Санкт-Петербургский государственный университет
Математико-механический факультет

Кафедра системного программирования

Автоматическая генерация каркасов клиентских приложений для систем с сервисно-ориентированной архитектурой

Дипломная работа студента 545 группы

Аязяна Арама Арменовича

Научный руководитель	…… / подпись /	Н.И. Артамонов
Рецензент	…… / подпись /	Д.Г. Глиненко
“Допустить к защите” заведующий кафедрой, д.ф.- м.н., профессор	…… / подпись /	А.Н. Терехов

Санкт-Петербург
2007

Оглавление

Введение

В настоящее время мы наблюдаем очередной виток в развитии технологии программирования и подходов к реализации сложных гетерогенных систем. Большинство приложений, особенно это заметно в секторе корпоративных разработок, переводятся на web-платформу. Системы CRM[1], ERP[2], B2B[3] и прочие информационные системы уровня предприятий представляют собой наилучший пример внедрения таких технологий и стандартов как: XML[4], SOAP[5], WSDL[6], UDDI[7] и ряда других, которые вместе составляют стек протоколов Web-служб[6] и применительно к которым термин СОА (Сервисно-Ориентированная Архитектура [7]) используется чаще всего.

СОА представляет собой подход к разработке приложений, где каждый сервис выполняет единственную, строго определенную для него задачу и является независимым от других компонент, обеспечивая слабую связность. СОА обеспечивает стандартизированный набор услуг предоставления, поиска, взаимодействия и использования служб для создания требуемого поведения системы, построенной из этих служб.

Важно отметить, что сервисы могут быть доступны клиентским приложениям без знания технических подробностей, деталей реализации или платформы на которой они построены, что выгодно отличает стек протоколов Web-служб от других стандартов и технологий построения распределенных приложений как RPC[8] или CORBA[9].

Клиентскому приложению достаточно располагать описанием Web-службы на языке WSDL, чтобы воспользоваться услугами предоставляемыми Web-службой. Для ситуаций, в которых клиенту не известен поставщик конкретной услуги, есть возможность, используя UDDI, получить всю необходимую информацию о том, кто предоставляет подобную услугу и как ей воспользоваться.

Наиболее распространенный способ реализации Web-служб – использование протоколов SOAP и HTTP[10]. Так как взаимодействие между сервисами осуществляется при помощи протокола HTTP, сервисы могут взаимодействовать через Интернет, они могут быть развернуты на разных платформах. Например, сервисы, написанные на Java и .NET, могут быть объединены в рамках процесса и предоставлять функциональность более высокого

1 CRM (Customer Relationship Management System) – система управления взаимодействием с клиентами.
2 ERP (Enterprise Resource Planning System) — система планирования ресурсов предприятия
3 B2B (Business To Business System) – система предназначенная для деловых отношений между корпоративными партнёрами
4 XML (eXtensible Markup Language) – расширяемый язык разметки [1]
5 SOAP (Simple Object Access Protocol) – простой протокол доступа к объектами [2]
6 WSDL (Web Services Description Language) – язык описания Web-служб [3,4]
7 UDDI (Universal Description Discovery & Integration) – инструмент для размещения описаний Web-служб и для последующего их поиска [5]
8 RPC (Remote Procedure Call) – удаленный вызов процедур
9 CORBA (Common Object Request Broker Architecture) – общая архитектура брокера объектных запросов
10 HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста

уровня. При этом отпадают специфичные ограничения на конфигурацию сетей передачи данных, так как в большинстве случаев HTTP трафик разрешен на всех брандмауэрах.

Стоит так же отметить, что на ряду с достаточно тяжеловесным и наиболее часто используемым стеком протоколов Web-служб, существуют более легковесные решения, например, REST[11]. REST представляет собой набор архитектурных принципов при разработке приложений. Основные принципы: состояние и функциональность приложения поделены на ресурсы; каждый ресурс может быть уникально идентифицирован на основе гиперссылок; ресурс предоставляет набор доступных операций; все операции над ресурсом доступны по HTTP; состояние ресурса также доступно по HTTP.

Неотъемлемой частью перевода приложений на web-платформу является создание функционально богатых клиентских приложений, так называемых RIA (Rich Internet Applications). Если раньше web-сайты предоставляли пользователю лишь возможность просмотра информации, то современные интернет-приложения приближаются по своим возможностям к настольным. В качестве примеров таких интернет-приложений можно привести широко известные GMail[12], Meebo[13] и Netvibes[14].

До появления платформы Web 2.0 основная трудность при разработке web-приложений заключалась в невозможности взаимодействия клиента с сервером в промежутке между просмотрами двух страниц. Чтобы поменять содержимое страницы, приходилось перезагружать всю страницу целиком.

Решением вышеупомянутой проблемы стала технология AJAX[15], с распространением которой и появился термин Web 2.0. С помощью AJAX можно отправлять запросы на сервер, не перезагружая страницу целиком. Такая возможность позволяет разрабатывать более функционально развитые и удобные в использовании web-приложения. Основным языком программирования, используемым для создания AJAX приложений, является JavaScript. JavaScript – интерпретируемый язык программирования с поддержкой объектов, функций первого порядка, автоматического приведения типов, сборки мусора и анонимных функций.

11 REST (Representational State Transfer) – набор архитектурных принципов, в основе которых лежит работа с ресурсами
12 http://www.gmail.com
13 http://www.meebo.com
14 http://www.netvibes.com
15 AJAX (Asynchronous JavaScript and XML) – подход к построению интерактивных пользовательских интерфейсов Web-приложений

Постановка задачи

В связи с разработкой и переводом большинства приложений на web-платформу, появляется необходимость эффективного построения клиентских приложения для взаимодействия с нижележащими сервисами.

При создании клиентских приложений приходится учитывать значительные различия в реализации, которые допускаются при создании Web-служб. Это является результатом гибкости стека протоколов Web-служб. Ситуация усугубляется отсутствием альтернатив языку JavaScript при создании клиентских приложений, работающих внутри браузера, так как он не рассчитан на использование в сложных приложениях.

Одной из самых сложных частей клиентского приложения является непосредственно часть, отвечающая за взаимодействие с Web-службой. Поэтому ставиться задача создать генератор програмного кода, отвечающего за взаимодействие с Web-службами, для языка JavaScript.

Таким образом работа распадается на следующие шаги:

Анализ различных версий спецификации WSDL

Необходимо изучить две различные версии спецификаций языка описания Web-служб. Версия 1.1 является на сегодняшний день стандартом дефакто, а версия 2.0 должна получить статус стандарта в ближайшее время и будет широко использоваться в будущем. Поэтому, с точки зрения архитектуры приложения, важно выделить их общие места и различия.

Анализ существующих реализаций

На сегодняшний день существуют широко используемые библиотеки для языков Java, .NET, Ruby и некоторых других. Также есть несколько аналогичных библиотек для JavaScript, которые обладают сильно ограниченным набором возможностей и плохо подходят для использования в сложных клиентских приложениях. Важно проанализировать и учесть их ключевые возможности и недостатки при решении поставленной задачи.

Разработка архитектуры ядра приложения

Необходимо разработать архитектуру ядра приложения с учетом необходимости поддержки различных версий спецификации WSDL.

Разработка архитектуры генерируемого кода

Необходимо разработать архитектуру генерируемого кода на языке JavaScript, чтобы упростить адаптацию этой технологии в различных проектах, использующих в своей основе

разные библиотеки, такие как: Dojo[16], jQuery[17] или аналогичные.

Реализация ядра приложения

Необходимо выбрать язык программирования для реализации ядра приложения, парсера документов, написанных на языке описания Web-служб и реализовать его согласно разработанной архитектуре.

Реализация генерации кода

Необходимо реализовать генератор кода, в соответствии с разработанной архитектурой. При этом важно обеспечить кросс-браузерность генерируемого кода, в силу наличия отличий в реализации интерпретаторов JavaScript в различных браузерах.

Таким образом задача данной дипломной работы заключается в разработке генератора програмного кода, отвечающего за взаимодействие с Web-службами для языка JavaScript, с учетом существующих сложностей, таких как:

- обеспечение возможности поддержки различных версий спецификации языка описания Web-служб
- обеспечение легкой адаптации сгенерированного програмного кода в приложениях, основанных на разных библиотеках
- обеспечение кросс-браузерности сгенерированного программного кода

16 http://dojotoolkit.org/
17 http://jquery.com/

Анализ спецификаций WSDL

WSDL (Web Service Description Language) – это основанный на XML язык, который предоставляет модель для описания Web-служб. Документ WSDL предоставляет исчерпывающую информации о возможностях Web-службы. Благодаря наличию WSDL, разработчики различных частей системы имеют возможность создавать успешно взаимодействующие компоненты. WSDL является ключевым звеном стека Web-служб. Результатом поиска Web-службы в реестре UDDI является её описание на языке WSDL.

Необходимо проанализировать две различные версии спецификаций языка WSDL. Версия 1.1 является на сегодняшний день стандартом де-факто, а версия 2.0 должна получить статус стандарта в ближайшее время и будет широко использоваться в будущем. Поэтому, с точки зрения архитектуры приложения, важно выделить их общие места и различия.

Описание WSDL 1.1

WSDL документ определяет Web-службы (services) как набор портов (ports). В WSDL абстрактное определение портов и сообщений отделено от их конкретного, зависящего от сети и формата данных, определения. Это позволяет переиспользовать абстрактные определения. В WSDL документе используются следующие элементы для определения Web-служб:

- Definitions – элемент верхнего уровня, все остальные элементы являются его потомками.
- Types – контейнер для определения типов, с использованием какой-нибудь системы типов (обычно XSD[18]).
- Message – абстрактное, типизированное определение передаваемых данных.
- Operation – абстрактное определение услуги, предоставляемой Web-службой.
- Port Type – абстрактный набор операций, поддерживаемый одним или несколькими портами.
- Binding – конкретное определение протокола и формата данных для определенного port type.
- Port – одна конечная точка (порт), определенная как пара binding'а и сетевого адреса.
- Service – набор связанных конечных точек.

