Уфа : УГАТУ, 2010 Вестник УГАТУ Т. 14, № 1 (36). С. 154–163

УДК 004.7

В. В. МИРОНОВ, К. Э. МАЛИКОВА

ИНТЕРНЕТ-ПРИЛОЖЕНИЯ НА ОСНОВЕ ВСТРОЕННЫХ ДИНАМИЧЕСКИХ МОДЕЛЕЙ: АРХИТЕКТУРА, СТРУКТУРА ДАННЫХ, ИНТЕРПРЕТАЦИЯ

Обсуждается класс интернет-приложений на основе встроенных динамических моделей. Рассматриваются вопросы разработки архитектурной модели данного класса приложений, определяется организация взаимодействия ключевых архитектурных компонентов и их функциональность. Описывается программная реализация алгоритмов интерпретации встроенной динамической модели. *Интернет-приложение; динамическая модель; встроенная модель; интерпретация динамических моделей; XML-технологии*

ВВЕДЕНИЕ

В работе [1] авторами предложен новый класс интернет-приложений – на основе встроенных динамических моделей. На концептуальном уровне обсуждались вопросы построения таких приложений, однако оставались открытыми вопросы выбора архитектурной модели, которая должна обеспечить производительность, масштабируемость и надежность предложенного решения. Данная статья является продолжением упомянутой работы в плане архитектуры, структуры данных и интерпретации динамической модели, управляющей функционированием интернет-приложения.

Построение архитектурной модели сводится к выбору основных функциональных компонентов системы, архитектурных решений на каждом из уровней проектируемого интернет-приложения, где центральное место занимают процессы интерпретации. Правильное разбиение объектной системы предметной области в процессе построения интернет-приложений является важной задачей и в значительной степени зависит от выбранной архитектуры и природы объектов, которые могут выполняться на серверной, на клиентской или на обеих сторонах. Постоянные, совместно используемые, а также связанные с серверными ресурсами объекты принадлежат серверному слою. Объекты проверки корректности данных, различные управляющие компоненты пользовательского интерфейса, а также навигационные элементы являются объектами-кандидатами для размещения в контексте клиентской стороны. При разработке интернет-приложений данного класса предполагается разделять бизнес-логику взаимодействия клиентской и серверной частей, чтобы в условиях низкоскоростных каналов и лимитированных ресурсов серверов достичь оптимальной производительности.

В основу рассматриваемого класса интернет-приложений положена трехуровневая архитектура клиент-серверного взаимодействия, предполагающая разделение обработки на три логически различимых сегмента: представления, бизнес-логики и хранения данных. На уровне представления данных необходимо обеспечить взаимодействие пользователя с интернет-приложением. Предоставить клиенту ряд возможностей по проверке введенных данных, выполнения URL-запросов, просмотра результатов запросов и управления ими после того, как они попадают на клиентскую сторону. На уровне бизнес-логики необходимо определить совокупность функциональных компонентов, правил, принципов их интерпретации и зависимостей их поведения. Службы хранения данных обеспечиваются различными структурированными и неструктурированными информационными хранилищами, вспомогательными инструментами, которые обеспечивают доступ к необходимым данным из различных областей приложения.

В работе использованы результаты применения встроенной динамической модели при построении электронных документов, описанные в работах [2–4].

Контактная информация: (347) 272-89-81

Работа выполнена в рамках научной школы УГАТУ «Теория и практика разработки информационных систем» и поддержана грантом РФФИ 10-07-00167-а «Электронные документы со встроенной динамической моделью»

1. АРХИТЕКТУРА ИНТЕРНЕТ-ПРИЛОЖЕНИЯ

В соответствии с концепцией, предложенной в работе [1], на рис. 1 приведена укрупненная архитектурная модель для данного класса интернет-приложений, которая строится вокруг обязательного набора функциональных компонентов, необходимого для ее успешной реализации.

Общие положения. Рассмотрим три ключевых компонента архитектурной модели с целью отражения их функциональности и организации взаимодействия. Как обычно, архитектура включает три подсистемы:

- клиент, представляющий собой веб-браузер, обслуживающий передачу URL-запросов на получение ресурсов веб-сервера и отображение полученной HTML-страницы;
- веб-сервер, обеспечивающий обработку запросов клиентов и формирование ответов на основе исполнения сценариев и запросов к серверным базам данных;
- серверные базы данных, содержащие данные, хранящиеся на стороне сервера.

Интернет-приложение представляет собой серверную страницу, размещенную на веб-сервере, запускаемую на исполнение по соответствующему запросу клиента. Вместе с запросом серверной странице передаются специфицирующие параметры POST. В качестве ответа серверная страница формирует HTML-код, возвращаемый клиенту.

Приложение обеспечивает сеансовую (сессионную) поддержку, т. е. возможность хранить индивидуальные данные каждого пользователя между последовательными запусками страницы клиентом. Это достигается путем генерации уникальных идентификаторов сеанса (SID[1]), сохранения их между запросами на стороне клиента и извлечения с их помощью сеансовых данных, сохраненных в серверной базе данных.

