서비스 지향 구조 및 컴포넌트 기본 개발을 사용하여 웹 서비스 어플리케이션 빌드

Rational Software 백서

Alan Brown,
Simon Johnston,
Kevin Kelly

Rational
the software development company

목차

소개

엔터프라이즈 규모의 소프트웨어 시스템을 빌드하는 것은 복잡한 작업입니다. 수 십 년 간의 기술 발전에도 불구하고 오늘날 정보 시스템의 요구사항은 업무 중심의 소프트웨어 솔루션을 설계, 구축 및 지속적으로 발전시켜나갈 수 있는 기업의 능력을 넘어서고 있습니다. 특히 시작 단계서부터 설계되는 새로운 시스템은 거의 없습니다. 그보다는 기존 데이터 저장소를 조작하는 새 비즈니스 논리에 대해 설명, 인터넷 브라우저 또는 휴대용 장치와 같은 새 채널을 통해 기존 데이터 및 트랜잭션 표시, 중복된 비즈니스 활동을 지원하는 이전에 연결 해제된 시스템 통합 등과 같은 작업으로 기존 솔루션의 수명을 연장시키는 것이 일반적인 소프트웨어 아키텍트의 타스크입니다.

소프트웨어 개발자를 지원하기 위해 Microsoft 및 IBM 과 같은 벤더에서 상업 소프트웨어 인프라스트럭처 제품을 제공하고 있습니다. 이러한 제품은 자사의 .NET 및 WebSphere 제품군에서 각각 선호하는 소프트웨어 개발 방법의 중심 역할을 하고 있습니다. 두 가지 접근법 모두 배포된 서비스의 시스템 어셈블리에 중점을 두고 있습니다. 그러나 서비스에서 엔터프라이즈 규모 솔루션을 빌드하는 데 새로운 사항이 있습니까? 컴포넌트 기반 시스템의 내용을 서비스 기반 구조(SOA) 구축에 적용할 수 있는 방법은 무엇입니까? 이 새로운 세대의 소프트웨어 인프라스트럭처 제품에 대한 전개용으로 고품질의 시스템을 빌드하기 위한 최상의 접근법은 무엇입니까? 이러한 중요한 질문이 바로 이 백서의 주제입니다.

최근 들어 소프트웨어 엔지니어링 커뮤니티에서 가장 많은 주목을 받고 있는 주제는 독립적이고 재활용 가능한 기능 콜렉션에서 대형 소프트웨어 시스템을 어셈블할 수 있는 개념을 지원하는 설계 접근법, 프로세스 및 관련 툴입니다. 일부 기능은 이미 사용가능하여 내부 또는 제 3 자가 인수하여 구현되고 있는 반면, 나머지 기능은 작성되어야 합니다. 이 경우, 전체 시스템은 이러한 모든 요소를 결합하여 하나의 단일체로 인식하여 설계해야 합니다. 현재 이는 *컴포넌트 기본 개발(CBD)*로 구체화되고 있으며, 이 개념은 Microsoft .NET 플랫폼과 IBM

WebSphere 및 Sun 의 iPlanet 과 같은 제품이 지원하는 J2EE(Java 2 Enterprise Edition) 표준과 같은 기술 접근법으로 구현되고 있습니다.

또 한 가지 고려해야 할 사항은 운영 체제가 일반적으로 많은 시스템에 배포되어 성능, 사용 가능성 및 확장성을 향상시킨다는 점입니다. 엔터프라이즈 솔루션은 하드웨어 콜렉션에서 실행되는 성능을 조정해야 합니다. 이러한 시스템을 구축하는 방법 중 하나는 상호작용 서비스의 콜렉션으로 구성하는 것입니다. 각 서비스는 잘 정의된 기능 콜렉션에 대한 액세스를 제공합니다. 시스템은 전체적으로 이러한 서비스 간의 상호작용 세트로서 설계 및 구현됩니다. 기능을 서비스로 표현하는 것이 유연성을 확보하는 데 있어 중요한 열쇠입니다. 이를 통해 특정 기능이 물리적 위치에 관계 없이 자연스럽게 다른 서비스를 이용할 수 있습니다(해당 기능은 서비스로 구현됨). 시스템은 새로운 서비스를 추가함으로써 진화됩니다. 결과 *서비스 지향 구조(SOA)는* 구성되는 시스템 서비스를 정의하고 특정 작동을 구현하기 위해 서비스에서 발생하는 상호작용에 대해 설명하며 서비스를 하나 이상의 특정 기술 구현으로 맵핑합니다.

서비스에는 비즈니스 기능이 포함되지만 서비스 상호작용 및 의사소통을 용이하게 하려면 특정 양식의 상호 서비스 인프라스트럭처가 필요합니다. 서비스는 단일 시스템에서 구현되거나 로컬 영역 네트워크의 컴퓨터 세트에 분배되거나 여러 회사 네트워크에 보다 광범위하게 분배될 수도 있으므로 이러한 인프라스트럭처의 다른 양식도 가능합니다. 특히 흥미로운 것은 서비스가 인터넷을 의사소통 메커니즘으로 사용하는 경우입니다. 결과 *웹 서비스는* 보다 일반적인 서비스의 특성을 공유하지만 서비스 간의 상호작용을 위한 충성도가 낮은 공용 비보안 메커니즘을 사용한 결과로 특별히 고려해야 할 사항이 있습니다.

지금까지 소프트웨어 산업에서는 주로 웹 서비스 및 해당 상호작용의 구현을 위한 기본 기술에 주력해왔습니다. 그러나 엔터프라이즈 규모 솔루션으로의 어셈블이 용이하도록 웹 서비스를 설계하는 가장 적합한 방법에 대한 관심이 높아지고 있습니다. 반대로 웹 서비스로 구성되는 엔터프라이즈 규모 소프트웨어 솔루션을 구축하기 위한 올바른 방법 및 툴에 대한 관심은 줄어들고 있습니다. 복잡한 구조의 설계와 같이, 설계 기술, 구조 패턴 및 스타일에 대한 충분한 이해를 바탕으로 한 빠른 구조 의사결정을 내려야 우수한 솔루션을 개발할 수 있습니다. 이러한 패턴은 확장성, 신뢰성 및 보안과 같은 일반적인 서비스 사안을 해결합니다.

이 백서에서는 엔터프라이즈 규모 소프트웨어 솔루션에 필요한 서비스 지향 구조 및 서비스에 대해 잘 이해할 수 있는 컨텍스트를 제공합니다. 특히 소프트웨어 컴포넌트의 보다 구체적인 개념과의 관계에서 서비스를 설명하며 현재 컴포넌트 기본 개발 방법과 서비스 지향 구조 구현을 위한 시범 및 테스트 기초와의 관계에 대해 설명합니다. 서비스 및 컴포넌트 설계 모두에 있어 인터페이스 기반 설계가 강조되며 이 두 가지로 표현되는 인터페이스에 특정 제한조건 및 서로를 구분하는 기준이 있는지에 대해 설명합니다. UML(Unified Modeling Language)[1]는 논리 및 구현 설계와 컴포넌트 및 서비스 설계 모두에 대한 특정 패턴에 대해 설명하는 툴로서 사용됩니다.

마지막으로 이 백서는 일반적인 서비스 구현과 관련된 사안을 확인하기 위한 수단으로서의 웹 서비스에 중점을 둡니다. 특히 이 백서에서는 서비스 지향 구조에 대한 일반 지침을 웹 서비스에 대한 특정 설계 패턴으로 해석하는 방법에 대해 설명합니다. 이 백서에서 설명하는 모든 패턴은 Rational XDE 의 고유한 패턴 및 코드 템플리트 성능을 사용하여 구현되었으며 Rational Developer Network[8]를 통해 출력되었습니다.