WSDL определяет стандартный механизм для описания binding'ов. В рамках спецификации WSDL 1.1 определены расширения для следующих протоколов и форматов сообщений:

18 XSD (XML Schema Definition)

- SOAP 1.1
- HTTP GET/POST
- MIME

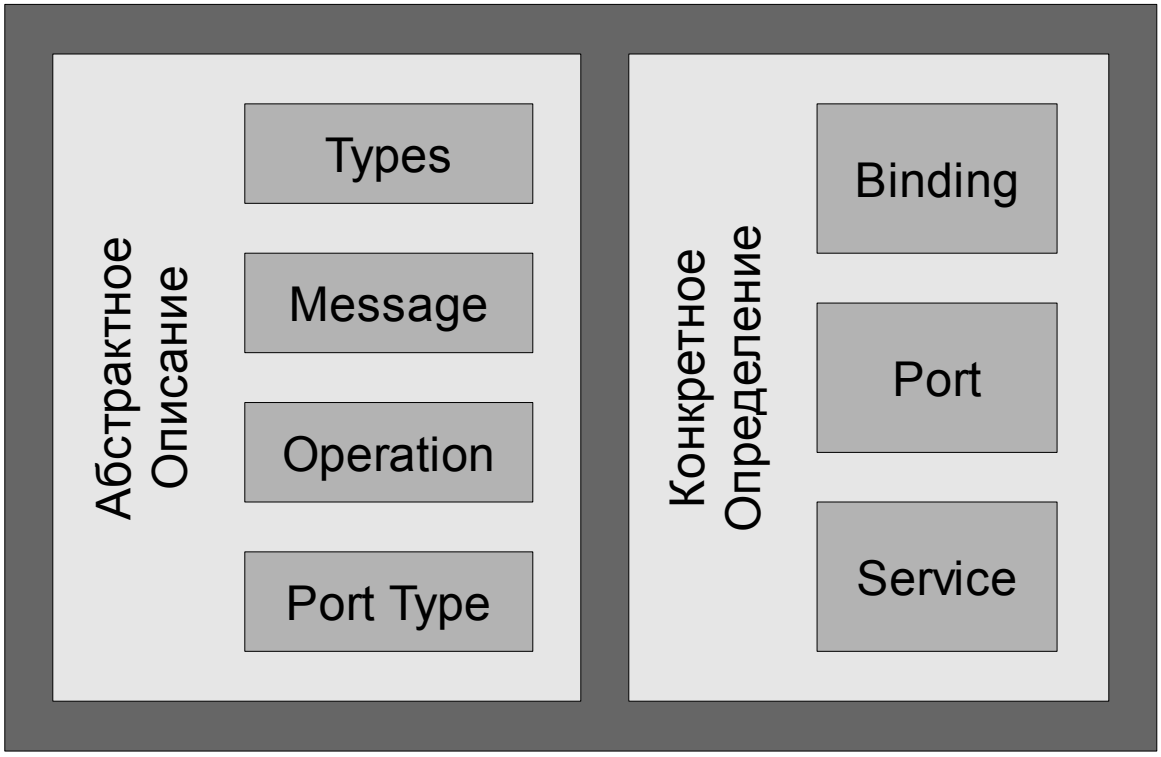

Диаграмма 1: Концептуальная модель WSDL 1.1

Таким образом, спецификация WSDL версии 1.1 позволяет удобно описывать Web-службы, разделяя их на абстрактную и конкретную части, что упрощает переиспользование, определяет общий механизм для связывания с различными протоколами и предоставляет готовые механизмы связывания для SOAP, HTTP GET/POST и MIME [8].

Пример стандартного документа WSDL 1.1:

```
<definitions name="Greeting" ...>
    <message name="GetNamedGreetingInput">
        <part name="name" element="xsd:string"/>
    </message>
    <message name="GetNamedGreetingOutput">
        <part name="result" type="xsd:string"/>
    </message>

    <portType name="NamedGreetingPortType">
        <operation name="GetNamedGreeting">
           <input message="tns:GetNamedGreetingInput"/>
           <output message="tns:GetNamedGreetingOutput"/>
        </operation>
```

```
    </portType>

    <binding name="NamedGreetingSoapBinding" type="tns:NamedGreetingPortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetNamedGreeting">
           <soap:operation soapAction="http://example.com/GetNamedGreeting"/>
           <input>
               <soap:body use="encoded" namespace="http://example.com/greeting"
                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
           </input>
           <output>
               <soap:body use="encoded" namespace="http://example.com/greeting"
                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
           </output>
        </operation>
    </binding>

    <service name="GreeterService">
        <port name="GreeterPort" binding="tns:NamedGreeterSoapBinding">
           <soap:address location="http://example.com/greeting"/>
        </port>
    </service>
</definitions>
```

К сожалению, коммерческое использование WSDL 1.1 получила только в связке с SOAP, так как расширение для протокола HTTP сильно ограничивает его возможности.

Отличия WSDL 2.0 от WSDL 1.1

Спецификация WSDL 1.1 была издана в 2001 году. Учитывая темпы развития интернет технологий, прошло много времени и версия 1.1 спецификации WSDL перестала отвечать всем требованиям современных Web-служб. В частности, при помощи WSDL 1.1 не возможно описать Web-службы использующие REST. Отставание должно было быть скомпенсировано в следующей версии, версии WSDL 1.2. Но в связи со значительными изменениями внесенными в спецификацию WSDL, было принято решение переименовать следующую версию в WSDL 2.0 [9].

Спецификация WSDL 2.0 во многом отличается от предыдущей версии. Рассмотрим лишь наиболее существенные, с точки зрения решения поставленной задачи, отличия новой версии спецификации от старой. В WSDL 2.0:

- появилась поддержка новых шаблонов обмена сообщениями (Message Exchange Patterns).
- была добавлена концепция интерфейсов (Interfaces), которая заменила Port Type. Основное отличие интерфейсов это поддержка наследования.
- была удалена конструкция сообщений. Теперь сообщения задаются в элементе types при помощи системы типов XML-схемы.
- элемент Port переименован в endpoint.

- улучшилась поддержка протокола HTTP, была добавлена поддержка методов PUT и DELETE. Что уже позволяет говорить о поддержке технологии REST версией 2.0 спецификации WSDL.
- упрощенно описание Web-служб использующих SOAP.

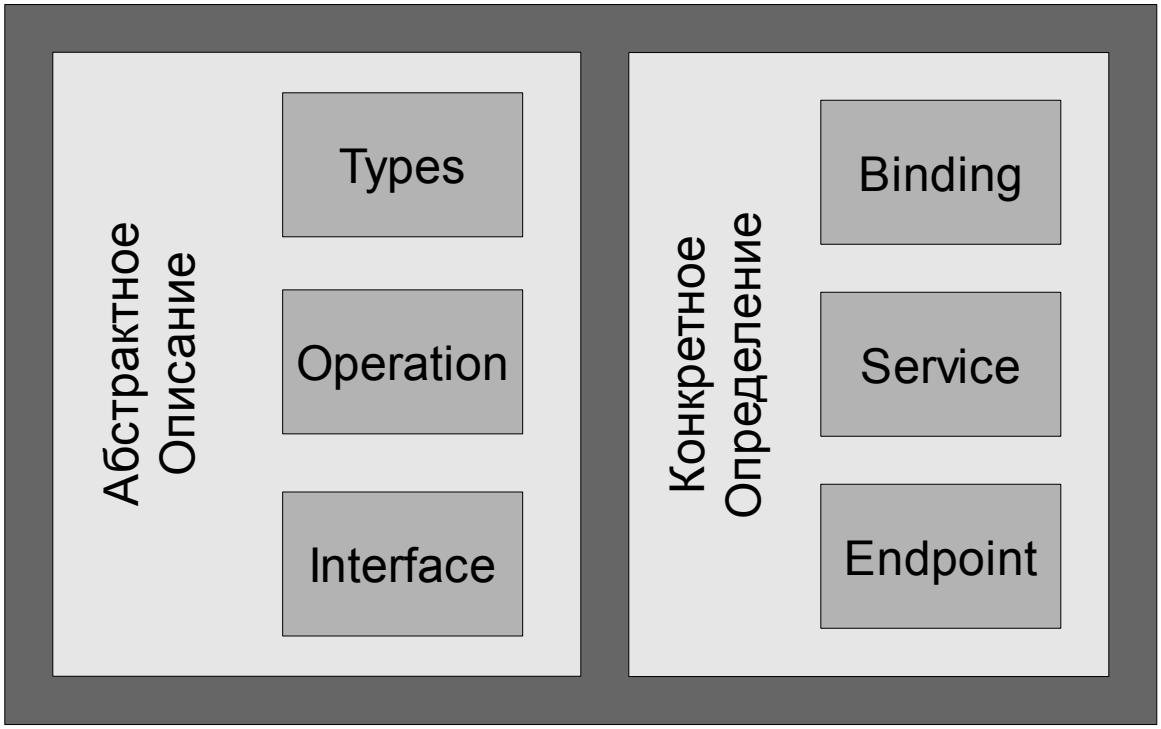

Диаграмма 2: Концептуальная модель WSDL 2.0

Пример стандартного документа WSDL 2.0:

```
<description name="Greeting" targetNamespace="http://example.com/greeter" ...>
  <types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.com/greeter"
        xmlns="http://example.com/greeter">
      <xs:element name="getNamedGreeting" type="xs:string"/>
      <xs:element name="getNamedGreetingResponse" type="xs:string"/>
    </xs:schema>
  </types>

  <interface name="greetingInterface">
    <operation name="getNamedGreeting" pattern="http://www.w3.org/ns/wsdl/in-out"
            style="http://www.w3.org/ns/wsdl/style/rpc">
        <input messageLabel="In" element="gns:getNamedGreeting"/>
        <output messageLabel="Out" element="gns:getNamedGreetingResponse"/>
    </operation>
  </interface>

  <binding name="greetingSoapBinding"
```

```
          interface="tns:greetingInterface"
          type="http://www.w3.org/ns/wsdl/soap"
          wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
    <operation ref="tns:getNamedGreeting"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
  </binding>

  <service name="greetingService" interface="tns:greetingInterface">
     <endpoint name="greetingEndpoint" binding="tns:greetingSoapBinding"
               address="http://example.com/greeting"/>
  </service>
</description>
```

Существующие инструментальные средства

В области генерации каркасов различных частей приложений, в системах с сервисно-ориентированной архитектурой, существует несколько успешных проектов, широко используемых в современных языках программирования. Кроме того, существует ряд попыток создать аналогичные инструментальные средства для языка программирования JavaScript. Анализ их главных достоинств и основных недостатков играет одну из ключевых ролей при успешном выполнении поставленной задачи.

