3. Веб

Пожалуй, Веб стал настолько распространенным не только из-за того, что он делает, но из-за того, что он не пытается делать. Так, Веб формирует очень простую основу, на базе которой можно легко создавать новые концепции. Однако сложность заключается в том, чтобы новые идеи не блокировали развитие самого изобретения. Именно поэтому основным направлением деятельности Консорциума Всемирной Паутины (W3C) и других заинтересованных групп является определение архитектуры Веба, способов ее поддержания и совершенствования. Это своеобразный баланс между естественным эволюционным развитием и сохранением уже накопленного опыта.

В основе веб-концепции лежит понятие *ресурса*. Ресурсом может быть что угодно – фотография, налоговая накладная, страна, политическое движение, алгоритм, мысль, человек – все, что имеет некоторые границы и поэтому может быть идентифицировано. Веб не задает никаких ограничений на допустимые ресурсы; все, что он определяет – это то, как эти ресурсы могут передаваться между компьютерами и, следовательно, людьми. Самая основная возможность любой распределенной системы – перемещать ресурсы с одной машины на другую. Для этого Веб поддерживает несколько очень простых технологий.

– *Именование ресурсов.* Веб определяет гибкие и расширяемые способы именования произвольных ресурсов, именуемые унифицированными идентификаторами ресурсов (URI – Uniform Resource Identifier).
– *Представление ресурсов.* Для передачи произвольных ресурсов между компьютерами, необходимо такое представление ресурса, которое может быть преобразовано в поток битов и передано по сети. В идеале формат представления должен быть принят всеми, что обеспечивает простоту создания и интерпретации ресурса. В настоящее время существует несколько типов представления ресурса. Первоначальный и наиболее важный из них – это язык гипертекстовой разметки (HTML – Hypertext Markup Language).
– *Передача ресурсов.* Hypertext Transfer Protocol (HTTP) стал основным механизмом передачи данных в Интернете. HTTP – это протокол передачи по типу клиент/сервер, который поддерживает минимальный необходимый набор операций передачи данных.

Вместе эти технологии стали очень мощным механизмом для создания широкого спектра распределенных приложений. После краткого введения о том, как зарождался Веб, мы более внимательно рассмотрим каждую из них.

3.1 Рассвет Веба

«Сайты должны обладать возможностью взаимодействия
в едином, универсальном пространстве».
Тим Бернерс-Ли

В 1990 году Тим Бернерс-Ли[1] создал первый веб-браузер. Это произошло во время его работы в лаборатории ЦЕРН в Швейцарии, где несколько тысяч ученых-ядерщиков использовали различные компьютеры. Бернерса-Ли не устраивал способ передачи данных между компьютерами сотрудников: для получения доступа к необходимой информации нужно было осуществить вход на компьютер другого пользователя. После написания нескольких программ для преобразования и передачи информации между различными системами, он задумался о более эффективных способах реализации. Вскоре возникла идея создания некоторой воображаемой информационной системы, которая могла бы использоваться всеми. На базе уже существующей концепции гипертекста был создан первый веб-браузер, который подключался через DNS и TCP, ставший началом развития Всемирной паутины (World Wide Web). По словам Тима Бернерса-Ли, это был лишь первый шаг, основная сложность заключалась в том, чтобы заинтересовать и привлечь пользователей.

Первый браузер, называвшийся WorldWideWeb, работал в среде NeXT, а первым веб-сервером стал `nxoc01.cern.ch`, позже переименованный в `info.cern.ch`. С этого момента начинается широкое распространение ПО, и к 1992 году насчитывается уже 50 веб-серверов по всему миру, а в 1993 NCSA выпускает альфа-версию браузера «*Mosaic for X*». Позже, в октябре 1994, Бернерс-Ли основал Консорциум Всемирной паутины (W3C), цель которого – стандартизация общих протоколов для содействия развитию Веба и обеспечения его совместимости. В это время происходит настоящий взрыв Всемирной паутины, так, число пользователей Интернета возросло с 40 миллионов в 1995 году до 150 млн. в 1998 году и 320 млн. в 2000 году. На сегодняшний день насчитывается 1 миллиард 350 миллионов пользователей Всемирной паутины по всему миру.