서비스 지향 구조의 개념

서비스 지향 구조(SOA)란 무엇입니까? 서비스 지향 구조란 발견 가능한 출력 인터페이스를 통해 일반 사용자 어플리케이션 또는 기타 서비스에 서비스를 제공할 수 있도록 소프트웨어 시스템을

설계하는 방법입니다. 대부분의 경우, 서비스는 분리된 비즈니스 기능을 표시할 수 있는 보다 나은 방법을 제공하므로 비즈니스 프로세스를 지원하는 어플리케이션을 개발하는 것이 좋습니다.

서비스 지향 구조는 새로운 개념이 아니며 새로운 웹 서비스 기술로 인해 이 시점에서 중요한 것입니다. 예를 들어, 다음은 2000 년 출간된 책에서 서비스 지향 구조의 가치에 대해 설명하는 부분을 인용한 것입니다.

> *서비스 지향 솔루션 어플리케이션을 잠재적인 사용자에게 잘 정의된 인터페이스를 제공하는 독립적인 상호작용 서비스 세트로서 개발해야 합니다. 또한 어플리케이션 개발자가 서비스 콜렉션을 검색하고 원하는 서비스를 선택하여 어셈블함으로써 원하는 기능을 작성할 수 있는 지원 기술이 필요합니다. [2]*

이 문서의 목적에 따라 다음과 같은 서비스 정의를 고려합니다.

> *서비스는 일반적으로 단일 인스턴스로서 존재하며, 느슨하게 커플링(일반적으로 비동기)된 메시지 기반 의사소통 모델을 통해 어플리케이션 및 기타 서비스와 상호작용하는 간략하게 세분화된 발견 가능한 소프트웨어 엔티티로서 구현됩니다.*

일반적으로 서비스라는 용어는 컴포넌트 기본 개발을 설명하는 데 사용되는 용어와 동일하지만 아래 그림 1 과 같이 웹 서비스 내 요소를 정의하기 위해서는 특정 용어를 사용합니다.

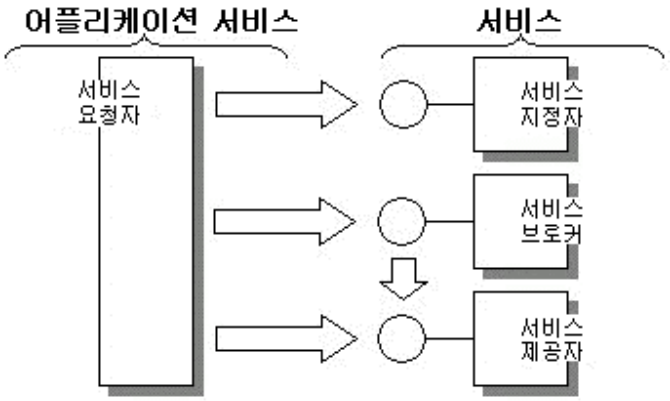

그림 1 – 서비스 용어

- **서비스** — 로컬 엔티티. 하나 이상의 출력 엔터페이스로 정의되는 계약
- **서비스 제공자** — 서비스 스펙을 구현하는 소프트웨어 엔티티
- **서비스 요청자** — 서비스 제공자를 호출하는 소프트웨어 엔티티 일반적으로 “클라이언트”라고 합니다. 그러나 서비스 요청자는 일반 사용자 어플리케이션 또는 다른 서비스가 될 수 있습니다.
- **서비스 지정자** — 레지스트리 기능을 수행하며 서비스 제공자 인터페이스 및 서비스 위치 검색을 허용하는 특정 유형의 서비스 제공자
- **서비스 브로커** — 하나 이상의 추가 서비스 제공자에게 서비스 요청을 전달할 수 있는 특정 유형의 서비스 제공자

이러한 서비스 설명 및 사용 컨텍스트에는 일련의 제한조건이 있습니다. 또한 서비스를 효율적으로 사용하기 위해서는 몇 가지 높은 수준의 베스트 프랙티스가 필요합니다. 서비스를 효과적으로 사용하기 위한 몇 가지 주요 특성은 다음과 같습니다.

- **간략한 세분화** — 서비스에 대한 조작은 일반적으로 보다 많은 기능을 포함하도록 구현되며 컴포넌트 인터페이스 설계와 비교하여 보다 많은 데이터 세트에서 작동됩니다.
- **인터페이스 기반 설계** — 서비스는 개별적으로 정의된 인터페이스를 구현합니다. 이러한 방법의 장점은 여러 서비스가 공통 인터페이스를 구현할 수 있으며 하나의 서비스가 여러 인터페이스를 구현할 수 있다는 것입니다.
- **발견 가능성** — 서비스는 고유한 ID 뿐 아니라 인터페이스 ID 및 서비스 유형에 의해 설계 시간과 런타임에서 모두 표시되어야 합니다.
- **단일 인스턴스** — 필요에 따라 컴포넌트를 인스턴스화하는 컴포넌트 기본 개발과 달리 각 서비스는 여러 클라이언트가 의사소통하며 항상 실행되는 단일 인스턴스입니다.
- **느슨한 커플링** — 서비스는 XML 문서 교환과 같이 종속성이 적고 분리된 표준 메시지 기반 메소드를 사용하여 기타 서비스 및 클라이언트에 연결됩니다.
- **비동기** — 일반적으로 서비스는 비동기 메시지 전달 방법을 사용하지만 필수는 아닙니다. 실제로는 많은 서비스가 때때로 동기 메시지 전달을 사용합니다.

이러한 기준 중 인터페이스 기반 설계 및 발견성과 같은 일부 기준은 컴포넌트 기본 개발에도 사용됩니다. 그러나 이러한 속성을 모두 함께 사용한다는 것이 J2EE 또는 .NET 와 같이 컴포넌트 구조를 사용하여 개발된 어플리케이션과 서비스 기반 어플리케이션을 차별화하는 요인입니다.

인터페이스 기반 설계

컴포넌트 및 서비스 개발 모두 인터페이스 설계는 소프트웨어 엔티티가 해당 정의의 중요 파트를 구현 및 표현함으로써 수행됩니다. 따라서 인터페이스뱒라는 표현과 개념이 컴포넌트 기반 및 서비스 지향 시스템 설계에 있어 가장 중요한 요인입니다. 다음은 몇 가지 중요한 인터페이스 관련 정의입니다.

- **인터페이스** — 논리적으로는 그룹화되지만 구현되지 않는 public 메소드 서명 세트를 정의합니다. 인터페이스는 서비스 제공자와 요청자 간의 계약을 정의합니다. 인터페이스를 구현하려면 모든 메소드를 제공해야 합니다.
- **출력된 인터페이스** — 클라이언트가 동적으로 발견하는 레지스트리를 통해 고유하게 식별 및 사용할 수 있는 인터페이스.[3]
- **공용 인터페이스** — 클라이언트가 사용할 수 있지만 출력되지 않아 클라이언트 파트에 대한 정적 지식이 필요한 인터페이스
- **이중 인터페이스** — 인터페이스는 종종 하나의 인터페이스가 다른 인터페이스에 의존하는 쌍으로서 개발됩니다. 예를 들어, 클라이언트 인터페이스는 특정 호출 메커니즘을 제공하므로 클라이언트는 요청자를 호출하는 인터페이스를 구현해야 합니다. 이 개념은 웹 서비스에서 도입되었습니다.

