Уфа : УГАТУ, 2009 Вестник УГАТУ • Управление, ВТ и И Т. 13, № 2 (35). С. 167–179

УДК 004.91

В. В. МИРОНОВ, К. Э. МАЛИКОВА

ИНТЕРНЕТ-ПРИЛОЖЕНИЯ НА ОСНОВЕ ВСТРОЕННЫХ ДИНАМИЧЕСКИХ МОДЕЛЕЙ: ИДЕЯ, КОНЦЕПЦИЯ, БЕЗОПАСНОСТЬ

Обсуждается новый класс интернет-приложений – на основе встроенных динамических моделей. Затрагиваются вопросы построения таких приложений. Рассматривается пример, иллюстрирующий концепцию. *Интернет-приложение; динамическая модель; встроенная модель; интерпретация динамических моделей; XML-технологии*

ВВЕДЕНИЕ

В настоящее время глобальное информационное пространство обеспечивается множеством интернет-приложений, т. е. программных компонентов, функциональные возможности которых обеспечиваются сервером и доставляются конечным пользователям по распределенной сети с помощью интернет-протоколов. За последние годы инфраструктура интернет-приложений расширилась за счет динамической обработки контента, динамического поведения и представления информационного наполнения, соответствующего потребностям пользователей при управлении разнообразными бизнес-процессами на основании гибких сценариев на сервере. Как правило, в качестве фундамента связи между пользователем и сервером применяется протокол HTTP (HyperText Transfer Protocol), известным ограничением которого является отсутствие учета информации о текущем состоянии, в результате чего каждая транзакция изолирована от предыдущих и последующих. Для преодоления этого ограничения серверные платформы, базирующиеся на HTTP, используют сессионные механизмы, позволяющие сохранить данные пользователя между отдельными шагами сеанса. Вместе с тем, обеспечение контроля текущего состояния сложных бизнес-процессов остается достаточно трудоемкой задачей при разработке интернет-приложений.

Разработку интернет-приложения, обслуживающего достаточно сложный бизнес-процесс, можно представить в виде двух этапов: проектирования концептуальной модели обработки и программирования серверного сценария. На первом этапе, в частности, разрабатывается динамическая модель бизнес-процесса, задающая его существенные состояния, переходы между ними и действия, ассоциированные с состояниями и переходами. Большинство методологий системного моделирования, таких как SADT, UML, ARIS[1], поддерживают построение динамических моделей. На втором этапе разрабатывается серверный сценарий, реализующий необходимые функции с учетом состояний, определенных в динамической модели. Сложность перехода от первого этапа ко второму связана с различными уровнями абстракциями этих этапов: на первом этапе рассматриваются состояния и переходы, а на втором – команды языка программирования серверного сценария (PHP, ASP, ASP.NET и др.).

Для снижения трудоемкости этого перехода целесообразно автоматизировать процесс построения серверного сценария на основе динамической модели соответствующего бизнес-процесса. Могут быть предложены два пути решения этой задачи:

1) На основе компиляции динамической модели в серверный сценарий. Этот путь предполагает построение компилятора, транслирующего динамическую модель бизнес-процесса в программу (или в «заготовку» программы) серверного сценария.

Контактная информация: (347) 272-89-81

[1] SADT (Structural Analysis and Design Technique) – методология структурного анализа и проектирования для описания бизнес-процессов; UML (Unified Modeling Language) – унифицированный язык моделирования бизнес-процессов; ARIS (Architecture of Integrated Information Systems) – среда моделирования для описания и анализа бизнес-процессов.

2) На основе интерпретации динамической модели. Этот подход предполагает встраивание динамической модели в явном виде в серверный сценарий и интерпретацию ее в процессе функционирования интернет-приложения. Обрабатывая на сервере встроенную динамическую модель, интерпретатор должен генерировать контент, возвращаемый клиенту в ответ на его запрос. В этой работе рассматривается второй подход.

Необходимо отметить, что идея встраиваемых динамических моделей давно и успешно применялась при построении систем управления техническими объектами [3], электронными документами [2, 4–9]. Здесь эта идея переносится в новую область – для организации управления процессом функционирования интернет-при-ложений. В этом плане ниже обсуждаются идея, концепция и особенности интернет-приложений со встроенной динамической моделью, а так же вопросы защищенности информации при таком подходе.

1. КОНЦЕПЦИЯ ИНТЕРНЕТ-ПРИЛОЖЕНИЙ НА ОСНОВЕ ВСТРОЕННЫХ ДИНАМИЧЕСКИХ МОДЕЛЕЙ

Идея встраиваемых динамических моделей в интернет-приложениях состоит в том, что в составе серверного сценария предусматривается специальный объект – динамическая модель, в котором заданы набор возможных состояний, переходов из состояния в состояния, правил, определяющих условия переходов, прикладных фрагментов кода, ассоциированных с состояниями и переходами. Предусмотренный интерпретатор динамической модели играет двоякую роль в процессе функционирования интернет-приложения:

- отслеживает и сохраняет текущие состояния модели, проверяя условия переходов;
- формирует ответ на запрос, собирая прикладные фрагменты, соответствующие текущим состояниям динамической модели.

Предполагается, что встроенная динамическая модель по своей форме достаточно близка к динамической модели бизнес-процесса, которая создается разработчиком на этапе концептуального проектирования интернет-приложения, поэтому этапы логического и физического проектирования приложения будут менее трудоемкими, чем программирование кода приложения «с нуля».

1.1. Взаимодействие клиента с интернет-приложением

Взаимодействие клиент-серверных Web-систем начинается с отправки URL-запроса (рис. 1). В случае, если клиентский запрос принимается на определенную статическую страницу, то серверное приложение, самостоятельно обслужив запрос, отправляет HTML-содержимое запрашиваемой страницы. Если же клиентская сторона обращается к серверному сценарию, то, получив соответствующий запрос, серверное приложение выполняет предусмотренные сценарием действия для динамического формирования страницы. Серверные сценарии по мере необходимости выполняют выборку данных из надлежащего источника и представляют динамически сгенерированное содержание клиенту. В простейшем случае это HTML-код, в соответствии с которым браузер формирует изображение. В более сложном случае ответ включает информационные объекты, предназначенные для дальнейшей интерпретации на стороне клиента. Агент пользователя интерпретирует ответ в соответствии с некоторым языком структуризации и в результате этого формирует изображение, отображаемое пользователю.

