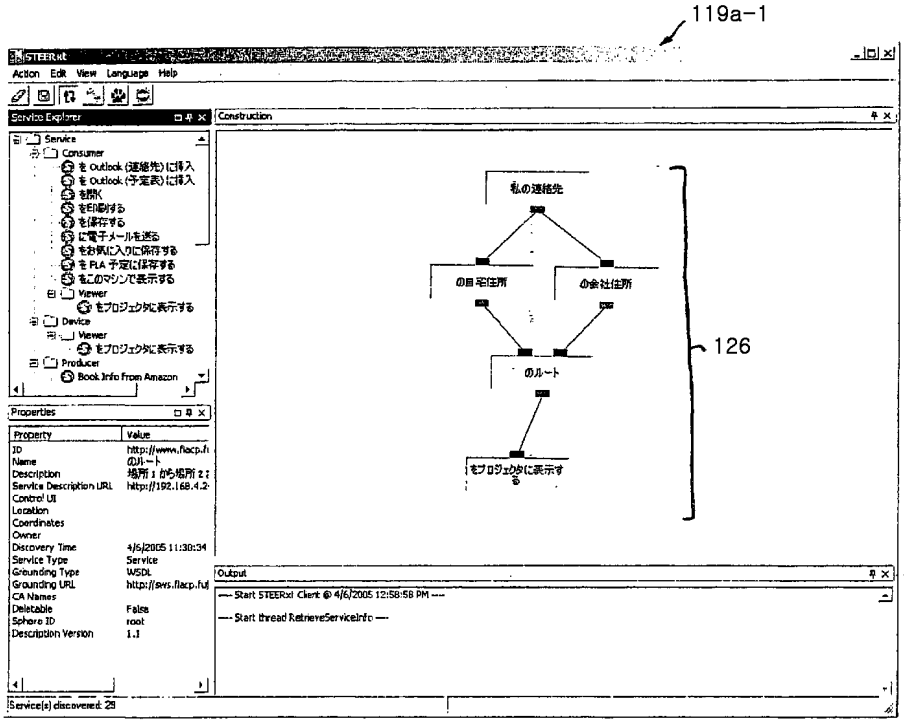
```
// Main code to build a diagram                                    1102
serviceList = findAllServicesWithNoOutput ();
startNode = new Node ();
endNode = new Node ();
foreach (service in serviceList)                                   1104
      buildDiagram (startNode, service);

// function buildDiagram
function buildDialgram (Node node, Service service)
{
      Edge e = new Edge (service);
      e.startNode = node;
      if (service has not input)
      {
            e.endNode = endNode;                                   1106
            return;
      }
      check if there is any node that represent service input
      if (no such node exists)
            create a new node called n
      else
            n is such a node

      serviceList = findAllMatchedService (service);
      foreach (s in serviceList)
            buildDiagram (n, s); // recursive call the
function
}
```