Отличительные особенности. Особенность данного класса интернет-приложений состоит в том, что его работа управляется встроенной динамической моделью, хранящейся на стороне сервера. В серверной базе данных размещена общая для всех пользователей динамическая модель, управляющая работой интернет-приложения, а также экземпляры памяти текущего состояния динамической модели для различных пользователей.

Архитектура поддерживает как зарегистрированных, так и незарегистрированных пользователей. Для зарегистрированных пользователей в серверной базе данных предусмотрено сохранение экземпляров памяти текущего состояния динамической модели средствами сценария серверной страницы. Для незарегистрированных (анонимных) пользователей память текущего состояния организована в форме сессионных переменных.

Для поддержания указанных особенностей архитектура серверной страницы интернет-приложения дополняется новыми компонентами.

1) В составе серверной базы данных, сохраняющей информацию между отдельными запросами к интернет-приложению:

- Исходная динамическая модель HSM, общая для всех пользователей (клиентов).

[1] SID (Session Identificator) – присвоенная при старте сессии структура данных переменной длины, которая идентифицирует учетную запись пользователя, группы, домена или компьютера

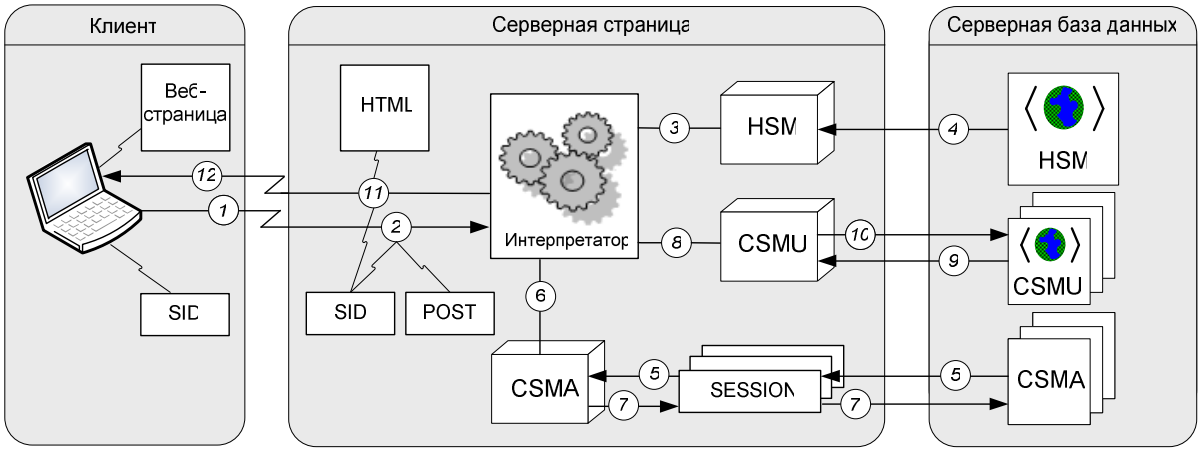

Рис. 1. Общая архитектурная модель интернет-приложений на основе встроенной динамической модели

● Экземпляры памяти текущего состояния пользователей CSMU (Current State Model / User), сохраняющие сведения о текущем состоянии динамической модели для каждого из зарегистрированных (т. е. известных приложению) пользователей, прошедших процедуры аутентификации, между запросами к приложению. Экземпляры CSMU создаются, обновляются, удаляются средствами интернет-приложения, использующего возможности серверных сценариев по взаимодействию с серверными базами данных.

● Экземпляры памяти текущего состояния для анонимных пользователей CSMA (Current State Model /Anonymous), сохраняющие сведения о текущем состоянии динамической модели для неаутентифицированных пользователей в пределах сеанса использования интернет-приложения. Экземпляры CSMA поддерживаются интернет-приложением с помощью стандартных сеансовых (сессионных) механизмов, предоставляемых платформой реализации интернет-приложения (PHP, ASP, ASP.NET и др.), которые обеспечивают их хранение в течение сеанса в серверной базе данных.

2) В составе серверной страницы, реализующей сценарий интернет-приложения:

● Интерпретатор динамической модели, обеспечивающий функциональность интернет-приложения путем обработки динамической модели с учетом ее текущего состояния и параметров, полученных в запросе пользователя.

● Программные объекты (DOM[2]-объекты), в которые из серверной базы данных загружаются динамическая модель и экземпляры памяти текущего состояния зарегистрированных и анонимных пользователей (CSMA, CSMU) и которые обрабатываются интерпретатором.

● Системные программные объекты, поддерживаемые платформой реализации интернет-приложения: POST – метод HTTP-протокола для передачи пользовательских данных; SESSION – суперглобальный массив для доступа к переменным сессии; SID – уникальное значение, позволяющее соотнести клиента с переменными сеанса.

3) На клиентской стороне, обращающейся с запросами к интернет-приложению, используется стандартный механизм поддержания сеанса в виде сохраняемого между запросами идентификатора SID.