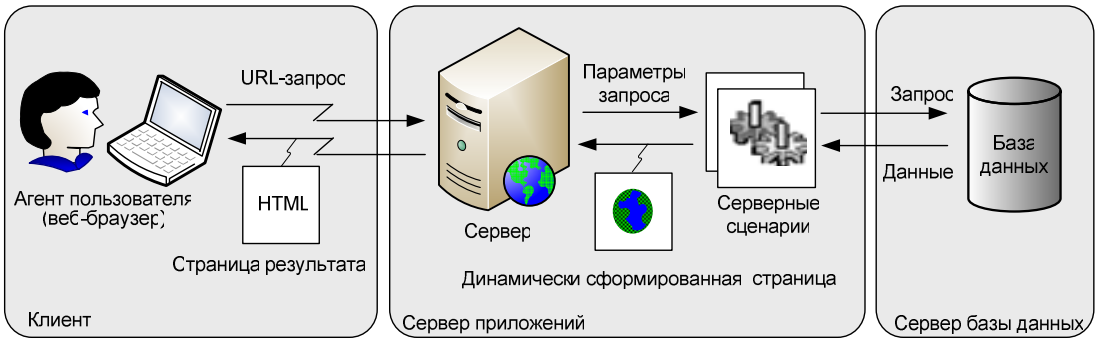
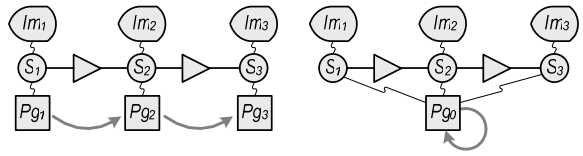


Рис. 1. Общая схема взаимодействия клиента и интернет-приложения

Традиционный подход (рис. 2, *а*) основан на выделении для каждого динамического состояния отдельной серверной страницы. Так, если бизнес-процесс, обслуживаемый данным приложением, имеет, скажем, три состояния S_1, S_2 и S_3, с каждым из которых ассоциировано некоторое изображение на экране пользователя Im_1, Im_2 или Im_3, то в составе приложения предусматривается три страницы Pg_1, Pg_2 и Pg_3, которые обеспечивают формирование этих изображений и переходы между страницами в соответствии с реакцией пользователя или другими событиями. Например, изображение Im_1, формируемое страницей Pg_1, представляет собой форму для ввода пользовательских данных, в параметре Action (действие), которой указаны URL-адрес страницы Pg_2, т. е. при нажатии пользователем кнопки типа Submit (принять), управление переходит к странице Pg_2. В свою очередь, страница Pg_2 формирует изображение, предусматривающее переход к странице Pg_3 по определенному действию пользователя. Таким образом, эмулируются сложные динамические модели, включающие много состояний со сложной логикой переходов между состояниями.



а *б*

Рис. 2. Состояния динамической модели и изображения на экране клиента при традиционном *(а)* и предлагаемом *(б)* подходах

Предлагаемый подход (рис. 2, *б*) реализует динамические состояния бизнес-процесса с помощью сценария серверной страницы. Здесь серверный сценарий содержит динамическую модель взаимодействия с клиентом, с элементами которой ассоциированы элемент-фрагменты формирования разметки изображения (Im_n). На каждом шаге взаимодействия с клиентом отслеживаются переходы текущего состояния динамической модели (S_n) и из соответствующих элемент-фрагментов собирается разметка изображения для следующего шага взаимодействия (Pg_n). Генерация изображения осуществляется в результате интерпретации встроенной динамической модели, размещенной на серверной стороне. При погружении в исходную динамическую модель с учетом памяти текущего состояния интерпретатор определяет доступные состояния и ассоциированные с ними фрагменты программного кода и, выполняя их последовательную обработку, формирует разметку изображения, соответствующую текущему состоянию. В свою очередь, для каждого текущего состояния контролируются доступные переходы, в результате выполнения которых происходит смена текущего состояния и собирается разметка изображения, соответствующая новому текущему состоянию.

Структура модели и структура изображения. Бизнес-процессам с достаточно сложной логикой, как правило, соответствуют динамические модели иерархического вида, где с каждым состоянием могут быть связаны внутренние подмодели, в свою очередь, включающие набор состояний. Отражаемый контент, ассоциированный с состояниями, в этих случаях также имеет иерархическую организацию. При реализации встроенных динамических моделей предлагается руководствоваться принципом соответствия структуры изображения структуре динамической модели. В соответствии с этим принципом каждой подмодели, входящей в состав динамической модели, на экране соответствует некоторая область изображения, которая заполняется контентом в соответствии с текущим состоянием подмодели. Область отображения подмодели, в свою очередь, включает постоянную область, содержимое которой не зависит от текущего состояния подмодели, и переменную область, содержимое которой определяется текущим состоянием подмодели. При смене текущего состояния содержимое переменной области изменится. Если текущее состояние, в свою очередь, содержит подмодель, то переменная область будет содержать подобласть, соответствующую этой подмодели. Таким образом, полная структура изображения является оверлейной (перекрывающейся), состоящей из налагаемых друг на друга областей.

1.2. Многопользовательский режим

Интернет-приложение – это изначально многопользовательское приложение, несколько экземпляров которого могут одновременно выполняться на сервере, обслуживая нескольких пользователей. В этой связи возникает вопрос о том, каким образом встроенная динамическая модель обеспечивает обслуживание клиентов в многопользовательском режиме.

Категории пользователей. Встроенная динамическая модель является общей для всех пользователей интернет-приложения, а ее текущее состояние отслеживается независимо для каждого пользователя. В этой связи рассматри-

ваются две категории пользователей интернет-приложения: незарегистрированные (анонимные) и зарегистрированные. Для выявления зарегистрированных пользователей в модели должны быть предусмотрены соответствующие процедуры аутентификации. Текущие состояния для динамической модели для анонимных пользователей могут отслеживаться лишь в течении сеанса взаимодействия с интернет-приложением, а для зарегистрированных – и между сеансами (например, запустив интернет-приложение на следующий день, зарегистрированный пользователь может продолжить работу с того состояния, в котором прервал ее накануне).

Многопользовательская модель. Для обслуживания как анонимных, так и зарегистрированных пользователей встроенная динамическая модель может содержать подмодели, доступные только для первых, только для вторых, а также для тех и других. На рис. 3 поясняется такая возможность.

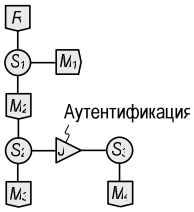


Рис. 3. Разделение встроенной модели на фрагменты, доступные различным категориям пользователей

Головная модель R в начальном состоянии S_1 содержит подмодель M_1. Поскольку S_1 является текущим независимо от аутентификации пользователя, подмодель M_1 доступна как зарегистрированным, так и анонимным пользователям. В состоянии S_1 предусмотрены также подмодель M_2, обеспечивающая аутентификацию. Состояние S_2 – начальное состояние в модели M_2 – соответствует незарегистрированному пользователю, а состояние S_3, в которое переходят из состояния S_2 в результате успешной аутентификации – зарегистрированному пользователю. Таким образом, подмодель M_3 доступна только анонимному (пока не аутентифицированному) пользователю, а модель M_4 – только зарегистрированному пользователю.

1.3. Состояния отображения и состояния обработки

Состояния встроенной динамической модели в зависимости от назначения встроенных ассоциированных с ними прикладных фрагментов можно разделить на:

- состояния отображения, предусматривающие отображение пользователю некоторого контента в тех ситуациях, когда эти состояния являются текущими, и ожидания той или иной реакции пользователя;
- состояния обработки, используемые для выполнения тех или иных вычислений и проверок, не связанных со взаимодействием с пользователем (например, обращения к серверной базе данных).

Возможны также смешанные состояния, в которых предусматривается и другие прикладные функции.

На рис. 4 иллюстрируются состояния отображения и обработки на примере фрагмента динамической модели, связанного с аутентификацией пользователя.

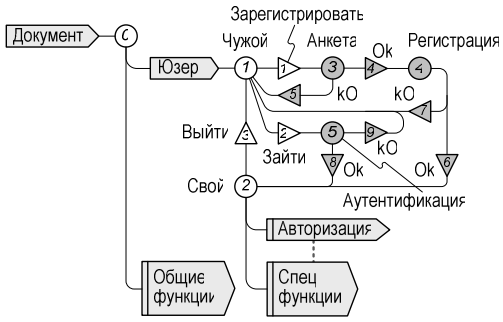


Рис. 4. Состояния отображения (светлые) и состояния обработки (темные)

Модель представлена в нотации так называемых иерархических ситуационных моделей (ИСМ) [1], позволяющих задавать состояния (круги) и переходы (треугольники) и внутренние подмодели (пятиугольники). Корневая подмодель «Документ» включает единственное состояние *«0»* с двумя внутренними подмоделями «Юзер» и «Общие функции», первая из которых обеспечивает процесс аутентификации и обслуживания зарегистрированного пользователя, а вторая – обслуживание как анонимного, так и зарегистрированного.

