ISSN 1992-6502 (Print) Вестник УГАТУ ISSN 2225-2789 (Online)
2014. Т. 18, № 2 (63). С. 185–196 http://journal.ugatu.ac.ru

УДК 004.65

ИЕРАРХИЧЕСКИЕ ВИДЖЕТЫ: ОПЫТ ПРИМЕНЕНИЯ В ВЕБ-ПРИЛОЖЕНИИ НА ОСНОВЕ СИТУАЦИОННО-ОРИЕНТИРОВАННОЙ БАЗЫ 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. В предыдущих статьях авторов предложены концепция, модели и алгоритмическое обеспечение иерархических виджетов, предназначенных для построения сложно-структурированного интерфейса пользователя в веб-приложениях на основе ситуационно-ориентированных баз данных (СОБД). Здесь обсуждаются результаты использования иерархических виджетов при построении веб-приложения, обслуживающего деятельность диссертационных советов вуза. На примере одного из состояний динамической модели подробно рассматриваются и сравниваются технологии организации тестирования данных, вводимых пользователем, а также формирования сообщений пользователю без использования и с использованием иерархических виджетов. Показано, что применение виджетов упрощает огранизацию XSL-трансформации и сокращает затраты на дополнительное программирование функций проверки пользовательских данных.

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 предыдущих статьях [10–12] 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] авторами 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 [12] разработано алгоритмическое обеспечение новых элементов динамической модели HSM: элемента-контролёра для контроля данных пользователя, входящих в его состав элементов-приемников для помещения введенных данных в DOM-буфер и фиксации выявленных ошибок, а также элементов-переходов, активность которых зависит от наличия / отсутствия выявленных ошибок.

Разработанные алгоритмы были реализованы в составе интерпретатора динамических моделей HSMI, функционирующего на платформе PHP, для практического использования при создании веб-приложений.

Работа поддержана грантом РФФИ 13-07-00011.

В данной работе рассматриваются результаты применения разработанных средств контроля входных данных и их алгоритмического и программного обеспечения в конкретном веб-приложении – исследовательском прототипе системы «Диссоветы». Веб-приложение реализовано в рамках исследовательского прототипа системы «Диссоветы». Оно предназначено для поддержки деятельности диссертационных советов вуза на различных уровнях – от отдельных диссертантов до руководства вуза. В нем достаточно много функций, предусматривающих ввод пользовательских данных – главным образом это многочисленные сведения, которые должны вводить диссертанты и секретари диссоветов: персональные данные диссертантов, научных руководителей, оппонентов и т. д. Эти данные должны проверяться на корректность перед сохранением в базе.

В качестве исходной была взята модель, разработанная и отлаженная ранее «вручную» (без использования виджетов) в рамках предшествующих исследований [3–7]. Исходная модель была модифицирована – переписана с использованием виджетов в рамках той же функциональности. Это позволило сравнить модифицированную модель, основанную на виджетах, с исходной моделью в плане сложности построения и объема требуемого программного кода.

Ниже представлено детальное сравнение исходной и модифицированной модели, программного кода, а также шаблонов XSL-трансформации для одного из состояний, в котором предусмотрены ввод и тестирование данных пользователя, после чего даны оценки востребованности виджетов и сокращения объема программного кода для модели в целом.

Обсуждаемые результаты могут быть полезны как иллюстрация технологических приемов, применимых при построении веб-приложений, основанных на ситуационно-ориентированных базах данных, в которых предусматривается ввод и обработка пользовательских данных.

СРАВНЕНИЕ HSM-МОДЕЛЕЙ БЕЗ ИСПОЛЬЗОВАНИЯ И С ИСПОЛЬЗОВАНИЕМ ВИДЖЕТА

Исходная модель (без виджетов)

На рис. 1 приведена диаграмма исходной модели состояния sta:EditPersInfo, предназначенного для редактирования персональных данных выбранного диссертанта. Модель относительно проста, на первом уровне иерархии она содержит doc-элемент, dom-элемент и три jmp-элемента.

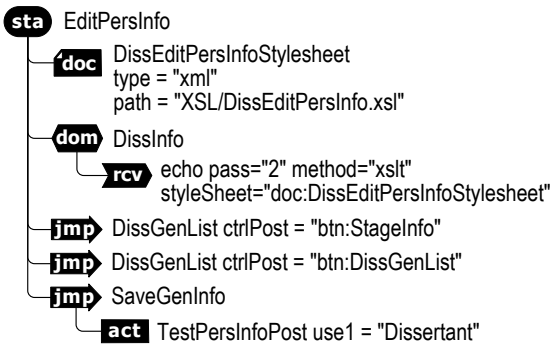


Рис. 1. Исходная модель (без использования виджетов и контролёров)

Элемент doc:DissEditPersInfoStylesheet задает местоположение используемой далее таблицы стилей XSL-трансформации в хранилище документов ADM.

Элемент dom:DissInfo обеспечивает на втором проходе интерпретации вывод в выходной поток персональных данных диссертанта. Данные выводятся из одноименного DOM-объекта, созданного ранее в состоянии более высокого уровня и загруженного XML-документом с данными об обрабатываемом диссертанте. Вывод данных обеспечивается элементом rcv:echo путем XSL-трансформации DOM-объекта в соответствии с указанной таблицей стилей.

Элементы jmp:DissGenList обеспечивают переход в состояние sta:DissGenList, если обнаруживается нажатие кнопок btn:StageInfo или btn: StageInfo. Эти кнопки отменяют изменения персональных данных выбранного диссертанта, а в целевом состоянии sta:DissGenList пользователю отображается общий список диссертантов.

Элемент jmp:SaveGenInfo обеспечивает переход в состояние sta: SaveGenInfo, в котором выполняется сохранение измененных персональных данных диссертанта. Активность этого элемента управляется вложенным элементом-акцией act:TestPersInfoPost, который ссылается на функцию Dissertant из библиотеки AFL.

Таким образом, в самой динамической модели никак не отражены обрабатываемые пользовательские данные, как и логика этой обработки. Эти моменты определяются, во-первых, используемой таблицей стилей DissEditPersInfo-Stylesheet, которая задает данные, отображаемые пользователю, во-вторых, процедурой-функцией Dissertant с параметром TestPersInfo-Post, которая проверяет данные, введенные пользователем, фиксирует обнаруженные ошиб-

ки, подготавливает данные к сохранению, сообщает переходу jmp:SaveGenInfo о возможности сохранения данных в базе. Поэтому необходимо проанализировать организацию этих объектов.

Процедура-функция тестирования входных данных

При обработке акции act:TestPersInfoPost производится вызов функции Dissertant с параметром TestPersInfoPost (именем акции). Указанная функция написана на языке PHP, ее фрагмент, соответствующий параметру TestPersInfoPost, приведен в приложении, листинг 1, а здесь рассмотрим обобщенную схему алгоритма, которая представлена на рис. 2.