음성 인식 상태도를 구축하는 샘플 코드

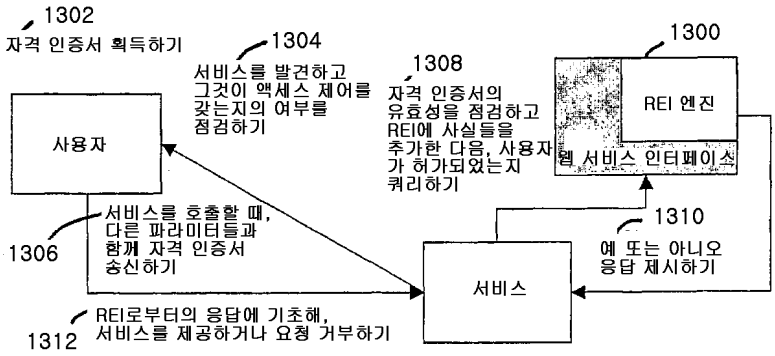
도면12a



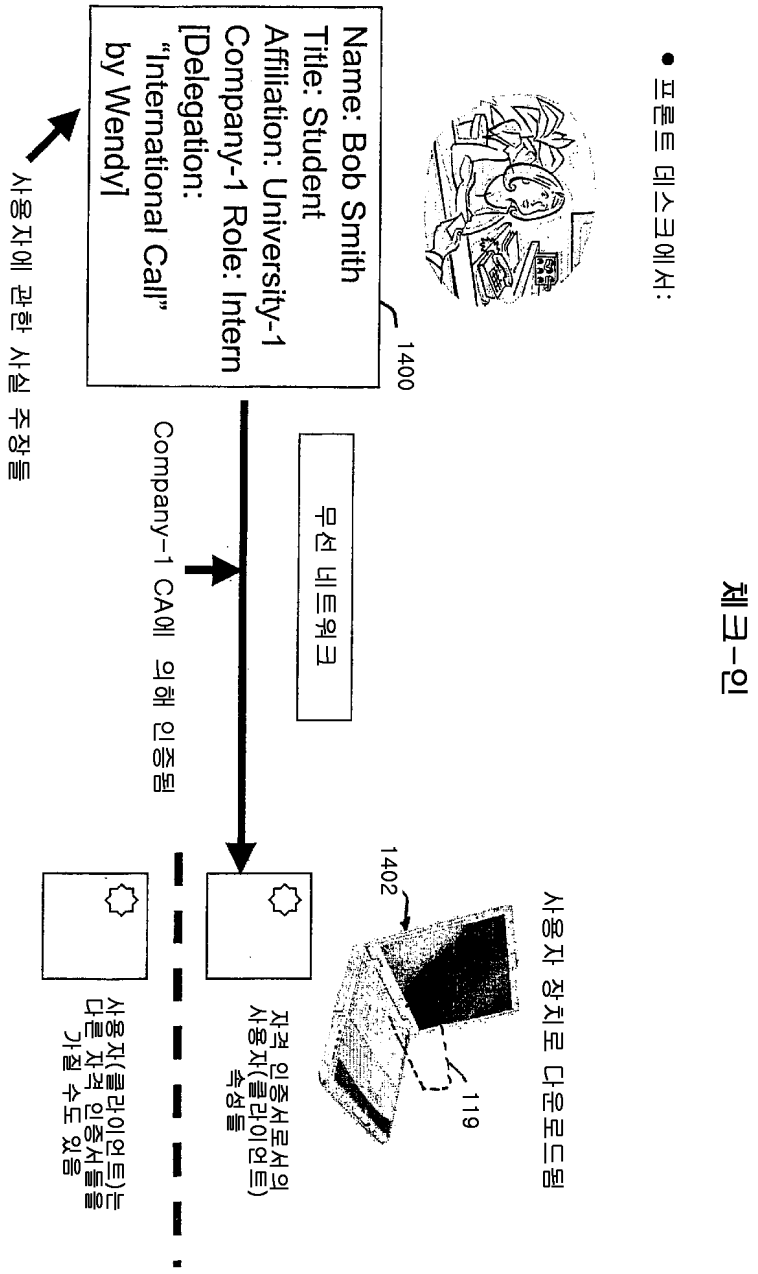
도면12b


119a-1
126

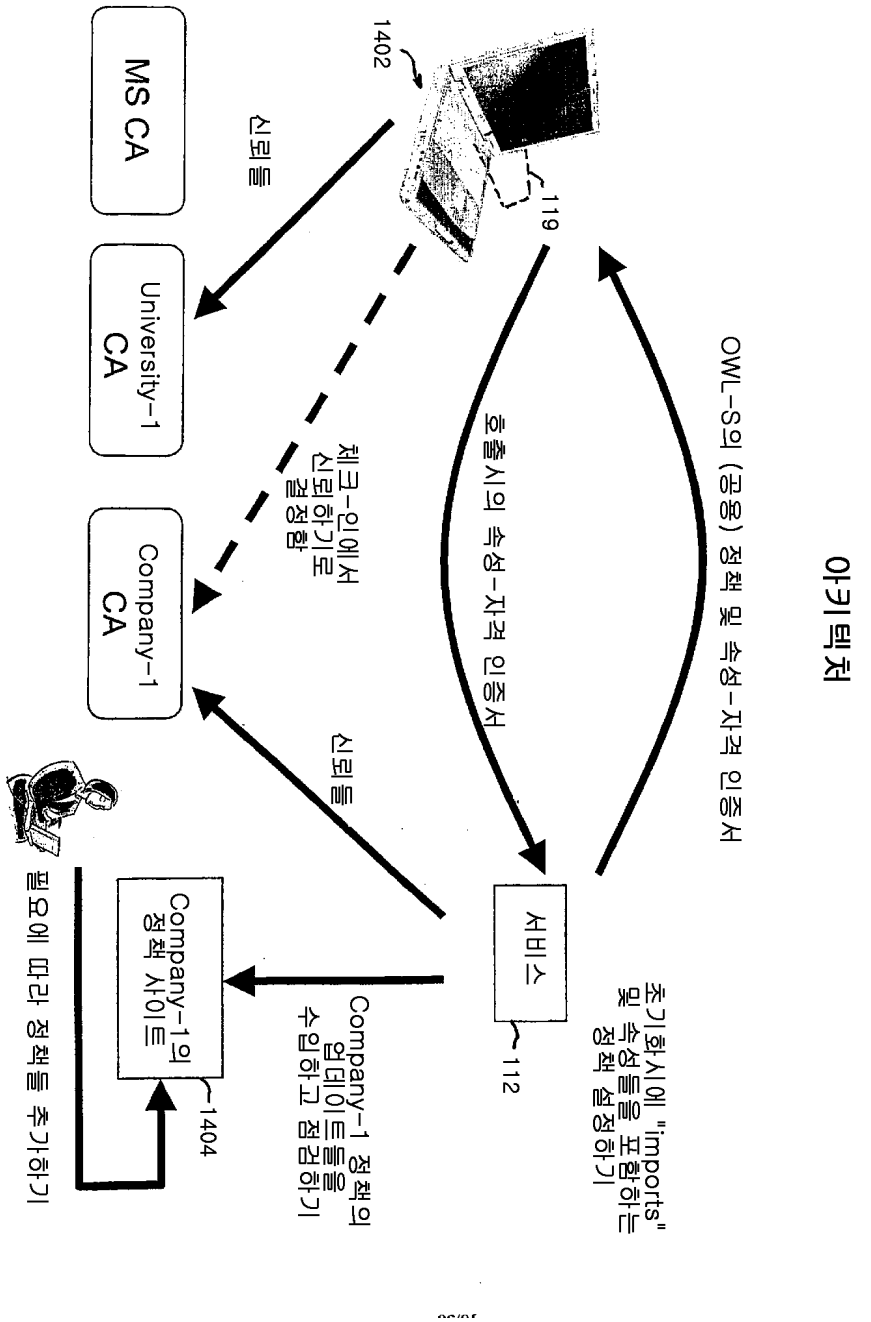
도면13


1302
자격 인증서 획득하기
1304
서비스를 발견하고 그것이 액세스 제어를 갖는지의 여부를 점검하기
1308
자격 인증서의 유효성을 점검하고 REI에 사실들을 추가한 다음, 사용자가 허가되었는지 쿼리하기
1300
REI 엔진
웹 서비스 인터페이스
사용자
서비스를 호출할 때, 다른 파라미터들과 함께 자격 인증서 송신하기
1306
1310
예 또는 아니오 응답 제시하기
서비스
REI로부터의 응답에 기초해, 서비스를 제공하거나 요청 거부하기
1312

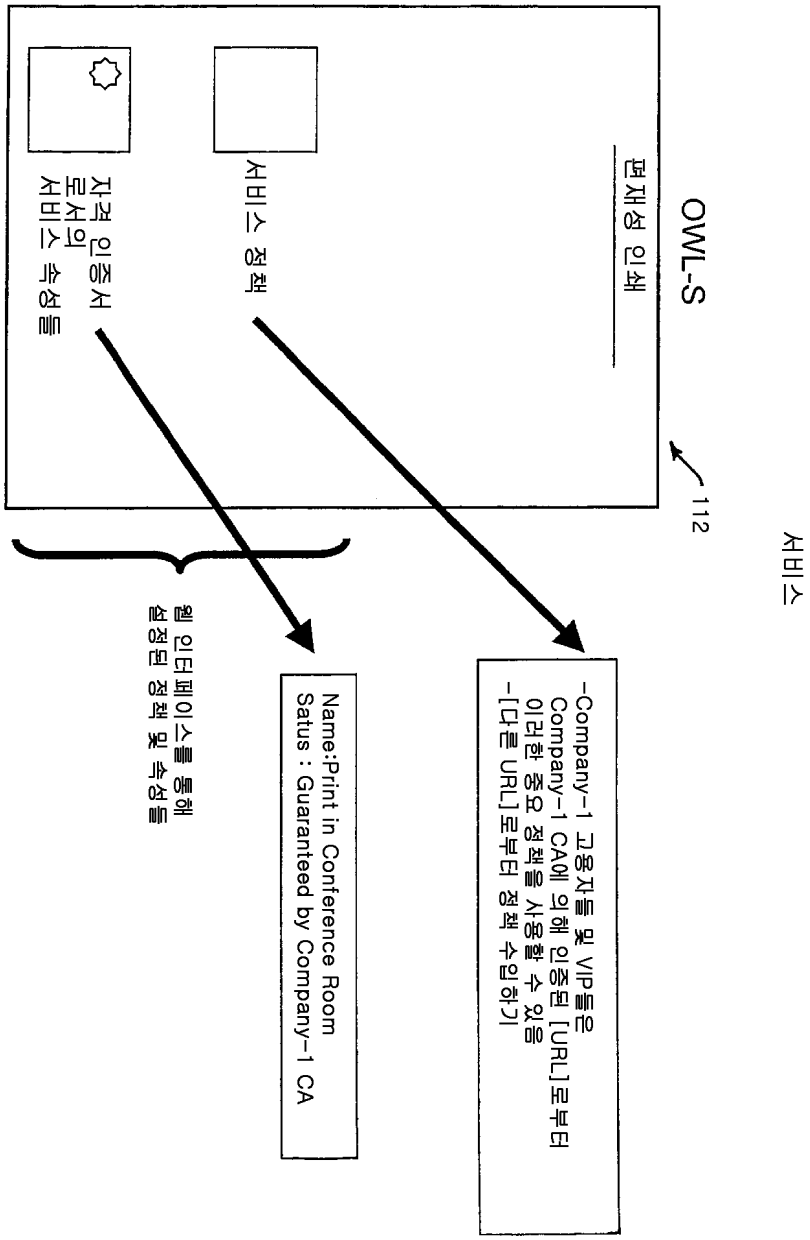
도면14a


체크-인
● 프론트 데스크에서:
Name: Bob Smith
Title: Student
Affiliation: University-1
Company-1 Role: Intern
[Delegation:
"International Call"
by Wendy]
1400
사용자에 관한 사실 주장들
무선 네트워크
Company-1 CA에 의해 인증됨
사용자 장치로 다운로드됨
1402
119
자격 인증서로서의
사용자(클라이언트)
속성들
사용자(클라이언트)는
다른 자격 인증서들을
가질 수도 있음