Инструментальные средства для Java, .NET и Ruby

В этой главе рассматриваются три инструментальных средства для наиболее популярных языков интернет программирования. Их можно рассматривать как эталонные реализации. Набор их функций является стандартом, к которому привыкли большинство разработчиков. При решении поставленной задачи важно максимально функционально приблизиться к данным инструментальным средствам.

Apache Axis2 [10]

Apache Axis2 – это новая, переписанная с нуля, версия широко используемого SOAP стека Apache Axis. Существуют две версии для языков C и Java. Далее весь анализ проводился для Apache Axis2 для Java.

В состав Axis2 входит ряд полезных инструментальных средств, основной интерес для нас представляет инструмент для генерации программного кода по WSDL (WSDL2Code).

Архитектура

Разработчики очень хорошо позаботились об обеспечении гибкости всего инструментального средства и WSDL2Code в частности [11].

Гибкость в WSDL2Code заложена на двух уровнях:

1. Расширения (Extensions) могут быть использованы как для реализации простых функций вроде фильтрации неподходящих WSDL документов, так и для более сложных задач, таких как связывание данных. Но внесение изменений на этом уровне доступно только разработчикам.

2. Шаблоны (Templates) предоставляют более простой способ повлиять на результат работы генератора кода. Их можно использовать как для простых задач, вроде добавления комментариев, так и для намного более сложных. В частности, для генерации кода для другого языка программирования.

Ядро генератора кода запускает по порядку различные модули расширения, в том порядке в каком они указаны в конфигурационном файле. Затем передает управление эмиттеру. Эмиттер является важнейшей компонентой генератора кода, так как большая часть работы делается в нем.

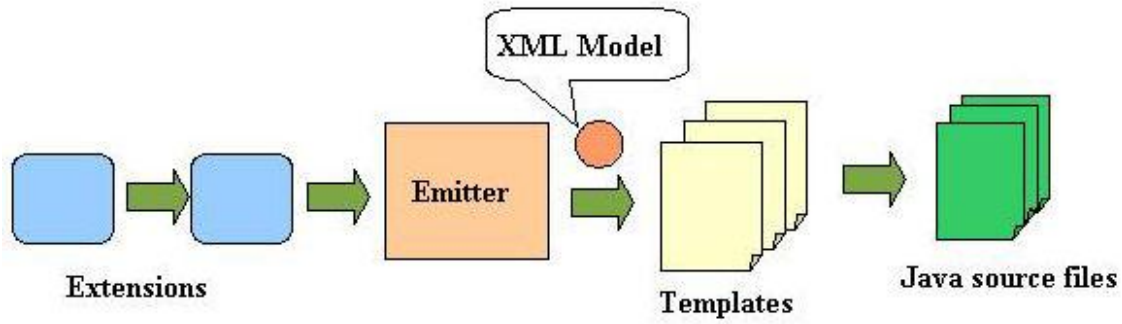

Диаграмма 3: Схема работы Apache Axis2

На выходе из эмиттера получается XML модель, которая далее преобразуется при помощи шаблонов. Шаблоны являются простыми XSL файлами и в результате их применения получается исходный код на языке программирования Java [12].

Основные достоинства

- Поддержка версий WSDL 1.1 и 2.0
- Поддержка REST
- Поддержка различных библиотек связывания данных (Axis Data Binding (ADB), XMLBeans, JibX, JaxMe, JaxBRI).

WSDL4R [13]

Стандартная библиотека языка Ruby для работы с WSDL. Поддерживает генерацию кода как для клиентской, так и для серверной части приложения.

Архитектура библиотеки ориентирована на простоту. Синтаксический анализатор базируется на XML парсере REXML, который имеет SAX-подобный интерфейс.

Основной недостаток – это отсутствие поддержки WSDL 2.0. Но учитывая тот факт, что язык Ruby на данный момент находится на этапе активно роста популярности, можно не сомневаться, что этот недостаток будет в ближайшем времени устранен.

Из плюсов стоит отметить поддержку сложных типов данных и хорошую структурированность сгенерированного программного кода.

.NET Framework WSDL Tool [14]

Стандартная компонента Microsoft .NET Framework для генерации кода по WSDL. Поддерживает генерацию кода как для клиентской, так и для серверной части приложения. Существует поддержка нескольких целевых языков, поддерживаемых платформой .NET.

Как и в двух предыдущих случаях, имеется поддержка сложных типов данных и сгенерированный код хорошо структурирован.

Основным недостатком данного инструментального средства является закрытость исходных кодов, что ограничивает возможности его анализа, а также лишает поддержки мирового сообщества разработчиков.

Заключение

Таким образом можно выделить следующие составляющие хорошего инструмента для генерации программных компонент по WSDL документу:

- генерация структурированного программного кода, удобного для восприятия и использования
- поддержка различных версий спецификации WSDL
  - обязательна хорошая поддержка версии 1.1, как стандарта дефакто
  - желательна либо поддержка спецификации версии 2.0, либо, в случае проектов с открытыми исходными кодами, простота и удобство её добавления
- поддержка типов данных
  - обязательна поддержка простых типов данных
  - обязательна поддержка простейших из сложных типов данных (массивы, списки)
  - желательна поддержка сложных типов данных

Инструментальные средства для языка JavaScript

wsdl2js (Python) [15]

Небольшая программа на языке программирования Python. Является побочным продуктом работы её автора над созданием расширения для браузера Mozilla Firefox, которое предоставляет возможность удобного использования различных сервисов компании Google, без необходимости переходить на соответствующие страницы.

В ней используется сильно упрощенный парсер WSDL документов, т. к. первоначально не подразумевалось никакого другого применения, кроме генерации кода для вышеуказанного расширения. Кроме этого, не проводилось никакого тестирования с другими Web-службами. Поэтому какое-либо использование этой программы не целесообразно. Из прочих недостатков можно отметить:

- жесткую привязку к объекту «SOAPCall» для пересылки данных, что ограничивает возможность использования сгенерированного кода строго браузерами от Mozilla
- плохую структурированность сгенерированного программного кода – он представляет из себя просто набор функций
- полное отсутствие документации

Единственным достоинством этого инструментального средства является наличие поддержки сложных типов данных.

wsdl2js (Java) [16]

Относительно зрелый программный продукт, написанный на языке программирования Java. Код хорошо документирован и имеется небольшое руководство пользователя.

К сожалению, реализована поддержка лишь небольшого числа простых типов данных. В исходящем запросе поддерживаются: строки, целые числа, числа с плавающей запятой и массивы этих трех типов. В ответе, кроме всех типов поддерживаемых в исходящем запросе, имеется поддержка объектов тип w3c DOM, которые обрабатываются в JavaScript коде как обычные DOM элементы.

Также, в сгенерированном коде нет разбиения на классы, просто набор функций.

The AJAX Engine [17, 18]

Это более широкий проект, который стремится упростить разработку AJAX приложений в связке с .NET и Java Web-службами. Но для нас из этого проекта представляют интерес лишь два файла. Один файл содержит утилитарный код на JavaScript, а второй это XSLT трансформация, при помощи которой из WSDL получается код на JavaScript.

Благодаря использованию XSLT, этот механизм легко можно использовать в связке почти с любым языком программирования, в частности JavaScript. Это дает возможность генерации и исполнения кода на лету.

Основные негативные следствия использования XSLT – это потери в скорости работы и сложность сопровождения и расширения функциональности.

Поддержка типов данных более полная чем в wsdl2js (Java), но все равно довольно скудная. Нет поддержки сложных типов.

JavaScript SOAP Client [19]

Единственная, полностью написанная на JavaScript, программа. Самое элегантное из имеющихся решений, всего 13 килобайт JavaScript кода. Из плюсов системы стоит отметить поддержку кеширования, что позволяет избежать повторной загрузки данных.

Имеется неплохая поддержка простых типов данных, есть поддержка массивов и классов реализующих интерфейс ICollection. Поддержка других сложных типов данных отсутствует.

Заключение

Основной причиной не всегда полной, а в случае с wsdl2js (Python) крайне скудной, поддержки WSDL, является попытка в каждом из инструментальных средств создавать собственный, новый подход для анализа WSDL документа, в то время как в большинстве современных языков программирования уже появились стандартные способы для работы с WSDL.

