УДК 681.3.06

КЛАССИФИКАЦИЯ СВЯЗЕЙ МЕЖДУ ЧАСТЯМИ ВЕБ-ПРИЛОЖЕНИЯ И ЕГО ОПИСАНИЕ С ИСПОЛЬЗОВАНИЕМ МОДЕЛИ КОНЕЧНЫХ АВТОМАТОВ

А.А. Ожиганов, А. И. Чепурной

Приводится классификация связей между элементами приложения и их описание для приложения-примера согласно выполненной классификации. Разработан способ описания связей с помощью модели конечных автоматов. Проведен анализ свойств полученного конечного автомата.
Ключевые слова: веб-приложение, конечные автоматы, проектирование программного обеспечения.

Введение

В настоящее время в индустрии разработки программного обеспечения наблюдается значительный рост количества веб-приложений, что делает важной задачу формализации процесса их проектирования, разработки и тестирования. Веб-приложение – это приложение, доступное посредством веб-браузера через сеть (Интернет или локальную корпоративную сеть – Интранет). В процессе работы с веб-приложением пользователь использует веб-браузер, запрашивая документы с веб-сервера, и получает их в ответ. Подобное взаимодействие осуществляется с помощью http-протокола и носит последовательностный характер. Браузер пользователя запрашивает с сервера с помощью http-запроса документ (обычно в формате HTML-кода, возможно, со встроенными мультимедийными компонентами, апплетами и скриптами, но возможны и другие варианты формы ответа – например, XML-документы, изображения, файлы текстовых документов), затем пользователь работает с полученным документом, и, когда в ходе работы оказывается необходимым получить новый документ с сервера (при переходе на новый документ по ссылке, отправке на сервер формы с данными, автоматическом перенаправлении и т.д.), описанный цикл повторяется сначала.

На стороне сервера код HTML-страниц может формироваться статически, при этом клиенту посылается содержимое хранящихся на сервере файлов с кодом разметки в формате HTML, или динамически, «на лету». Само веб-приложение может выступать в качестве клиента других служб, например, базы данных или другого веб-приложения, расположенного на другом сервере.

Проблемы, возникающие в процессе разработки и тестирования веб-приложений

Одной из проблем в разработке, тестировании и поддержке разработанных веб-приложений является гетерогенный характер их структуры. Компоненты приложения работают на различных аппаратных и программных платформах, написаны на различных языках с использованием различных стилей программирования. Одно приложение может состоять из компонент, написанных на разных языках – процедурных, объектно-ориентированных или гибридных, наподобие JSP. Клиентская часть включает в себя браузеры, исполняемые в их среде встроенные скриптовые языки и апплеты, серверная – HTML, CGI, Java Server Pages, Java Servlets, .NET.

Все эти части взаимодействуют между собой, с программным обеспечением промежуточного уровня, с системами управления базами данных и с другими компонентами, которые могут находиться на стороне клиента, стороне сервера или других компьютерах, связанных с сервером. Кроме того, современные веб-приложения имеют, как правило, способность менять свой графический интерфейс пользователя в зависимости от команд пользователя, состояния клиента, состояния сервера; настольные же приложения обычно имеют полностью статичный интерфейс пользователя.

Постановка задачи

Исходя из приведенного анализа, представляется целесообразным упростить процесс разработки веб-приложений. С этой целью в данной работе предлагается способ выделения в веб-приложениях элементов, описание связей между ними и моделирование веб-приложения как системы элементов и связей между ними с помощью теории конечных автоматов.

Разбиение веб-приложения на элементы и связи между элементами

Определим веб-страницу как информацию, которая отображается в одном окне браузера. На сервере она, как уже отмечалось выше, может храниться как статичный HTML-файл или генерироваться динамически различными способами. Веб-сайт – это множество веб-страниц, объединенных семантически (по содержимому) и синтаксически (с помощью ссылок или других механизмов связи). Связи между статичными и динамическими страницами и программным обеспечением приложения могут быть разбиты на следующие категории [1].

