멀티테넌시 지원 동적 비지니스 로직 개발에 관한 연구

이지현*
*한국전자통신연구원
e-mail : jihyun@etri.re.kr

A Study on Development of Multitenancy Supportive Business Logic

Jihyun Lee*
*BigData Software Laboratory, Electronics and Telecommunications Research Institute

요 약

본 논문은 전통적인 설치 기반 소프트웨어와 달리 웹을 통해 접근 가능한 서비스 형태의 소프트웨어에 사용자 고유의 코드를 동적으로 생성하여 반영하기 위한 방법을 다룬다. 이는 서비스 사용 회사들에게 회사별 비즈니스 로직을 생성하기 위한 플랫폼을 제공하고, 회사 별 고유한 비즈니스 로직을 개발할 수 있어 웹 기반의 소프트웨어를 손쉽게 확장할 수 있는 장점을 갖는다.

1. 서론

멀티테넌시(multitenancy)란 하나의 동일한 어플리케이션 객체를 동작시켜 다수의 기업에게 필요한 서비스를 제공하는 것을 의미한다[1][2]. 소프트웨어 서비스(Software as a Service 또는 SaaS)는 공통의 공유 호스팅 환경에서 여러 회사들(즉, 멀티테넌트)을 위한 서비스를 제공하기 위한 소프트웨어 개발 및 전달 모델이다[3][4]. 다수의 기업에게 필요한 동종의 유사한 서비스를 개별적으로 개발하거나 유지 보수하는 것은 비용이 매우 많이 든다. 또한, 기업(즉, 테넌트)의 수가 증가함에 따라 증가하는 비용은 어플리케이션 서비스를 제공하는 서비스 사업자에게는 큰 부담이 아닐 수 없다. 이와 같이 소프트웨어 서비스를 보다 효율적으로 기업에 제공하기 위한 방법으로 멀티테넌시를 지원하는 개발 방법 및 서비스 제공 방식을 고려할 수 있다.

멀티테넌시를 지원해야 하는 궁극적인 이유는 규모의 경제에 있다. 소프트웨어 서비스를 개발하거나 유지 보수하기 위한 비용을 낮추면서, 서비스를 제공받는 기업의 요구사항을 만족시킬 수 있기 때문이다.

본 논문은 멀티테넌트가 사용하는 소프트웨어의 근간이 되고 있는 멀티테넌시 지원 방법을 비지니스 로직 개발 및 변경에 대해 적용하여 살펴보고, 멀티테넌트용 로직 개발 지원 플랫폼의 구조 및 기능에 대해 살펴본다.

2. 멀티테넌시 지원 동적 비지니스 로직의 요구사항

멀티테넌트들에게 효과적으로 멀티테넌시를 지원하기 위해서 멀티테넌트 지원 플랫폼과 멀티테넌트 지원 어플리케이션이 마련되어야 한다. 멀티테넌시를 플랫폼에서 지원한다는 것은 테넌트가 스스로 기업의 요구사항에 맞는 서비스를 쉽고 저렴하게 생성할 수 있음을 의미한다. 이를 위해 플랫폼은 테넌트가 원하는 구성 요소를 변경시킬 수 있도록 개별화(customization) 도구 및 체계화된 방법을 제공해야 한다. 여기서 구성 요소는 사용자 인터페이스(user interface, 즉 UI), 비지니스 로직, 데이터 스키마에 해당한다. 본 논문에서는 비지니스 로직의 개별화에 대해 한정하여 다루며, 특별히 코드 기반의 비지니스 로직을 대상으로 개발 및 개별화한다.

멀티테넌시를 지원하는 동적 비지니스 로직을 처리하는데 고려해야 하는 플랫폼 요구사항과 어플리케이션 요구사항에 대해 살펴보면 다음과 같다.