[1] Сэр Тимоти Джон Бернерс-Ли (родился в 1955 году) – британский учёный, изобретатель URI, URL, HTTP, HTML, изобретатель Всемирной паутины (Wold Wide Web), и действующий глава Консорциума Всемирной паутины (W3C).

3.2 Единообразное именование ресурсов

Центральное место в архитектуре веба занимает унифицированный (единообразный) идентификатор ресурса (URI – Uniform Resource Identifier). URI представляет собой последовательность символов, которая однозначно идентифицирует ресурс в Интернете. URI обеспечивает единый механизм именования произвольных ресурсов и имеет очень простой синтаксис.

Рассмотрим некоторые примеры:

1. http://www.google.com
2. urn:isbn:0-395-36341-1
3. urn:jxta:uuid:59616261646162614A78746150325033F3BC76
4. mailto:john.doe@example.com
5. http://www.bbc.co.uk/weather/
6. http://www.google.com/search?client=safari&q=web

Каждый URI начинается со схемы обращения к ресурсу (часто под схемой обращения к ресурсу подразумевается сетевой протокол). В приведенных выше примерах мы видим схемы http, схему mailto, используемую почтовыми клиентами, и две схемы urn. Также доступны такие схемы, как ftp для протокола передачи файлов, sip для протокола установления сеанса, который в основном используется для передачи голоса по IP-телефонии, и даже схема tel, предназначенная для идентификации телефонных номеров.

После схемы обращения к ресурсу в URI указывается адрес хоста и номер порта, где расположен необходимый ресурс. По умолчанию HTTP использует 80-й порт, и, если номер порта не указан, подразумевается, что ресурс находится именно на этом порту. Если ресурс расположен на нестандартном порту, например, 8080, то необходимо после адреса хоста через двоеточие прописать нужный порт, к примеру, http://www.bbc.co.uk:8080/weather/.

Стандарт URI включает в себя стандарты *имени ресурса* (URN – Uniform Resource Name) и *унифицированного локатора ресурса* (URL – Uniform Resource Locator).

URL описывает ресурс с точки зрения протокола, который используется для доступа к нему. Например, вышеприведенные идентификаторы http и mailto. *URN* описывает ресурс в некотором пространстве имен, к примеру, уникальный номер книжного издания ISBN.

Традиционно URL и URN рассматривались как формальные подмножества URI-пространства. Однако в настоящее время понятия URI, URL и URN часто используются как синонимы. Основное различие между URN и URL состоит в том, что URL содержит некоторый сетевой адрес ресурса. Это мы можем наблюдать в примерах с http и mailto. В таких адресах указано имя хоста

(Google.com, bbc.co.uk и example.com), которое используется приложением для получения физического адреса сервера, где расположен ресурс. Получение физического адреса происходит при помощи системы доменных имён (DNS – Domain Name System). Однако URN также может быть использован для получения физического сетевого адреса, для этого проводят некоторую предварительную обработку URI запроса перед его непосредственным выполнением. К примеру, вышеприведенный идентификатор urn:jxta. Платформа JXTA преобразовывает уникальный JXTA-идентификатор в сетевой адрес вычислительного узла JXTA.

Схема HTTP является одной из наиболее распространенных в Интернете. Она классифицируется как иерархический URI, поскольку в ней можно выделить несколько отдельных частей. Так, в примере №5, мы наблюдаем три части. Первая – это схема http, которую мы уже упомянули. Вторая часть – это сетевой адрес ресурса (www.bbc.co.uk). Он используется браузером для получения физического IP-адреса сервера, где расположен ресурс, при помощи DNS. Последняя часть – это непосредственный путь к ресурсу, в нашем примере, weather. Путь определяется относительно хоста, найденного браузером при помощи DNS. Другими словами, символьная строка weather ссылается на ресурс, расположенный на хосте BBC.

URI типа HTTP может также содержать строку запроса. Примером такого идентификатора является пункт №6 из приведенного ранее списка. Можно наблюдать URI такого типа, если ввести поисковый запрос на главной странице Google. Начало запроса обозначается вопросительным знаком (?). Далее идут последовательности пар ключ/значение, каждая из которых отделена знаком амперсанд (&). Между ключом и значением ставится знак равенства (=). Рассматриваемый нами пример URI в строке запроса содержит две пары ключ/значение:

– client = safari
– q = web

Ключи и их значения распознаются сервером, получающим запрос. В этом примере ключ client указывает на браузер, где формировался запрос. Здесь это браузер Safari от компании Apple. Сервер может выдавать ответ в разных форматах в зависимости от указанного браузера. Ключ q отражает текст поискового запроса, введенного пользователем. В нашем случае это текст web.