Данный класс интернет-приложений является изначально многопользовательским, URL-запросы на обслуживание от различных пользователей могут приходить в любом порядке и должны обрабатываться одновременно. Обеспечение многопользовательского режима, при котором с одним приложением могут взаимодействовать несколько клиентов, достигается за счет того, что для каждого пользователя сохраняется свой экземпляр памяти текущих состояний. При этом для аутентифицированных пользователей хранение экземпляров памяти текущих состояний обеспечивается в серверной базе данных, а для анонимных – в сеансовых переменных.

В этой связи для каждого анонимного пользователя необходимо предусмотреть механизмы, которые позволили бы серверной стороне однозначно идентифицировать пользователя и отслеживать именно его текущие состояния, для дальнейшей интерпретации динамической модели при его обращении к интернет-приложению в рамках сеанса.

Для решения данной задачи на начальном шаге клиент-серверного взаимодействия выполняется присвоение каждой клиентской стороне уникального сессионного идентификатора – SID с дальнейшим его размещением в виде файла на стороне клиента. На серверной стороне создается ассоциированное с данным идентификатором временное хранилище с пустым экземпляром модели текущих состояний.

Взаимодействие клиентов. В рамках рассматриваемой архитектурной модели взаимодействие основных компонентов интернет-приложения начинается с отправки клиентом на сервер URL-запроса (рис. 1, *1*). Из запроса извлекается присоединенная информация (*2*).

Работа приложения управляется встроенной динамической моделью, поэтому действия интерпретатора для всех групп пользователей начинается с создания DOM-объекта (*3*), в который загружается динамическая модель из серверной базы данных (*4*).

Платформа интернет-приложения поддерживает сеанс взаимодействия, в рамках которого интерпретатору предоставляется доступ к сеансовым переменным SESSION, которые загружаются из серверной базы данных (*5*) на основе идентификатора сессии SID, полученного в запросе клиента. Интерпретатор создает DOM-объект CSMA, куда загружает из сеан-

[2] DOM (Document Object Model) — не зависящий от платформы и языка программный интерфейс, позволяющий получить доступ к содержимому XML-документов

совых переменных экземпляр ПТС анонимного пользователя (*6*). По завершении обработки запроса экземпляр CSMA сохраняется в сеансовых переменных (*7*), которые, в свою очередь, сохраняются в серверной базе данных (*7*).

В случае, если пользователь аутентифицирован как зарегистрированный, интерпретатор создает DOM-объект CSMU (*8*), в который загружает экземпляр ПТС зарегистрированного пользователя (*9*). По завершении обработки запроса обновленный экземпляр ПТС сохраняется в серверной базе данных (*10*).

Интерпретатор выполняет рекурсивный обход динамической модели HSM, основываясь на информации о текущих состояниях из CSMA и CSMU. В процессе обхода формируется результирующий HTML-код путем сборки фрагментов кода, ассоциированных с состояниями динамической модели. По завершении обработки результирующий HTML-код отправляется клиенту (*11*), обеспечивая формулирование ответа в окне браузера (*12*).

Таким образом, рассмотренная архитектура интернет-приложения, как и известные, основана на 3-слойной схеме «клиент-сервер-база данных» и стандартных механизмах поддержания сеансов и передачи параметров в URL-запросах. Архитектура отличается тем, что в серверной базе данных предусмотрено размещение динамической модели и экземпляров ПТС зарегистрированных пользователей, а в сеансовых переменных – экземпляров ПТС анонимных пользователей, а на серверной странице предусмотрено создание соответствующих DOM-объектов, которые загружаются из серверной базы данных и обрабатываются интерпретатором динамической модели.

2. СТРУКТУРА ДАННЫХ

В соответствии с концепцией [1] уровень хранения включает компоненты, принадлежащие серверному слою, для которых необходимо специфицировать структуру данных:

- исходная динамическая модель;
- модель памяти текущих состояний.

Динамическая модель обеспечивает централизованное хранение ассоциированных управляющих и прикладных фрагментов и нивелирует необходимость выполнения операций по их синхронизации. Конечные пользователи, работающие с приложением через интернет, не должны иметь возможностей модификации данной модели. Предполагается, что такие действия составляют прерогативу администратора или ответственного за разработку динамической модели.

Концептуальная схема (рис. 2) отражает допустимые типы объектов, которые могут содержаться внутри состояния модели – контейнера внутренних объектов. При XML-реализации эта схема задает допустимую структуру вложенных элементов модели. На схеме круг символизирует состояние, пятиугольник – субмодель, треугольник – переход состояния, пятиугольник с вертикальной чертой – область погружения, прямоугольник – прикладное действие; тип соответствующего объекта задается обязательным атрибутом type. Вложенность субмоделей, переходов и прикладных действий внутри контейнера состояния отражается присоединением их ниже символа родителя. Двойная линия означает произвольную последовательность экземпляров разнотипных объектов. Светлый концевой треугольник соединительной линии означает множественность, т. е. возможность 0, 1 или нескольких экземпляров объектов данного типа. Атрибуты объектов связаны с символами объектов соединительными линиями, причем светлый концевой кружок означает необязательный атрибут, а темный – обязательный.