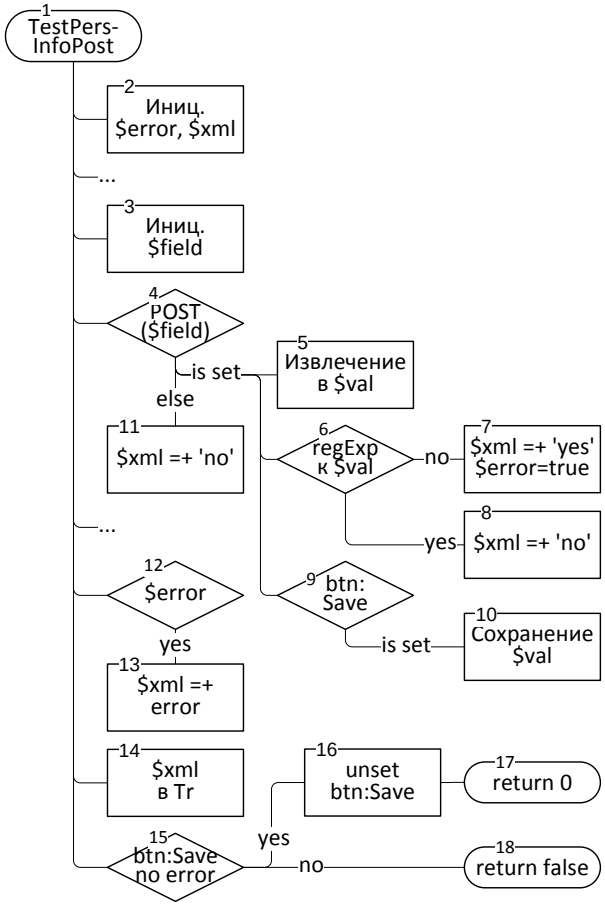


Рис. 2. Схема алгоритма тестирования входных данных для исходной модели

Алгоритм (блок 1) начинается с инициализации переменных $error и $xml (блок 2). Первая переменная – это флаг наличия ошибок в проверяемых данных, ей присваивается начальное значение false; вторая переменная содержит XML-отчет об обнаруженных ошибках, в нее заносится открывающий тег корневого элемента "<PersInfo>".

Блоки 3–11 выполняют типовую проверку отдельного элемента входных данных. Рассматриваемая акция проверяет 11 таких элементов: фамилию, имя, отчество, пол, должность (в именительном и родительном падежах), подразделение, степень, звание, окончание аспирантуры / докторантуры, биографические сведения.

Блок 3 заносит в переменную $field имя проверяемого элемента в массиве Post, а в блоке 4 проверяется существование этого элемента. Если элемент существует, его значение извлекается, подвергается фильтрации, экранированию специальных символов и записывается в переменную $val. Далее к $val применяется регулярное выражение для проверки его корректности (блок 6). В случае отрицательного результата (обнаружения ошибки) в блоке 7, во-первых, к содержимому $xml присоединяется справа XML-элемент, имя которого берется из переменной $field, с атрибутом err = 'yes', во-вторых, $error устанавливается в значение true. В случае положительного результата в блоке 8 к содержимому $xml присоединяется такой же XML-элемент, но с атрибутом err = 'no'. Далее в блоке 9 проверяется, нажата ли кнопка сохранения введенных данных, и в этом случае выполняется запись значения $val в соответствующий узел DOM-объекта dom:DissInfo (блок 10). Если же в блоке 4 не обнаруживаются тестируемые данные, то в блоке 11 к содержимому $xml присоединяется XML-элемент с атрибутом err = 'no'.

Таким образом, в результате выполнения блоков 3–11 в $xml заносится имя тестируемого элемента данных с указанием, была ли обнаружена ошибка.

По завершении тестирования всех элементов данных блок 12 проверяет, была ли обнаружена хотя бы одна ошибка, и в этом случае в $xml дополнительно заносится элемент <error err = 'yes'> (блок 13). После этого отчет $xml дополняется закрывающим корневым тегом "</PersInfo>" и помещается в глобальную переменную Tr (блок 14). Переменная Tr далее используется в таблице стилей XSL-трансформации для отображения ошибок пользователю.

На заключительном этапе вновь проверяется, нажата ли кнопка сохранения данных и отсутствуют ли ошибки (блок 15), и в этом случае кнопка сбрасывается для предотвращения зацикливания (блок 16) и выполняется завершение процедуры с результатом 0 (блок 17), в противном случае выполняется завершение с ре-

зультатом false (блок 18). Результат 0 означает активность соответствующего элемента-перехода, а результат false – пассивность.

Модифицированная модель с использованием виджетов и контролёров данных

На рис. 3 представлена модифицированная модель, в которой для выполнения тех же функций, что и в модели на рис. 1, применены виджеты и контролёры данных, позволившие обойтись без программирования специальной процедуры-функции (см. рис. 2, листинг 1).

Как и исходная, данная модель содержит переходы, управляемые кнопками btn:StageInfo и btn:Cancel, а также doc-элемент, который задает местоположение таблицы стилей XSL-трансформации.

DOM-буфер. Вместо явно заданного dom-элемента в модифицированной модели предусмотрен виджет wdg:PersInfo-Buf, задающий DOM-объект неявным образом. Этот DOM-объект используется в качестве DOM-буфера входных элементов данных. Его содержимое определяется элементом-источником src:PersInfo, который предусматривает загрузку данных из ранее созданного DOM-объекта dom:DissInfo (сведения о диссертанте). Правило загрузки (атрибут join) предписывает загрузить перечисленные XML-элементы, являющиеся непосредственными потомками корневого элемента.

Схема XML-контента DOM-буфера приведена на рис. 4. Здесь обязательные элементы gen, fam, …, bio загружаются элементом-источником src:PersInfo, а необязательные элементы error формируются элементами-контролёрами виджета.

Рис. 3. Модифицированная модель с использованием виджетов и контролёров

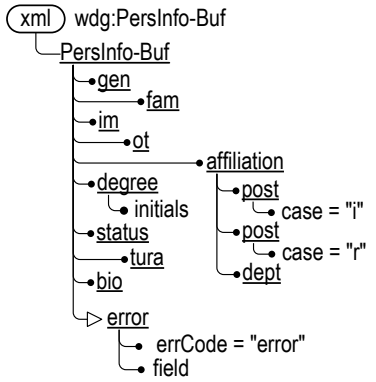


Рис. 4. Схема XML-содержимого DOM-буфера (нотация [12])

Элементы-контролёры. Внутри виджет-элемента (см. рис. 3) размещено 11 элементов-контролёров, соответствующих тестируемым элементам входных данных. Все контролёры содержат атрибут mode = "regular", что предотвращает их срабатывание на прологовом этапе интерпретации (mode = "prolog"), на котором пользователю отображается только форма для ввода элементов данных, а сами введенные данные еще не поступили.

