Метаданные в REST - модели

Намиот Д.Е., к.ф.-м.н., старший научный сотрудник, dnamiot@gmail.com
Московский государственный университет имени М.В. Ломоносова

Аннотация. Анализируются подходы к представлению метаданных в программных моделях, основанных на Representational State Transfer (REST) подходе. В настоящее время, архитектура на основе REST- одна из наиболее часто используемых программных архитектур. Распространение данного подхода, не в последнюю очередь, было связано с простотой программной реализации и лаконичностью его описания в сравнении сервис-ориентированной архитектурой (Service Oriented Architecture), с которой REST чаще всего и сравнивают. Вместе с тем, стремление к простоте (лаконичности) реализации привело к тому, что такой элемент как метаданные в классической модели REST отсутствует. Настоящая работа посвящена поиску ответов на вопросы: зачем метаданные могут быть нужны; что и как они могут представлять; какие инструментальные средства могут быть использованы для их представления в системах на основе REST-архитектуры.

Ключевые слова: веб-сервисы, SOAP, REST, метаданные

Введение

Подход к построению программных систем на основе архитектуры, называемый Representational State Transfer (REST или REST-сервисы) является наиболее часто используемым, особенно для веб-приложений [1]. Архитектурная модель REST является довольно простой в реализации и привлекательной для архитекторов программных систем и разработчиков. Она предлагает небольшой набор базовых принципов, которые позволяют

реализовывать высокопроизводительные и масштабируемые программные сервисы. К числу таких базовых принципов могут быть отнесены: унифицированный интерфейс; отсутствие необходимости поддержки состояний; понятная структура запросов (в отношении последнего в англоязычной литературе часто используется словосочетание self-descriptive messages).

Единое (унифицированное) представление означает наличие одного и того же интерфейса для всех клиентов. В других системах это может быть совсем не так. Например, в сервисной архитектуре (SOA), с которой чаще всего сравнивают REST, для каждого из ресурсов может быть задан свой собственный интерфейс [2]. Именно персонализация вызывает необходимость сохранять информацию о состоянии в SOA - запросах. Технология REST (что особенно важно в связи с ростом нагрузки в сервисах) хорошо масштабируется и характеризуется прозрачными принципами оценки производительности. Унифицированное наименование ресурсов также относится к числу базовых принципов REST модели. Уникальные идентификаторы ресурсов в модели REST (URI) просты для понимания и использования в работе. Перечисленные качества обеспечивают популярность подхода REST по сравнению с SOA. Например, все программные интерфейсы для популярных Интернет-сервисов (Facebook, Twitter и т.п.) реализованы на основе REST- архитектуры. Подход на основе архитектуры REST лежит в основе моделей, которые предлагаются для Internet of Things (Web of Things) [3]. Вместе с тем, в модели REST отсутствует один важный момент из архитектуры SOA, а именно – метаданные. Необходимость описывать различные ресурсы и поддерживать механизмы персонализации доступа (обращения) привела к тому, что в SOA изначально присутствовал элемент, который описывал сами данные. Описание сервисов – неотъемлемая часть SOA, и Web Service Definition Language (WSDL) [4] есть неотъемлемая часть спецификации любого веб-сервиса. Утверждение о том, что различные сервисы могут иметь разные

