Расширение языка WebML средствами моделирования интерфейсов полнофункциональных Web-приложений, интенсивно работающих с данными*

А. Н. Иванов
iw@tercom.ru

Д. В. Кознов
dkoznov@yandex.ru

М. Г. Тыжгеев
mtyzhgeyev@openwaygroup.com

Полнофункциональные Web-приложения (Rich Internet Applications, RIA) — это новое и перспективное направление эволюции современных информационных систем. RIA-приложения являются следующим поколением Web-приложений, все меньше отличаясь от настольных (desktop) приложений — по внешнему виду и возможностям пользовательского интерфейса, по сложности бизнес-логики и проч. RIA-приложения объединяют в себе черты мультимедиа систем, настольных информационных систем и обычных (HTML-ориентированных) Web-приложений. Соответственно необходимы новые средства моделирования этих систем. В данной работе мы интегрируем средства моделирования пользовательского интерфейса технологии REAL-IT, предназначенной для настольных приложений интенсивной обработки данных, в подход WebML/WebRatio, ориентированный

* Исследование выполнено при частичной финансовой поддержке РФФИ (грант 05-0951/07).

на Web-приложения интенсивной обработки данных. Подход WebML является стандартом де-факто в области моделирования Web-приложений, однако он плохо приспособлен для создания полнофункциональных интерфейсов, что важно для RIA-приложений. Мы расширяем язык WebML средствами REAL-IT, что позволяет создавать компактные модели интерфейса, используя распространенные типы экранных форм, а также встраивать в них сложные фильтры. Простота и выразительная сила этих моделей — залог применимости модельного подхода на практике.

Введение

В настоящее время активно развивается рынок Web-приложений. Все больше различных услуг предоставляется через Интернет, например заказ авиабилетов, бронирование гостиниц, подготовка договоров, доступ к различной справочной информации. Классические информационные системы в различных компаниях также становятся Web-приложениями, предоставляя доступ к ресурсам компании не только внутри локальной сети, но также из произвольной точки Интернета.

Все это привело к созданию концепции полнофункциональных Web-приложений (Rich Internet Applications, RIA), которые по возможностям пользовательского интерфейса не уступают настольным (desktop) приложениям, имеют данные и бизнес-логику на стороне клиента, умеют своевременно обновлять экран, перерисовывая его лишь частично (в том месте, где произошли изменения), функционировать как с подключенным Интернетом, так и без [8, 11]. В настоящее время активно развиваются технологии для создания и функционирования RIA-приложений — AJAX, MacroMedia Flash/Flex, Microsoft SilverLight и др., однако существенно отстают высокоуровневые средства разработки и моделирования, как отмечалось в работах [7, 13].

Сегодня стандартом де-факто в области Web-моделирования является подход WebML [13], включающий в себя ряд моделей, шаблоны проектирования и CASE-пакет WebRation [5], поддерживающий автоматическую кодогенерацию[1]. В последнее время язык WebML расширен средствами моделирования RIA-приложений [7,

[1] Обзор и анализ других подходов к моделированию Web-приложений можно найти, например, в работе [13].

15]. Однако в нем отсутвуют средства задания полнофункциональных интерфейсов — даже для небольших HTML-ориентированных приложений гипертекстовые модели, на которых в WebML задается пользовательский интерфейс, получаются очень громоздкими, а для сложных RIA-приложений они оказываются просто гигантскими, их трудно создавать и еще труднее анализировать, изучать, обсуждать и т. д.

Для решения этой задачи WebML интегрируется с методом RUX [12]. Однако RUX делает акцент на объединении аспекта баз данных и мультимедиа [10], не предлагая высокоуровневых абстракций для задания полнофункциональных интерфейсов. Вместе с тем RUX предоставляет средства для создания детальной модели пользовательского интерфейса в стиле редакторов экранных форм. Подобные редакторы имеются сейчас практически в любой среде разработки ПО, например в Microsoft Visual Studio. Тем не менее трудоемкость разработки пользовательского интерфейса для крупных систем (а именно в эту сторону двигаются RIA-приложения) требует введения дополнительных модельных абстракций, позволяющих задавать емкие, лаконичные, хорошо читаемые модели с последующей эффективной кодогенерацией.

Разработка модельно-ориентированных средств создания пользовательских интерфейсов ведется давно (обзор этой области можно найти, например, в [9]). Однако главными проблемами здесь оказываются сложность и громоздкость моделей для реальных систем, а также негибкость средств автоматической кодогенерации. В результате на практике оказывается, что использование этих подходов не дает выигрыша в производительности. Это привело к упадку исследований в данной области.

