ISSN 1992-6502 (Print) Вестник УГАТУ ISSN 2225-2789 (Online)
2014. Т. 18, № 1 (62). С. 204–213 http://journal.ugatu.ac.ru

УДК 004.65+004.738.5

ИЕРАРХИЧЕСКИЕ ВИДЖЕТЫ: АЛГОРИТМЫ КОНТРОЛЯ ДАННЫХ ПОЛЬЗОВАТЕЛЯ В ВЕБ-ПРИЛОЖЕНИЯХ НА ОСНОВЕ СИТУАЦИОННО-ОРИЕНТИРОВАННЫХ БАЗ ДАННЫХ

В. В. Канашин[1], **В. В. Миронов**[2]

[1] vitas.k@rambler.ru, [2] mironov@list.ru

ФГБОУ ВПО «Уфимский государственный авиационный технический университет» (УГАТУ)

Поступила в редакцию 12.12.2013

Аннотация. Рассматривается алгоритмическое обеспечение сложно-структурированного интерфейса пользователя в веб-приложениях, функционирующих на основе ситуационно-ориентированных баз данных (СОБД). В рамках концепции иерархических виджетов, разработанной в предыдущих статьях авторов, рассматриваются алгоритмы организации ввода и контроля пользовательских данных. Описываются алгоритмы: элемента-контролёра для контроля данных, вводимых пользователем; элементов-приемников для помещения введенных данных в DOM-буфер и фиксации выявленных ошибок; элементов-переходов, активность которых зависит от наличия / отсутствия выявленных ошибок. Использование правил умолчания, предусмотренных в алгоритмах, позволяет заметно сократить объем программного кода. Алгоритмы реализованы в составе интерпретатора динамических моделей HSM, функционирующего на платформе PHP.

Ключевые слова: веб-приложение; интерфейс пользователя; ситуационно-ориентированная база данных; динамическая модель; иерархические виджеты; пользовательские данные; регулярные выражения; HSM; XML; XSLT; model-driven development.

ВЕДЕНИЕ

Данная работа связана с ситуационно-ориентированными базами данных (СОБД) [1, 2]. Развитие СОБД идет в различных направлениях: веб-приложения на основе СОБД [3–5]; обработка XML-данных [6, 7]; OLAP-аналитика [8, 9]; формирование интерфейсов пользователя [10]. В предыдущих статьях [10, 11] авторами был предложен и исследован подход к созданию сложно структурированных пользовательских интерфейсов, основанных на иерархических виджетах. Концепция иерархических виджетов состоит в том, что в состояниях динамической модели предусматриваются виджет-элементы, соответствующие фрагментам изображения на экране пользователя и задающие способ и параметры формирования результирующего HTML-кода, а также ссылки на родительские виджеты, объединяющие виджет-элементы в иерархию. В процессе интерпретации динамической модели в зависимости от ее текущего состояния автоматически формируется результирующий контент иерархии виджет-элементов, который по завершении выводится в выходной информационный поток. Тем самым разработчикам СОБД предоставляется инструмент декларативного описания структуры пользовательского интерфейса в зависимости от текущих состояний динамической модели.

В работе [10] были рассмотрены вопросы вывода данных для отображения пользователю, а в работе [11] – вопрос о том, как в рамках данной концепции организовать ввод и контроль данных, которые пользователь передает серверу через интерфейс, предоставляемый виджетами.

В работе [11] в рамках концепции иерархических виджетов предложены новые элементы динамической модели HSM, предназначенные для контроля данных, вводимых пользователем: элемент-контролёр, указывающий контролируемый элемент пользовательских данных, а также входящие в его состав элемент-приемник для помещения введенных данных в буферный DOM-объект, и элемент-фиксатор ошибок, предназначенный для выявления и записи ошибок в данных. Это позволяет декларативно задать в динамической модели, какие входные данные нужно контролировать, каким проверкам их следует подвергнуть, куда записать све-

Работа поддержана грантом РФФИ 13-07-00011.

