Уфа : УГАТУ, 2013 Вестник УГАТУ Т. 17, № 5 (58). С. 166–176

УДК 004.738.5:004.65

ИЕРАРХИЧЕСКИЕ ВИДЖЕТЫ: ВВОД И КОНТРОЛЬ 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 22.05.2013

Аннотация. Рассматривается построение сложноструктурированного интерфейса пользователя в веб-приложениях, функционирующих на основе ситуационно-ориентированных баз данных (СОБД). В рамках концепции иерархических виджетов, предложенной в предыдущей статье авторов, рассматривается задача организации ввода и контроля пользовательских данных. Предложены новые элементы динамической модели СОБД, обеспечивающие решение этой задачи, обсуждаются их состав, структура и функциональность. Разбирается содержательный пример ввода и контроля пользовательских данных с использованием иерархических виджетов.

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

ВВЕДЕНИЕ

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 предыдущей статье [10] авторами был предложен и исследован подход к созданию сложноструктурированных 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.

Основное внимание в указанной работе было уделено вопросам вывода данных для отображения пользователю. При этом остался в стороне важный вопрос о том, как в рамках данной концепции организовать ввод и контроль данных, которые пользователь передает серверу через интерфейс, предоставляемый виджетами.

С помощью иерархических виджетов в браузере пользователя формируются экранные формы с элементами управления (полями ввода, кнопками, переключателями и т. п.), состояние которых, измененное пользователем, необходимо учитывать на стороне сервера. Для этого в серверном сценарии требуется предусмотреть программы контроля пользовательских данных на предмет отсутствия ошибок или злонамеренных инъекций, а также обработки данных для приведения их к требуемому формату и сохранения в базе. В случае обнаружения ошибочных данных необходимо указать на это пользователю и организовать их повторный ввод. Все эти

Работа поддержана грантом РФФИ 13-07-00011.

задачи вполне решаются и традиционными средствами, но требуют достаточно объемного и рутинного программирования.

Целью данной работы является уменьшение трудоемкости программирования процедур ввода и контроля данных в веб-приложениях на основе СОБД за счет декларативного задания этих функций в иерархических виджетах. Предлагается предусмотреть в составе виджетов набор дополнительных элементов, позволяющих специфицировать операции контроля, преобразования, обработки ошибок пользовательских данных. Выполнение указанных операций возлагается на интерпретатор динамической модели СОБД.

ИСХОДНЫЕ ПОНЯТИЯ

СОБД мы рассматриваем как набор следующих компонентов [7]: HSML (HSM Library) – библиотека динамических моделей, содержащая набор иерархических ситуационных моделей HSM (Hierarchical Situational Models); CSM (Current State Memory) – память текущего состояния, хранящая сведения о текущих состояниях динамических моделей; ADM (Associated Data Memory) – хранилище XML-документов, соотнесенных с различными состояниями динамической модели; AFL (Associated Functions Library) – библиотека ассоциированных функций обработки данных, соотнесенных с состояниями динамической модели; HSMI (HSM Interpreter) – интерпретатор динамической модели, который в ответ на внешний запрос *Q* формирует ответ *R* путем обработки некоторой динамической модели HSM из HSML на основе отслеживания ее текущих состояний, сохраняемых в CSM, обработки ассоциированных документов из ADM и выполнения ассоциированных функций из AFL. Если СОБД используется на сервере как основа для веб-приложения, то в качестве входного запроса *Q* выступает набор параметров, получаемый вместе с URL (например, параметры экранной формы в режиме POST), а в качестве результата *R* — ответный HTML-код, отправляемый клиенту. Параметры запроса указывают обрабатываемую динамическую модель и необходимость смены ее текущих состояний.

Динамическая модель HSM представляет собой иерархию графов переходов с конечным числом состояний (Finite State Model). Состояния нагружены элементами, задающими связь с определенными данными из ADM, функциями из AFL. С дугами графов – переходами состояний – ассоциированы предикаты, определяющие их активность, обеспечивающую смену текущего состояния. Динамическая модель имеет графическое представление (HSM-диаграмма) и эквивалентное текстовое представление с XML-образным синтаксисом. Основные элементы HSM:

- **sub** – *субмодель* – контейнер, включающий набор состояний одного уровня иерархии, из которых только одно является текущим;
- **sta** – *состояние* – контейнер элементов (субмоделей, переходов, погружений и др.), обрабатываемых последовательно, если это состояние является текущим;
- **jmp** – *переход* – задает смену состояний в пределах субмодели;
- **div** – *погружение* – задает переход во внутреннюю субмодель из текущего состояния;
- **act** – *акция* – задает функцию из AFL, выполняемую в данном текущем состоянии;
- **var** – *переменная* – задает переменную, ассоциированную с состоянием, значение которой доступно и может изменяться, пока состояние является текущим;
- **dom** – *DOM* – задает подготовку и обработку XML-данных состояний в DOM-объекте; является контейнером для источников и приемников;
- **src** – *источник* – задает для DOM-элемента XML-документ в ADM или в DOM-объекте для загрузки в данный DOM-объект;
- **rcv** – *приемник* – задает для DOM-элемента вывод данных в ADM или в выходной поток путем преобразования содержимого DOM-объекта;
- **wdg** – *виджет* – задает формирование фрагмента кода пользовательского интерфейса для текущего состояния.

