(19)대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl.
G06F 9/44 (2006.01)

(11) 공개번호 10-2006-0044313
(43) 공개일자 2006년05월16일

(21) 출원번호 10-2005-0020607
(22) 출원일자 2005년03월11일

(30) 우선권주장 10/799,463 2004년03월12일 미국(US)

(71) 출원인 마이크로소프트 코포레이션
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이

(72) 발명자 미니엄 데니스 더블유
미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내
하피조귈라리 오잔
미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내
푸 성지안
미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내

(74) 대리인 주성민
백만기
이중희

심사청구 : 없음

(54) 비통합 어플리케이션들의 인터페이스를 용이하게 하는시스템

요약

비통합 어플리케이션들의 인터페이스를 용이하게 하는 아키텍처가 제공된다. 개시된 아키텍처는 이전에는 통합되도록 설계되지 않았던 툴들 간의 통합을 가능하게하는 데 사용되는 규약 및 API의 세트를 포함한다. 이는 툴의 서버 기반 파트너 통합 및 클라이언트 통합에 대한 기반을 제공하여, 제3자 에코시스템의 구성에 대한 설립을 용이하게 할 수 있다. 그를 지원하여, 아티펙트 제공자 API는 제1 어플리케이션의 아티펙트를 노출시키고, 아티펙트 소비자 API는 제2 어플리케이션의 참조를 노출시키며, 참조는 아티펙트 제공자의 아티펙트 중 적어도 하나로의 링크와 연관된다.

대표도

도 1

색인어

비통합 어플리케이션, 툴, 아티펙트

명세서

도면의 간단한 설명

도 1은 본 발명의 시스템을 나타낸 도면.

도 2는 본 발명에 따른 프로세스의 흐름도.

도 3은 소비자와 제공자의 링크 패턴, 참조 및 피참조 아티펙트 결합을 나타낸 도면.

도 4는 아티펙트 의존성의 표시를 용이하게 하는 사용자 인터페이스(UI)를 나타낸 도면.

도 5는 본 발명에 따른 GAP API의 일 구현의 블럭도.

도 6은 GAP를 사용하는 비통합 툴을 준비하는 일 구현의 흐름도.

도 7은 GAP 및 캐시 모두에 대해 사용되는 일반 아티펙트 데이터베이스 스키마(GADS)를 도시.

도 8은 본 발명에 따른 GAP를 구현하는 한 방법의 흐름도.

도 9는 본 발명에 따른 GAP 어댑터를 구현하는 한 방법의 흐름도.

도 10은 본 발명에 따른 비통합 툴에 액세스하는 것을 용이하게 하는 시스템을 나타낸 도면.

도 11은 본 발명에 따른 캐시의 일 구현에 대한 프로세스의 흐름도.

도 12는 개시된 아키텍처를 실행할 수 있는 컴퓨터의 블럭도.

도 13은 본 발명에 따른 예시적 컴퓨팅 환경의 개요도.

<도면의 주요부분에 대한 부호의 설명>

302: 아티펙트 제공자

304: 아티펙트 소비자

504: 일반 아티펙트 제공자

508: 툴-특정 GAP 어댑터

1020: 링크 관리자 API

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 어플리케이션 프로그램 인터페이스 및 그것의 사용에 관한 것이다.

통상적으로, 함께 사용될 때 보다 나은 응집성 및 매우 향상된 사용자 경험을 제공할 수 있는 자립형 소프트웨어 툴이 있다. 이러한 환경은 제3의 벤더 및 다른 사람들에게 어플리케이션 및 문서 개발을 강화하는 툴을 제공하는 수단을 제공한다.

발명이 이루고자 하는 기술적 과제

비통합 툴들의 특징에 대한 액세스를 허용하는 링크 아키텍처가 필요하다.

발명의 구성 및 작용

이하에서는, 본 발명의 일부 양상에 대한 기본 이해를 제공한다. 이러한 개요는 본 발명의 광범위한 개관은 아니다. 이는 본 발명의 핵심/중요 구성요소를 식별하거나 본 발명의 범위를 서술하도록 의도된 것은 아니다. 이것의 유일한 목표는, 후술되는 보다 더 상세한 설명에 대한 서문으로서 본 발명의 일부 개념을 단순화된 형태로 나타내는 것이다.

여기에 개시되고 청구된 본 발명은, 그것의 일 양상에서, 비통합 어플리케이션, 서비스 또는 툴의 통합을 용이하게 하는 아키텍처를 포함한다. 개시된 통합 서비스(IS)는 느슨하게 결합되고 개별적으로 설계된 툴이 기술과 규약의 조합을 통해 간단한 데이터 통합을 제공하는 것을 허용하는 경량 아키텍처이다. 따라서, 이것은 리포지토리 없이도 일반적인 리포지토리에 의해 제공되는 핵심적인 이점의 일부를 제공한다. 예를 들어, IS는 소스 제어 및 결함 추적을 포함하며, 각각은 다른 원리에 의해 개별적으로 설계된다. 각각의 툴이 공개한 주요 항목[예를 들어, 그들의 아티펙트(artifact)]을 식별하기 위한 표준 방식을 도입함으로써, 예를 들어, 각각이 툴이 서로에 대한 상세한 지식을 갖지 않아도, 결함들이 소스 제어를 포인트하는 것(그 역도 성립)이 가능하다. 게다가, 제3자가 새로운 툴을 그들 자신의 특수화된 링크와 함께 IS에 도입하여, IS 툴에 대한 어떠한 수정 없이도 경량 데이터를 통합할 수 있게 된다.

통합되도록 아키텍처되지 않았던 툴들 사이의 통합을 가능하게 하는데 사용되는 규약 및 어플리케이션 프로그래밍 인터페이스(API)의 세트가 개시된다. 이 해결법은 참여자들에게 적은 비용 및 커미트로도 리포지토리의 이점의 상당 부분를 제공한다. 이것은 개발 툴을 위한 서버 기반의 파트너-통합 및 클라이언트-통합에 대한 아키텍처를 제공한다.

그를 지원하여, 하나 이상의 개별 어플리케이션의 아티펙트를 노출시키는 아티펙트 제공자 및 아티펙트 제공자의 아티펙트로의 링크와 연관된 참조를 노출시키는 아티펙트 소비자를 포함하는 시스템을 제공한다. 링크 컴포넌트는 제공자 아티펙트와 소비자 참조의 링크을 용이하게 한다.

본 발명의 다른 양상에서, 아티펙트와 아티펙트 참조를 캐싱하고 링크하는 링크 캐시가 제공된다.

본 발명의 또다른 양상에서, 아티펙트간 참조를 나타내는 것을 용이하게 하는 사용자 인터페이스가 제공된다.

본 발명의 또다른 양상에서, 적어도 각각의 툴로 캐시 동기화를 용이하게 하는 링크 관리자가 제공된다.

그것의 다른 양상에서, 일반 아티펙트 제공자(GAP)가 제공되며, 여기서 아티펙트 제공자 및 소비자로부터의 모든 메소드들은 GAP에 의해 구현된다. 일반 아티펙트 스키마-포맷의 아티펙트는 GAP 데이터베이스로 푸시(push)된다.

상술한 것 및 관련 목적의 달성을 위해, 본 발명의 예시적인 양상들이 상세한 설명 및 첨부된 도면과 관련되어 여기에 설명된다. 이러한 양상은 직설적이지만, 본 발명의 원리가 사용될 수 있는 다양한 방식 중 몇몇이 사용될 수 있으며, 본 발명은 모든 이러한 양상 및 그들의 등가물을 포함하도록 의도된다. 본 발명의 다른 이점 및 새로운 특징들은 도면을 참조하여 본 발명의 상세한 설명을 숙지함으로써 명백해질 것이다.

이하에서는, 도면을 참조하여 본 발명이 설명되며, 이 도면들에서 동일한 참조 번호는 동일한 구성요소를 지칭하는 데 사용된다. 이하에서는, 설명의 목적으로, 수많은 구체적 세부사항들이 본 발명의 완전한 이해를 제공하도록 설명된다. 그러나, 본 발명이 이러한 구체적 세부사항들 없이 구현될 수 있음이 분명하다. 다른 예에서, 공지된 구조 및 디바이스들은 블럭도 형태로 도시되어, 본 발명의 설명을 용이하게 한다.

본 명세서에서, "컴포넌트" 및 "시스템"이라는 용어는 하드웨어, 하드웨어와 소프트웨어의 조합, 소프트웨어, 또는 실행중인 소프트웨어와 같은 컴퓨터 관련 엔티티를 지칭하기 위한 것이다. 예를 들어, 컴포넌트는 프로세서 상에서 실행되는 프로세스, 프로세서, 객체, 실행가능물, 실행 스레드, 프로그램 및/또는 컴퓨터일 수 있지만, 이에 한정된 것은 아니다. 예를 들어, 서버 상에서 실행중인 어플리케이션과 서버 둘다 컴포넌트일 수 있다. 하나 이상의 컴포넌트가 프로세스 및/또는 실행 스레드에 상주할 수 있으며, 컴포넌트는 하나의 컴퓨터에 국부화될 수 있고/있거나 2개 이상의 컴퓨터 간에 분산될 수 있다.

본 명세서에서, "추론하다" 또는 "추론"이라는 용어는, 일반적으로 이벤트 및/또는 데이터를 통해 캡처된 관측값의 세트에 의해 시스템, 환경 및/또는 사용자의 상태를 추론 또는 추리하는 프로세스를 의미한다. 예를 들어, 추론은 특정 문맥 또는 액션을 식별하는 데 사용될 수 있거나, 상태들 간에서의 확률 분포를 생성할 수 있다. 추론은 확률적일 수 있다. 즉, 데이터 및 이벤트의 고려에 기초하여, 관심있는 상태들 간의 확률 분포를 계산하는 것일 수 있다.. 추론은 이벤트 및/또는 데이터의 세트로 보다 더 높은 레벨의 이벤트를 구성하는 데 사용되는 기술을 의미할 수 있다. 이러한 추론은, 이벤트들이 시간적으로 근접하게 상관되어 있는지에 상관없이, 또한 이벤트 및 데이터가 하나 또는 여러개의 이벤트 및 데이터 소스로부터 온 것인지에 상관없이, 관측된 이벤트 및/또는 저장된 이벤트 데이터의 세트로부터 새로운 이벤트 또는 액션을 구성한다.

정의

다음의 용어들은 상세한 설명 전체에서 사용되며, 본 발명의 다양한 양상의 이해를 돕기 위해 그 정의가 제공된다.

아티펙트(artifact): 하나의 툴이 공개적으로 노출시켜, 다른 툴들이 그에 대한 포인터를 홀딩할 수 있게 하는 데이터의 임의의 항목. 각각의 아티펙트는 소정의 아티펙트 유형을 가져야 한다. 각각의 아티펙트는 고유한 불변의 식별자에 의해 식별되어야 한다. 예를 들어, 소스 파일 foo.cs, 작업 항목 125, 빌드 20041205.3으로부터의 테스트 결과가 있다.

아티펙트 소비자(artifact consumer): 일반적으로는, 자신이 홀딩하는 참조를, 아티펙트 제공자에 의해 호스트되는 아티펙트에 노출시키는 툴 또는 서비스. 보다 엄격하게는, 자신이 홀딩하는 참조를 아티펙트 제공자에 의해 호스트되는 아티펙트에 노출시키는 서비스. 예를 들어, C의 작업 항목(그 자체가 아티펙트임)은 자신이 프로세싱될 때 확인되었던 P의 변화 집합에 대한 참조를 홀딩할 수 있다. 이 경우에, C는 아티펙트 소비자이고, P는 아티펙트 제공자이다. 아티펙트 소비자는 소정의 표준 거동을 제공해야 한다. 2개의 아티펙트 간에는 정의에 의한 링크가 존재하므로, 아티펙트 소비자가 아티펙트 제공자이기도 해야 한다는 점에 유의해야 한다.

아티펙트 식별자: 특정 아티펙트에 대한 고유한 불변의 식별자의 객체화된 버전(objectified version)(ArtifactUri). 아티펙트 식별자의 속성(property)은 URI에 대한 규칙을 따른다.

아티펙트 제공자(artifact provider): 일반적으로는, 아티펙트 소비자가 링크할 수 있는 아티펙트를 호스트하고 노출시키는 툴 또는 서비스. 보다 엄격하게는, 아티펙트를 노출시키는 서비스. 예를 들어, C는 유형 결함(type defect), 요구조건(requirement) 및 태스크의 아티펙트를 웹 서비스를 통해 노출시킴으로써, 아티펙트 제공자로 된다. 아티펙트 제공자는 소정의 표준 거동을 제공해야 한다. 2개의 아티펙트 간에는 정의에 의한 링크가 존재하므로, 다수의 아티펙트 제공자가 아티펙트 소비자로도 될 수 있음에 유의해야 한다.

아티펙트 프록시: 일반 아티펙트 제공자(Generic Artifact Provider, GAP)에 의해 저장되어, 비통합 서비스 인에이블 저장장치(non-integrated service enabled store)에 저장된 데이터를 나타내는 아티펙트.

아티펙트 유형: 툴이 공개적으로 노출시키는 데이터의 유형. 특정 아티펙트 유형의 아티펙트는 동일 유형의 다른 아티펙트들과의 링크 유형의 공통 집합을 갖는다. 예를 들어, 소스 파일, 결함, 요구조건, 테스트 결과 및 빌드 등이 포함되지만, 이에 한정되는 것은 아니다.

아티펙트 URI: 적형성(well formedness)에 대한 구체적 규칙을 따르는 URI(Uniform Resource Identifier). IS(Integration Services) Uri의 유사어.

통합 서비스(IS): 기술 및 규약의 조합을 통하여, 느슨하게 결합되고 개별적으로 설계된 비통합 툴들의 간단한 데이터 통합을 허용하는 개시된 경량의 아키텍처. IS 아키텍처에 의해 인에이블되지 않은 임의의 서비스 또는 툴은 "논(non) IS" 또는 비통합으로 칭해진다.

IS 네임스페이스: "논리적 서버"로 생각될 수 있는 IS 인에이블된 제품 및 아티펙트의 컬렉션에 대응하는 네임스페이스로서, 그 물리적 구성요소들은 다수의 물리적 서버에 분산될 수 있다. 툴은 설치시에 IS 네임스페이스에 자신을 등록하고, 모든 툴은 IS 위치표를 통해 액세스될 수 있다. 각각의 아티펙트는 하나의 IS 네임스페이스에 의해 소유된다.

IS Uri: 적형성에 대한 IS별 규칙을 따르는 URI. ArtifactUri의 유사어.

외부 식별자: IS GAP의 문맥에서 의미있음. 논 IS 인에이블된 툴이 아티펙트를 참조할 수 있게 하는 식별자를 참조.

일반 아티펙트 제공자(GAP): IS-GAP는, 아티펙트 프록시의 구성이 완전히 IS인에이블되지 않은 툴에 아티펙트 핸들링 거동을 전할 수 있게 하는 IS 제공된 설비이다.

일반 아티펙트 스키마(Generic Artifact Schema, GAS): 아티펙트들의 컬렉션을 서술하는 공통 스키마. 각각의 아티펙트는 확장된 속성, 및 자신이 홀딩하고 있는 다른 아티펙트에 대한 링크를 포함한다.

일반 링크 스키마(Generic Link Schema, GLS): 링크의 컬렉션을 서술하는 공통 스키마. 각각의 링크는 참조 및 피참조 ArtifactUri들과 링크 유형을 포함한다.

인바운드 링크(inbound link): 피참조 아티펙트(즉, 포인트된 아티펙트)의 관점으로부터의 링크. 참조 아티펙트로의 인바운드 링크는, 아티펙트 소비자에 의한 구현에 사용되는 표준 GetReferencingArtifacts 메소드를 사용하여 발견될 수 있다.

링크: 아티펙트를 포인트하기 위해 아티펙트 소비자에 의해 홀딩되는 아티펙트 식별자. 링크는 바이너리이고, 정확하게 2개의 아티펙트, 즉 참조 아티펙트 및 피참조 아티펙트와 관련된다. 툴(아티펙트 소비자)이 다른 아티펙트로의 아티펙트 식별자를 홀딩할 때, 링크가 형성된다. 각각의 링크는 소정의 링크 유형을 가져야 한다. 적합하게 형성된 링크는 참조 아티펙트와 피참조 아티펙트의 2개의 아티펙트 간의 링크임에 유의해야 한다. 아티펙트와 논-아티펙트 간의 포인터는 링크가 아니다. 예를 들어, "버그 125가 빌드 20041205.4에서 발견되었다"에서, "버그" 및 "빌드"는 아티펙트 유형이고, "에서 발견되었다"는 링크 유형이다.

링크 유형: 해당 유형의 링크의 목적을 서술. 유효한 링크가 구성될 수 있는 아티펙트들의 아티펙트 유형을 제한할 수 있음. 각각의 링크는 각 방향(아웃바운드 및 인바운드)으로부터 하나씩, 총 2개의 판독값(또는 역할)을 포함한다. 예를 들어, '버그 125가 빌드 20041205.4에서 발견되었다'와, '빌드 20041205.4가 버그 125를 노출시켰다'.

느슨한 결합(loose coupling): 링크의 문맥에서, 아티펙트 제공자가 임의의 아티펙트 소비자에 대한 지식을 가질 것을 요구하지 않는 거동의 품질. 예를 들어, 아티펙트 제공자가 IS 서비스간 역전 쿼리 툴(IS cross-service inversion query tool)을 사용하여 참조 역전 쿼리(reference inversion query)를 발행할 때, IS는 모든 아티펙트에 공통인 스키마에 따라서 참조들의 세트를 리턴한다.

아웃바운드(outbound) 링크: 참조 아티펙트(즉, 소정의 다른 아티펙트를 포인트하는 아티펙트)의 관점으로부터의 링크. 피참조 아티펙트로의 아웃바운드 링크는 아티펙트 제공자에 의한 구현에 요구되는 표준 GetArtifacts 메소드에 의해 리턴된다.

참조 역전(reference inversion)[역전 쿼리(inversion query)]: 일반적으로, 참조 역전은 어떤 항목이 특정 항목으로의 포인터를 홀딩하는지(즉, 특정 항목을 참조하는지)를 찾아내는 프로세스이다. IS에서, 어떤 참조 아티펙트가 특정 피참조 아티펙트로의 링크를 홀딩하는지를 찾아내는 프로세스이다.

피참조 아티펙트(referneced artifact): 링크의 문맥에서 사용됨. 피참조 아티펙트는, 링크에서 참조 아티펙트가 포인트하는 쪽이다. 피참조 아티펙트는 항상 아티펙트 제공자에 의해 호스트된다.

참조 아티펙트(referencing artifact): 링크의 문맥에서 사용됨. 참조 아티펙트는, 링크에서 피참조 아티펙트로의 참조를 홀딩하는 쪽이다. 참조 아티펙트는 아티펙트 소비자에 의해 호스트된다.

밀접한 결합(Tight coupling): 링크의 문맥에서, 아티펙트 소비자가 아티펙트 제공자에 관한 특정 지식을 가질 것을 요구하는 거동의 품질. 예를 들어, 제공자가 특수화된 참조 역전 메소드(특정한 필터를 포함하거나, 관심있는 소비자에게 특히 관심있는 특정 정보를 전달할 수 있음)를 제공하고, 소비자가 그 메소드를 활용하는 경우, 그 상호작용은 밀접하게 결합된 것으로 칭해진다.