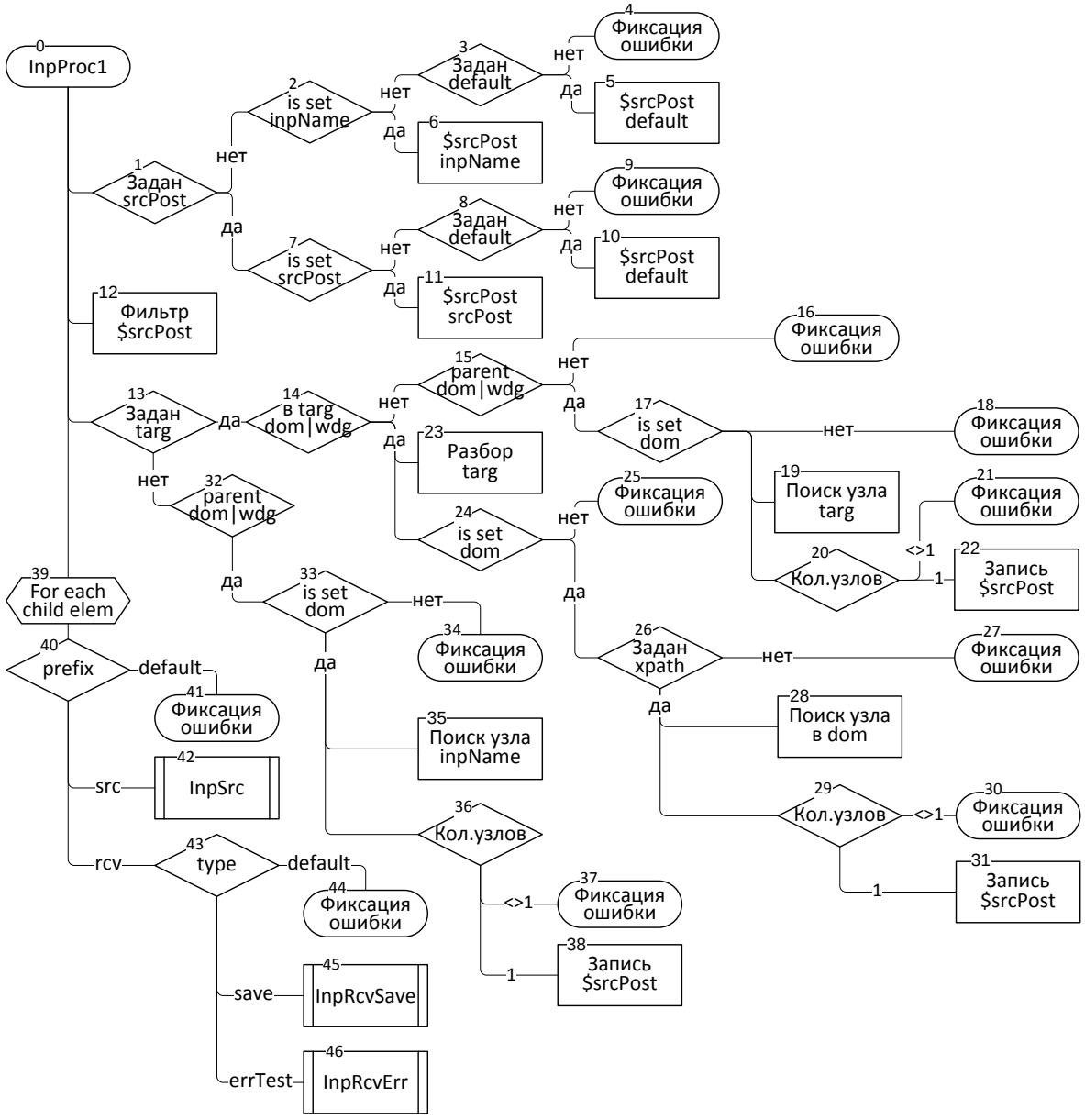

Рис. 1. Схема алгоритма интерпретации элемента-контролёра

дения об обнаруженных ошибках, в какое состояние переходить при обнаружении ошибок или при их отсутствии.

В данной работе рассматривается алгоритмическое обеспечение функций контроля входных данных в виде алгоритмов обработки указанных элементов в процессе интерпретации динамической модели HSM:

- алгоритм интерпретации элемента-контролёра;
- алгоритмы тестирования данных, выявления и фиксации ошибок;
- алгоритм управления активностью переходов в зависимости от выявленных ошибок.

Алгоритм интерпретации элемента-контролёра

На рис. 1 приведена схема алгоритма интерпретации элемента-контролёра, представленная в «иерархической» нотации[1]. Алгоритм оформлен в виде процедуры-функции InpProc1, кото-

[1] Особенности используемого «иерархического» способа задания схемы [13]: соединительные линии, входящие в верхнюю или левую стороны блока, связывают его с родительским блоком, а исходящие из правой или нижней сторон – с дочерними блоками. Такая нотация в большей степени соответствует блочной структуре современных процедурных языков высокого уровня без оператора «go to».

рой в качестве входного параметра передается ссылка на обрабатываемый элемент-контролёр.

Извлечение проверяемых данных из входного массива Post выполняется с помощью блоков 1–11. Блок 1 проверяет, задан ли в обрабатываемом контролёре атрибут srcPost, содержащий явное имя проверяемых данных. Если srcPost не задан, то в качестве неявного имени проверяемых данных берется имя обрабатываемого контролёра и проверяется (блок 2), имеется ли в массиве Post элемент с таким именем. При отсутствии такого элемента проверяется (блок 3), задан ли в контролёре атрибут default (значение по умолчанию), и в случае его отсутствия фиксируется ошибка (блок 4)[2], а в случае наличия – используется значение по умолчанию (блок 5), которое заносится в переменную $srcPost. В случае обнаружения блоком 2 соответствующего элемента, его значение заносится в переменную $srcPost (блок 6).

В случае если блок 1 обнаруживает атрибут srcPost, значение атрибута используется в качестве явного имени проверяемого элемента (блоки 7–11): проверяется (блок 7), имеется ли в массиве Post элемент с таким именем; при отсутствии такого элемента проверяется (блок 8), задан ли в контролёре атрибут default, и в случае его отсутствия фиксируется ошибка (блок 9), а в случае наличия – используется значение по умолчанию (блок 10), которое заносится в переменную $srcPost; в случае обнаружения блоком 7 соответствующего элемента, его значение заносится в переменную $srcPost (блок 11).

Таким образом, по завершении обработки блока 1 переменной $scrPost присваивается проверяемое контролёром значение (при отсутствии ошибок) или происходит ошибочное завершение функции (при обнаружении ошибки).

Очистка проверяемых данных выполняется в блоке 12. У значения переменной $scrPost убираются возможные концевые пробелы и экранируются специальные символы для блокирования хакерских инъекций в данных[3].

Сохранение проверяемого значения в DOM-буфере выполняется с помощью блоков 13–38 по-разному в зависимости от того, задан ли в контролёре явно DOM-буфер и существует ли он. В блоке 13 проверяется, задан ли в контролере атрибут targ, с помощью которого можно указать 1) DOM-буфер (в виде имени dom- или wdg-элемента) и 2) место в нем для сохранения данных (в виде XPath-выражения, задающего узел в DOM-объекте, в котором нужно сохранить проверяемое значение). Если атрибут targ задан, но при этом его значение не равно «off» – «отключить», то в блоке 14 выявляется, содержит ли значение targ ссылку на DOM-объект. Если такой ссылки нет, то блок 15 проверяет, является ли непосредственным родителем обрабатываемого контролёра dom- или wdg-элемент, и если нет – фиксируется ошибка (блок 16).

Если блок 15 выявляет, что контролёр размещен непосредственно в dom- или wdg-элементе, то применяется правило умолчания – проверяемые данные сохраняются в соответствующем родительском DOM-объекте. В этом случае блок 17 проверяет, существует ли такой DOM-объект, и если нет – фиксируется ошибка (блок 18), а если да – выполняется поиск узла в DOM-объекте согласно значению targ, рассматриваемому в данном случае как XPath-адрес (блок 19). Блок 20 проверяет количество найденных узлов: если узлов не найдено или найдено несколько, фиксируется ошибка (блок 21), если же найден ровно один узел – в него записывается контролируемое значение из переменной $srcPost (блок 22).