그림 2 – 구현된 서비스

그림 2 에는 UML 컴포넌트로 표시되며 AccountManagement, ContactManagement 및 SystemsManagement 인터페이스를 구현하는 CRM(Customer Relationship Management) 서비

스의 UML 정의가 표시됩니다. 이 중 처음 두 개만 출력된 인터페이스이며, 나머지는 공용 인터페이스입니다. SystemsManagement 인터페이스와 ManagementService 인터페이스는 이중 인터페이스를 구성합니다. CRM 서비스는 이 중 원하는 인터페이스를 모두 구현할 수 있으며 이러한 서비스(또는 컴포넌트) 기능으로 작동 구현시 보다 나은 유연성을 허용하는 클라이언트에 따라 여러 방법으로 구현될 수 있습니다. 특정 클라이언트 클래스에 다른 또는 추가 서비스를 제공할 수도 있습니다. 일부 런타임 환경에서는 이러한 성능을 사용하여 단일 컴포넌트 또는 서비스에 대해 동일한 인터페이스의 다른 버전을 지원할 수도 있습니다.

인터페이스 작동

Java 또는 C#과 같은 언어나 IDL 과 같은 언어로 작성한 인터페이스 정의는 한 세트의 메소드 서명만 제공합니다. 정의는 방법에 대한 지침없이 대상을 제공합니다. 예를 들어, 그림 3 의 보안 인터페이스를 고려해 보십시오. 이 인터페이스의 구현을 호출하는 클라이언트가 세 가지 공용 메소드를 호출할 수 있습니까?

그림 3 – UML 의 인터페이스

대상을 간단히 정의함으로써 사용자가 로그온할 때까지 클라이언트가 GetUserName( ) 또는 GetUserDomain( )을 호출할 수 없다는 사실을 설명할 수 없습니다. 다음 상태 시스템은 이러한 종속성 또는 작동을 보여줍니다. 이러한 유형의 제한조건은 인터페이스 기반 설계에 대한 설명에 자주 포함되지만 인터페이스 구현시 작동 스펙을 준수하는지 확신할 수 없습니다.

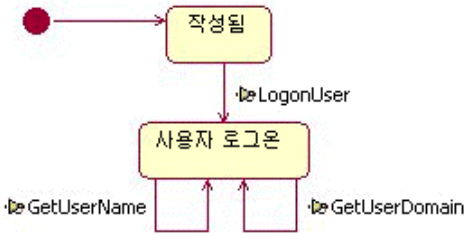

그림 4 – 인터페이스 작동

그러나 서비스 협업을 통해 비즈니스 프로세스를 구현할 수 있도록 비즈니스가 보다 쉽게 통합 및 구축될 수 있을 것이라는 희망으로 비즈니스는 점점 더 서비스 지향 시스템으로 나아가고 있습니다. 결과적으로 인터페이스 작동 및 나아가 관련 인터페이스 세트의 작동을 정의하는 표현에 대한 업계 관심이 높아지고 있습니다. 그러나 아직까지는 이와 관련된 표준 방법은 거의 없습니다.

한 가지 방법으로, 이 백서에서 소개하는 모델처럼 서비스 인터페이스 간의 종속을 설명하는 UML 과 같은 표준 언어로 정의되는 설계 모델을 사용할 수 있습니다. 이러한 모델을 공유, 공개 및 사용함으로써 필요시 특정 표준을 적용할 수 있습니다.

또한 Rational 은 이러한 문제점에 적용될 수 있는 자원 패키지화 및 공유를 위한 메커니즘을 제공하는 재활용 자원 스펙(RAS)을 후원해왔습니다. 예를 들어, RAS 메커니즘을 사용하여

서비스에 대한 세부사항을 분배하는 경우, 해당 작동에 대해 설명하는 모델을 패키지화할 수도 있습니다. 이러한 모델에서는 순서 다이어그램을 사용하여 인터페이스에 대한 호출 간에 필요한 상호작용을 표시합니다.

서비스 지향 시스템 구축

새로운 소프트웨어 엔지니어링 개발에서는 이전 프로젝트에서 사용한 동일한 기술 및 툴을 적용할 수 있는 것으로 가정할 수 있습니다. 이미 컴포넌트와 서비스가 유사하지만 동일한 개념이 아님을 설명했습니다. 이 두 가지는 설계 기준 및 설계 패턴에서 차이점이 있습니다. 이 섹션에서는 중요한 실제 결과에 대해 설명합니다. *중요한 점은 서비스로 변환된 모든 올바른 컴포넌트가 올바른 서비스를 만들어내는 것은 아니라는 점입니다.*

어플리케이션 설계 계층 구성

오래된 솔루션을 사용하여 새로운 문제점을 해결하려는 시도는 새로운 것이 아닙니다. 즉, 개발자가 컴포넌트 기반 시스템을 작성할 때 이들은 유사한 문제점에 있어 객체 지향 개발 경험을 활용했습니다. 경험이 축적되면서 객체 지향 기술 및 언어가 컴포넌트를 구현하기 위한 훌륭한 방법이기는 하지만 이러한 결정 및 구현을 통해 발생하는 트레이드오프에 대해 이해해야 된다는 사실을 알게 되었습니다.여러 작동 구현을 위한 상속 대 집계 또는 단일 C++ 어플리케이션의 기본이 아닌 컴포넌트 세트에서 사용할 수 있도록 클래스 라이브러리 재설계와 관련한 트레이드오프

마찬가지로 컴포넌트가 서비스를 구현하기 위한 가장 적합한 방법이라고 *할 수 있지만* 올바른 컴포넌트 기반 어플리케이션이 반드시 올바른 서비스 지향 어플리케이션을 만드는 것은 아님에 유의해야 합니다. 어플리케이션 구조의 서비스가 수행하는 역할을 잘 '이해하면 회사의 컴포넌트 개발자와 기존 컴포넌트를 효과적으로 활용할 수도 있습니다. 이러한 전환에서 가장 중요한 점은 서비스 지향 방법으로 인해 추가 어플리케이션 구조 계층이 필요하다는 점을 이해하는 것입니다. 아래 그림 5 는 어플리케이션 구조에 기술 계층을 적용함으로써 어플리케이션 소비자에게 접근할수록 보다 적게 세분화된 구현을 제공할 수 있는 방법을 보여줍니다.. 이 시스템 파트를 표현하기 위해 사용하는 용어는 어플리케이션 엣지로서 시스템이 기존의 컴포넌트 설계를 사용하는 내부 재활용 및 합성으로 외부 시스템 보기를 효과적으로 표시할 수 있다는 사실을 반영합니다.

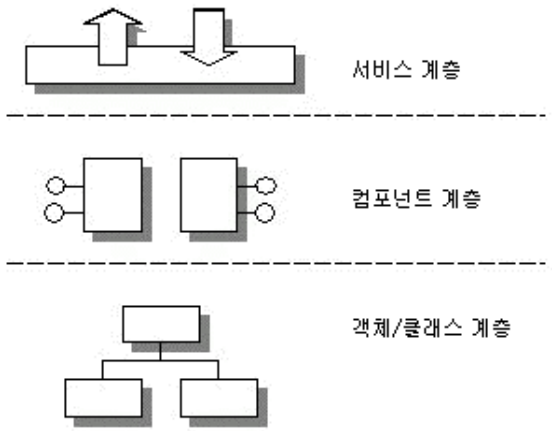

그림 5 – 어플리케이션 구현 계층

일반적으로 객체 지향 사고를 컴포넌트 기반 사고로 전환하는 데는 6 – 18 개월이 소요되며 개발자는 이 기간 동안 새로운 기술 및 관련 요구사항에 대해 학습합니다. 서비스 지향

시스템으로의 전환을 보다 빨리 수행할 수도 있습니다. 이를 위해서는 개발자가 서비스 지향 어플리케이션의 지원 하에 컴포넌트의 개발 및 재활용을 허용하는 도전 과제, 트레이드오프 및 설계 결정에 대해 이해해야 합니다.