특정 Id(tool-specific Id, TSId): 적합하게 형성된 IS Uri의 마지막 부분. 툴 인스턴스, 아티펙트 유형 및 특정 Id의 조합은 IS 네임스페이스에서 고유해야 한다. 또한, TSId는 불변이어야 한다.

이제 도 1을 참조하면, 본 발명의 시스템(100)이 도시된다. 이 설명의 목적을 위해, 단 2개의 비통합 툴(또는 서비스, 이하에서는 툴로 통칭함), 즉 제1 툴(102) 및 제2 툴(104)만이 제공된다. 그러나, 본 발명에 따른 하나 이상의 인터페이스를 사

용하여 상호작용할 수 있는 임의의 수의 비통합 툴이 있을 수 있음을 알 것이다. 본 구현예에서, 제1 툴(102)은 제공자로 지정되고, 제2 툴은 소비자로 지정된다. 양쪽 툴(102 및 104)이 비통합 툴이므로, 본 발명의 인터페이스는 이러한 툴(102 및 104)의 데이터의 노출을 통합 툴에 의해 수행될 수 있는 것처럼 용이하게 한다. 노출되는 데이터는 아티펙트, 및 이러한 아티펙트에 대한 참조의 형태이다. 그것을 지원하여, 아티펙트 제공자 인터페이스(106)(예를 들어, 어플리케이션 프로그래밍 인터페이스 API의 형태)는 제1 툴(102)의 하나 이상의 아티펙트(108)($ARTIFACT_1$, $ARTIFACT_2$,..., $ARTIFACT_N$으로 표시)를 노출하는 것을 용이하게 하는 제1 툴(102)로 구현된다. 제2 툴(104)은 아티펙트(108)의 소비자로 지정된다. 본 발명에 따르면, 제1 툴(102)의 하나 이상의 아티펙트(108)를 참조할 수 있는 제2 툴(104)의 하나 이상의 참조(112)($REFERENCE_1$, $REFERENCE_2$,..., $REFERENCE_M$으로 표시)를 노출시키는 아티펙트 소비자 인터페이스(110)가 제공된다.

제1 툴(102)의 아티펙트(108)는 링크 컴포넌트(114)에 의해 제2 툴(104)의 참조(112)와 연관된다. 링크 컴포넌트는 소비자 인터페이스(110)의 일부분일 수 있지만, 이것이 필수적인 것은 아니다. 링크 컴포넌트(114)는 적어도 하나의 아티펙트(108)를 포인트하는 아티펙트 식별자를 포함하는 링크를 정의한다. 링크는 소정의 링크 유형을 갖고, 2개의 아티펙트, 즉 참조 아티펙트 및 피참조 아티펙트와 관련된다는 점에서 바이너리이다.

따라서, 개시된 아키텍처는, 이전에는 통합되도록 설계되지 않았던 툴들 간의 통합을 가능하게 하는 데에 사용되는 규약 및 API의 세트를 포함한다. 이는 툴의 서버 기반 파트너 통합 및 클라이언트 통합에 대한 기초를 제공하고, 제3자 에코시스템을 구축하기 위한 기반을 용이하게 할 수 있다.

이제 도 2를 참조하면, 본 발명에 따른 프로세스의 흐름도가 도시된다. 본 명세서에서 예를 들어 흐름도 형태로 도시된 하나 이상의 방법론은 설명을 간단하게 하기 위해 일련의 동작들로서 도시되고 설명되지만, 본 발명은 동작들의 순서에 제한되지 않으며, 본 발명을 따르면 일부 동작들은 다른 순서로 및/또는 여기에 도시되고 설명된 다른 동작들과 동시에 발생할 수 있음을 알아야 한다. 예를 들어, 본 기술에 숙련된 자들은, 방법론이 다르게는 상태도에서와 같이 일련의 상호관련된 상태 또는 이벤트로 나타내질 수 있음을 이해하고 인식할 것이다. 또한, 본 발명에 따른 방법론을 구현하기 위해, 여기에 설명된 모든 동작들이 필요한 것은 아니다.

블럭(200)에서, 비통합 툴들이 사용가능하지만, 통합 툴로서는 사용될 수 없다. 블럭(204)에서, 아티펙트 제공자 API는, 툴들 중 하나가 자신의 아티펙트를 공개적 사용을 위해 노출시키는 데에 사용된다. 블럭(206)에서, 아티펙트 소비자 API는 다른 툴이 아티펙트와 연관된 자신의 참조를 노출시키는 데에 사용된다. 블럭(208)에서, 참조가 아티펙트에 링크되어, 툴들을 통합한다. 그 다음, 프로세스는 종료 블럭에 도달한다.

이제 도 3을 참조하면, 소비자와 제공자의 링크 패턴(300), 및 참조 아티펙트와 피참조 아티펙트의 연관이 도시된다. 아티펙트 제공자(302)는, 적어도 느슨하게 결합된 서버 기반 상호작용, 느슨하게 결합된 클라이언트, 캐싱, 및 호출자들과의 계약에 의한 임의의 아티펙트-특정 함수를 지원하는 밀접하게 결합된 상호작용과 같은 엔티티들을 지원하기 위한 다수의 상이한 유형의 URI를 생성하고 드러낸다. 도시된 바와 같이, 제공자(302)는 아티펙트-특정 데이터를 노출시키는 것을 용이하게 하는 피참조 아티펙트의 URI를 생성한다. 아티펙트 소비자(304)는 아티펙트-특정 데이터를 노출시키는 참조 아티펙트로의 URI, 및 피참조 아티펙트로의 링크를 홀딩한다. 링크 유형은 링크와 함께 한다. 피참조 아티펙트로의 URI는 느슨하게 결합된 상호작용 및 밀접하게 결합된 상호작용을 지원하며, 밀접하게 결합된 상호작용은 호출자들과의 계약에 의한 임의의 아티펙트-특정 함수를 지원한다. 소비자(304)로부터 제공자(302)로의 링크(306)가 구축되어, 아티펙트 데이터를 소비자(304)에게 노출시킨다. 링크의 사용은 URI 또는 개시된 URI에 의해 나타나는 구조 어느 것에 의해서도 제한되지 않지만, 일반적으로 임의의 유형의 링크 메커니즘일 수 있다는 점에 유의해야 한다. 이제 도 4를 참조하면, 아티펙트 종속성(artifact dependency)의 표시를 용이하게 하는 사용자 인터페이스(UI; 400)가 도시된다. UI(400)는 등록된 링크 정의로부터의 링크 유형 정보(402), 등록된 아티펙트 정의로부터의 아티펙트 유형 정보(404), GetArtifact로부터의 아티펙트 네임 정보(406), 및 GetArtifact로부터의 최종 수정된 정보(408)를 디스플레이하는 데에 사용될 수 있다.

본 발명의 기본적인 구성요소는 다음을 포함한다. 일관적 데이터는 아티펙트 메타데이터, 링크 인스턴스 및 아티펙트/링크 인스턴스 캐시 정보를 포함한다. 아티펙트 메타데이터와 관련하여, 각각의 아티펙트 제공자는 링크 서비스에 참여하기 위하여, 자신이 제공하는 각 아티펙트의 아티펙트 유형을 등록하고, 각 아티펙트가 호스트할 수 있는 링크 유형을 등록한다. 링크 인스턴스(일반 아티펙트 제공자 데이터베이스)와 관련하여, 개시된 링크 아키텍처는, 링크를 본래대로 저장하는 것이 어렵거나 불편한 툴을 위하여 "일반 아티펙트 제공자" 구현을 포함한다. 캐시는 아티펙트들 및 그들의 링크를 캐시하여 쿼리 성능을 최적화하는 것을 용이하게 한다.

공통 포맷 및 스키마는 아티펙트 ID, 일반 아티펙트 스키마(GAS), 일반 아티펙트 데이터베이스 스키마(GADS)를 포함한다. 아티펙트 ID는 불변적이고 고유하게 구성된 키이다. 이것은 아티펙트 제공자에 의해 제공된 서비스를 통해 노출되고 URI로서 실현된다. GAS는 임의의 아티펙트가 일반 요청을 통해 자신이 아웃바운드 링크와 함께 리턴될 수 있게 하는 공통의 단축된 형태이다. GADS는 일반 아티펙트 스키마 형태로 표시된 링크 및 아티펙트의 저장을 가능하게 한다. 이것은 일반 아티펙트 제공자 및 아티펙트/링크 캐시에 대한 스키마로서 사용된다.

본 발명의 아키텍처는 링크-인에이블된 아티펙트 제공자에 대한 API 및 규약을 포함한다. 이러한 API는 통합에 참여하는 각각의 서비스 또는 툴에 의해 구현된다. GetArtifacts 메소드는 아티펙트 ID의 스트링에 기초하여 일반 아티펙트 스키마 포맷으로 1:m 아티펙트를 리턴한다. 규약에 의해서, 각각의 GetArtifact 구현은 일반 아티펙트 스키마-순응 아티펙트를 리턴하는 링크 캐시 관리자(및 환경 내의 다른 툴)에 대하여 ArtifactChanged 이벤트를 일으킨다. SynchronizeArtifactCache 메소드는 ArtifactChanged와 유사한 이벤트를 일으킴으로써 아티펙트 및 링크를 캐시에 비동기적으로 보고한다.

아키텍처는 링크-인에이블된 아티펙트 소비자에 대한 API를 포함한다. 이러한 API는 링크에 참여하고 다른 아티펙트로의 링크를 홀딩하는 각각의 툴 또는 서비스에 의해 구현되어야 한다. GetReferencedArtifact 메소드는 일반 아티펙트 스키마 포맷으로 아티펙트 ID의 공급된 세트를 포인트하는 아티펙트의 리스트를 리턴한다.

또한, 링크 관리자에 대한 API도 제공된다. GetArtifacts 메소드는 잠재적으로 다수의 제공자로부터의 아티펙트의 세트를 일반 아티펙트 스키마 포맷으로 리턴한다. 아티펙트/링크 캐시가 인에이블되면, 오버라이드되지 않는 한 그로부터 대답이 유도되며, 이 경우에 제공자는 개별적으로 쿼리되고 그들의 응답이 모아진다. GetReferencedArtifact 메소드는 일반 아티펙트 스키마-포맷으로 아티펙트 ID의 공급된 세트를 포인트하는 잠재적인 복수의 제공자로부터 아티펙트의 리스트를 리턴한다. SynchronizeArtifactCache 메소드는 전체 캐시 동기화를 위하여 각각의 툴의 SynchronizeArtifactCache 메소드를 인보크(invoke)한다.

아키텍처는 일반 아티펙트 제공자(GAP)에 대한 API를 제공한다. 아티펙트 제공자 API로부터의 모든 메소드는 GAP에 의해 구현된다. 아티펙트 소비자 API로부터의 모든 메소드는 GAP에 의해 구현된다. UpdateArtifactData 메소드는 일반 아티펙트 스키마-포맷인 아티펙트를 GAP 데이터베이스로 푸쉬한다. GetArtifactsByExternalID 메소드는, 아티펙트 ID을 따르지는 않지만 기초 툴에 의해 이해되는 식별자를 사용하여 GAP 데이터베이스를 쿼리할 수 있게 한다.

재호스트가능한 UI(re-hostable UI)가 아티펙트간 참조를 나타내기 위해 제공된다.

적합하게 형성된 IS URI

아티펙트의 식별자는 적합하게 형성된 IS Uri로 구성된다. 각각의 IS Uri는 아티펙트의 제공자에 의해 형성된다. 적합하게 형성된 Uri를 구성하는 일부 구성요소는 IS 환경으로부터 유도되며, 다른 것들은 툴 자체에 의해 공급된다. 이하에서는, Uri의 각 구성요소를 식별하고, 그 값이 어떻게 유도되는지를 설명한다.

다음은 적합하게 형성된 IS Uri의 샘플이다:

```
http://<ISNamespace>/vset/<tooltype>.<toolinstance>/{<artifacttype>/}<t
ool-specific id>
```

- http는 Uri가 IDE(통합 개발 환경)의 외부에서 나타날 수 있기 때문에 나타나는 상수이다. Uri가 IDE 외부에서 나타나는 경우, 그 Uri는 "외부 어드레스가능성"에서 후술되는 규칙에 따라 핸들링된다.

- 각각의 IS 논리적 서버는 네임스페이스를 정의한다. <ISNamespace>는 단순히 서버의 네임, 예를 들어 "Server22"이다.

- vset는 IS Uri 핸들러가 스트링을 IS Uri로 인식할 수 있게 하는 상수이다.

- <tooltype>은 아티펙트에 관한 질문을 유지하고 그에 대답하는 것을 책임지는 툴(즉, 아티펙트 제공자)에 의해 지원되는 툴 인터페이스를 식별한다. 이것은 호출 툴(링크를 홀딩하는 툴)이 툴에 의해 제공된 인터페이스에 기초하여 아티펙트를 어떻게 핸들링할지를 결정할 수 있게 한다(예를 들어, vsbug, vsversionstore 등). tooltype은 API를 식별하고, 툴에 의해 지정된다.

- <toolinstance>는 식별된 아티펙트를 책임지는 아티펙트 제공자의 특정 인스턴스를 설명한다. 조합 <tooltype><toolinstance>는 IS 네임스페이스 내에서 고유하다. 예를 들어, tooltype 인스턴스는 설치시에 IS에 의해 할당되어 툴에 저장된다.

- <artifacttype>(선택사항)은 툴에 의해 유지되는 아티펙트의 유형이다. 툴은 유형이 불변적인 경우에만(즉, 인스턴스의 유형이 그 수명동안 변경될 수 없는 경우에만), 아티펙트 유형을 공급할 것이다. 아티펙트 유형은 설치시에 툴에 의해 등록된다.

- <tool-specific id>는 아티펙트 인스턴스로의 불변적인 참조이다. tool-specific id는 툴에 의해 생성되고 유지된다.

예:

H에 저장된 파일을 포인트하는 Uri:

http://Server08/vset/vsversionstore.2/file/b4e3216

C의 요구조건 레코드를 포인트하는 Uri:

http://Server08/vset/vsbug.1/req/345

<u>아티펙트 URI를 엔코드/디코드하는 유틸리티 함수</u>

URI의 일관적 포맷팅을 장려하기 위해, IS는 아티펙트 URI를 구조에 기초하여 엔코딩 및 디코딩하는 유틸리티 함수의 쌍을 제공한다.

```
string EncodeUri(ArtifactId artifactId)
```

EncodeUri는 유형 ArtifactId의 구조를 취하며, 다음의 URL 규칙에 따르는 스트링을 생성한다. ArtifactID.ToolInstance (toolType.toolInstance 유형) 및 ArtifactID.ArtifactType에 대한 널이 아닌 값이 호출자에 의해 공급된다. ArtifactID.ServerNamespace = null이면, 현재 활성인 네임스페이스의 네임은 URI의 ISNamespace부를 위해 사용된다. 대부분의 경우에, 툴은 ArtifactID.ToolSpecificID에 대해 널이 아닌 값을 공급한다. ArtifactID.ToolSpecificID = null이면, EncodeUri는 스트링 "<?>"를 tool-specific id에 삽입한다. 이러한 스트링은 아티펙트를 일반 아티펙트 제공자에 추가할 때, 특별한 의미를 가지며, 그렇지 않은 경우에는 그 사용이 회피될 수 있다.

```
ArtifactId DecodeUri(string artifactUri)
```

DecodeUri는 IS Uri 규칙을 따르는 스트링을 취하여, 유형 ArtifactId의 구조를 리턴한다.

예:

아래의 코드 단편은 EncodeUri 메소드를 사용하여 아티펙트 Id를 작성하고, 그것을 적합하게 형성된 IS Uri로 변환하는 것을 나타내고 있다. 그에 의한 URI는 다음과 같다.

```
http://IS001/vset/Work%20Item%20%Tracking.1/defect/5291
```

DecodeUri는 역으로 작용한다.

```
public void EncodeUriExample()
{
      ISServerProxy IS = new ISServerProxy();

      ArtifactId artId = new ArtifactId();
      artId.ServerNamespace = null;
      artId.ToolInstance = "WorkItems.1";
      artId.ArtifactType = "defect";
      artId.ToolSpecificId = "5291";

      string Uri = IS.EncodeUri(artId);
}
```

느슨한 결합 대 밀접한 결합

IS 링크 환경에서, 툴은 링크를 사용하여 느슨하게 결합된 거동을 인에이블시킨다. 또한, 툴은 밀접하게 결합된 거동을 통해 보다 더 높은 레벨의 특수화된 통합을 제공하는 인터페이스 및 특수화된 상호작용을 제공할 수 있다. 각각은 적절한 문맥에서 유효하다.

느슨한 결합은 다음의 이점을 제공한다. 기존의 어플리케이션을 분산시키지 않고서도 새로운 아티펙트 유형을 아티펙트들의 웹에 추가하는 것이 간단하다. 이는 제3자 통합에 대해서 특히 유용하다. 결과를 모아서 다양한 유형의 아티펙트에 대한 완벽한 "사용처" 픽처("where-used" picture)를 제공하는 것이 가능하다. 아티펙트 핸들러(제공자 및 소비자)는 거동을 철저하게 변경할 수 있고, 그들이 일반 링크 인터페이스를 계속 사용하는 한, 모든 느슨하게 결합된 거동은 계속 작용할 것이다.

밀접하게 결합된 거동을 제공하는 툴은, 링크 서비스를 따르는 링크는 사용하지만, 일반 링크 인터페이스는 사용하지 않는다. 대신에, 아티펙트 소비자는 아티펙트 제공자에 의해 제공되는 특수한 인터페이스의 이점을 취한다. 이러한 거동은 참여하는 툴들에 매우 특정한 툴간 거동(cross-tool behavior)을 제공하는 데 사용될 수 있다.

느슨한 결합은, 피참조 아티펙트에 대한 포인터가 그 피참조 아티펙트의 제공자에게 알려지지 않은 참조 아티펙트에 제시될 수 있는 경우에 사용될 수 있다.

예 - 제3자 테스트 경우의 관리 툴은 슈트 내에 도입된다. 이것은 IS-인에이블된 것이므로, 예를 들어 결함에 대한 링크를 작성하기 위한 메커니즘을 제공한다. 결함은 역전 쿼리 내에 그에 대한 리턴 정보를 나타내기 위해 이러한 새로운 유형의 참조 아티펙트에 대해 알 필요가 없다.

느슨한 결합은 참조 아티펙트가 자신이 알지 못하는 피참조 아티펙트를 포인트할 수 있는 경우에, 또한 피참조 아티펙트에 이용가능한 거동이 참조 아티펙트에 의한 영향을 받지 않는 경우에 사용될 수 있다.

예 - 사용자가 어떤 작업 항목이 빌드에 연관됐는지를 알고자 할 때, IDE 내부에서, 작업 항목의 리스트가 빌드의 역전 쿼리 UI 내에 리턴된다. 마찬가지로, 사용자가 어떤 작업 항목이 특정한 변경 세트에 의해 영향을 받았는지를 알고자 할 때, 작업 항목의 리스트는 변경 세트의 역전 쿼리 UI에 리턴된다. 어떤 경우에서도, 사용자는 임의의 작업 항목을 더블클릭할 수 있어야 하며, 해당 윈도우에 대한 문서 윈도우가 디스플레이될 것이다. 바람직한 거동은 참조 아티펙트가 빌드인지 변경 세트인지의 여부에 상관없이 동일하므로, (이러한 2가지 경우에서) 거동은 참조 아티펙트에 의한 영향을 받지 않는다.

느슨하게 결합된 상호작용은 (a) IS 규약을 따르고, (b) 표준 인터페이스를 구현하는 아티펙트 제공자 및 소비자를 통해 인에이블된다.