К ряду общих критериев оценки инструментов для генерации программных компонент по WSDL описанию, следует добавить следующие два критерия, специфичные для генерации кода на JavaScript:

- кросс-браузерность сгенерированного программного кода
- простота адаптации в различных проектах, основанных на различных JavaScript библиотеках

Инструментальное средство	**Структурированность кода**	**Поддержка WSDL**	**Поддерживаемые типы данных**	**Кросс-браузерность**	**Простота адаптации**
Wsdl2js (Python)	Средняя (просто набор функций)	Только плохая поддержка версии 1.1. Добавление поддержки новых версий не предусмотрено.	int, double, string, boolean, base64Binary, массивы и сложные типы без верификации	Только браузеры Mozilla	Нет
Wsdl2js (Java)	Средняя (просто набор функций)	Только версия 1.1. Добавление поддержки новых версий не предусмотрено.	int, float, string, массивы и XML объекты (только в ответе)	Да	Нет
The AJAX Engine	Плохая (код представляет из себя набор свойств)	Только версия 1.1. Добавление поддержки новых версий не предусмотрено.	int, string, double, boolean, dateTime, массивы и XML объекты	Да	Нет
JS SOAP Client	Не применимо	Только версия 1.1. Добавление поддержки новых версий не предусмотрено.	int, string, boolean, dateTime, массивы (включая ассоциативные) и классы реализующие интерфейс ICollection	Да	Нет

Таблица 1: Сравнение инструментальных средств для JavaScript

Анализ возможных подходов

Результаты предыдущей главы показали бесперспективность решений, использующих собственные механизмы для работы с WSDL документами.

В главе “Инструментальные средства для Java, .NET и Ruby” рассматривались три инструментальных средства для различных языков и даже целых семейств языков. Использование возможностей какого-либо из этих средств, значительно повысило бы качество предлагаемого решения. К сожалению инструментальное средство для .NET поставляется только в виде исполняемого файла и его исходные коды закрыты. Поэтому для решения поставленной задачи рассматривались только два подхода – один на основе Apache Axis2, а второй на основе WSDL4R.

Обзор решения на основе Apache Axis2

Для решения задачи с использованием Apache Axis2 требовалось разработать компоненту, содержащую множество шаблонов на языке трансформаций XSLT.

Основным преимуществом данного подхода является наличие поддержки WSDL 2.0.

К недостаткам можно отнести общую тяжеловесность Apache Axis2, что безусловно оправданно в случае использования всех его возможностей в рамках языка программирования Java, но является большим недостатком при использовании его лишь для генерирования программного кода для клиентских приложений на языке JavaScript. Так же ряд неудобств связан с использованием XSLT. Этот язык базируется на XML и плохо пригоден для редактирования человеком.

Обзор решения на основе WSDL4R

В случае использования WSDL4R, приложение получает надежную и легковесную основу для анализа WSDL документов.

Использование WSDL4R означает решение поставленной задачи на языке Ruby, что позволяет воспользоваться всеми преимуществами этого современного языка программирования. В частности, для генерации программного кода, вместо сложных и тяжеловесных XSLT трансформаций, использующихся в Apache Axis2, есть возможность использовать ERB – простой и легковесный механизм шаблонов, встроенный в язык Ruby.

Очевидным недостатком является отсутствие поддержки WSDL 2.0. Но учитывая тот факт, что язык Ruby на данный момент находится на этапе активно роста популярности, можно не сомневаться, что этот недостаток будет в ближайшем времени устранен.

Предлагаемое решение

Основываясь на анализе преимуществ и недостатков подходов, описанных в предыдущей главе, было принято решение использовать подход на основе WSDL4R. Так как он обеспечивает большую степень свободы, по сравнению с подходом на основе Apache Axis2. Потому как использование Apache Axis2 сводит решение задачи к созданию двух громоздких компонент, а решение на основе WSDL4R требует в меньшей степени программирования и в большей степени проектирования.

Архитектура приложения

Предлагаемое решение было реализовано на языке программирования Ruby.

В работе приложения можно выделить три основных этапа:

1. Разбор WSDL документа синтаксическим анализатором.
2. Оборачивание объектного представления WSDL документа конкретной версии в обобщенное объектное представление.
3. Генерация программного кода, с использованием шаблонов ERB.

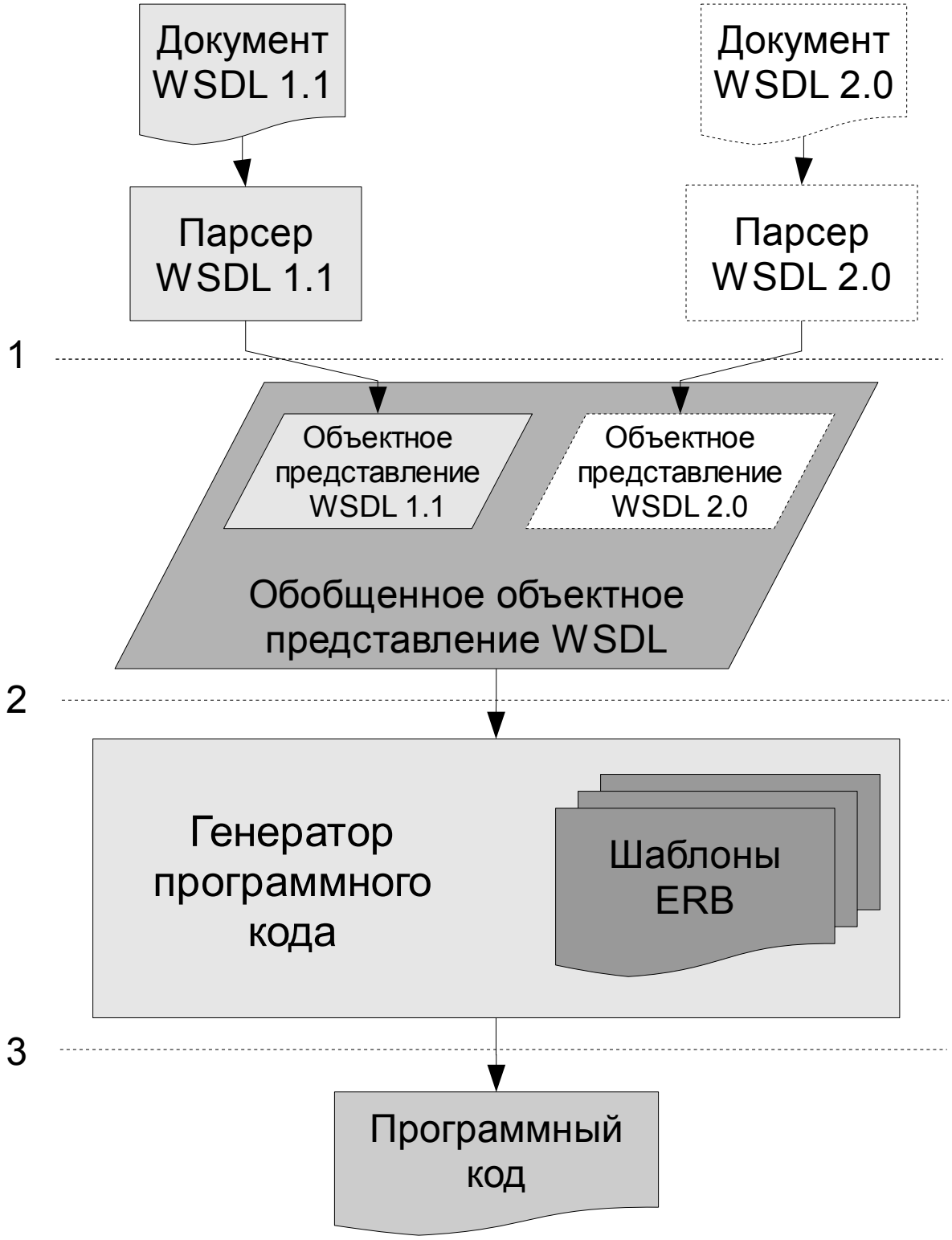

Диаграмма 4: Архитектура приложения

Синтаксический анализ WSDL документа

Первой стадией работы приложения является синтаксический анализ WSDL документа. При этом происходит преобразование текстового представления WSDL документа в объектное.

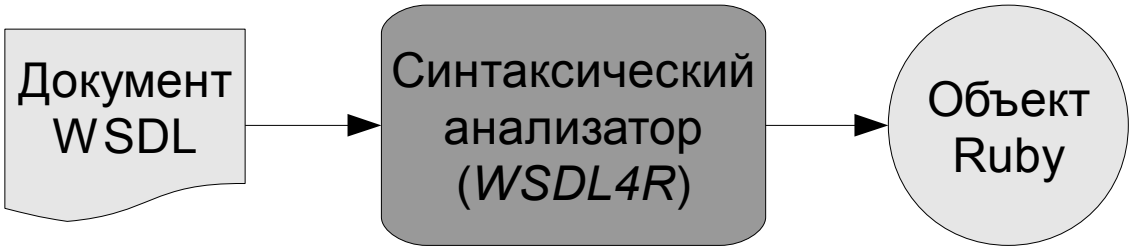

Диаграмма 5: Синтаксический анализ

Для синтаксического анализа WSDL документов в языке Ruby используется парсер WSDL4R. На стадии лексического анализа в парсере WSDL4R используется XML парсер REXML, с SAX-подобным интерфейсом.

REXML, как и сам WSDL4R входит в число стандартных библиотек, поставляемых вместе с языком программирования Ruby.

В результате работы синтаксического анализатора получается объектное представление WSDL документа.

Создание обобщенного объектного представления

Результаты синтаксического анализа WSDL документа необходимо обернуть в некоторое обобщенное представление, которое позволило бы генератору программного кода единым образом взаимодействовать с различными объектными представлениями WSDL документа.

Для этих целей была разработана простая иерархия интерфейсов, включающая в себя такие интерфейсы как Class, Method и прочие. Эта иерархия интерфейсов в первую очередь адаптированна для удобства кодогенерации, в то время как объектное представление WSDL документа, полученное как результат синтаксического анализа, лишь отражает содержимое WSDL документа в терминах объектов языка Ruby.