При разработке технологии REAL-IT [3, 4, 9] мы во многом решали именно эту проблему. Мы сузили класс целевых приложений, ограничившись массовыми, но простыми информационными системами, имеющими базу данных и несложный интерфейс для ее заполнения/просмотра/редактирования. Мы типизировали интерфейс таких приложений и предложили пользователю, помимо классических модельных средств (несколько видов диаграмм языка UML для задания базовой архитектуры приложения), систему диалоговых мастеров, позволяющих по базовым моделям быстро и удобно создавать модель пользовательского интерфейса с последующей автоматической кодогенерацией. То есть за счет сужения класса задач удалось создать простые абстракции и эффективный

способ их использования, дающий выигрыш в реальной разработке [9].

В рамках данной работы мы предлагаем решение задачи моделирования полнофункциональных интерфейсов для RIA-приложений с помощью WebML, расширяя последний средствами REAL-IT. Мы вводим в WebML новые конструкции для удобства задания типовых экранных форм, введенных в REAL-IT, — карточек, списков и форм-отношений «многие-ко-многим»[2] [3]. Мы расширяем гипертекстовую модель WebML новым видом модуля — ComlexFilterUnit — для задания фильтров с зависимыми полями. Для удобства формализации дополнительных ограничений на поля фильтров (с последующим их использованием при кодогенерации), основываясь на [2], мы вводим в WebML специальную модель ограничений, которая позволяет существенно упростить модельные спецификации и одновременно повысить функциональность генерируемого по ним программного кода. Мы также предлагаем специальный шаблон проектирования для задания формы-отношения «многие-ко-многим», вводим в модель данных WebML n-артные связи «многие-ко-многим» с атрибутами.

1. Контекст

1.1. Подход WebML/WebRatio

Данный подход развивается в Политехническом институте города Милана (Италия) с 2000 года[3]. О нем опубликовано несколько десятков работ, подход реализован в виде программного средства WebRatio, поддерживающего автоматическую генерацию кода по диаграммам для платформы JEE/Tomcat и СУБД Oracle/MS SQL Server, а последняя версия продукта интегрирована со средой разработки Eclipse. Язык моделирования WebML состоит из следующих частей.

Модель данных (data model) — средство моделирования структуры базы данных, является вариантом модели сущность—связь. Можно задавать данные, хранимые как на сервере, так и на клиенте.

Модель запросов (derivation model) — средство для расширения модели данных вычислимыми конструкциями, такими как кон-

[2]Форма для редактирования связей «многие-ко-многим».

[3]http://ww.webml.org/

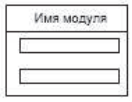
Название	Описание	Нотация
DataUnit	Публикует информацию об одном экземпляре сущности модели данных.	
MultuDataUnit	Публикует информацию о множестве экземпляров сущностей модели данных, но без возможности выбора какой-либо из них.	
IndexUnit	Публикует информацию о множестве объектов модели данных с возможностью выбора одной из них или целого набора.	
ScrolUnit	То же, что Index Unit, но с добавлением возможности прокрутки списка элементов.	
EntryUnit	Предназначен для ввода данных в Web-систему. Запись в базу данных осуществляется не самим этим модулем: он выдает значения, которые используются как параметры в операциях модели контента.	

Рис. 1. Контент-модули языка WebML.

струкция языка SQL под названием view, а также вычислимыми атрибутами и связями. Эта модель является текстовым языком запросов к модели данных, ее конструкции используются в двух следующих типах моделей для связи интерфейсных модулей и операций с моделью данных.

Гипертекстовая модель (hypertext model) — служит для задания схемы интерфейса Web-приложения, т. е. позволяет определить страницы, находящиеся на них элементы управления — контент-модули (content units), — их связь с базой данных и навигацию между ними. Детали внешнего вида интерфейса (вид отображения элементов управления, их цвета, геометрические размеры и проч.) задаются с помощью стилей уже на уровне модельного инструмента. Бизнес-логика Web-приложения задается с помощью модели кон-

тента (см. ниже), которая в действительности составляет с гипертекстовой моделью неразрывное целое. Набор предопределенных контент-модулей WebML представлен и кратко описан на рис. 1[4].

Контент-модули можно группировать в страницы (pages), а те, в свою очередь, в области (areas) и рабочие места системы (site views), например рабочее место администратора, неавторизованного пользователя, авторизованного пользователя. Таким образом, WebML поддерживает структурную декомпозицию[5]. Страницы и контент-модули можно соединять связями (links), которые могут передавать управление и/или данные.

Модель контента (content model) — определяет дополнительные конструкции — операции (operation units), которые можно использовать в гипертекстовой модели для задания поведения экранных форм. Операции не могут присутствовать на странице, но их можно соединять со страницами, находящимися на страницах контент-модулями и друг с другом при помощи связей (links). На рис. 2 представлены базовые операции WebML[6].