툴이 링크에 참여하는 방법

아티펙트를 제공하는 각각의 툴은 아티펙트 제공자이다. 아티펙트 제공자는 자신의 아티펙트 유형을 등록하고, 느슨하게 결합된 툴이 그들의 아티펙트를 판독할 수 있도록 하는 인터페이스의 세트를 구현한다. 아티펙트 제공자의 책임은 "아티펙트 제공자 의무"에서 상세하게 설명된다.

다른 아티펙트로의 링크도 포함하는 아티펙트 제공자는 아티펙트 소비자이다. 아티펙트 소비자는 제공자의 의무 외에 소정의 추가 의무를 갖는다. 그것은 링크 유형을 등록하고, 쿼리 및 역전 쿼리에 답하기 위한 표준 메소드를 제공한다. 아티펙트 소비자 책임은 "아티펙트 소비자 의무"에서 상세하게 설명된다.

마지막으로, IS 자체는 등록, 발견가능성 및 서비스간 링크 쿼리에 대한 다수의 설비를 제공하며, "IS API"에서 상세하게 설명된다.

아티펙트 제공자 의무

아티펙트를 호스트하는 임의의 툴은 아티펙트 제공자이다. 아티펙트 제공자는 다음의 책임을 갖는데, 여기서 일부는 선택적이며, 다른 것들은 강제적이다. 등록시에, 각각의 아티펙트 제공자는 자신이 호스트하는 각각의 아티펙트 유형을 IS에 등록하고, 자신이 일으킬 수 있는 임의의 이벤트(아티펙트 변경 이벤트 포함)를 등록하며, 자신의 제공자 웹 서비스 인터페이스의 URL을 등록하고, 자신이 소유하는 각각의 아티펙트에 대한 IS Uri을 리턴할 수 있는 웹 서비스 인터페이스를 제공하며, 자신이 소유하는 각각의 아티펙트 유형의 인스턴스를 검색하는 데에 사용될 수 있는 GetArtifacts 메소드를 제공하고, 아티펙트 인스턴스를 리턴하는 각각의 메소드에서의 리턴값으로 아티펙트의 Uri를 포함할 수 있으며, 자신이 소유하는 아티펙트가 생성, 삭제 또는 변경되어, 해당 아티펙트에 대한 GetArtifacts 메소드 실행이 상이한 결과를 리턴하게 될 때 이벤트를 일으키고, 자신이 제공하는 각각의 아티펙트 유형으로의 외부 어드레스가능성을 핸들링하는 방식을 제공하며, 자신이 소유하는 아티펙트에 대한 IDE내 요구(ExecuteDefaultAction, GetAllowableActions, ExecuteAction)에 응답하는 클라이언트 API를 구현할 수 있다. IS 및 다른 것들은 이것을 사용하여 아티펙트 소비자와 제공자 사이의 결합되지 않은 액세스를 제공할 수 있다.

아티펙트 유형 등록

아티펙트 유형 및 링크 유형의 등록은 설치시에 발생한다. 각각의 툴은 IS 등록 서비스 XML 스키마를 따르는 XML (eXtensible Markup Language) 문서를 준비한다. 각각의 아티펙트 유형에 대하여, 해당 유형의 아티펙트에 대한 책임이 있는 제공자, 아티펙트 유형 네임(국부화 불가능) 및 아티펙트 유형 레벨(국부화 가능)과 같은 정보가 요구된다. 각각의 링크 유형에 대하여, 참조 아티펙트의 아티펙트 유형 네임; 링크 유형 네임(국부화 불가능), 링크 유형 레벨(국부화 가능), 역전 링크 유형 레벨(국부화 가능), 및 선택적 피참조 아티펙트 유형 어레이와 같은 정보가 요구된다. 링크 유형 레벨에 관하여, 양방향으로의 판독을 제공하기 위하여, 순방향과 역방향이 제공된다. 예를 들면, "사람이 개를 문다"와, "개가 사람에 의해 물렸다"와 같다. 피참조 아티펙트 유형 어레이가 비어있으면, 링크의 피참조측은 유형이 정해지지 않는다. 즉, 이러한 유형의 링크는 어떠한 아티펙트도 포인트할 수 있다. 피참조 아티펙트 유형 어레이의 각 구성요소는 피참조 아티펙트 유형의 서비스 및 피참조 아티펙트 유형의 네임 둘다를 나타낸다.

제공자 서버 API - IToolServerLinking

아티펙트 제공자는 자신이 소유하는 각각의 아티펙트 유형의 인스턴스를 검색하기 위한 GetArtifacts 메소드를 제공한다. GetArtifacts 메소드는 Uri의 어레이를 입력으로 취한다. 리턴값은 일반 아티펙트 디스플레이 스키마(GAS)를 따르는 객체들의 어레이이다.

```
Artifact[] GetArtifacts (string[] ArtifactUriList)
```

GetArtifacts는 IS Uri의 형태인 아티펙트 Id의 어레이를 취할 수 있다. 제공자는 GAS 포맷으로 나타난 각각의 아티펙트 Id에 대한 아티펙트 인스턴스를 리턴할 책임이 있다.

예:

"IS001"로 명명된 IS 네임스페이스와 "WorkItems. 1"로 명명된 IS001에 등록된 결함 추적 툴, 및 2개의 작업 항목(결함 152, 및 요구조건 153)이 주어진다.

마지막으로, 다음의 2개의 구성요소를 가진 ArtifactUri[]로 명명된 어레이를 가정해보자.

http://IS001/vset/WorkItems.1/Defect/152,

http://IS001/vset/WorkItems.1/Defect/153

이 호출 이후 MyArtifact의 값

```
Artifact[] MyArtifacts = workItemTool.GetArtifacts(ArtifactUri[])
```

은 직렬화(serialize)된 후, 다음의 XML 문서와 유사하게 보일 것이다.

```
<?xml version="1.0" encoding="utf-8" ?>
<Artifacts
xmlns="http://www.company.com/Tool/GenericArtifactSchema.xsd">
    <Artifact>
        <Uri>http://IS001/vset/WorkItems.1/Defect/152</Uri>
        <ProviderName>MS Work Item Tracking</ProviderName>
        <Title>152 (Open)</Title>
        <Type>Defect</Type>
        <LastChangedOn>2003-10-24T20:47:58.170</LastChangedOn>
        <LastChangedBy>dminium</LastChangedBy>
        <ExtendedAttributes>
            <ExtendedAttribute>
                <Name>Status</Name>
                <Value>Open</Value>
            </ExtendedAttribute>
            <ExtendedAttribute>
                <Name>Assigned To</Name>
                <Value>Allen Clark (allclark)</Value>
            </ExtendedAttribute>
        </ExtendedAttributes>
        <Links>
            <Link>
                <LinkType>depends on</LinkType>

<ReferencedUri>http://IS001/vset/WorkItems.1/Defect/173</ReferencedUri>
            </Link>
            <Link>
                <LinkType>found in</LinkType>

<ReferencedUri>http://IS001/vset/Builds.1/Build/2003.11.15</ReferencedU
ri>
            </Link>
            <Link>
                <LinkType>checked in</LinkType>

<ReferencedUri>http://IS001/vset/Hat.1/ChangeSet/987Urt5B</ReferencedUr
i>
            </Link>
        </Links>
    </Artifact>

    <Artifact>
        <Uri>http://IS001/vset/WorkItems.1/Req/153</Uri>
        <ProviderName>Work Item Tracking</ProviderName>
        <Title>153 (In Work)</Title>
        <Type>Requirement</Type>
        <LastChangedOn>2003-10-27T20:36:52.170</LastChangedOn>
        <LastChangedBy>eeykholt</LastChangedBy>
        <ExtendedAttributes>
            <ExtendedAttribute>
                <Name>Status</Name>
                <Value>In Work</Value>
            </ExtendedAttribute>
            <ExtendedAttribute>
                <Name>Assigned To</Name>
                <Value>Raj Selvaraj (naselvar)</Value>
            </ExtendedAttribute>
        </ExtendedAttributes>
        <Links>
            <Link>
                <LinkType>authored in</LinkType>

<ReferencedUri>http://IS001/vset/Teamhare/ReqDoc/38976FBA</ReferencedUr
i>
            </Link>
            <Link>
              <LinkType>depends on</LinkType>

<ReferencedUri>http://IS001/vset/WorkItems.1/Defect/173</ReferencedUri>
            </Link>
        </Links>
    </Artifact>
</Artifacts>
```

SynchronizeArtifactCache 메소드

SynchronizeArtifactCache 메소드는 아티펙트 캐시가 손상된 경우 그것을 재구성하는 데 사용된다.

```
void SynchronizeArtifactCache()
```

SynchronizeArtifactCache 메소드는 IS에 의해 인보크되어 아티펙트 소비자에 의해 홀딩된 순방향 링크의 전체 세트의 통지를 초기화한다. 아티펙트 소비자는 그것의 아티펙트의 각각에 대한 CacheArtifact 이벤트를 일으켜야 한다(복수의 아

티펙트가 단일 이벤트로 패키징될 수 있음). IS는 이러한 이벤트를 듣고, 그 결과를 사용하여, 소비자의 GetArtifacts 또는 GetReferencingArtifacts 메소드를 인보크하지 않고서 간단한 쿼리를 해결하는 데 사용될 수 있는 링크 캐시를 파퓰레이트한다.

```
void SynchronizeArtifactCache(DateTime from, DateTime to)
```

SynchronizeArtifactCache는 데이터 범위에 의해 제한될 수 있다. 이 경우에, 아티펙트 소비자는 "from"과 "to" 사이("from", "to" 도 포함)에 수정된 아티펙트에 대한 CacheArtifact 이벤트를 발행할 의무만 있다.

아티펙트 제공자에 의해 홀딩되는 각각의 아티펙트에 대해서, SynchronizeArtifactCache 메소드는 아티펙트, 및 만약 있다면 그것의 아웃바운드 링크를 XML 문서로 패키징한다(GAS 스키마를 따름). 이러한 포맷팅은 정확히 GetArtifacts에 의해 사용되는 것과 완전히 동일한 논리를 필요로 한다는 점에 유의해야 한다. 이 메소드는 또한 GAS-순응 XML 문서를 IS로 전달하는 CacheArtifact 이벤트를 발생시킨다.

제공자는 자신의 캐싱 동작을 완료한 때에 EndCacheLoad 이벤트를 발행한다. GAS 문서가 다수의 아티펙트를 홀딩할 수 있음에 유의해야 한다. 따라서, 제공자는 단번에 수개의 아티펙트를 캐시로 전달하는 옵션을 갖는다. IS는 일반적으로 ArtifactChanged 이벤트 - 다른 툴들이 가입한 것과 동일한 ArtifactChanged 이벤트- 에 응답함으로써, 자신의 캐시를 갱신한다. 그러나, IS는 다른 툴들 및 서비스들이 가입하지 않은 CacheArtifact 이벤트도 듣는다.

제공자 클라이언트 API - IToolClientLinking

아티펙트 제공자 클라이언트 API는 IDE 내의 느슨하게 결합된 상호작용을 처리하기 위해 존재한다. 예를 들어, 툴이 GetArtifacts 호출의 결과로서, 링크 컬렉션 내에 있는 아티펙트 Id를 갖는 아티펙트들의 리스트를 디스플레이하는 시나리오를 가정해보자. 사용자는 (임의의) 아티펙트 중 하나를 디스플레이하기를 원한다. 아티펙트의 제공자가 여기 설명된 ExecuteDefaultAction 메소드를 구현하면, 리스트의 홀더는 아티펙트 또는 아티펙트의 유형에 대한 어떠한 지식도 없이 액션(대부분 아티펙트를 디스플레이하는 네비게이션 액션)을 인보크할 수 있다.

ExecuteDefaultAction 메소드

이 메소드는 아티펙트 제공자에 의해 그것의 제공자 클라이언트의 일부분으로서 제공될 수 있다. 그것은 제공자의 클라이언트 설치의 일부분으로서 설치되고, 발견될 수 있다.

```
bool ExecuteDefaultAction(string artifactUri)
```

ExecuteDefaultAction은 단일 아티펙트 URI를 입력으로 취한다. 제공자 클라이언트는 아티펙트에 적합한 임의의 디폴트 거동을 실행할 책임이 있다. 디폴트 액션의 실행이 성공이었으면 참을 리턴한다. ExecuteDefaultAction은 일반적으로 직접 인보크되지 않는다. 그 대신에, IS 클라이언트의 ExecuteDefaultAction 메소드를 통해 인보크된다.

ArtifactChanged 이벤트

아티펙트가 생성, 삭제 또는 변경되어 해당 아티펙트에 대한 GetArtifacts 메소드 실행이 상이한 결과를 리턴하게 되는 경우, 아티펙트 제공자는 ArtifactChanged 이벤트를 일으켜야 한다.

아티펙트가 변경될 때, 제공자는 FirAsyncEvent 메소드를 사용하여 IS Eventing에 비동기 ArtifactChanged 이벤트를 일으킨다. ArtifactChanged 이벤트 유형은 등록 동안 IS의 일부분으로서 자동적으로 설치된다. 이것은 GAS 스키마를 따르는 XML 문서를 포함한다. GAS가 아티펙트의 순방향 링크를 포함하므로, ArtifactChanged는 청취자에게 링크 변경 및 기타 아티펙트 변경을 알리는 데 사용된다.

ArtifactChanged 이벤트는 아티펙트가 추가, 변경 또는 삭제될 때마다 일어난다. GAS 스키마는, 호출자가 자신이 포함하는 아티펙트가 추가, 변경 또는 삭제되었는지의 여부를 나타낼 수 있게 하는 아티펙트 구성요소 상의 특성을 포함한다. 이하는, 단일 ArtifactChanged 이벤트에 보고된 2개의 변경, 즉 결함 152가 추가되었고 요청 153이 삭제되었다는 것을 나타내고 있다..

```
<?xml version="1.0" encoding="utf-8" ?>
<Artifacts
xmlns="http://www.company.com/Tool/GenericArtifactSchema.xsd">
    <Artifact ChangeType="Add">
        <Uri>http://IS001/vset/WorkItems.1/Defect/152</Uri>
        <ProviderName>MS Work Item Tracking</ProviderName>
        <Title>152 (Resolved)</Title>
        <Type>Defect</Type>
        <LastChangedOn>2003-10-24T20:47:58.170</LastChangedOn>
        <LastChangedBy>dminium</LastChangedBy>
        <ExtendedAttributes>
            <ExtendedAttribute>
                <Name>Status</Name>
                <Value>Resolved</Value>
            </ExtendedAttribute>
            <ExtendedAttribute>
                <Name>Assigned To</Name>
                <Value>Peanuts McFadden (nuts)</Value>
            </ExtendedAttribute>
        </ExtendedAttributes>
        <Links>
            <Link>
                <LinkType>depends on</LinkType>

<ReferencedUri>http://IS001/vset/WorkItems.1/Defect/173</ReferencedUri>
            </Link>
            <Link>
                <LinkType>found in</LinkType>

<ReferencedUri>http://IS001/vset/Builds.1/Build/2003.11.15</ReferencedU
ri>
            </Link>
            <Link>
                <LinkType>checked in</LinkType>

<ReferencedUri>http://IS001/vset/Hatteras.1/ChangeSet/987Urt5B</Referen
cedUri>
            </Link>
        </Links>
    </Artifact>
    <Artifact ChangeType="Delete">
        <Uri>http://IS001/vset/WorkItems.1/Defect/23</Uri>
    </Artifact>
</Artifacts>
```

임의의 툴은 일반 ArtifactChanged 이벤트에 등록할 수 있다. 추가적으로, 아티펙트/링크 캐시는 이러한 이벤트를 듣고, 그러한 이벤트를 이용하여 자신을 갱신한다.

외부 어드레스가능성

아티펙트 제공자는, 자신이 제공하는 각각의 아티펙트에 대한 외부 어드레스가능성을 핸들링하는 방식을 제공한다. 사용자가 IS 아티펙트에 대한 URI를 이메일에 넣는 것을 가정해보자. URI가 외부적으로 어드레스가능하지 않은 경우, 이메일의 수신자는 그 URI를 이용하여 아무것도 하지 못한다. URI가 외부적으로 어드레스 \\VSServer8\FinProj\Acctg\WebUI\taccts.cs 가능한 경우, 브라우저는 그것을 이용하여 무엇을 할지를 알아낼 수 있다.

예:

예를 들어, 아래와 같은 위치의 버전 저장장치에 저장된 소스 파일을 가정해보자:

\\VSServer8\FinProj\Acctg\WebUI\taccts.cs

이것은 아래와 같은 URI로 나타내진다.

http://VSServer8/vset/vsversionstore.2/file/b4e3216

이제, tacct 프로그램이 실패했음을 나타내는 이메일이 사용자에게 송신되는 것을 고려해보자("tacct"는 IS Uri가 뒤에 있는 하이퍼링크임). 사용자가 하이퍼링크를 선택할 때, 사용자는 URI와 연관된 것에 대한 소정의 표현이 생성될 것임을 정당하게 예측한다. 이 경우에, 그것은 파일에 관한 소정의 메타데이터 및 설명적 정보(그의 메모리를 조그하는 것을 돕기 위한 것임)이다. 이는, 각각의 아티펙트 제공자가 자신의 아티펙트를 외부적으로 어드레스가능한 것으로 하기 때문에 가능한 것이다. 이것은, 설치시에, 자신이 관리하는 유형의 아티펙트를 핸들링할 수 있는 페이지를 제공함으로써 행해진다. 이러한 페이지는 IS 서버 상의 잘 알려진 가상 디렉토리에 저장되고, 그것의 툴 인스턴스에 대해 명명된다. 페이지는 매우 단순하며, 단순한 XSL 변형을 인보크하여 판독 전용 페이지를 생성한다. 대안적으로, 페이지는 완전 블로된 어플리케이션(full-blown application)으로의 게이트웨이를 제공하도록 복잡하게 될 수 있다.

IS는 URI를 적절한 디렉토리로 지향시키는 URL로 변환하고 페이지가 아티펙트 유형 및 별명을 통상의 매개변수로서 처리할 수 있게 하는 ISAPI(인터넷 서버 어플리케이션 프로그램 인터페이스) 필터를 제공한다.

예:

이러한 인입 Uri: http://Server8/vset/vsversionstore.2/file/b4e3216는 ISAPI 필터에 의해 다음과 같은 것으로 변환된다:

```
http://Server8/vsversionstore.2.aspx?artifactType=file&id=b4e3216
```

여기서, "artasp"는 ASP 페이지가 저장된 Server8 상의 폴더이다.

아티펙트 소비자 의무

등록시에, 각각의 아티펙트 소비자는 자신이 호스트하는 각각의 링크 유형을 IS에 등록하고, 자신의 소비자 웹 서비스 인터페이스의 URL을 등록하며, 자신이 응답할 수 있는 임의의 이벤트(아티펙트 변경 이벤트를 포함)에 가입하고, "당신은 나를 포인트하는가?"란 쿼리에 응답할 수 있는 GetReferencingArtifacts 메소드를 제공하며, ArtifactChanged 이벤트에 응답할 수 있다. 삭제에 대하여, 참조 아티펙트 링크는 삭제되야 한다. 변경에 대하여, 참조 아티펙트는 링크를 무효화시킬 수 있는 상황이 있는지를 확인할 수 있다.