(1) 플랫폼 요구사항
- 플랫폼 지원 동적 컴파일 기능이 필요함
- 플랫폼 지원 비지니스 로직의 동적 로딩이 필요함
- 비지니스 로직 생성을 지원하는 도구가 필요함
- 비지니스 로직은 플랫폼을 재 시작 시키지 않고 런타임에 반영되어야 함

(2) 어플리케이션 요구사항
- 비지니스 로직을 설정할 수 있는 가변점이 정해 져 있어야 함
- 새롭게 비지니스 로직의 코드를 생성할 수 있어야 함
- 비지니스 로직 코드에서 필요한 쿼리를 새로 정의할 수 있어야 함
- 변경된 비지니스 로직은 사용자 인터페이스에 연결되어 호출되어야 함
- 테넌트 별 개별화된 비지니스 로직은 다른 테넌트에게는 보이지 않아야 함

멀티테넌시를 지원하는 어플리케이션의 구조는 비지니스 로직의 변경에 따라 데이터 스키마, 사용자 인터페이스, 비지니스 로직의 구조 및 기능에 영향 주지 않도록 구조화되어야 한다. 이를 위해 멀티테넌시를 지원하는 소프트웨어 서비스의 구조는 모델-뷰-컨트롤러의 구조로 나눠 구성하고, 컨트롤러의 변경

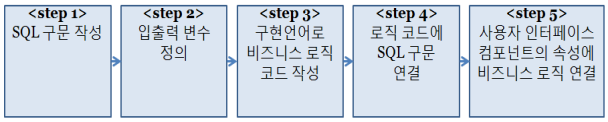
을 통해 비지니스 로직의 코드를 변경시킨다.

3. 멀티테넌시 지원 비지니스 로직 개발 프로세스

멀티테넌시 지원 비지니스 로직 개발은 크게 두 단계로 구성된다.

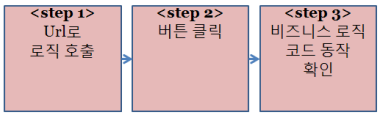
- 1 단계: 비지니스 로직을 코드로 작성한다.
- 2 단계: 비지니스 로직에서 다루고자 하는 데이터를 CRUD 연산의 결과로 얻는다.

(그림 1)은 테넌트 독립적으로 비지니스 로직의 코드와 얻고자 하는 쿼리 구문을 작성하는 과정을 보여준다. 두 가지 구성 요소는 사용자 인터페이스의 특정 컴포넌트 (예: 버튼)에 연결되고 특정 URL 을 호출하여 실행된다.



(그림 1) 비지니스 로직 개별화

멀티테넌시 지원 비지니스 로직은 개발된 후 플랫폼의 중단없이 동적으로 탑재되어 실행 결과가 브라우저로 보인다. 코드 기반 비지니스 로직의 실행은 (그림 2)와 같이 특정 URL 을 통해 웹 브라우저에서 호출되면, 테넌트별로 개별화된 비지니스 로직을 찾아 플랫폼 안에서 동적으로 컴파일하고 가상머신으로 로딩하여 결과를 브라우저로 전달한다.



(그림 2) 비지니스 로직 호출

4. 구현 기술 및 결과

멀티테넌시를 지원하는 핵심 기술인 비지니스 로직의 동적 동작 지원 기술인 재배포와 재로딩, 동적 컴파일에 대해 살펴본다.

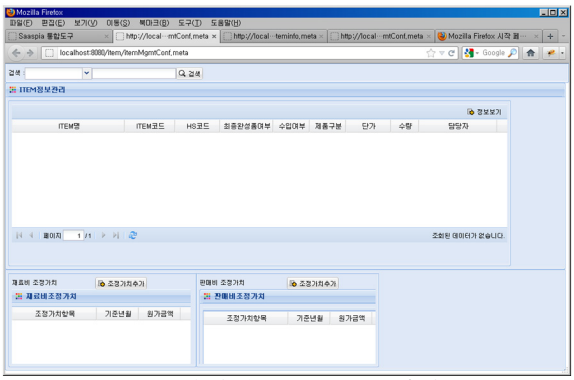
먼저, 재배포(redeploy)와 재로딩(dynamic reloading)의 차이점을 살펴본다. 웹 어플리케이션에 작은 변경이 발생했을 때, 전체 어플리케이션을 재배포할 필요가 있는가? 없다. 대신 변경된 부분에 대한 실행 가능한 인스턴스를 생성하여 웹 서버에 직접적으로 반영할 수 있다면 어플리케이션 전체 재배포보다 효율적이다.