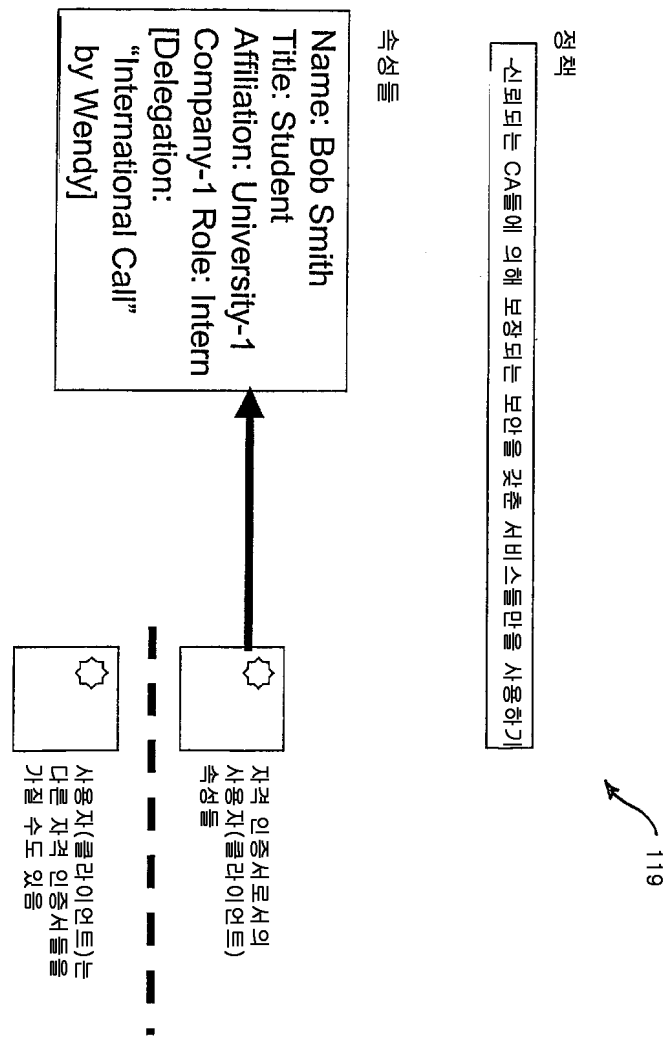
도면14b


아키텍처
OWL-S의 (공용) 정책 및 속성-자격 인증서
초기화시에 "imports" 및 속성들을 포함하는 정책 설정하기
서비스
112
호출시의 속성-자격 인증서
119
1402
신뢰들
MS CA
University-1 CA
Company-1 CA
체크-인에서 신뢰하기로 결정함
신뢰들
Company-1 정책의 업데이트들을 수입하고 점검하기
Company-1의 정책 사이트
1404
필요에 따라 정책들 추가하기

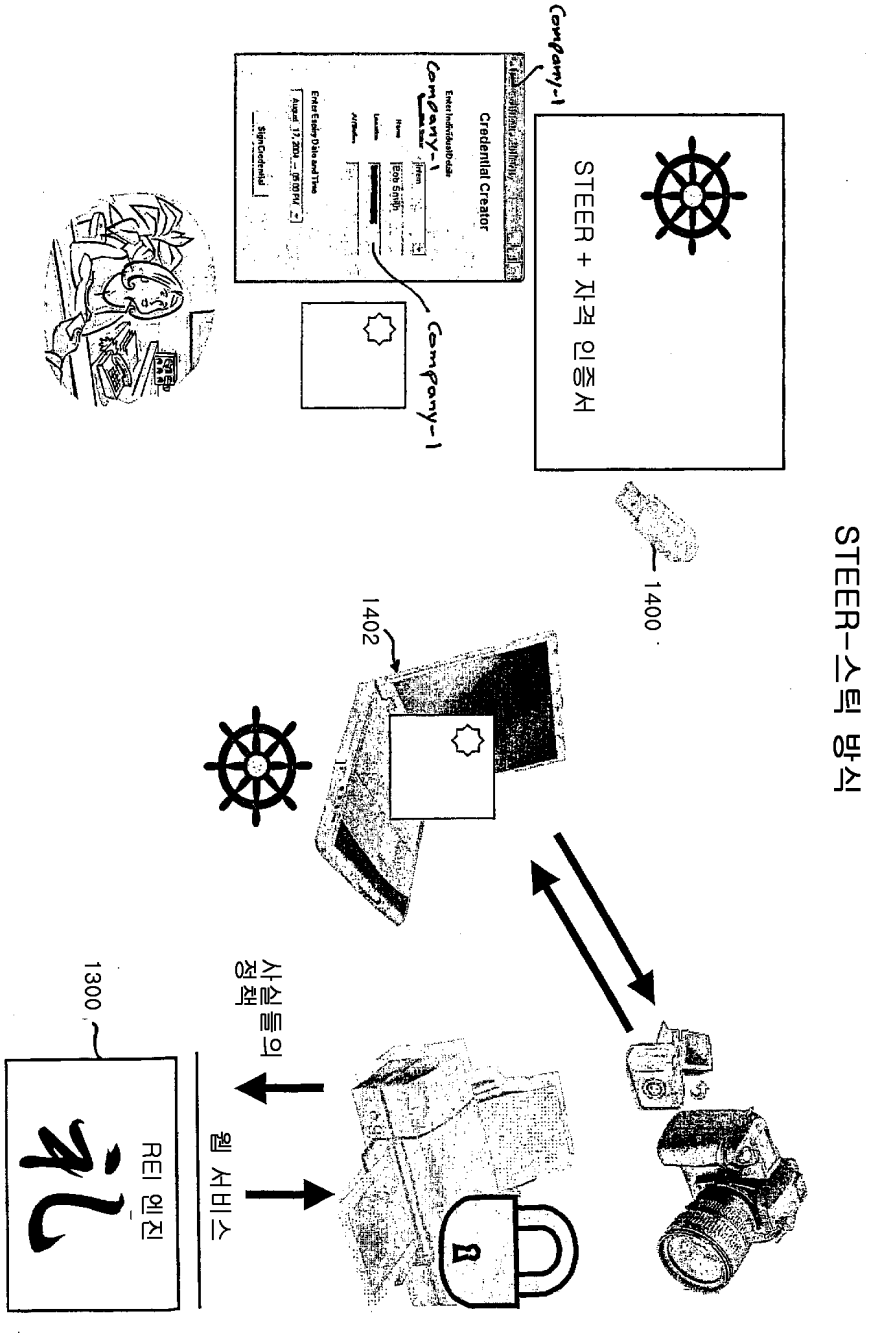
도면14c


서비스
OWL-S
112
편재성 인쇄
서비스 정책
자격 인증서로서의 서비스 속성들
웹 인터페이스를 통해 설정된 정책 및 속성들
-Company-1 고용자들 및 VIP들은 Company-1 CA에 의해 인증된 [URL]로부터 이러한 중요 정책을 사용할 수 있음
-[다른 URL]로부터 정책 수입하기
Name:Print in Conference Room
Satus : Guaranteed by Company-1 CA