소비자 서버 API

아티펙트 소비자는 "당신은 나를 포인트하는가?"란 쿼리에 응답할 수 있는 GetReferencingArtifacts 메소드를 제공해야 한다. GetReferencingArtifact는 URI들의 어레이를 입력으로서 취한다. 리턴값은 일반 아티펙트 데이터베이스 스키마(GAS)를 따르는 아티펙트 객체들의 어레이다. 다음은 메소드 서명이다.

```
Artifact[] GetReferencingArtifacts (string[] artifactUriList)
```

이러한 버전의 GetReferencingArtifact는 피참조 아티펙트 Id들의 어레이를 취한다. 소비자는, 자신이 소유한 참조 아티펙트들 중, 포함된 피참조 아티펙트들 중 임의의 것을 GAS 순응 아티펙트 객체로서 포인트하는 모든 참조 아티펙트들의 리스트를 리턴할 책임이 있다.

```
    Artifact[] GetReferencingArtifacts (string[] artifactUriList,
LinkFilter[] linkFilterList)
```

이러한 버전의 GetReferencingArtifacts는 피참조 아티펙트 Id들의 어레이 및 링크 필터들의 어레이를 취한다. 링크 필터들은 리턴된 아티펙트를 아티펙트, 아티펙트 유형 또는 링크 유형을 제공하는 툴 유형에 의해 제한하는 데 사용될 수 있다.

ArtifactChanged 이벤트 핸들러

여기에 설명된 바와 같이, 아티펙트가 실질적으로 변경될 때마다, 아티펙트 제공자는 ArtifactChanged 이벤트를 일으킨다. 각각의 아티펙트 소비자는 ArtifactChanged 이벤트 핸들러를 등록함으로써 이러한 ArtifactChanged 이벤트를 들을 수 있다.

각 소비자의 ArtifactChanged 이벤트 핸들러는 이벤트를 검사하여, 그것이 관심있는 아티펙트를 홀딩하는지를 결정한다. 관심있는 아티펙트를 홀딩하는 경우, 소비자의 이벤트 핸들러는 변경에 응답할 수 있다. 일반적으로 이러한 응답은 다음의 형태들 중 하나를 취한다.

- 소비자가 오직 아티펙트로의 포인터만을 홀딩하면, 소비자는 삭제의 경우에 해당하는 ArtifactChanged에만 종종 관심이 있다. 소비자는 삭제된 아티펙트로의 모든 링크를 제거한다.

- 일부 경우에서, 피참조 아티펙트가, 참조 아티펙트가 그 피참조 아티펙트로의 링크를 홀딩하는 특수한 상태에 있을 것을 요구하는 특수한 정책 정보가 있을 수 있다. 그러한 경우, 소비자는 변경후에 피참조 아티펙트가 여전히 올바른 상태에 있음을 확신할 수 있다. 그렇지 않은 경우, 소비자는 그것의 링크를 제거한다.

아티펙트/링크 캐시는 ArtifactChanged 이벤트를 사용하여 최신 상태로 된다.

IS API

이하의 API들은 IS의 다양한 구성요소들에 의해 직접적으로 제공된다.

서버 API - EncodeUri/DecodeUri 유틸리티 메소드

GetArtifacts 메소드

GetArtifacts 메소드는 URI의 어레이를 입력으로 취한다. 리턴값은 GAS를 따르는 객체이다. 논리적 관점에서 볼 때, GetArtifacts의 IS 버전은, 아티펙트를 요청받은 각각의 아티펙트 제공자의 GetArtifacts 메소드를 인보크한다. 물리적으로, 직접 매개변수가 참이 아닌 경우, 쿼리는 아티펙트/링크 캐시에 대하여 실행된다. 메소드 서명은 다음과 같다.

```
Artifact[] GetArtifacts (string[] artifactUriList)
```

GetArtifacts는 ISUri 형태의 아티펙트 Id의 어레이를 취한다. 제공자는 GAS 포맷으로 표현된 각각의 아티펙트 Id에 대한 아티펙트 인스턴스를 리턴할 책임이 있다. 이러한 버전의 GetArtifacts는 IS 아티펙트/링크 캐시에 대하여 실행된다.

```
Artifact[] GetArtifacts (string[] artifactUriList, bool direct)
```

이러한 버전의 GetArtifacts는, 호출자가 직접 매개변수에 대한 값을 제공할 수 있다는 것을 제외하고, 정확히 상기 형태처럼 행동한다. direct = true인 경우, 결과 세트는 아티펙트/링크 캐시를 이용하는 대신 개별 제공자의 GetArtifacts 메소드를 직접적으로 호출함으로써 작성된다. 이는 고가의 수집(aggregate) 쿼리의 비용으로 최신 데이터를 제공한다.

GetReferencingArtifacts 메소드

IS 서비스간 역전 쿼리인 GetReferencingArtifact는 소트의 쿼리 수집자로 실행된다. 논리적 관점에서 볼 때, IS로부터 요청된 서비스간 역전 쿼리는, 쿼리에게 정보를 제공할 수 있는 모든 IS-인에이블된 툴에 대해서 링크 역전 쿼리를 실행시키고, 모든 응답들로부터의 결과를 수집하며, 단일의 결과 세트(XML 문서의 형태)를 사용자에게 다시 송신할 수 있다. 물리적 관점에서 볼 때, IS 서비스간 역전 쿼리는 IS 아티펙트/링크 캐시에 대해서 실질적으로 실행될 수 있다.

```
Artifact[] GetReferencingArtifacts (string[] artifactUriList)
```

이러한 버전의 GetReferencingArtifacts는 피참조 아티펙트 Id들의 어레이를 취한다. 아티펙트들의 전체 리스트를 작성하기 위하여, (등록 정보에 기초한) 각각의 잠재적 아티펙트 소비자가 참고된다. 각각의 소비자는, 자신이 소유하는 참조 아티펙트들 중에서, 포함된 피참조 아티펙트 중 임의의 것을 GAS-순응 아티펙트 객체로서 포인트하는 모든 참조 아티펙트들의 리스트를 리턴할 책임이 있다. 그 후 IS는 리스트들을 병합하고, 단일의 아티펙트 어레이를 클라이언트에 리턴한다.

```
    Artifact[] GetReferencingArtifacts (string[] artifactUriList,
LinkFilter[] linkFilterList)
```

이러한 버전의 GetReferencingArtifacts는 링크 필터들의 추가적 어레이를 취한다는 것만 제외하고 이전 버전과 동일하다. 링크 필터는 리턴되는 아티펙트를 아티펙트, 아티펙트 유형 또는 링크 유형으로 제한하는 데에 사용될 수 있다.

```
    Artifact[] GetReferencingArtifacts (string[] artifactUri, bool
direct)
```

이러한 버전의 GetReferencingArtifacts는 호출자가 직접 매개변수에 대한 값을 제공할 수 있다는 것만 제외하고는, 정확하게 제1 형태로서 거동한다. direct = true인 경우, 결과 세트는 아티펙트/링크 캐시를 사용하는 대신 개별 제공자의 GetReferencingArtifacts 메소드를 직접적으로 호출함으로써 작성된다. 이는 고가의 수집 쿼리의 비용으로 최신 데이터를 제공한다.

```
Artifact[] GetReferencingArtifacts (string[] artifactUriList,
bool direct, LinkFilter[] linkFilterList)
```

이러한 버전의 GetReferencingArtifacts는 호출자가 직접 매개변수에 대한 값을 제공할 수 있다는 것만 제외하고는, 정확하게 제2 형태로서 거동한다. direct = true인 경우, 결과 세트는 아티펙트/링크 캐시를 사용하는 대신 개별 제공자의 GetReferencingArtifacts 메소드를 직접적으로 호출함으로써 작성된다. 이는 고가의 수집 쿼리의 비용으로 최신 데이터를 제공한다.

링크 헬퍼 메소드 - ILinking

IS 링크 헬퍼 메소드는 아웃바운드(참조) 링크가 각각의 아티펙트 내에 네스트된 아티펙트 오브젝트들의 어레이를 사후처리하여, GLS를 따르는 링크들의 단순 리스트를 생성한다. ExtractLink는 선택적인 사용자 제공 필터에 순응하도록 된 아티펙트 객체 어레이의 내부에서 발견되는 모든 링크의 단순 리스트를 리턴한다.

ExtractLink의 제3 형태는 피참조 Id가 스트링 어레이를 통해 전달되는 링크만을 제외하고 동일한 것을 수행한다. 이러한 방식에서, ExtractLinks는 GetReferencingArtifacts와 쌍을 이루어, 관심있는 아티펙트 Id를 포인트하는 링크들만의 필터링된 단순 리스트를 생성할 수 있다.

```
Link[] ExtractLinks(Artifact[] artifactList)
```

각각의 링크가 자신의 참조 아티펙트와 함께 패키징되는 아티펙트 객체 어레이가 주어지면, ExtractLinks는 GLS를 따르는 Link 어레이 내부에 링크의 단순 리스트를 생성한다.

```
Link[] ExtractLinks(Artifact[] artifactList, LinkFilter[]
linkFilterList)
```

이러한 형태의 ExtractLinks에서, 링크는 linkFilterList 내의 필터를 따르는 경우에만 포함된다.

```
Link[] ExtractLinks(Artifact[] artifactList, LinkFilter[]
linkFilterList, string[] artifactUriList)
```

이러한 형태에서, 피참조 URI가 ArtifactUri 스트링 어레이에 포함된 링크들만이 리턴된다. 어떠한 링크 유형 필터링도 요구되지 않으면, LinkFilterList는 널일 수 있다.

ExtractLinks 예1: 링크들의 단순 리스트를 생성.

아티펙트 어레이를 결과로 낳는 아티펙트를 획득한다. 그 후, 그 아티펙트 객체 어레이로부터 링크들의 단순 리스트를 추출한다.

```
ISServerProxy IS = new ISServerProxy();
string[] artifactUris = new string[]
{"http://IS001/vset/workitems.1/defect/152","http://IS001/vset/WorkItem
s.1/Req/153"};
Artifact[] artifacts = IS.GetArtifacts(artifactUris);
Link[] links = IS.ExtractLinks(artifacts);
```

ExtractLinks로부터 리턴된 링크 객체는 다음의 XML 스키마에 대응한다.

```
      <?xml version="1.0" encoding="utf-8" ?>
      <Links xmlns="http://tempuri.org/ISGenericLinkSchema.xsd">
          <Link>

<ReferringUri>http://IS001/vset/WorkItems.1/Defect/152</ReferringUri>
              <LinkType>depends on</LinkType>
<ReferencedUri>http://IS001/vset/WorkItems.1/Defect/173</ReferencedUri>
          </Link>
          <Link>
<ReferringUri>http://IS001/vset/WorkItems.1/Defect/152</ReferringUri>
              <LinkType>found in</LinkType>
<ReferencedUri>http://IS001/vset/Builds.1/Build/2003.11.15</ReferencedU
ri>
          </Link>
          <Link>
<ReferringUri>http://IS001/vset/WorkItems.1/Defect/152</ReferringUri>
              <LinkType>checked in</LinkType>
      <ReferencedUri>http://IS001/vset/Hatteras.1/ChangeSet/987Urt5B</R
eferencedUri>
          </Link>
          <Link>
<ReferringUri>http://IS001/vset/WorkItems.1/Req/153</ReferringUri>
              <LinkType>authored in</LinkType>
      <ReferencedUri>http://IS001/vset/eLeadSharePoint/ReqDoc/38976FBA<
/ReferencedUri>
          </Link>
          <Link>
<ReferringUri>http://IS001/vset/WorkItems.1/Req/153</ReferringUri>
              <LinkType>depends on</LinkType>
<ReferencedUri>http://IS001/vset/WorkItems.1/Defect/173</ReferencedUri>
          </Link>
      </Links>
```

ExtractLinks 예2: 특정 결함을 포인트하는 링크들의 리스트를 생성.

결함 173이 결함 152 및 요청 153으로부터만 참조된다고 가정하자. 그러한 경우, 결함 173에 대한 GetReferringArtifacts는 정확하게 동일한 아티펙트 어레이를 리턴할 것이다. 이 예에서, ExtractLinks는 실질적으로 결함 173을 다시 포인트하는 링크들만을 리턴하는 데 사용된다.

```
      ISServerProxy IS = new ISServerProxy();
      string[] artifactUris = new string[1]
{"http://IS001/vset/workitems.1/defect/173"};
      Artifact[] artifacts = IS.GetReferencingArtifacts(artifactUris);

      Link[] links = IS.ExtractLinks(artifacts, null, artifactUris);
      The links object returned from ExtractLinks (now filtered by the
same set of artifactUris as GetReferencingArtifacts) is shown in Figure
5.

      <?xml version="1.0" encoding="utf-8" ?>
      <Links xmlns="http://tempuri.org/ISGenericLinkSchema.xsd">
          <Link>

<ReferringUri>http://IS001/vset/WorkItems.1/Defect/152</ReferringUri>
              <LinkType>depends on</LinkType>
      <ReferencedUri>http://IS001/vset/WorkItems.1/Defect/173</Referenc
edUri>
          </Link>
          <Link>

<ReferringUri>http://IS001/vset/WorkItems.1/Req/153</ReferringUri>
              <LinkType>depends on</LinkType>
      <ReferencedUri>http://IS001/vset/WorkItems.1/Defect/173</Referenc
edUri>
          </Link>
      </Links>
```

SynchronizeArtifactCache

이 메소드는 아티펙트/링크 캐시에 배경 갱신이 일어나도록 인보크된다.

```
void SynchronizeArtifactCache()
```

SynchronizeArtifactCache 메소드는, IS가 각각의 아티펙트 제공자의 SynchronizeArtifactCache 메소드를 인보크함으로써 아티펙트 캐시를 동기화시키도록 한다.

```
void SynchronizeArtifactCache(DateTime from, DateTime to)
```

SynchronizeArtifactCache 메소드는 날짜 범위에 의해 제약될 수 있다. 이러한 경우에, 각각의 아티펙트 소비자는 "from"과 "to" 사이에("from", "to" 포함) 아티펙트에 대한 CacheArtifact 이벤트를 발행할 의무만 있다.

클라이언트 API - IClientLinking

링크에 관련된 IS 클라이언트 API는 단일 메소드로 구성된다. 이 메소드의 함수는 툴의 디폴트 액션을 인보크하여 아티펙트를 디스플레이한다.

ExecuteDefaultAction

```
bool ExecuteDefaultAction(string artifactUri)
```

ExecuteDefaultAction는 단일 아티펙트 Id를 입력으로서 취한다. 클라이언트는, 다음과 같은 디폴트 액션을 핸들링하기 위하여 등록된 메커니즘 및 아티펙트 유형에 기초하여 어떤 액션이 실행될지를 결정한다. 아티펙트 유형이 등록된 제공자 클라이언트와 연관되면, 제공자 클라이언트의 ExecuteDefaultAction이 인보크된다. 그렇지 않으면, 아티펙트 Id는 외부 어드레싱가능성 메커니즘을 사용하는 해결법을 위해 IS 서버에게 보내진다. 성공적인 경우, IDE 내에 페이지가 열리게 된다. 성공적이지 못한 경우, 사용자는 http 404 에러를 볼 것이다.

일반 아티펙트 제공자

이제 도 5를 참조하면, 본 발명에 따른 GAP API의 일 구현의 블럭도가 도시되어 있다. GAP은 본질적으로 표준 아티펙트 제공자 및 소비자 메소드를 제공하는 데이터베이스 어플리케이션이다. 하나 이상의 IS-인에이블된 툴(500), 및 일부 컨텐츠가 통합된 툴로서 액세스되기를 원하는 논 IS 인에이블된 툴이 제공된다. IS-인에이블된 툴(500)과의 상호작용을 용이하게 하기 위하여, GAP 데이터베이스(506)와 연관된 GAP(504)가 사용된다. GAP 데이터베이스(506)는 도 7과 관련하여 후술되는 일반 아티펙트 데이터베이스 스키마(GADS)의 포맷으로 구성된다. 논 IS 인에이블된 툴과 상호작용하기 위해, GAP 어댑터(508)를 기록하여, 본래의 툴(502)에 최소한의 혼란만을 주면서 툴 데이터가 IS 아티펙트로서 노출되고 IS 아티펙트로의 링크를 홀딩할 수 있게 하는 것이 가능하다.

GAP는, 특히 아티펙트를 홀딩하는 툴 내에서 아티펙트 제공자/소비자 의무를 직접적으로 만족시키는 것이 불가능하거나 어려운 상황을 해결하기 위한 것이다. 예를 들어, 팀은 공유 서버에 저장된 문서들이 마치 완성된 아티펙트(full-fledged artifact)인 것처럼, 그 문서들로부터 및 그 문서들로 링크하기를 원한다. 유감스럽게도, 공유 서버는 IS-인에이블된 API를 구현하지 않으므로, 그것의 아티펙트에 대해 IS-순응 URI를 제공하지 않거나, 또는 요구된 메소드(예를 들어, GetArtifacts/GetReferencingArtifacts)를 제공하지 않는다. GAP를 사용하여, 팀은 공유 문서에 대한 "아티펙트 프록시"를 구성할 수 있다. 이러한 프록시는 다른 아티펙트에 의해 참조될 수 있고, 그것을 나타내는 실제 공유 문서를 모두 대신하여 다른 아티펙트로의 링크를 홀딩할 수 있다.

GAP는 소스 코드 또는 파일 시스템에서 아티펙트 컨테이너가 파일로서 홀딩되는 상황을 해결하기 위한 것이다. 이러한 경우, 아티펙트를 생성할 책임이 있는 툴은 그 지속성에 발언권이 없으므로, 소스 제어 및 파일 시스템이 아티펙트로부터의 링크를 인식하고 저장할 것을 강요할 수 없다. 예를 들어, 하나의 툴은 모델과 그 모델로부터 생성된 소스 코드 간의 링크를 홀딩하기를 원하며, 여기서 모델은 소스 코드로서 저장된다. GAP를 사용하여, 툴은 아티펙트 프록시를 작성하여 이러한 링크들을 홀딩할 수 있다. 또한, 추가적 아티펙트 프록시는 툴 모델의 XML 웹 서비스 또는 소스 파일의 메소드와 같이, 파일 내부의 아티펙트를 나타내도록 작성될 수 있다.

이제 도 6을 참조하면, GAP 사용에 대한 비통합 툴을 준비하는 일 구현의 흐름도가 도시되어 있다. 블럭(600)에서, 하나 이상의 비통합 툴이 수신된다. 블럭(602)에서, 하나 이상의 비통합 툴이 본 발명에 따른 GAP를 사용하여 아티펙트 및 참조를 노출하기 위해 등록시에 준비된다. 블럭(604)에서, 툴 유형이 등록되어, GAP 서비스에 매핑하는 각각의 GAP-관리 아티펙트에 대한 아티펙트 제공자로서 활동한다. 자가-관리 및 GAP-관리 아티펙트를 포함하는 제공자에 대해서, 각각에 대한 툴 유형은 반드시 구별되어야 한다. 블럭(606)에서, 각각의 GAP-관리 아티펙트에 대해서, 만약 있다면, 그 아티펙트 유형 및 아웃바운드 링크 유형이 정의된다. 그 후 프로세스는 종료 블럭에 도달한다.