Таким образом, схема (рис. 2, *а*) означает, что каждый элемент динамической модели может содержать внутри себя набор элементов субмоделей, переходов, действий, следующих в произвольном порядке.

Количество уровней логического разбиения исходной модели ограниченно. Наличие субмоделей в составе состояний обуславливает многоуровневую иерархию встроенной динамической модели. Субмодели, входящие в состояние верхнего уровня, могут содержать состояния, субмодели которых, в свою очередь, содержат другие состояния, и т. д. Внутренняя структура субмодели как совокупности одного или нескольких внутренних состояний задана на рис. 2, *б*.

В этой связи разработчик динамической модели может добавлять столько уровней вложенности, сколько считает оправданным для построения интернет-приложения – последовательно интегрируя дополнительные подмодели, состояния и ассоциированные прикладные фрагменты-действия. Благодаря объектной модели XML, обеспечивающей универсальный формат для хранения данных любого уровня сложности, можно оперировать встроенной моделью как единым программным объектом, не зависящим от программного кода,

применяя к нему операции создания, редактирования и преобразования.

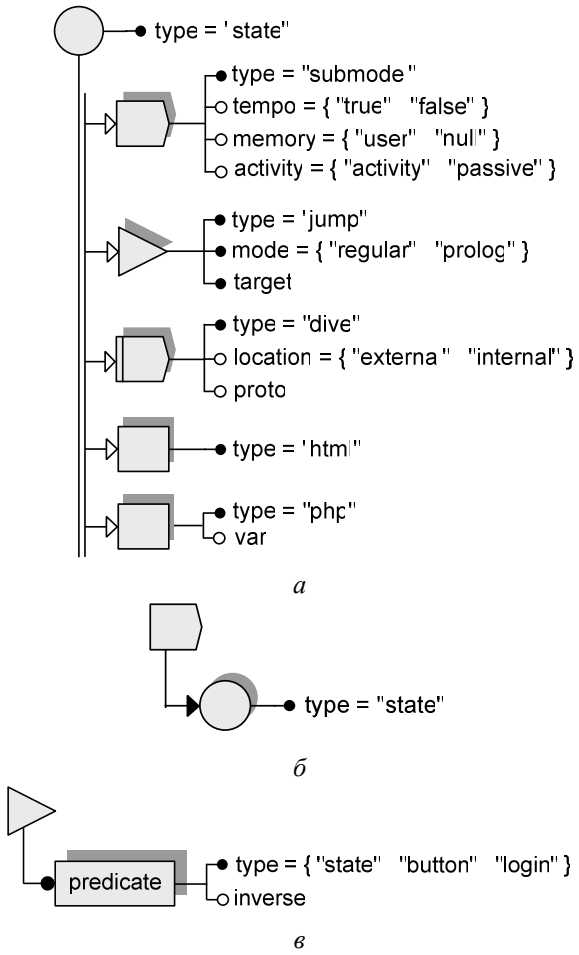

Рис. 2. Концептуальная схема встроенной динамической модели: *а* – структура состояния; *б* – структура субмодели; *в* – структура перехода

Субмодель. Каждый из элементов множества субмоделей содержит обязательный атрибут type и необязательные атрибуты: tempo, activity и memory. Атрибут activity необходим для ускорения обхода модели DOM-средствами посредством пропуска субмоделей, значение данного атрибута которых принимает истинное значение. В связи с тем, что встроенная динамическая модель изначально является многопользовательской, то в ней могут содержаться подмодели доступные как для анонимных, так и для аутентифицированных пользовательских групп. Для определения возможностей доступа к содержимому подмодели введен атрибут memory. Если субмодель содержит данный атрибут и он принимает значение user, то данная субмодель доступна только аутентифицированным пользователям. Следующий необязательный атрибут tempo указывает на вариант ведения пользовательских моделей памяти текущих состояний. В случае присвоения данному атрибуту значения true, модель памяти текущих состояний будет являться темпоральной, что позволит просмотреть предысторию всех состояний, которые когда-либо были текущими.

Переход. За исключением состояний начальных субмоделей, которые становятся текущими после выполнении погружения в динамическую модель, все состояния связаны друг с другом при помощи переходов. Срабатывание перехода связано с выполнением входящих в его состав предикатов. Выполнение предиката означает, что некоторое выражение, которое он задает, истинно. В соответствии с типом такого выражения множество предикатов делится на следующие непересекающиеся подмножества:

- state – предикат, проверяющий условие того, что некоторое состояние является текущим;
- button – предикат, проверяющий условие того, что данные были отправлены обработчику методом POST;
- login – предикат, проверяющий условие того, что зарегистрированный пользователь успешно прошел аутентификационную проверку.