интерфейсы – базовое положение SOA. Соответственно, без наличия описания таких интерфейсов система функционировать не может. Язык WSDL позволяет описывать интерфейсы в терминах входящих и выходящих сообщений. Напротив, вся идея REST-модели построена на простых сообщениях, которые не требуют описания и, как следствие, метаданные в REST-модели отсутствуют. Они не являются частью процесса (модели) как в SOA. Отказ от метаданных при введении REST-подхода представлялся как его преимущество. В каких-то случаях – это верное утверждение. Так объем подготавливаемых данных в REST модели существенно меньше, чем, например в SOA. Вместе с тем нельзя не отметить, что есть достаточно примеров, когда наличие метаданных просто необходимо. Например, все проекты, связанные с автоматизацией программирования (автоматизацией использования API). Такая автоматизация возможна только при наличии формальных описаний программных интерфейсов. Генератор программ может получить исходную информацию об используемых программных интерфейсах только из некоторого формального описания этих интерфейсов. Именно такое формальное описание и представляет собой метаданные. Самым очевидным примером, где использование автоматизации программирования будет необходимо, являются приложения для Internet of Things (IoT) или для Machine to Machine (M2M) - взаимодействия [5]. Большое количество устройств, и, как следствие, большое количество программных интерфейсов, затруднит (сделает в итоге невозможным) ручное кодирование. Таким образом, задача представления метаданных для REST-интерфейсов является актуальной. Настоящая статья содержит краткий анализ подходов к представлению метаданных в REST - модели.

Роль метаданных в REST- модели

Во-первых, следует отметить, какая информация может быть представлена в качестве метаданных для модели REST. Список потенциальных кандидатов, на самом деле, следует из описания WSDL (рис. 1). Например, самый

очевидный кандидат на представление в наборе метаданных – это точка доступа (в WSDL 1.1 – Port). В случае SOA-модели разработчики могут получить адрес для обращения к веб-сервису из документа WSDL. В случае REST API такой возможности нет. Как правило, описание API для REST-модели представляет собой текстовый (HTML) документ.