이제, 도 7을 참조하면, GAP와 캐시 둘다에 대해 사용되는 일반 아티펙트 데이터베이스 스키마(GADS)가 도시된다. GADS(700)는 정확하게 하나의 디스플레이가능한 아티펙트를 포함하는 아티펙트 구성요소(702)를 포함하며, 아티펙트 구성요소(702)는 리턴된 아티펙트의 URI, 리턴된 아티펙트의 유형의 국부화된 익숙한 네임인 아티펙트 타이틀, (가능하다면) 최종 변경 날짜 및 시간(LastChangedOn), (가능하다면) 이 아티펙트를 변경하는 최종 사람의 userId, 구성요소가 IS

일반 아티펙트 제공자에 의한 사용을 위해 예약되고, IS 및 그것의 에이전트가 직접 아티펙트 제공자를 갖지 않은 항목에 액세스할 수 있도록 하는 정보를 포함하는 ExternalID, 및 캐시에서 사용되고 인스턴스-특정 동기화 상태를 홀딩하는 선택 비트 속성 SyncFlag와 같은 하위구성요소들을 포함한다.

GADS(700)는 참조 링크 URI(ReferringURI), 링크 유형(LinkType) 및 피참조 링크 URI(ReferencedURI)를 포함하는 아티펙트 구성요소(702)의 링크 하위구성요소(704)도 포함한다. 아티펙트 구성요소(702)의 확장된 속성 하위구성요소(706)는 제공자가 제공할 수 있는 선택적 아티펙트-특정 속성을 설명하는 네임/값 쌍의 컬렉션을 포함한다.

이제, 도 8을 참조하면, 본 발명에 따른 GAP를 구현하는 한 방법의 흐름도가 도시된다. 블럭(800)에서, 제공자는 노출된 각각의 아티펙트에 대한 아티펙트 유형 정보를 등록한다. 블럭(802)에서, 제공자는 자신이 호스트하는 각각의 아티펙트에 대한 링크 유형을 등록한다. 블럭(804)에서, 등록 정보가 국부적으로 또는 고유하게(natively) 저장될 수 있는지가 결정된다. 저장될 수 없는 경우, 흐름은 블럭(806)으로 진행하여, GAP가 사용된다. 블럭(808)에서, 데이터는 GAP 데이터베이스에 대하여 처리된다. 블럭(810)에서, 비통합 툴은 이제 IS-인에이블된 툴과 접속할 수 있다. 그 다음, 프로세스는 종료 블럭에 도달한다. 그러나, 등록 정보가 고유하게 저장될 수 있는 경우, 흐름은 블럭(804)에서 블럭(812)으로 진행하여, GAP 없이 처리된다. 그 다음, 흐름은 종료 블럭으로 진행한다.

이제, 도 9를 참조하면, 본 발명에 따른 GAP 어댑터를 구현하는 한 방법의 흐름도가 도시된다. 아티펙트가 수정될 때, GAP 데이터베이스 내에는 각 아티펙트마다 하나의 레코드가 있다. 어댑터를 구성하기 위해서는, 기초 툴에 저장된 아티펙트 데이터의 변경을 탐지할 수 있어야만 한다.

블럭(900)에서, 항목은 툴에 추가되어 아티펙트로서 나타난다. 블럭(902)에서, 아티펙트에 대한 URI가 생성된다. 아티펙트로서 나타내질 항목이 툴에 추가될 때, 블럭(904)에 나타낸 바와 같이, 툴 어댑터는 GAP UpdateArtifactData 메소드를 사용하여 GAP 내의 대응하는 아티펙트 레코드를 생성한다. 이 레코드는 이 아티펙트 URI를 툴에 알려진 식별자(소위 그것의 외부Id)에 매핑한다. 예를 들어, 파일 공유에 저장된 문서에 대한 외부 Id는 그것의 폴더 계층구조 및 파일 네임일 수 있다.

아티펙트 레코드는, 예를 들어 아티펙트의 URI, 타이틀, 외부Id 및 확장된 속성을 포함한다는 점에서 GAS를 따른다. 블럭(906)에서, 시스템은 툴의 아티펙트 데이터의 수정을 탐지하도록 동작한다. 블럭(908)에서, 시스템은 수정이 발생하는지를 결정한다. 수정이 발생하지 않은 경우, 흐름은 블럭(906)으로 되돌아가, 수정을 계속 모니터링한다. 수정이 발생한 경우, 흐름은, 항목 "변경"에 대하여, 툴 어댑터가 GAP UpdateArtifactData 메소드를 사용하여 수정을 포스트하는 블럭(910)으로 진행한다. GAP 아티펙트로의 링크와 GAP 아티펙트로부터의 링크의 통합을 유지하기 위하여, 삭제/추가와 변경을 구별하는 것이 중요하다는 것에 유의해야 한다. 블럭(912)에서, 수정이 "추가/삭제"인 경우, 어댑터는 대응하는 레코드를 GAP 데이터베이스에 추가하거나 GAP 데이터베이스로부터 삭제한다. 아티펙트에 관련된 항목이 삭제될 때, 툴 어댑터는 GAP UpdateArtifactData 메소드를 사용하여 GAP로부터 대응하는 레코드를 삭제한다. 아티펙트로부터의 링크가 생성되거나 수정될 때, UpdateArtifactData 메소드가 사용된다. 블럭(914)에서, 링크 생성 및 수정이 처리된다. 그 다음, 프로세스는 종료 블럭에 도달한다. 외부 어드레스가능성을 핸들링하기 위해, 아티펙트 핸들러가 정의되고 등록된다.

GAP API

GAP API는 아티펙트 제공자 및 아티펙트 소비자 둘다이므로, GAP API는 표준 서버 제공자 및 소비자의 메소드 및 거동 각각(즉, IToolServerLinking 인터페이스의 모든 메소드)을 제공한다. 추가적으로, GAP API는 자신이 홀딩하는 아티펙트 데이터의 유지를 가능하게 하기 위하여, 다음과 같은 메소드를 제공한다.

```
bool UpdateArtifactData(Artifact[] artifacts)
```

호출자는 아티펙트 어레이 객체를 파퓰레이트하고, UpdateArtifactData를 인보크하여 GAP 데이터베이스를 갱신한다. 아티펙트 어레이는 GAP 데이터베이스에 홀딩될 때, 확장된 속성 및 링크를 포함하는 전체 아티펙트 설명을 포함한다. 각각의 아티펙트는 자신의 ChangeType 속성(추가, 변경, 삭제)에 대한 유효값을 가져야 한다.

추가시: 아티펙트의 Uri에 대한 <tool-specific id>가 특수값 "<?>"로 설정되면, GAP는 고유 ID를 생성하고 그것의 Artifact.Uri 속성에서 관련 Uri를 대체시킨다. 특수값으로 설정되지 않는 경우, <tool-specific id>의 값이 직접적으로 사용된다. 인입 Artifact.Uri 값에 의해 키잉된 레코드가 GAP에 이미 존재할 때, 아티펙트를 추가하려고 시도하면, 에러가 발생한다.

변경 또는 삭제시, 인입 Artifact.Uri 값에 의해 키잉된 레코드는 데이터베이스에 있어야만 하며, 그렇지 않으면, 에러가 발생한다.

```
Artifact[] GetArtifactsByExternalId(string[] externalId)
```

이 메소드는 GAP 아티펙트의 URI와 일치하는 외부 Id에 기초하여 GAS-순종 아티펙트를 리턴하는 데 사용된다. 아티펙트 어레이가 URI 및 외부 Id 둘다를 포함하므로, 표준 GetArtifacts 메소드는 URI가 주어진 때 외부 Id를 결정하는 데 사용될 수 있으며, 이 메소드는 외부 Id가 주어진 때 URI가 결정되도록 할 수 있다.

이제, 도 10을 참조하면, 본 발명에 따라서 비통합 툴의 액세스를 용이하게 하는 시스템(1000)이 도시된다. 시스템(1000)은 수개의 비통합 툴, 즉, 제1 툴(1002), 제2 툴(1004) 및 제3 툴(1006)의 사용을 도시하고 있다. 제1 툴(1002)은, GAP(1008)의 구현이 IS-인에이블된 툴(1010)로의 아티펙트 제공자로서 인에이블될 것을 요구한다. 이를 지원하여, 인에이블된 툴(1010)에 하나 이상의 아티펙트를 노출하는 것을 용이하게 하는 툴-특정 어댑터(1012)가 생성된다. 링크 및 참조는 GADS를 따르는 GAP 데이터베이스(506)에 저장될 수 있다.

제2 툴(1004)은 고유 인터페이스 상에 아티펙트를 공개적으로 노출시키는 제공자 인에이블먼트 API(1014)를 통합한다. 제3 툴(1006)은 고유 인터페이스의 상에 제2 툴(1004)의 제공자 아티펙트 및 제1 툴(1002)의 아티펙트에 대한 참조를 노출시키는 소비자 인에이블먼트 API(1016)를 통합한다.

시스템(1000)은 본 발명에 따라 아티펙트 및 링크 정보를 캐싱하는 아티펙트/링크 캐시(1018)를 더 포함한다. 링크 관리자(1020)는 캐시(1018)를 관리한다. 캐시(1018)는 적어도 제1, 제2, 및 제3 툴(1002, 1004, 1006)과 접속하여, 적어도 아티펙트, 링크 정보 및 여기 설명된 다른 정보를 캐싱하는 것을 용이하게 한다.

본 발명은 (예를 들어, 선택과 관련하여) 본 발명의 다양한 자동 프로세스를 수행하는, 예를 들어 분류자(1022)(선택사항) 형태의 다양한 아티펙트 지능 기반 스키마를 사용할 수 있다. 여기에 설명되는 구현은 일반적으로 한번의 홉(hop)으로만 다루어진다. 즉, 툴은 다른 툴에 직접 인터페이스한다. 분류자(1022)는 보다 복잡한 쿼리를 용이하게 하는데 사용되고, 한번의 홉을 넘어 확장된 링크 설명이 유도될 수 있다.

분류자는 입력 속성 벡터, x=(x1, x2, x3, x4, xn)를, 한 입력이 클래스에 속할 신뢰도에 매핑하는 함수, 즉 f(x) =confidence(class)이다. 이러한 분류는, 사용자가 자동적으로 수행되기를 원하는 액션을 추론 또는 예지하기 위하여, 확률 및/또는 통계 기반 분석(예를 들어, 분석 유틸리티 및 비용으로 인수분해)을 사용할 수 있다.

사용될 수 있는 분류기의 일례로는 써포트 벡터 머신(SVM)이 있다. SVM은 가능한 입력들의 공간에서, 트리거되지 않은 이벤트로부터 트리거되는 기준을 분할하려고 시도하는 초표면(hypersurface)를 찾음으로써 동작한다. 직관적으로, 이것은 트레이닝 데이터에 가깝지만 완전히 동일하지는 않은 테스팅 데이터에 대한 분류를 정확하게 해준다. 다른 지향성 및 무지향성 모델 분류 접근법에는, 예를 들어, 네이브 베이즈(naive Bayes), 베이지안 네트워크(Bayesian network), 결정 트리가 포함되며, 상이한 독립성의 패턴들을 제공하는 확률 분류 모델들이 사용될 수 있다. 본 명세서에서의 분류는 또한 우선 순위의 모델들을 개발하는 데 사용되는 통계적 회귀를 포함한다.

본 명세로부터 쉽게 이해되는 바와 같이, 본 발명은 (예를 들어, 일반적으로 트레이닝 데이터를 통해) 명시적으로 트레이닝되는 분류기는 물론, (예를 들어, 사용자의 거동을 관측하거나 외부 정보를 수신하는 것을 통해) 암시적으로 트레이닝되는 분류기를 사용할 수 있다. 예를 들어, SVM은 분류기 구축자 및 특징 선택 모듈 내에서의 위상 학습 또는 트레이닝 단계를 통해 구성된다. 따라서, 식별자(들)는, 어떤 아티펙트가 노출될지, 어떤 우선순위가 아티펙트를 노출하는 데 적용될지, 어느 툴이 다른 툴보다 먼저 노출되어야 할지, 예를 들어 데이터 유형, 사용자, 데이터의 위치에 기초하여 어떤 아티펙트가 노출될 수 있는지를, 소정의 기준에 따라 결정하는 것 등을 비롯한 수많은 기능을 자동적으로 수행하는 데 사용될 수 있다.

아티펙트/링크 캐시

아티펙트/링크 캐시는 아티펙트들과 그들이 홀딩하는 링크가 복제되어 고성능 쿼리 액세스를 가능하게 하는 중앙화된 저장장치이다. 간단한 아티펙트 및 링크 데이터를 단일 데이터베이스에 함께 모음으로써, 서비스간 참조 역전은 보다 효율적으로 처리될 수 있다.

이제, 도 11을 참조하면, 본 발명에 따른 캐시의 일 구현에 대한 프로세스의 흐름도가 도시된다. 블럭(1100)에서, 아티펙트/링크 관계를 저장하기 위하여 캐시가 사용된다. 블럭(1102)에서, GAS를 따라서 관계가 포맷된다. 블럭(1104)에서, 아티펙트 제공자에 의해 일어나는 ArtifactChanged 이벤트에 응답하여 캐시가 갱신된다. IS 캐시 관리자는 ArtifactChanged에 가입하고 이벤트 객체(GAS에 따르는 객체)를 사용하여, 자신을 갱신한다. 캐시가 실제 아티펙트 갱신에 비동기적으로 갱신되므로, 캐시는 기초 저장장치에 대하여 아웃-오브-싱크(out-of-sync)일 수 있다. 그러나, 지연은 짧고, 대부분의 어플리케이션에서 수용될 수 있다. 제로 지연이 요구되는 경우, 툴은 캐시를 바이패싱하고, 그것의 표준 GetArtifacts/GetReferencingArtifacts 메소드를 통해 툴에 직접적으로 요구할 수 있다. 캐시는 일반 아티펙트 데이터베이스 스키마를 따르는 데이터베이스에 저장된다. GAP 데이터베이스가 매핑 정보 및 링크의 마스터 카피인데 반하여, 캐시 데이터베이스는 어느 것의 마스터도 아니고 완전히 다시 생성될 수 있다는 점에서, 캐시는 GAP 데이터베이스와 다르다.

다시 도 11을 참조하면, 블럭(1106)에서, 캐시가 기초 데이터에 동기화되었는지에 대한 결정이 행해진다. 네트워크 또는 시스템 실패로 인해, 캐시와 기초 데이터가 영구적으로 아웃-오브-싱크 상태로 되는 경우, IS는 캐시를 복구하기 위하여 각각의 아티펙트 제공자와 협동하는 운용 SynchronizeArtifactCache 유틸리티를 제공한다. 이 유틸리티는 각각의 아티펙트 제공자의 SynchronizeArtifactCache 메소드를 인보크한다. 동기화가 요구되지 않는 경우, 흐름이 블럭(1106)에서 블럭(1104)로 진행하여 캐시가 갱신된다. 동기화가 요구되는 경우, 흐름은 블럭(1106)에서 블럭(1108)로 진행하여 동기화가 인보크된다. 캐시 동기화는 다음과 같이 거동한다. IS SynchronizeArtifactCache 유틸리티가 (프로그램적으로 또는 운용 함수를 통해) 인보크된다. 캐시 동기화가 이미 진행되고 있으면, 경고가 제공된다. 그러나, 경고는 오버라이드될 수 있다. IS 운용 데이터베이스에서, 상태 "글로벌 동기화 진행중"이 설정된다.

블럭(1110)에서, 동기화된 각각의 캐싱된 아티펙트(데이터 범위에 기초)는 "캐싱됨"이란 플래그로 마킹된다. 블럭(1112)에서, 마킹된 아티펙트를 가진 각각의 제공자가 접촉된다. 아티펙트 제공자의 SynchronizeArtifactCache 메소드가 인보크되고, 상태 "<tool-instance> 동기화 진행중"을 운용 데이터베이스에 설정한다. 블럭(1114)에서, 아티펙트 제공자의 SynchronizeArtifactCache 메소드는 각각의 후보 아티펙트에 대한 (또는 후보 아티펙트의 세트에 대한) CacheArtifact 이벤트를 발행한다. 블럭(1116)에서, 이벤트되고 마킹된 아티펙트가 캐시에 업로드된다. 아티펙트 제공자가 모든 후보 아티펙트에 대한 CacheArtifact 이벤트를 발행한 경우, 그것은 자기 자신의 툴 인스턴스를 매개변수로 하여 EndCacheLoad 이벤트를 점화한다. IS 캐시 관리자는 ArtifactChanged 이벤트 및 CacheArtifact 이벤트를 듣는다. 2가지 이벤트는 동일한 방식으로 처리된다. 즉, 이벤트 상의 아티펙트 매개변수에 포함된 각각의 아티펙트가 캐시 내에 위치된다.

블럭(1118)에서, 캐시 내에 동일한 URI를 가진 아티펙트가 존재하면, 그 아티펙트는 대체된다. 그렇지 않으면, 그 아티펙트는 추가된다. 대체되거나 추가된 엔트리는 "캐싱됨" 플래그 세트를 갖지 않는다. IS 캐시 관리자는 EndCacheLoad 이벤트를 듣는다. 수신되면, 블럭(1120)에서, 캐시 클린업이 수행되어, "캐싱됨"으로 마킹된 EndCacheLoad에 의해 구체화된 툴 인스턴스에 대한 캐시 내의 모든 아티펙트가 삭제된다. 이에 의해, 대응 아티펙트가 주 저장장치로부터 삭제된 임의의 캐싱된 항목이 제거된다. IS는 "<tool-instance> 동기화 진행중" 상태를 턴오프한다. 최종 제공자의 "<tool-instance> 동기화 진행중" 상태가 턴오프되면, "글로벌 동기화 진행중" 상태가 턴오프된다. 그 다음, 프로세스는 종료 블럭에 도달한다.

API참조

본 섹션은 해당 메소드의 구현으로의 참조를 갖는, 문서의 임의의 위치에 정의된 모든 메소드 서명을 포함한다. 추가적으로, IS에 의해 제공된 메소드에 대한 과부하는 메시지 한정자(Message Qualifier)로 인해 발생된다. ASP.Net 규약에 의해, 과부하된 메소드의 MessageName 속성은 이하에 설명되는 바와 같이 메소드 네임과 메시지 한정자를 연관시킴으로써 형성된다. 이러한 메시지 네임은 서비스의 소비자에게 가시화되지 않는다.