고객 모델 예

어플리케이션을 구현하는 데 있어 컴포넌트와 서비스의 상호작용에 대해 설명하기 위해 아래 그림 6 의 UML 클래스 다이어그램에서 정의하는 대로 일부 고객 관계 정보를 관리하는 예를 고려합니다.

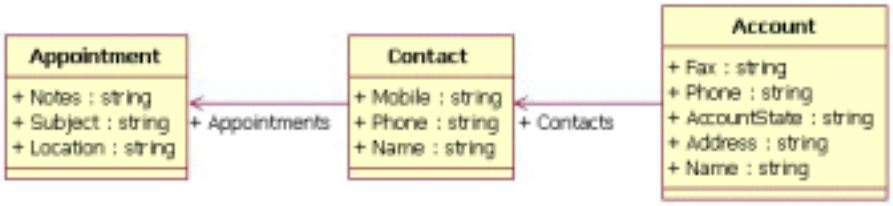

그림 6 – 논리적 고객 모델

다음 섹션에서는 개발자가 논리적 모델을 컴포넌트 기반 어플리케이션에 대한 구현 모델과 서비스 기반 어플리케이션에 대한 구현 모델로 차례로 복사하는 방법에 대해 설명합니다. (이 모델은 시스템 작동에 대해서는 설명하지 않습니다.) 이러한 많은 변환 단계가 자동으로 생성될 수 있다는 사실이 명백해질 것입니다. Rational Software 에는 어플리케이션 구조, 결과 및 해당 패턴의 어플리케이션을 모델링하고 전체 개발 라이프사이클에서 이러한 모델/코드 결과물을 관리하는 툴이 있습니다.

컴포넌트 기본 설계

논리적 모델 예를 컴포넌트 설계로 변환하는 방법은 무엇입니까(예: COM 또는 J2EE 의 경우)? 모델에 대한 컴포넌트 기반 설계는 아래 그림 7 과 같습니다.

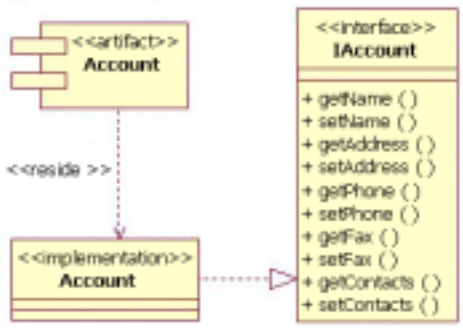

그림 7 – 일반 컴포넌트 다이어그램

위의 인터페이스에는 하나의 *핵심* 설계 기능이 있습니다. 중요한 사실은 기존 컴포넌트 플랫폼의 경우 공통 패턴이 있다는 것입니다. 즉, 분석 클래스의 각 속성에 대해 값을 설정하는 하나의 조작과 값을 리턴하는 하나의 조작 등 두 개의 조작을 제공해야 합니다. 로컬 컴포넌트의 경우, 메소드 호출 오버헤드를 무시할 수 있으며 원격 오브젝트의 경우 오버헤드를 최소화하도록 RPC(Remote Procedure Call) 메커니즘을 최적화합니다. 일반적으로 어플리케이션에서는 클라이언트가 특성의 서브세트만 필요하므로 필요에 따라 액세스할 수 있습니다.

서비스 지향 설계

위의 설계 모델은 각 컴포넌트 인스턴스가 단일 객체를 표시하는 컴포넌트 구현을 고려할 수 있는 올바른 방법을 보여줍니다. 예를 들어, CRM 데이터베이스의 각 개별 연락처는 논리적으로는 개별 컴포넌트가 됩니다. 따라서 컴포넌트 ID 는 연락처 ID 와 관련이 있습니다.

그러나 서비스의 경우, 단일 인스턴스가 자원 세트를 관리하므로 이러한 자원은 대부분 Stateless 상태입니다. 이는 서비스를 하나의 유형 또는 유형 세트의 인스턴스를 작성 및 관리할 수 있는 *관리자* 객체로 간주해야 함을 의미합니다. 이를 통해 실제로는 단순 직렬 상태의 오브젝트인 인스턴스 상태를 나타내는 가치 오브젝트를 사용하는 설계 패턴(컴포넌트 간의 전송에 대한 상태가 지속되는 분산 시스템의 공통 패턴)이 생성됩니다. 이 경우 흥미로운 결과는 그림 7 의 모델과 같은 컴포넌트 정의를 사용하여 서비스로 변환하는 규칙을 정의할 수 있는 경우, 이러한 직렬화를 패턴으로 구현할 수 있다는 것입니다. 이러한 패턴의 작성 및 재활용은 Rational XDE 를 사용하여 가능합니다.

이처럼 제공자에서 요청자로 상태를 전달하는 것은 여러 개의 소규모 조작을 사용하여 컴포넌트 상태를 검색하지 않고 하나의 대규모 조작을 사용함을 의미합니다. 이제 이는 대규모 가치 객체를 처리할 때의 요청자 작동뿐 아니라 원격 서비스(대부분의 서비스는 원격)에 대한 네트워크 사용에도 특정 의미가 있습니다. 또한 요청자에게 특정 엔티티의 상태 사본이 제공되는 상황에서 이 사본이 실효성이 있는지를 확인하는 데도 의미가 있습니다. 주식 시세 또는 일기 예보를 검색하는 경우 지난 정보가 표시될 수 있으며 이를 그대로 받아들이게 됩니다. 데이터 유형 또한 그대로 받아들입니다. 주식 시세 데이터는 일기 데이터보다도 자주 갱신되어야 합니다. 여기에서 설명하는 구조에서는 요청자가 상태 사본을 그대로 받아들여야 합니다. 세부사항은 이 백서 뒷 부분의 간략하게 세분화된 상호작용의 영향을 참조하십시오.

이러한 요구사항을 고려할 때 어떠한 서비스를 예상할 수 있습니까? 그림 8 의 모델 단편은 컴포넌트에 의해 출력된 인터페이스와 인터페이스가 조작하는 가치 객체를 보여줌으로써 설계 레벨에서 패턴을 설명하는 방법을 보여줍니다.