У многих контролёров (gen, fam, im, ot, status, tura, bio) отсутствует атрибут targ и соответствующие тестируемые элементы данных сохраняются в DOM-буфере согласно правилу умолчания – в одноименных элементах второго уровня иерархии. Для остальных контролёров (postI, postR, dept, degree) это правило не применимо и адреса сохранения тестируемых данных заданы явно в атрибутах targ. Для одного элемента данных – tura (признак окончания аспирантуры / докторантуры) – потребовалось ввести значение по умолчанию с помощью атрибута default (этот элемент данных присутствует в массиве POST, если признак установлен, и отсутствует, – если признак сброшен).

Тестирование данных и фиксация ошибок выполняется с помощью элементов-приемников типа type = "errTest", размещенных внутри элементов-контролёров. Тестирование производится с помощью регулярных выражений, заданных в атрибутах regExp. Фиксация выявленных ошибок производится в соответствии с правилами умолчания: в DOM-объект родительского виджета, имя элемента-ошибки «error», код ошибки «error» (т. е. в соответствии со схемой на рис. 4).

Копирование данных при отсутствии ошибок выполняется с помощью элемента-приемника rcv:SaveInDissInfo. Проверенные данные из DOM-буфера копируются в DOM-объект dom: DissInfo, содержащий полные сведения о диссертанте. Копирование производится на режиме mode = "regular" методом method = "replNamesakes" («замена одноименных»). Этот метод предполагает, что в указанном элементе целевого DOM-объекта (атрибут targ) дочерние элементы заменяются на одноименные дочерние элементы указанного элемента DOM-буфера (атрибут from). Таким образом, выполняется операция, обратная загрузке DOM-буфера из DOM-объекта.

Отображение виджета выполняется с помощью элемента-приемника rcv:echo на втором проходе интерпретации путем XSL-трансформации содержимого DOM-буфера. На режиме mode = "prolog" пользователю отображается содержимое, загруженное из DOM-объекта dom: DissInfo, а на режиме mode = "regular" – данные, полученные от пользователя, если в них были обнаружены ошибки.

Сохранение данных в базе. Элемент-переход jmp:SaveGenInfo обеспечивает переход в состояние sta:SaveGenInfo, на котором содержимое DOM-объекта dom:DissInfo, куда предварительно были скопированы введенные пользователем данные, сохраняется в ADM в виде XML-документа. Указанный переход выполняется на режиме mode = "regular" в случае если, во-первых, не зафиксировано ни одной ошибки аварийного завершения (атрибут ctrlNoErrs), во-вторых, не зафиксировано ни одной ошибки тестируемых данных в DOM-буфере wdg:PersInfo-Buf (атрибут ctrlNotExists).

Сравнение моделей

По объему собственно исходная модель (см. рис. 1) заметно меньше модифицированной модели (см. рис. 3) – исходная модель требует 478 символов кода, в то время как модифицированная – 2 543 символа (в 5,2 раза больше). Однако при этом нужно учитывать процедуру непосредственного тестирования входных данных, составляющую в данном случае 7 319 символов (см. листинг 1). С учетом этого получается, что исходной модели требуется 478 + 7 319 = 7 797 символов, т. е. модифицированная модель требует в 3 раза меньше кода, чем исходная.

При этом отметим, что нет необходимости процедурного программирования, удалось обойтись моделью с декларативным заданием функций.

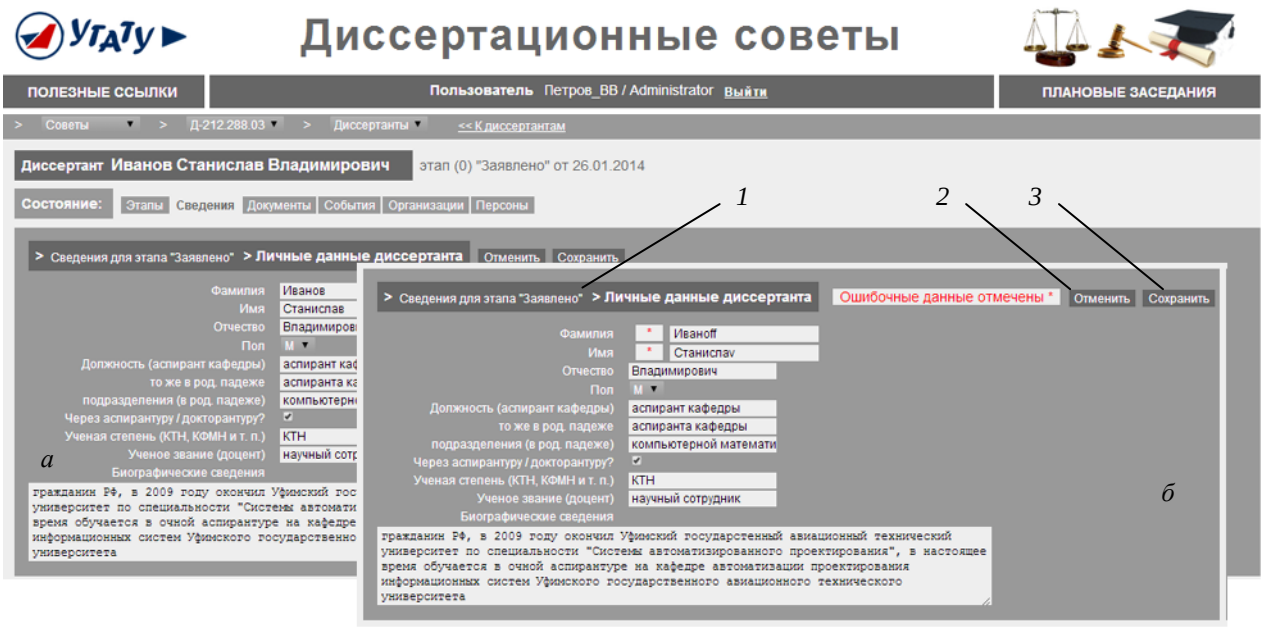


Рис. 5. Изображение, формируемое в состоянии sta:EditPersInfo в окне браузера (данные фиктивные): *а* – при отсутствии ошибок в даннных; *б* – при обнаружении ошибок. Элементы управления: *1* – кнопка btn:StageInfo; *2* –кнопка btn:Cancel; *3* – кнопка btn:SavePersInfo

СРАВНЕНИЕ XSL-ТРАНСФОРМАЦИИ БЕЗ ИСПОЛЬЗОВАНИЯ И С ИСПОЛЬЗОВАНИЕМ ВИДЖЕТОВ

Формирование изображения пользователю (создание пользовательского интерфейса), как в исходной модели, так и в модифицированной, происходит путем XSL-трансформации XML-контента DOM-объекта в выходной HTML-код, отправляемый в клиентский браузер. На рис. 5 представлен конечный результат этой процедуры в виде скриншотов экрана. Фрагмент изображения, формируемый в состоянии sta:EditPersInfo, является частью общего изображения, другие фрагменты которого формируются на более высоких уровнях иерархии. Изображение содержит элементы управления (кнопки, текстовые поля ввода) и поясняющие надписи. Об обнаруженных ошибках во введенных пользователем данных сообщают соответствующие надписи (рис. 5, *б*).