Если блок 14 выявил, что атрибут targ содержит ссылку на DOM-объект, то с помощью блока 23 выполняется синтаксический разбор значения targ, и из него вычленяются имя DOM-объекта и XPath-адрес узла в нем. Далее блок 24 проверяет существование заданного в targ DOM-объекта, в случае отсутствия фиксируется ошибка (блок 25), а в случае присутствия – проверяется наличие в targ XPath-адреса узла (блок 26). При отсутствии XPath-адреса фиксируется ошибка (блок 27), а при наличии – с помощью блоков 28–31 выполняются действия по сохранению значения $srcPost аналогично тому, как это делалось с помощью блоков 19–22.

[2] Отметим, что фиксация ошибки в блоке 4, как и в других аналогичных блоках 9, 16, 21 и т. д., – это не фиксация ошибок, обнаруженных в проверяемых контролёром данных. Здесь имеются в виду ошибки, связанные с неправильным заданием элементов динамической модели, когда, например, в контролёре обнаружена ссылка на dom-элемент, который не был ранее задан. Фиксация этих ошибок предполагает: 1) занесение информации об ошибке в глобальный массив ошибок ERRs; 2) вывод сообщения об ошибке; 3) аварийное завершение функции командой return false.

[3] В дальнейшем предполагается предусмотреть возможность управления процессом фильтрации с помощью соответствующих атрибутов элемента-контролёра.

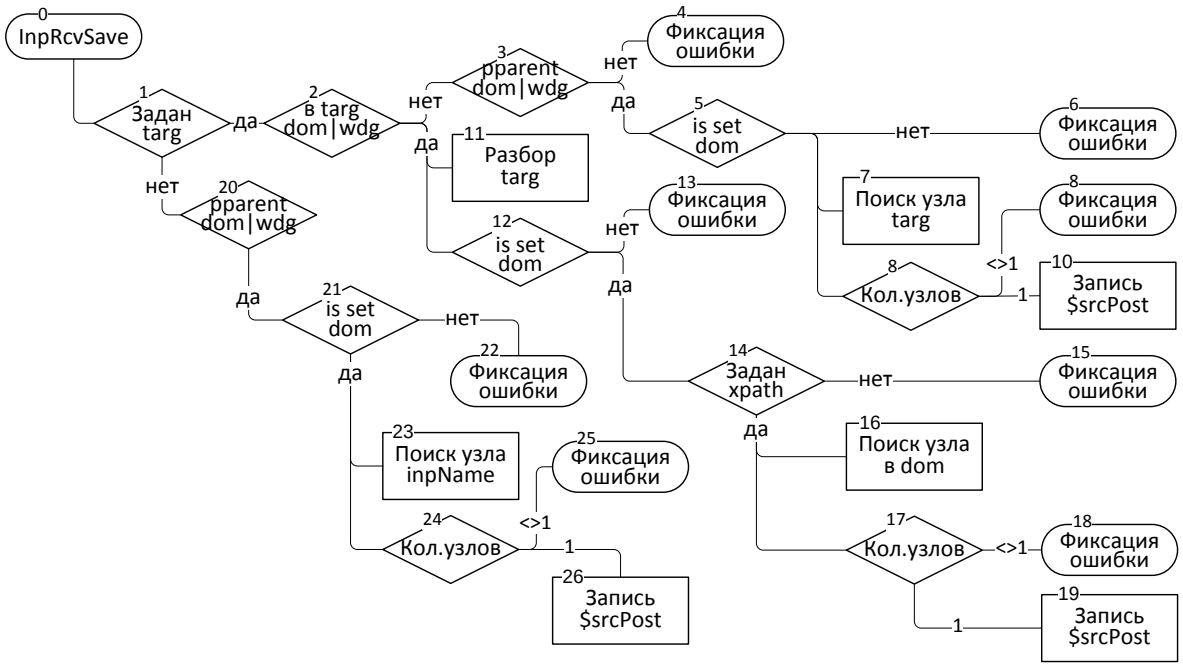

Рис. 2. Схема алгоритма интерпретации элемента-приемника для сохранения данных в DOM-буфере

Если блок 13 выявил, что атрибут targ не задан, делается попытка применить правило поиска DOM-объекта по умолчанию. В этом случае считается, что если контролёр без атрибута targ размещен непосредственно в dom- или wdg-элементе, то проверяемые данные следует сохранить в DOM-объекте родителя на втором уровне иерархии в элементе, имя которого совпадает с именем элемента-контролёра. Эти действия выполняют блоки 33–38, аналогичные блокам 15, 17–22. Заметим, что при этом в блоке 35 ищется узел – непосредственный потомок корневого узла DOM-объекта.

Отметим также, что в ситуации, когда не задан атрибут targ и контролер не размещен в dom- или wdg-элементе, как и в ситуации, когда атрибут targ задан, но имеет значение «off», значение в DOM-буфере не сохраняется.

Обработка внутренних элементов. Внутри элемента-контролера могут быть заданы элементы-источники и элементы-приемники данных. Блок 39 выполняет циклический перебор дочерних элементов обрабатываемого контролёра. Внутренний блок 40 селектирует префиксы дочерних элементов: непредусмотренный префикс (default) вызывает фиксацию ошибки (блок 41), префикс src приводит к вызову функции InpSrc (блок 42) для обработки элемента-источника данных, а префикс rcv – к блокам 43–46 для обработки элемента-приемника данных. В последнем случае блок 43 селектирует тип элемента-приемника, задаваемый атрибутом type, для неопознанного типа фиксируется ошибка (блок 44), для type = "save" вызывается функция InpRcvSave обработки приемника сохранения данных (блок 45), а для type = "errTest" – функция InpRcvErr обработки приемника тестирования ошибок в данных (блок 46).

Функция InpSrc предназначена для обработки источников данных контролёра, задающих проверяемые контролёром внешние данные. В простейшем случае алгоритм этой функции аналогичен начальному фрагменту алгоритма функции InpProc1 (фрагмент «извлечение проверяемых данных из входного массива Post», см. рис. 1, блоки 1–11) и здесь не рассматривается.