다음의 코드는 메시지 네임이 메소드 네임(본 경우에서는 GetArtifact)에 디폴트인 것을 나타낸다.

```
[WebMethod]
public ArtifactID GetArtifacts(string[] ArtifactUri)
{ /* Get Artifacts; use cache if possible */ }
```

다음에서, 메시지 네임은 명시적으로 지정된다. 규약에 의하면, 이것은 메소드 네임을 서명 테이블로부터의 메시지 한정자에 연쇄시킴으로써 형성된다. 이 경우에, 메시지 한정자는 Direct이므로, 메시지 네임은 GetArtifactDirect이다.

```
[WebMethod(MessageName="GetArtifactsDirect")]
public ArtifactID GetArtifacts(string[] ArtifactUri)
{ /* Get Artifacts directly from artifact provider */ }
```

스키마 및 데이터 유형

일반 아티펙트 스키마(GAS) 및 아티펙트 클래스: GAS는 디스플레이가능한 아티펙트에 대한 일반 포맷을 정의하는 XML 스키마이다. 스키마가 아래에 나타나고, 그것의 구성요소의 각각에 대한 설명이 후술된다. GAS 순응 문서가 객체로 탈직렬화되면, 객체는 "아티펙트 클래스"에 나타난 형태를 취한다. API에서, 아티펙트 객체가 결코 나타나지 않는 대신, 아티펙트 객체의 어레이가 사용됨에 유의해야 한다.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema
targetNamespace="http://www.company.com/Tool/GenericArtifactSchema.xsd"
    elementFormDefault="qualified" xmlns="http://www.company.com/ Tool
/GenericArtifactSchema.xsd"
    xmlns:mstns="http://www.company.com/ Tool /GenericArtifactSchema.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <!-- By declaring ISUri a simple type we can add validation later. -->
    <xs:simpleType name="ISUri">
        <xs:restriction base="xs:string" />
    </xs:simpleType>
    <!-- Enumeration for usage in ArtifactChanged event. -->
    <xs:simpleType name="ChangeType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Add" />
            <xs:enumeration value="Change" />
            <xs:enumeration value="Delete" />
            <xs:enumeration value="NoChange" />
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="Artifacts">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Artifact" type="Artifact" minOccurs="1"
maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="Artifact">
        <xs:sequence>
            <xs:element name="Uri" type="ISUri" />
            <xs:sequence minOccurs="0" maxOccurs="1">
                <xs:element name="ArtifactTitle" type="xs:string" />
                <xs:element name="ArtifactTypeName" type="xs:string" />
                <xs:element name="LastChangedOn" type="xs:dateTime" />
                <xs:element name="LastChangedBy" type="xs:string" />
```

```
                <xs:element name="ExternalId" type="xs:string"
minOccurs="0" maxOccurs="1" />
                <xs:element name="ExtendedAttributes" minOccurs="0"
maxOccurs="1">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="ExtendedAttribute"
type="ExtendedAttribute" minOccurs="0"

      maxOccurs="unbounded" />
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="OutboundLinks" minOccurs="0"
maxOccurs="1">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="OutboundLink"
type="OutboundLink" minOccurs="0"
                                maxOccurs="unbounded" />
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:sequence>
        <xs:attribute name="ChangeType" type="ChangeType"
default="NoChange" use="optional" />
    </xs:complexType>
    <xs:complexType name="ExtendedAttribute">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" />
            <xs:element name="Value" type="xs:string" />
            <xs:element name="FormatString" type="xs:string" minOccurs="0"
/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="OutboundLink">
        <xs:sequence>
            <xs:element name="LinkType" type="xs:string" />
            <xs:element name="ReferencedUri" type="ISUri" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

구성요소 아티펙트들

1:m 아티펙트 구성요소들을 포함하며, 각각의 아티펙트 구성요소는 일반 IS 아티펙트를 설명한다. 인입 아티펙트 Id마다 하나의 아티펙트 구성요소가 아티펙트 구성요소에 나타날 것이다.

구성요소 아티펙트

다음과 같은 하위구성요소로 구성된 정확하게 하나의 디스플레이가능한 아티펙트를 포함한다.

- Uri: 리턴된 아티펙트의 Uri.

- ArtifactTitle: 리턴된 아티펙트의 국부화된 익숙한 네임.

- ArtifactTypeName: 리턴된 아티펙트의 유형의 국부화된 익숙한 네임.

- LastChangedOn: 가능하면, 최종 변경의 날짜 및 시간.

- LastChangedBy: 가능하면, 이 아티펙트를 최종적으로 변경한 사람의 사용자Id.

- External Id: 이 구성요소는 IS 일반 아티펙트 제공자에 의한 사용을 위해 보존된다. IS 및 그 에이전트가 어떠한 직접 아티펙트 제공자도 갖지 않는 항목을 액세스하도록 하는 정보를 포함한다.

- ExtendedAttributes: 제공자가 제공할 수 있는 선택적 아티펙트-특정 속성을 설명하는 네임/값 쌍의 집합.

- OutboundLinks: 이 아티펙트로부터의 아웃바운드 링크를 설명하는 링크 유형/참조되는 Uri 쌍의 집합.

- ChangeType은 오직 ArtifactChanged 이벤트의 문맥에서만 의미있는 아티펙트 구성요소 상의 선택적 속성이다.

아티펙트 클래스

```
//------------------------------------------------------------------------
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.1
//
//     Changes to this file may cause incorrect behavior and will be
//    lost if the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------
//
// This source code was auto-generated by xsd, Version=1.1
//
using System.Xml.Serialization;

[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://www.compan
y.com/Tool/GenericArtifactSchema.xsd")]
[System.Xml.Serialization.XmlRootAttribute(Namespace="http://www.
company.com/Tool/GenericArtifactSchema.xsd", IsNullable=false)]
public class Artifacts {

    [System.Xml.Serialization.XmlElementAttribute("Artifact")]
    public Artifact[] Artifact;
}

[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://www.
company.com/Tool/GenericArtifactSchema.xsd")]
public class Artifact {


  public string Uri;
  public string ArtifactTitle;
  public string ArtifactTypeName;
  public System.DateTime LastChangedOn;
  public string LastChangedBy;
  public string ExternalID;

  [System.Xml.Serialization.XmlArrayItemAttribute(IsNullable=false)]
  public ExtendedAttribute[] ExtendedAttributes;

  [System.Xml.Serialization.XmlArrayItemAttribute(IsNullable=false)]
  public OutboundLink[] OutboundLinks;

  [System.Xml.Serialization.XmlAttributeAttribute()]
  public ChangeType ChangeType = ChangeType.NoChange;
}

[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://www.
company.com/Tool/GenericArtifactSchema.xsd")]
public class ExtendedAttribute{

    public string Name;
    public string Value;
    public string FormatString;
}

[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://www.
company.com/Tool/GenericArtifactSchema.xsd")]
public class OutboundLink{

    public string LinkType;
    public string ReferencedUri;
}

[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://www.
company.com/Tool/GenericArtifactSchema.xsd")]
public enum ChangeType {

    Add,
    Change,
    Delete,
    NoChange,
}
```

일반 링크 스키마(GLS) 및 링크 클래스

GLS는 아티펙트 링크에 대한 일반 포맷을 정의하는 XML 스키마이다. 그것은 ExtractLinks 메소드에 의해 리턴된다. 스키마가 아래에 나타나며, 그것의 구성요소 각각이 후술된다. GLS 순응 문서가 객체로 탈직렬화될 때, 객체는 "링크 클래스" 섹션에 나타난 형태를 취한다.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="ISGenericLinkSchema"
targetNamespace="http://tempuri.org/ISGenericLinkSchema.xsd"
 elementFormDefault="qualified"
xmlns="http://tempuri.org/ISGenericLinkSchema.xsd"
 xmlns:mstns="http://tempuri.org/ISGenericLinkSchema.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="Links">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Link" type="Link" minOccurs="1"
maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="Link">
        <xs:sequence>
            <xs:element name="ReferringUri" type="xs:string" />
            <xs:element name="LinkType" type="xs:string" />
            <xs:element name="ReferencedUri" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

구성요소 링크들

1:m 링크 구성요소들을 포함하며, 각각의 링크 구성요소는 2개의 IS 아티펙트 간의 링크를 기술한다.

구성요소 링크

2개의 아티펙트 간의 하나의 링크를 설명한다:

- ReferringUri: 참조(소비자) 아티펙트의 Uri.

- LinkType: 링크의 유형.

- ReferencedUri: 피참조(제공자) 아티펙트의 URI.

Link 클래스

```
//------------------------------------------------------------------------
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.1
//
//     Changes to this file may cause incorrect behavior and will be
//    lost if the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------
//
// This source code was auto-generated by xsd, Version=1.1
//
using System.Xml.Serialization;


[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://tempuri.or
g/ISGenericLinkSchema.xsd")]
[System.Xml.Serialization.XmlRootAttribute(Namespace="http://tempuri.or
g/ISGenericLinkSchema.xsd", IsNullable=false)]
public class Links {

    [System.Xml.Serialization.XmlElementAttribute("Link")]
    public Link[] Link;
}

[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://tempuri.or
g/ISGenericLinkSchema.xsd")]
public class Link {

    public string ReferringUri;
    public string LinkType;
    public string ReferencedUri;
}
```

LinkFilter 클래스

LinkFilter 클래스는 쿼리 메소드에서 결과를 필터링하는 데에 사용된다. 각각의 아티펙트 소비자는 자신의 GetReferencingArtifats 구현에서의 필터를 존중해야 한다.

```
public class LinkFilter
{
      public enum filterType
      {
            ToolInstance,
            ArtifactType,
            LinkType
      }

      public filterType Type;
      public string[] Values;
}
```

ArtifactId 클래스

ArtifactId 클래스는 EncodeUri 및 DecodeUri 메소드를 사용하는 아티펙트의 Uri를 형성하는 데 사용되는 데이터를 수집하는 데에 사용된다.

```
public class ArtifactId
{
      public string VisualStudioServerNamespace;
      public string Tool;
      public string ArtifactType;
      public string ToolSpecificId;
}
```

일반 아티펙트 데이터베이스 스키마

일반 아티펙트 제공자 및 아티펙트/링크 캐시 둘다에 대해 사용되는 일반 아티펙트 데이터베이스 스키마가 후술된다. 필수적으로, 그것은 제공자 네임 및 아티펙트 유형이 표시되지 않는다는 예외를 갖는 GAS 스키마의 데이터베이스 구현이다. 그들은 아티펙트 인스턴스가 판독될 때 URI로부터 자동으로 유도되고, 아티펙트 인스턴스가 기록될 때 무시된다. 선택 비트 속성 "Sync Flag"는 오직 캐시에서만 사용되고, 인스턴스-특정 동기화 상태를 홀딩한다.

예시적인 컴퓨팅 시스템

이제, 도 12를 참조하면, 개시된 아키텍처를 실행하도록 동작가능한 컴퓨터의 블럭도가 도시된다. 본 발명의 다양한 양상에 대한 추가적 문맥을 제공하기 위해, 도 12 및 이하의 설명은 본 발명의 다양한 양상이 구현될 수 있는 적절한 컴퓨팅 환경(1200)의 간략하고 개괄적인 설명을 제공하기 위한 것이다. 본 발명이 하나 이상의 컴퓨터 상에서 작동될 수 있는 컴퓨터 실행가능 명령어의 일반적 문맥으로 상술되었지만, 본 기술에 숙련된 자들은 본 발명이 또한 다른 프로그램 모듈들의 조합 및/또는 하드웨어 및 소프트웨어의 조합으로도 구현될 수 있음을 인식할 것이다.

일반적으로, 프로그램 모듈은 특정 태스크를 수행하거나 특정 추상 데이터 유형을 구현하는 루틴, 프로그램, 컴포넌트, 데이터 구조 등을 포함한다. 또한, 본 기술에 숙련된 자들은, 본 발명이 하나 이상의 연관된 디바이스에 각각 동작적으로 연결될 수 있는 단일-프로세서 또는 멀티 프로세서 컴퓨터 시스템, 미니컴퓨터, 메인프레임 컴퓨터, 개인용 컴퓨터, 핸드헬드 컴퓨팅 디바이스, 마이크로프로세서-기반 또는 프로그램 가능 가전제품 등을 포함하는 다른 컴퓨터 시스템에서 구현될 수 있음을 알 것이다.

본 발명의 도시된 양상은 또한 특정 태스크가 통신 네트워크를 통해 링크된 원격 프로세싱 디바이스에 의해 수행되는 분산 컴퓨팅 환경에서 구현될 수도 있다. 분산 컴퓨팅 환경에서, 프로그램 모듈은 로컬 메모리 저장 장치 및 원격 메모리 저장 장치에 위치될 수 있다.

컴퓨터는 전형적으로 다양한 컴퓨터 판독가능 매체를 포함한다. 컴퓨터 판독가능 매체는 컴퓨터에 의해 액세스될 수 있는 임의의 사용가능 매체일 수 있으며, 휘발성 및 비휘발성 매체, 분리형 및 비분리형 매체를 모두 포함한다. 예를 들어, 컴퓨터 판독가능 매체는 컴퓨터 저장 매체 및 통신 매체를 포함할 수 있다. 컴퓨터 저장 매체는 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 다른 데이터 등의 정보의 저장에 대한 임의의 방법 또는 기술로 구성된 휘발성 및 비휘발성 매체, 분리형 및 비분리형 매체를 모두 포함한다. 컴퓨터 저장 매체는 RAM, ROM, EEPROM, 플래시 메모리, 다른 메모리

기술, CD-ROM, 디지털 비디오 디스크(DVD), 다른 광학 디스크 저장 장치, 자기 카세트, 자기 테이프, 자기 디스크 저장 장치, 다른 자기 저장 디바이스, 또는 원하는 정보를 저장하는데 사용될 수 있고 컴퓨터에 의해 액세스될 수 있는 임의의 다른 매체를 포함하지만 이에 한정된 것은 아니다.

통신 매체는 전형적으로 반송파 또는 다른 전송 메커니즘 등의 변조된 데이터 신호에 컴퓨터 판독가능 명령, 데이터 구조, 프로그램 모듈, 또는 다른 데이터를 구현하고, 임의의 정보 전달 매체를 포함한다. "변조된 데이터 신호"라는 용어는 신호 내에 정보를 인코딩하는 방식으로 설정되거나 변경된 특성을 하나 이상을 갖는 신호를 의미한다. 예를 들어, 통신 매체는 유선 네트워크 또는 직접-유선 접속 등의 유선 매체와, 음향, RF, 적외선 및 기타 무선 매체 등의 무선 매체를 포함하지만, 이에 한정되지 않는다. 상술한 것들 중의의 임의의 조합이 또한 컴퓨터 판독가능 매체의 범위 내에 포함되어야 한다.

도 12를 다시 참조하면, 프로세싱 유닛(1204), 시스템 메모리(1206) 및 시스템 버스(1208)를 포함하는 컴퓨터(1202)를 포함하는 본 발명의 다양한 양상을 구현하는 예시적 환경(1200)이 도시된다. 시스템 버스(1208)는 시스템 메모리(1206)를 포함하는 시스템 컴포넌트 등을 프로세싱 유닛(1204)에 접속시킨다. 프로세싱 유닛(1204)은 통상적으로 사용가능한 다양한 프로세서 중 임의의 것일 수 있다. 듀얼 마이크로프로세서 및 다른 멀티프로세서 아키텍처도 프로세싱 유닛(1204)으로 사용될 수 있다.

시스템 버스(1208)는 메모리 버스(메모리 제어기를 가질 수도 있고 갖지 않을 수도 있음), 주변 버스 및 통상적으로 사용가능한 다양한 버스 아키텍처 중 임의의 것을 사용하는 로컬 버스에 더 상호연결할 수 있는 버스 구조의 몇몇 유형 중 임의의 것일 수 있다. 시스템 메모리(1206)는 ROM(1210) 및 RAM(1212)을 포함한다. 기본 입출력 시스템(BIOS)은 ROM, EPROM, EEPROM 등의 비휘발성 메모리(1210)에 저장되며, BIOS는 시동중과 같은 동안 컴퓨터(1202) 내의 구성요소들 간의 정보 전송을 돕는 기본 루틴을 포함한다. RAM(1212)은 또한 데이터를 캐싱하는 정적 RAM 등의 고속 RAM을 포함할 수도 있다.

컴퓨터(1202)는 내장형 하드 디스크 드라이브(HDD; 1214)(예를 들어, EIDE, SATA)를 포함하며, 내장형 하드 디스크 드라이브(1214)는 적절한 새시(chassis)(도시되지 않음)에서의 외부적 사용을 위해서도 구성될 수 있다. 또한, 컴퓨터(1202)는 자기 플로피 디스크 드라이브(FDD; 1216)(예를 들어, 분리형 디스켓(1218)으로부터 판독하거나 그것에 기록), 및 광학 디스크 드라이브(1220)(예를 들어, CD-ROM 디스크(1222)를 판독하고, DVD 등의 다른 고성능 광학 매체로부터 판독하고 그것에 기록)를 더 포함한다. 하드 디스크 드라이브(1214), 자기 디스크 드라이브(1216), 및 광학 디스크 드라이브(1220)는 또한 하드 디스크 드라이브 인터페이스(1224), 자기 디스크 드라이브 인터페이스(1226) 및 광학 드라이브 인터페이스(1228)에 의해 시스템 버스(1208)에 각각 연결될 수 있다. 외부 드라이브 구현을 위한 인터페이스(1224)는 USB(유니버셜 시리얼 버스) 및 IEEE 1394 인터페이스 기술 중 적어도 하나 또는 모두를 포함한다.

드라이브 및 그들의 연관된 컴퓨터 판독가능 매체는 데이터, 데이터 구조, 컴퓨터 실행가능 명령어 등의 비휘발성 저장을 제공한다. 컴퓨터(1202)에 대해서, 드라이브 및 매체는 적절한 디지털 포맷으로 임의의 데이터의 저장을 수용한다. 비록 상기 컴퓨터 판독가능 매체의 설명이 HDD, 분리형 자기 디스켓, 및 CD 또는 DVD 등의 분리형 광학 매체를 의미하지만, 본 기술에 숙련된 자는 집(zip) 드라이브, 자기 카세트, 플래시 메모리 카드, 카트리지 등의 컴퓨터에 의해 판독가능한 다른 유형의 매체들도 예시적인 동작 환경에서 사용될 수 있으며, 또한, 이러한 매체들 중 임의의 것은 본 발명의 방법을 수행하는 컴퓨터 실행가능 명령어를 포함할 수 있음을 인식할 것이다.

오퍼레이팅 시스템(1230), 하나 이상의 어플리케이션 프로그램(1232), 다른 프로그램 모듈(1234) 및 프로그램 데이터(1236)를 포함하는 다수의 프로그램 모듈은, RAM(1212) 및 드라이브에 저장될 수 있다. 오퍼레이팅 시스템, 어플리케이션, 모듈 및/또는 데이터의 전부 또는 일부는 RAM(1212)에 캐싱될 수도 있다.

본 발명이 다양한 통상적으로 사용가능한 오퍼레이팅 시스템 또는 오퍼레이팅 시스템들의 조합으로 구현될 수 있음이 인식된다.

사용자는 하나 이상의 유선/무선 입력 디바이스, 예를 들어, 키보드(1238) 및 마우스(1240) 등의 포인트 디바이스를 통해 컴퓨터(1202)로 명령어 및 정보를 입력할 수 있다. 다른 입력 디바이스(도시되지 않음)는 마이크로폰, IR 원격 제어, 조이스틱, 게임 패드, 스타일러스 펜, 터치 스크린 등을 포함할 수 있다. 이러한 및 다른 입력 디바이스는 주로 시스템 버스(1208)에 접속된 입력 디바이스 인터페이스(1242)를 통해 프로세싱 유닛(1204)에 접속되지만, 병렬 포트, IEEE 1394 시리얼 포트, 게임 포트, USB 포트, IR 인터페이스 등의 다른 인터페이스에 의해 접속될 수 있다.

모니터(1244) 또는 다른 유형의 디스플레이 디바이스는 또한 비디오 어댑터(1246) 등의 인터페이스를 통해 시스템 버스(1208)에 접속된다. 모니터(1244) 외에, 컴퓨터는 전형적으로 스피커, 프린터 등의 다른 주변 출력 디바이스(도시되지 않음)를 포함한다.

컴퓨터(1202)는 원격 컴퓨터(들)(1248) 등의 하나 이상의 원격 컴퓨터들로의 유선 및/또는 무선 통신을 통한 논리적 연결을 사용하는 네트워크 환경에서 동작할 수 있다. 원격 컴퓨터(들)(1248)는 워크스테이션, 서버 컴퓨터, 라우터, 개인용 컴퓨터, 휴대용 컴퓨터, 마이크로프로세서-기반 오락 가전제품, 피어 디바이스, 또는 다른 일반 네트워크 노드일 수 있고, 간략화를 위해, 단지 메모리 저장 디바이스(1250)만이 도시되어 있지만, 컴퓨터(1202)와 관련하여 설명된 많은 또는 모든 구성요소를 전형적으로 포함한다. 도시된 논리적 연결은 LAN(1252) 및/또는 보다 큰 네트워크, 예를 들어 WAN(1254)으로의 무선/유선 연결을 포함한다. 이러한 LAN 및 WAN 네트워크 환경은 사무실 및 회사에서 일반적이고, 인트라넷과 같은 기업내 컴퓨터 네트워크를 용이하게 하며, 이들 모두는 인터넷과 같은 글로벌 통신 네트워크에 접속될 수 있다.