Рассмотрим особенности формирования этого изображения в исходной модели (без виджетов, см. рис. 1) и в модифицированной модели (с виджетом, см. рис. 3).

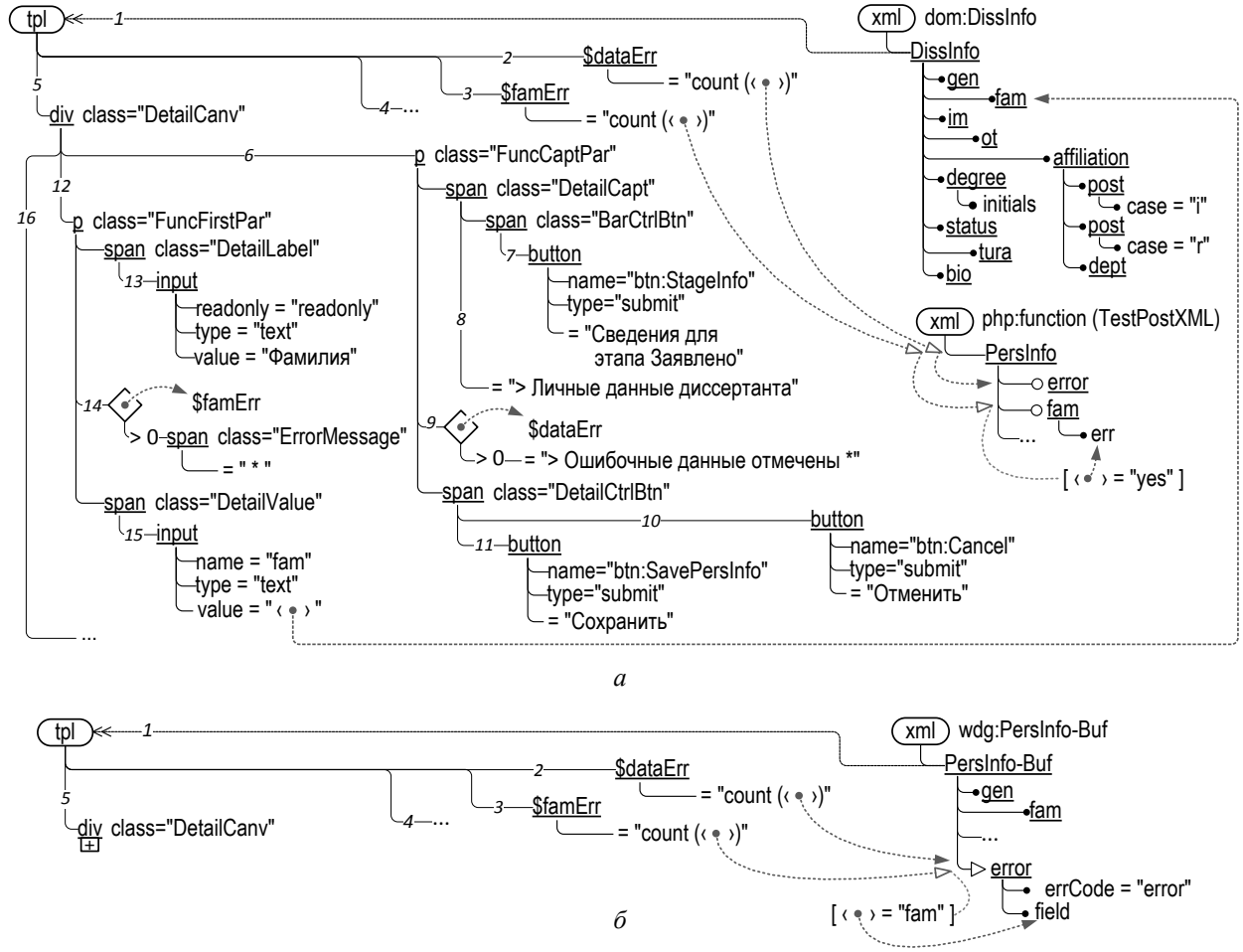
XSL-трансформация в исходной модели

На рис. 6, *а* приведена диаграмма модели XSL-трансформации при формировании HTML-кода изображения в окне браузера для исходной HSM-модели без использования виджетов (модель представлена в нотации [13], полный текст таблицы стилей трансформации на языке XSLT приведен в приложении, листинг 2). Справа на диаграмме представлена модель XML-документа, подвергаемого трансформации, а слева – усеченная модель шаблона трансформации.

В исходной HSM-модели sta:EditPersInfo трансформации подвергается документ DissInfo.xml, загруженный в DOM-объект dom:DissInfo (на модели приведены только элементы, участвующие в трансформации). С корневым элементом DissInfo этого документа ассоциирован шаблон трансформации, обеспечивающий требуемое преобразование.

Шаблон *1* составлен в стиле «извлекающей трансформации» [13, с. 278], он представляет собой HTML-код, в определенные места которого вставляются данные, извлекаемые из обрабатываемого XML-документа.

Для учета ошибок, обнаруженных в данных, предусмотрено обращение из шаблона к глобальному массиву ошибок Tr, в который, как показано выше, в исходной модели заносятся сведения об обнаруженных ошибках тестируемых данных. Для этого используется предусмотренная в XSLT возможность вызова из XSL-шаблона внешних функций, которые могут возвращать DOM-объект.



а

б

Рис. 6. Модель XSL-трансформации при формировании HTML-кода изображения в окне браузера:
а – для исходной HSM-модели без использования виджетов;
б – для модифицированной HSM-модели с использованием виджетов

В составе интерпретатора HSMI предусмотрена функция TestPostXML, которая создает DOM-объект, загружает в него данные из массива Tr и возвращает DOM-объект в качестве результата. На рис. 4, *а* (справа внизу) показана схема XML-содержимого DOM-объекта, возвращаемого функцией TestPostXML.

Шаблон *1* (рис. 6) запускается в соответствии со встроенным правилом трансформации при обнаружении в трансформируемом XML-документе элемента DissInfo. Первым делом в шаблоне создаются XSLT-переменные (*2, 3, 4*), значения которых отражают обнаруженные те или иные ошибки в данных.

Переменная *2* $dataErr отражает наличие любых выявленных ошибок. Для формирования ее значения вызывается внешняя функция TestPostXML и в возвращенном результате подсчитывается количество элементов error.