Интерпретация динамической модели. Двухпроходный интерпретатор HSMI на каждом цикле интерпретации дважды выполняет рекурсивный обход иерархии текущих состояний динамической модели. На первом проходе отслеживается смена текущих состояний субмоделей, а на втором – формирование результата для текущих состояний.

КОНЦЕПЦИЯ

Общая процедура контроля пользовательских данных, на которой мы основываемся при построении интерфейса пользователя, следующая:

Шаг 1. Подготовка данных, предназначенных для первичного отображения пользователю, в DOM-объекте. Достигается с помощью DOM-элементов динамической модели.

Шаг 2. Формирование на основе содержимого DOM-объекта кода фрагмента отображения, содержащего элементы интерфейса для ввода данных пользователем (поля ввода текста и т. п.). Достигается путем XSL-трансформации содержимого DOM-объекта с помощью виджет-элементов динамической модели.

Шаг 3. Отправка кода фрагмента отображения в браузер пользователя и получение ответных данных пользователя. Достигается с помощью средств веб-протокола обмена данными.

Шаг 4. Проверка корректности полученных данных пользователя, выявление ошибок.

Шаг 5. Если обнаружены ошибки, то подготовка данных, предназначенных для повторного отображения пользователю с учетом ошибок, в DOM-объекте, и возврат к шагу 2. Иначе – переход к следующему шагу.

Шаг 6. Сохранение (если требуется) введенных данных пользователя.

Здесь шаги 1–3 вполне поддерживаются имеющимися средствами СОБД, а шаги 4–6 требуют «ручного» программирования соответствующих функций в AFL. Наша задача – ввести новые и расширить имеющиеся средства построения HSM так, чтобы поддержать эти шаги на декларативном уровне.

Контролер входных данных

Введем в составе динамической модели HSM новый элемент – контролёр входных данных inp (inp – от input, вход), предназначенный для проверки входных данных пользователя на первом проходе интерпретации динамической модели. Контролёр закрепляется за определенным элементом входных данных и задает:

- адрес контролируемого элемента данных;
- адрес, по которому значение элемента данных сохраняется в определенном DOM-объекте;
- условия, которым должно удовлетворять значение элемента данных, и адреса, по которым записываются коды ошибок в случае нарушения условий.

Таким образом, на каждый элемент пользовательских данных, содержащихся в заполненной пользователем форме, нужно предусмотреть отдельный элемент-контролёр. В результате интерпретации контролёра как элемента динамической модели значение контролируемого элемента данных и сведения об обнаруженных ошибках будут сохранены в DOM-объекте. На основе этой информации можно организовать дальнейшую обработку правильно введенных пользовательских данных или повторный запрос данных у пользователя с указанием допущенных ошибок.

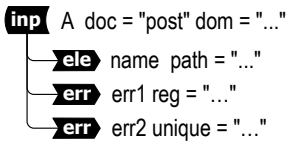

Рис. 1. Контролёр входных данных

На рис. 1 представлен фрагмент динамической модели HSM – контролёр входных данных. Здесь обработчик по имени A ссылается на одноименный элемент массива POST (в массиве POST сохраняются данные, содержащиеся в пришедшей на сервер форме при использовании платформы PHP) и привязан к указанному DOM-объекту. В качестве внутренних элементов контролёр может содержать элементы-приемники специальных типов: ele (элемент), att (атрибут) и err (ошибка).

Первый элемент-приемник, приведенный на рис. 1, имеет тип ele. Он записывает в DOM-объект значение A в виде нового XML-элемента, дочернего для узла, путь к которому задан в атрибуте path. Если бы этот приемник имел тип att, то значение A было бы записано в DOM-объект в качестве XML-атрибута.

Второй и третий элементы-приемники имеют тип err. Это *фиксаторы ошибок*, выполняющие проверку значения элемента данных и запись в DOM-объект информации об ошибках. Первый фиксатор проверяет данные, применяя к значению элемента данных регулярное выражение, заданное атрибутом reg (само регулярное выражение на рисунке опущено). При обнаружении ошибки ее код err1 сохраняется в DOM-объекте по определенному адресу (соответствующие атрибуты опущены). Второй фиксатор проверяет данные на уникальность среди множества значений, заданного XPath-выражением в атрибуте unique (само XPath-выражение на рисунке опущено). При обнаружении такой ошибки ее код err2 сохраняется в DOM-объекте. В контролёре может быть задано несколько фиксаторов ошибок.