LAN 네트워크 환경에서 사용될 때, 컴퓨터(1202)는 유선 및/또는 무선 통신 네트워크 인터페이스 또는 어댑터(1256)를 통해 로컬 네트워크(1252)에 연결된다. 어댑터(1256)는 무선 어댑터(1256)와 통신하기 위해 배치된 무선 액세스 포인트를 또한 포함할 수 있는 LAN(1252)으로의 무선 또는 유선 통신을 용이하게 할 수 있다. WAN 네트워크 환경에서 사용될 때, 컴퓨터(1202)는 모뎀(1258)을 포함할 수 있거나, LAN 상의 통신 서버와 연결되거나, 인터넷을 이용하는 것과 같이 WAN(1254)을 통한 통신을 구축하기 위한 다른 수단을 가질 수 있다. 외장형이나 내장형, 및 무선이나 유선 디바이스일 수 있는 모뎀(1258)은 직렬 포트 인터페이스(1242)를 통해 시스템 버스(1208)에 연결된다. 네트워크 환경에서, 컴퓨터(1202)에 관련하여 도시된 프로그램 모듈 또는 그 일부는 원격 메모리/저장 디바이스(1250)에 저장될 수 있다. 도시된 네트워크 연결은 예시적이며, 컴퓨터들 간의 통신 링크를 구축하는 다른 수단이 사용될 수 있음을 인식할 것이다.

컴퓨터(1202)는 무선 통신에 동작적으로 배치된 임의의 무선 디바이스 또는 엔티티, 예를 들어, 프린터, 스캐너, 데스크탑, 휴대용 컴퓨터, PDA(portable data assistant), 위성 접시, 무선 탐지가능 태그와 연관된 장치 또는 위치의 임의의 부분(예를 들어, 키오스크, 신문 가판대, 화장실) 및 전화와 통신하도록 동작된다. 이는 적어도 Wi-Fi 및 Bluetooth™ 무선 기술을 포함한다. 따라서, 통신은 종래 네트워크를 갖는 소정의 구조일 수 있고, 또는 단순히 적어도 2개의 디바이스 간의 임시의 통신일 수 있다.

Wi-Fi(Wireless Fidelity)는 집의 소파, 호텔방의 침대, 또는 직장의 회의실로부터 인터넷에 무선 접속하는 것을 허용한다. Wi-Fi는 컴퓨터 등의 디바이스가 기지국의 범위 내에 있는 곳 어디에서든지, 실내외에서 데이터를 송수신할 수 있도록 하는 셀 폰과 같은 무선 기술이다. Wi-Fi 네트워크는 IEEE 802.11(a, b, g 등)라 불리는 무선 기술을 사용하여, 안전하고 신뢰할 수 있으며 신속한 무선 연결을 제공한다. Wi-Fi 네트워크는 컴퓨터들끼리, 컴퓨터를 인터넷에, 및 컴퓨터를 무선 네트워크(IEEE 802.3 또는 이더넷)에 연결하는 데 사용될 수 있다. Wi-Fi 네트워크는, 11Mbps(802.11b) 또는 54Mbps(802.11a) 데이터 레이트를 갖거나, 양 대역을 모두 포함하는 프로덕트(이중 대역)를 갖는 미허가 2.4 및 5GHz 무선 대역에서 동작하므로, 네트워크는 많은 사무실에서 사용되는 기본 10BaseT 무선 이더넷 네트워크와 유사한 실재 성능을 제공할 수 있다.

이제, 도 13을 참조하면, 본 발명에 따르는 예시적 컴퓨팅 환경(1300)의 개요도가 도시된다. 시스템(1300)은 하나 이상의 클라이언트(들)(1302)를 포함한다. 클라이언트(들)(1302)는 하드웨어 및/또는 소프트웨어(예를 들어, 스레드, 프로세스, 컴퓨팅 디바이스)일 수 있다. 클라이언트(들)(1302)는 예를 들어, 본 발명을 사용하여 쿠키(들) 및/또는 연관된 문맥상 정보를 하우징(house)할 수 있다. 시스템(1300)은 또한 하나 이상의 서버(들)(1304)를 포함할 수 있다. 서버(들)(1304)는 또한 하드웨어 및/또는 소프트웨어일 수 있다(예를 들어, 스레드, 프로세스, 컴퓨팅 디바이스). 서버(1304)는 본 발명을 사용하는 변환을 수행하기 위해 스레드를 하우징할 수 있다. 클라이언트(1302)와 서버(1304) 사이의 하나의 가능한 통신은 2개 이상의 컴퓨터 프로세스 간에 전송되도록 적응된 데이터 패킷의 형태일 수 있다. 데이터 패킷은 예를 들어, 쿠키 및/또는 연관된 문맥상 정보를 포함할 수 있다. 시스템(1300)은 클라이언트(들)(1302)와 서버(들)(1304) 사이의 통신을 용이하게 하는 데 사용될 수 있는 통신 프레임워크(1306)(예를 들어, 인터넷 등의 글로벌 통신 네트워크)를 포함한다.

통신은 무선(광섬유 포함) 및/또는 무선 기술을 통해 용이하게 될 수 있다. 클라이언트(들)(1302)는 클라이언트(들)(1302)에 로컬인 정보(예를 들어, 쿠키(들) 및/또는 연관된 문맥상 정보)를 저장하는 데 사용될 수 있는 하나 이상의 클라이언트 데이터 저장장치(들)(1308)에 동작적으로 연결된다. 유사하게, 서버(들)(1304)는 서버(1304)에 로컬인 정보를 저장하는 데 사용될 수 있는 하나 이상의 서버 데이터 저장장치(들)(1310)에 동작적으로 연결된다.

상술된 것은 본 발명의 예를 포함한다. 물론 본 발명을 설명하기 위해 컴포넌트 또는 방법론의 모든 고려가능한 조합을 설명하는 것은 불가능하지만, 본 분야에 숙련된 자들은 본 발명의 다른 많은 조합 및 변경들이 가능함을 인식할 것이다. 따라서, 본 발명은 첨부된 청구항의 취지 및 영역 내에 속하는 모든 이러한 대체, 수정 및 변형을 포함하도록 의도된다. 또한, 상세한 설명 또는 청구항에서 사용되는 용어 "포함한다"는 포괄적인 의미를 갖는다.

발명의 효과

본 발명은 비통합 어플리케이션의 접속을 용이하게 하는 아키텍처를 제공한다.

(57) 청구의 범위

청구항 1.

비통합 어플리케이션들(non-integrated applications)의 인터페이스를 용이하게 하는 시스템으로서,

상기 어플리케이션들 중 제1 어플리케이션의 아티펙트(artifact)들을 노출시키는 아티펙트 제공자; 및

상기 어플리케이션들 중 제2 어플리케이션의 참조를 노출시키는 아티펙트 소비자 -상기 참조는 상기 아티펙트 제공자에 의해 홀딩된 상기 아티펙트들 중 적어도 하나로의 링크임-

를 포함하는 시스템.

청구항 2.

제1항에 있어서,

상기 링크는 URI(uniform resource identifier)인 시스템.

청구항 3.

제1항에 있어서, 상기 아티펙트 제공자 및 상기 아티펙트 소비자는 각각의 어플리케이션에 인터페이스하는 API(application program interface)인 시스템.

청구항 4.

제1항에 있어서, 상기 참조와 대응 아티펙트를 링크시키는 링크 컴포넌트(linking component)를 더 포함하는 시스템.

청구항 5.

제4항에 있어서, 상기 링크 컴포넌트는 아티펙트를 포인트하는 상기 아티펙트 소비자에 의해 홀딩된 아티펙트 식별자인 시스템.

청구항 6.

제4항에 있어서, 상기 링크 컴포넌트는 바이너리이고, 참조 아티펙트(referring artifact) 및 피참조 아티펙트(referred artifact)와 관련된 시스템.

청구항 7.

제1항에 있어서, 상기 제공자 및 상기 소비자 중 적어도 하나는 툴 또는 서비스인 시스템.

청구항 8.

제1항에 있어서, 상기 아티펙트 제공자는 자신이 제공하는 각각의 아티펙트에 대한 아티펙트 유형을 등록하고, 각각의 아티펙트가 호스트할 수 있는 대응 링크 유형을 등록하는 시스템.

청구항 9.

제1항에 있어서, 상기 아티펙트 및 아티펙트 링크 둘다를 저장하고 노출시키는 것을 용이하게 하기 위해 툴에 인터페이스하는 일반 아티펙트 제공자(GAP)를 더 포함하는 시스템.

청구항 10.

제9항에 있어서, 상기 GAP와 비통합 어플리케이션 간의 인터페이스를 제공하는 GAP 어댑터를 더 포함하는 시스템.

청구항 11.

제1항에 있어서, 상기 아티펙트 및 관련된 아티펙트 링크를 저장하는 캐시를 더 포함하는 시스템.

청구항 12.

제1항에 있어서, 아티펙트간 참조(inter-artifact reference)를 표시하는 것을 용이하게 하는 사용자 인터페이스를 더 포함하는 시스템.

청구항 13.

제1항의 시스템을 실행하기 위한 컴퓨터 실행가능 명령어들이 저장된 컴퓨터 판독가능 매체.

청구항 14.

제1항의 시스템을 사용하는 컴퓨터.

청구항 15.

제1항의 시스템을 사용하는 서버.

청구항 16.

비통합 어플리케이션들의 인터페이스를 용이하게 하는 시스템으로서,

상기 어플리케이션들 중 제1 어플리케이션의 아티펙트를 노출시키는 툴 또는 서비스인 아티펙트 제공자;

상기 어플리케이션들 중 제2 어플리케이션의 참조를 노출시키는 툴 또는 서비스인 아티펙트 소비자; 및

상기 아티펙트를 포인트하는 상기 아티펙트 소비자에 의해 홀딩된 링크

를 포함하는 시스템.

청구항 17.

제16항에 있어서, 상기 링크는 불변적이고 고유하게 구성된 키인 아티펙트 식별자인 시스템.

청구항 18.

제16항에 있어서, 캐시 컨텐츠를 갱신 및 소거하여 캐시를 관리하는 링크 관리자를 더 포함하는 시스템.

청구항 19.

제16항에 있어서, 상기 아티펙트 제공자 및 상기 아티펙트 소비자는 느슨하게 결합된(loosely coupled) 것과 밀접하게 결합된(tightly coupled) 것 중 적어도 하나인 시스템.

청구항 20.

제16항에 있어서, 상기 아티펙트 소비자, 상기 아티펙트 제공자 및 상기 비통합 어플리케이션 중 적어도 하나에 관련된 매개변수에 기초하여 추론하는 식별자를 더 포함하는 시스템.

청구항 21.

제16항에 있어서, 상기 아티펙트 제공자는 느슨하게 결합된 서버 기반 상호작용, 느슨하게 결합된 클라이언트, 캐싱, 및 호출자와의 계약에 의한 아티펙트-특정 함수를 지원하는 밀접하게 결합된 상호작용 중 적어도 하나에 대한 URI를 생성하고 나타내는 시스템.

청구항 22.

제16항에 있어서, 상기 아티펙트 소비자는 상기 링크를 홀딩하고, 상기 링크는 상기 피참조 아티펙트로의 URI인 시스템.

청구항 23.

비통합 어플리케이션들 간의 인터페이스를 용이하게 하는 방법을 수행하기 위한 컴퓨터 실행가능 명령어를 갖는 컴퓨터 판독가능 매체로서,

상기 방법은,

제1 비통합 어플리케이션과 통신하는 아티펙트 제공자를 제공하는 단계;

상기 아티펙트 제공자를 사용하여 상기 제1 어플리케이션의 아티펙트를 노출시키는 단계;

제2 비통합 어플리케이션과 통신하는 아티펙트 소비자를 제공하는 단계;

상기 아티펙트 소비자를 사용하여 상기 제2 어플리케이션의 참조를 노출시키는 단계; 및

아티펙트 식별자를 사용하여 상기 아티펙트에 상기 참조를 링크하는 단계

를 포함하는 컴퓨터 판독가능 매체.

청구항 24.

제23항에 있어서,

상기 방법은,

상기 아티펙트에 대한 아티펙트 유형을 등록하는 단계; 및

상기 아티펙트가 호스트하는 링크 유형을 등록하는 단계

를 포함하는 컴퓨터 판독가능 매체.

청구항 25.

제23항에 있어서,

상기 방법은,

사용자에게 상기 아티펙트의 종속성 정보를 표시하는 단계를 더 포함하고,

상기 정보는 링크 유형, 아티펙트 유형, 아티펙트 네임, 및 수정 날짜 중 적어도 하나를 포함하는 컴퓨터 판독가능 매체.

청구항 26.

제23항에 있어서, 상기 아티펙트 소비자와 상기 아티펙트 제공자 중 적어도 하나는 웹 서비스인 컴퓨터 판독가능 매체.

청구항 27.

제23항에 있어서, 비통합 어플리케이션에 저장된 정보를 나타내는 아티펙트 프록시를 생성하는 단계를 더 포함하는 컴퓨터 판독가능 매체.

청구항 28.

제23항에 있어서, 상기 아티펙트는 소스 파일, 결함, 요구조건(requirement), 테스트 결과, 및 빌드(build) 중 적어도 하나를 대표하는 컴퓨터 판독가능 매체.

청구항 29.

제23항에 있어서, 상기 링크하는 단계는 참조 URI, 피참조 URI 및 링크 유형을 포함하는 링크를 포함하는 컴퓨터 판독가능 매체.

청구항 30.

제23항에 있어서, 상기 방법은, 어떤 참조 아티펙트가 특정 피참조 아티펙트로의 링크를 홀딩하는지를 알아내는 단계를 더 포함하는 컴퓨터 판독가능 매체.

청구항 31.

제23항에 있어서, 상기 방법은, 상기 아티펙트가 생성, 삭제 및 변경된 것 중에 적어도 하나일 때, 이벤트를 발생시키는 단계를 더 포함하는 컴퓨터 판독가능 매체.

청구항 32.

제23항에 있어서, 상기 방법은, 상기 아티펙트 제공자에 의해 상기 아티펙트에 대한 외부 어드레스가능성을 제공하는 단계를 더 포함하는 컴퓨터 판독가능 매체.

청구항 33.

제23항에 있어서, 상기 방법은, 상기 아티펙트 제공자 및 상기 아티펙트 소비자 둘다 일반 API를 제공하는 단계를 더 포함하는 컴퓨터 판독가능 매체.

청구항 34.

비통합 어플리케이션들의 인터페이스를 용이하게 하는 시스템으로서,

제1 어플리케이션의 아티펙트를 노출시키는 수단;

제2 어플리케이션의 참조를 노출시키는 수단;

아티펙트 식별자를 이용하여, 상기 참조를 상기 아티펙트에 링크하는 수단; 및

캐싱 수단을 이용하여, 상기 아티펙트 식별자 및 상기 아티펙트를 캐싱하는 수단

을 포함하는 시스템.

청구항 35.

제34항에 있어서, 상기 캐싱 수단은 상기 아티펙트의 소스 및 상기 아티펙트 식별자의 소스에 동기화하는 수단을 더 포함하는 시스템.

청구항 36.

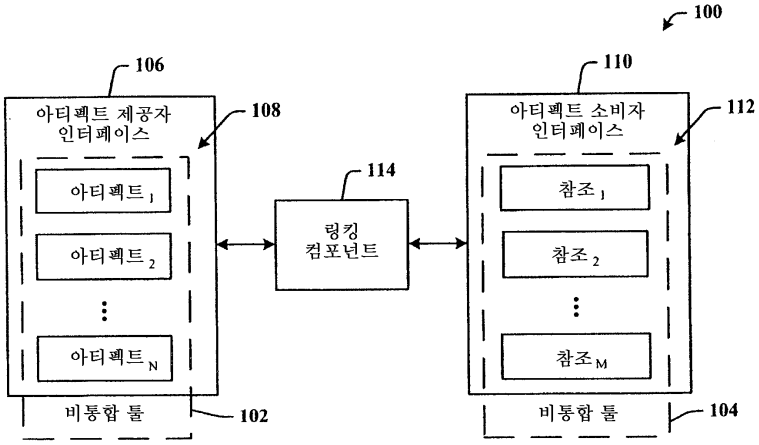
제34항에 있어서, 상기 아티펙트 및 아티펙트 식별자를 XML로 정의하는 수단을 더 포함하는 시스템.

청구항 37.

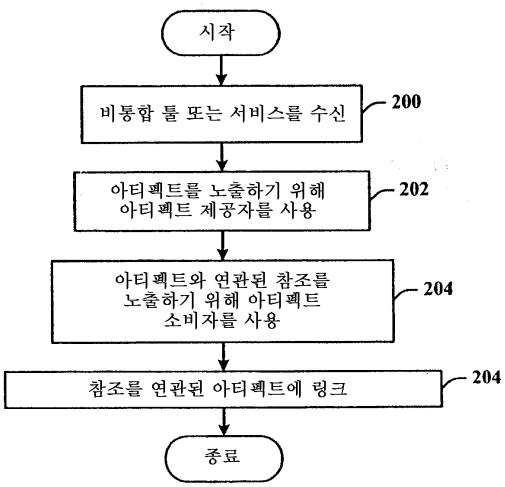
제34항에 있어서, 쿼리를 필터링하는 수단을 더 포함하는 시스템.