Переменная *3* $famErr отражает наличие выявленных ошибок в фамилии диссертанта. Для формирования ее значения вызывается внешняя функция TestPostXML и в возвращенном результате подсчитывается количество элементов fam, у которых атрибут err = "yes".

Остальные переменные *3* (на рис. 6 не показаны) аналогичным образом отражают ошибки, выявленные в других элементах данных («имя», «отчество», «пол» и т. д.).

Далее в шаблоне выполняется формирование элементов изображения. Используется блочная верстка изображения, при которой код изображения представляет собой иерархию блоков div и span, содержащих ссылки на стили отображения (атрибуты class), что позволяет управлять параметрами размещения и отображения блоков в окне браузера (цвета, размеры и начертания шрифтов и т. п.), используя технологию CSS.

Все элементы изображения упакованы в div-блок *5*, содержащий набор p-блоков (параграфов) *6, 12, 16*. Параграфы содержат определения элементов управления того или иного рода.

Параграф *6* задает элементы управления, относящиеся ко всему изображению виджета:

- кнопку *7* btn:StageInfo, обеспечивающую возврат к состоянию sta:DissGenList;
- текстовое поле *8*, заглавие;
- текстовое поле *9* – сообщение о наличии ошибок во введенных пользователем данных. Формирование этого поля производится с учетом значения XSL-переменной $dataErr: проверяется условие $dataErr > 0 и в случае его выполнения поле включается в результирующий код;
- кнопку *10* btn:Cancel, обеспечивающую отмену сделанных изменений в данных;
- кнопку *11* btn:SavePersInfo, обеспечивающую сохранение введенных данных.

Параграф *12* задает элементы управления для ввода фамилии диссертанта:

- текстовое поле *13* – заглавие;
- текстовое поле *14* – метку наличия ошибок во введенных данных. При формировании этого поля учитывается значения XSL-переменной $famErr: проверяется условие $famErr > 0 и в случае его выполнения метка ошибок включается в результирующий код;
- поле ввода *15* fam, обеспечивающее пользователю возможность изменения фамилии диссертанта. Начальное значение, которое отображается в этом поле, берется из одноименного элемента трансформируемого XML-документа.

Параграфы *16* аналогичным образом задают элементы управления для ввода остальных данных диссертанта (на рис. 6 не раскрыты, см. листинг 2).

Таким образом, в исходной модели для отображения пользователю сведений об обнаруженных ошибках во введенных данных приходится прибегать к «изощренному программированию» – вызовам из XSLT-шаблона внешней PHP-функции, возвращающей DOM-объект со сведениями об ошибках, загружаемыми из глобального массива. Заметим, что использование такой возможности – вызова внешних функций – не одобряется в технических руководствах по XSLT, поскольку создает потенциальную уязвимость для злонамеренных инъекций.

XSL-трансформация в модифицированной модели

На рис. 6, *б* приведена диаграмма модели XSL-трансформации для модифицированной HSM-модели с использованием виджетов. Трансформация запрограммирована так, чтобы максимально использовать конструкции исходной модели (отличающиеся строки XSLT-кода приведены в приложении, листинг 3). Используются те же XSL-переменные *2*, *3*, *4*, такой же div-блок *5*, отличия состоят в трансформируемых данных и получении используемых значений.

В модифицированной модели трансформации подвергается содержимое DOM-буфера wdg:PersInfo-Buf. Из него извлекаются как отображаемые значения, так и сведения об ошибках в данных. Значение переменной $dataErr формируется путем подсчета количества элементов error, а переменной $famErr и ей подобных – путем подсчета элементов error, у которых атрибут field имеет соответствующее значение.

Таким образом, модель трансформации получилась проще в том смысле, что использование DOM-буфера, формируемого виджетом, позволило обойтись без применения внешних функций, вызываемых из XSLT-шаблона. Хотя это и не сопровождается каким-либо заметным сокращением объема программного кода, такое упрощение потенциально повышает защищенность веб-приложения от злонамеренных инъекций.

КОЛИЧЕСТВЕННАЯ ОЦЕНКА ВОСТРЕБОВАННОСТИ ВИДЖЕТОВ

Приведенные выше результаты демонстрируют эффект применения виджетов на примере одного состояния динамической модели СОБД. Чтобы оценить востребованность виджетов для всего веб-приложения в целом, был проведен соответствующий анализ всей совокупности HSM-состояний динамической модели.

1. **Состояния, взаимодействующие с пользователем.** Динамическая модель веб-приложения содержит 110 состояний на разных уровнях иерархии. Из них в 59 состояниях (54 %) предусмотрено взаимодействие с пользователем (односторонний вывод изображения пользователю или вывод изображения с последующим вводом данных пользователя) – как раз тех состояний, в которых могут эффективно использоваться виджеты.

2. **Состояния с вводом данных пользователя.** Из 59 состояний, предусматривающих взаимодействие с пользователем, в 22 состояниях (37 %) предполагается двунаправленное взаимодействие, т. е. наряду с отображением данных предусмотрен ввод данных пользователя. Это состояния, в которых могут эффективно использоваться виджеты с контролёрами, они составляют 20 % от общего числа состояний.

3. **Количество тестирований входных данных.** В состояниях, предполагающих ввод данных пользователя, предусмотрено в целом 74 проверки входных данных, от 1 до 14 проверок в одном состоянии, т. е. в среднем на одно состояние приходится 3,4 проверки. Это характеризует количество элементов-контролёров, используемых в виджетах.

4. **Объем дополнительного программного кода**. В дополнение к динамической HSM-модели в веб-приложении используется PHP-код (AFL – Associated Functions Library), в исходной модели он имеет объем 139 650 символов, из которых 69 852 символа (50 %) – это код обработки пользовательских данных. Таким образом, при использовании виджетов в данном приложении вдвое уменьшается объем дополнительного программного кода.

Результаты анализа, свидетельствующие о востребованности виджетов и контролёров данных, строго говоря, справедливы для рассмотренного примера, и их, конечно, нельзя распространять на любые веб-приложения). Однако они дают основание утверждать, что существуют веб-приложения, и достаточно типичные веб-приложения, в которых применение иерархических виджетов дает заметный эффект.

ЗАКЛЮЧЕНИЕ

В данной статье на примере конкретного веб-приложения (исследовательского прототипа системы «Диссоветы») продемонстрированы преимущества использования иерархических виджетов для организации взаимодействия с пользователем.

Применение виджетов и входящих в их состав контролёров данных позволило в рассмотренном примере обойтись без дополнительного процедурного программирования функций проверки данных. Кроме того, это позволило обойтись без вызовов внешних функций из шаблонов XSL-трансформации.