1. Статические связи (HTML → HTML) – гиперссылки в статичном HTML-коде. Динамически сгенерированные ссылки, а также динамически генерируемое содержимое или элементы программного обеспечения не входят в данный тип связей.
2. Динамические связи (HTML → ПО). В коде страницы содержатся формы, пользователь заполняет одну из них и отсылает обратно на сервер, где данные формы обрабатываются программным обеспечением.
3. Динамически генерируемый HTML-код (ПО → HTML). HTML-код не хранится на сервере, а создается программным обеспечением сервера, в зависимости от входных воздействий.
4. Зависящий от состояния интерфейс пользователя (ПО + состояние → HTML). В данном случае формируемый HTML-код определяется не только входными воздействиями на компоненты программного обеспечения, но и состоянием сервера, например, временем, информацией о сессии пользователя, его привилегиями и т.п.
5. Прямые переходы (пользователь). Не контролируемые веб-приложением переходы – переписывание URL в адресной строке браузера или использование в нем кнопок «вперед» и «назад».
6. Связи между различными элементами ПО. Данный тип связей включает в себя вызовы между компонентами программного обеспечения веб-приложения.
7. Внешние связи. Программные вызовы к ресурсам, находящимся за пределами веб-сервера приложения.
8. Динамические подключения. Платформы Java и .Net позволяют подключать новые компоненты в веб-приложение прямо во время его работы. В платформе Java, к примеру, для достижения этого используются технологии Java reflection и Java beans.

На рис. 1 приведена классификация связей между элементами веб-приложения.

Связи в веб-приложении можно разделить на внешние (тип 7 из вышеперечисленных) и внутренние (все остальные). Внутренние связи, в свою очередь, делятся на связи реакции пользователя – переходы по ним определяются действиями пользователя, и автономные – переходы по ним определяются программной логикой (типы 3, 4, 6, 8). Связи реакции пользователя делятся на контролируемые связи реакции пользователя (они определены в веб-страницах, типы 1 и 2), и прямые переходы (тип 5).

Рассмотренные связи определяют структуру веб-приложения. Моделирование множества связей, т.е. формализованное описание существующих в приложении связей, способно упростить разработку приложений и их тестирование.

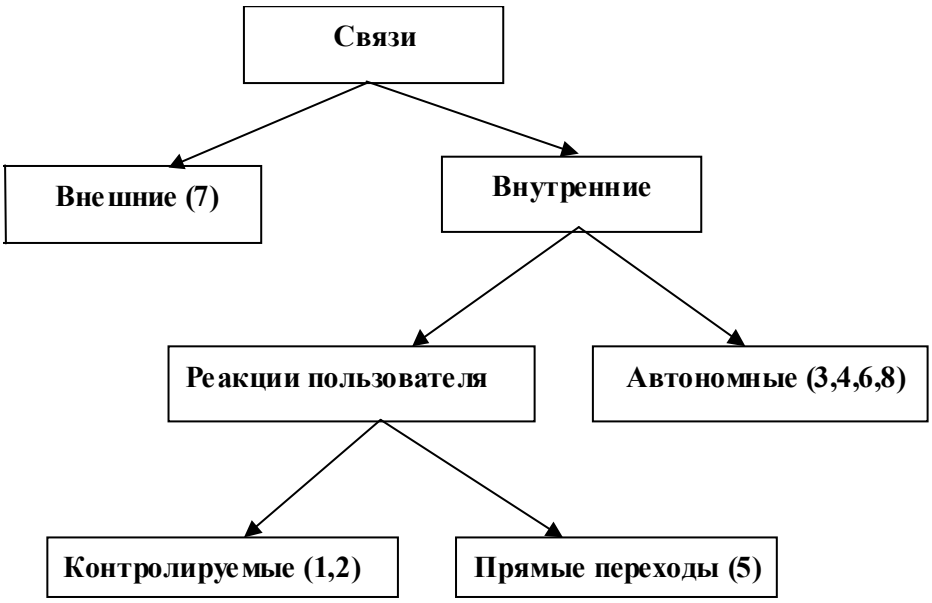

Рис. 1. Классификация связей между элементами веб-приложения

Пример веб-приложения

Рассмотрим в качестве примера веб-приложения гостевую книгу. Пусть алгоритм ее работы будет следующим:

- на начальном экране этого приложения у пользователя есть два выбора – «Добавить запись» и «Просмотреть книгу»;
- если пользователь выбрал «Добавить запись», то выводится экран ввода новой записи в гостевую книгу (имя пользователя, почтовый адрес, текст сообщения). Пользователь может ввести новое сообщение и отправить его или вернуться в меню;
- в случае выбора пользователем режима просмотра гостевой книги записи извлекаются из базы данных и отображаются на экране. При желании пользователь может ввести новое сообщение.