Обнаружение ошибок

Способ выявления ошибок в данных, введенных пользователем, задается в атрибутах фиксатора ошибки. Можно предусмотреть сле-

дующие возможности задания правил обнаружения ошибок в проверяемых данных:

1) диапазон допустимых значений, атрибут between – введенное пользователем значение проверяется на принадлежность заданному диапазону допустимых значений;

2) список допустимых значений, атрибут list – введенное пользователем значение проверяется на принадлежность заданному списку допустимых значений;

3) регулярное выражение, атрибут reg – к проверяемому значению применяется заданное регулярное выражение, которое возвращает логический результат, свидетельствующий о наличии или отсутствии ошибки;

4) уникальность, атрибут unique – введенное пользователем значение проверяется на отсутствие в списке значений, сформированных с помощью заданного XPath-выражения, примененного к указанному DOM-объекту;

5) пользовательская функция, атрибуты func, param – к проверяемому значению применяется указанная AFL-функция (возможно, с заданными параметрами), которая возвращает логический результат, свидетельствующий о наличии или отсутствии ошибки.

Применимость первых трех способов ограничена простыми случаями, когда для выявления ошибки достаточно проанализировать само проверяемое значение без сопоставления его с другими значениями из базы данных. При этом третий способ по своим возможностям перекрывает первые два, но более сложен как в плане задания, так и в плане исполнения. Четвертый способ предназначен для проверки уникальности значения введенного пользователем идентификатора среди других значений, уже занесенных в базу.

Фиксация ошибок в DOM-объекте

В общем случае у элемента-контролёра может быть предусмотрено несколько элементов-фиксаторов, что позволяет выполнять проверки одного элемента данных несколькими способами (например, несколькими регулярными выражениями) со своими кодами ошибок.

При обнаружении ошибки фиксатор должен разместить информацию о ней в заданном DOM-объекте. Необходима стандартизация этой процедуры, поскольку информация об ошибках должна учитываться другими элементами динамической модели.

Например, можно условиться (рис. 2), что сведения об обнаруженной ошибке заносятся в поддерево inpErrs корневого элемента DOM-объекта виде элемента inpData, в котором XML-атрибуты name и errCode содержат соответственно имя ошибочного элемента данных и код ошибки, а текстовое содержимое – само ошибочное значение.

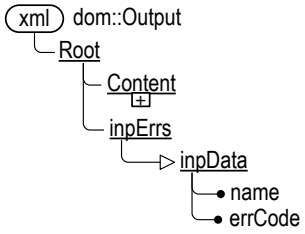

Рис. 2. Информация в DOM-объекте об обнаруженных ошибках (синтаксис XML-схемы из работы [11])

Содержимое inpErrs может быть использовано для формирования сообщений об ошибках при интерпретации виджет-элемента, основанного на подобном DOM-объекте.

Использование контролера для организации смены состояний

Функциональность элемента-контролёра позволяет использовать его для организации смены текущего состояния динамической модели в зависимости от наличия или отсутствия ошибок во входных данных. Для этого потребуется соответствующим образом дополнить функциональность элемента-перехода динамической модели.

На рис. 3 приведены два варианта использования контролёра для смены текущего состояния. В первом варианте элемент-контролёр inp:A (см. рис. 3) размещается непосредственно в элементе-состоянии sta:Проверка, а во втором – в элементе-переходе jmp:Продолжение.

Состояние sta:Проверка предназначено для проверки элементов данных пользователя (в данном случае – элемента A), состояние sta:Ошибки – для обработки обнаруженных ошибок (повторной отправки формы пользователю с указанием ошибок – содержимое не детализовано), а состояние sta:Продолжение – для обработки корректных данных (сохранение в базе документов – не детализовано).

В первом случае (рис. 3, *а*) для смены ситуации требуется, чтобы элемент-переход проверял наличие стандартизованных записей об ошибках в DOM-объекте. Традиционным путем это можно сделать, запрограммировав для пере-

хода функцию-предикат, которая обращается к определенному DOM-объекту и проверяет наличие в нем непустого поддерева записей об ошибках. Для уменьшения трудоемкости такого «ручного» программирования целесообразно передать эту функцию интерпретатору, предусмотрев для элемента-перехода атрибут inpErrDOM, задающий имя проверяемого DOM-объекта. Наличие этого атрибута в переходе будет указывать интерпретатору на необходимость выполнения такого рода проверки в заданном DOM-объекте.