URI может также содержать идентификатор необходимого фрагмента, представляющий собой последовательность символов, отделенную от пути к ресурсу решеткой (#).

Идентификатор фрагмента в URI позволяет адресовать определенную часть ресурса. Например, очень большой документ, который структурно поделен на разделы. Использование идентификатора фрагмента позволяет указать браузеру, какую часть документа необходимо отобразить, при этом избавив пользователя от необходимости прокрутки всего документа: http://server.com/documents/bigdocument.html#section3.

Для возможности навигации по HTML-документу соответствующий раздел должен быть помечен тегом со значением section3, уникальным в пределах этого документа.

Иерархический URI, может быть абсолютным или относительным. Последний указывается относительно другого URI, который тоже в свою очередь может быть относительным. На веб-страницах часто можно встретить такие относительные ссылки, например: ../images/fido.jpg.

Как правило, такой URI определяется от места расположения документа, в котором он упоминается. Следовательно, если вышеупомянутая ссылка находится в документе http://server.com/pets/dogs/index.html, то она будет обрабатываться следующим образом: сначала будет определен путь к родительскому элементу страницы index.html, т.е. dogs. Далее для каждого вхождения символов ../ будет получен путь к родительскому элементу абсолютного URI. Как и в Unix-подобных системах, символы ../, содержащиеся в пути, означают подъем на один уровень вверх. В данном случае мы поднимаемся на один уровень вверх к элементу pets. Наконец, к абсолютному URI добавляем относительный, чтобы получить следующий адрес: http://server.com/pets/images/fido.jpg.

Обратите внимание, что относительный URI не может адресовать ресурс, находящийся по другому сетевому адресу. Следовательно, вы не можете задавать URI относительно идентификатора, указывающего на другой хост.

3.2.1 Шаблоны URI

По словам Бернерса-Ли, URI должен быть непрозрачным для пользователя, даже если составляющие его компоненты могут быть легко выделены [6]:

«Идентификатор может быть использован только для обращения к объекту. Не следует смотреть на содержимое URI-строки с целью извлечь дополнительную информацию».

Аргументом в пользу такого подхода является то, что чем больше пользователь понимает и вчитывается в URI, тем более хрупкой становится система, так как пользователь может сформировать подобные URI, которые не существуют в пространстве имен сервера. Данный метод также делает систему более связанной, поскольку появляется возможность совместного использования зна-

чений в URI, которые подаются на сервер. Это в свою очередь раскрывает внутреннюю логику сервера и затрудняет ее изменение без последующего влияния на механизм внешней идентификации.

Однако, помимо непосредственной идентификации ресурсов, различные компоненты URI интуитивно задают отношения между частями идентификатора. Например, путь к ресурсу в URI представляет собой последовательность символов, разделенных косой чертой. Порядок расположения компонентов пути в URI по структуре напоминает граф или дерево, подобно иерархии файлов и папок на локальном компьютере. Этот порядок может быть использован для логической группировки ресурсов. Например, URI http://server.com/media/photos и http://server.com/media/music показывают, что раздел media содержит два подмножества: photos и music. URI сам по себе не дает дополнительных сведений о внутренней работе или организации ресурсов на стороне сервера. Тогда как сервер, использующий URI, предоставляет пользователю возможность строить логические связи между существующими ресурсами. Такое устройство подобно супермаркету, где однотипные товары выложены таким образом, что поблизости с чайным прилавком можно найти и кофе. В ином случае, посетитель может растеряться.

Запрос в URI также задает некоторый алгоритм, который будет выполняться на сервере в ответ на конкретные значения, введенные в поле запроса, что не следует путать с вызовом методов на сервере, которые принимают определенные параметры. Возвращаясь к примеру с супермаркетом, это как попросить консультанта показать, где представлен кофе, и натуральный, и крепкий одновременно. Продавец может показать отдельно натуральный кофе, отдельно крепкий, но такой подход может стать обременительным, если представлен очень широкий ассортимент.