Функция InpRcvSave предназначена для обработки элемента-приемника сохранения данных. Схема алгоритма этой функции приведена на рис. 2. Из схемы видно, что она аналогична фрагменту схемы алгоритма InpProc1 (фрагмент «сохранение проверяемого значения в DOM-буфере», см. рис. 1, блоки 11–38). Отличие состоит в том, что элементы-приемники вложены в элемент-контролёр и если контролёр, в свою очередь, вложен в dom- или wdg-элемент, то последний является для элемента-приемника «дедом», а не родителем. Поэтому в блоках 3 и 20 на рис. 2 проверяется «родитель родителя», а не просто «родитель», как в блоках 15 и 32 на рис. 1.

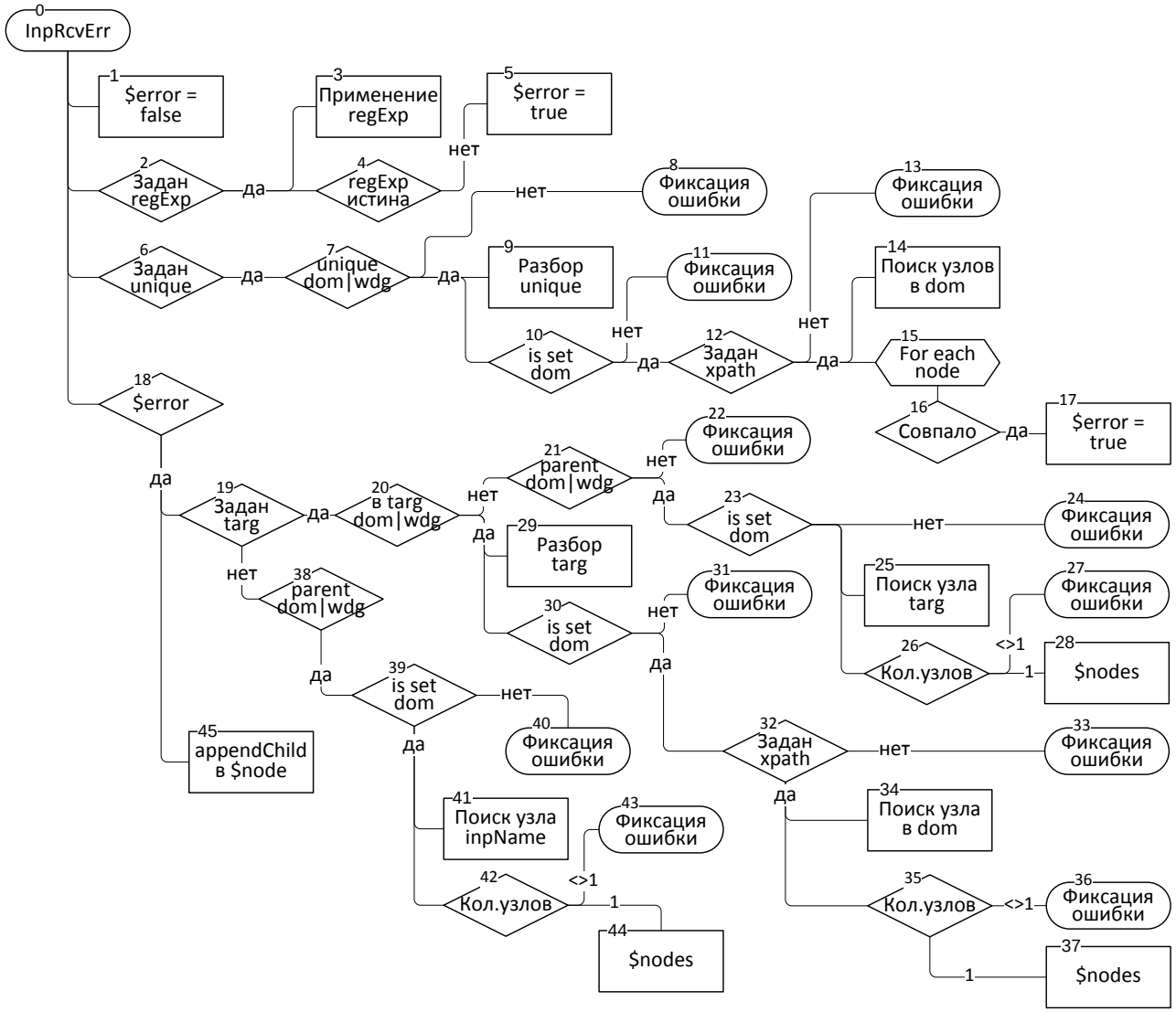

Рис. 3. Схема алгоритма тестирования ошибок в данных

Таким образом, функциональность элемента-приемника типа «save» дублирует одну из функций элемента-контролёра. Такое дублирование предусмотрено на тот случай, когда экземпляры проверяемых данных требуется сохранить в нескольких местах (в различных DOM-буферах или в различных узлах одного DOM-буфера).

Алгоритмы тестирования данных, выявления и фиксации ошибок

Тестирование ошибок в данных и запись информации об обнаруженных ошибках в DOM-буфер выполняется с помощью элементов-приемников типа «errTest», размещенных внутри элемента-контролёра. Обработка таких приемников выполняется функцией InpRcvErr, схема которой приведена на рис. 3. При вызове функции ей в качестве входного параметра передается, в частности, указатель на обрабатываемый элемент-приемник. Факт обнаружения ошибок отмечается с помощью переменной-флага $error, которой при инициализации присваивается значение «ложь» (блок 1).

Тестирование регулярным выражением обеспечивают блоки 2–5. Блок 2 проверяет наличие в приемнике атрибута regExp, содержащего регулярное выражение тестирования данных. Регулярное выражение применяется к проверяемым данным (блок 3), правильные данные должны давать результат «истина» (блок 4). Если результат «ложь» (ошибка в данных), переменной $error присваивается значение «истина» (блок 5).