Кроме них в WebML есть еще дополнительные операции:

- часто используемые функции: входа в систему (login) и выхода из нее (logout), распределения прав между пользователями системы;
- операции для задания хранимой процедуры;
- операции для хранения и доступа к данным, полученным в результате определенного запроса к базе данных (селекторы — selectors);
- операция логического ветвления и проч.

Операции не входят в страницы, но могут объединяться в транзакции, выполняться на сервере и на клиенте.

1.2. Технология REAL-IT

Технология REAL-IT предназначена для быстрой разработки (моделирования и автоматической генерации) приложений, бизнес-логика которых целиком определяется схемой данных. В этом случае все целевое приложение является лишь средством заполнения и редактирования данных и не содержит сложных функций по их

[4]В рамках пакета WebRatio этот набор существенно расширен.

[5]Однако на практике этих средств декомпозиции оказывается недостаточно, но это — тема отдельного исследования.

[6]В рамках пакета WebRatio этот набор операций расширяется.

Название	Описание	Нотация
Delete	Удаляет экземпляр сущности.	
Create	Создает экземпляр сущности.	
Modify	Изменяет связь между экземплярами двух сущностей.	
Connect	Создает связь между экземплярами двух сущностей.	
Disconnect	Удаляет связь между экземплярами двух сущностей	
Selector	Делает запрос к схеме данных, но не отображает извлеченную информацию в интерфейсе. Используется как в страницах, так и вне их для вспомогательных запросов.	

Рис. 2. Операции языка WebML.

обработке. На практике таких небольших приложений создается очень много, а почти все крупные информационные систем содержат подобные подсистемы. В то же время интерфейсы таких системы содержат десятки и сотни однотипных экранных форм, которые непросто создавать и поддерживать. REAL-IT позволяет здесь существенно снизить трудозатраты.

1.2.1. Типовые экранные формы

Пусть в схеме данных приложения есть сущность A, и для нее создается экранная форма для ввода/редактирования атрибутов

ее экземпляров. Эта форма называется *карточкой* (пример см. на рис. 3, *а*). Если у сущности есть связь с множественностью противоположного конца большей, чем единица, то в карточке появляется *список* объектов другой сущности, связанных с этим экземпляром сущности A. Список может быть встроенным в карточку и вынесенным. Пример встроенного списка показан на рис. 3, *в*, а внесенный список открывается, например, при нажатии кропки Add на этой же форме.

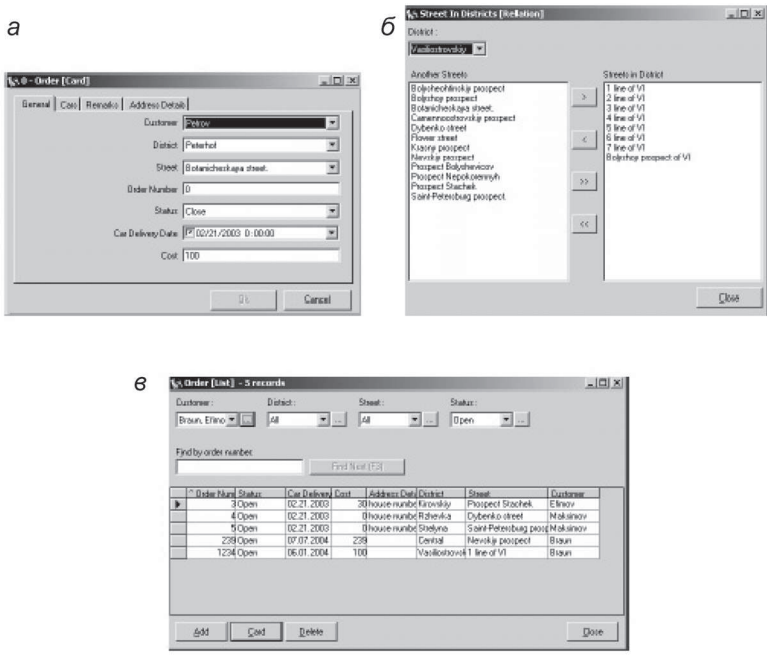


Рис. 3. Типовые экранные формы REAL-IT:
а — карточка; *б* — форма-отношение; *в* — список.