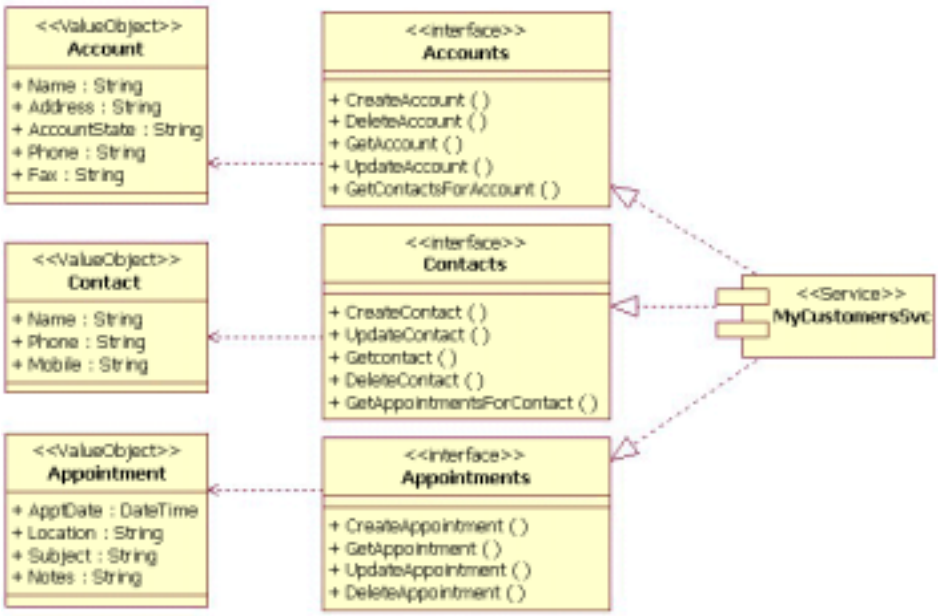

그림 8 – 일반 서비스 지향 설계

지금까지 서비스 설계 방법에 대해 설명했으므로 이제 이 특정 설계의 몇 가지 속성에 대해 알아보겠습니다. 먼저 제공자 MyCustomerSvc 로부터 해당 상호작용에 대한 요청자로의 간단한 get 또는 set 보다 많은 정보가 가치 객체에 전달되며 이로 인해 네트워크 대역폭에 영향을 줄 수 있습니다. 그러나 웹 서비스의 속성을 고려할 때 서비스 구현에서 사용되는 프로토콜은 컴포넌트 구현에서 사용되는 프로토콜과 상당히 다릅니다. 이러한 플랫폼은 아키텍트 또는 정보 엔니지어에게 추가 부담을 주어 네트워크에 무리를 주지 않고 각 가치 객체의 컨텐츠를 최대화할 수 있는 가치 객체 및 해당 합성을 선택하도록 합니다.

서비스 지향 설계 캐시

이전에 제공자에서 요청자로 오래된 "정보" 사본을 전달하는 섹션에서 설명한 개념을 다시 고려합니다. 예를 들어, 현재 주식 포트폴리오 관리 어플리케이션을 개발하고 있으며 각 주식에 대해 반복적으로 현재 주식 가격 웹 서비스를 요청하지 않으려고 합니다. 주식 데이터에는 3 - 5 자를 전달하고 주식 가격에는 5 - 7 자를 전달합니다. 이 경우 네트워크 및 서비스 제공자가 받아들일 수 없는 로드가 발생할 수 있습니다. 이 경우 요청자가 수행해야 하는 작업은 서비스에 기호 목록을 전달하거나 포트폴리오 ID 를 전달하여 각 주식에 대한 모든 정보를 검색함으로써 전체 포트폴리오의 컨텐츠를 요청하는 것입니다. 사용자가 단일 기호에 대한 갱신을 요청한 경우,무리한 요청이 될 수 있습니다. 그러나 이제 요청자는 결과를 캐시할 수 있으며 그런 다음 사용자가 다른 기호 갱신을 요청하면 캐시에서 해당 요청을 충족시킬 수 있습니다. 요청자에 대한 다음 작업은 데이터 "릴리즈" 기간을 식별하는 것입니다. 따라서 포트폴리오의 경우, 주식 시세 서비스가 20 분 지연된다는 것을 알고 있는 상황에서는 25% 여백을 두고 5 분 동안 결과를 캐시할 수 있습니다.

이 패턴은 정보 시스템에 반복적으로 표시됩니다. 사용자가 주문 관리 시스템에서 주문을 검색할 때마다 해당 사용자에게는 효과적으로 주문 사본이 제공됩니다. 이는 시스템이 주문에 대한 추가 액세스를 차단하지 않는 한 다른 사용자가 동시에 주문을 갱신하고 있기 때문입니다. 이제 웹 서비스 제공자가 요청자와의 상호작용의 일부로서 실제로 캐시 또는 릴리즈 기간을 식별해야 합니다. 이러한 사안은 MSMQ(Microsoft Message Queue Server) 및 IBM MQSeries 와 같이 메시지 제한 시간 및 만기 시간이 일상적으로 관리되는 메시지 전달 시스템에서 잘 인식됩니다.

이 문제점은 이 문서 뒷 부분에서 다시 설명하며 이 사안을 처리하기 위한 요청자 및 제공자 개발에 대한 모든 특정 지침을 제공합니다.

XML 웹 서비스 어플리케이션 설계

이제 XML 웹 서비스라는 용어를 공식적으로 사용하며 웹 서비스를 전개 및 개발하기 위한 여러 플랫폼 및 툴과 함께 다양한 벤더 지원이 제공됩니다. 웹 서비스 스택의 요소는 다른 곳에서도 자세히 설명하므로 이 문서에서는 별도로 검토하지 않습니다. 다음 정의는 W3C(World Wide Web Consortium) 웹 서비스 구조 작업 그룹에서 사용합니다.

> *웹 서비스는 URI 로 식별되는 소프트웨어 어플리케이션으로서 해당 인터페이스 및 바인딩은 XML 결과물을 정의, 설명 및 발견할 수 있습니다. 이 서비스는 또한 인터넷 기반 프로토콜을 통한 XML 기반 메시지를 사용하여 다른 소프트웨어 어플리케이션과의 직접 상호작용을 지원합니다.*[4]

서비스 지향 구조 용어를 설명할 때 그림 1 에 그린 그림에 현재 사용되는 특정 웹 서비스 기술이 적용되는 방법에 대해 간단히 설명하겠습니다.

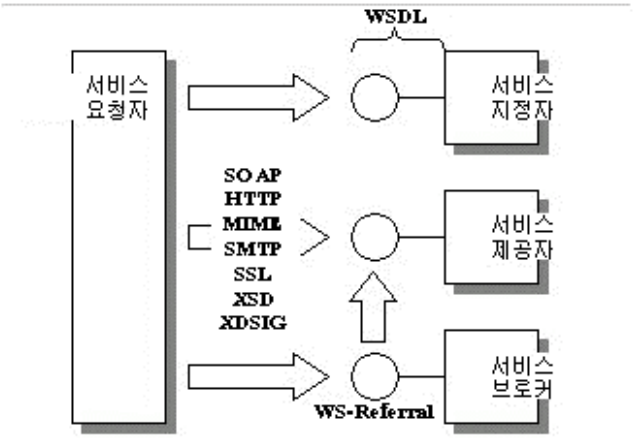

그림 9 – XML 웹 서비스 표준

먼저 모든 웹 서비스가 실제 웹 프로토콜인 HTTP 를 통한 단순 객체 액세스 프로토콜(SOAP)로 XML 메시지를 사용한다는 오해가 있습니다. 이는 사실이 아닙니다. 웹 서비스 메시지가 XML 을 사용할 수 있지만 이 메시지는 2 진 데이터를 전송할 수도 있습니다. 일반적으로 SOAP 헤더를 사용하지만 메시지 본문에 반드시 SOAP 인코딩을 사용할 필요는 없습니다. 또한 HTTP 를 전송 수단으로 사용할 수 있지만 SMTP 또는 기타 방법도 사용할 수 있습니다. 웹 서비스의 설명 또는 발견의 경우, 두 가지 잘 정의된 표준, WSDL(Web Services Definition Language) 및 UDDI(Universal Discovery, Description and Integration)가 있습니다.

이러한 형식 및 전송 프로토콜의 유연성이 현재 웹 서비스 관련 사안 중 하나인 상호 운영성입니다. 선택한 SOAP 형식, 인벨로프, 전송 프로토콜 등을 고려할 때 두 가지 구현으로 정보를 전송할 수 있는 방법은 무엇입니까? 이와 관련해서는 현재 및 신규 표준 사용에 대한 지침을 업계에 제공하는 WS-I(Web Services Interoperability) 그룹을 들 수 있습니다. 벤더 또한 IBM WebSphere Studio Application Developer Integration Edition 과 같이 여러 형식 및 전송 프로토콜을 사용하여 웹 서비스를 작성함으로써 필요에 따라 가장 빠르고 정확한 세트를 사용할 수 있는 유연한 웹 서비스 개발 환경을 제공하고 있습니다.