Подмодель «Юзер», в свою очередь, содержит два состояния отображения (на рис. 4 светлая заливка): *1* – «Чужой», соответствующее не зарегистрированному анонимному пользователю и *2* – «Свой», соответствующее зарегистрированному пользователю, а также три состояния обработки (на рис. 4 темная заливка): *3* – «Анкета» – проверка правильности заполнения регистрационной анкеты; *4* – «Регистрация» – создание учетной записи пользователя; *5* – «Аутентификация» – проверка введенных имени и пароля. Состояния отображения сопровождаются вводом соответствующего контента и ожидания на сервере той или иной реакции пользователя.

На модели этим реакциям соответствуют переходы (треугольники со светлой заливкой): *1* – «Зарегистрировать» – пользователь хочет зарегистрироваться, он заполнил регистрационную анкету и отправил ее на сервер для регистрации. *2* – «Зайти» – пользователь хочет войти как зарегистрированный, он ввел свои имя и пароль и нажал кнопку входа. *3* – «Выйти» – зарегистрированный пользователь хочет выйти из этого режима (стать анонимным пользователем), для этого он нажал кнопку выхода. Состояния обработки сопровождаются обращением к серверной базе данных и проверкой определенных условий, при выполнении которого срабатывают соответствующие пере переходы (треугольники с темной заливкой). Так,

в состоянии «Анкета» введенные пользователем анкетные данные проверяются с учетом уже имеющихся зарегистрированных пользователей. Если проверка прошла успешно, срабатывает переход *4* – «Ok» в состояние обработки «Регистрация», а если не успешно – переход *5* – «kO», возвращающий в состояние отображения «Чужой». В состоянии «Регистрация» в серверной базе данных создается учетная запись нового пользователя. Если эта процедура выполнена без ошибок, срабатывает переход *6* – «Ok», возвращающий в состояние отображения «Чужой». В состоянии обработки «Аутентификация» выполняется поиск в базе данных учетной записи, соответствующей введенным имени пользователя и паролю. В случае успеха срабатывает переход *8* – «Ok» в состояние отображения «Свой», в противном случае – переход *9* – «kO» в состояние отображения «Чужой».

1.4. Клиентские и серверные подмодели, иерархическая структура изображения

Исходная динамическая модель бизнес-процесса обычно строится без учета того, на какой стороне выполняются функции, ассоциированные с состояниями на серверной или клиентской. При встраивании модели в интернет-приложения этот вопрос становится существенным, поскольку серверные и клиентские функции реализуются по-разному: серверные выполняются в серверном сценарии, а клиентские – в Web-браузере клиента и должны быть соответствующим образом загружены в браузер в виде скрипта на языке JavaScript или VBA. Поэтому необходимо отделить в динамической модели серверные и клиентские составляющие путем выделения клиентских составляющих в отдельные подмодели. На рис. 5 этот процесс иллюстрируется для рассмотренной выше модели аутентификации пользователя.

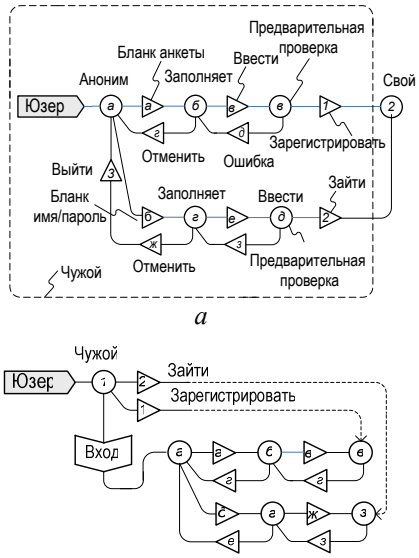
Пусть в исходной модели динамической подмодели «Юзер» подробно специфицированы состояния, связанные с заполнением пользователем регистрационной анкеты и вводом имени/пароля (рис. 5, *а*).

Здесь начальное состояние *а* – «Аноним» соответствует анонимному пользователю, из него по инициативе пользователя возможны два перехода: *а* – «Бланк анкеты» в состояние *б* – «Заполняет», *б* – «Бланк имя/пароль» в состояние *г* – «Заполняет».

В ситуациях *б* и *г* пользователю предоставляются для заполнения формы ввода регистрационной анкеты или имени и пароля соответственно. Переходы *в* и *е* «Ввести» по инициативе пользователя переводят модель в состояния *в* и *д* «Предварительная проверка», где выполняется проверка корректности введенных данных на клиентском уровне. Переходы *д* и *з* – «Ошибка» возвращают процесс в состояния *б* и *г*, соответственно в случае обнаружения ошибок, а переходы по инициативе пользователя *г* и *ж* – «Отменить» – в состояния *а* – «Аноним».

Переходы *1* – «Зарегистрировать» и *2* – «Зайти», рассмотренные выше (см. рис. 3), обеспечивают перевод процесса в состояние *2* – «Свой».

Рассмотренные состояния *а–д*, а также связанные с ними переходы *а–з* могут быть реализованы на стороне клиента, в Web-браузере, поскольку они представляют собой последовательность действий, не нуждающуюся в услугах сервера. Поэтому они выделяются в самостоятельную подмодель «Вход» (рис. 5, *б*).



а

б

Рис. 5. Выделение клиентских подмоделей: *а* – подмодели «Юзер» исходной динамической модели; *б* – клиентская подмодель и ее связь с серверной стороной

Здесь для обозначения клиентской подмодели вместо символа – пятиугольника применяется особый символ – шестиугольник. Клиентская подмодель означает, что они должны быть загружены в Web-браузер клиента в виде скрипта, когда ее родительское серверное состояние (в данном случае – состояние *1* – «Чужой») станет текущем. Серверные переходы *1* – «Зарегистрировать» и *2* – «Зайти» срабатывают в зависимости от текущего состояния клиентской подмодели «Вход», а именно, если текущим являются состояния *в* и *д* соответственно.

Рассмотрим процедуру выполнения переходов и процесс погружения в подмодели различных уровней на примере (рис. 4).

На верхнем уровне динамической модели (рис. 6, *а*) представлена основная подмодель верхнего уровня иерархии с именем *A*. Два состояния данной подмодели – *B* и *D*. соединены друг с другом переходами *C* и *E*. Состояние *B* изначально является начальным соответствующей подмодели и становится текущим автоматически при первой загрузке динамической модели. Состояние *D* не является начальным и становится текущим только при выполнении перехода – *C* в рамках данной подмодели.

Согласно структуре встроенной динамической модели [8], состояние *D* подмодели *A* содержит собственные подмодели – *F* и *G* которые, в свою очередь, также структурно организованы. В этой связи, при выполнении погружения в исходную подмодель *A*, текущим сразу становится начальное состояние *B*.