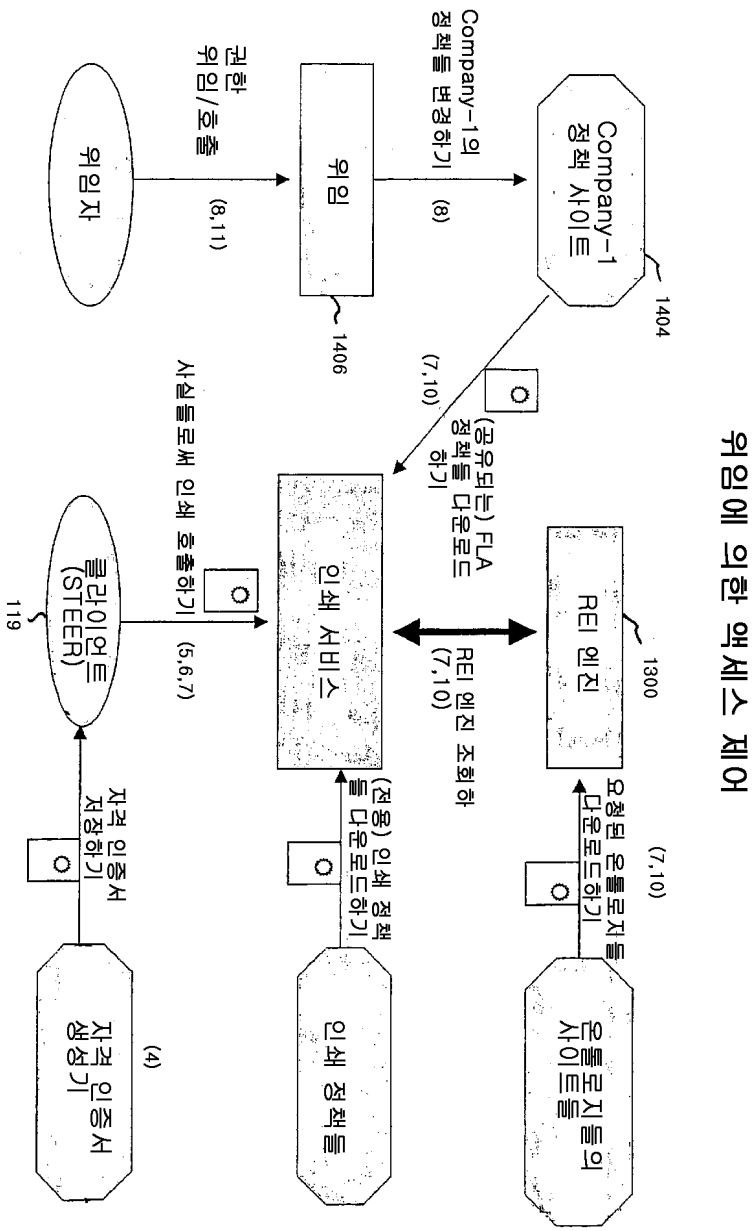
도면14d


클라이언트
119
정책
신뢰되는 CA들에 의해 보장되는 보안을 갖춘 서비스들만을 사용하기
속성들
Name: Bob Smith
Title: Student
Affiliation: University-1
Company-1 Role: Intern
[Delegation:
"International Call"
by Wendy]
자격 인증서로서의 사용자(클라이언트) 속성들
사용자(클라이언트)는 다른 자격 인증서들을 가질 수도 있음

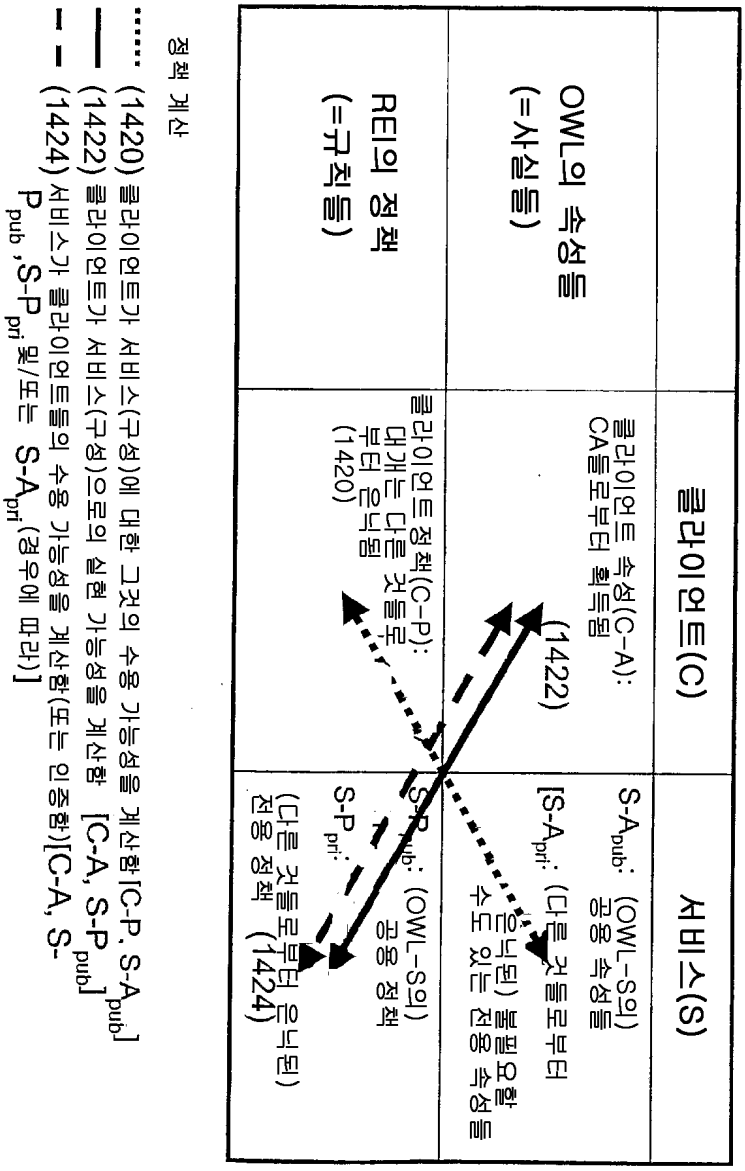
도면14e


STEER-스틱 방식
STEER + 자격 인증서
Company-1
Credential Creator
Company-1
1400
1402
사실들의 정책
웹 서비스
1300
REI 엔진

도면14f


위임에 의한 액세스 제어
Company-1 정책 사이트
1404
위임
1406
위임자
권한 위임/호출
(8,11)
Company-1의 정책들 변경하기
(8)
(공유되는) FLA 정책들 다운로드 하기
(7,10)
사실들로써 인쇄 호출하기
(5,6,7)
REI 엔진
1300
REI 엔진 조회하기
(7,10)
요청된 온톨로지들 다운로드하기
(7,10)
인쇄 서비스
클라이언트 (STEER)
119
자격 인증서 저장하기
(전용) 인쇄 정책들 다운로드하기
온톨로지들의 사이트들
인쇄 정책들
자격 인증서 생성기
(4)

도면14g



정책 계산

..... (1420) 클라이언트가 서비스(구성)에 대한 그것의 수용 가능성을 계산함[C-P, S-A_{pub}]

—— (1422) 클라이언트가 서비스(구성)으로의 실현 가능성을 계산함 [C-A, S-P_{pub}]

– – (1424) 서비스가 클라이언트들의 수용 가능성을 계산함(또는 인증함)[C-A, S-P_{pub}, S-P_{pri} 및/또는 S-A_{pri},(경우에 따라)]