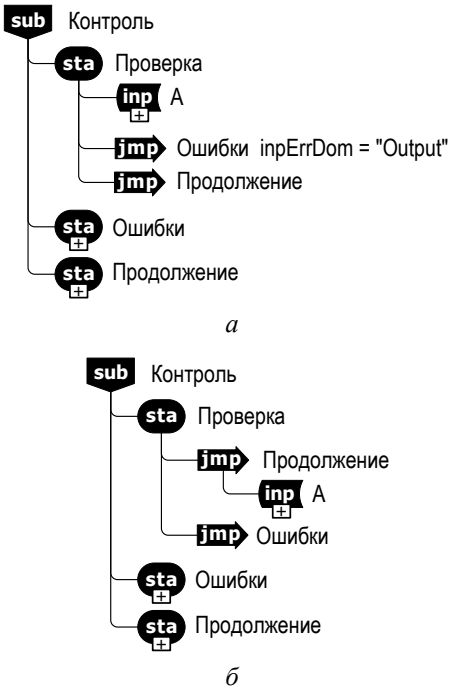

Рис. 3. Использование контролёра входных данных для смены текущего состояния: *а* – в состоянии; *б* – в переходе

Во втором случае (рис. 3, *б*) необходимо, чтобы элемент-контролёр сам по себе мог использоваться в качестве вложенного предиката элемента-перехода. Для этого условимся, что в качестве результата интерпретации элемент-контролёр возвращает значение «истина», если при интерпретации вложенных фиксаторов ошибки не было зафиксировано ни одной ошибки. Этот подход дает более изящные решения, когда смену состояния необходимо выполнить по результатам контроля одного элемента данных пользователя.

СОДЕРЖАТЕЛЬНЫЙ ПРИМЕР

Технику применения введенных новых элементов проиллюстрируем на гипотетическом веб-приложении, рассмотренном в предыдущей статье [10] и предназначенном для отображения пользователю сведений о студентах, о предметах и о сдачах студентами предметов. Дополним функциональность указанного приложения (ориентированную лишь на отображение пользователю данных, уже имеющихся в базе) функциями вставки новых и обновления имеющихся данных. А именно – предусмотрим возможность вставки сведений о новых и обновления сведений об имеющихся студентах (применительно к предметам эти функции реализуются аналогичным образом и здесь не рассматриваются). На рис. 4 представлен фрагмент динамической модели, а на рис. 5 показаны экранные формы, выводимые пользователю, когда соответствующие состояния становятся текущими.

Состояния динамической модели

На верхнем уровне модель содержит три состояния, соответствующие трем основным режимам работы пользователя с базой студентов:

- состояние sta:СписокСтудентов отвечает за работу со списком имеющихся студентов (это состояние присутствует в прежней версии модели);
- состояние sta:НовыйСтудент обеспечивает занесение в базу сведений о новом студенте (новое состояние, обеспечивающее новую функциональность);
- состояние sta:ВыбранСтудент поддерживает работу с данными, соответствующими выбранному из базы студенту (модификация аналогичного состояния прежней модели).

Для расширения функциональности последнее состояние дополнено субмоделью, содержащей следующие подсостояния:

- состояние sta:ПоказСтудента обеспечивает отображение пользователю детальных сведений по выбранному студенту – о сданных студентом предметах и полученных оценках (состояние присутствует в прежней модели);
- состояние sta:РедСтудента поддерживает новую функцию редактирования пользователем сведений выбранного студента (новое состояние).

Экранные формы состояний формируются с помощью виджет-элементов, предусмотренных в составе каждого состояния.

В состоянии sta:СписокСтудентов пользователю отображаются сведения о зарегистрированных студентах (см. рис. 5, *а*). Верхняя строка формы содержит два раскрывающегося списка-меню.