웹 서비스 설계 및 구현 패턴

웹 서비스는 어플리케이션에 대한 기능 요구사항과 관련한 분석 및 설계 프로세스를 실제로 변경하지 않습니다. 보험금 청구 처리 어플리케이션의 경우 계속 보험금 청구를 처리해야 합니다. 여기서 설명하는 내용은 비기능 요구사항 영역에서의 제한조건 및 잠재적 사안에 대한 것입니다. 다음 섹션에서는 이러한 가능한 몇 가지 위험에 대해 설명합니다.

성능 및 신뢰성

본질적으로 속도가 느리며 신뢰할 수 없는 HTTP 또는 SOAP 를 기반으로 하는 구조가 웹 서비스 성능,신뢰성 및 확장성에 필요한 성능을 제공할 수 있는지 여부에 대한 질문이 자주 제기됩니다. 먼저 느린 속도와 신뢰 불가능을 정의한 다음 신뢰할 수 있는 전송 수단도 결국에는 신뢰할 수 없는 방법에 의존한다는 사실을 알아야 합니다. 엔터프라이즈 규모 솔루션을 구축 및 설계하는 경우, 항상 기능 및 비기능 요구사항을 염두에 두고, 비즈니스 목표를 지원하기 위한 올바른 트레이드오프 및 결정을 수행했는지 확인해야 합니다.

예를 들어, HTTP 를 통해 SOAP 를 사용하는 경우, 메시지 수신 확인 및 보안에 대한 추가 성능을 제공하는 어플리케이션 레벨 프로토콜 및 상호작용을 빌드할 수 있습니다. 그러나 동일한 보안

또는 어플리케이션 컨텍스트에서 특정 서비스를 송수신하는 경우, HTTP 와 다른 방법을 사용할 수 있습니다. 그림 10 의 예를 고려하십시오.

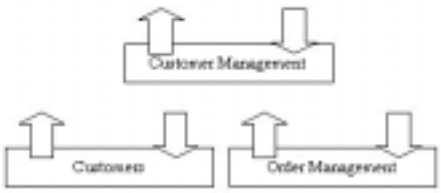

그림 10 – 서비스 종속성

기본적으로 모든 외부 클라이언트는 고객 관리 서비스와 상호작용합니다. 그러나 이 서비스는 두 개의 내부 서비스와 상호작용합니다. 여기서의 결정은 이러한 내부 서비스 의사소통에 왜 HTTP 또는 SOAP 의 유연성이 필요한가 하는 것입니다. 고객 관리와 고객 간의 상호작용에 있어 가장 중요한 요구사항을 성능으로 가정합니다. 이 경우, 2 진 인코딩 형식 및 보다 우수한 성능 특성을 제공하는 컴포넌트 RPC 통신(예: Microsoft .NET Remoting 또는 Java RMI)을 사용하도록 결정할 수 있습니다. 반면에 고객 관리에서 주문 관리로 주문을 할 때 가장 중요한 요구사항은 정확한 배달에 대한 것이므로 특정 대기열 지정 기술(IBM MQSeries 또는 MSMQ)을 사용하여 메시지를 전달할 수 있습니다. 이 경우, 성능보다는 신뢰성에 보다 높은 비중을 둡니다.

웹 서비스가 간단한 모델과 간단하면서도 유연한 프로토콜 세트로 제공되는 경우라도 이러한 선택으로 제한되지는 않습니다. WSDL 에는 이미 SOAP 및 HTTP GET/PUT 에 대한 바인딩이 있으므로 요청자에게 추가 선택사항을 제공해야 합니다. 예를 들어, 단일 서비스가 메시지 대기열 바인딩 *및* SOAP 바인딩을 사용하여 메시지를 표시함으로써 요청자가 사용하기에 보다 적합한 바인딩을 선택할 수 있습니다. 이 경우, 메시지 대기열을 사용하지만 HTTP 변환에 대한 서비스 보증이 없는 경우 제공자는 확실한 서비스 레벨과 같은 인센티브를 제공할 수 있습니다.

빠른 결정이 필요한 다른 설계 부분은 메시지 교환 방식을 멱등, 상호 또는 두 가지 모두로 할지 여부를 결정하는 것입니다. 멱등 방식을 *선택하는 경우,* 동일한 메시지가 두 번 이상 도달하여 실행되어도 각 경우에 있어 문제가 발생하지 않음을 의미합니다. 상호 방식을 *선택하는 경우,* 두 개의 관련 메시지가 순서에 관계 없이 도달해도 문제가 발생하지 않음을 의미합니다. 메시지 교환 방식이 최소 멱등 방식으로 식별되도록 서비스를 설계하는 경우, 전송 수단의 신뢰성이 낮을수록 보다 바람직한(또한 저렴한) 옵션입니다.

보안(이 백서에서는 설명하지 않음)이 실제로 단순하고 저렴한 방식에서 복잡하고 비용이 많이 드는 방식까지 여러 선택사항이 있는 것과 같이, 성능, 신뢰성 및 확장성의 설계 목표에도 여러 가지 옵션을 결정할 수 있습니다. 즉, 어떠한 보안을 어느 정도 원하는지 결정해야 합니다. 또한 비용 측면도 고려해야 합니다. 서비스는 기존 개발 방법만큼 많은 솔루션과 많은 선택사항을 제공합니다.

비동기 작동 및 대기열 지정을 통한 확장성

서비스 지향 구조에 대한 소개에서 설명한 대로 웹 서비스는 본질적으로 비동기식으로 작성하는 것이 바람직합니다. 웹 서비스와 연관된 추가 전송 오버헤드와 서비스는 본질적으로 원격이라는 기대로 인해 요청자가 응답 대기에 소요하는 시간을 줄이는 것이 중요합니다. 서비스 호출을 비동기로 작성함으로써, 별도 리턴 메시지를 사용하여 프로바이더가 응답할 수 있는 상태에서 요청자가 계속 실행할 수 있습니다. 이는 동기 서비스 작동이 바람직하지 않음을 의미하는 것은

아니며 단지 경험에 비추어 볼 때 특히 통신 비용이 높고 네트워크 지연 시간을 예측할 수 없는 경우 비동기 서비스 작동이 바람직한 것으로 입증되었기 때문입니다.

그림 11 의 예를 고려하십시오.

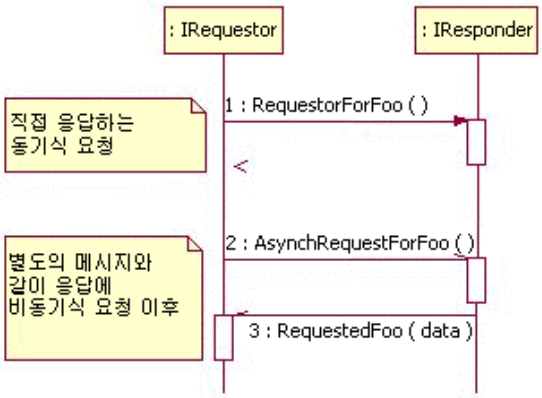

그림 11 – 동기 대 비동기

그림 11 에서 설명하는 작동은 확장성이 뛰어난 웹 서비스를 구현하는 데 있어 커다란 발전이라고 할 수 있습니다. 서비스 호출을 비동기로 작성함으로써 제공자는 여러 작업자 스레드를 사용하여 여러 클라이언트 요청을 처리할 수 있습니다. 단지 클라이언트로 빨리 리턴하는 것 외에도 조작의 비동기 모드를 지원하기 위해 수행해야 할 사항은 더 많이 있습니다. 먼저 이중 인터페이스를 지정해야 합니다. 요청자는 리턴된 메시지를 승인할 수 있는 인터페이스를 구현하는 서비스로 리턴 주소를 전달해야 합니다. 이는 대상 간의 대화 상태를 관리해야 함을 의미합니다. 이 작업을 수행하기 위한 다양한 방법에 대해 학습하려면 웹 서비스를 기반으로 하지 않는 웹 세션 설계를 참조하십시오.