В результате достигается упрощение программирования и уменьшение объема программного кода. Анализ показал высокую востребованность виджетов в приложениях, предусматривающих ввод и проверку пользовательских данных. В рассмотренном примере их использование сокращает в целом на 50 % объем дополнительного процедурного PHP-кода веб-приложения. Суммарный объем кода, приходящегося на одно состояние динамической модели, предусматривающее проверку пользовательских данных, уменьшился в 3 раза.

ПРИЛОЖЕНИЕ

Листинг 1

PHP-код проверки входных данных в состоянии sta:EditPersInfo исходной модели

```
001  case "TestPersInfoPost": //-------- Проверка данных персоны
002      $error = false;
003      $xml = "<PersInfo>";
004      // Проверка пола диссертанта ------------------------------
005      $field = "gen";
006      if ( isset ($_POST [$field]) )
007      {    $val = trim ($_POST [$field]);
008           if ( ! preg_match ("/^(m|f)\z/u", $val) )
009           {    $xml = $xml."<".$field." err='yes'/>";
010                $error = true;
011           } else
012                $xml = $xml."<".$field." err='no'/>";
013           if ( isset ($_POST ["btn:SavePersInfo"]) )
014                $GLOBALS ["dom:DissInfo"] ->
                         getElementsByTagname ($field) -> item (0) ->
                         nodeValue = $val;
016      } else
017           $xml = $xml."<".$field." err='no'/>";
018      // Проверка фамилии ---------------------------------
019      $field = "fam";
020      if ( isset ($_POST [$field]) )
021      {    $val = trim (htmlentities ($_POST [$field],
                                  ENT_QUOTES, 'UTF-8'));
022           if ( ! preg_match ("/^[А-ЯЁ][а-яё]+\z/u", $val) )
023           {    $xml = $xml."<".$field." err='yes'/>";
024                $error = true;
025           } else
026                $xml = $xml."<".$field." err='no'/>";
027           if ( isset ($_POST ["btn:SavePersInfo"]) )
028                $GLOBALS ["dom:DissInfo"] ->
                         getElementsByTagname ($field) -> item (0) ->
                         nodeValue = $val;
029      } else
030           $xml = $xml."<".$field." err='no'/>";
031      // Проверка имени -----------------------------------
032      $field = "im";
033      if ( isset ($_POST [$field]) )
034      {    $val = trim (htmlentities ($_POST [$field],
                                       ENT_QUOTES, 'UTF-8'));
035           if ( ! preg_match ("/^[А-ЯЁ][а-яё]+\z/u", $val) )
036           {    $xml = $xml."<".$field." err='yes'/>";
037                $error = true;
038           } else
039                $xml = $xml."<".$field." err='no'/>";
040           if ( isset ($_POST ["btn:SavePersInfo"]) )
                     $GLOBALS ["dom:DissInfo"] ->
                         getElementsByTagname ($field) -> item (0) ->
                         nodeValue = $val;
041      } else
042           $xml = $xml."<".$field." err='no'/>";
043      // Проверка отчества --------------------------------
044      $field = "ot";
045      if ( isset ($_POST [$field]) )
046      {    $val = trim (htmlentities ($_POST [$field],
                                       ENT_QUOTES, 'UTF-8'));
047           if ( ! preg_match ("/^[А-ЯЁ][а-яё]+\z/u", $val) )
048           {    $xml = $xml."<".$field." err='yes'/>";
049                $error = true;
050           } else
051                $xml = $xml."<".$field." err='no'/>";
052           if ( isset ($_POST ["btn:SavePersInfo"]) )
053                $GLOBALS ["dom:DissInfo"] ->
                         getElementsByTagname ($field) -> item (0) ->
                         nodeValue = $val;
054      } else
055           $xml = $xml."<".$field." err='no'/>";
056      // Проверка должности (именительный падеж) ------------
057      $field = "postI";
058      if ( isset ($_POST [$field]) )
059      {    $val = trim (htmlentities ($_POST [$field],
                                       ENT_QUOTES, 'UTF-8'));
060           if ( ! preg_match
                      ("/^[а-яё][а-яёА-ЯЁ ,;\-\.\(\)]+\z/u", $val) )
061           {    $xml = $xml."<".$field." err='yes'/>";
062                $error = true;
063           } else
064                $xml = $xml."<".$field." err='no'/>";
065           foreach ( $GLOBALS ["dom:DissInfo"] ->
                              getElementsByTagname ("post") as $x)
```

```
066             if ( $x -> getAttribute ("case") == "i" &&
                        $x -> parentNode -> parentNode ->
                                tagName == "DissInfo" )
067                 $x -> nodeValue = $val;
068     } else
069         $xml = $xml."<".$field." err='no'/>";
070     // Проверка должности (родительный падеж) ----------
071     $field = "postR";
072     if ( isset ($_POST [$field]) )
073     {   $val = trim (htmlentities ($_POST [$field],
                                ENT_QUOTES, 'UTF-8'));
074         if ( ! preg_match
                    ("/^[а-яё][а-яёА-ЯЁ ,;\-\.\(\)]+\z/u", $val) )
075         {   $xml = $xml."<".$field." err='yes'/>";
076             $error = true;
077         } else
078             $xml = $xml."<".$field." err='no'/>";
079         foreach ( $GLOBALS ["dom:DissInfo"] ->
                    getElementsByTagname ("post") as $x)
080             if ( $x -> getAttribute ("case") == "r" &&
                            $x -> parentNode -> parentNode ->
                                tagName == "DissInfo" )
081                 $x -> nodeValue = $val;
082     }  else
083         $xml = $xml."<".$field." err='no'/>";
084     // Проверка подразделения ------------------------------
085     $field = "dept";
086     if ( isset ($_POST [$field]) )
087     {   $val = trim (htmlentities ($_POST [$field],
                                ENT_QUOTES, 'UTF-8'));
088         if ( ! preg_match ("/^[а-яё][а-яёА-ЯЁ \(\)]+\z/u", $val) )
089         {   $xml = $xml."<".$field." err='yes'/>";
090             $error = true;
091         } else
092             $xml = $xml."<".$field." err='no'/>";
093         foreach ( $GLOBALS ["dom:DissInfo"] ->
                    getElementsByTagname ($field) as $x)
094             if ( $x -> getAttribute ("case") == "r" && $x ->
                        parentNode -> parentNode ->
                                tagName == "DissInfo")
                    $x -> nodeValue = $val;
095     } else
096         $xml = $xml."<".$field." err='no'/>";
097     // Проверка ученой степени -----------------------------
098     $field = "degree";
099     if ( isset ($_POST [$field]) )
100     {   $val = trim (htmlentities ($_POST [$field],
                                ENT_QUOTES, 'UTF-8'));
101         if ( (strlen ($val) > 0) &&
                    ( ! preg_match ("/^[ДК](Б|Т|СХ|И|Филол|
                    Филос|Иск|К|Пед|Психол|Ю|Полит|Фарм|
                    ГМ|Г|ФМ|Х|Э|С|М)Н\z/u", $val)) )
102         {   $xml = $xml."<".$field." err='yes'/>";
103             $error = true;
104         } else
105             $xml = $xml."<".$field." err='no'/>";
106         if ( isset ($_POST ["btn:SavePersInfo"]) )
107             $GLOBALS ["dom:DissInfo"] ->
                    getElementsByTagname ($field) -> item (0) ->
                        setAttribute ("initials", $val);
108     } else
109         $xml = $xml."<".$field." err='no'/>";
110     // Проверка ученого звания -----------------------------
111     $field = "status";
112     if ( isset ($_POST [$field]) )
113     {   $val = trim (htmlentities ($_POST [$field],
                                ENT_QUOTES, 'UTF-8'));
114         if ( (strlen ($val) > 0) &&
115             ( ! preg_match ("/^(доцент|профессор|научный
                    сотрудник)\z/", $val)) )
116         {   $xml = $xml."<".$field." err='yes'/>";
117             $error = true;
118         } else
119             $xml = $xml."<".$field." err='no'/>";
120         if ( isset ($_POST ["btn:SavePersInfo"]) )
121             $GLOBALS ["dom:DissInfo"] ->
                    getElementsByTagname ($field) -> item (0) ->
                    nodeValue = $val;
122     } else
123         $xml = $xml."<".$field." err='no'/>";
124     // Проверка флага аспирантуры --------------------------
125     $field = "tura";
126     if ( isset ($_POST [$field]) )
127         $val = "yes";
128     else
129         $val = "no";
130     if ( isset ($_POST ["btn:SavePersInfo"]) )
131         $GLOBALS ["dom:DissInfo"] ->
                getElementsByTagname ($field) -> item (0) ->
                nodeValue = $val;
132     // Проверка биографии ----------------------------------
133     $field = "bio";
134     if ( isset ($_POST [$field]) )
135     {   $val = trim ($_POST [$field]);
136         $val = stripslashes ($val);
137         if ( ! preg_match ("/^граждан/u", $val) )
138         {   $xml = $xml."<".$field." err='yes'/>";
139             $error = true;
140         } else
141             $xml = $xml."<".$field." err='no'/>";
142         if ( isset ($_POST ["btn:SavePersInfo"]) )
143         $GLOBALS ["dom:DissInfo"] ->
                getElementsByTagname ($field) -> item (0) ->
                nodeValue = $val;
144     } else
145         $xml = $xml."<".$field." err='no'/>";
146     // Проверки закончены ----------------------------------
147     if ( $error )
148         $xml = $xml."<error err='yes'/>";
149     $xml = $xml."</PersInfo>";
150     $GLOBALS ["Tr"] .= $xml;
151     if ( isset ($_POST ["btn:SavePersInfo"]) && (! $error) )
152     {   unset ($_POST ["btn:SavePersInfo"]);   // Сброс
153         return 0;
154     } else
155         return false;
```