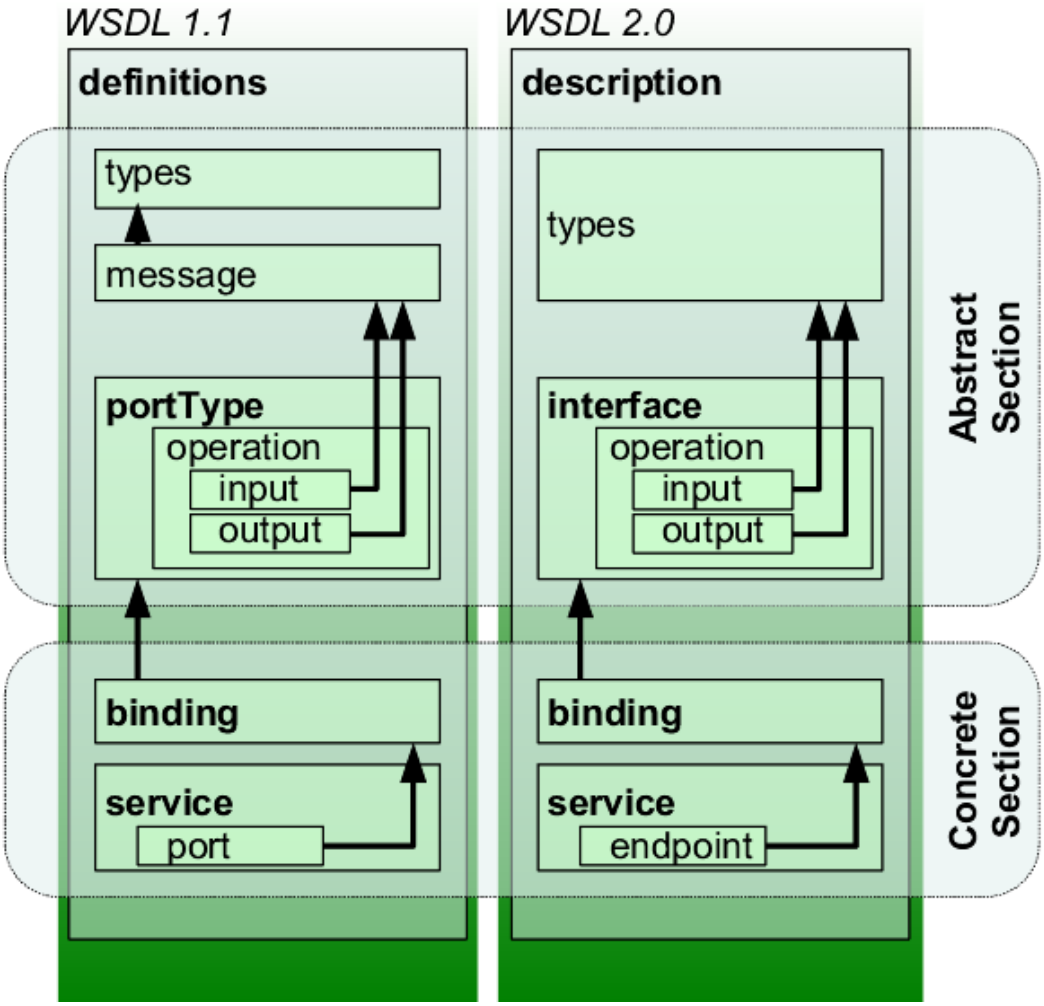

Рис. 1 Структура WSDL-документа [6]

В целом, можно выделить предложенные далее элементы для метаданных в модели REST.

1. Описание точек доступа, URLs для запросов, пути для ресурсов.

2. Методы для доступа к ресурсам. Это HTTP-команды, однако каким-то образом необходимо понимать, например, где использовать HTTP GET, а где HTTP POST.

3. Параметры в запросах.

4. Заголовки HTTP. Они часто используются в REST, например, для определения типов данных.

5. Коды возвратов и сообщения об ошибках.

6. Поддержка версий.

7. Форматы для результатов запросов. С практической точки зрения (по частоте использования), JSON, несомненно, превалирует в представлении результатов, однако единственно возможным не является.

Можно отметить, что для некоторых из перечисленных выше элементов существуют уже устоявшиеся решения (своего рода де-факто стандарты), которым негласно следует большинство реализаций. В частности, задание версий часто производится в URL для запроса. Например, обращение к Twitter API:

https://api.twitter.com/1.1/statuses/mentions_timeline.json,

Здесь *1.1* – это и есть номер версии.

Другой "общепринятый" (повсеместно используемый) элемент, который также присутствует в данном примере – это расширение. Оно используется в URL и описывает формат результата. В данном случае (расширение *.json*) – это формат JSON, а не XML. Вместе с тем, необходимо заметить, что все представленные выше примеры не более чем "псевдо-метаданные", которые не могут служить полноценной заменой (аналогом) WSDL в SOA.

Поддержка метаданных в подходе REST

Далее остановимся на программных средствах, поддерживающих определение метаданных в REST-моделях.

Превалирующей на настоящее время идеей является разметка кода (аннотации), которая может быть использована для автоматической генерации документации, клиентского кода и тестовых примеров. К такого рода продуктам, например, относятся Swagger [7] и API Blueprint [8].

Программное средство Swagger создает JSON- файлы по аннотированному серверному коду. С его помощью может также создаваться клиентский код на различных языках программирования. Средство API Blueprint, в первую очередь, ориентировано на создание интерактивной документации. Далее представлен пример разметки кода (описания). Эта разметка (аннотации) и используется для автоматического создания программной документации:

```
FORMAT: 1A
HOST: http://api.gtdtodoapi.com
# GTD TODO API
This is an example API, written as a companion to a blog post at SendGrid.com
## Folder [/folder{id}]
A single Folder object, it represents a single folder.
Required attributes:

- `id` Automatically assigned
- `name`
- `description`
Optional attributes:
- `parent` ID of folder that is the parent. Set to 0 if no parent
- `meta` A catch-all attribute to add custom features
+ Parameters
+ id (required, int) ... Unique folder ID in the form of an integer
+ Model (application/hal+json)
+ Body
 {
"id": 1,
```

```
"name": "Health",
"description": "This represents projects that are related to health"
"parent": 0,
"meta": "NULL"
}
```