그러나 이 경우 특정 규모로만 확장됩니다. 대용량 로드가 예상되는 서비스의 경우,요청자를 청취하는 파트 및 요청을 처리하는 파트의 연결을 해제해야 합니다. 이는 이미 잘 알려진 패턴으로서 메시지 대기열을 사용하여 서비스 구현으로부터 façade 서비스를 연결 해제합니다. 그림 12 의 다이어그램은 대기열을 사용하여 구현 요청을 연결 해제함으로써 제공자를 구현하는 방법을 보여줍니다.

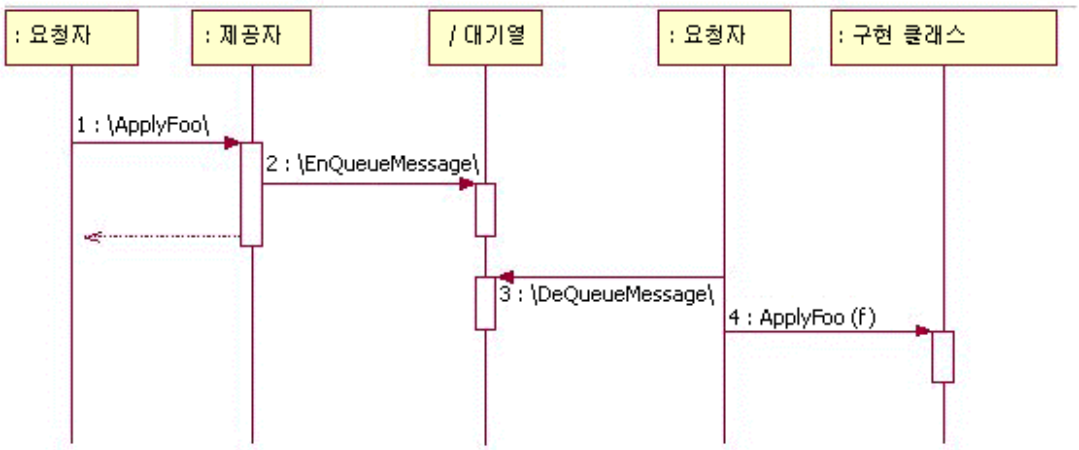

그림 12 – 대기열 지정 구현

이러한 패턴은 .NET 및 J2EE 에서 모두 해당 플랫폼에서 제공하는 서비스(.NET 의 경우 MSMQ, J2EE 의 경우 JMS(Java Message Queue Service) 또는 메시지 구동 Bean J2EE)를 사용하여 쉽게 구현할 수 있습니다. 개발자는 이러한 방식으로 보다 간단한 확장성 모델을 제공할 수 있습니다. 즉, 요청 동기화를 통해 스레드 세트를 처리하지 않고 구현으로 간단히 별도 대기열 리스너를 추가하여 여러 시스템에서도 대기열에서 메시지를 선택할 수 있습니다.

정보 릴리즈 재확인

정보 릴리즈에 생각할 때 일반적으로 집 또는 자동차와 같은 자산을 릴리즈하는 것보다는 도서관에서 책을 빌리는 관점에서 생각하는 경우가 많습니다. 요청자가 서비스를 요청하는 것은 암묵적으로 특정 정보의 사본을 요청하는 것이며 여기에는 항상 해당 시점에서의 상태 스냅샷이 제공됩니다. 이제 이 점을 명확히 이해하고 설명하지 않으면 문제점이 발생할 수 있습니다. 한 가지 전략은 제공자가 해당 정보와 함께 만기 시간을 부여할 수 있도록 하는 것입니다. 또는 요청자에게 도서관 책과 같이 릴리즈 “티켓”을 주는 것입니다. 이 티켓을 사용하면 요청자가 정보 유효성을 확인하여 릴리즈 기간을 잠재적으로 연장할 수 있으며 서버는 데이터를 다시 검색하지 않고 릴리즈를 재설정할 수 있습니다.

이제 이 문제는 HTTP, SOAP 또는 기타 전송 프로토콜이 자동으로 처리하는 근본적인 사안일 수도 있습니다. 브라우저 및 방화벽이 페이지를 캐시할 수 있는 HTTP 캐싱 의미론을 재활용할 수 있지만 이는 실제로 제공자가 제어할 수 없으며 요청자가 HTTP 를 전송 수단으로 사용하지 않을 수도 있습니다. 한 가지 옵션은 문서 교환시 이러한 지원을 빌드함으로써 그림 13 과 같이 요청자와 제공자 간의 메시지가 클라이언트에 대한 릴리즈 정보를 인코드하는 것입니다.

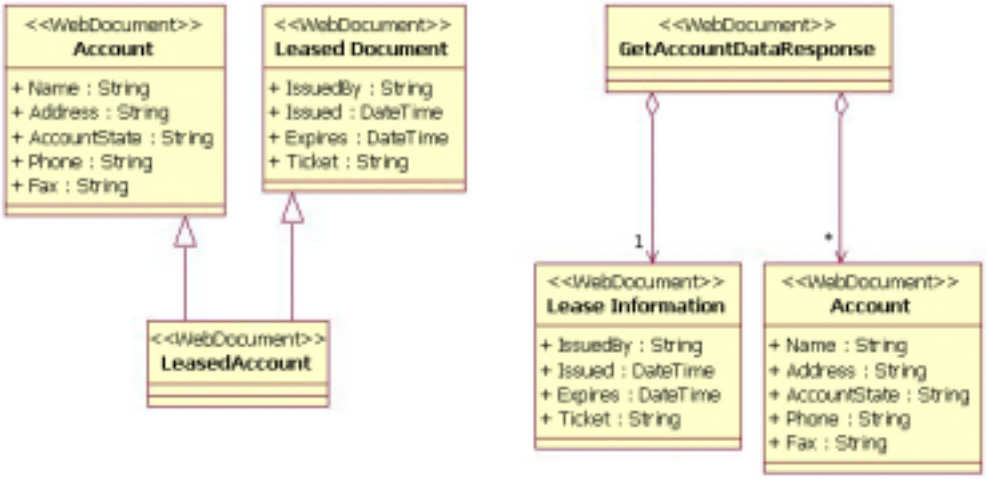

그림 13 – 두 가지 정보 릴리즈 구현

그림 13 은 정보 릴리즈 패턴의 두 가지 대안 구현을 보여줍니다. 첫 번째 방법은 상속을 사용하여 계정 XML 문서를 계정뿐 아니라 릴리즈 문서이기도 한 특별 양식으로 특수화함으로써 추가 정보를 포함하는 것입니다. 두 번째 방법은 계정과 함께 리턴되는 릴리즈 정보를 응답 메시지의 별도 파트로 사용하는 것입니다. 이러한 두 가지 방법은 모두 유효하며 구조가 다른 데이터가 발생하며 양식, 상속, 집계 중 하나만 선택하면 됩니다.

결과 웹 서비스 모델 설계

그림 14 는 웹 서비스 설계 및 개발에 적용되는 UML 프로파일을 사용하여 그림 7 의 일반 서비스 설계를 모델링하는 방법을 보여줍니다. 이 프로파일은 매우 간단하며 «WebService» 및 «WebDocument»에 대한 두 가지 새로운 스테레오타입(기존 UML 언어의 확장)에 대해 설명합니다. 이미 UML 에 있는 인터페이스 의미론을 재활용함으로써 WSDL 에 정의된 대로 서비스의 출력 측면을 쉽게 시각화할 수 있습니다.