Для различных объектных представлений документа WSDL (это могут быть как объектные представления документов, соответствующих различным версиям спецификации WSDL, так и различные объектные представления, создаваемые различными парсерами) нужно лишь реализовать указанную выше иерархию классов.

Генерация программного кода

Генератор программного кода архитектурно разделен на две части:

- Логика, связанная с генерацией программного кода на языке JavaScript, содержится в ряде классов и модулей на языке программирования Ruby.
- Все детали, относящиеся к внешнему виду генерируемого программного кода, сосредоточены в ряде шаблонов ERB.

Использование шаблонов ERB позволяет упростить внесение изменений в генерируемый код и предоставляет удобный способ редактирования внешнего вида генерируемого кода.

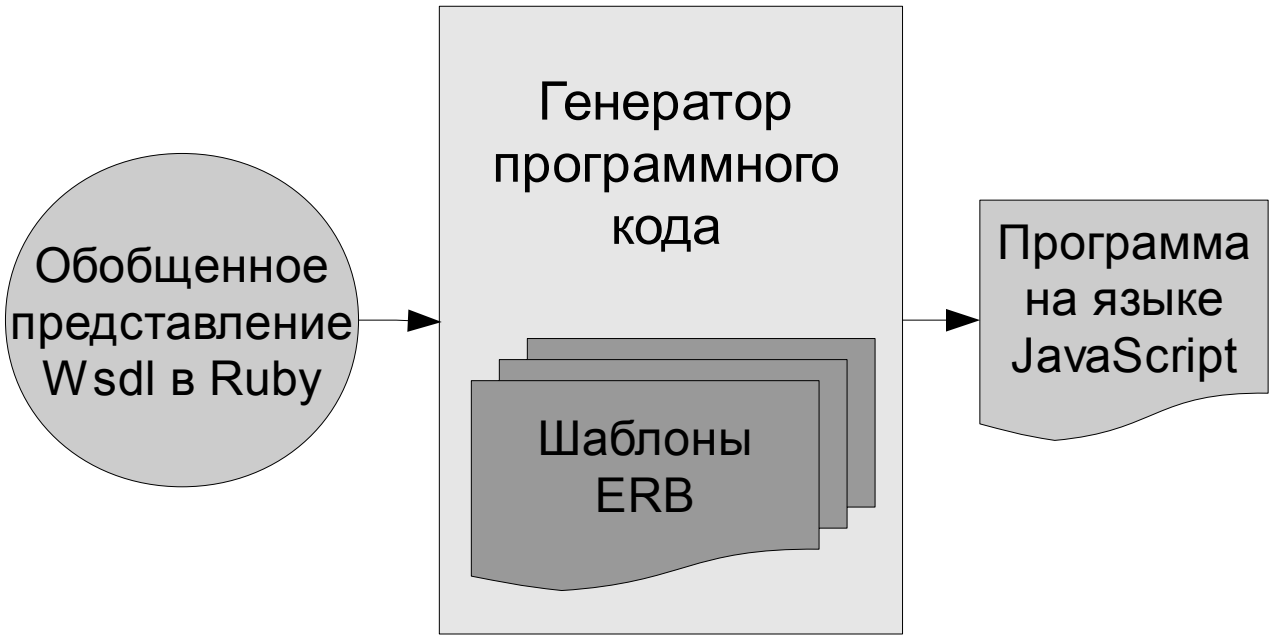

Диаграмма 6: Генерация программного кода

Описание работы генератора

На вход генератору поступает обобщенное представление WSDL, в виде объектов языка Ruby.

Для каждой Web-службы описанный в WSDL документе, генерируется один класс.

Для генерации класса используется шаблон class.erb, который имеет следующий вид:

```
//
// Start of <%= class_name %> class
//
```

```
function <%= class_name %> ()
{
          // Port location
          this.location = "<%= location %>";
}

<%= class_name %>.prototype = new AbstractProxy();
<%= render_methods(class) %>
//
// End of <%= class_name %> class
//
```

Между `<%=` и `%>` заключены переменные, которые определяются в коде на языке Ruby. Исключением является лишь `<%= render_methods(class) %>`, в данном случае производится вызов функции, отвечающей за генерацию всех методов данного класса.

В процессе обработки шаблона, значения переменных вычисляются в генераторе и подставляются в соответствующие места в шаблоне. Также происходит вызов метода `render_methods`.

После вызова метода начинается генерация методов класса. При этом используется шаблон method.erb. При этом последовательность действий такая же, как и при генерации класса.

Аналогичным образом генерируются документация, различные части тел методов, а также классы, соответствующие типам данных, которые определенны в WSDL документе.

Архитектура генерируемого кода

Полученный в результате работы приложения программный код на языке JavaScript состоит из трех основных частей:

- Вспомогательный код, содержащий различные общие классы и методы. Он не зависит от конкретной Web-служб или ее описания.
- Код сгенерированный по конкретному описанию Web-службы.
- Серверное прокси.

Стоит также отметить, что непосредственно кроме программного кода на языке JavaScript генерируется еще и документация, в виде JSDoc. Она включает в себя как информацию о типах данных (это упрощает использование кода, так язык JavaScript относится к динамически типизируемым и программный код не содержит подобной информации), так и комментарии содержащиеся в исходном WSDL документе.

Вспомогательный код

В основе вспомогательного кода лежит интерфейс Base. Он описывает все внешние

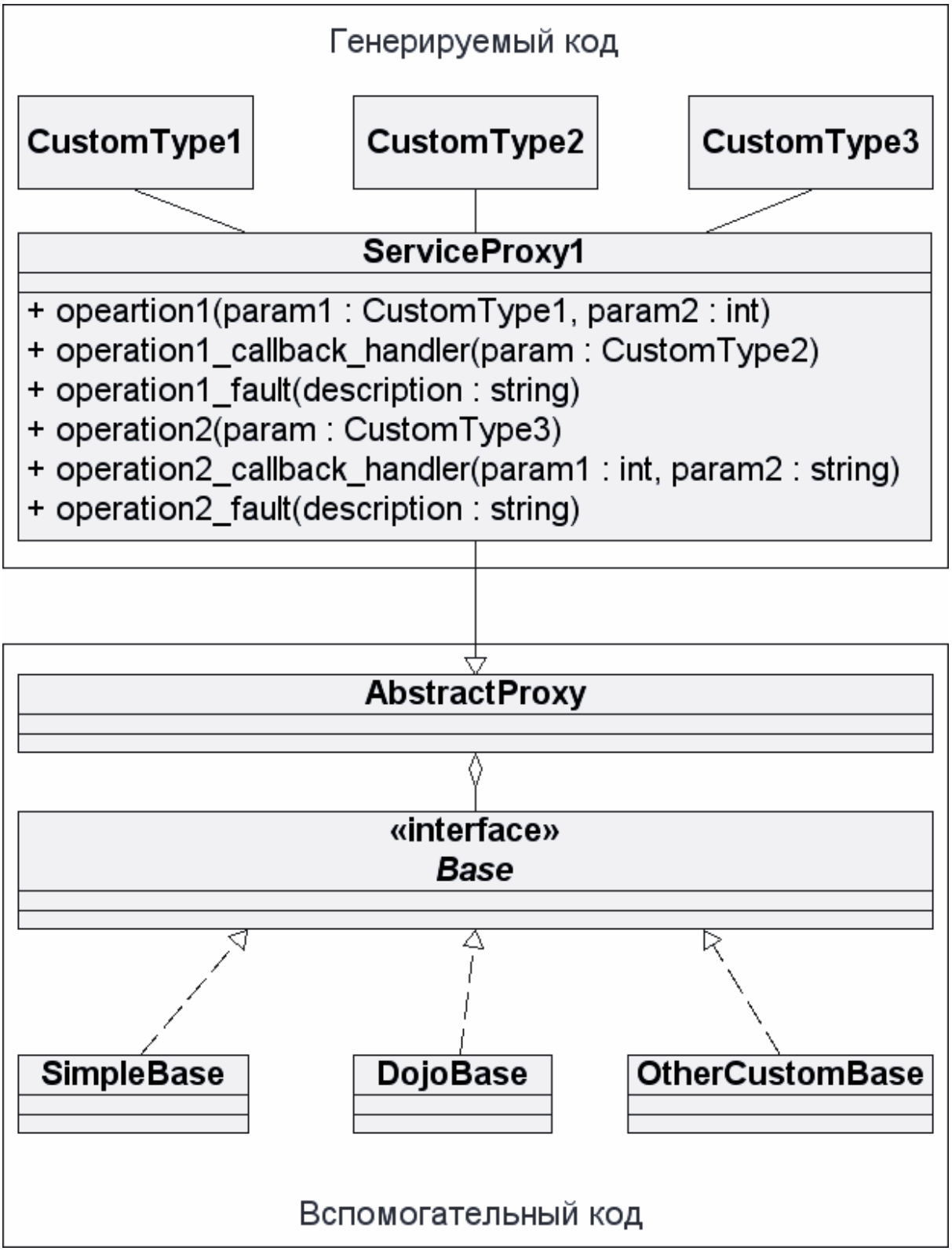

Диаграмма 7: Взаимосвязь классов сгенерированного кода

операции, используемые генерируемым кодом. Такие как: работа с XML, отправка и прием сообщений и т.п..

На данный момент существует единственная реализация этого интерфейса, которая основывается на библиотеке Sarissa [20]. Это одна из самых простых библиотек для создания кроссбразуерных решений на языке JavaScript.

Различные реализации интерфейса Base позволят лучше использовать возможности более богатых и функционально развитых библиотек, например таких как Dojo или jQuery.