Но если для выполнения был выбран доступный переход *C* состояния *B* модели *A* (рис. 6, *б*), то произойдет смена текущего состояния с начального *B* на *D*. Так как состояние *D*, в свою очередь, содержит подмодели *F* и *G*, то автоматически выполняется погружение в его подмодели, то есть наряду с данным текущим состоянием – *D*, начальное состояние модели *F* – *H* и начальное состояние модели *G* – *L* становятся текущими (рис. 5, *в*). При выборе

доступного перехода *M* состояния *L* модели *G*, начинается процедура изменения текущего состояния подмодели *G*, в результате текущими состояниями становятся состояния *D*, *H* и *N* (рис. 6, *г*).

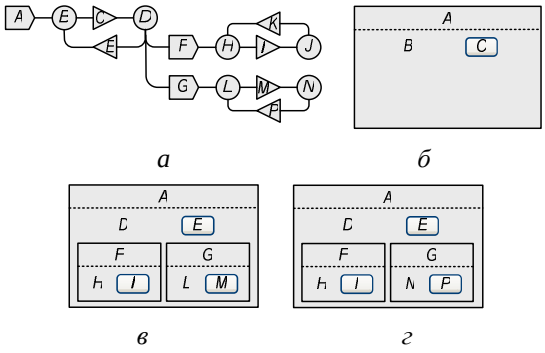


Рис. 6. Соответствие элементов динамической модели и фрагментов изображения в агенте пользователя: *а* – динамическая модель; *б* – изображение для начального состояния *B*; *в* – при переходе из *B* в *D*; *г* – при переходе из *L* в *N*

Как видно, изменения текущего состояния происходят только в рамках той подмодели, которой принадлежит выполненный переход. При этом текущие состояния других подмоделей того же уровня остаются неизменными, что позволяет одновременно оперировать несколькими состояниями, входящими в подмодели различных уровней.

1.5. Полная и частичная перегрузка страниц

Итак, встроенная динамическая модель должна обеспечивать смену отображаемого контента в соответствии с изменением своего текущего состояния. В настоящее время для формирования в браузере пользователя изображения, динамически меняющегося под управлением сервера, применяются две технологии:

1) Традиционный метод полной перегрузки страницы (ППС), который предусматривает формирование на сервере всей страницы отображения заново при изменении ее контента и отправку страницы пользователю для замены текущей (полной перегрузки). Очевидный недостаток этого простого в реализации подхода состоит в необходимости пересылать всю страницу целиком доже в тех случаях когда изменению подлежит ее небольшой фрагмент. При этом пользователь должен ожидать окончания загрузки страницы, чтобы продолжить работу с ней.

2) Относительно новый метод частичной перегрузки страницы (ЧПС, получивший также название Ajax[2] [10]), позволяющий заменять отдельные фрагменты изображения в браузере. ЧПС основан на специальных запросах типа HttpRequest, которые браузер посылает серверу и в ответ на которые сервер возвращает заменяемый фрагмент. Посылка запросов HttpRequest и вставка в HTML-код заменяемого изображения выполняется в соответствии с клиентскими сценариями на JavaScript, иногда достаточно громоздким, которые сервер предварительно должен загрузить в браузер. Такой подход дает возможность быстро менять необходимые фрагменты изображения, позволяя пользователю асинхронно работать в это время с другими фрагментами (например, заполнять поля формы) [9].

Серверный сценарий на основе встроенной динамической модели может основываться как на том, так и на другом методе перегрузки страницы. Процесс интерпретации динамической модели в каждом случае имеет свои особенности.

Полная перегрузка страниц. Цикл интерпретации динамической модели для реализации динамического изображения методом ППС должен обеспечивать полное формирование страницы, следовательно, он долен охватывать всю модель, начиная с корня. Все циклы интерпретации выполняются единообразно: динамическая модель рекурсивно обходится «сверху вниз» по начальным/текущим состояниям, с выполнение переходов текущего состояния и отображением его в ПТС. При этом выполняется сборка кода изображения из прикладных фрагментов, ассоциированных с элементами динамической модели. По завершению цикла собранный код отправляется в браузер клиента. Необходимо различать начальный и последующий циклы интерпретации:

- начальный цикл соответствует началу клиентского сеанса и запускается в результате обращения к интернет-приложению методом GET;
- последующие циклы запускаются в рамках сеанса методом POST в результате воздействий пользователя на элементы управления (кнопки, переключатели и т. п.). В отличие от начального, этот цикл сопровождает набор входных параметров, характеризующих действия пользователя (например, идентифицирующих их нажатую кнопу экранной формы). В со-

[2] Ajax (Asynchronous Javascript and XML – асинхронный JavaScript и XML») – подход к построению динамических интернет-приложений, позволяющих выполнять динамические запросы к серверу без перезагрузки интернет-страницы.

ответствии со входными параметрами происходит срабатывание тех или иных переходов в динамической модели и соответственное изменение его текущего состояния, приводящее к изменению изображения, отправляемого клиенту.

Частичная перегрузка страниц. Цикл интерпретации динамической модели в этом случае имеет различия для начального и последующих запросов заметней, чем в случае с ППС.

Начальный цикл выполняется в ответ на GET-запрос клиента, от отправляет сеанс и предусматривает интерпретацию всех динамической модели, начиная с корня. В результате интерпретации формируется HTML-код начального изображения страницы, а также JavaScript-код клиентских сценариев обработки запросов HttpRequest.

Последующие циклы выполняется в ответ на запросы HttpRequest, требующие перегрузки того или иного фрагмента изображения страницы. Если динамическая модель построена так, что чтобы каждому логическому фрагменту страницы в динамической модели соответствует некоторая подмодель, а конкретное содержание фрагмента определяется текущим состоянием этой подмодели, то для получения обновленного фрагмента контента достаточно выполнить интерпретацию соответствующей подмодели динамической модели. Таким образом, интерпретатор должен позволять выполнять интерпретацию отдельных подмоделей динамической модели (естественно, вместе со своими внутренними подмоделями) в соответствии с указаниями, содержащимися в параметрах запроса HttpRequest. Совокупность данных, полученных в результате интерпретации подмодели, представляет собой HTML-код нового содержимого фрагмента, а также JavaScript-сценарии обработки запросов HttpRequest для внутренних состояний этого фрагмента (если таковые предусмотрены).

На рис. 7 иллюстрируется полное и частичное обновление изображения при смене текущего состояния динамической модели. Исходное изображение соответствует состоянию *D* корневой подмодели *A* и состоянию *L* подмодели *G*. Пользователь нажимает на кнопку *M*, соответствующую переходу, и в результате формируется результирующее изображение. В случае полного обновления (рис. 6, *ветвь 1*) происходит запуск интерпретации корневой модели A, в результате чего в модели сменяется состояние $B \rightarrow D$ и формируется изображение целиком. В случае частичного обновления (рис. 6, *ветвь 2*) происходит запуск интерпретации перехода *M*, в результате чего выполняется смена состояния $L \rightarrow N$ и формируется фрагмент изображения *N*, соответствующий новому состоянию. Этот фрагмент поступает в браузер и замещает фрагмент *L* в исходном изображении.