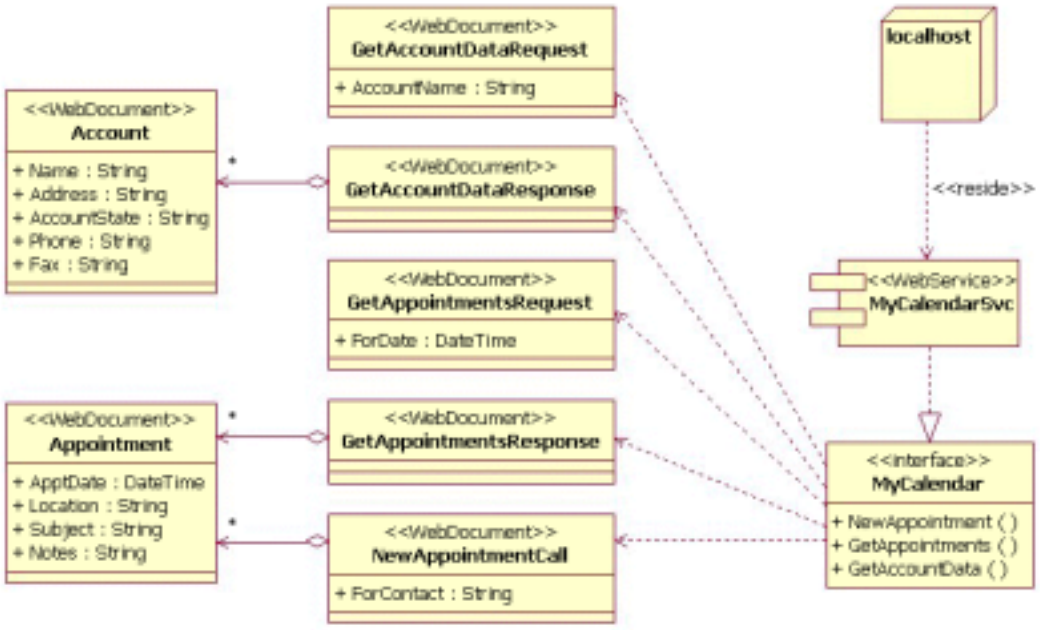

그림 14 – XML 웹 서비스 설계

아래 표에서는 그림 14 의 프로파일 요소와 MyCalendar 서비스에 대한 WSDL 의 결과물과의 관계를 보여줍니다.

WSDL 결과물	UML 요소	주석
서비스	«WebService»	서비스는 하나 이상의 인터페이스를 구현하고 특정 위치에 상주하는 UML 컨포넌트로 표시됩니다. «reside» 관계는 실제 URL 위치 정보를 캡처합니다.
포트 유형	인터페이스	각 portType 은 하나 이상의 서비스로 구현되는 UML 인터페이스로 표시됩니다. 구현 관계는 바인딩 정보를 캡처합니다.
메시지	«WebDocument»	각 메시지는 UML 클래스로 표시됩니다. XML 스키마와 UML 간의 맵핑이 메시지 및 파트 구조를 모델링해야 합니다.
파트	속성 또는 연관 종료	메시지의 각 파트는 «WebDocument»의 UML 속성 또는 다른 «WebDocument»에 대한 연관으로 표현할 수 있습니다.
주소 위치	노드	노드는 서비스가 상주하는 서버를 나타냅니다. 노드는 상주 서비스 세트를 식별할 수 있으며 서비스는 두 개 이상의 노드에 상주할 수 있습니다.

이 방법으로 웹 서비스를 설계하면 데이터 교환을 정의하는 문서와 서비스가 지원하는 인터페이스를 모두 재활용할 수 있습니다. 이는 엔터프라이즈 규모 솔루션을 설계할 때 중요한 성능입니다. 계정 문서와 상호작용하는 모든 서비스는 동일한 문서 정의를 따르도록 해야 합니다. 또한 그림 2 의 SystemsManagement 와 같이 출력되지 않은 작동 인터페이스는 이 분야의 전문가가 정의한 다음 솔루션에서 공통으로 구현되도록 사용할 수 있음을 발견했습니다.

결론

웹 서비스가 소프트웨어 산업에 있어 혁신적인 역할을 수행할 것으로 기대하고 있는 사람이 있는 반면 웹 서비스는 단지 과장된 것이라고 하는 사람도 있습니다. 다른 모든 새로운 기술처럼 진실은 더 지켜보아야 할 일입니다. 지금까지 연구한 결과로는 서비스 지향 어플리케이션의 구축에는 몇 가지 근본적인 차이점이 있지만 기존 컴포넌트 기본 개발 기술이 이러한 서비스를 구현하는 데 있어 계속 유효합니다.

그림 6 의 단일 도메인 모델(예: 계정, 연락처, 약속 모델)에서 컴포넌트 및 서비스 구현을 모두 도출할 수 있습니다. 기타 중요한 결론은 다음과 같습니다.

- 서비스 모델은 *기존* 컴포넌트 기반 모델에서 도출할 수 있습니다.
- 공통 패턴 및 설계를 적용하여 서비스 지향 모델로의 변환을 지원할 수 있습니다.

이러한 공통 패턴 및 설계는 일반화, 자동화한 다음 Rational XDE 와 같은 모델 대 모델 및 모델 대 코드 변환 툴을 사용하여 인스턴스화할 수 있습니다(아래 그림 참조).

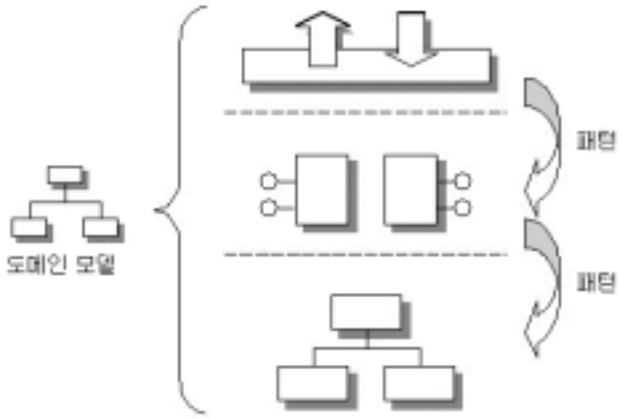

그림 15 – 서비스 구현 방법

이 방법을 사용하면 개발자의 오버헤드를 줄이고 구현의 예측성을 향상시키며 입증된 패턴을 재활용함으로써 품질을 향상시키는 등 많은 장점이 있습니다. 서비스 지향 구조 사용과 같은 베스트 프랙티스를 전문 지식에 추가하고 Rational XDE 와 같은 자동화 툴 세트에서 이러한 베스트 프랙티스의 일부를 분류하면 서비스를 구현하는 데 있어 올바른 선택과 트레이드오프 결정을 보다 빠르고 정확하게 수행할 수 있습니다.

참조서

[1] 자세한 정보는 http://www.rational.com/uml/을 참조하십시오.
[2] Large-Scale, Component-Based Development by Alan W. Brown, Prentice Hall, 2000
[3] Public versus Published Interfaces by Martin Fowler, IEEE Software, March/April 2002 (Vol. 19, No. 2)

[4] W3C Web Services Architecture Requirements; http://www.w3.org/TR/2002/WD-wsa-reqs-20020429
[5] Web Services Security, Version 1,0; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-security.asp
[6] Web Services Referral Protocol, Draft; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-referral.asp
[7] XML Web Services Basics; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-referral.asp
[8] Rational Developer Network http://www.rational.net/