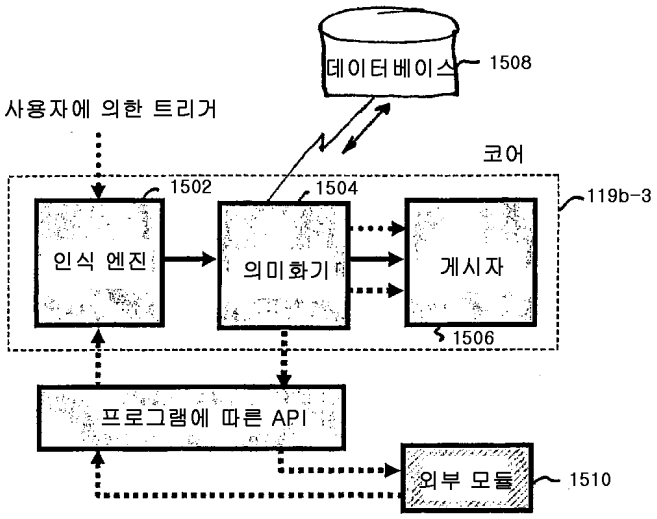
도면14h

1412

```
<!- Fact from Task Computing client -->
<rdf:RDF ...>
  <rdfs:label lang=en>Bob Smith</rdfs:label>
  <company1:Name  ...>Bob Smith</company1:Name>
  <company1:Expiry ...>2004-08-23T23:05:28Z</company1:Expiry>
  <company1:Status ...>&company1;Company1Visitor</company1:Status>
  <company1:Affiliation ...>University-1</company1:Affiliation>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      ...
    </SignedInfo>
    <SignatureValue>ZrbEVA7JWWGNbpqc...Jo6dDw=</SignatureValue>
  </Signature>
</rdf:RDF>

<!- Printer Private Policy -->
...
<deontic:Permission rdf:about="&company1policy;right_to_be_printed_on"
  policy:desc="All senior employees have the right to print">
  <deontic:actor rdf:resource="&company1policy;var1"/>
  <deontic:action rdf:resource="&company1policy;printing_in_conference"/>
  <deontic:constraint rdf:resource="&company1policy;preOrSenior"/>
</deontic:Permission>
...

<!- Delegation Inserted (and Removed) in Shared Policy-->
<action:Delegation
  rdf:ID="Delegation2004-08-23T19:32:19ZJohn">
  <action:sender rdf:resource="&inst;John"/>
  <action:receiver rdf:resource="&inst;BobSmith"/>
  <action:content>
    <deontic:Permission>
      <deontic:action rdf:resource="&inst;ASeniorEmployeePrintingAction"/>
    </deontic:Permission>
  </action:content>
</action:Delegation>
```

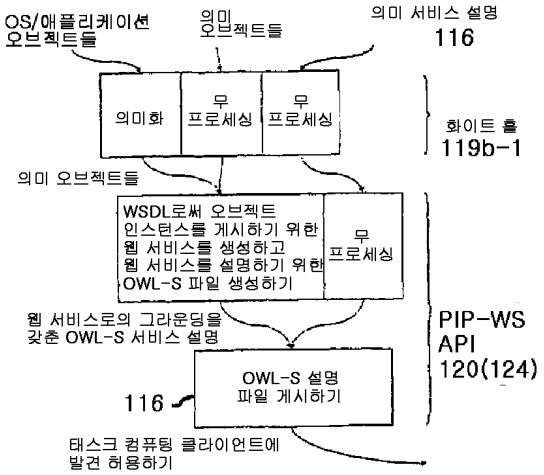
FACT, PRIVATE POLICY, AND SHARED POLICY

도면15



물리적 실사 오브젝트 의미화기: 아키텍처

도면16a



오브젝트들 및 서비스들의 의미화, 서비스화 및 게시의 절차

도면16b

1550

입력	OS/애플리케이션 오브젝트	OWL의 의미 오브젝트 또는 그것으로의 URL	OWL-S의 SSD 또는 그것으로의 URL
펑크션	오브젝트 유형 식별하기	의미 오브젝트 생성하기	OWL 파일 획득하기 또는 다운로드하기
출력	OWL의 의미 오브젝트		OWL-S의 SSD

화이트 홀 애플리케이션의 펑크션

도면16c

1555

입력	OWL의 의미 오브젝트	OWL-S의 SSD
펑크션	(OWL의) 의미 오브젝트를 제공하는 웹 서비스 생성하기; 서비스를 위한 OWL-S 파일 생성하기; 발견 범위에 걸쳐 그것을 이용 가능하게 하기	발견 범위내에서 OWL-S 파일을 이용 가능하게 하기
출력	게시되는 의미 서비스	

PIPE 애플리케이션의 펑크션

도면16d

1570

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE uridef[
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-
s/1.1/Grounding.owl">
  <!ENTITY company1
"http://www.company1.com/tce/ontologies/2005/01/service.owl">
  <!ENTITY obj
"http://www.company1.com/tce/ontologies/2004/03/object.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
]>
<rdf:RDF
  xmlns:owl="&owl;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:rdf="&rdf;#"
  xmlns:service="&service;#"
  xmlns:process="&process;#"
  xmlns:profile="&profile;#"
  xmlns:grounding="&grounding;#"
  xmlns:company1="&company1;#"
  xml:base="http://www.company1.com/tce/services/Company1Document.owl#"
>
      <owl:Ontology rdf:about="">
          <owl:imports rdf:resource="&obj;"/>
            <owl:imports rdf:resource="&service;" />
            <owl:imports rdf:resource="&profile;" />
            <owl:imports rdf:resource="&process;" />
            <owl:imports rdf:resource="&grounding;" />
          <owl:imports rdf:resource="&company1;"/>
      </owl:Ontology>

<!-- Service description -->
<service:Service rdf:ID="FileProviderService">
  <service:presents rdf:resource="#FileProviderProfile" />
  <service:describedBy rdf:resource="#FileProvider" />
  <service:supports rdf:resource="#FileProviderGrounding" />
</service:Service>
<!-- Profile description -->
<company1:ProducerProfile rdf:ID="FileProviderProfile">
  <service:isPresentedBy rdf:resource="#FileProviderService"/>

  <profile:serviceName xml:lang="en">Company1 Document
</profile:serviceName>
  <profile:serviceName xml:lang="ja">Company1 文書
  <profile:serviceName xml:lang="zh-cn">Company1 文件
  <profile:serviceName xml:lang="es">Documentos Company1</profile:serviceName>
  <profile:serviceName xml:lang="ko">Company1 서류들</profile:serviceName>
 <profile:serviceName xml:lang="zh-tw">Company1 檔案庫</profile:serviceName>
 <profile:serviceName xml:lang="tr">Company1 Belgeleri</profile:serviceName>
 <profile:serviceName xml:lang="hi">Company1 दोकुमेंट</profile:serviceName>
 <profile:serviceName xml:lang="el">Εγγραφο Company1</profile:serviceName>
```

도면16e

1570

```
 <profile:textDescription xml:lang="en">Lets you select a document from the Company1
document folder</profile:textDescription>
 <profile:textDescription xml:lang="ja">Company1 の文書フォルダから文書を選びます。
</profile:textDescription>
 <profile:textDescription xml:lang="zh-cn">让你从Company1的文件列表中选取一个文件。
</profile:textDescription>
 <profile:textDescription xml:lang="es">Permite seleccionar un documento de la carpeta de
documentos de Company1</profile:textDescription>
 <profile:textDescription xml:lang="ko">Company1 서류들중에서 선택하기
</profile:textDescription>
 <profile:textDescription xml:lang="zh-tw">讓您自美國富士通實驗室之檔案資料夾中選擇您需要的
檔案</profile:textDescription>
 <profile:textDescription xml:lang="tr">Company1 belge klasörinden bir belge seçmenizi sağlar
</profile:textDescription>
 <profile:textDescription xml:lang="hi">दोकुमेंट फोल्दर से एफ एल ए सि पि दोकुमेंट चून सकते है
</profile:textDescription>
 <profile:textDescription xml:lang="el">Σας δίνει την δυνατότητα να επιλέξετε ένα έγγραφο από
τον φάκελο εγγράφων του Company1</profile:textDescription>

 <company1:hasServiceControlUI>
  <company1:ServiceControlUI>
   <profile:sParameter rdf:resource="http://sws.company1.com/DirectoryPublishUI/Select.aspx%
3fname%3dCompany1+Documents%26dir%3dC%253a%255cMy%2bPublished%2bDirectories%
255cMy%2bDocuments%26type%3dFile" />
  </company1:ServiceControlUI>
 </company1:hasServiceControlUI>