Области погружения. Нередко интернет-приложение включает различные группы функций, каждая из которых является частью общей функциональности. В этой связи группы функций, выполняющие определенную работу, целесообразно выделить из общей встроенной модели в отдельные структурированные области (dive), в которые при необходимости выполнять погружения. Данные областей погружения могут располагаться как внутри встроенной модели (атрибут location принимает значение internal), так и в расположенном в некотором внешнем файле (атрибут location принимает значение external), месторасположение которого указывается в значении атрибута proto.

Отдельные области для хранения исходного программного кода позволяют:

- независимо работать с различными частями интернет-приложения;
- разделять и многократно использовать ресурсы проекта;
- создавать различные модификации модулей без переработки всего интернет-приложения в целом;

● использовать исходные файлы меньшего размера, более удобные в редактировании.

Необходимо отметить, что разделение приложения на отдельные модули на этапе проектирования и разработки не означает, что конечный проект будет состоять из множества файлов. После завершения разработки интересующие функции при необходимости могут быть собраны в единую файловую область.

Прикладные фрагменты действия. Согласно концепции [1] с каждым состоянием встроенной модели ассоциированы один или несколько прикладных фрагментов, основное содержание которых составляет программный код для формирования разметки изображения. В этой связи необходимо предусмотреть возможности последовательной обработки каждого фрагмента, ассоциированного с той или иной ситуацией. В качестве прикладных представлены две группы фрагментов-действий: текстовые, содержащие программный код на языке HTML и/или фрагменты, содержанием которых являются различные участки сценариев скриптовых языков программирования, применяющихся для разработки интернет-приложений.

Для определения правил интерпретации и нивелирования противоречий между двумя группами прикладных фрагментов атрибут type принимает следующие значения: «html» – в случае если в качестве содержимого прикладного фрагмента принимается HTML-разметка, или «php», если в качестве содержания принимаются различные участки программмного кода.

По умолчанию интерпретатор проверяет на предмет поиска прикладных фрагментов только состояния, игнорируя внутреннее содержание субмоделей. Поскольку содержательную часть фрагментов-действий могут составлять теги разметки, а также различные символьные данные, то между управляющими элементами встроенной модели и содержанием прикладных фрагментов могут возникнуть конфликты, которых необходимо избежать.

Для устранения конфликтных ситуаций такого рода возможны следующие варианты действий. Содержимое прикладных фрагментов действия оформляется как раздел CDATA[3], что указывает интерпретатору на то, что выделенный фрагмент следует рассматривать как неразмеченный символьный текст независимо от того, какие символы в нем встречаются.

Текстовое содержимое элемента оформляется в соответствии со спецификацией XHTML[4].

Рассмотрим подробно каждый вариант. В первом случае раздел CDATA начинается со следующей последовательности символов: <![CDATA[и заканчивается с первым появлением последовательности:]]>. Все символы, заключенные между данными двумя последовательностями, интерпретируются как символы, а не как разметка или ссылки на объект. В качестве недостатка рассматриваемого варианта можно привести следующую особенность. Чтобы получить доступ к искомому текстовому содержимому, необходимо использовать дополнительные свойства DOM-интерфейса, позволяющие получать из узла все содержимое, включая разметку, начиная от начального и заканчивая закрывающим тегами элемента. Что касается второго случая, спецификация XHTML была создана для устранения разрывов между HTML и XML и представляет собой обычный HTML, записанный в соответствии с синтаксическими правилами XML. Для того чтобы содержимое прикладного фрагмента было XML-совместимым, необходимо придерживаться следующих основных синтаксических дисциплин XHTML. Прежде всего, не разрешены непарные теги. Следовательно, все начинающиеся теги должны иметь соответствующие закрывающиеся. Все значения атрибутов должны заключаться в кавычки. Также XHTML, соответствуя XML, чувствителен к регистру.

Рассмотрим основные преимущества данного варианта. Во-первых, большинство веббраузеров адекватно прорисовывают большинство XHTML-документов. Во-вторых, содержимое ассоциированных фрагментов будет являться самостоятельными XML-элементами, тем самым открывается возможность выполнения их напрямую интерпретатором без обращения к дополнительным свойствам, обеспечивающим доступ к значению искомого фрагмента.

Если в качестве содержания прикладного фрагмента принимаются различные участки программного кода, то дополнительно выпол-

[3] CDATA – специальная форма задания текстовых значений, которая допускает использование зарезервированных символов

[4] XHTML (Extensible Hypertext Markup Language) – расширяемый язык разметки интернет-страниц, сопоставимый с HTML, удовлетворяющий стандартам XML

няется проверка на предмет наличия атрибута var. Содержанием данного атрибута является массив переменных, значения которых необходимо получить. Данный массив формируется путем разделения переменных по границам, образованным сепаратором (в рассматриваемом случае в качестве сепаратора используется пробел).