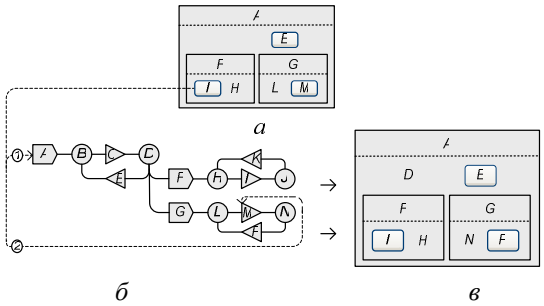


Рис. 7. Пример полного и частичного обновления изображения при смене текущего состояния динамической модели: *а* – исходное изображение, соответствующее состояниям *D* и *L*; *б* – динамическая модель; *в* – результирующее изображение, соответствующее состояниям *D* и *N*; *1* – интерпретация всей динамической модели при полном обновлении изображения; *2* – интерпретация перехода *M* при частичном обновлении изображения в состоянии обработки

1.7. Одно- и двухпроходная интерпретация

Интерпретация динамической модели предполагает обработку двух видов ее элементов: управляющих, служащих для отслеживания текущих состояний иерархии подмоделей, и прикладных, содержащих фрагменты HTML и скрипт-кода формирования изображения. Совместная обработка таких элементов в некоторых случаях может приводить к коллизиям, которых необходимо избежать. На рис. 8 иллюстрируется одна из таких возможных коллизий.

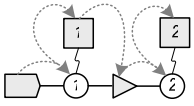


Рис. 8. К пояснению коллизий при совместной обработке управляющих и прикладных действий динамической модели (пунктиром показана последовательность обработки)

Модель содержит два состояния: *1* и *2*, с каждым из которых ассоциирован свой прикладной фрагмент. Пусть во время обработки текущим было состояние *1* и сработал переход в состояние *2*. При обработке состояния *1* интерпретатор сначала обрабатывает и поместит в буфер результата фрагмент *1*, а затем, отследив смену текущего состояния, добавит фрагмент *2*. На самом деле требуется так, чтобы фрагмент *2* заменял фрагмент *1*, а не дополнял его.

Для устранения коллизий такого рода нужно либо ввести определенные правила, задающие условия обработки прикладных фрагментов относительно обработки управляющих элементов, в рамках однопроходной интерпретации динамической модели, либо разделить процесс интерпретации на два последовательных этапа:

контроль текущего состояния и сборка результирующего контента (т. е. перейти к двухпроходной интерпретации).

Однопроходная интерпретация. Для предотвращения коллизий зададим для прикладных элементов приоритеты, определяющие условия их обработки относительно срабатывающих управляющих элементов. Например, для каждого состояния введем общую упорядоченность переходов, погружений и прикладных элементов и условимся выполнять обработку прикладных действий, только если отсутствуют сработавшие управляющие элементы.

Например, на рис. 8, приоритет перехода из состояния *1* в состояние *2* зададим выше приоритета прикладного фрагмента *1*, тогда при обработке состояния *1* в случае пассивного перехода фрагмент *1* будет обработан, а в случае активного – не будет, а будет выполнен переход в состояние *2*, где будет обработан прикладной фрагмент *2*. Отметим, что в этом случае прикладные действия могут быть ассоциированы как с состояниями, так и с переходами.

Двухпроходная интерпретация. На первом проходе интерпретатор обрабатывает динамическую модель, игнорируя прикладные фрагменты, а только контролируя переходы текущего состояния. На втором, напротив, игнорируются переходы состояния, а выполняется сборка прикладных фрагментов в результате рекурсивного обхода текущих состояний динамической модели.

Возвращаясь к примеру на рис. 8, видим, что при первом проходе будет выполнен переход из состояния *1* в состояние *2*, а при втором – обработан и помещен в буфер результата фрагмент *2*, как и должно быть.

Отметим, что в этом случае прикладные действия могут быть ассоциированы только с состояниями.

Сравнивая между собой два подхода к интерпретации, можно отметить, что если в первом изображение формируется на основе всей истории смены текущих состояний на данном цикле интерпретации динамической модели, то во втором – на основе результирующих текущих состояний, достигнутых по завершении цикла интерпретации. Можно также высказать предположение, что первый подход требует более изощренного проектирования динамической модели и результат будет более запутанным для анализа, хотя внешне модель может выглядеть проще. Вместе с тем, более простой для анализа второй подход требует так спроектировать динамическую модель, чтобы она была «марковской по отношению к результату» в том смысле, что формируемое на ее основе результирующее изображение страницы зависело бы только от текущего состояния и не зависело от истории попадания в это состояние (от предшествующих состояний). Возможно, что в некоторых состояниях это усложнит саму модель.

9. ПРИМЕР

Динамическая модель верхнего уровня (рис. 9) «Dynamo» («динамическая модель») обеспечивает аутентификацию пользователя. Начальное состояние «Аноним» соответствует зарегистрированному пользователю, состояние «Проверка» обеспечивает аутентификацию, состояние «Свой» соответствует зарегистрированному пользователю, успешно прошедшему аутентификацию Подмодель «Детали» состояния «Свой» обеспечивает вывод на экран сведений о зарегистрированном пользователе: в состоянии «Телефон» – номера телефона, а в состоянии «Email» – электронного адреса.

Действия. С элементами модели – состояниями, переходами, погружениями – ассоциированы элементы-действия управляющего типа (серая заливка), предназначенные для управления сменой текущего состояния динамической модели и прикладного типа (светлая заливка), предназначенные для взаимодействия с прикладной базой данных сервера и формирования кода изображения, отправляемого пользователю. Сводка содержимого фрагментов представлена в табл. 1. Фрагменты написаны на языке серверных сценариев PHP и содержат HTML-код с вкраплениями PHP-тегов (табл. 1). В разрывах соединительных линий, связывающих фрагменты с элементами модели, указаны приоритеты их обработки (если они существенны).

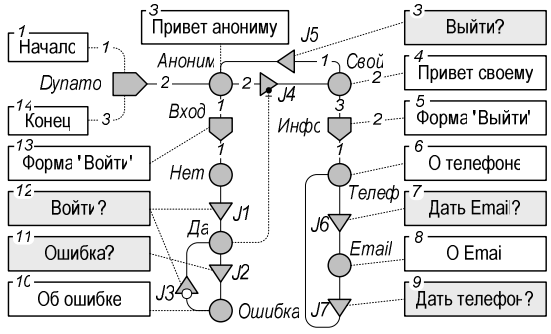


Рис. 9. Пример встроенной динамической модели с управляющими (серая заливка) и прикладными (белая заливка) действиями

Двухпроходная интерпретация. Рассматриваемый пример ориентирован на двухпроходную интерпретацию встроенной динамической модели. Первый проход обеспечивает отслеживание текущих состояний, сохраняемых в ПТС;

он учитывает только управляющие фрагменты-действия. Второй проход формирует возвращаемый пользователю код; он учитывает только прикладные фрагменты действия. Далее рассматривается цепочка ситуаций, возникающих в ходе взаимодействия пользователя с приложением, а также соответствующие состояния ПТС и изображения в окне браузера (рис. 10).