Таким образом, приложение имеет три экранные формы. Меню первой предлагает пользователю выбор – добавлять ли сразу новую запись в гостевую книгу или просмотреть существующие записи. Вторая предназначена для добавления нового сообщения, в ней есть форма ввода сообщения и данных об отправителе. Также на ней есть две кнопки – «Отправить» и «Вернуться в меню». При нажатии на первую сообщение добавляется в базу данных приложения, при нажатии на вторую происходит возврат в меню. Третья экранная форма предназначена для просмотра уже введенных предварительно в книгу и хранящихся в базе данных приложения сообщений: если при просмотре пользователь решит ввести новое сообщение, он может перейти на экран ввода новой записи, нажав на кнопку «Оставить сообщение».

Конкретная реализация может быть выполнена с использованием скриптового языка PHP и СУБД MySQL и состоять из следующих четырех страниц:

- select.html – в данной странице хранится код разметки экрана начального выбора;
- viewRecords.php – страница просмотра записей гостевой книги. При обращении к этой странице происходит соединение с базой данных, выборка из нее записей и отображение полученных данных в HTML-коде, отсылаемому клиенту;
- insertForm.html – эта страница содержит форму ввода нового сообщения;
- doInsert.php – при обращении к данной странице методом POST передаются данные параметры новой записи гостевой книги. PHP-код страницы соединяется с базой

данных и добавляет в таблицу записей гостевой книги новую запись на основе параметров. Затем происходит перенаправление на страницу viewRecords.php.

Приведенный пример гостевой книги является упрощенной версией реальных приложений гостевых книг. Оно имеет лишь три поля данных, в то время как в реальных приложениях их могут быть десятки [2]. В коде страницы doInsert.php отсутствует проверка вводимых значений на корректность, а профессионально написанное веб-приложение должно иметь такие проверки. Тем не менее, наш пример близок к эксплуатируемым в реальных условиях веб-приложениям и достаточен для иллюстрирования типичных проблем в разработке веб-приложений с помощью популярных сегодня технологий динамических страниц, таких как ASP, JSP, PHP [3]. Более того, взаимодействие гостевой книги с располагающейся на стороне сервера базой данных типично для других подобных приложений, таких как сервер новостей, хранилище статей, фотогалерея.

Рассмотрим связи между страницами согласно приведенной выше классификации и пронумеруем их.

1. Статические связи – select.html → insertForm.html (по нажатию на кнопку «Добавить запись») (С1), select.html → viewRecords.php (по нажатию на кнопку «Просмотреть книгу») (С2), viewRecords.php → insertForm.html (по нажатию на кнопку «Оставить сообщение») (С3), insertForm.html → select.html (по нажатию на кнопку «Вернуться в меню») (С4)**.**
2. Динамические связи – insertForm.html → doInsert.php (отсылка формы после нажатия на кнопку «Отправить») (С5).
3. Динамически генерируемый HTML-код – viewRecords.php → viewRecords.php (С6).
4. Зависящий от состояния интерфейс пользователя (ПО+состояние → HTML) – связей данного типа нет.
5. Прямые переходы – так как никаких ограничений на прямые переходы в коде приложения нет, прямые переходы связывают каждую страницу с каждой.
6. Связи между различными элементами ПО – doInsert.php → viewRecords.php (С7).
7. Внешние связи – связей данного типа нет.
8. Динамические подключения – связей данного типа нет.

Рассмотрим подробнее связь типа 3 viewRecords.php → viewRecords.php (С6). В работе [2] указываются следующие вехи в веб-разработке.

А. Модель статичных страниц (static page model). В первой половине 90-х г.г. все страницы были статическими, т.е. содержали в себе готовый документ в формате HTML (или другом формате представления данных, например, XML). Соответственно, страницы не требовали предварительной обработки и не зависели от аргументов.

Б. CGI-сценарии. Исполняемые на сервере программы используют для вывода результата библиотечные функции записи в стандартный вывод (print для языков C/C++, write для языка Pascal и т.п.). Код представления данных разбросан среди кода программы в аргументах вызовов этих функций.

В. Модель динамических страниц (dynamic page model). Динамическая страница пишется в терминах языков представления данных, а функциональная часть встраивается в код страницы с помощью специальных тегов – скриптлетов.

Г. Модель MVC. Реализация шаблона проектирования «модель–представление–контроллер» (Model–View–Controller) для разделения зависимости кода представления и кода программного обеспечения, работающего с ресурсами сервера.