Пользовательские модели памяти текущего состояния. Обработка встроенной динамической модели основана на отслеживании пользовательских экземпляров памяти текущих состояний. Данный элемент строится для каждого пользователя интернет-приложения при первой загрузке динамической модели и обеспечивает хранение информация обо всех пройденных состояниях и их источниках. В этой связи данный объект является также объектом-кандидатом для размещения в контексте серверной стороны.

На рис. 3 изображена концептуальная схема модели памяти текущего состояния, в отличие от динамической модели данный компонент демонстрирует динамическое поведение, поскольку его структурные составляющие и содержимое корректируются в процессе работы пользователя с интернет-приложением.

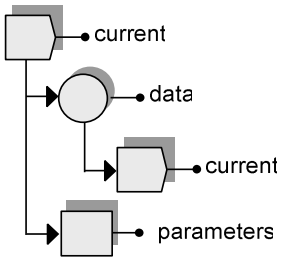

Рис. 3. Концептуальная схема модели памяти текущего состояния

Благодаря информационному содержанию экземпляров моделей памяти текущих состояний предоставляется возможность отслеживать пользовательские действия, а также осуществлять сбор статистической информации посредством определения последовательности интернет-страниц, просмотренных в процессе сеанса до достижения желаемого результата.

Вследствие этого открывается возможность осуществлять построение достаточно полной картины предпочтений как конкретного пользователя, так и определенной пользовательской группы. В этой связи в процессе работы пользователя с интернет-приложением предполагается предоставлять необходимые процедуры для создания, хранения, использования и удаления содержимого данных пользовательских моделей. Структурное содержание пользовательских моделей текущих состояний повторяет структуру встроенной динамической модели за одним лишь исключением. Корневым элементом модели памяти текущего состояния является корневой элемент встроенной подмодели. Смена текущего состояния осуществляется при активизации некоторого перехода по выбору пользователя или при срабатывании соответствующего предиката. При этом текущим может являться не одно состояние, а их совокупность, с учетом всех текущих состояний в подмоделях данного текущего состояния. Кроме того, разметка пользовательской модели дополнена несколькими дополнительными атрибутами, раскрывающими свойства модели. Поскольку объекты данного серверного компонента динамически формируются в процессе использования пользователем интернет-приложения, то необходимо фиксировать дату и время их занесения в модель. Для этого используется атрибут data.

3. ИНТЕРПРЕТАЦИЯ ВСТРОЕННОЙ ДИНАМИЧЕСКОЙ МОДЕЛИ

В основе поведения предложенной архитектурной модели (рис. 1) лежит принцип: работа интернет-приложений управляется встроенной динамической моделью, размещенной на стороне сервера. В этой связи процесс интерпретации встроенной модели взаимодействия с клиентом является в большей части серверным и ориентирован на двухпроходный обход модели (рис. 4, *а*).

Работа с содержимым динамической модели и моделями текущих состояний начинается с создания экземпляров DOM-объектов, на основании которых осуществляется представление моделей в целом (*1* на рис. 4, *а*). Заданный серверным сценарием набор функций позволяет загрузить в DOM-объект экземпляр встроенной динамической модели для осуществления дальнейшего обхода желаемым образом, открывая доступ к элементам и значениям атрибутов модели в процессе интерпретации (*2* на рис. 4, *а*).

Процесс интерпретации динамической модели предполагает выполнение двух последовательных этапов. На первом этапе (первый проход) осуществляется контроль пользовательского текущего состояния (*3* на рис. 4, *а*) на основании обработки элементов управляющих типов. На втором этапе (второй проход) выполняется сборка прикладных фрагментов

и формирование результирующего контента, соответствующего HTML-спецификации (*4* на рис. 4, *а*). Чтобы обеспечить сохранение созданной на предыдущих сеансовых шагах пользовательской модели текущих состояний в рамках сессии на заключительном этапе выполняется сериализация содержимого пользовательской модели в сеансовую переменную (*5* на рис. 4, *а*). При последующих обращениях к интернет-приложению в рамках данного сеанса содержимое сеансовой переменной десериализуется в DOM-объект до начала первого интерпретационного прохода. Важным аспектом в процессе интерпретации динамической модели является процесс ведения пользовательских моделей памяти текущих состояний. Рассмотрим более подробно первый проход интерпретации динамической модели, а именно – контроль модели памяти текущего состояния (рис. 4, *б*).

На первом проходе запускается механизм обработки ассоциированных с состоянием дочерних элементов в соответствии с принятой логикой алгоритма (*1* на рис. 4, *б*). Алгоритм состоит в следующем. Вначале осуществляется проверка типа управляющего дочернего элемента. Если в качестве управляющего элемента представлена субмодель, то осуществляется формирование модели текущих состояний. В данном классе интернет-приложений предполагается поддержка двух вариантов ведения моделей текущих состояний – для анонимных и аутентифицированных пользователей. Если с интернет-приложением взаимодействует пользователь, успешно прошедший аутентификационные процедуры, то выполняется проверка наличия в серверной базе данных его модели текущих состояний – CSMU. Если пользовательская модель найдена, осуществляется переход (*2* на рис. 4, *б*) и дальнейшая обработка данной модели (*3* на рис. 4, *б*). В случае если пользовательская модель не была обнаружена, то осуществляется ее создание с дополнением соответствующими субмоделями и их начальными состояниями.