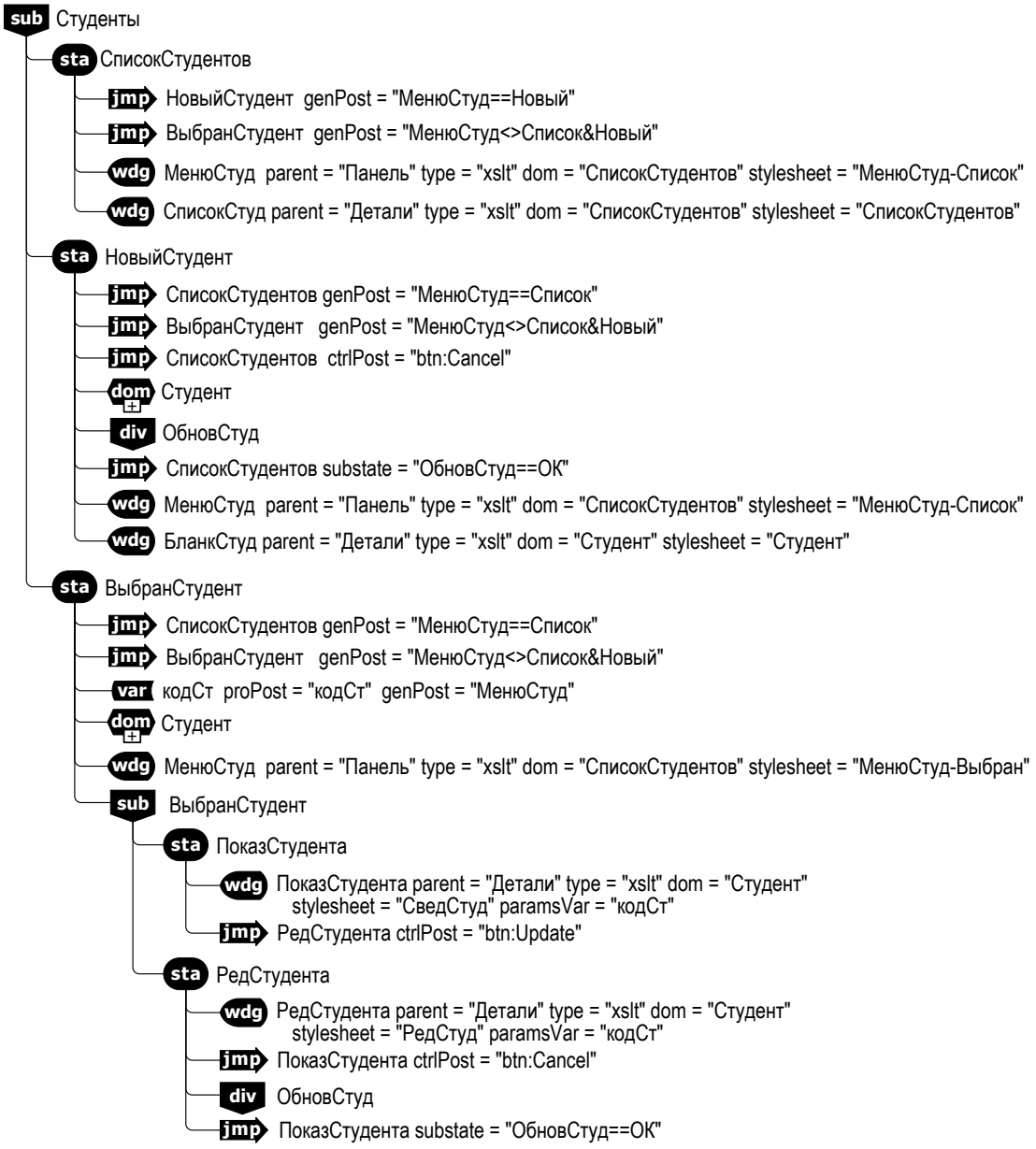

Рис. 4. Модель состояний, предусматривающих вставку и обновление данных пользователя

Левое меню, одинаковое на всех формах данного примера, предназначено для выбора функции («Студенты» или «Предметы»); оно формируется виджет-элементом, размещенном в состоянии вышестоящего уровня динамической модели.

Правое меню предназначено для выбора режима отображения сведений о студентах, данное состояние соответствует опции «*Список*». Это меню формируется виджет-элементом wdg:МенюСтуд путем XSL-трансформации содержимого DOM-объекта dom:СписокСтудентов (dom-элемент, порождающий этот DOM-объект, размещен в состоянии вышестоящего уровня динамической модели).

Центр формы занимает собственно список студентов (приведены лишь идентификаторы и фамилии студентов). В нижней строке размещены кнопки управления, дублирующие функциональность правого меню. Список и кнопки формируются виджет-элементом wdg:СписокСтуд путем XSL-трансформации содержимого упоминавшегося DOM-объекта dom:СписокСтудентов.