Результатом обработки такого кода становится готовая к публикации документация. Она представляет собой набор HTML-файлов (XML с соответствующими стилями). Результат их отображения выглядит примерно так:

```
Function: Retrieve a single Folder
Command: GET
Path: /folder{id}
Parameters Name:     id
Description  Details: Unique folder ID in the form of an integer
Type:    int, required
```

По существу, система работает как генератор статических веб-сайтов. Системы, базирующиеся на разметке кода, не ставят своей целью создание некоторого репозитория с описанием API. Их основная задача – помощь конечным разработчикам, которые сами пишут код, а не автоматизация программирования. Другой вопрос, который в этой связи возникает – это отсутствие стандарта на такую разметку. Если следовать модели (архитектуре) SOA, то синтаксис разметки следовало бы как-то стандартизовать. Это, по крайней мере, позволило бы переиспользовать код в разных проектах.

Следующий пример – это системы, которые ориентированы на разработку API и поддержку всего жизненного цикла работы с ними

(документирование, тестирование, анализ производительности и т.д.). Типичный пример – Anypoint Platform [9]. Эта система включает средства для создания (дизайна) API, тестирования, кодо-генерации, шлюз для запуска разработанного REST API, а также средства сбора статистики и мониторинга (рисунок 2).

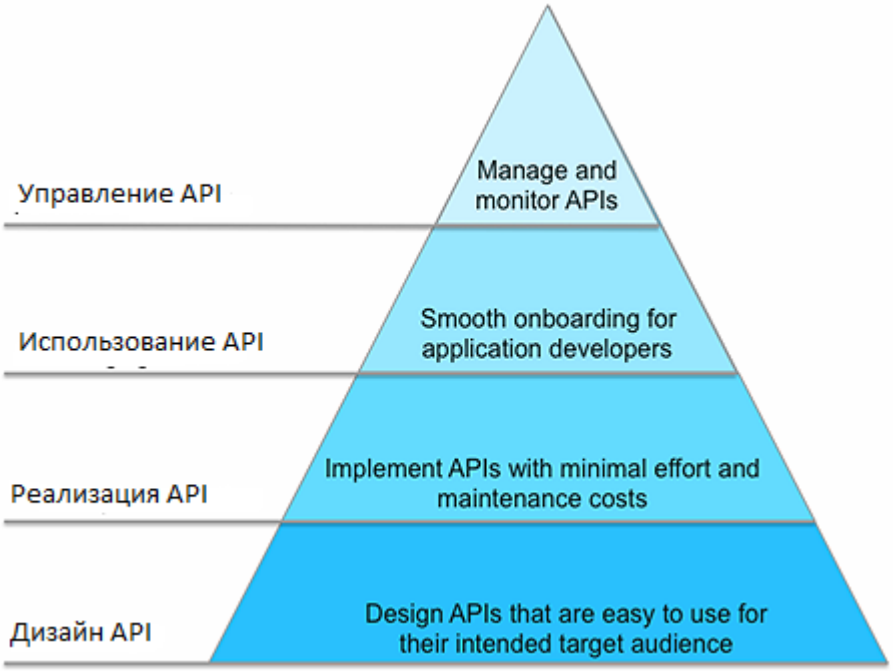

Рис. 2 Полная система поддержки API [10].

Проблема с такого рода порталом (порталами) состоит в том, что любая практическая разработка должна не только создавать новые API, но и оперировать с несколькими уже существующими. Применительно к веб-приложениям такой подход (одновременное использование нескольких API) часто называют мэшап. В этом случае подключение к закрытой системе управления новых API – это отдельная задача. Она как-то может быть решена самим производителем программного продукта за счет выпуска набора модулей подключения (адаптеров), например, для популярных веб-интерфейсов (Facebook, Twitter и т.д.). Однако вопрос подключения остается открытым для множества API в M2M и IoT- приложениях. Сам по себе такой портал представляет собой закрытую систему. Например, существует используемый внутри системы язык разметки (аннотаций), однако он также не является каким-либо стандартом. Аннотированный в данной системе код

нельзя использовать в другом приложении. В этом смысле, система ничем не отличается от описанного выше Blueprint.