Связи «многие-ко-многим» между таблицами переходят в *формы-отношения*. Пусть у нас в модели данных есть несколько сущностей, связанных отношением «многие-ко-многим» арности n, где $n >= 2$. Тогда часто возникает потребность предоставить ин-

терфейс для добавления/удаления связей объектов этих сущностей. Общий подход к построению соответствующей экранной формы выглядит так. Выделяются две стороны: f-сторона (fixed) и e-сторона (editing). *F-сторона* позволяет зафиксировать объекты $n-1$ конца n-арного отношения «многие-ко-многим». *E-сторона* — это объекты оставшейся стороны. Именно их можно добавлять/удалять в связь для зафиксированных $n-1$ объектов f-стороны. На объекты f-стороны могут быть наложены фильтры, которые представляются связными полями выбора (comboboxes). Объекты e-стороны можно представить в виде контент-модуля WebML под названием IndexUnit — списка с кнопками выбора (checkboxes), а также в виде двух списков с кнопками перемещения, как показано на рис. 3, *б*.

Рассмотрим пример, представленный на рис. 3, *б*. Сущность District связана отношением «многие-ко-многим» с сущностью Street — в одном районе может быть несколько улиц, а одна улица может пролегать через несколько районов. В поле выбора District мы выбираем какой-то определенный район, и после этого в списке справа, сразу под ним, отображаются все улицы, которые к нему относятся. А в списке слева показываются остальные улицы (т. е. все содержимое сущности Street) и из него вправо можно что-то добавить. И наоборот, слева можно что-то удалить. Это и будет редактированием связей в данном отношении «многие-ко-многим».

Составной частью форм-отношений и списков является *фильтр*. На рис. 3, *б* поле выбора District является фильтром. Фильтр может быть сложным (пример показан на рис. 3, *в*). Этот фильтр содержит целый набор полей выбора, некоторые из них связаны друг с другом. Так, при выборе определенного района (поле выбора District) в поле выбора Street будут предложены для выбора только те улицы, которые связаны с этим районом, а не вообще все улицы города.

Выше мы рассмотрели *фильтр-представление*, осуществляющий фильтрацию набора данных для их предъявления пользователю с последующим использованием в запросах к базе данных. Существует другой вид фильтра — *фильтр ввода*, который предназначен для ввода данных на основе уже существующих значений. Он также состоит из полей выбора и в зависимости от выбора в одном поле в другом предлагаются определенные значения. Так, на рис. 3, *а* поля ввода Distric и Street являются связными.

1.2.2. Ограничения на модель данных

Классическая схема данных, выполненная с помощью диаграмм сущность-связь или какой-либо их разновидности, оставляет неопределенными много вариантов, которые на практике, в «живой» базе данных, не должны существовать. Например, многие объекты не могут быть одновременно связаны всеми возможными по схеме связями с другими объектами. В таких случаях можно пользоваться языком ограничений типа OCL [1], но последний гораздо богаче, чем те возможности, которые требуются здесь. Вместе с тем хотелось бы иметь ограничения в доступном, наглядном и хорошо воспринимаемом виде. Мы выбрали графическое представление.

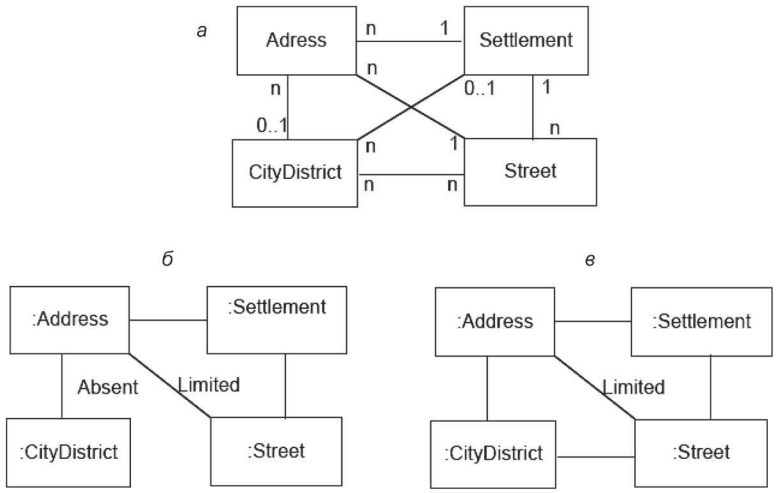


Рис. 4. Пример ограничений в REAL-IT:
а — схема данных; *б* — ограничение 1: городских районов нет;
в — ограничение 2: городские районы есть.