Абстрактный класс AbstractProxy содержит имплементацию общих методов, используемых в его потомках, для работы с Web-службами.

Генерируемый код

Для каждого WSDL документа генерируется один текстовый файл с программным кодом.

Генерируемый код включает в себя два основных блока:

- Описание типов данных определенных в WSDL документе в терминах языка JavaScript.
- Прокси класс, маскирующий работу с самой Web-службой.

Типы данных

Для каждого типа, описанного в WSDL документе, создается JavaScript класс, который соответствует ему.

При этом каждый класс поддерживает сериализацию объекта JavaScript в тот формат исходящих сообщений, который описан в WSDL. А также каждый класс имеет механизмы создания объектов JavaScript из данных, получаемых во входном сообщении.

Прокси класс

Для каждой Web-службы, описанной в WSDL документе, генерируется один прокси класс. Этот класс наследуется от класса AbstarctProxy, который является частью вспомогательного кода.

Затем для каждой операции данной Web-службы генерируются три метода:

- Метод, который непосредственно используется разработчиком для того, чтобы послать запрос Web-службе. Он называется также, как и сама операция. В качестве параметров метода передаются те объекты, которые в дальнейшем должны быть сериализованы и переданы Web-службе в запросе.

- Метод, который вызывается, когда полученный от Web-службы ответ уже преобразован в объекты JavaScript. Тело данного метода автоматически не заполняется. Разработчик должен в теле этого метода написать свой код для обработки вызова. В качестве параметров метода передаются объекты JavaScript, содержащие заранее обработанный ответ Web-службы. Название состоит из имени операции и суффикса CallbackHandler.

- Метод, который вызывается если происходит какой-либо сбой при общении с Web-службой, либо если она сама возвращает ошибку. Разработчик должен написать свой код, обрабатывающий ошибку в теле этого метода. В качестве параметра этого метода передается текстовое сообщение, описывающее ошибку. Название состоит из имени операции и суффикса ErrorHandler.

Серверное прокси

В тех случаях, когда клиентское приложение взаимодействует с Web-службой из другого домена, необходимо использовать серверное прокси, находящееся в том же домене, что и само клиентское приложение.

Серверное прокси необходимо из-за ограничений наложенных интернет обозревателями из соображений безопасности. Единственная его задача – передать по указанному адресу, полученное сообщение, а затем вернуть ответ.

Оно может быть реализовано с использованием любых технологий. Единственное требование выдвигаемое к серверному прокси заключается в том, чтоб все полученные сообщения должны передаваться по указанному адресу, а ответы возвращаться на клиентское приложение.

Если при генерации устанавливается соответствующий флаг, то программный код на языке JavaScript генерируется с учетом наличия серверного прокси.

Пример использования

В данной главе рассмотрим немного упрощенный пример использования данного решения в контексте всего стека Web-служб (см. диаграмма 8):

1. Клиент отсылает запрос на UDDI реестр. Ему нужна Web-служба для поиска в Интернете.

2. UDDI реестр возвращает описание Web-службы на языке WSDL, и оно передается в генератор JavaScript.

3. На выходе получается код на языке JavaScript, который помещается в файл Google.js.

4. Разработчик дописывает несколько строк для выполнения вызова:

   - `var gp = new GoogleProxy();` – код, создающий новый объект для работы с Web-службой

   - `gp.search(page, query, parm);` – код, вызывающий метода search, в качестве параметров передаются простые JavaScript объекты.

5. Затем разработчик заполняет тело метода, обрабатывающего результаты вызова:

   - `alert(sr.toString());` – код, выводящий результаты.

6. Когда программа попадает в браузер, происходит вызов метода search у объекта gp.

7. При этом переданные JavaScript объекты соответствующим образом преобразуются, формируется SOAP сообщение и передается на локальное прокси. Оно необходимо из-за ограничений безопасности налагаемых браузером. И его единственное назначение – передача полученного сообщения.

8. Серверное прокси передает полученное сообщение Web-службе Google.

9. Получив запрос, Web-служба Google формирует SOAP сообщение с результатом поиска и передает его обратно на серверное прокси.

10. Серверное прокси передает полученное сообщение на интернет обозреватель клиента, где результаты из SOAP сообщения превращаются в JavaScript объекты и передаются в качестве параметров функции обработчика.

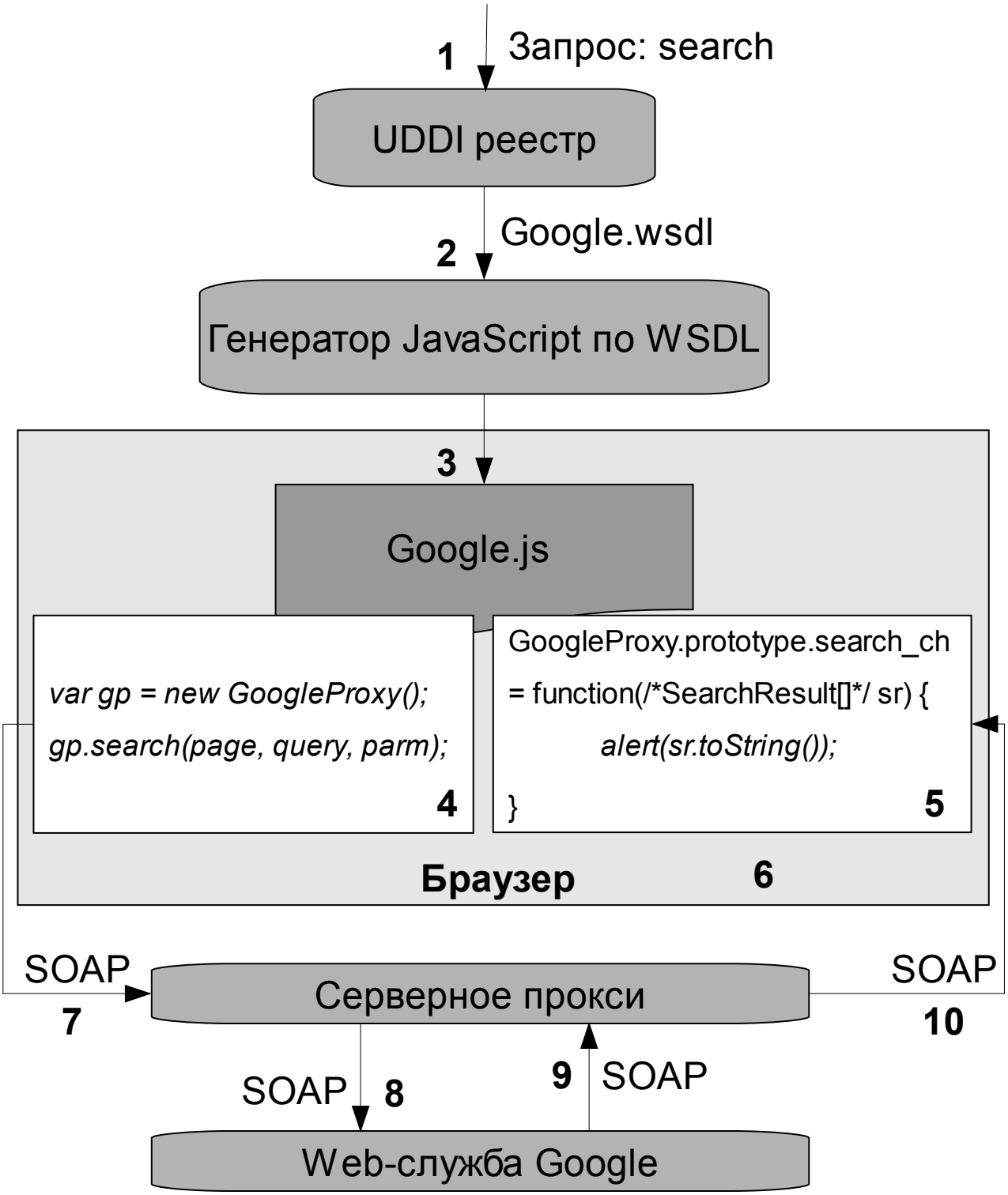

Диаграмма 8: Пример использования

Заключение

В рамках данной работы были получены следующие результаты:

- Проведен анализ различных версий спецификации языка описания Web-служб WSDL и составлен общий набор свойств необходимых для кодогенерации на языке JavaScript.
- Проведен анализ наиболее удачных инструментальных средства для кодогенерации по WSDL для Java, .NET и Ruby.
- Проведен анализ недостатков существующих инструментальных средств для языка JavaScript
- Разработана и реализована гибкая архитектура приложения, позволяющая максимально упростить добавление поддержки различных версий спецификации WSDL, или любого схожего языка описания Web-служб.
- Разработана и реализована архитектура генерируемого кода, позволяющая применять данное решение в связке с различными базовыми библиотеками для языка JavaScript.

Итогом всей работы является разработка генератор программного кода, отвечающего за взаимодействие с Web-службами, для языка JavaScript, который по своим характеристикам не уступает аналогичным средствам для Java, .NET или Ruby.

На данный момент имеется поддержка WSDL только версии 1.1, но благодаря успешной работе по выделению общих свойств существующих спецификаций языка WSDL, которые необходимы при генерации клиентских приложений, добавление поддержки других версий максимально упрощено.