 <profile:hasOutput rdf:resource="#FileOutput" />
</company1:ProducerProfile>

<process:AtomicProcess rdf:ID="FileProvider">
 <process:hasOutput rdf:resource="#FileOutput" />
 <process:hasInput rdf:resource="#wsdlServiceId" />
</process:AtomicProcess>

<process:Input rdf:ID="wsdlServiceId">
 <process:parameterType rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
 <company1:useRandomInput>true</company1:useRandomInput>
</process:Input>

<process:Output rdf:ID="FileOutput">
 <process:parameterType rdf:datatype="&xsd;#anyURI">&obj;#File</process:parameterType>
</process:Output>
<!-- Grounding description -->
<grounding:WsdlGrounding rdf:ID="FileProviderGrounding">
```

도면16f

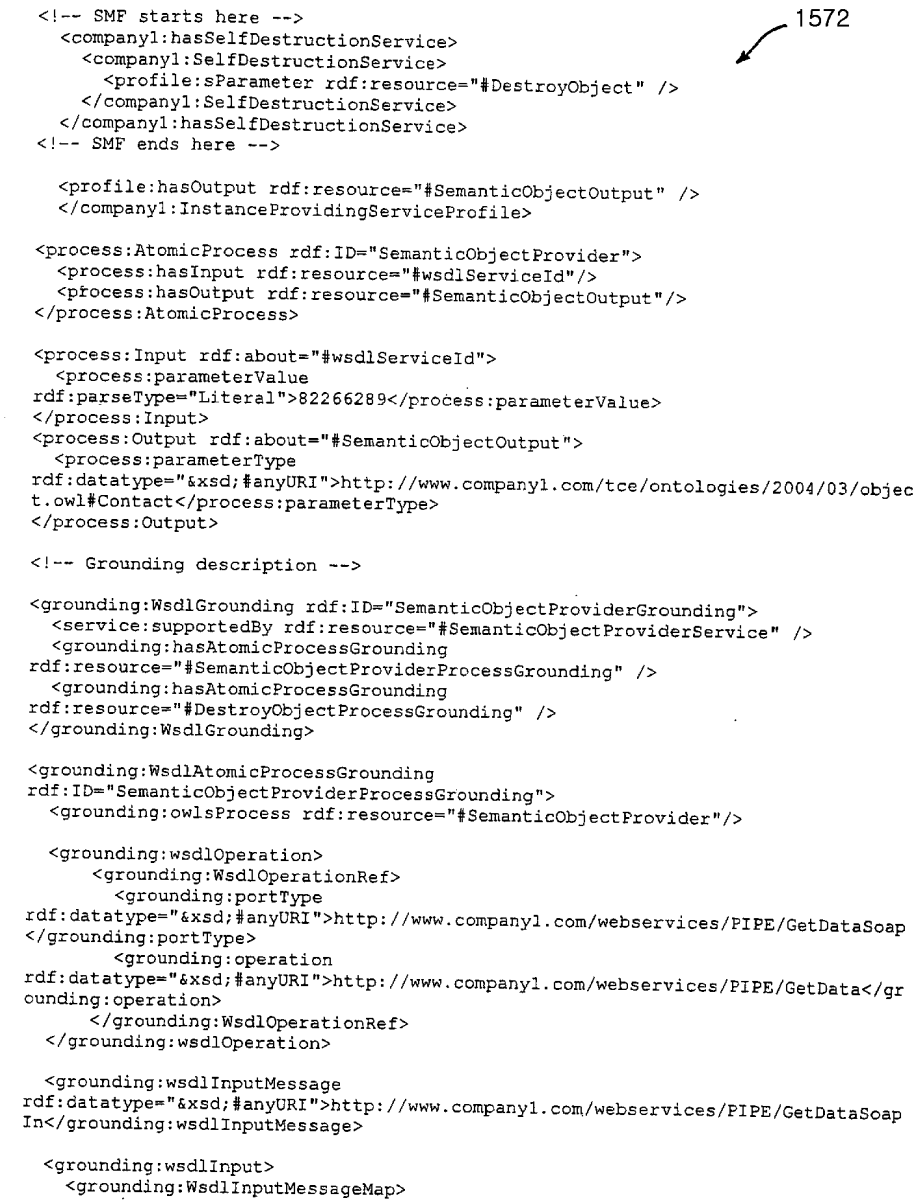
1570

```
 <service:supportedBy rdf:resource="#FileProviderService" />
 <grounding:hasAtomicProcessGrounding rdf:resource="#FileProviderProcessGrounding" />
</grounding:WsdlGrounding>

<grounding:WsdlAtomicProcessGrounding rdf:ID="FileProviderProcessGrounding">
 <grounding:owlsProcess rdf:resource="#FileProvider"/>

 <grounding:wsdlOperation>
  <grounding:WsdlOperationRef>
   <grounding:portType rdf:datatype="&xsd;#anyURI">
http://www.company1.com/tce/services/DirectoryPublishService/GetURLSoap
</grounding:portType>
   <grounding:operation rdf:datatype="&xsd;#anyURI">
http://www.company1.com/tce/services/DirectoryPublishService/GetURL</grounding:operation>
  </grounding:WsdlOperationRef>
 </grounding:wsdlOperation>

 <grounding:wsdlInputMessage rdf:datatype="&xsd;#anyURI">
http://www.company1.com/tce/services/DirectoryPublishService/GetURLSoapIn
</grounding:wsdlInputMessage>

 <grounding:wsdlInput>
  <grounding:WsdlInputMessageMap>
   <grounding:owlsParameter rdf:resource="#wsdlServiceId" />
   <grounding:wsdlMessagePart rdf:datatype="&xsd;#anyURI">
http://www.company1.com/tce/services/DirectoryPublishService/sid
</grounding:wsdlMessagePart>
  </grounding:WsdlInputMessageMap>
 </grounding:wsdlInput>

 <grounding:wsdlOutputMessage rdf:datatype="&xsd;#anyURI">
http://www.company1.com/tce/services/DirectoryPublishService/GetURLSoapOut
</grounding:wsdlOutputMessage>

 <grounding:wsdlOutput>
  <grounding:WsdlOutputMessageMap>
   <grounding:owlsParameter rdf:resource="#FileOutput"/>
   <grounding:wsdlMessagePart rdf:datatype="&xsd;#anyURI">
http://www.company1.com/tce/services/DirectoryPublishService/GetURLResult
</grounding:wsdlMessagePart>
  </grounding:WsdlOutputMessageMap>
 </grounding:wsdlOutput>

 <grounding:wsdlDocument rdf:datatype="&xsd;#anyURI">
http://sws.company1.com/DirectoryPublishService/Service.asmx?wsdl
</grounding:wsdlDocument>
</grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

도면16g


1572
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE uridef[
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
<!ENTITY owl "http://www.w3.org/2002/07/owl">
<!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
<!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
<!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
<!ENTITY grounding "http://www.daml.org/services/owl-s/1.1/Grounding.owl">
<!ENTITY company1
"http://www.company1.com/tce/ontologies/2005/01/service.owl">
<!ENTITY obj "http://www.company1.com/tce/ontologies/2004/03/object.owl">
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
]>
<rdf:RDF
xmlns:owl="&owl;#"
xmlns:rdfs="&rdfs;#"
xmlns:rdf="&rdf;#"
xmlns:service="&service;#"
xmlns:process="&process;#"
xmlns:profile="&profile;#"
xmlns:grounding="&grounding;#"
xmlns:company1="&company1;#"
xml:base="http://192.168.4.26/UPnPFile/82266289_SemanticObjectBroadcastingServic
eDesc.owl"
>
<owl:Ontology rdf:about="">
<owl:imports rdf:resource="&obj;"/>
<owl:imports rdf:resource="&service;" />
<owl:imports rdf:resource="&profile;" />
<owl:imports rdf:resource="&process;" />
<owl:imports rdf:resource="&grounding;" />
<owl:imports rdf:resource="&company1;"/>
</owl:Ontology>
<!-- Service description -->
<service:Service rdf:ID="SemanticObjectProviderService">
<service:presents rdf:resource="#SemanticObjectProviderProfile" />
<service:describedBy rdf:resource="#SemanticObjectProvider" />
<service:supports rdf:resource="#SemanticObjectProviderGrounding" />
</service:Service>
<!-- Profile description -->
<company1:InstanceProvidingServiceProfile
rdf:ID="SemanticObjectProviderProfile">
<service:isPresentedBy rdf:resource="#SemanticObjectProviderService" />
<profile:serviceName><![CDATA[Bob Smith]]></profile:serviceName>
<profile:textDescription><![CDATA[Object published by
PIPE]]></profile:textDescription>