Для анонимных пользователей загрузка модели текущих состояний выполняется при помощи процедуры десериализации модели в DOM-объект из сессионной переменной. Согласно структуре встроенной модели состояние может включать погружения в одну или несколько субмоделей. Поэтому вносимое в пользовательскую модель состояние проверяется на наличие субмоделей. Если данные субмодели существуют, то происходит рекурсивный вызов алгоритма (*1* на рис. 4, *б*), обеспечивающий дальнейшее погружение в субмодели и занесение их состояний в пользовательскую модель. Если при обработке динамической модели интерпретатор обнаруживает активный переход (*5* на рис. 4, *б*), то выполняется перевод модели в новое текущее состояние в рамках той субмодели, которой принадлежит выполненный переход. Если активность перехода не установлена, то продолжается дальнейшее погружение (*1* на рис. 4, *б*). Дочерний элемент также может быть представлен внешней или внутренней областью погружения (*6* на рис. 4, *б*), в этом случае происходит рекурсивный вызов алгоритма, выполняющего погружение в субмодель, соответствующую заданной области (*1* на рис. 4, *б*).

В качестве результата первого прохода ожидается сформированная модель текущих состояний. Экземпляр данной модели сохраняется в базу данных для аутентифицированных пользователей (*4* на рис. 4, *б*) и в сессионный массив для анонимных.

На втором проходе интерпретации предусмотрена возможность последовательной обработки содержимого каждого прикладного фрагмента-действия, ассоциированного с той или иной ситуацией, с дальнейшим помещением результата в буфер результата. Если в качестве результата получен ассоциированный с некоторым состоянием HTML-фрагмент, то выполняется его интерпретация. Несколько сложнее дело обстоит с фрагментами, содержанием которых являются различные участки сценариев скриптовых языков программирования. Для прикладных фрагментов такого типа необходимо не только извлечь содержимое, но и запустить сценарий на выполнение. В этой связи для выполнения ассоциированных фрагментов, содержанием которых является правильный программный код, используются методы, которые осуществляют выполнение содержимого прикладных фрагментов и возвращают необходимое значение во время выполнения основного кода интерпретатором.

На заключительном этапе содержимое прикладных фрагментов возвращается клиенту, в результате чего браузер формирует соответствующее текущему состоянию изображение.