Наш пример выполнен с помощью модели динамических страниц, поэтому связь вида 3 связывает код работы с базой данных и код представления внутри одной страницы viewRecords.php. При реализации в модели MVC данная связь существовала бы между разными страницами.

Описание структуры веб-приложений с использованием модели конечных автоматов

Конечный автомат (КА) представляет собой модель дискретного устройства, имеющего внутреннюю память (переменные состояния), а также набор входов и выходов. В общем случае КА может быть задан с помощью набора из пяти элементов $K=(Q, \sum, \delta, q_0, F)$ [4], где Q – конечное множество состояний автомата; q_0 – начальное состояние автомата ($q_0 \subset Q$); F – множество заключительных (или допускающих) состояний, таких, что F является подмножеством Q (при достижении одного из этих состояний работа автомата прекращается); Σ – допустимый входной алфавит (конечное множество допустимых входных символов), из которого формируются строки, считываемые автоматом; δ – функция переходов автомата.

Автомат начинает работу в состоянии q_0, считывая по одному символу входной строки. Считанный символ переводит автомат в новое состояние из Q в соответствии с функцией переходов. Существуют и другие модели для описания конечных автоматов, наиболее наглядным является представление в виде графов переходов (диаграмм состояний).

Веб-приложение является сетью связанных между собой страниц и взаимодействующих с ними элементов ПО. Зададимся целью описать внутренние связи в приложении, при этом будем считаем, что у пользователя нет возможности осуществить прямые переходы. Таким образом, рассматриваются связи видов 1, 2, 3, 4, 6 приведенной выше классификации.

Большинство веб-приложений являются диалоговыми [5]. Сначала приложение находится в начальном состоянии. В процессе работы приложение ожидает в текущем состоянии наступления некоторого события, например, запроса пользователя. В зависимости от типа наступившего события приложение переходит в другое состояние. Диалог завершается при достижении финального состояния.

Обозначим состояниями страницы приложения и последовательно вызываемые части ПО, тогда последовательность работы веб-приложения можно описать в терминах конечного автомата. Для приведенного выше примера веб-приложения обозначим страницы приложения состояниями автомата: страницу select.html – состоянием $q0$, insertForm.html – $q1$, doInsert.php – $q2$, viewRecords.php – $q3$, а внутренние связи обозначим символами входного алфавита C1–C7 соответственно нумерации связей. Связь типа 3 (связывающая исполняемый код и код представления в файле viewRecords.php) мы не рассматриваем, соответственно, рассматриваем лишь связи типов 1, 2, 4, 6.

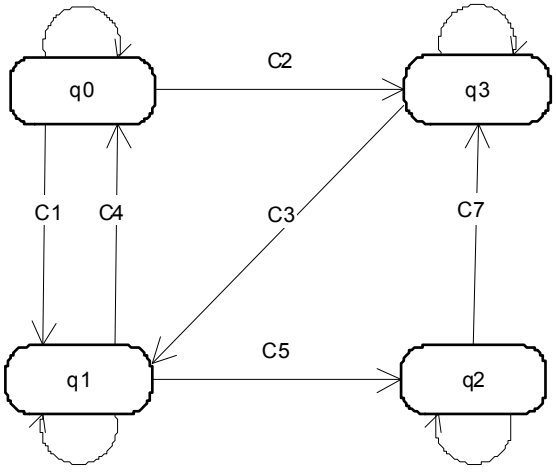

Рис. 2. Граф конечного автомата

Граф полученного автомата приведен на рис. 2. Петли у состояний показывают, что для всех входных сигналов, не указанных явно на графе, происходит возврат в то же самое состояние.

Чтобы учесть связь типа 3 в нашем примере, нужно разделить состояние ViewRecords на два состояния – $q4$, при переходе в которое происходит извлечение информации из базы данных, и $q3$, при переходе в которое происходит отображение полученных данных. Граф полученного автомата изображен на рис. 3.