Таблица 1

Спецификация действий
для примера динамической модели

№ п/п	Обозначение	у/п	Программный код
1	Начало	п	<html><head><title>Это Dynamo </title></head><
2	Привет аноniму	п	Hello, Аноним!
3	Выйти?	у	<?php $pred=$_POST['exit']; ?>
4	Привет своему	п	Тов.,<?php echo $name; ?>
5	Форма «Выйти»	п	<form method='post'> <input type='submit' name= 'exit' value='Выйти' title='Выйти'/> </form>
6	О телефоне	п	Ваш Телефон: <?php GetTel(); ?> <form method='post'> <input type='submit' name= 'email'' value='Дать Email' title='Дать Email'/> </form>
7	Дать E-mail?	у	<?php $pred=$_POST['email']; ?>
8	О E-mail	п	Ваш Email: <?php GetEmail(); ?> <form method='post'> <input type='submit' name= 'tel' value='Дать телефон' title='Дать телефон'/> </form>
9	Дать телефон?	у	<?php $pred=$_POST['tel']; ?>
10	Об ошибке	п	Неверное имя!
11	Ошибка?	у	<?php $pred=TestName ($name); ?>
12	Войти?	у	<?php pred=$_POST['$enter']; ?>
13	Форма «Войти»	п	<form method='post'> Введите имя: <input type='name' name='name'/> <input type='submit' name='enter ' value='Войти' title='Войти'/> </form>
14	Конец	п	</head><body></html>

Ситуация 1. Это ситуация начального шага сеанса взаимодействия клиента с интернет-приложением. Клиент посылает URL-запрос по адресу приложения. Интерпретатор создает пустой экземпляр ПТС для этого сеанса и приступает к первому проходу интерпретации динамической модели, начиная с головной подмодели «Dynamo». Поскольку текущего состояния для этой модели в ПТС нет, устанавливается начальное состояние «Аноним» и запускается интерпретация этого состояния, в ходе которой последовательно обрабатываются ассоциированные элементы. Первой обрабатывается подмодель «Вход», для которой, ввиду отсутствия в ПТС, в качестве текущего устанавливается начальное состояние «Нет».

В ходе интерпретации этого состояния, обрабатывается переход *J1*, для чего выполняется управляющее действие *12* «Войти?». Это действие проверяет, была ли нажата пользователем кнопка «Войти», которая на начальном этапе, естественно, не может быть нажата, поскольку отсутствует на экране. Переход *J1* пассивен, подмодель «Вход» остается в состоянии «Нет». Продолжая обработку исходящих элементов состояния «Аноним», выполняется обработка перехода *J4*. Этот переход имеет внутренний предикат, тестирующий текущее состояние «Да» подмодели «Вход». Поскольку эта подмодель находится в состоянии «Нет», переход *J4* пассивен и головная модель остается в состоянии «Аноним». Первый проход интерпретации завершен, в результате чего в ПТС для головной модели фиксируется состояние «Аноним», а для его внутренний подмодели «Вход» – состояние «Нет» (рис. 10, *а*).

На втором проходе интерпретации выполняется сборка прикладных фрагментов, соответствующих текущим состояниям динамической модели. В буфере результата оказывается следующая последовательность фрагментов:

1, 2, 13, 14.

Содержимое последовательности возвращается клиенту, в результате чего в окне браузера формируется изображение (рис. 10, *А*), содержащее приветствие анонимному пользователю.

Ситуация 2. Это ситуация следующего шага сеанса, когда пользователь ввел свое имя и нажал кнопку «Войти», но при этом указал неправильное (несуществующее) имя. Интерпретатор извлекает из ПТС экземпляр, соответствующий предварительному шагу клиента, и в рамках первого прохода приступает к интерпретации головной модели. Обрабатывая текущее состояние «Аноним», интерпретатор погружается в модель «Вход», в текущее состояние «Нет». Обрабатывая это состояние, он проверяет переход *J1* и в результате выполнения управляющего действия *12* «Войти» обнаруживает активность, поскольку пользователем была нажата соответствующая кнопка. Фиксируется переход из состояния «Нет» в состояние «Да». Запускается обработка состояния «Да», в ходе которой проверяется активность перехода *J2*. Поскольку пользователем введено неправильное имя, отсутствующее в регистрационной базе приложения, выполняется переход из состояния «Да» в состояние «Ошибка». Переход *J3* является *нетранзитным* (кружок на треугольнике – символе перехода, см. рис. 9), т. е. не может быть активным, если исходное состояние стало текущим в результате перехода из другой ситуации на данном цикле интерпретации, как это имеет место в рассматриваемой ситуации. Благодаря этому предотвращается зацикливание переходов *J2 – J3*). Переход *J4* пассивен, поскольку состояние «Да» не является текущим в модели «Вход», поэтому состояние «Аноним» остается текущим для головной модели на этом шаге сеанса (рис.10, *б*).

На втором проходе в буфер результата заносится следующая последовательность прикладных фрагментов:

1, 2, 10, 13, 14,

которая формирует в окне браузера изображение, дополненное сообщением об ошибке ввода (рис. 10, *Б)*.

Ситуация 3. Это следующая ситуация, в которой пользователь ввел правильное имя и нажал кнопку «Войти». На первом проходе интерпретатор обрабатывает головную модель с текущей ситуацией «Аноним». При обработке подмодели «Вход» из текущего состояния «Ошибка» обнаруживается активный переход *J3* в результате обработки управляющего действия *J2* «Войти». Текущим состоянием становится состояние «Да», поскольку в нем переход *J2* пассивен, так как пользователем введено правильное имя (действие *11* «Ошибка?»). Далее интерпрета-

тор обнаруживает активный переход *J4*, поскольку текущим состоянием модели «Вход» является состояние «Да», и выполняет смену текущего состояния головной модели на состояние «Свой». Обрабатывая это состояние, интерпретатор убеждается в пассивности перехода *J5*, поскольку действием *3* «Выйти?» не подтверждается нажатие пользователем соответствующей кнопки (такой кнопки пока нет на экране). Погружаясь в подмодель «Инфо», интерпретатор устанавливает начальное состояние «Телефон» в качестве текущего (так как для этой подмодели не задано текущее состояние в ПТС) и убеждается в том, что переход *J6* пассивен (управляющее действие *7* «Дать Email?» не подтверждает нажатие пользователем соответствующей кнопки, которой пока еще нет на экране). Таким образом, текущими состояниями становятся «Свой» для головной модели и «Телефон» для модели «Инфо» (рис. 10, *в*). Эти текущие состояния сохраняются в ПТС данного зарегистрированного пользователя. На втором проходе в буфере результатов в соответствии с текущим состоянием помещается следующая последовательность фрагментов кода:

1, 4, 6, 5, 14,

формирующая изображение, содержащее приветствие зарегистрированному пользователю, кнопку возврата в режим анонимного пользователя, сведения о телефоне, кнопку запроса сведений об адресе электронной почты (рис. 10, *В*).

Ситуация 4. Это ситуация, в которой пользователь нажал кнопку «Дать Email». Интерпретатор обрабатывает головную модель с состояния «Свой», полученного из ПТС. Переход *J5* пассивен, так как кнопка «Войти» не была нажата, поэтому текущее состояние головной модели не меняется. Обработка подмодели «Инфо» начинается с текущего состояния «Телефон». Поверяя переход *J6*, интерпретатор убеждается в его активности с помощью действия *7* «Дать Email» и переводит подмодель в новое текущее состояние «Email». Поэтому состояние «Ошибка» остается текущим для модели «Вход». Первый проход завершается текущими состояниями: «Свой» – для головной модели, «Email» – для подмодели «Инфо» (рис. 10, *г*).

На втором проходе извлекается следующая последовательность фрагментов:

1, 4, 8, 5, 14,

соответствующая изображению на рис. 10, *Г*.