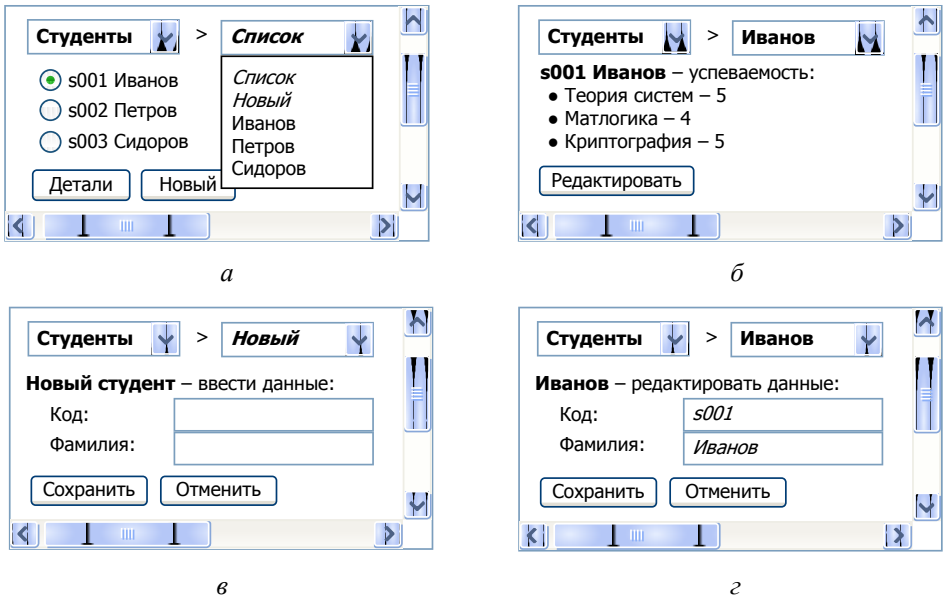

Рис. 5. Формы пользовательского интерфейса для различных состояний динамической модели: *а* – список студентов (в состоянии sta:СписокСтудентов); *б* – детальные сведения о выбранном студенте (в состоянии sta:ПоказСтудента); *в* – форма нового студента (в состоянии sta:НовыйСтудент); *г* – форма редактирования (в состоянии sta:РедСтудента)

Переходы из данного состояния обеспечивается элементами jmp. Элемент jmp:НовыйСтудент выполняет переход в состояние sta:НовыйСтудент в случае выбора в правом меню опции «*Новый*». Аналогичным образом элемент jmp:ВыбранСтудент выполняет переход в состояние sta:ВыбранСтудент в случае выбора в правом меню опции, соответствующей одному из студентов списка.

В состоянии sta:НовыйСтудент пользователю отображаются бланк для ввода сведений о новом студенте (только код-идентификатор нового студента и его фамилия, см. рис. 5, *в*). Меню в верхней строке формируются как в предыдущем состоянии, а собственно бланк – с помощью виджет-элемента wdg:БланкСтуд путем XSL-трансформации содержимого DOM-объекта dom:Студент. Этот DOM-объект порождается одноименным dom-элементом dom:Студент, размещенном в данном состоянии.

Элемент jmp:ВыбранСтудент выполняет переход в соответствующее состояние в случае выбора в правом меню одного из студентов списка. Первые два элемента jmp:СписокСтудентов выполняют переход в случае выбора пользователем в правом меню опции «*Список*» и в случае нажатия кнопки «Отменить» (btn:Cancel). Третий элемент jmp:СписокСтудентов выполняет переход, если субмодель sub:ОбновСтуд находится в состоянии «ОК».

Погружение в субмодель sub:ОбновСтуд выполняет div-элемент div:ОбновСтуд.

В состоянии sta:ВыбранСтудент пользователю отображаются детальные сведения о выбранном студенте: либо для просмотра (см. рис. 5, *б*), либо для редактирования в зависимости от подсостояния (см. рис. 5, *г*). Меню в верхней строке формируются как в предыдущих состояниях. Идентификатор выбранного студента хранится в переменной var:КодСт, а сведения о выбранном студенте – в DOM-объекте dom:Студент, который порождается одноименным dom-элементом. Элемент jmp:СписокСтудентов выполняет переход в случае выбора пользователем в правом меню опции «*Список*». Элемент jmp:ВыбранСтудент выполняет обновление состояния в случае выбора пользователем другого студента.

Подсостояния этого состояния заданы в субмодели sub:ВыбранСтудент.

В подсостоянии sta:ПоказСтудента пользователю с помощью виджета wdg:ПоказСтудента отображаются детальные сведения о выбранном студенте путем XSL-трансформации содержимого DOM-объекта dom:Студент (см. рис. 5, *б*). Элемент jmp:РедСтудента выполняет переход

состояния в случае нажатия пользователем кнопки «Редактировать» (btn:Update).

В подсостоянии sta:РедСтудента пользователю с помощью виджета wdg:РедСтудента отображается форма редактирования детальных сведений (см. рис. 5, *г*). Элемент jmp:ПоказСтудента выполняет переход состояния в случае нажатия пользователем кнопки «Отменить» (btn:Cancel). Dive-элемент div:ОбновСтуд выполняет погружение в субмодель sub:ОбновСтуд, по результатам которого второй элемент jmp:ПоказСтудента выполняет переход состояния, если субмодель находится в состоянии «ОК».

Контроль вводимых данных

Таким образом, пользователь вводит данные в двух состояниях: в состоянии sta:НовыйСтудент – сведения о новом студенте; в состоянии sta:РедСтудента – измененные сведения уже имеющегося студента. В обоих случаях требуется одинаковая проверка введенных данных, поэтому она реализована с помощью единой субмодели sub:ОбновСтуд (рис. 6).