도면16h

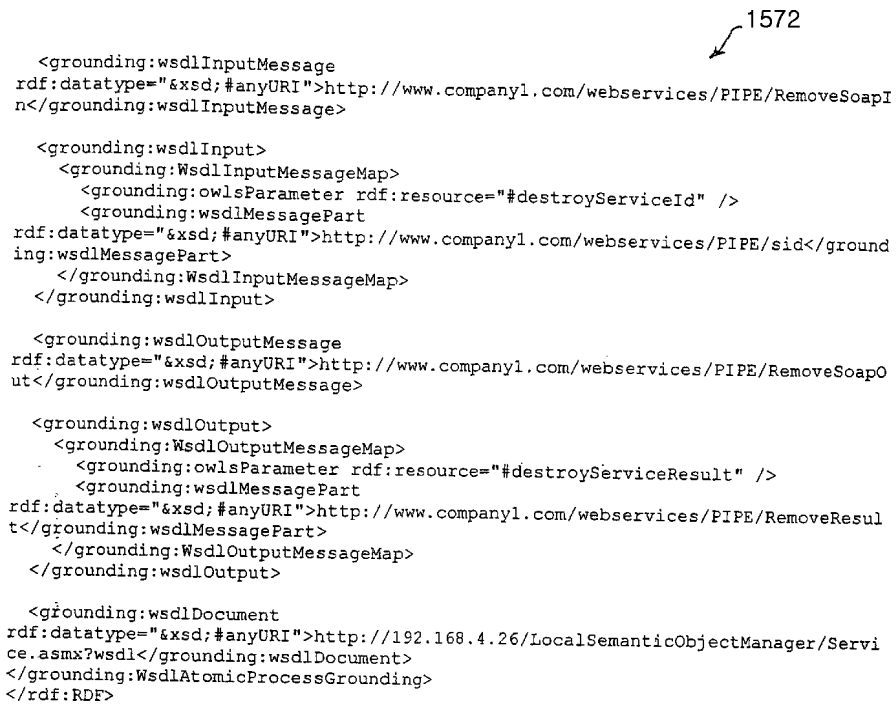

1572
<!-- SMF starts here -->
<company1:hasSelfDestructionService>
<company1:SelfDestructionService>
<profile:sParameter rdf:resource="#DestroyObject" />
</company1:SelfDestructionService>
</company1:hasSelfDestructionService>
<!-- SMF ends here -->
<profile:hasOutput rdf:resource="#SemanticObjectOutput" />
</company1:InstanceProvidingServiceProfile>
<process:AtomicProcess rdf:ID="SemanticObjectProvider">
<process:hasInput rdf:resource="#wsdlServiceId"/>
<process:hasOutput rdf:resource="#SemanticObjectOutput"/>
</process:AtomicProcess>
<process:Input rdf:about="#wsdlServiceId">
<process:parameterValue
rdf:parseType="Literal">82266289</process:parameterValue>
</process:Input>
<process:Output rdf:about="#SemanticObjectOutput">
<process:parameterType
rdf:datatype="&xsd;#anyURI">http://www.company1.com/tce/ontologies/2004/03/objec
t.owl#Contact</process:parameterType>
</process:Output>
<!-- Grounding description -->
<grounding:WsdlGrounding rdf:ID="SemanticObjectProviderGrounding">
<service:supportedBy rdf:resource="#SemanticObjectProviderService" />
<grounding:hasAtomicProcessGrounding
rdf:resource="#SemanticObjectProviderProcessGrounding" />
<grounding:hasAtomicProcessGrounding
rdf:resource="#DestroyObjectProcessGrounding" />
</grounding:WsdlGrounding>
<grounding:WsdlAtomicProcessGrounding
rdf:ID="SemanticObjectProviderProcessGrounding">
<grounding:owlsProcess rdf:resource="#SemanticObjectProvider"/>
<grounding:wsdlOperation>
<grounding:WsdlOperationRef>
<grounding:portType
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/GetDataSoap
</grounding:portType>
<grounding:operation
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/GetData</gr
ounding:operation>
</grounding:WsdlOperationRef>
</grounding:wsdlOperation>
<grounding:wsdlInputMessage
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/GetDataSoap
In</grounding:wsdlInputMessage>
<grounding:wsdlInput>
<grounding:WsdlInputMessageMap>

도면16i

1572

```
  <grounding:wsdlInput>
    <grounding:WsdlInputMessageMap>
      <grounding:owlsParameter rdf:resource="#wsdlServiceId" />
      <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/serviceId</
grounding:wsdlMessagePart>
    </grounding:WsdlInputMessageMap>
  </grounding:wsdlInput>

  <grounding:wsdlOutputMessage
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/GetDataSoap
Out</grounding:wsdlOutputMessage>

  <grounding:wsdlOutput>
    <grounding:WsdlOutputMessageMap>
      <grounding:owlsParameter rdf:resource="#SemanticObjectOutput" />
      <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/GetDataResu
lt</grounding:wsdlMessagePart>
    </grounding:WsdlOutputMessageMap>
  </grounding:wsdlOutput>

  <grounding:wsdlDocument
rdf:datatype="&xsd;#anyURI">http://192.168.4.26/LocalSemanticObjectManager/Servi
ce.asmx?wsdl</grounding:wsdlDocument>
</grounding:WsdlAtomicProcessGrounding>

<!-- Process for self-destruction function -->
<process:AtomicProcess rdf:ID="DestroyObject">
  <process:hasInput rdf:resource="#destroyServiceId" />
  <process:hasOutput rdf:resource="#destroyServiceResult"/>
</process:AtomicProcess>

<process:Input rdf:about="#destroyServiceId">
  <process:parameterValue
rdf:parseType="Literal">82266289</process:parameterValue>
</process:Input>

<process:Output rdf:about="#destroyServiceResult">
  <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
</process:Output>

<!-- Grounding for self-destruction function -->
<grounding:WsdlAtomicProcessGrounding rdf:ID="DestroyObjectProcessGrounding">
  <grounding:wsdlOperation>
    <grounding:WsdlOperationRef>
      <grounding:portType
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/RemoveSoap<
/grounding:portType>
      <grounding:operation
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/Remove</gro
unding:operation>
    </grounding:WsdlOperationRef>
  </grounding:wsdlOperation>

  <grounding:owlsProcess rdf:resource="#DestroyObject" />
```

도면16j


1572
<grounding:wsdlInputMessage
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/RemoveSoapIn</grounding:wsdlInputMessage>
<grounding:wsdlInput>
<grounding:WsdlInputMessageMap>
<grounding:owlsParameter rdf:resource="#destroyServiceId" />
<grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/sid</grounding:wsdlMessagePart>
</grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
<grounding:wsdlOutputMessage
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/RemoveSoapOut</grounding:wsdlOutputMessage>
<grounding:wsdlOutput>
<grounding:WsdlOutputMessageMap>
<grounding:owlsParameter rdf:resource="#destroyServiceResult" />
<grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/RemoveResult</grounding:wsdlMessagePart>
</grounding:WsdlOutputMessageMap>
</grounding:wsdlOutput>
<grounding:wsdlDocument
rdf:datatype="&xsd;#anyURI">http://192.168.4.26/LocalSemanticObjectManager/Service.asmx?wsdl</grounding:wsdlDocument>
</grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>