다음, 동적 컴파일(dynamic compilation)의 동작 방법에 대해 살펴보자. 동적 컴파일은 웹 어플리케이션을 구성하는 페이지와 관련 기능을 구현한 코드 파일들을 다시 컴파일하기 때문에 사용자에 대한 웹 어플리케이션의 반응이 느려질 수 있음을 의미한다. 물론 웹 어플리케이션이 맨 처음 실행될 때에는 코드 파일에 대한 동적 컴파일이 필수이다. 하지만, 그 이 후에는 매번 컴파일 할 필요가 없다. 게다가 컴파일 해야 하는 파일이 많아지면 사용자는 더 많은 반응 시간을 기다려야 함을 의미하므로, 변경된 부분에 대한 코드만 동적으로 컴파일 하는 것이 바람직하다. 변경된 비지니스 로직에 대한 동적 컴파일 하도록 한다.

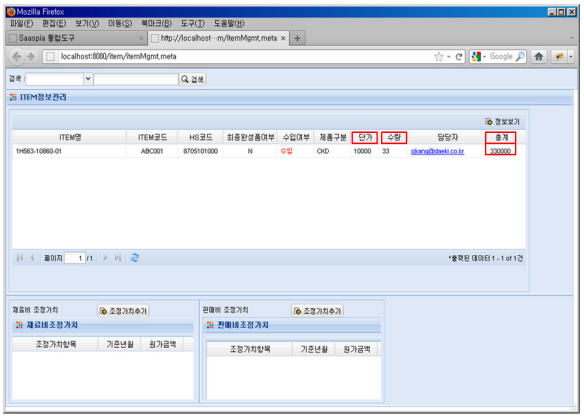
비지니스 로직을 생성하여 저장하는 핵심 기술로는 메타데이터 기반 비지니스 로직 코드 생성이 있다. 비지니스 로직의 개별화 정도를 높이기 위해 쿼리 구문, 코드, UI 컴포넌트의 메타데이터로 나눠 저장하고, 테넌트로부터 특정 비지니스 로직의 실행이 요청될 때 UI 컴포넌트에 매핑된 비지니스 로직과 쿼리 구문을 찾아 동적 파일을 생성한다.

본 논문의 제안 방식은 멀티테넌시 지원 비지니스 로직을 테넌트 별로 재로딩, 신규 로직 코드에 대한 컴파일, 메타데이터 기반의 로직 저장을 원칙으로 한다.

동적으로 생성되고 실행되는 멀티테넌시 지원 비지니스 로직의 실행 결과는 (그림 4)와 같으며, 실행 전 (그림 3)에서 “총계”의 필드가 UI 에 추가되고, 총계는 “단가*수량”을 계산 하여 추가된 “총계” 영역에 출력한다.



(그림 3) 비지니스 로직 개별화 전



(그림 4) 비지니스 로직 개별화 후

참고문헌

[1] Craig D Weissman, Steve Bobrowski, “The Design of the Force.com Multitenant Internet Application Development Platform,” Proc. of International Conference on Management of Data, pp. 889-896, 2009.

[2] Javier Espadas, David Concha, Arturo Molina, “Application Development over Software-as-a-Service platforms,” Proc. of International Conference on Software Engineering Advances, pp. 97-104, 2009.

[3] Frederick Chong, Gianpaolo Carraro, “Architecture strategies for catching the long tail,” MSDN, 2006.

[4] Wei Sun, Kuo Zhang, Shyh-Kwei Chen, Xin Zhang, Haiqi Liang, “Software as a Service: An Integration Perspective,” Proc. of International Conference on Service Oriented Computing, pp. 558-569, 2007.