В последнее время специалисты полагают, что такая логическая организация URI полезна, особенно в связи с широким распространением веб-сервисов. Частью этого движения является создание спецификации шаблона URI, которая в настоящее время находится на стадии разработки в Инженерном совете Интернет (IETF Network Working Group).

Шаблон URI – это идентификатор с внедренными в него переменными. Переменная заключена в фигурные скобки (фигурные скобки не допускаются в URI, поэтому такой идентификатор не может быть спутан с действительным URI). Чтобы шаблон URI преобразовать в стандартный идентификатор, необходимо заменить переменные их фактическими символьными значениями, характерными для конкретной ситуации. Например, шаблон

http://www.server.com/users/{userId} будет заменен на http://www.server.com/users/1234, если значение идентификатора конкретного пользователя равно 1234. Шаблоны URI особенно подходят для машинной обработки, поскольку они позволяют веб-сервису описать способ доступа к ресурсам в абстрактных терминах. Личные страницы каждого из пользователей вымышленной службы на www.server.com, одинаковы и описаны службой только один раз до тех пор, пока переменная в URI не будет заменена на идентификатор конкретного пользователя. Это в свою очередь позволяет службам публиковать свои интерфейсы в общем, машиночитаемом виде, чего зачастую не хватает в Вебе.

URI оказался невероятно универсальным. Эта универсальность обеспечивает простой и достаточно гибкий синтаксис для именования любых веб-объектов. URI со схемой HTTP сегодня используется для именования практически бесконечного количества ресурсов. Идентификаторы URI не только связывают разрозненные веб-элементы, позволяя ссылаться на ресурсы, географически расположенные в разных концах Земли, но и прочно вошли в нашу жизнь и широко используются вне Интернета: в беседах, на обратной стороне конвертов, в газетах, на рекламных щитах и автомобилях, в поездах и самолетах.

Перенос понятия URI из цифрового мира в реальный, естественно, повлиял на то, как мы воспринимаем и понимаем Веб с точки зрения его архитектуры. Например, Веб никогда не был нацелен на надежное, понятное обнаружение ресурса. В целом, обнаружение ресурса либо излишне завязано на Веб, например, осуществляется через поисковую систему, или услуги, такие как Technorati (поисковая машина для блогов), либо заключается в случайном переходе по ссылкам.

Тем не менее, тот факт, что URI встречается практически повсеместно, означает, что процесс обнаружения ресурсов может стать настолько децентрализованным, что выходит за пределы Интернета. Обедая в кафе, вы можете натолкнуться на рекламную брошюру печи для пиццы, на которой будет ссылка на официальный сайт производителя. Однако вы никогда не встретите ссылку на объект CORBA, или даже WSDL-документ веб-сервиса, в аналогичных обстоятельствах.

3.3 Общее представление ресурсов

Бернерс-Ли выдвинул и описал принцип наименьшей силы (the Principle of Least Power), который гласит, что *для выражения чего-либо необходимо использовать наименее мощный язык из доступных*. Этот принцип заключается

в следующем: чем сложнее выражение, тем меньше шансов, что оно будет понятным другим людям, и они смогут преобразовать его в язык или представление, наиболее подходящее для их нужд. При совместном использовании представлений различными организациями и сообществами, важно, чтобы формат этого представления был наименее централизованным и эксклюзивным.

Язык гипертекстовой разметки HTML обеспечивает очень низкий порог вхождения в описание ресурсов. Он в первую очередь нацелен на способ представления ресурса в человеческом восприятии, а не на описание возможностей или внутренней структуры ресурса. Это очень отличается от обмена сериализованными Java-объектами, например, в Jini, где представление содержит информацию о возможностях и внутренней структуре ресурса. Обратная сторона медали состоит в том, что только клиенты, знающие внутреннюю структуру и представление ресурса, могут восстановить его. С другой стороны, структура HTML не привязана к структуре ресурса – они ортогональны. Автор представления вынужден адаптировать свою концепцию ресурса (автомобиля, ходатайства, собаки) к ограниченному словарю HTML. Пусть полученное представление не будет совершенным, но, по крайней мере, все желающие, будь то автомеханик, политик или любитель собак, смогут получить к нему доступ. И в зависимости от сферы их деятельности, они могут извлечь из представления необходимую информацию, чтобы преобразовать ее в свой собственный формат, определяющий внутреннюю структуру ресурса. HTML действует как метаязык типа «многие-ко-одному» для преобразования ресурсов. В результате, он накладывает массу ограничений.