Список литературы

1. Спецификация XML, http://www.w3.org/TR/xml
2. Спецификация SOAP, http://www.w3.org/TR/soap
3. Спецификация WSDL версия 1.1, http://www.w3.org/TR/wsdl
4. Спецификация WSDL версия 2.0, http://www.w3.org/TR/wsdl20
5. Спецификация UDDI версия 3, http://uddi.org/pubs/uddi_v3.htm
6. Web Services Activity, http://www.w3.org/2002/ws
7. Определение сервисно-ориентированной архитектуры, http://opengroup.org/projects/soa/doc.tpl?gdid=10632
8. Хабибулин И., “Разработка Web-служб средствами Java”. СПб, “БХВ-Петербург”, 2003. 183-211 с.
9. Arulazi Dhesiaseelan, “What's new in WSDL 2.0”, http://www.xml.com/pub/a/ws/2004/05/19/wsdl2.html
10. Apache Axis2, http://ws.apache.org/axis2
11. S. W. Eran Chinthaka, “Axis2: The Next Generation of Apache Web Services”, http://today.java.net/pub/a/today/2006/09/07/axis2-next-generation-web-services.html
12. Ajith Ranabahu, “Inside the Axis2 Code Generator”, http://wso2.org/node/35
13. WSDL4R, http://dev.ctor.org/soap4r
14. .NET Web Service Description Language Tool, http://msdn2.microsoft.com/en-us/library/7h3ystb6(VS.80).aspx
15. wsdl2js (Python), http://www.bolinfest.com/wsdl2js
16. wsdl2js (Java), http://sourceforge.net/projects/wsdl2js
17. The AJAX Engine, http://www.mathertel.de/AJAXEngine
18. A proxy generator to WebServices for JavaScript and AJAX http://www.codeproject.com/soap/JavaScriptProxy_01.asp
19. JavaScript SOAP Client, http://www.codeplex.com/JavaScriptSoapClient
20. JavaScript библиотека Sarissa, http://dev.abiss.gr/sarissa/

Приложения

Приложение 1. Спецификация WSDL 1.1

Definitions

Элемент definitions является корневым элементом WSDL документа. Все остальные элементы являются его потомками.

```
<definitions name="имя"? targetNamespace="uri"?>
    ...
</definitions>
```

Types

В элементе types описываются все типы данных, имеющие отношение к используемым сообщениям. Для обеспечения максимальной платформонезависимости, в WSDL обычно используется XML Schema.

```
<definitions .... >
    <types>
        <xsd:schema .... />*
    </types>
</definitions>
```

Messages

Элемент messages состоит из одной или нескольких логических частей, заключенных в элементы part.

```
<definitions .... >
    <message name="имя">
        <part name="имя1" element="ссылка на тип описанный в types"/>
        <part name="имя2" element="ссылка на тип описанный в types"/>
    </message>
</definitions>
```

Элементы part являются гибким механизмом для абстрактного описания частей передаваемых сообщений. В дальнейшем элементе binding может, используя абстрактное описание из элемента part, задать конкретный формат сообщения, для конкретно используемых протоколов.

Port Types

Элемент portType описывает именованный набор абстрактных операций вместе с используемыми абстрактными сообщениями (messages). В рамках одного WSDL документа, имя элемента portType является уникальным.

```
<definitions .... >
    <portType name="имя">
        <operation name="имя.." .... /> *
    </portType>
</definitions>
```

Bindings

Элемент binding определяет формат сообщения и используемый протокол для операций и сообщений заданных в определенном элементе portType. Конкретный элемент portType может быть связан с произвольным количеством элементов binding.

Имя элемента binding является уникальным среди всех элементов binding в рамках одного WSDL документа.

```
<wsdl:definitions .... >
    <wsdl:binding name="имя" type="имя-portType"> *
        <-- элемент расширения --> *
        <wsdl:operation name="имя"> *
           <-- элемент расширения --> *
           <wsdl:input name="имя"? > ?
               <-- элемент расширения -->
           </wsdl:input>
           <wsdl:output name="имя"? > ?
               <-- элемент расширения --> *
           </wsdl:output>
           <wsdl:fault name="имя"> *
               <-- элемент расширения --> *
           </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
</wsdl:definitions>
```

Ports

Элемент port определяет конкретную конечную точку путем указания её адреса. И связывает её с некоторым элементом binding.

```
<wsdl:definitions .... >
    <wsdl:service .... > *
        <wsdl:port name="имя" binding="имя-bindig'a"> *
           <-- элемент расширения -->
        </wsdl:port>
```

```
    </wsdl:service>
</wsdl:definitions>
```

Services

Элемент service позволяет сгруппировать несколько связанных конечных точек вместе:

```
<definitions .... >
    <service name="имя"> *
        <port .... /> *
    </service>
</definitions>
```

Расширение для протокола SOAP

Для связи с протоколом SOAP 1.1 используются следующие элементы:

- soap:binding – указывает на использование протокола SOAP.
- soap:operation – описывает детали реализации конкретной услуги, предоставляемой Web-службой.
- soap:body – описывает как различные части сообщения отображаются в элементе Body SOAP сообщения.
- soap:fault – описывает содержимое элемента Fault Details в SOAP сообщении.
- soap:header и soap:headerfault – аналоги soap:body и soap:fault для элемента Header в SOAP сообщении.
- soap:address – связывает конкретную конечную точку с а адресом (URI).

Расширение для протокола HTTP

Для связи с протоколом HTTP 1.1 используются следующие элементы:

- http:binding – указывает на использование протокола HTTP.
- http:address – указывает базовый адрес для конечной точки (port).
- http:operation – указывает относительный адрес для операции.
- http:urlEncoded – указывает, что все части (parts) сообщения подставляются в URI запроса стандартным образом (имя1=значение1&имя2=занчение2...)
- http:urlReplacement – указывает, что все части (parts) сообщения подставляются в URI

запроса при помощи особого алгоритма подстановки.

Поддерживаются только методы GET и POST.

Приложение 2. Спецификация WSDL 2.0

Description

Компонента Description является контейнером для двух видов компонент: компонент WSDL 2.0 и компонент системы типов.

Interface

Компонента Interface описывает последовательность сообщений посылаемых и/или принимаемых Web-службой, путем группировки связанных сообщений в операции. Операция – это последовательность входящих и исходящих сообщений, а интерфейс – это набор операций.

Интерфейс может расширять один или несколько других интерфейсов.

```
<description>
  <interface name="xs:NCName" extends="list of xs:QName"?
        styleDefault="list of xs:anyURI"? >
    <documentation />*
    [ <fault /> | <operation /> ]*
  </interface>
</description>
```

Binding

Компонента Binding описывает конкретный формат сообщения и протокол передачи, которые могут использоваться для определения конечной точки. Таким образом компонента Binding определяет детали реализации необходимые для использования службы.

```
<description>
  <binding name="xs:NCName" interface="xs:QName"? type="xs:anyURI" >
    <documentation />*
    [ <fault /> | <operation /> ]*
  </binding>
</description>
```

Service

Компонента Service описывает набор конечных точек, в которых конкретная развернутая

реализация Web-службы доступна.

```
<description>
  <service name="xs:NCName" interface="xs:QName" >
    <documentation />*
    <endpoint />+
  </service>
</description>
```

Endpoint

Определяет детали конкретной конечной точки, где предоставляется данная услуга.

```
<description>
  <service>
    <endpoint name="xs:NCName" binding="xs:QName" address="xs:anyURI"? >
      <documentation />*
    </endpoint>+
  </service>
</description>
```

Расширение для протокола SOAP

```
<description>
  <binding name="xs:NCName" interface="xs:QName"?
           type="http://www.w3.org/ns/wsdl/soap"
           whttp:queryParameterSeparatorDefault="xs:string"??
           whttp:contentEncodingDefault="xs:string"??
           whttp:cookies="xs:boolean"?
           wsoap:version="xs:string"?
           wsoap:protocol="xs:anyURI"
           wsoap:mepDefault="xs:anyURI"? >
    <documentation />*

    <wsoap:module ref="xs:anyURI" required="xs:boolean"? >
      <documentation />*
    </wsoap:module>*

    <fault ref="xs:QName"
           wsoap:code="union of xs:QName, xs:token"?
           wsoap:subcodes="union of (list of xs:QName), xs:token"?
           whttp:contentEncoding="xs:string"?? >

      <documentation />*

      <wsoap:module ... />*
      <wsoap:header element="xs:QName" mustUnderstand="xs:boolean"?
                    required="xs:boolean"? >
        <documentation />*
      </wsoap:header>*
      <whttp:header ... />*??
```

```
    </fault>*

    <operation ref="xs:QName"
               whttp:location="xs:anyURI"??
               whttp:contentEncodingDefault="xs:string"??
               whttp:queryParameterSeparator="xs:string"??
               whttp:ignoreUncited="xs:boolean"??
               wsoap:mep="xs:anyURI"?
               wsoap:action="xs:anyURI"? >

      <documentation />*

      <wsoap:module ... />*

      <input messageLabel="xs:NCName"?
             whttp:contentEncoding="xs:string"?? >
        <documentation />*
        <wsoap:module ... />*
        <wsoap:header ... />*
        <whttp:header ... />*??
      </input>*

      <output messageLabel="xs:NCName"?
              whttp:contentEncoding="xs:string"?? >
        <documentation />*
        <wsoap:module ... />*
        <wsoap:header ... />*
        <whttp:header ... />*??
      </output>*

      <infault ref="xs:QName"
                 messageLabel="xs:NCName"?>
        <documentation />*
        <wsoap:module ... />*
      </infault>*

      <outfault ref="xs:QName"
                  messageLabel="xs:NCName"?>
        <documentation />*
        <wsoap:module ... />*
      </outfault>*

    </operation>*

  </binding>

  <service>
    <endpoint name="xs:NCName" binding="xs:QName" address="xs:anyURI"?
              whttp:authenticationScheme="xs:token"??
              whttp:authenticationRealm="xs:string"?? >
      <documentation />*
    </endpoint>
  </service>
</description>
```