Рассмотрим пример. На рис. 4, *а* представлен фрагмент схемы данных для адреса сотрудника компании. Адрес включает два разных случая: 1) сотрудник проживает в большом городе, и его адрес содержит район этого города; 2) сотрудник проживает либо в маленьком городе, либо в сельской местности, и тогда район в

адресе отсутствует. Две эти ситуации изображены на рис. 4, *б, в*, которые являются диаграммами объектов UML и задают конфигурацию реальных записей и их связей в базе данных, в рамках которой определенная связь должна быть. Эта определяемая связь помечается словом «limited». В нашем примере такой связью является связь адреса и улицы. То есть для двух записей из таблиц Address и Street связь между ними реализуется, если в базе данных есть одна из конфигураций записей, связанных между собой, как показано на рис. 4, *б, в*. В случае, представленном на рис. 4, *б*, связи между районом и адресом не должно быть.

Если мы захотим определить аналогичные ограничения для отношения между районом и населенным пунктом (сущностями City-District и Settlement), то нам нужно создать аналогичные ограничения, определяющие контекст и альтернативы именно для этого отношения.

1.2.3. Процесс разработки

Модель данных в REAL-IT задается с помощью диаграмм классов UML. Ограничения на эту модель определяются с помощью диаграмм объектов, реализующих язык ограничений из [2]. Модель пользовательского интерфейса строится с помощью специальных программных инструментов, позволяющих задать, для каких таблиц нужны карточки, по каким связям создавать списки, а по каким — формы-отношения, а также задать различные параметры этих экранных форм. Основываясь на этой информации, REAL-IT автоматически генерирует диаграммы классов UML, описывающие созданную модель интерфейса. Первоначально в REAL-IT не было диалоговых инструментов для создания модели интерфейса, и пользователям предлагалось задавать ее с помощью UML. Однако этот подход не оправдал себя на практике — процесс моделирования занимал много времени. Тогда и были созданы соответствующие программные инструменты для упрощения задания интерфейса.

По UML-моделям REAL-IT автоматически генерирует схему базы данных на SQL/DDL, объектно-ориентированный программный интерфейс к ней на Microsoft C++ и VisualBasic, а также экранные формы и код приложения [3].

2. Расширение модели данных

Авторы WebML не используют n-арные отношения и сущности-отношения, хотя во многих вариантах диаграмм сущность-связь они есть (например, в модели классов UML [1]). Мы предлагаем ввести эти конструкции в модель данных WebML, поскольку они повышают выразительную силу модели и могут использоваться при автоматической генерации кода. В процессе концептуального моделирования, при появлении n-арных связей, появляется возможность задавать их непосредственно, а не «реализационно». Например, факт наличия у преподавателя ученой степени удобно определять как наличие связи между тремя сущностями — преподавателем, видом наук (физ.-мат., техн. и т. д.) и видом степени (кандидат/доктор). Кроме того, единое, высокоуровневое представление n-арных отношений «многие-ко-многим» с возможностью задавать атрибуты этого отношения через сущность-отношение позволяет нам определить единый шаблон для формы-отношения «многие-ко-многим».

На рис. 5 приведен пример сущности-отношения. Сущности «Студент» и «Экзамен» связаны отношением «многие-ко-многим» — один экзамен могут сдавать многие студенты, а один студент может сдавать много разных экзаменов. У этой связи есть атрибут — оценка, полученная студентом по экзамену. Мы предлагаем поместить этот атрибут в специальную сущность, связанную с отношением (в данном случае это сущность «Оценка»)[7].

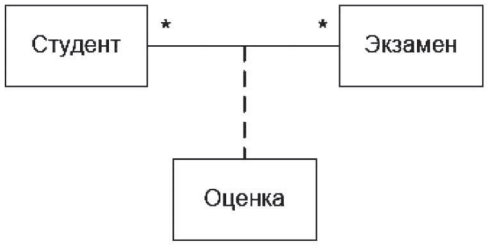


Рис. 5. Пример сущности-связи «многие-ко-многим».

[7]При этом мы пользуемся тем же способом, что и в модели классов UML (там это называется класс-ассоциация [1]).

Бинарное отношение «многие-ко-многим» можно представить через дополнительную сущность и пару отношений «один-ко-многим» этой сущности с прежними. Тогда атрибуты этого отношения будут содержаться в новой сущности. Однако n-арное отношение, где $n > 2$, так не представляется, а между тем такие отношения удобны, встречаются на практике и по ним также требуется создавать формы «многие-ко-многим» — все сущности, кроме одной, помещаются в фильтр, а оставшаяся редактируется.

3. Моделирование фильтров и связных полей ввода

3.1. Контент-модуль ComplexFilterUnit

В языке WebML хотелось бы иметь средства для моделирования сложных фильтров с зависимыми полями. Фильтры в WebML можно задавать с помощью контент-модуля EntryUnit, но при этом нет возможности задать также зависимости между полями. Для задания фильтров с зависимыми полями введем новый контент-модуль ComplexFilterUnit, определив его так:

```
<ComplexFilterUnit> :: = ComplexFilterUnit <Name> <MainCaption>
{
  FilterType <browse/input>
  source <Entity>;
  <Body>
}
```

< `Name` > — это модельное имя конструкции, имеющей тип ComplexFilterUnit.