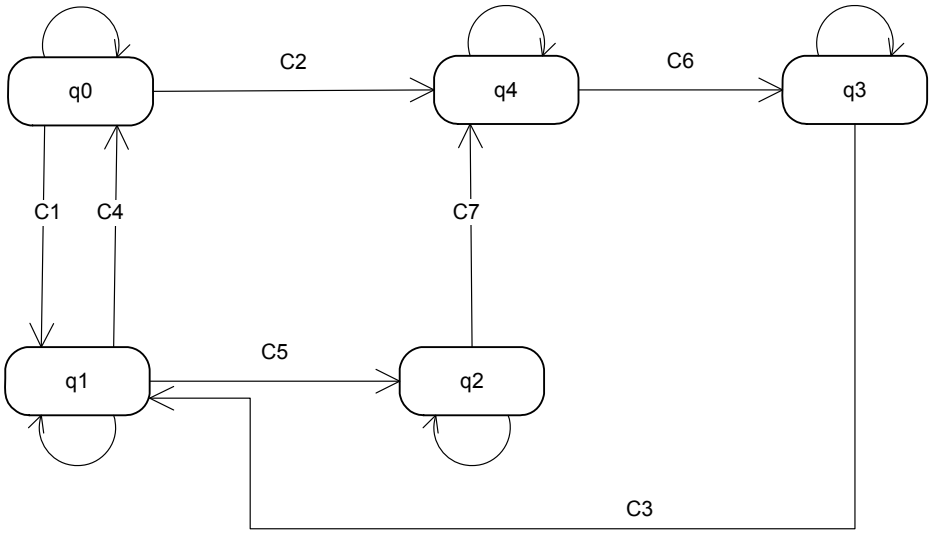

Рис. 3. Граф конечного автомата при разделении состояний

Проведем анализ полученного нами в предыдущем разделе конечного автомата (возьмем второй вариант реализации, граф конечного автомата которого представленный на рис. 3).

Автомат является детерминированным, так как из любого своего состояния по поступлению любого входного символа он может перейти лишь в одно состояние, а не во множество (состояния меняются одно за другим последовательно). Также данный конечный автомат, согласно [6], является автоматом без выходов и может быть описан четверкой $<Q, \sum, q0, \delta>$, где $Q=\{q0, q1, q2, q3, q4\}$– множество внутренних состояний автомата, $\sum=\{C1, C2, C3, C4, C5, C6, C7\}$ – множество входных символов, $q0$ – начальное состояние автомата, δ – функция переходов. Для рассматриваемого примера она задана в таблице.

	C1	C2	C3	C4	C5	C6	C7
$q0$	$q1$	$q4$	$q0$	$q0$	$q0$	$q0$	$q0$
$q4$	$q4$	$q4$	$q4$	$q4$	$q4$	$q3$	$q4$
$q3$	$q3$	$q3$	$q1$	$q3$	$q3$	$q3$	$q3$
$q1$	$q1$	$q1$	$q1$	$q0$	$q2$	$q1$	$q1$
$q2$	$q2$	$q2$	$q2$	$q2$	$q2$	$q2$	$q4$

Таблица. Функция переходов конечного автомата

Заключение

В работе рассмотрена классификация связей между составными частями веб-приложения и их описание с использованием модели конечного автомата. Выделено восемь типов связей между частями веб-приложений, произведена их классификация.

Разработан способ описания модели структуры веб-приложения как детерминированного конечного автомата.

Применение модели конечных автоматов позволяет тестировать веб-приложение без необходимости знания деталей его реализации. Кроме того, при тестировании приложения, описанного в терминах конечного автомата, становится возможным использовать соответствующие методы тестирования, разрабатываемые с конца 1970-х г.г.

Литература

1. Anneliese Amschler Andrews, Jeff Offutt, Roger T. Alexander. Testing Web Applications by Modeling with FSMs // Software and System Modeling. – 2005. – № 4(3). – P. 326–345.
2. P. Merialdo, P. Atzeni, and G. Mecca. Design and development of data-intensive web sites: The Araneus approach // ACM Transactions on Internet Technology. – 2003. – № 3(1). – P. 49–92.
3. Sergei Kojarski, David H. Lorenz. Domain Driven Web Development With WebJinn // OOPSLA, Anaheim, CA, October 26–30, 2003.
4. Хопкрофт Дж. Э., Мотвани Р., Ульман Дж. Д. Введение в теорию автоматов, языков и вычислений. – 2-е изд. – М.: Вильямс, 2002.
5. Konstantin Läufer. Interactive Web Applications Based on Finite State Machines. Invited paper // Proc. Symp. Information Systems Analysis and Synthesis (ISAS), Baden-Baden, Germany, August 1995.
6. Трахтенброт Б.А., Бардзинь Я.М. Конечные автоматы. Поведение и синтез. – М.: Наука, 1970.

Ожиганов Александр Аркадьевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, ojiganov@mail.ifmo.ru

Чепурной Александр Иванович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, alexch@bk.ru