Тестирование на уникальность обеспечивают блоки 6–17. Блок 6 проверяет наличие

в приемнике атрибута unique, задающего множество значений, ни одно из которых не должно совпадать с проверяемыми данными. Это множество задается в виде множества узлов некоторого DOM-объекта – атрибут unique содержит ссылку на dom- или wdg-элемент, а также XPath-адрес узлов в соответствующем DOM-объекте; синтаксис атрибута unique совпадает с синтаксисом атрибута targ элемента-контролёра с той разницей, что здесь XPath-выражение может адресовать несколько узлов.

В блоке 7 проверяется, что атрибут unique ссылается на dom- или wdg-элемент, в противном случае фиксируется ошибка (блок 8). В блоке 9 выполняется синтаксический разбор атрибута unique, а в блоке 10 проверяется существование DOM-объекта, при отсутствии фиксируется ошибка (блок 11). Блок 12 проверяет наличие XPath-выражения для поиска узлов, при его отсутствии фиксируется ошибка (блок 13). Блок 14 отыскивает множество узлов, а блок цикла 15 перебирает найденные узлы. В ходе перебора текстовое значение узла сравнивается с проверяемым значением (блок 16) и в случае совпадения устанавливается флаг ошибки (блок 17).

Запись информации об обнаруженных ошибках (фиксация ошибок) осуществляется блоками 18–45, при этом блок 18 запускает процедуру только при установленном флаге ошибки $error.

Информация об ошибке записывается в DOM-объект в виде специального узла-элемента, при этом DOM-объект и адрес в нем задаются с помощью атрибута targ, аналогичному одноименному атрибуту элемента-контролёра. В связи с этим блоки 19–44 на рис. 3, выполняющие обработку атрибута targ, аналогичны блокам 13–38 на рис. 1. Отличие лишь в том, что блоки 28, 37 и 44 запоминают ссылку на найденный узел в переменной $node, а не изменяют его текстовое значение.

В случае успешной обработки атрибута targ блок 45 формирует XML-элемент ошибки со сведениями об обнаруженной ошибке и заносит его в DOM-объект в качестве дочернего узла по отношению к найденному узлу $node. Имя элемента ошибки задается атрибутом errElement (по умолчанию, при отсутствии у элемента-приемника атрибута errElement, присваивается имя «error»). Элемент ошибки содержит три атрибута:

- field – локальное имя элемента-контролёра, которому принадлежит элемент-приемник;
- test – локальное имя элемента-приемника, обнаружившего ошибку;
- code – код ошибки – определяется значением атрибута errCode (по умолчанию, при отсутствии у элемента-приемника атрибута errCode, присваивается значение «error»).

Управление активностью перехода в зависимости от ошибок

Чтобы воспользоваться сведениями об обнаруженных контролёром ошибках, необходимо иметь возможность управлять активностью элементов-переходов jmp (или элементов-акций act) динамической модели в зависимости от наличия или отсутствия сведений об ошибках, обнаруженных контролёрами. Поскольку сведения об обнаруженных ошибках размещаются в DOM-объекте в виде нового узла, то данную задачу можно трактовать шире: как возможность управления активностью элементов динамической модели в зависимости от наличия или отсутствия определенных узлов в определенном DOM-объекте.

Кроме того, возможны ошибки, связанные с некорректным заданием динамической модели, которые фиксируются в специальном глобальном массиве ошибок при аварийном завершении алгоритмов. Поэтому необходимо иметь возможность управлять активностью элементов-переходов или элементов-акций динамической модели в зависимости от наличия или отсутствия сведений об ошибках этого рода.

Алгоритм проверки DOM-объекта. На рис. 4 представлена схема алгоритма проверки отсутствия определенных узлов в DOM-объекте. Алгоритм применяется как фрагмент в общем алгоритме InterProc1 для определения активности переходов и акций при наличии в них атрибута ctrlNotExists.

Атрибут ctrlNotExists задает проверяемый DOM-объект и адрес узлов в нем аналогично тому, как это делается в атрибуте targ элемента-контролёра. Блок 1 проверяет, что атрибут ctrlNotExists содержит ссылку на dom- или wdg-элемент и в этом случае блок 2 выполняет синтаксический разбор значения атрибута, вычленяя имя DOM-объекта и XPath-адрес искомых узлов в нем. Далее блок 3 проверяет существование DOM-объекта и в случае его отсутствия блок 4 фиксирует ошибку. Аналогично блок 5 проверяет наличие XPath-выражения и в случае его отсутствия блок 6 фиксирует ошибку. Блок 7 выполняет поиск в DOM-объекте узлов, удовлетворяющих XPath-выражению, а блок 8 проверяет количество найденных узлов. В слу-

чае если найден хотя бы один узел, блок 9 завершает алгоритм возвратом return false с результатом «ложь».

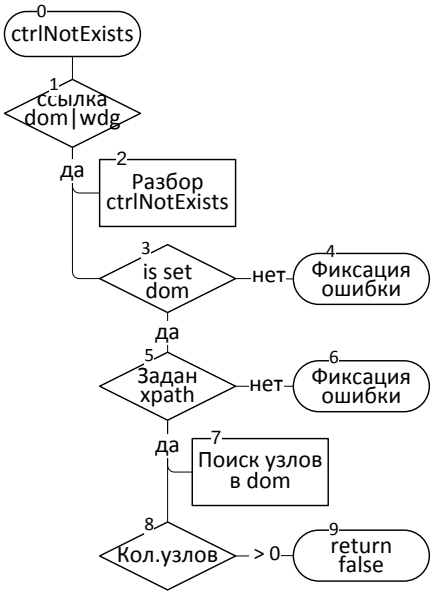

Рис. 4. Схема алгоритма проверки отсутствия определенных узлов в DOM-объекте

Отметим, что при фиксации ошибки алгоритм также завершается возвратом return false. Поскольку алгоритм ctrlNotExists встроен в охватывающий алгоритм InterProc1 как простой фрагмент (не как процедура или функция), это приводит к завершению алгоритма InterProc1 возвратом return false, что означает пассивность обрабатываемого элемента-перехода или элемента-акции. Если же алгоритм ctrlNotExists завершается нормально, охватывающий алгоритм InterProc1 продолжает работу, проверяя другие условия активности.