< `MainCaption` > — это имя группы зависимых полей фильтрации, отображаемое в итоговом, сгенерированном по данной конструкции фрагменте интерфейса.

`FilterType` < `browse/input` > определяет вид фильтра. Возможны два варианта: 1) `browse`, если задается фильтр-представление; тогда значения его полей используются как параметры селекторов выбора в других модулях модели; 2) `input`, если задается фильтр ввода; тогда информация, выбранная пользователем Web-интерфейса в полях ComplexFilterUnit, заносится в базу данных Web-приложения (с использованием модулей Create, Modify и др.).

< `Entity` > — это имя сущности, которую фильтрует данный фильтр. Каждый ComplexFilterUnit фильтрует объекты какой-

нибудь одной сущности — по ее атрибутам и/или связям. От этой сущности сложный фильтр, по отношениям в модели данных, может захватить для фильтрации некоторый фрагмент модели данных.

Конструкция ComplexFilterUnit состоит из набора фильтрующих полей, каждое из которых является ограничением на выборку объектов из сущности <Entity>:

```
<Body> :: = <Field>+
```

Поля фильтра бывают двух типов — по атрибутам сущности и по связям:

```
<Field> :: = <AttributeField> | <RelationField>
```

Каждое поле фильтра определяет поле выбора, и пользователь Web-интерфейса совершает там выбор — единичный или множественный. Таким образом, каждое поле выдает одно значение или коллекцию. Все эти значения являются выходными параметрами ComplexFilterUnit, которые используются другими модулями в соответствии с типом фильтра.

Поле фильтрации по атрибуту задается так:

```
<AttributeField> ::= attributefield [<FieldCaption>],
                     <SourceAttribute>,
                     [<Format>] [<AttributeOperation>]
```

<FieldCaption> — это статический текст, который является именем фильтрующего поля и отображается в целевом интерфейсе рядом с полем. Если конструкция ComplexFilterUnit состоит лишь из одного поля, то <FieldCaption> не нужен — его роль выполняет <MainCaption>.

<SourceAttribute> — это имя атрибута сущности, которое фильтруется данным полем фильтра.

<Format> — это формат отображения результатов фильтрации. Например, способ отображать десятичные числа (с точкой или запятой), формат даты и т. д.

<Attributeperation> — это выражение (на языке SQL), которое задает условие фильтрации атрибута сущности.

Поле фильтрации по отношению задается так:

```
<RelationField> ::= relationfield  [<FieldCaption>],
                    <SourceRelation>, [<Display>]  [<Format>]
                    [<RelationOperation>] {<EnclosedFields> }
```

<SourceRelation> — это имя связи фильтруемой сущности, которая фильтруется данным полем.

<Display> — это селектор (выражение на языке SQL), который указывает, какой именно атрибут (набор атрибутов) той, другой сущности или связанных с ней нужно показывать в списке результатов фильтрации. Например, если мы из сущности «Группа» фильтруем студентов, имеющих академическую задолженность (фильтруем связь из сущности «Группа» в сущность «Студент»), то показывать нужно не ID студентов, а их фамилии, имена, отчества.

<Format> означает то же, что и выше.

<Relationperation> — это выражение (на языке SQL), которое задает условие фильтрации отношения.

На отношение можно накладывать дополнительные ограничения по фильтрации, используя другие связи фильтруемой второй сущности, связанной этим отношением с первой, фильтруемой. То есть с полем фильтрации по связи можно связывать дополнительные поля.

```
<EnclosedFields> ::= <Filter>+
```

Так и строятся связные поля — то, ради чего была создана конструкция ComplexFilterUnit. При этом фильтруемой сущностью для <Enclosedfields> будет та, которая получена по отношению. Изображаться новый модуль будет, как показано на рис. 6.

Рис. 6. Графическая нотация
для модуля ComplexFilterUnit.

3.2. Пример

Рассмотрим модель данных, представленную на рис. 4, *а*. Пусть сущность Address дополнительно имеет атрибут — номер дома (BuildingNumber). И пусть мы хотим задать на объекты сущности

Address фильтр со следующими зависимыми полями: имя населенного пункта, название улицы, номер дома. Это можно сделать с помощью следующей спецификации:

```
ComplexFilterUnit  AddressFilter
{
  filter true
  source Address;
  {
    attributefield  "Builging", BuildingNumber, "", equal;
     relationfield "Street" , Address-Street,
     "Select Name FROM Street", "", in
    {
        relationfield "City", Street-City, "Select Name FROM City",
        "", in
     }
   }
}
```

Результирующая форма для этой спецификации показана на рис. 7.