В основе субмодели sub:ОбновСтуд лежит модифицированная модель контроля пользовательских данных и обработки ошибок с использованием элемента-контролёра (см. рис. 3).

Она содержит 5 состояний, отражающих различные ситуации процесса контроля:

- sta:Исходное – состояние, с которого начинается обработка данных. В этом состоянии пользователю отображаются формы для ввода новых или редактирования имеющихся сведений (рис. 5, *в* или *г*). На следующем цикле интерпретации выполняется переход в состояние sta:Тест при нажатии пользователем кнопки «Сохранить» (btn:Save);
- sta:Тест – транзитное состояние тестирования введенных данных. В этом состоянии предусмотрено два контролёра: inp:КодСтуд и inp:ФиоСтуд, которые проверяют: 1) уникальность и структуру введенного кода студента (фиксаторы err:id01 и err:st01); 2) запись фамилии студента только русскими буквами (фиксатор err:rb01). Обнаруженные ошибки заносятся в DOM-объект dom:Студент. Переход в состояние sta:Ошибки выполняется в случае наличия ошибок, иначе – переход в состояние sta:Сохранение.
- sta:Сохранение – транзитное состояние сохранения введенных данных в случае отсутствия ошибок. Переход в состояние sta:Ошибки выполняется в случае ошибки сохранения, иначе – переход в состояние sta:ОК;
- sta:Ошибки – финальное состояние обнаружения ошибок в данных, что требует отобразить ошибочные данные пользователю с соответствующими комментариями и предоставить ему возможность повторного ввода. На следующем цикле интерпретации после попадания в это состояние возможен переход в состояние sta:Тест, если пользователь нажмет кнопку «Сохранить» (повторная попытка ввода данных);
- sta:ОК – финальное состояние успешного тестирования и сохранения пользовательских данных.

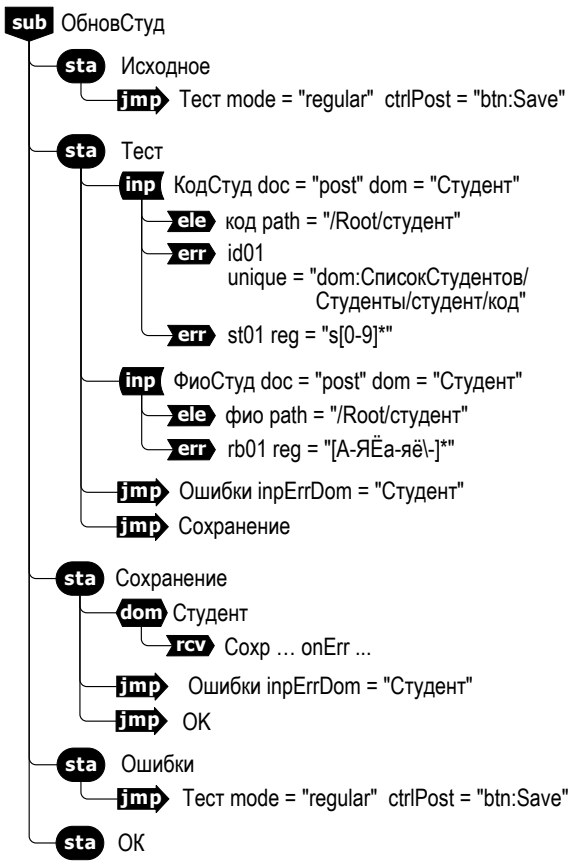

Рис. 6. Субмодель для обработки введенных пользователем данных

Таким образом, в результате цикла интерпретации субмодель sub:ОбновСтуд может находиться в одном из трех финальных состояний: sta:Исходное, в котором пользователю отображаются формы ввода в исходном виде; sta:Ошибки, в котором пользователю отображаются формы с введенными данными и сообщениями об ошибках; sta:ОК, которое служит основанием для продолжения работы на вышестоящем уровне динамической модели.

Отображение ошибок пользователю

Итак, с помощью элементов-контролёров введенные пользователем данные на первом проходе интерпретации заносятся в DOM-объект dom:Студент. В случае обнаружения ошибок в пользовательских данных, сведения об ошибках тоже записываются в этот же DOM-объект (тоже на первом проходе интерпретации).

На рис. 7 приведен пример XML-контента DOM-объекта dom:Студент для случая двух обнаруженных ошибок в данных о новом студенте (используется нотация записи XML из книги [11]).