Алгоритм проверки глобального массива ошибок. На рис. 5 представлена схема алгоритма проверки отсутствия записей в глобальном массиве ошибок. Как и предыдущий, этот алгоритм применяется как фрагмент в общем алгоритме InterProc1 для определения активности переходов и акций при наличии в них атрибута ctrlNoErrs = "any".

Блок 1 проверяет наличие у элемента перехода или акции атрибута ctrlNoErrs и его значение. Если значение атрибута "any" («любой»), блок 2 подсчитывает число строк в глобальном массиве ошибок ERRs. В случае если число строк больше нуля (зафиксирована хотя бы одна ошибка), алгоритм также завершается возвратом return false, обеспечивая пассивность соответствующего перехода или акции.

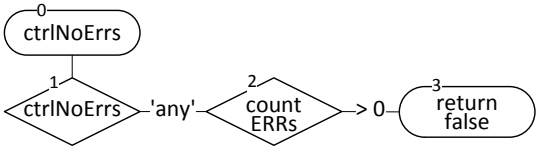

Рис. 5. Схема алгоритма проверки отсутствия записей в глобальном массиве ошибок ERRs

Учет ошибок в динамической модели. Итак, представленные выше алгоритмы контроля данных, поступивших от пользователя, фиксируют факт наличия ошибки в данных в DOM-буфере. Убедившись в отсутствии сведений об ошибках в DOM-буфере, введенные пользователем данные можно направить на дальнейшую обработку, например, сохранить в базе. Важная особенность состоит в том, что в случае обнаружения ошибок в спецификациях проверки данных и аварийного завершения алгоритма, интерпретатор не может отразить это обстоятельство в DOM-буфере и проверка DOM-буфера не даст указания на некорректность данных. В этом случае, хотя и будет выдано сообщение об ошибке, некорректные данные пользователя могут быть записаны в базу данных, что весьма нежелательно.

Возможный выход из этой ситуации состоит в том, чтобы наряду с отсутствием ошибок в DOM-буфере контролировать их отсутствие в глобальном массиве ошибок ERRs. Например, следующий элемент-переход:

```
<jmp:Сохранить
    ctrlNotExists = "dom:Буфер#/студент/error"
    ctrlNoErrs = "any"/>
```

обеспечивает переход в состояние sta:Сохранить, в котором пользовательские данные сохраняются в базе, во-первых, если буферный DOM-объект dom:Буфер не содержит узлов по XPath-адресу "/студент/error", во-вторых, если нет записей в глобальном массиве ошибок.

Другой вариант – предусмотреть в интерпретаторе HSMI автоматическую блокировку активности акций и переходов и выдачу соответствующих сообщений при наличии ошибок в глобальном массиве ERRs.

Использование правил умолчания при задании DOM-объектов в контролёрах

Представленные выше алгоритмы InpProc1, InpRcvSave, InpRcvErr предусматривают в атрибутах targ контролёров и вложенных в них приемников возможность умолчания при задании

информации о DOM-объектах с учетом контекста. Это позволяет заметно сократить объем спецификаций, когда контролёры размещаются внутри виджет-элементов.

На рис. 6 в качестве примера приведены две функционально эквивалентные модели проверки входных данных пользователя о новом студенте. В состоянии sta:НовыйСтудент пользователю отображается бланк для ввода сведений о новом студенте, введенные пользователем данные (код и фамилия студента) проверяются, при обнаружении ошибок бланк с ошибочными данными повторно отображается, а при отсутствии ошибок – выполняется переход в состояние sta:Сохранение для сохранения данных о новом студенте.

В первом варианте модели (рис. 6, *а*) явно задан dom-элемент dom:Студ-Buf для создания соответствующего DOM-буфера. Виджет-элемент wdg:НовСтуд, обеспечивающий отображение данных пользователю, ссылается на этот dom-элемент. Контролёры inp:код и inp:фам размещены в модели как дочерние элементы состояния, для указания DOM-буфера (в них и во вложенных элементах-приемниках) приходится полностью прописывать атрибут targ. Кроме того, в приемниках-фиксаторах ошибок явно указаны имена элементов и коды ошибок.

Во втором варианте модели (рис. 6, *б*) DOM-буфер задан неявно в виджет-элементе, что позволило обойтись без dom-элемента. Контролёры размещены внутри виджет-элемента, что позволило использовать правила умолчания и не ссылаться явно на DOM-буфер. Далее здесь учтено, что проверяемые элементы размещены в DOM-буфере как дети корневого элемента, и это позволило обойтись без соответствующих XPath-адресов. В результате это позволило вовсе обойтись без атрибутов targ. Кроме того, использованы умолчания для элементов и кодов ошибок.

В итоге второй вариант модели оказался заметно меньшего объема, чем первый. Количество символов текста модели сократилось с 725 до 446 (более чем в 1,6 раза).

Программная реализация алгоритмов

Разработанные алгоритмы реализованы в составе интерпретатора динамических моделей HSMI, функционирующего на платформе PHP.