Язык гипертекстовой разметки HTML, в отличие от любого другого языка разметки, превосходит традиционное понятие линейного, плоского текста, содержание которого определено автором и неизменяемо читателем. Напротив, HTML предоставляет пользователю множество вариантов.

3.4 Протокол передачи гипертекста

Протокол передачи гипертекста (HTTP – Hypertext Transfer Protocol) стал «де-факто» способом обмена ресурсами в Интернете. На самом деле, он настолько полезный, гибкий и повсеместно распространенный, что при определенных обстоятельствах используется в других системах, таких как JXTA и Gnutella. Кроме того, веб-сервисы на основе SOAP почти всегда используют туннелирование HTTP для SOAP-конвертов.

HTTP является простым протоколом прикладного уровня. Он поддерживает несколько методов, аналогичных операциям, которые выполняются в базе данных или хранилище на основе ключ/значение. Наряду с именем метода, клиент посылает серверу идентификатор ресурса и несколько заголовков HTTP. Идентификатором ресурса является часть URI, расположенная после имени хоста. Заголовки HTTP представляют собой пары ключ/значение. Наиболее широко используемые методы HTTP.

- **GET** извлекает ресурс по данному URI. Этот запрос не содержит представления ресурса.
- **DELETE** удаляет ресурс по данному URI. Этот запрос не содержит представления ресурса.
- **PUT** обновляет или изменяет ресурс по данному URI. Если требуемого ресурса не существует, сервер может создать новый ресурс по данному URI. Этот запрос содержит представление ресурса.
- **POST** используется по-разному в разных контекстах. В целом, этот метод предназначен для создания нового ресурса, представление которого направляется серверу в запросе. Созданный ресурс логически подчинен тому ресурсу, на URI которого пришел POST-запрос. На практике именно сервер решает, что значит «подчиненный». Запрос содержит представление ресурса.
- **HEAD** получает метаданные о ресурсе. Запрос возвращает все те же данные клиенту, что и GET, за исключением тела ресурса, то есть описание не включено в ответ. Этот запрос не содержит представления ресурса.
- **OPTIONS** возвращает список методов HTTP, которые доступны по данному URI. Этот запрос не содержит представления ресурса.

Итак, как выглядит запрос HTTP? На рис. 4 приведен простой пример запроса HTTP, который может быть отправлен на страницу погоды BBC http://www.bbc.co.uk/weather/.

```
GET / weather / HTTP/1.1
User-Agent: curl/7.16.3
Host: www.bbc.co.uk
Accept: * / *
```

Рис. 4. Пример HTTP-запроса GET

Первая строка запроса определяет метод – GET – путь ресурса, а именно, часть полного URI после адреса хоста, и используемую версию HTTP – в данном случае версия 1.1. Далее следуют заголовки. Заголовок User-Agent указы-

вает серверу, какое программное обеспечение используется для создания запроса. В заголовке Host указан адрес хост-сервера из полного URI. Заголовок Accept определяет тип данных, который готов принять клиент. Это может быть любой из типов многоцелевого расширения почты Интернет (MIME – Multipurpose Internet Mail Extensions): text/html, text/plain или image/jpeg и image/png. Как видим, MIME-тип имеет тип – значение перед косой чертой, и подтип – значение после косой черты. Таким образом, html и plain – подтипы text, так же, как jpeg и png являются подтипами типа image. В нашем примере, звездочка (*) используется, чтобы обозначить любой тип. Следовательно, клиент будет рад принять ответ любого типа и любого подтипа.

```
HTTP/1.1 200 OK
Date: Ср, 26 Mar 2008 16:03:01 GMT
Server: Apache/2.0.59 (Unix)
Content-Length: 12345
Content-Type: text/html
```

Рис. 5. Пример ответа на GET-запрос

На рис. 5 представлен пример ответа на указанный ранее GET-запрос. В первой строке указана используемая версия HTTP, затем код статуса. Код ответа 200 означает, что все прошло удачно и запрашиваемый ресурс содержится в сообщении. Другими распространенными кодами ответа являются 201 Created, если ресурс успешно создан на сервере, 404 Not Found, если сервер не может отобразить ресурс по заданному пути, и 403 Forbidden, если клиент не имеет прав для доступа к ресурсу. Всего имеется 40 стандартных кодов ответа, покрывающих все возможные ситуации. В приведенном выше примере сервер возвращает заголовки, описывающие дату и время выполнения запроса, серверное программное обеспечение, длину возвращаемого клиенту представления в байтах и MIME-тип представления. После заголовков следуют фактические данные, если таковые имеются.