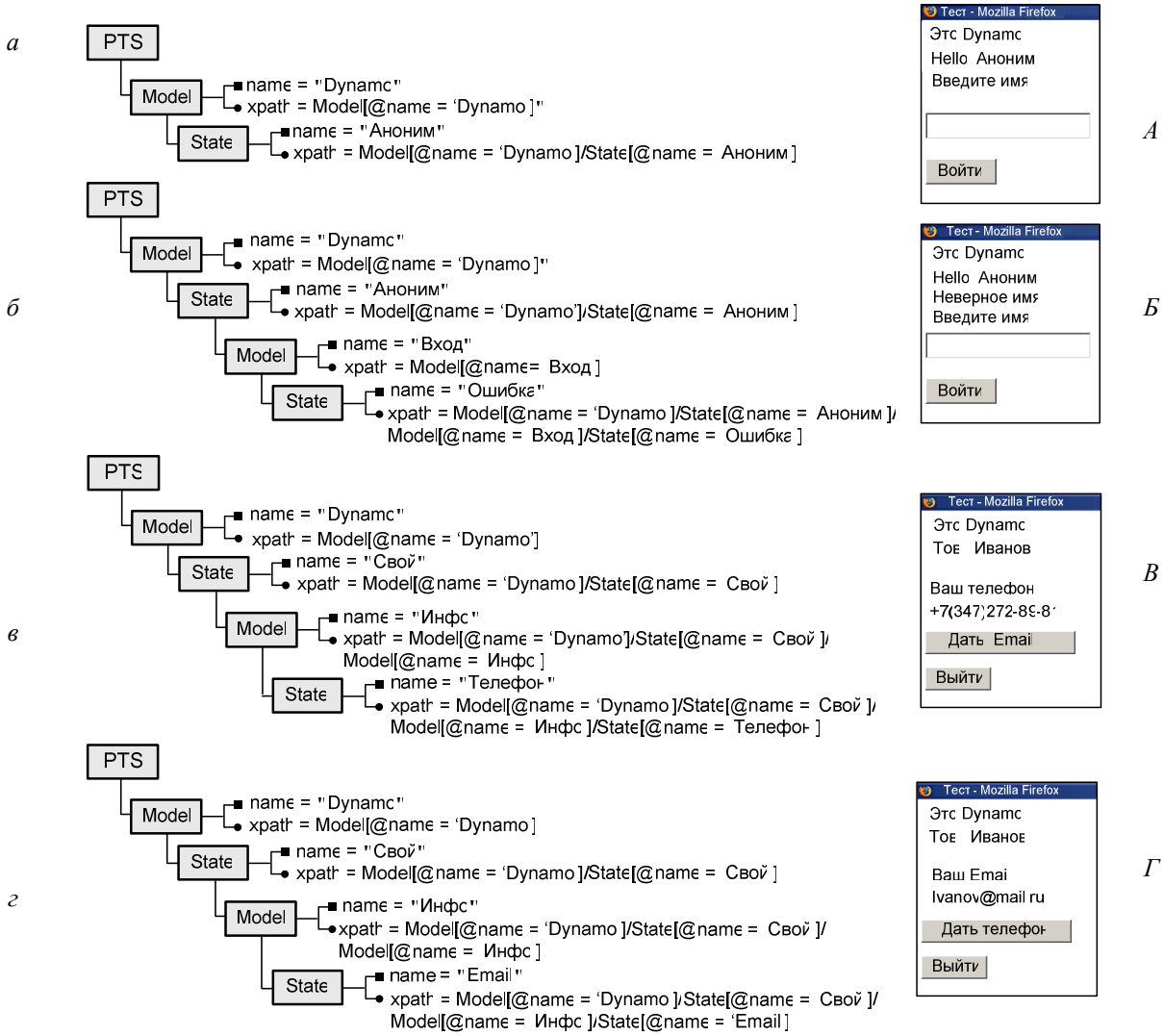


Рис. 10. Пример изменения памяти текущего состояния и изображения в окне браузера пользователя в результате погружения в исходную динамическую модель *(а)*, выполнения предиката перехода «Ошибка» *(б)*, выполнения предиката перехода «Войти» *(в)*; выполнения предиката перехода «Дать Email *(г)*

Однопроходная интерпретация. Рассмотренная выше динамическая модель специально спроектирована так, чтобы обеспечить свойство «марковости относительно результата». Свойство «марковости» гарантирует, что изображение, отправляемое клиенту может быть полностью сформировано на основе текущего состояния динамической модели, достигнутого на данном шаге сеанса работы интернет-приложения, что позволяет применить двухпроходную интерпретацию.

Если не требовать «марковости», то динамическая модель может получиться более простой в плане структуры (возможно, в ущерб понятности функционирования для разработчика и пользователя). Такая модель потребует однопроходной интерпретации при формировании кода изображения. На рис. 11 представлен фрагмент динамической модели, не являющейся «марковской» и предполагающей однопроходную интерпретацию.

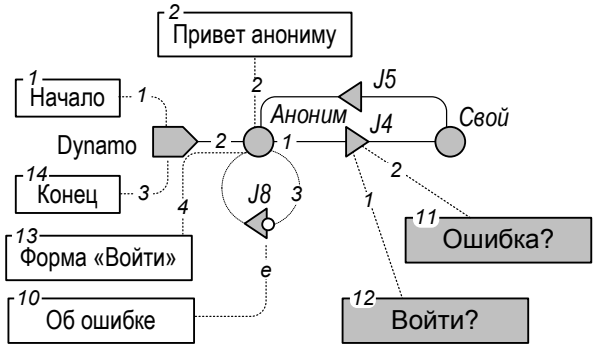


Рис. 11. Вариант динамической модели (фрагмент), не являющейся «Марковской относительно результата» и предполагающей однопроходную интерпретацию

Изменения по сравнению с «Марковской моделью» (см. рис. 10) коснулись состояния «Аноним». В нем теперь нет внутренней подмодели, вместо нее предусмотрен петлевой нетранзитный переход *J8*. С этим переходом ассоциировано эпилоговое действие-фрагмент *10* «Об ошибке», выполняемое в случае активности перехода. Активность перехода *J4* определяется двумя контролирующими действиями: *12* «Войти?» и *11* «Ошибки?» по схеме «логическое И». Изменилась и последовательность обработки элементов, ассоциированных с состоянием «Аноним», – первым обрабатывается переход *J4*, затем – действие *2*, далее – переход *J8*, последним – фрагмент *13*. В результате на начальном шаге сеанса переходы *J4* и *J8* пассивны, поскольку действие *12* не обнаруживает нажатие кнопки «Войти», поэтому изображение формируется действиями-фрагментами *2* и *13*. Если пользователем была нажата кнопка «Войти» и было введено правильное имя, то переход *J4* становится активным и модель сразу переходит в состояние «Свой». Если же имя неправильное, то переходит *J4* пассивен и обрабатывается действие *2* «Привет аноним», затем – активный переход *J8* с действием *10* «Об ошибке», после чего – *13* «Форма Войти». Нетранзитность перехода *J8* предотвращает зацикливание. Сравнивая модели на рис. 9 и 11, видим, что во втором случае модель получилась заметно проще по структуре, однако более сложной для понимания, чтобы понять, какое изображение она возвращает клиенту в различных ситуациях. Недостаточно знать только текущие состояния, но необходимо учитывать праисторию смены состояний. Так, зная, что текущим является состояние «Аноним», нельзя узнать, выводится ли сообщение об ошибке (действие *10*), для этого необходимо знать, сработал ли переход *J8*.