도면16k

1574

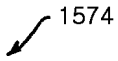
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE uridef[
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-s/1.1/Grounding.owl">
  <!ENTITY company1
"http://www.company1.com/tce/ontologies/2005/01/service.owl">
  <!ENTITY obj "http://www.company1.com/tce/ontologies/2004/03/object.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
]>

<rdf:RDF
  xmlns:owl="&owl;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:rdf="&rdf;#"
  xmlns:service="&service;#"
  xmlns:process="&process;#"
  xmlns:profile="&profile;#"
  xmlns:grounding="&grounding;#"
  xmlns:company1="&company1;#"

xml:base="http://192.168.4.26/UPnPFile/995622400_SemanticObjectBroadcastingServi
ceDesc.owl"
>

<owl:Ontology rdf:about="">
          <owl:imports rdf:resource="&obj;"/>
            <owl:imports rdf:resource="&service;" />
            <owl:imports rdf:resource="&profile;" />
            <owl:imports rdf:resource="&process;" />
            <owl:imports rdf:resource="&grounding;" />
          <owl:imports rdf:resource="&company1;"/>
  <owl:imports
rdf:resource="http://www.company1.com/tce/ontologies/2004/03/project.owl"/>

</owl:Ontology>

<!-- Service description -->

<service:Service rdf:ID="SemanticObjectProviderService">
  <service:presents rdf:resource="#SemanticObjectProviderProfile" />
  <service:describedBy rdf:resource="#SemanticObjectProvider" />
  <service:supports rdf:resource="#SemanticObjectProviderGrounding" />
</service:Service>

<!-- Profile description -->

  <company1:InstanceProvidingServiceProfile
rdf:ID="SemanticObjectProviderProfile">
    <service:isPresentedBy rdf:resource="#SemanticObjectProviderService" />

  <profile:serviceName><![CDATA[XYZ Project]]></profile:serviceName>
```

도면16l

```
  <profile:textDescription><![CDATA[Object published by
PIPE]]></profile:textDescription>

  <company1:hasSelfDestructionService>
    <company1:SelfDestructionService>
      <profile:sParameter rdf:resource="#DestroyObject" />
    </company1:SelfDestructionService>
  </company1:hasSelfDestructionService>

  <profile:hasOutput rdf:resource="#SemanticObjectOutput" />
  </company1:InstanceProvidingServiceProfile>

<process:AtomicProcess rdf:ID="SemanticObjectProvider">
  <process:hasInput rdf:resource="#wsdlServiceId"/>
  <process:hasOutput rdf:resource="#SemanticObjectOutput"/>
</process:AtomicProcess>

<process:Input rdf:about="#wsdlServiceId">
  <process:parameterValue
rdf:parseType="Literal">995622400</process:parameterValue>
</process:Input>
<process:Output rdf:about="#SemanticObjectOutput">
  <process:parameterType
rdf:datatype="&xsd;#anyURI">http://www.company1.com/tce/ontologies/2004/03/proje
ct.owl#Project</process:parameterType>
</process:Output>

<!-- Grounding description -->

<grounding:WsdlGrounding rdf:ID="SemanticObjectProviderGrounding">
  <service:supportedBy rdf:resource="#SemanticObjectProviderService" />
  <grounding:hasAtomicProcessGrounding
rdf:resource="#SemanticObjectProviderProcessGrounding" />
  <grounding:hasAtomicProcessGrounding
rdf:resource="#DestroyObjectProcessGrounding" />
</grounding:WsdlGrounding>

<grounding:WsdlAtomicProcessGrounding
rdf:ID="SemanticObjectProviderProcessGrounding">
  <grounding:owlsProcess rdf:resource="#SemanticObjectProvider"/>

  <grounding:wsdlOperation>
      <grounding:WsdlOperationRef>
        <grounding:portType
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/GetDataSoap
</grounding:portType>
        <grounding:operation
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/GetData</gr
ounding:operation>
      </grounding:WsdlOperationRef>
  </grounding:wsdlOperation>

  <grounding:wsdlInputMessage
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/GetDataSoap
In</grounding:wsdlInputMessage>

  <grounding:wsdlInput>
```

1574

도면16m




```
    <grounding:WsdlInputMessageMap>
      <grounding:owlsParameter rdf:resource="#wsdlServiceId" />
      <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/serviceId</
grounding:wsdlMessagePart>
    </grounding:WsdlInputMessageMap>
  </grounding:wsdlInput>

  <grounding:wsdlOutputMessage
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/GetDataSoap
Out</grounding:wsdlOutputMessage>

  <grounding:wsdlOutput>
    <grounding:WsdlOutputMessageMap>
      <grounding:owlsParameter rdf:resource="#SemanticObjectOutput" />
      <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/GetDataResu
lt</grounding:wsdlMessagePart>
    </grounding:WsdlOutputMessageMap>
  </grounding:wsdlOutput>

  <grounding:wsdlDocument
rdf:datatype="&xsd;#anyURI">http://192.168.4.26/LocalSemanticObjectManager/Servi
ce.asmx?wsdl</grounding:wsdlDocument>
</grounding:WsdlAtomicProcessGrounding>

<!-- Process for self-destruction function -->
<process:AtomicProcess rdf:ID="DestroyObject">
  <process:hasInput rdf:resource="#destroyServiceId" />
  <process:hasOutput rdf:resource="#destroyServiceResult"/>
</process:AtomicProcess>

<process:Input rdf:about="#destroyServiceId">
  <process:parameterValue
rdf:parseType="Literal">995622400</process:parameterValue>
</process:Input>

<process:Output rdf:about="#destroyServiceResult">
  <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
</process:Output>

<!-- Grounding for self-destruction function -->
<grounding:WsdlAtomicProcessGrounding rdf:ID="DestroyObjectProcessGrounding">
  <grounding:wsdlOperation>
    <grounding:WsdlOperationRef>
      <grounding:portType
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/RemoveSoap<
/grounding:portType>
      <grounding:operation
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/Remove</gro
unding:operation>
    </grounding:WsdlOperationRef>
  </grounding:wsdlOperation>

  <grounding:owlsProcess rdf:resource="#DestroyObject" />
```

도면16n

1574

```
  <grounding:wsdlInputMessage
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/RemoveSoapI
n</grounding:wsdlInputMessage>

  <grounding:wsdlInput>
    <grounding:WsdlInputMessageMap>
      <grounding:owlsParameter rdf:resource="#destroyServiceId" />
      <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/sid</ground
ing:wsdlMessagePart>
    </grounding:WsdlInputMessageMap>
  </grounding:wsdlInput>

  <grounding:wsdlOutputMessage
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/RemoveSoapO
ut</grounding:wsdlOutputMessage>

  <grounding:wsdlOutput>
    <grounding:WsdlOutputMessageMap>
      <grounding:owlsParameter rdf:resource="#destroyServiceResult" />
      <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">http://www.company1.com/webservices/PIPE/RemoveResul
t</grounding:wsdlMessagePart>
    </grounding:WsdlOutputMessageMap>
  </grounding:wsdlOutput>

  <grounding:wsdlDocument
rdf:datatype="&xsd;#anyURI">http://192.168.4.26/LocalSemanticObjectManager/Servi
ce.asmx?wsdl</grounding:wsdlDocument>
</grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

도면17