Методы HTTP и условия, в которых они используются, часто рассматриваются с точки зрения двух свойств – *безопасность* и *идемпотентность* (метод называется идемпотентным, если при множественном вызове возвращает тот же результат, что и при единичном вызове).

Безопасная операция не производит побочного эффекта. С точки зрения пользователя, побочный эффект – это изменение состояния ресурса, с которым происходит взаимодействие. В соответствии с соглашением, методы GET, HEAD и OPTION считаются безопасными, и поэтому не должны применяться в тех случаях, когда их использование может вызвать изменение состояния на сервере. Даже если такое изменение происходит, клиент, делающий запрос, не

несет ответственности за побочный эффект. Методы PUT, POST и DELETE, как правило, не безопасны. Они должны использоваться в ситуациях, сопровождающихся изменением состояния ресурса на сервере. Когда такие методы используются, безопасность подразумевает негласный контракт между сервером и клиентом. Выполнение безопасного метода, например, GET, не подразумевает наличие контракта; сервер не ожидает от клиента последующих запросов и на сервере не произойдет никаких изменений, за которые ответственен клиент. Поэтому, если сервер использует безопасный метод в ситуации, когда происходит побочный эффект, подразумевается, что он нарушает договор с клиентом.

Поддержание этого контракта помогает сохранить Веб слабосвязанным, нарушение же может привести к нежелательным результатам. Например, обычно после регистрации пользователь получает письмо, содержащее ссылку для подтверждения своей подписки. Он переходит по этому адресу, и браузер выполняет GET запрос на сервер. После возвращения контента браузеру, пользователь видит страницу с поздравлениями о вступлении в список рассылки. Однако, клиент посылал безопасный GET запрос, не вступая в договорные отношения с сервером. В таком случае на GET-запрос сервер должен вернуть страницу с кнопкой «Нажмите здесь, чтобы подтвердить подписку». И уже при нажатии этой кнопки, будет сформирован POST-запрос к серверу, который добавляет пользователя в базу данных списка рассылки.

Как уже было сказано выше, *идемпотентный метод* – это такой метод, который при множественном вызове возвращает результат, как будто был вызван единожды. Этим свойством, как правило, обладают методы GET, HEAD, OPTIONS, PUT и DELETE. Безопасные методы по своему определению идемпотентны. Свойство идемпотентности особенно полезно в распределенном контексте, поскольку позволяет повторять запрос при возникновении ошибки во время выполнения. Например, в случае отказа сервера клиент остается без ответа. Если запрос ни идемпотентный, ни безопасный, то клиенту предстоит принять серьезное решение о том, стоит ли повторять запрос, потому что повторение может вызвать нежелательный побочный эффект.

URI, HTTP и HTML составляют основу веб-технологий. Однако не существует программного обеспечения, которое бы понимало и объединяло их. На стороне сервера, требуется веб-сервер, который может в ответ на HTTP-запросы обслуживать контент ресурсов. На стороне клиента, необходимо программное обеспечение, такое как браузер, для отображения HTML и предоставления возможности пользователю переходить по ссылкам.

3.4.1 HTTP и безопасность

Безопасные веб-соединения снабжены разнообразными методами защиты в зависимости от их потребностей. HTTP поддерживает аутентификацию пользователя непосредственно через определенные заголовки и коды ответов. В типичном сценарии браузер переходит по ссылке на конкретный ресурс. Если ресурс ограничен правами пользователей, то сервер возвращает код состояния 401 Unauthorized. Вместе с кодом ответа, сервер посылает заголовок WWW-Authenticate, где указывается ожидаемый тип и название аутентификации. Существует два типа аутентификации – базовая и дайджест. Для аутентификации обоих типов необходимы имя пользователя и пароль.

В базовом типе аутентификации имя и пароль кодируются при помощи кодировки Base64. Base64 – это система кодирования, предназначенная для представления бинарных объектов в тексте письма. Эта кодировка отнюдь не является безопасной – с помощью простого алгоритма можно преобразовать закодированную строку обратно в обычный текст.