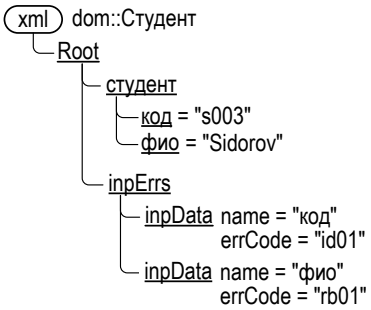

Рис. 7. Пример XML-контента DOM-объекта с данными о новом студенте и сведениями об обнаруженных ошибках

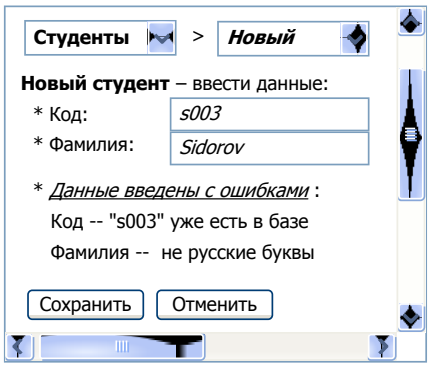

Рис. 8. Форма пользовательского интерфейса с сообщениями об ошибках

При создании DOM-объекта в нем присутствовал корневой элемент Root с пустыми дочерними элементами студент и inpErrs. После обработки субмодели sub:ОбновСтуд контролёр inp:КодСтуд в элемент студент добавил с помощью приемника ele:код дочерний элемент код со значением "s003", а по результатам контроля ошибок (фиксаторы err:id01 и err:st01) – в элемент inpErrs дочерний элемент inpData с атрибутами name = "код" и errCode = "id01". Аналогичным образом контролёр inp:ФиоСтуд в элемент студент добавил с помощью приемника ele:фио дочерний элемент фио со значением "Sidorov", а по результатам контроля ошибок (фиксатор err:rb01) – в элемент inpErrs дочерний элемент inpData с атрибутами name = "фио" и errCode = "rb01". В результате в DOM-объект были занесены введенные пользователем данные, а также сведения об обнаруженных в этих данных ошибках.

Поскольку в случае ошибок состояние sta:ОК не является текущим состоянием субмодели sub:ОбновСтуд, то на вышестоящем уровне не сработают соответствующие jump-элементы и родительские состояния этой субмодели (sta: НовыйСтудент или sta:РедСтудента) останутся текущими. В результате на втором проходе интерпретации соответствующие виджеты (wdg: БланкСтуд или wdg:РедСтудента) сформируют для пользователя экранные формы, содержащие введенные некорректные данные и соответствующие сообщения об ошибках (рис. 8). В этой ситуации пользователь может либо исправить данные и повторить ввод, нажав кнопку «Сохранить», либо отказаться от занесения нового студента, нажав кнопку «Отменить».

ЗАКЛЮЧЕНИЕ

В данной работе 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.

Использование предложенных средств позволяет декларативно задать в динамической модели, какие введенные данные нужно контролировать, какие проверкам их следует подвергнуть, куда записать сведения об обнаруженных ошибках, в какое состояние переходить при обнаружении ошибок.

Рассмотренный содержательный пример продемонстрировал возможность в декларативной форме запрограммировать как ситуации сохранения безошибочных данных в базе, так и ситуации повторной отправки формы пользователю с указанием обнаруженных ошибок.

Как ожидается, реализация предложенных средств даст разработчикам СОБД гибкий инструмент, облегчающий за счет декларативного описания проектирование пользовательского интерфейса веб-приложений, предусматривающих ввод и контроль пользовательских данных.

Дальнейшие усилия в данном направлении предполагается направить на алгоритмическую и программную реализацию предложенных концептуальных решений в виде соответствующих дополнений к XML-структуре динамической модели и к программному обеспечению интерпретатора.

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.
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.
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.
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.
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.
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.
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.
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.
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.
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.
11. **Миронов В. В., Юсупова Н. И., Шакирова Г. Р.** Иерархические модели данных: концепции и реализация на основе XML. М.: Машиностроение, 2011. 453 с.

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: input and control of the user data 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. 17, no. 5 (58), pp. 166-176, 2013. ISSN 2225-2789 (Online), ISSN 1992-6502 (Print).

Abstract: Development of complex 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 offered in the previous article, the problem of the organization of input and control of the user data is considered. The new elements of the SOBD dynamic model providing the solution of this problem are offered, their range, structure and functionality are discussed. The substantial example of input and control of the user data with use of hierarchical widgets is reviewed.

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

References (English Transliteration):

1. 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. 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. 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. 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. 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. 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. 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. 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. 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. V. V. Kanashin and V. V. Mironov, “Dynamic DOM objects in situation-oriented databases: lingware and knoware of data sources,” (in Russian), *Vestnik UGATU*, vol. 17, no. 2 (55), pp. 138-149, 2013.
11. V. V. Mironov, N. I. Yusupova, and G. R. Shakirova, *Hierarchical data models: concepts and realization based on XML*, (in Russian). Moscow: Mashinostroenie, 2011.

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).