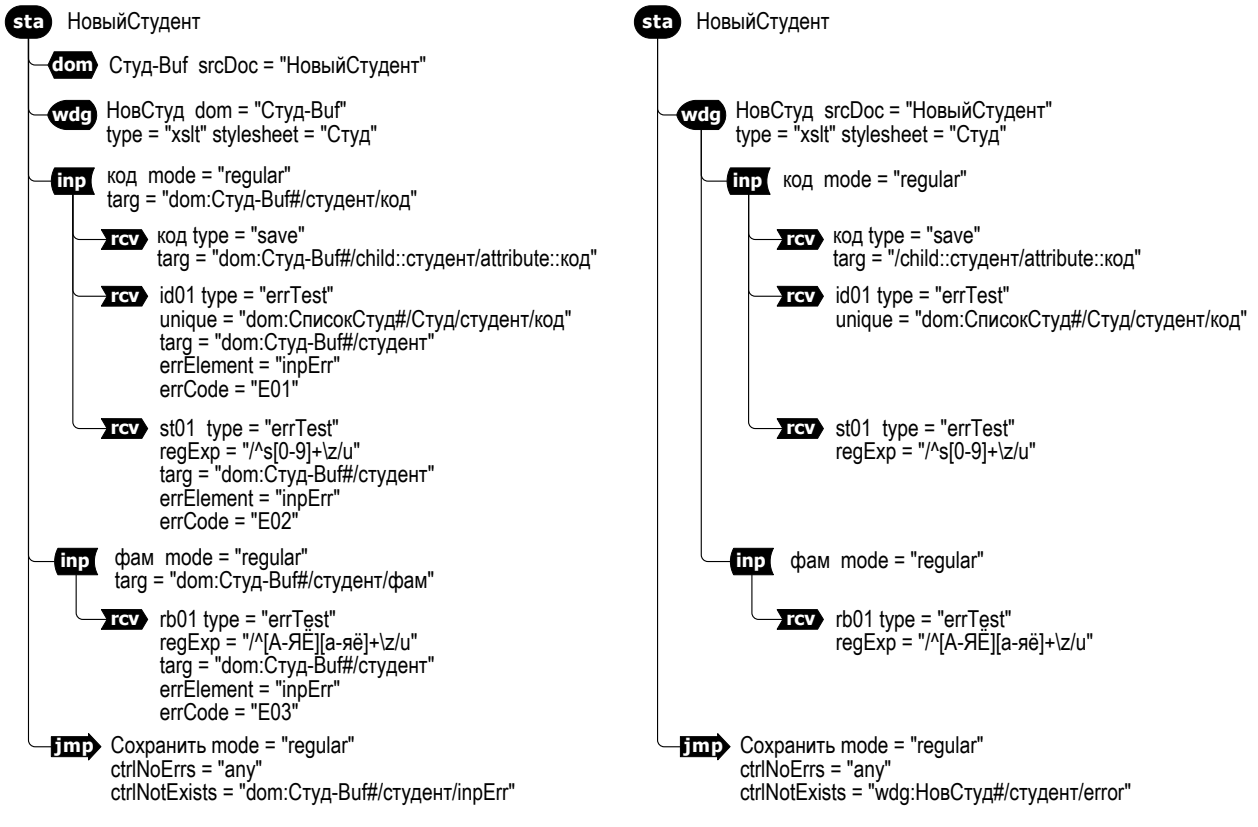

Рис. 6. К использованию правил умолчания при задании DOM-объектов в контролёрах:
а – без использования; *б* – с использованием

Работоспособность и корректность алгоритмов подтверждена на серии тестовых примеров. В настоящее время основанные на алгоритмах средства динамической модели HSM используются при создании реального веб-приложения.

ЗАКЛЮЧЕНИЕ

В данной статье в рамках концепции иерархических виджетов разработано алгоритмическое обеспечение новых элементов динамической модели HSM: 1) элемента-контролёра для контроля данных, вводимых пользователем; 2) входящих в его состав элементов-приемников для помещения введенных данных в DOM-буфер и фиксации выявленных ошибок; 3) элементов-переходов, активность которых зависит от наличия / отсутствия выявленных ошибок.

Реализация функциональности алгоритмов имеет следующие особенности:

- для компактности модели DOM-буфер и XPath-адрес узла в нем задаются совместно в одном атрибуте и вычленяются алгоритмом путем синтаксического разбора;
- для имен DOM-объектов, адресов узлов, имен и кодов ошибок и др. предусмотрены правила умолчания, действующие, когда элементы-контролёры размещены внутри dom- или виджет-элементов;
- в алгоритмах предусмотрено выявление ошибок в спецификациях проверки данных и запись их в глобальном массиве ошибок перед аварийным завершением;
- проверка наличия / отсутствия ошибок, выявленных контролёрами, реализована как частный случай проверки наличия / отсутствия определенных узлов в определенном DOM-объекте.

Анализ алгоритмов выявил, что при аварийном завершении в базу данных могут быть ошибочно занесены некорректные пользовательские данные. Для предотвращения этого предложено в элементах-переходах дополнительно проверять отсутствие записей в глобальном массиве ошибок.

Показано, что использование правил умолчания при размещение контролёров внутри виджетов позволяет заметно сократить объем программного кода. В рассмотренном примере объем кода уменьшился более чем в 1,6 раза.

Разработанные алгоритмы реализованы в составе интерпретатора динамических моделей HSMI, функционирующего на платформе PHP, и готовы к практическому использованию при создании веб-приложений.

СПИСОК ЛИТЕРАТУРЫ

1. **Миронов В. В., Юсупова Н. И., Шакирова Г. Р.** Ситуационно-ориентированные базы данных: концепция, архитектура, XML-реализация // Вестник УГАТУ. 2010. Т. 14, № 2 (37). С. 233–244. **[** V. V. Mironov, N. I. Yusupova, and G. R. Shakirova, “Situation-oriented databases: concept, architecture, XML realization,” (in Russian), *Vestnik UGATU*, vol. 14, no. 4 (39), pp. 200-209, 2010. **]**

2. **Миронов В. В., Юсупова Н. И., Шакирова Г. Р.** Ситуационно-ориентированные базы данных: внешние представления на основе XSL // Вестник УГАТУ. 2010. Т. 14, № 4 (39). С. 200–209. **[** V. V. Mironov, N. I. Yusupova, and G. R. Shakirova, “Situation-oriented databases: external view in the basis of XSL,” (in Russian), *Vestnik UGATU*, vol. 14, no. 2 (37), pp. 233-244, 2010. **]**

3. **Миронов В. В., Маликова К. Э.** Интернет-приложения на основе встроенных динамических моделей: идея, концепция, безопасность // Вестник УГАТУ. 2009. Т. 13, № 2 (35). С. 167–179. **[** V. V. Mironov and K. E. Malikova, “Internet applications based on embedded dynamic models: idea, concept,” (in Russian), *Vestnik UGATU*, vol. 13, no. 2 (35), pp. 167-179, 2009. **]**

4. **Миронов В. В., Маликова К. Э.** Интернет-приложения на основе встроенных динамических моделей: архитектура, структура данных, интерпретация // Вестник УГАТУ. 2010. Т. 14, № 1 (36). С. 154–163. **[** V. V. Mironov and K. E. Malikova, “Internet applications based on embedded dynamic models: architecture, data structure, interpretation,” (in Russian), *Vestnik UGATU*, vol. 14, no. 1 (36), pp. 154-163, 2010. **]**