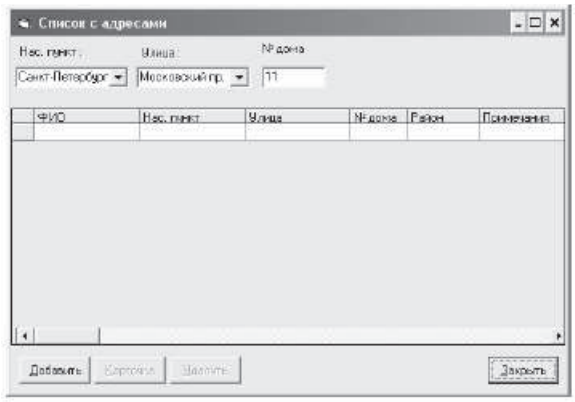


Рис. 7. Пример экранной формы, созданной на основе модуля ComplexFilterUnit.

3.1. Модель ограничений

Мы предлагаем добавить в WebML модель ограничений REAL-IT, чтобы задавать дополнительные ограничения на поля фильтров

и использовать эту информацию при генерации целевого кода. Эту модель мы добавляем как новый тип диаграмм. Чтобы не загромождать изложения, мы не будем делать это формально.

4. Карточки и списки

Полнофункциональные карточки и списки можно легко строить, используя различные контент-модули WebML, а также введенный нами модуль ComplexFilterUnit.

5. Шаблон для реализации формы-отношения «многие-ко-многим»

В данном случае мы создаем шаблон проектирования, а не новую конструкцию, как это было в случае с ComplexFilterUnit. Нам удалось сделать этот шаблон компактным, независимым от параметров его участников, в частности от арности связи «многие-ко-многим». Этот шаблон представлен на рис. 8.

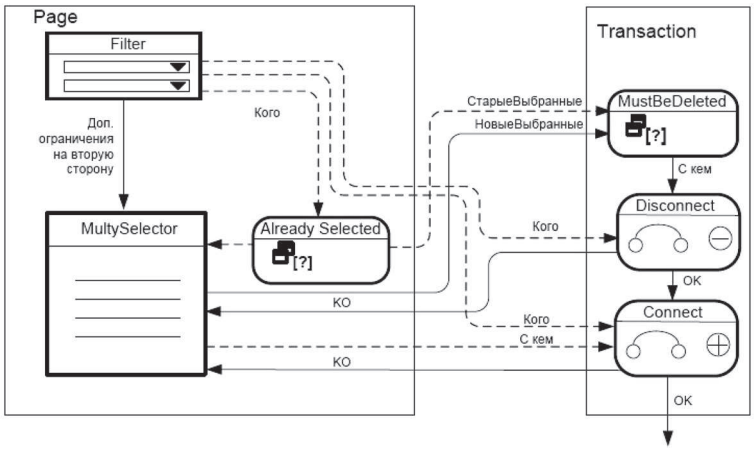


Рис. 8. Шаблон для формы-отношения «многие-ко-многим».

Он состоит из одной страницы, на которой размещается соответствующая экранная форма, и одной транзакции, которая включает в себя все операции по обработке ввода данных в эту форму. Страница состоит из трех элементов:

- Filer, имеющего тип ComplexFilterUnit; этот элемент включает в себя f-сторону формы-отношения;
- AlreadySelected, имеющего тип Selector; этот элемент делает запрос к базе данных и выдает все объекты e-стороны, которые уже связанны с f-стороной;
- MultySelector, имеющего тип IndexUnit; этот элемент включает в себя e-сторону формы-отношения; т. е. здесь показываются результаты запроса к базе данных на основе выборки фильтра Filter — объекты e-стороны; на этот список «накладывается» результат выборки AlreadySelected, и либо в кнопках выбора (checkboxes) появляются галочки для уже выбранных объектов, либо выбранные объекты отображаются в окне справа, а не выбранные — в окне слева, как показано на рис. 3, *б*.

После того как пользователь добавил/удалил какие-то связи в MultySelector, управление передается операции MustBeDeleted, которая удаляет связи из базы данных для удаленных пользователем объектов. При успешном выполнении этой операции управление передается операции Connect, которая добавляет в базу данных связи для тех объектов, которые пользователь добавил. При неуспешном выполнении одной из этих операций управление снова передается форме MultySelector. В [6] представлен похожий шаблон. Но, во-первых, он создан для очень простых фильтров, во-вторых, он не вполне корректно работает с записями базы данных: при создании новых связей сначала удаляются все старые. Однако при такой перегенерации связей, которые в действительности не были удалены пользователем, может быть утеряна какая-либо информация, связанная с ними. В-третьих, шаблон из [6] не инвариантен относительно того, как представлено отношение «многие-ко-многим» — в обычном или «раскрытом» виде (через отношения «один-ко-многим»). В-четвертых, с помощью данного шаблона невозможно задать форму-отношение для n-арной связи «многие-ко-многим», а также для цепочки таких отношений. В нашем же случае лишь фильтр будет сложнее, а общий вид шаблона остается неизменным.