3. БЕЗОПАСНОСТЬ ИНТЕРНЕТ-ПРИЛОЖЕНИЯ

Интернет-приложения считаются достаточно открытой зоной для атак со стороны злоумышленников. В этой связи одним из наиболее сложных и актуальных аспектов функционирования интернет-приложения является задача обеспечения информационной безопасности, к решению которой необходимо подходить еще на этапе разработки. Обеспечение безопасности разрабатываемого интернет-приложения представляет сложный комплексный процесс, сочетающий в себе набор действий по исследованию разрабатываемого интернет-приложения на предмет безопасности и последующего определения комплекса организационных, программных, аппаратных и криптографических методов, позволяющих в каждом конкретном случае обеспечить необходимый уровень защиты. Прежде чем приступить к построению интернет-приложения, необходимо оценить наиболее актуальные преднамеренные угрозы безопасности, результатом реализации которых может являться не только временное прекращение функционирования ресурса, но и потеря доверия пользователей к интернет-приложению. Каждый из перечисленных методов информационной безопасности может использоваться как самостоятельно, так и в интеграции с другими в целостный механизм. Это предоставляет возможность построения систем информационной защиты любой сложности и конфигурации, не зависящих от используемых платформ. Специфика построения интернет-приложений на основе встроенных динамических моделей обуславливает задачу разработки концепции информационной безопасности, охватывающей вопросы формулирования возможных угроз, а также разработку совокупности решений по их преодолению, необходимых для обеспечения бесперебойного информационного обеспечения бизнес-процесов. Согласно предлагаемой концепции, возникает угроза извлечения структуры и содержания как встроенной динамической модели, так и моделей памяти текущих состояний или их составных частей с последующим унич-

тожением, частичной и значительной модификацией или ознакомлением без нарушения целостного содержания. В случае модификации структуры или содержания динамически выполняемых элемент-фрагментов встроенной динамической модели в течение определенного временного промежутка осуществляется переадресация легитимных пользователей на источник, не имеющий отношения к изначальному запросу, что открывают широкие возможности для злоумышленников. Используя информацию, содержащуюся в моделях памяти текущих состояний, предоставляется возможность отслеживания пользовательских действий, а также осуществления сбора статистической информации посредством определения последовательности интернет-страниц, просмотренных в процессе сеанса до достижения желаемого результата. Вследствие этого возможно осуществить построение достаточно полной картины предпочтений, как конкретного пользователя, так и определенной пользовательской группы. В случае обнаружения повреждения или удаления содержимого модели, предполагается использование резервной копии ресурса, в качестве возможности восстановления работоспособности интернет-приложения. Для защиты моделей текущих состояний, предполагается использование алгоритмов криптографического преобразования и аутентификационных механизмов, позволяющих не только предупредить, но и минимизировать размер потерь обусловленных несанкционированный доступом к данным при передаче по сетевым протоколам. В связи с тем, что предварительно определить возможную совокупность угроз безопасности и результатов их реализации практически невозможно, осуществлять построение модели потенциальных угроз безопасности необходимо еще на этапе проектирования интернет-приложения, с последующим уточнением на этапе функционирования. В табл. 2 приведены основные типы угроз информационной безопасности и решения по их преодолению.

Таблица 2

Основные типы угроз информационной безопасности и решения по их преодолению

Основные типы угроз	Описание угроз	Методы противодействия
Отказ в обслуживании	Представляет попытку сделать недоступной определенную службу, что существенно замедляет работу интернет-приложения	Дублирование систем, которые продолжат обслуживать пользователей в случае, если некоторые элементы станут недоступны.
Фиксирование идентификатора сеанса	Получение действующего идентификатора сеанса.	Генерация нового идентификатора сеанса при каждом изменении пользовательских привилегий практически исключает риск фиксирования.
Похищения сеанса	Механизм перехвата содержимого сеанса связи отдельного пользователя с целью получения доступа к информации или услугам.	Передача идентификаторов сеансов только через файлы cookies, генерация дополнительной метки сеанса, передаваемой через единый указатель ресурсов.
Несанкционированный доступ от имени легитимного пользователя	Маскировка под легитимного пользователя с использованием имени и пароля данного пользователя или их эквивалентов.	Идентификаторы сеансов должны быть доступны в течение фиксированного времени, после чего пользователь снова должен выполнить аутентификацию повторно.
Отслеживание действий пользователя	Сбор статической информации и построение картины предпочтений пользователя или пользовательских групп.	Детально отображать историю и время нескольких предыдущих входов с межсетевыми протоколами и изменениями учетной записи.
Автоматическая регистрация	Заполнение регистрационных форм специализированными сервисами на основании базы каталогов.	Вывод дополнительного параметра, который необходимо ввести пользователю в определенное поле формы.
Инъекции	Встраивание деструктивного программного кода в запросы к базе данных с целью получения закрытой информации, обхода механизмов ограничения доступа или стандартной проверки авторизации.	Проверка параметров пользовательского ввода на соответствие ожидаемому типу или в принудительном порядке указание типа, используя учетную запись суперпользователя.

ЗАКЛЮЧЕНИЕ

В статье обсуждается новый класс интернет-приложений на основе встроенных динамических моделей, характеризующийся следующими особенностями:

- в составе серверного сценария предусматривается специальный объект – динамическая модель взаимодействия с клиентом, в котором заданы набор возможных состояний, переходов и прикладных фрагментов кода формирования изображения;
- формирование клиентской страницы, отсылаемой пользователю, основывается на принципе соответствия иерархической структуры изображения и структуры динамической модели;
- многопользовательский режим интернет-приложения обеспечивается за счет поддержания экземпляров памяти текущего состояния для каждого из пользователей: для незарегистрированных – в пределах сеанса, а для зарегистрированных – и между сеансами;
- для отслеживания текущих состояний динамической модели в ней могут предусматриваться промежуточные состояния обработки, предназначенные для выполнения соответствующих вычислений и проверок;
- в исходной динамической модели бизнес-процесса выделяются серверные подмодели, реализуемые на сервере, и клиентские, реализуемые в браузере клиента;
- смена изображения в браузере в ответ на смену текущего состояния динамической модели может выполняться как традиционным методом полной перегрузки страницы, так и путем частичной перегрузки (на основе технологии Ajax) фрагментов изображения, соответствующих новому состоянию;
- формирование кода изображения может осуществляться путем одно- или двухпроходной интерпретации динамической модели. В первом случае динамические модели могут получаться менее наглядными, во втором – более громоздкими.

СПИСОК ЛИТЕРАТУРЫ

1. **Миронов В. В., Юсупова Н. И., Ильясов Б. Г.** Иерархические модели процессов управления: описание, интерпретация и лингвистическое обеспечение. Уфа : УГАТУ, 1994. 152 с.

2. **Миронов В. В., Шакирова Г. Р.** Концепция динамических XML-документов // Вестник УГАТУ : научн. журн. Уфимск. гос. авиац. техн. ун-та. 2006. Т. 8. № 5. С. 58–63.

3. **Миронов В. В., Ахметшин Р. Ф.** Асинхронная децентрализованная интерпретация иерархических ситуационных моделей // Там же. 2003. Т. 4. № 1. С. 108.

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

5. **Миронов В. В., Шакирова Г. Р., Яфаев В. Э.** Информационная технология персонализации электронных документов Microsoft Office в web-среде на основе XML // Там же. 2008. Т. 10. № 2. С. 112–122.

6. **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.

7. **Миронов В. В., Шакирова Г. Р., Яфаев В. Э.** Иерархическая модель персонализованных документов и ее XML-реализация // Там же. 2008. Т. 11. № 1. С. 164–174.

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

9. **Миронов В. В., Шакирова Г. Р.** Электронные документы со встроенной динамической моделью на основе XML: монография. 2009. 179 с.

10. **Кристиан Д., Бринзаре Б., Черчез-Тоза Ф.** Ajax и PHP. Разработка динамических веб-приложений. СПб. : Символ-Плюс, 2007. 336 с.

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. сист. (УГАТУ, 1995). Иссл. в обл. иерархич. 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.