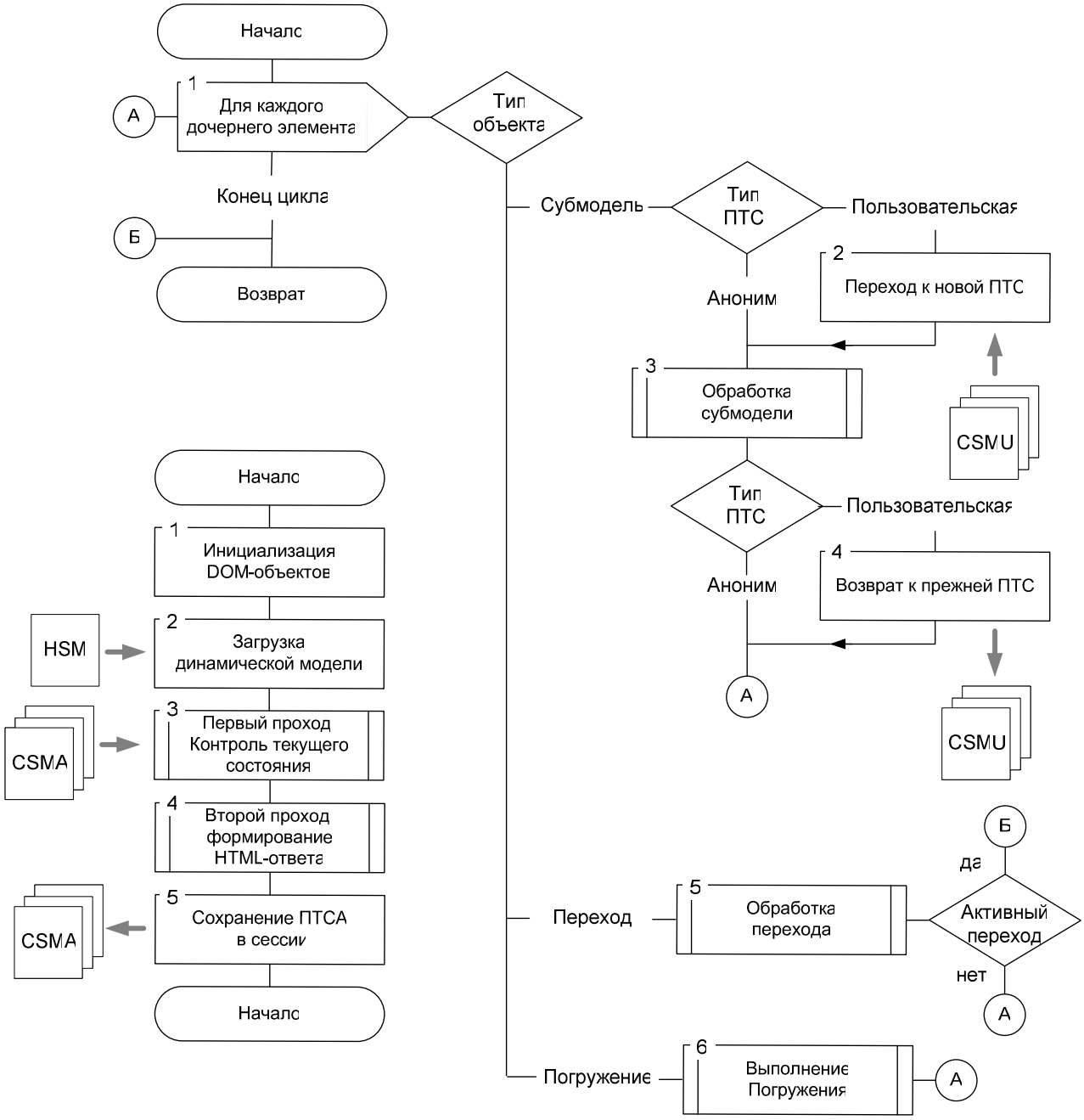

Рис. 4. Схема обработки встроенной динамической модели на основании двухпроходной интерпретации: *а* – обобщенная схема процесса двухпроходной интерпретации; *б* – укрупненная схема интерпретации динамической модели на втором проходе модели

Таким образом, предлагаемый интерпретатор динамической модели для интернет-приложения, как и известные, выполняет рекурсивный обход «сверху вниз» дерева динамической модели по текущим состояниям с коррекцией текущих состояний, сохраняемых в ПТС. Интерпретатор отличается тем, что с целью обеспечения функциональности интернет-приложения в нем предусмотрено два прохода интерпретации, причем на втором проходе выполняется сборка результата из HTML-фрагментов, ассоциированных с текущими состояниями динамической модели, а с целью обеспечения многопользовательской обработки – загрузка и подключение экземпляров ПТС зарегистрированных пользователей в состояниях их успешной аутентификации.

ЗАКЛЮЧЕНИЕ

В отличие от традиционных интернет-приложений, интернет-приложения на основе встроенных динамических моделей характеризуются следующими особенностями:

- работа данного класса интернет-приложений управляется встроенной динамической моделью, размещенной на стороне серверной базы данных;
- для каждого типа элементов встроенной динамической модели: состояний, переходов, областей погружения, прикладных действий, вводятся собственные правила интерпретации, которые объединяются в единый интерпретационный алгоритм;
- непосредственная обработка встроенной динамической модели в процессе работы пользователя с интернет-приложением основана на отслеживании экземпляров памяти текущих состояний в многопользовательском режиме;
- для неаутентифицированных (анонимных) пользователей модель памяти текущего состояния (CSMA) сохраняется в формате сессионных переменных в рамках сеанса;
- для зарегистрированных пользователей предусмотрено сохранение экземпляров модели текущего состояния (CSMU) на стороне серверной базы данных методами сценария серверной страницы;
- формирование результирующего кода интернет-страницы осуществляется путем двухпроходной интерпретации динамической модели. На первом проходе интерпретатор контролирует переходы текущего состояния, на втором выполняет сборку прикладных фрагментов.

СПИСОК ЛИТЕРАТУРЫ

1. **Миронов В. В., Маликова К. Э.** Интернет-приложения на основе встроенных динамических моделей: идея, концепция, безопасность // Вестник УГАТУ: научн. журн. Уфимск. гос. авиац. техн. ун-та. 2009. Т. 13, № 2. С. 167–179.

2. **Миронов В. В., Шакирова Г. Р.** Интерпретация XML-документов со встроенной динамической моделью // Там же. 2007. Т. 9, № 2. С. 88–97.

3. **Миронов В. В., Шакирова Г. Р.** Программно-инструментальное средство для создания и ведения динамических XML-документов // Там же. 2007. Т. 9, № 5. С. 54–63.

4. **Миронов В. В., Шакирова Г. Р.** Обработка XML-документов со встроенной моделью // Обозрение прикладной и промышленной математики. 2008. Т. 15, № 2. С. 336–336.

ОБ АВТОРАХ

Миронов Валерий Викторович, проф. каф. автоматизир. систем упр-я. Дипл. радиофизик (Воронежск. гос. ун-т, 1975). Д-р техн. наук по упр. в техн. системах (УГАТУ, 1995). Иссл. в обл. иерарх. моделей и ситуац. управления.

Маликова Карина Эмильевна, асп., асс. той же каф. Дипл. информатик-экономист (УГАТУ, 2007). Готовит дис. в обл. использования встроенных моделей в интернет-приложениях.