도면

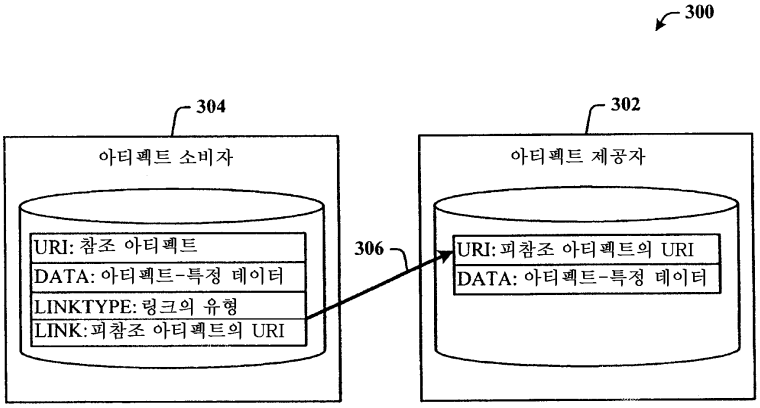
도면1



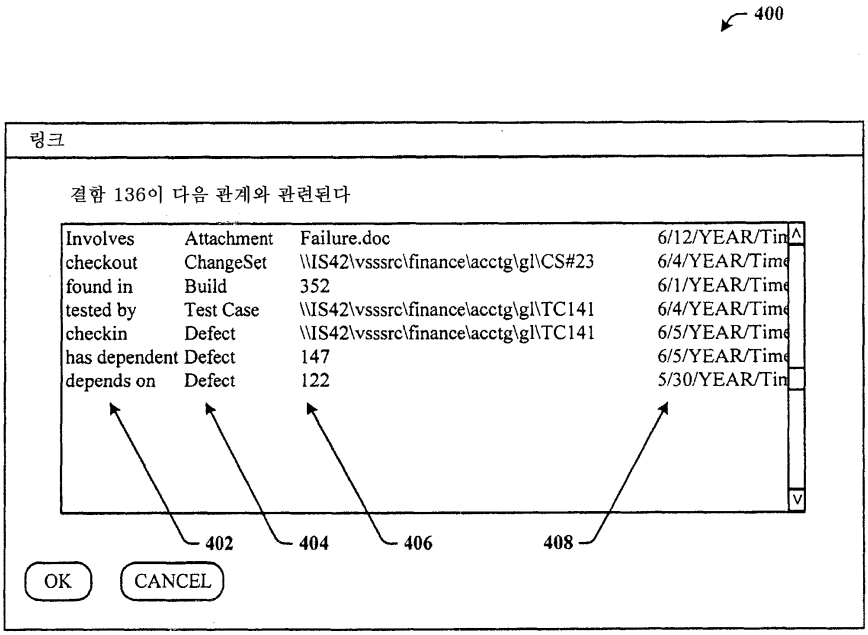
도면2



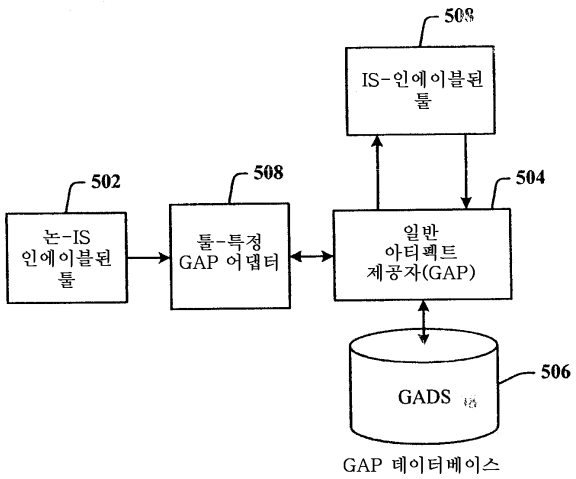
도면3


300
304
아티펙트 소비자
URI: 참조 아티펙트
DATA: 아티펙트-특정 데이터
LINKTYPE: 링크의 유형
LINK: 피참조 아티펙트의 URI
306
302
아티펙트 제공자
URI: 피참조 아티펙트의 URI
DATA: 아티펙트-특정 데이터

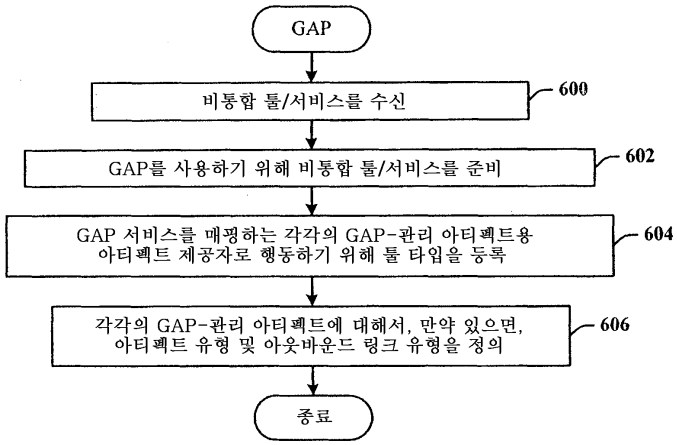
도면4


400
링크
결함 136이 다음 관계와 관련된다
Involves Attachment Failure.doc 6/12/YEAR/Tin
checkout ChangeSet \\IS42\vsssrc\finance\acctg\gl\CS#23 6/4/YEAR/Time
found in Build 352 6/1/YEAR/Time
tested by Test Case \\IS42\vsssrc\finance\acctg\gl\TC141 6/4/YEAR/Time
checkin Defect \\IS42\vsssrc\finance\acctg\gl\TC141 6/5/YEAR/Time
has dependent Defect 147 6/5/YEAR/Time
depends on Defect 122 5/30/YEAR/Tin
402
404
406
408
OK
CANCEL

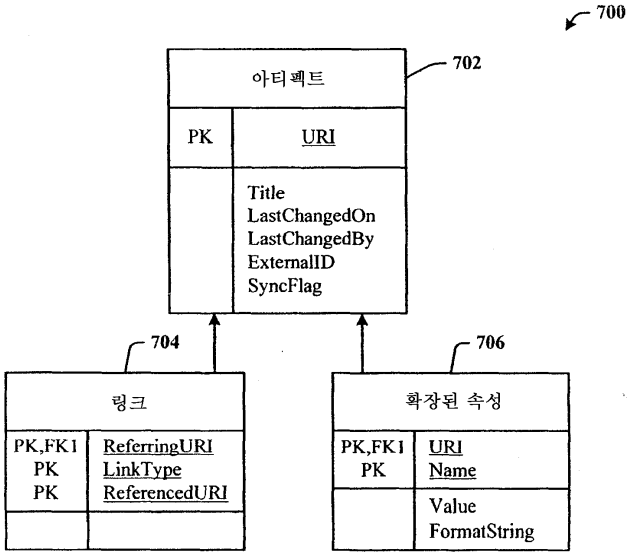
도면5


508
IS-인에이블된 툴
502
논-IS 인에이블된 툴
508
툴-특정 GAP 어댑터
504
일반 아티펙트 제공자(GAP)
506
GADS
GAP 데이터베이스

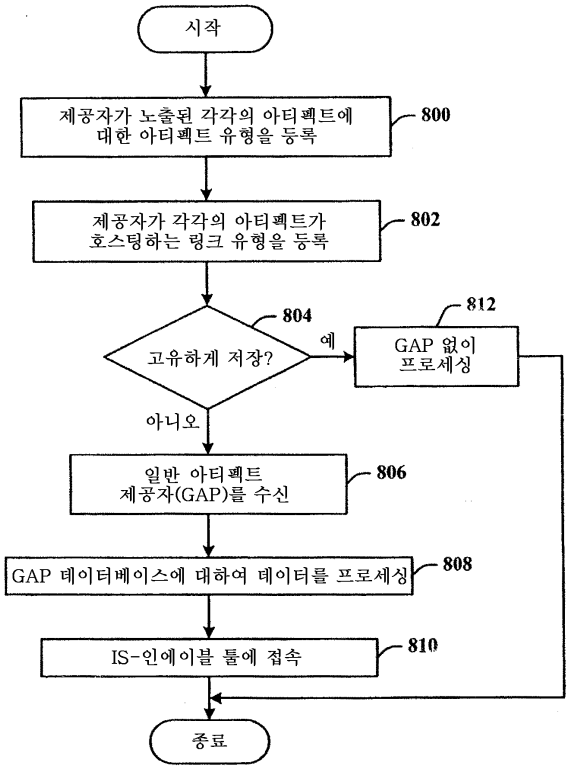
도면6


GAP
비통합 툴/서비스를 수신
600
GAP를 사용하기 위해 비통합 툴/서비스를 준비
602
GAP 서비스를 매핑하는 각각의 GAP-관리 아티펙트용 아티펙트 제공자로 행동하기 위해 툴 타입을 등록
604
각각의 GAP-관리 아티펙트에 대해서, 만약 있으면, 아티펙트 유형 및 아웃바운드 링크 유형을 정의
606
종료

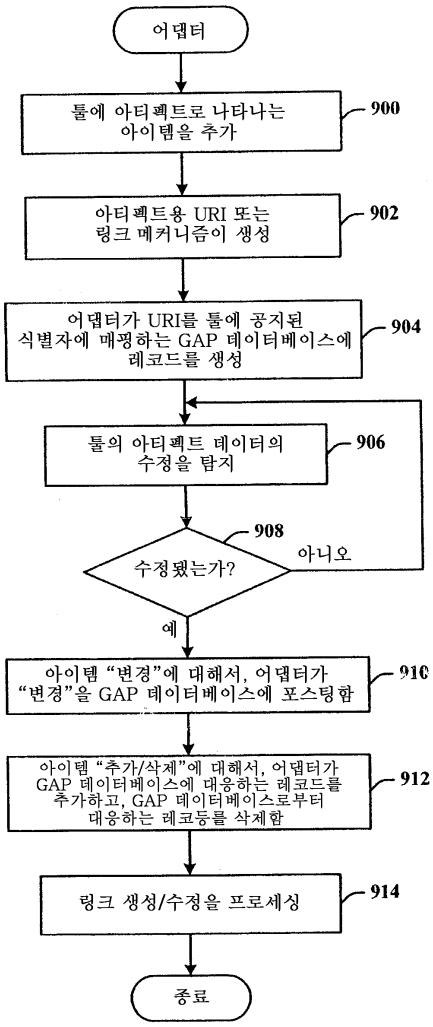
도면7


700
아티펙트
702
PK
URI
Title
LastChangedOn
LastChangedBy
ExternalID
SyncFlag
704
링크
PK,FK1
PK
PK
ReferringURI
LinkType
ReferencedURI
706
확장된 속성
PK,FK1
PK
URI
Name
Value
FormatString

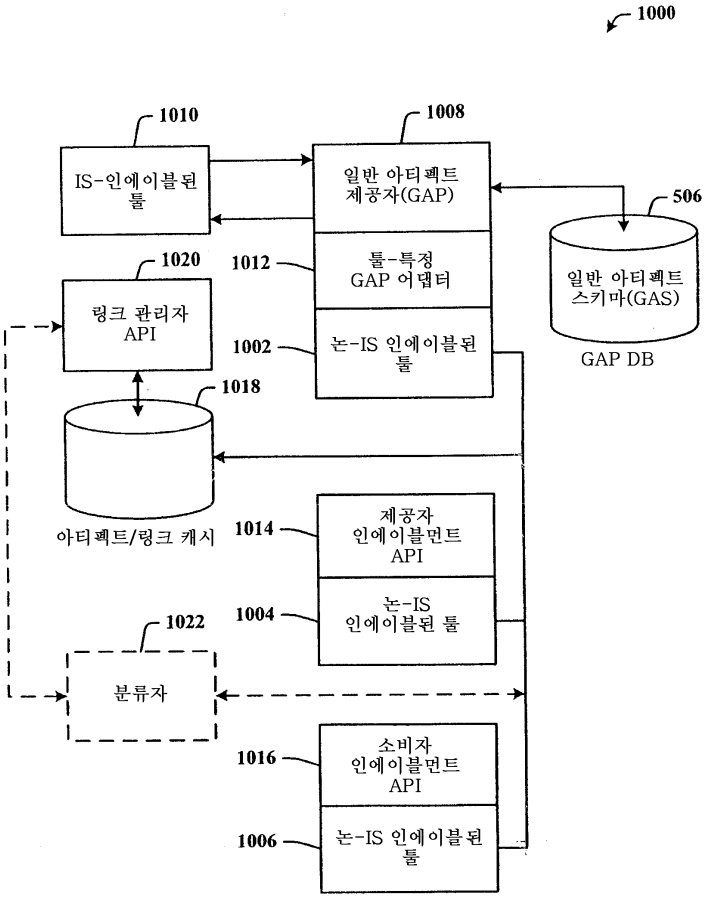
도면8


시작
제공자가 노출된 각각의 아티펙트에 대한 아티펙트 유형을 등록
800
제공자가 각각의 아티펙트가 호스팅하는 링크 유형을 등록
802
804
고유하게 저장?
예
812
GAP 없이 프로세싱
아니오
일반 아티펙트 제공자(GAP)를 수신
806
GAP 데이터베이스에 대하여 데이터를 프로세싱
808
IS-인에이블 툴에 접속
810
종료

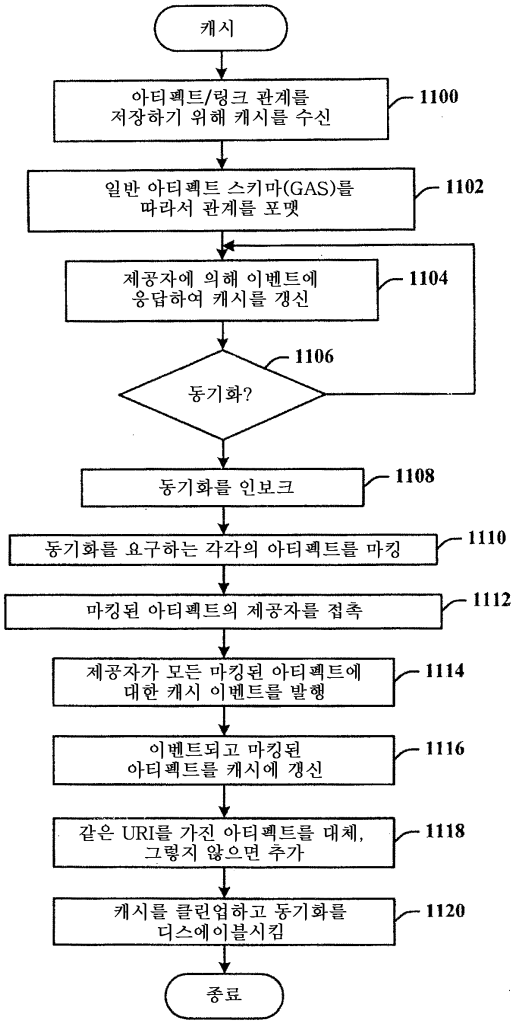
도면9


어댑터
툴에 아티펙트로 나타나는 아이템을 추가
900
아티펙트용 URI 또는 링크 메커니즘이 생성
902
어댑터가 URI를 툴에 공지된 식별자에 매핑하는 GAP 데이터베이스에 레코드를 생성
904
툴의 아티펙트 데이터의 수정을 탐지
906
908
수정됐는가?
아니오
예
아이템 "변경"에 대해서, 어댑터가 "변경"을 GAP 데이터베이스에 포스팅함
910
아이템 "추가/삭제"에 대해서, 어댑터가 GAP 데이터베이스에 대응하는 레코드를 추가하고, GAP 데이터베이스로부터 대응하는 레코등를 삭제함
912
링크 생성/수정을 프로세싱
914
종료

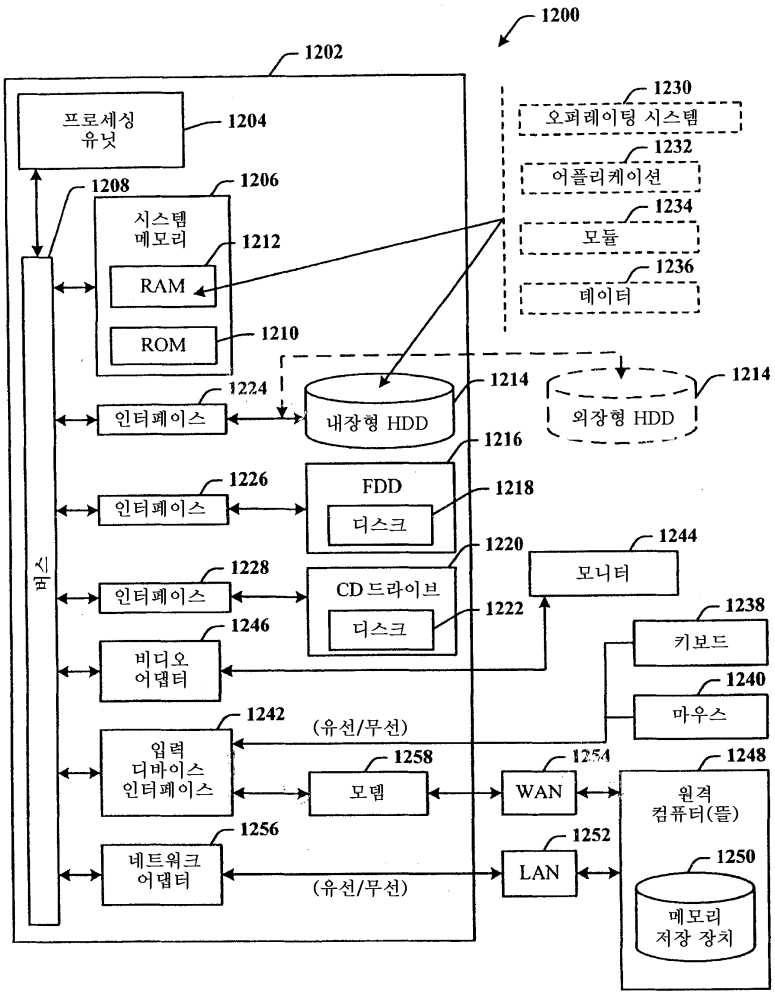
도면10


1000
1010
IS-인에이블된 툴
1008
일반 아티펙트 제공자(GAP)
506
일반 아티펙트 스키마(GAS)
GAP DB
1012
툴-특정 GAP 어댑터
1002
논-IS 인에이블된 툴
1020
링크 관리자 API
1018
아티펙트/링크 캐시
1014
제공자 인에이블먼트 API
1004
논-IS 인에이블된 툴
1022
분류자
1016
소비자 인에이블먼트 API
1006
논-IS 인에이블된 툴

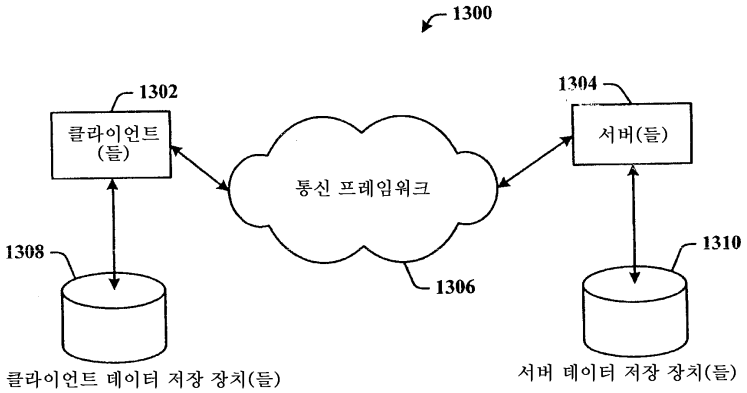
도면11


캐시
아티펙트/링크 관계를 저장하기 위해 캐시를 수신
1100
일반 아티펙트 스키마(GAS)를 따라서 관계를 포맷
1102
제공자에 의해 이벤트에 응답하여 캐시를 갱신
1104
1106
동기화?
동기화를 인보크
1108
동기화를 요구하는 각각의 아티펙트를 마킹
1110
마킹된 아티펙트의 제공자를 접촉
1112
제공자가 모든 마킹된 아티펙트에 대한 캐시 이벤트를 발행
1114
이벤트되고 마킹된 아티펙트를 캐시에 갱신
1116
같은 URI를 가진 아티펙트를 대체, 그렇지 않으면 추가
1118
캐시를 클린업하고 동기화를 디스에이블시킴
1120
종료

도면12


1200
1202
프로세싱 유닛
1204
1206
시스템 메모리
1208
1212
RAM
ROM
1210
1230
오퍼레이팅 시스템
1232
어플리케이션
1234
모듈
1236
데이터
1214
내장형 HDD
외장형 HDD
1224
인터페이스
1216
FDD
디스크
1218
1226
인터페이스
1220
CD 드라이브
디스크
1222
1228
인터페이스
1244
모니터
1238
키보드
1240
마우스
1246
비디오 어댑터
1242
입력 디바이스 인터페이스
(유선/무선)
1258
모뎀
1254
WAN
1248
원격 컴퓨터(들)
1256
네트워크 어댑터
(유선/무선)
1252
LAN
1250
메모리 저장 장치
버스

도면13


1300
1302
클라이언트(들)
통신 프레임워크
1304
서버(들)
1308
1306
1310
클라이언트 데이터 저장 장치(들)
서버 데이터 저장 장치(들)