Листинг 2

XSLT-код формирования HTML-кода изображения в состоянии sta:EditPersInfo исходной модели

```
001 <?xml version="1.0" encoding="utf-8"?>
002 <xsl:stylesheet    version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                      xmlns:php = "http://php.net/xsl" xsl:extension-element-prefixes = "php">
003 <xsl:output encoding = "utf-8" />
004 <xsl:template match = "/DissInfo">
005     <xsl:variable name = "dataErr"><xsl:value-of select = "count (php:function ('TestPostXML')/PersInfo/error)"/></xsl:variable>
006     <xsl:variable name = "famErr"><xsl:value-of select = " count (php:function ('TestPostXML')/PersInfo/fam)"/></xsl:variable>
007     <xsl:variable name = "imErr"><xsl:value-of select = " count (php:function ('TestPostXML')/PersInfo/im)"/></xsl:variable>
008     <xsl:variable name = "otErr"><xsl:value-of select = " count (php:function ('TestPostXML')/PersInfo/ot)"/></xsl:variable>
009     <xsl:variable name = "deptErr"><xsl:value-of select = " count (php:function ('TestPostXML')/PersInfo/dept)"/></xsl:variable>
010     <xsl:variable name = "postItErr"><xsl:value-of select = " count (php:function ('TestPostXML')/PersInfo/postI)"/></xsl:variable>
011     <xsl:variable name = "postRtErr"><xsl:value-of select = " count (php:function ('TestPostXML')/PersInfo/postR)"/></xsl:variable>
012     <xsl:variable name = "degreeErr"><xsl:value-of select = " count (php:function ('TestPostXML')/PersInfo/degree)"/></xsl:variable>
013     <xsl:variable name = "statusErr"><xsl:value-of select = " count (php:function ('TestPostXML')/PersInfo/status)"/></xsl:variable>
014     <xsl:variable name = "bioErr"><xsl:value-of select = " count (php:function ('TestPostXML')/PersInfo/bio)"/></xsl:variable>
015     <xsl:variable name = "fioI"><xsl:value-of select = "fam"/> <xsl:value-of select = "im"/> <xsl:value-of select = "ot"/></xsl:variable>
016     <xsl:variable name = "fI"><xsl:value-of select = "fam"/></xsl:variable>
017     <xsl:variable name = "iI"><xsl:value-of select = "im"/></xsl:variable>
018     <xsl:variable name = "oI"><xsl:value-of select = "ot"/></xsl:variable>
019     <div class="DetailCanv">
020     <p class="FuncCaptPar">
021         <span class="DetailCapt">
022             <b><span class="BarCtrlBtn"><button type="submit" title="Отменить и вернуться" name="btn:StageInfo"
                    value="1">Сведения для этапа &amp;quot;Заявлено&amp;quot;</button></span> > Личные данные диссертанта</b>  </span>
023         <xsl:if test = "$dataErr &gt;0"><span class="ErrorMessage">Ошибочные данные отмечены *</span></xsl:if>
024             <span class="DetailCtrlBtn"><nobr>
```

```
025                     <button type="submit" title="Отменить изменения" name="btn:Cancel" value="1">Отменить</button>
026                     <button type="submit" title="Сохранить и вернуться" name="btn:SavePersInfo" value="1">Сохранить</button>
027                 </nobr></span></p>
028         <p class="FuncFirstPar"><span class="DetailLabel"><input type="text" readonly="readonly" value="Фамилия"/></span>
029             <xsl:if test = "$famErr &gt;0"><span class="ErrorMessage"> * </span></xsl:if>
030             <span class="DetailValue"><input type="text" name="fam"><xsl:attribute name="value"><xsl:value-of
                    select = "fam"/></xsl:attribute></input></span></p>
031         <p class="FuncGentPar"><span class="DetailLabel"><input type="text" readonly="readonly" value="Имя"/></span>
032             <xsl:if test = "$imErr &gt;0"><span class="ErrorMessage"> * </span></xsl:if>
033             <span class="DetailValue"><input type="text" name="im"><xsl:attribute name="value"><xsl:value-of
                    select = "im"/></xsl:attribute></input></span></p>
034         <p class="FuncGentPar"><span class="DetailLabel"><input type="text" readonly="readonly" value="Отчество"/></span>
035             <xsl:if test = "$otErr &gt;0"><span class="ErrorMessage"> * </span></xsl:if>
036             <span class="DetailValue"><input type="text" name="ot"><xsl:attribute name="value"><xsl:value-of
                    select = "ot"/></xsl:attribute></input></span></p>
037         <p class="FuncGentPar"><span class="DetailLabel"><input type="text" readonly="readonly" value="Пол"/></span>
038             <select type="text" name="gen">
039                 <option value="m"><xsl:if test = "gen='m'"><xsl:attribute name="selected"/></xsl:if>М</option>
040                 <option value="f"><xsl:if test = "gen='f'"><xsl:attribute name="selected"/></xsl:if>Ж</option>
041             </select></p>
042         <p class="FuncGentPar"><span class="DetailLabel"><input type="text" readonly="readonly"
                    value="Должность (аспирант кафедры)"/></span>
043             <xsl:if test = "$postItErr &gt;0"><span class="ErrorMessage"> * </span></xsl:if>
044             <span class="DetailValue"><input type="text" name="postI"><xsl:attribute name="value"><xsl:value-of
                    select = "affiliation/post [@case = 'i']"/></xsl:attribute></input></span></p>
045         <p class="FuncGentPar"><span class="DetailLabel"><input type="text" readonly="readonly" value="то же в род. падеже"/></span>
046             <xsl:if test = "$postRtErr &gt;0"><span class="ErrorMessage"> * </span></xsl:if>