Заключение

Итак, используя WebML с нашими расширениями, становится возможно моделировать полнофункциональные интерфейсы для

RIA-приложений интенсивной обработки данных, имеющих «бедную» бизнес-логику и целиком определяемых схемой данных и моделью ограничений на нее. В данной работе сделан акцент на таких модельных абстракциях, которые позволяют генерировать существенно нетривиальный программный код по несложным моделям. Мы учли уроки истории развития модельно-ориентированных подходов к созданию пользовательских интерфейсов, стремясь создать практически эффективный подход и не боясь сужения целевой области его применения. На наш взгляд, если история для чего-то и нужна, то затем, чтобы все не развивалось по второму и третьему кругу.

Предложенный в данной работе подход может быть интегрирован с гипертекстовыми моделями других модельно-ориентированных методов к разработке Web-приложений (обзор таких методов можно найти в работе [13]), а также с подходом RUX в качестве более высокоуровневой надстройки над ним для решения более узкого класса задач.

Наше расширение языка WebML мы постарались представить максимально формально. Надстройки над WebML, подобно нашей, должны появиться во множестве, если этот язык из лидера университетских разработок превратится в широко используемый практический подход. С этой целью WebML должен быть существенно строже формализован и отделен от реализующих его программных инструментов.

Список литературы

[1] *Буч Г., Якобсон А., Рамбо Дж.* UML. 2-е. изд. / Пер. с англ. СПб.: Питер, 2006. 735 с.

[2] *Иванов А. Н.* Графический язык описания ограничений на диаграммы классов UML // Программирование. 2004. № 4. С. 204–208.

[3] *Иванов А. Н., Стригун С. А.* Технологическое решение REAL-IT: Моделирование и генерация пользовательского интерфейса // Системное программирование. СПб., 2004. С. 124–147.

[4] *Иванов А. Н.* Технологическое решение REAL-IT: создание информационных систем на основе визуального моделирования // Системное программирование. СПб., 2004. С. 89–100.

[5] *Acerbis R., Bongio A, Brambilla M. et al.* WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications. LNCS. Vol. 4607. Springer Berlin. Heidelberg, 2007. P. 501–505.

[6] Advanced Features Tutorial. WebRatio 4.3. 2005. 60 p.

[7] *Bozzon A., Comai S., Fraternali P. et al.* Conceptual Modeling and Code Generation for Rich Internet Applications. ACM International Conference Proceeding Series. Vol. 263. Proceedings of the 6th International Conference on Web Engineering. 2007. P. 353–360.

[8] *Duhl J.* White paper: Rich Internet Applications. Technical Report, IDC, November 2003.

[9] *Ivanov A., Koznov D.* REAL-IT: Model-Based User Interface Development Environment. Proceedings of IEEE/NASA ISoLA 2005 Workshop on Leveraging Applications of Formal Methods, Verification, and Validation. Loyola College Graduate Center Columbia, Maryland, USA, 23–24 September. 2005. P. 31–41.

[10] *Linaje M., Preciado J. C., Shnchez-Figueroa F.* A Method for Model Based Design of Rich Internet Application Interactive User Interfaces. ICWE 2007. LNCS 4607. P. 226–241.

[11] *Paulson L. D.* Building Rich Web Applications with Ajax // Computer. 38(10). 2005. P. 14–17.

[12] *Preciado J. C., Linaje M., Comai S. et al.* Designing Rich Internet Applications with Web Engineering Methodologies Web Site Evolution, 2007. WSE. 2007. 9th IEEE International Workshop on Web Site Evolution. 5–6 Oct. 2007. P. 23–30.

[13] *Preciado J. C., Trigueros M. L., Sanchez F. et al.* Necessity of Methodologies to Model Rich Internet Applications. WSE. 2005. P. 7–13.

[14] *Rossi G., Pastor O., Schwabe D. et al.* Web Engineering: Modelling and Implementing Web Applications. Springer, 2007. 464 p.

[15] *Toffetti C. G.* Modeling data-intensive Rich Internet Applications with Server Push Support // Int. Workshop Model-Driven Web Engineering in Conjunction with ICWE. Como (Italy), 2007.