Расширение для протокола HTTP

```
<description>
  <binding name="xs:NCName" interface="xs:QName"?
           type="http://www.w3.org/ns/wsdl/http"
           whttp:methodDefault="xs:string"?
           whttp:queryParameterSeparatorDefault="xs:string"?
           whttp:cookies="xs:boolean"?
           whttp:contentEncodingDefault="xs:string"? >
   <documentation />?

    <fault ref="xs:QName"
           whttp:code="union of xs:int, xs:token"?
           whttp:contentEncoding="xs:string"? >
      <documentation />*
      <whttp:header name="xs:string" type="xs:QName"
                    required="xs:boolean"? >
        <documentation />*
      </whttp:header>*
    </fault>*

    <operation ref="xs:QName"
               whttp:location="xs:anyURI"?
               whttp:method="xs:string"?
               whttp:inputSerialization="xs:string"?
               whttp:outputSerialization="xs:string"?
               whttp:faultSerialization="xs:string"?
               whttp:queryParameterSeparator="xs:string"?
               whttp:contentEncodingDefault="xs:string"?
               whttp:ignoreUncited="xs:boolean"? >
          <documentation />*

      <input messageLabel="xs:NCName"?
             whttp:contentEncoding="xs:string"? >
        <documentation />*
        <whttp:header ... />*
      </input>*

      <output messageLabel="xs:NCName"?
              whttp:contentEncoding="xs:string"? >
        <documentation />*
        <whttp:header ... />*
      </output>*

      <infault ref="xs:QName"
                messageLabel="xs:NCName"? >
        <documentation />*
      </infault>*

      <outfault ref="xs:QName"
                messageLabel="xs:NCName"? >
        <documentation />*
      </outfault>*

    </operation>*
```

```
  </binding>

  <service>
    <endpoint name="xs:NCName" binding="xs:QName" address="xs:anyURI"?
              whttp:authenticationScheme="xs:token"?
              whttp:authenticationRealm="xs:string"? >
      <documentation />*
    </endpoint>
  </service>
</description>
```

Приложение 3. Пример WSDL документа

WSDL документ, описывающий Web-службу компании Google:

```
<?xml version="1.0"?>
<definitions name="GoogleSearch" targetNamespace="urn:GoogleSearch"
    xmlns:typens="urn:GoogleSearch"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  <!-- Types for search - result elements, directory categories -->
  <types>
    <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:GoogleSearch">
      <xsd:complexType name="GoogleSearchResult">
        <xsd:all>
          <xsd:element name="documentFiltering" type="xsd:boolean"/>
          <xsd:element name="searchComments" type="xsd:string"/>
          <xsd:element name="estimatedTotalResultsCount" type="xsd:int"/>
          <xsd:element name="estimateIsExact" type="xsd:boolean"/>
          <xsd:element name="resultElements" type="typens:ResultElementArray"/>
          <xsd:element name="searchQuery" type="xsd:string"/>
          <xsd:element name="startIndex" type="xsd:int"/>
          <xsd:element name="endIndex" type="xsd:int"/>
          <xsd:element name="searchTips" type="xsd:string"/>
          <xsd:element name="directoryCategories"
            type="typens:DirectoryCategoryArray"/>
          <xsd:element name="searchTime" type="xsd:double"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:schema>
  </types>
  <message name="doGoogleSearch">
    <part name="key" type="xsd:string"/>
    <part name="q" type="xsd:string"/>
    <part name="start" type="xsd:int"/>
    <part name="maxResults" type="xsd:int"/>
    <part name="filter" type="xsd:boolean"/>
    <part name="restrict" type="xsd:string"/>
```

```
    <part name="safeSearch" type="xsd:boolean"/>
    <part name="lr" type="xsd:string"/>
    <part name="ie" type="xsd:string"/>
    <part name="oe" type="xsd:string"/>
  </message>
  <message name="doGoogleSearchResponse">
    <part name="return" type="typens:GoogleSearchResult"/>
  </message>
  <!-- Port for Google Web APIs, "GoogleSearch" -->
  <portType name="GoogleSearchPort">

    <operation name="doGoogleSearch">
      <input message="typens:doGoogleSearch"/>
      <output message="typens:doGoogleSearchResponse"/>
    </operation>
  </portType>
  <!-- Binding for Google Web APIs - RPC, SOAP over HTTP -->
  <binding name="GoogleSearchBinding" type="typens:GoogleSearchPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="doGoogleSearch">
      <soap:operation soapAction="urn:GoogleSearchAction"/>
      <input>
        <soap:body use="encoded" namespace="urn:GoogleSearch"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded" namespace="urn:GoogleSearch"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
  <!-- Endpoint for Google Web APIs -->
  <service name="GoogleSearchService">
    <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
      <soap:address location="http://api.google.com/search/beta2"/>
    </port>
  </service>
</definitions>
```

Приложение 4. Пример сгенерированного кода

```
//
// Types
//

/**
 * @param documentFiltering is of type boolean
 * @param searchComments is of type string
 * @param estimatedTotalResultsCount is of type int
 * @param estimateIsExact is of type boolean
 * @param resultElements is of type ResultElementArray
 * @param searchQuery is of type string
 * @param startIndex is of type int
 * @param endIndex is of type int
 * @param searchTips is of type string
 * @param directoryCategories is of type DirectoryCategoryArray
 * @param searchTime is of type double
 */
function GoogleSearchResult (documentFiltering, searchComments,
          estimatedTotalResultsCount, estimateIsExact, resultElements,
          searchQuery, startIndex, endIndex, searchTips, directoryCategories,
          searchTime)
{
          var nameMap = {"documentFiltering":documentFiltering,
                    "searchComments":searchComments,
                    "estimatedTotalResultsCount":estimatedTotalResultsCount,
                    "estimateIsExact":estimateIsExact,
                    "resultElements":resultElements,
                    "searchQuery":searchQuery,
                    "startIndex":startIndex,
                    "endIndex":endIndex,
                    "searchTips":searchTips,
                    "directoryCategories":directoryCategories,
                    "searchTime":searchTime};
          ComplexType.call(this, nameMap);
}

GoogleSearchResult.prototype = new ComplexType();

/**
 * @param summary is of type string
 * @param URL is of type string
 * @param snippet is of type string
 * @param title is of type string
 * @param cachedSize is of type string
 * @param relatedInformationPresent is of type boolean
 * @param hostName is of type string
 * @param directoryCategory is of type DirectoryCategory
 * @param directoryTitle is of type string
 */
function ResultElement (summary, URL, snippet, title, cachedSize,
          relatedInformationPresent, hostName, directoryCategory,
```

```
          directoryTitle)
{
          var nameMap = {"summary":summary,
                    "URL":URL,
                    "snippet":snippet,
                    "title":title,
                    "cachedSize":cachedSize,
                    "relatedInformationPresent":relatedInformationPresent,
                    "hostName":hostName,
                    "directoryCategory":directoryCategory,
                    "directoryTitle":directoryTitle};
          ComplexType.call(this, nameMap);
}

ResultElement.prototype = new ComplexType();

/**
 * @param elements is an array of ResultElement
 */
function ResultElementArray (elements)
{
          ComplexType.call(this, elements);
}

ResultElementArray.prototype = new ComplexType();

/**
 * @param elements is an array of DirectoryCategory
 */
function DirectoryCategoryArray (elements)
{
          ComplexType.call(this, elements);
}

DirectoryCategoryArray.prototype = new ComplexType();

/**
 * @param fullViewableName is of type string
 * @param specialEncoding is of type string
 */
function DirectoryCategory (fullViewableName, specialEncoding)
{
          var nameMap = {"fullViewableName":fullViewableName,
                    "specialEncoding":specialEncoding};
          ComplexType.call(this, nameMap);
}

DirectoryCategory.prototype = new ComplexType();

//
// Start of GoogleSearchPort class
//

function GoogleSearchPort ()
{
```

```
        // Port location
        this.location = "http://api.google.com/search/beta2";
}

GoogleSearchPort.prototype = new AbstractProxy();
/**
 * @param key is of type string
 * @param q is of type string
 * @param start is of type int
 * @param maxResults is of type int
 * @param filter is of type boolean
 * @param restrict is of type string
 * @param safeSearch is of type boolean
 * @param lr is of type string
 * @param ie is of type string
 * @param oe is of type string
 */
GoogleSearchPort.prototype.doGoogleSearch = function (key, q, start,
maxResults, filter, restrict, safeSearch, lr, ie, oe) {
        // build message
        var serializer = new SoapSerializer("rpc");
        var paramMap = {"key":key,
                "q":q,
                "start":start,
                "maxResults":maxResults,
                "filter":filter,
                "restrict":restrict,
                "safeSearch":safeSearch,
                "lr":lr,
                "ie":ie,
                "oe":oe};
        var message = serializer.toSoapMessage("doGoogleSearch",
"urn:GoogleSearch", paramMap);

        var callbackHandler = this.doGoogleSearchCallbackHandler;
        var errorHandler = this.doGoogleSearchErrorHandler;

        var responseParser = function (rawdata, isError) {
                if (!isError)
                {
                        var deserializer = new SoapDeserializer(rawdata);
                        // check if contains soap fault
                        if (!deserializer.hasFault())
                        {
                                // parse data into objects

        callbackHandler(deserializer.getVariable("return"));
                        }
                        else
                        {
                                errorHandler(deserializer.getFault());
                        }
                }
                else
                {
```

```
                            errorHandler("Network error");
                    }
            };

            var soapTransport = new SoapTransport(this.location, responseParser)

            soapTransport.send(message, "urn:GoogleSearchAction");
};

/**
 * @param return_ is of type GoogleSearchResult
 */
GoogleSearchPort.prototype.doGoogleSearchCallbackHandler = function (return_) {
            //Please write your callback handling code here!
};

/**
 * @param error is a string containing error message
 */
GoogleSearchPort.prototype.doGoogleSearchErrorHandler = function (error) {
            //Please write your error handling code here!
};

//
// End of GoogleSearchPort class
//
```