Фактически, в таком портале по управлению API метаданные для REST API создаются, однако их использование носит внутренний характер. Все ограничено рамками самого портала. Из других похожих систем можно назвать Intel Mashery [11], которая также представляет собой платформу для управления API.

Следующая группа инструментальных средств – это языки описания. К их числу можно отнести WADL [12], HAL [13], RSDL [14].

HAL (Hypertext Application Language) представляет собой текстовый документ (JSON или XML) для описания ссылок и ресурсов. Если для описания API используется HAL, то это описание должно помочь в определении того, по каким ссылкам необходимо обращаться к тем или иным ресурсам. Ниже приведен пример HAL-документа:

```
{
            "_links": {
                "self": { "href": "/orders/124" },
                "ea:basket": { "href": "/baskets/97213"
},
                "ea:customer": { "href":
"/customers/12369" }
            },
            "total": 20.00,
            "currency": "USD",
            "status": "processing"
        }
```

В этом фрагменте описаны ссылки (*self, basket, customer*) и ресурсы (*total, currency, status*).

Web Application Description Language (WADL) предлагает некоторую схему XML для описания веб-приложений. Наиболее близкий аналог подобного

подхода - WSDL. Такой XML- документ не зависит от платформы. Рассмотрим пример того, как выглядит описание сервиса на WADL:

```
<resources
base="http://api.search.yahoo.com/NewsSearchService/V1/
">
     <resource path="newsSearch">
       <method name="GET" id="search">
         <request>
           <param name="appid" type="xsd:string"
             style="query" required="true"/>
           <param       name="type"       style="query"
default="all">
             <option value="all"/>
             <option value="any"/>
           </param>
           <param      name="results"      style="query"
type="xsd:int" default="10"/>
         </request>
         <response status="200">
           <representation mediaType="application/xml"
             element="yn:ResultSet"/>
         </response>
         <response status="400">
           <representation mediaType="application/xml"
             element="ya:Error"/>
         </response>
       </method>
     </resource>
   </resources>
```

Здесь указан запрос (*request*), параметры и описан отклик (*response*). Язык WADL был предложен как возможный стандарт для консорциума W3C. Формально, эта спецификация не изменялась с 2009 года [15]. Имея спецификацию WADL, можно, естественно, автоматически создавать программные клиенты. Есть готовые решения для такого рода продуктов (Oracle [16]). Решение обратной задачи (создание спецификаций по некоторому программному коду) требует опять использование некоторой разметки. Упомянутое выше решение от Oracle ориентировано на язык программирования Java и использует разметку, соответствующую спецификациям JavaDoc. Это выглядит следующим образом:

```
// The Java class will be hosted at the URI path
"/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified
by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

Курсивом выделены элементы, которые могут быть использованы для генерации WADL - файла. Естественно, что это решение только для языка Java и разметка не совместима, например, с упомянутым выше API Blueprint.

Для языка Java эти аннотации описывались соответствующей спецификацией JSR-311 [17].

По факту (на практике), сообществом разработчиков WADL не принят и массового применения не имеет. Доказательством этого утверждения служит факт отсутствия формальных спецификаций для массово используемых REST API (Facebook, Twitter). По мнению автора, WADL мог бы стать самым реальным кандидатом на роль переносимого стандарта для представления метаданных в REST - моделях.

RESTful Service Description Language (RSDL) – также предлагает использовать XML для описания веб-приложений. Можно считать, что в этом случае реализуется более простая схема, чем WADL. Для каждого ресурса (URI) в RSDL описывается формат HTTP запроса (заголовки и параметры), а также соответствующий отклик. Ниже приведен фрагмент RSDL описания:

```
<general rel="*" href="/*">
    <request>
      <headers>
        <header required="true|false">
          <name />
          <description />
          <value />
        </header>
      </headers>
      <url>
        <parameters_set>
          <parameter              context="query|matrix"
type="xs:string"
            required="true|false">
            <name />
```

```
            <value />
          </parameter>
        </parameters_set>
      </url>
    </request>
    <name />
    <description />
  </general>
```

Следует отметить, что RSDL также на настоящее время не получил широкого распространения.

Заключение

В настоящей работе анализируются различные программные средства (продукты), которые применяются или могут применяться для поддержки задания метаданных в приложениях, использующих архитектуру REST. Можно заключить, что на настоящее время не существует некоторого единого стандарта для решения таких задач. Такого стандарт для представления метаданных в модели REST не существует как де-юре (нет никакой единой стандартной спецификации), так и де-факто (нет какого-то лидирующего продукта/подхода используемого большинством разработчиков). Вместе с тем, представляется, что задача описания метаданных для REST-модели является весьма актуальной. Основным движущим мотивом к ее решению является необходимость автоматизации программирования и верификации созданных программ. Наибольшую актуальность этот вопрос приобретет в приложениях, связанных с большим количеством интерфейсов, основанных на REST-архитектуре. Как потенциально важные на перспективу можно рассматривать M2M и IoT-приложения. Как следствие, высока вероятность того, что появятся некоторые де-факто стандартные решения по представлению метаданных для

REST-модели, ориентированные на конкретный класс моделей (конкретную прикладную область).

Библиография

[1] Riva C., Laitkorpi M. Designing web-based mobile services with REST //Service-Oriented Computing-ICSOC 2007 Workshops. – Springer Berlin Heidelberg, 2009. – pp. 439-450.

[2] Monfort V., Hammoudi S. Towards Adaptable SOA: Model Driven Development, Context and Aspect //Service-Oriented Computing-ICSOC 2007 Workshops. – Springer Berlin Heidelberg, 2009. – pp. 175-189.

[3] Namiot D., Sneps-Sneppe M. On IoT Programming //International Journal of Open Information Technologies. – 2014. – V. 2. – №. 10. – pp. 25-28.

[4] Curbera F., Duftler M., Khalaf R., et al. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI //IEEE Internet computing. – 2002. – V. 6. – №. 2. – pp. 86-93.

[5] Namiot D., Sneps-Sneppe M. On M2M Software //International Journal of Open Information Technologies. – 2014. – V. 2. – №. 6. – pp. 29-36.

[6] WSDL Tutorial http://www.teqlog.com/wsdl-example-explained-step-by-step.html

[7] Izquierdo J. L. C., and Jordi **C.** "Composing JSON-Based Web APIs. Web Engineering. Springer International Publishing, 2014. 390-399.

[8] Apiary http://apiary.io

[9] Li W., Svard P. REST-based SOA application in the cloud: a text correction service case study// In Services (SERVICES-1), 2010 6th World Congress on. IEEE, 2010. P. 84-90

[10] API Management http://blogs.mulesoft.org/raml-apikit-api-hierarchy/

[11] Raivio Y., Luukkainen S., Seppala S. Towards Open Telco-Business models of API management providers //System Sciences (HICSS), 2011 44th Hawaii International Conference on. – IEEE, 2011. – pp. 1-11.

[12] Hadley M. J. Web application description language (WADL). – Technical Report, Sun Microsystems, Inc. Mountain View, CA, USA 2006.

[13] Kelly M. JSON Hypertext Application Language. IETF Draft. https://tools.ietf.org/html/draft-kelly-json-hal-02

[14] Pasternak M. Restful service description language. U.S. Patent Application 13/656,032.

[15] Спецификация WADL http://www.w3.org/Submission/wadl/

[16] Project Jersey http://docs.oracle.com/cd/E19776-01/820-4867/book-info/index.html

[17] JSR-311 https://jcp.org/en/jsr/detail?id=311