5. **Миронов В. В., Маликова К. Э.** Интернет-приложения на основе встроенных динамических моделей: элементы управления пользовательского интерфейса // Вестник УГАТУ. 2010. Т. 14, № 5 (40). С. 170–175. **[** V. V. Mironov and K. E. Malikova, “Internet applications based on embedded dynamic models: user interface controls,” (in Russian), *Vestnik UGATU*, vol. 14, no. 5 (40), pp. 170-175, 2010. **]**

6. **Миронов В. В., Гусаренко А. С.** Ситуационно-ориентированные базы данных: концепция управления xml-данными на основе динамических dom-объектов // Вестник УГАТУ. 2012. Т. 16, № 3 (48). С. 159–172. **[** V. V. Mironov and A. S. Gusarenko, “Situation-oriented databases: concept of XML data management based of dynamic DOM objects,” (in Russian), *Vestnik UGATU*, vol. 16, no. 3 (48), pp. 159-172, 2012. **]**

7. **Гусаренко А. С., Миронов В. В.** Динамические dom-объекты в ситуационно-ориентированных базах данных: лингвистическое и алгоритмическое обеспечение источников данных // Вестник УГАТУ. 2012. Т. 16, № 6 (51). С. 167–176. **[** A. S. Gusarenko and V. V. Mironov, “Dynamic DOM objects in situation-oriented databases: lingware and knoware of data sources,” (in Russian), *Vestnik UGATU*, vol. 16, no. 6 (51), pp. 176-167, 2012. **]**

8. **Макарова Е. С., Миронов В. В.** Проектирование концептуальной модели данных для задач Web-OLAP на основе ситуационно-ориентированной базы данных // Вестник УГАТУ. 2012. Т. 16, № 6 (51). С. 177–188. **[** E. S. Makarova and V. V. Mironov, “Web OLAP conceptual data model design on the basis of situation-oriented database,” (in Russian), *Vestnik UGATU*, vol. 16, no. 6 (51), pp. 177-188, 2012. **]**

9. **Макарова Е. С., Миронов В. В.** Функции аналитики в веб-приложениях на основе ситуационно-

ориентированных баз данных // Вестник УГАТУ. 2013. Т. 17, № 5 (58). С. 150–165. **[** E. S. Makarova and V. V. Mironov, “Analytical functions in web applications based on situation-oriented databases,” (in Russian), *Vestnik UGATU*, vol. 17, no. 5 (58), pp. 150-165, 2013. **]**

10. **Канашин В. В., Миронов В. В.** Иерархические виджеты: организация интерфейса пользователя в веб-приложениях на основе ситуационно-ориентированных баз данных // Вестник УГАТУ. 2013. Т. 17, № 2 (55). С. 138–149. **[** V. V. Kanashin and V. V. Mironov, “Hierarchical widgets: user interface organization in web applications based on situation-oriented databases,” (in Russian), *Vestnik UGATU*, vol. 17, no. 2 (55), pp. 138-149, 2013. **]**

11. **Канашин В. В., Миронов В. В.** Иерархические виджеты: ввод и контроль данных пользователя в веб-приложениях на основе ситуационно-ориентированных баз данных // Вестник УГАТУ. 2013. Т. 17, № 5 (58). С. 166–176. **[** V. V. Kanashin and V. V. Mironov, “Hierarchical widgets: input and control of user data in web applications on the basis of situation-oriented databases,” (in Russian), *Vestnik UGATU*, vol. 17, no. 5 (58), pp. 166-176, 2013. **]**

12. **Миронов В. В., Юсупова Н. И., Шакирова Г. Р.** Иерархические модели данных: концепции и реализация на основе XML. М.: Машиностроение, 2011. 453 с. **[** V. V. Mironov, N. I. Yusupova, and G. R. Shakirova, *Hierarchical Data Models: Concepts and Realization Based on XML*, (in Russian). Moscow: Mashinostroenie, 2011. **]**

ОБ АВТОРАХ

КАНАШИН Виталий Владленович, асп. каф. АСУ. Дипл. инж. по АСУ (УГАТУ, 2011). Готовит дис. об иерархических виджетах в ситуационно-ориентированных базах данных.

МИРОНОВ Валерий Викторович, проф. каф. АСУ. Дипл. радиофизик (Воронежск. гос. ун-т, 1975). Д-р техн. наук по упр. в техн. системах (УГАТУ, 1995). Иссл. в обл. иерархических моделей и ситуационного управления.

METADATA

Title: Hierarchical widgets: user data control algorithms in web applications on the basis of situation-oriented databases.

Authors: V. V. Kanashin and V. V. Mironov

Affiliation: Ufa State Aviation Technical University (UGATU), Russia.

Email: vitas.k@rambler.ru, mironov@list.ru.

Language: Russian.

Source: Vestnik UGATU (Scientific journal of Ufa State Aviation Technical University), vol. 18, no. 1 (62), pp. 204-213, 2014. ISSN 2225-2789 (Online), ISSN 1992-6502 (Print).

Abstract: Algorithmic support for difficult structured user interface in the web applications functioning on the basis of the situation-oriented databases (SODB) is considered. Within the concept of hierarchical widgets developed in the previous articles of authors, algorithms of the organization of user data input and control are considered. Algorithms are described: tester elements for control of the data entered by the user; receiver elements for the room of the entered data in the DOM buffer and fixings of the revealed errors; jump elements which activity depends on existence / lack of the revealed errors. Default rules provided in algorithms, allows reducing the program code volume considerably. Algorithms are realized as a part of the interpreter of the dynamic HSM models functioning on the PHP platform.

Key words: Web application; user interface; situation-oriented database; dynamic model; hierarchical widgets; user data; regular expressions; HSM; XML; XSLT; model-driven development.

About authors:

KANASHIN, Vitaliy Vladlenovich, Postgrad. (PhD) Student, Automated Systems Dept. Dipl. Eng. (UGATU, 2011).

MIRONOV, Valeriy Viktorovich, Prof., Automated Systems Dept. Dipl. Radiophysicist (Voronezh State Univ., 1975). Dr. (Habil.) Tech. Sci. (UGATU, 1995).