Дайджест-аутентификация использует хэш-функции для шифрования учетных данных. С помощью хэш-функции получают необратимый ключ фиксированной длины. Распространенным примером такой функции является MD5. Имя аутентификации представляет собой защищенную область веб-сайта, как правило, относящуюся к идентификатору домена сервера.

Когда браузер получает код состояния 401, он предоставляет пользователю поля для ввода логина и пароля. Когда пользователь заполняет эту информацию, браузер ретранслирует первоначальный запрос, на этот раз, добавляя заголовок Authenticate, содержащий тип аутентификации и учетные данные, закодированные с помощью либо Base64 для базовой аутентификации, либо хэш-функции, в случае использования дайджест-аутентификации. Если сервер принимает учетные данные, он предоставляет право доступа, и браузер получает содержимое защищенной страницы.

Механизм авторизации HTTP имеет ряд ограничений. Во-первых, при базовой аутентификации учетные данные отправляются в виде обычного текста, который может быть перехвачен третьими лицами и легко декодирован. Во-вторых, даже если для аутентификации используется безопасная хэш-функция, все последующие обмены данными производятся в открытом виде. Когда передаются такие конфиденциальные данные, как номера кредитной карты, это недопустимо. В таких ситуациях веб-серверы обычно используют уровень защищенных сокетов (*SSL – Secure Socket Layer*) и/или протокол безопасности транспортного уровня (*TLS – Transport Layer Security*), являющийся его преем-

ником. Эти протоколы являются «прослойкой» между протоколами TCP и HTTP, располагаясь на 6-м уровне (уровне представления) в рамках модели OSI. Ресурс, который требует безопасного соединения, использует протокол HTTPS в схеме URI, для того, чтобы указать, что он защищен.

Зачастую сервер с поддержкой TLS имеет пару открытый/закрытый ключ и цифровой сертификат, подтвержденный удостоверяющим центром (CA – Certification Authority). Этот сертификат включает в себя открытый ключ с подписью CA, которая гарантирует, что это на самом деле сервер. Когда клиент запрашивает ресурс через защищенное соединение, он посылает серверу ряд параметров, включающих случайное число, используемое в дальнейшем для генерации ключа сеанса, а также версию TLS и поддерживаемые алгоритмы шифрования (cyphers) и хэш-функции. Сервер выбирает наиболее устойчивые алгоритмы шифрования и хэш-функции, доступные и клиенту, и серверу, и сообщает клиенту, какой алгоритм будет использован во время последующего обмена сообщениями. Наряду с этой информацией, сервер также отправляет клиенту случайное число, а затем и сертификат.

На этом этапе клиент может обратиться в удостоверяющий центр для проверки подлинности сертификата сервера. Браузеры обычно обладают списком известных центров сертификации, где они могут проверить подлинность подписи сертификата. Если подпись не известна браузеру, как в случае самоподписанного сертификата, он предложит пользователю принять или отклонить данный сертификат.

Если сертификат считается надежным, то клиент генерирует еще одно случайное число, известное как PreMasterKey, шифрует его с помощью открытого ключа сервера, и отправляет на сервер. Только сервер может расшифровать это число, используя свой закрытый ключ. Затем обе стороны генерируют ключ сеанса на основе случайных чисел, первоначально посланных друг другу, и PreMasterKey. Ключ сеанса используется для шифрования и дешифрования всех сообщений обмена в пределах открытой сессии.

TLS также поддерживает взаимную аутентификацию, при которой клиент также должен обладать парой открытый/закрытый ключ, хотя этот способ менее распространен в Интернете. В таком случае, сервер запрашивает сертификат от клиента. После отправки PreMasterKey на сервер, клиент подписывает сообщения, которыми они обменялись во время процедуры handshake (рукопожатия). Сервер может проверить эту подпись с помощью открытого ключа клиента, убедившись, что клиент является тем, за кого он себя выдает.

3.5 Заключение

Методы именования, предоставления и передачи ресурсов, предоставляемые Веб, на сегодняшний день, применяются множеством различных распределенных вычислительных систем. Такой подход позволят разработчикам РВС реализовывать решения, совместимые с существующими стандартами и методами взаимодействия удаленных вычислительных систем.