047             <span class="DetailValue"><input type="text" name="postR"><xsl:attribute name="value"><xsl:value-of
                    select = "affiliation/post [@case = 'r']"/></xsl:attribute></input></span></p>
048         <p class="FuncGentPar"><span class="DetailLabel"><input type="text" readonly="readonly"
                    value="подразделения (в род. падеже)"/></span>
049             <xsl:if test = "$deptErr &gt;0"><span class="ErrorMessage"> * </span></xsl:if>
050             <span class="DetailValue"><input type="text" name="dept"><xsl:attribute name="value"><xsl:value-of
                        select = "affiliation/dept [@case = 'r']"/></xsl:attribute></input></span></p>
051         <p class="FuncGentPar"><span class="DetailLabel"><input type="text" readonly="readonly"
                    value="Через аспирантуру / докторантуру?"/></span>
052             <input type="checkbox" name="tura"><xsl:if test = "tura='on'"><xsl:attribute name="checked"></xsl:attribute></xsl:if></input></p>
053         <p class="FuncGentPar"><span class="DetailLabel"><input type="text" readonly="readonly"
                    value="Ученая степень (КТН, КФМН и т.&amp;#160;п.)"/></span>
054             <xsl:if test = "$degreeErr &gt;0"><span class="ErrorMessage"> * </span></xsl:if>
055             <span class="DetailValue"><input type="text" name="degree"><xsl:attribute name="value"><xsl:value-of
                    select = "degree/@initials"/></xsl:attribute></input></span></p>
056         <p class="FuncGentPar"><span class="DetailLabel"><input type="text" readonly="readonly" value="Ученое звание (доцент)"/></span>
057             <xsl:if test = "$statusErr &gt;0"><span class="ErrorMessage"> * </span></xsl:if>
058             <span class="DetailValue"><input type="text" name="status"><xsl:attribute name="value"><xsl:value-of
                    select = "status"/></xsl:attribute></input></span></p>
059         <p class="FuncGentPar"><span class="DetailLabel"><input type="text" readonly="readonly"
                    value="Биографические сведения"/></span></p>
060         <xsl:if test = "$bioErr &gt;0"><span class="ErrorMessage"> * </span></xsl:if>
061             <span class="DetailValue"><textarea name="bio" rows="5" cols="100"><xsl:value-of select = "bio"/></textarea></span>
062         </div>
063 </xsl:template>
064 </xsl:stylesheet>
```

Листинг 3

Модифицированные строки XSLT-код формирования HTML-кода изображения

```
004 <xsl:template match = "/PersInfo-Buf">
005         <xsl:variable name = "dataErr"><xsl:value-of select = "count(error)"/></xsl:variable>
006         <xsl:variable name = "famErr"><xsl:value-of select = "count(error[@field='fam'])"/></xsl:variable>
007         <xsl:variable name = "imErr"><xsl:value-of select = "count(error[@field='im'])"/></xsl:variable>
008         <xsl:variable name = "otErr"><xsl:value-of select = "count(error[@field='ot'])"/></xsl:variable>
009         <xsl:variable name = "deptErr"><xsl:value-of select = "count(error[@field='dept'])"/></xsl:variable>
010         <xsl:variable name = "postItErr"><xsl:value-of select = "count(error[@field='postI'])"/></xsl:variable>
011         <xsl:variable name = "postRtErr"><xsl:value-of select = "count(error[@field='postR'])"/></xsl:variable>
012         <xsl:variable name = "degreeErr"><xsl:value-of select = "count(error[@field='degree'])"/></xsl:variable>
013         <xsl:variable name = "statusErr"><xsl:value-of select = "count(error[@field='status'])"/></xsl:variable>
014         <xsl:variable name = "bioErr"><xsl:value-of select = "count(error[@field='bio'])"/></xsl:variable>
```

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. **Канашин В. В., Миронов В. В.** Иерархические виджеты: алгоритмы контроля данных пользователя в веб-приложениях на основе ситуационно-ориентированных баз данных // Вестник УГАТУ. 2014. Т. 18, № 1 (62). С. 204–213. **[** V. V. Kanashin and V. V. Mironov, “Hierarchical widgets: user data control algorithms in web applications on the basis of situation-oriented databases,” (in Russian), *Vestnik UGATU*, vol. 18, no. 1 (62), pp. 204-213, 2014. **]**

13. **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: experience of use in the web application on the basis of situation-oriented database.

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. 2 (63), pp. 185-196, 2014. ISSN 2225-2789 (Online), ISSN 1992-6502 (Print).

Abstract: In the previous articles of authors the concept, models and algorithms of the hierarchical widgets intended for creation of difficult structured user interface in web applications on the basis of the situation-oriented databases (SODD) are offered. Here results of use of hierarchical widgets are discussed when developing the web application serving activity of dissertation councils of higher education institution. On the example of dynamic model states the technology of user data testing organization and messages formation are considered in detail. It is shown that widgets using simplifies an organization of XSL transformation and reduces costs of additional programming of user data verification functions.

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).