Технологическое решение REAL-IT: автоматизированная разработка пользовательского интерфейса информационных систем

А.Н.Иванов iw@tercom.ru
С.С.Стригун asa@tercom.ru

Санкт-Петербургский государственный университет
198504, Университетский пр., 28
Санкт-Петербург, Россия

Аннотация

В работе рассматривается подход к созданию оконного пользовательского интерфейса, основанный на применении модельно-ориентированного метода разработки. Создаваемый интерфейс состоит из набора типизированных элементов (экранных форм). Каждая экранная форма является параметризацией типового шаблона фрагментом модели данных (этот фрагмент указывает разработчик при моделировании интерфейса). Использование модели данных и шаблонов экранных форм делает возможным генерацию полностью работающего кода, а широкие возможности по настройке шаблонов позволяют выбирать представление данных исходя из требований пользователя.

Введение

Существенную часть современной информационной системы составляет пользовательский интерфейс. В настоящее время существуют различные подходы к созданию *графического пользовательского интерфейса (GUI)* [16], например, широко используются среды разработки, которые позволяют визуально конструировать интерфейс из уже готовых графических элемен-

тов. Примерами таких сред являются Visual Basic, Delphi, Power Builder и т.д.

Одним из подходов является *модельно-ориентированная разработка пользовательского интерфейса (MB UID — Model Based User Interface Development)* [12], характеризирующаяся использованием высокоуровневых декларативных моделей для проектирования интерфейса, а также автоматическим связыванием моделей и программного кода системы (например путем генерации кода по моделям).

На базе объектно-ориентированного CASE-пакета REAL и одноименной методологии [6] на кафедре системного программирования СПбГУ было создано модельно-ориентированное технологическое решение REAL-IT, предназначенное для разработки и сопровождения промышленных информационных систем. Основным принципом данного решения является построение визуальных моделей предметной области и генерация кода по этим моделям. Обзор REAL-IT можно найти в [4].

В REAL-IT основной моделью разрабатываемой системы является модель данных. Эта модель строится с помощью расширенной модели классов UML [5], поддерживаемой CASE-пакетом REAL. Модель данных описывает статическую структуру предметной области и используется для создания базы данных.

В данной статье описывается подход к созданию пользовательского интерфейса, реализованный в REAL-IT. Этот подход характеризуется следующими чертами:

1. Пользовательский интерфейс разрабатывается в рамках модельно-ориентированного подхода.

2. Модель интерфейса строится на основе модели данных.

3. Пользовательский интерфейс строится из набора элементов (экранных форм) различных типов. Каждое окно приложения формируется из одной или нескольких экранных форм.

4. По модели интерфейса генерируется работающий код, а не только прототип экранной формы.

5. Используется инкрементальный подход к созданию интерфейса. Экранные формы создаются разработчиком последовательно на основе типовых шаблонов.

6. Поддерживается итеративный процесс разработки. Изменения, вносимые в модель данных, автоматически отражаются в модели интерфейса, не разрушая ее. Повторная генерация кода по модели интерфейса сохраняет модификации, вносимые разработчиком “вручную”.

Генерация работающего кода и поддержка итеративного процесса разработки позволяют эффективно использовать предложенный подход при разработке промышленных информационных систем.

1 Обзор средств модельно-ориентированной разработки пользовательского интерфейса ИС

Исходной информацией для создания интерфейса информационной системы являются, с одной стороны, алгоритмы выполнения бизнес-процессов (динамика предметной области), а с другой — структура данных (статика предметной области).

Задача создания модели интерфейса, одинаково полно учитывающей оба этих аспекта (статический и динамический), а также поддерживающей связь между ними, т.е. модели, по которой можно было бы осуществить генерацию работоспособного интерфейса, не требующего дальнейшей доработки и модификации, пока не имеет удовлетворительного решения. В существующих на сегодняшний день подходах один из аспектов является главным — именно на его основе строится интерфейс, а другой — учитывается крайне слабо либо не учитывается совсем.

К решениям, основывающимся на динамическом аспекте, можно отнести системы TEALLACH [9], MASTERMIND [10], а также подходы, изложенные в [7, 19]. К решениям, основывающимся на модели данных, относятся системы JANUS [8], GENIUS [15], Mecano [17]. К этому же направлению можно отнести генераторы форм, встроенные в ряд сред разработки приложений (например MS Access, Delphi) и CASE-пакеты (например ERwin).

Решения первой группы позволяют описать схему бизнес-логики системы, не предоставляя средств для детальной спецификации алгоритмов. Такой подход позволяет генерировать

только каркас кода системы. Кроме того, в этих подходах требуется формально описывать интерфейс системы, что, как правило, занимает существенно больше времени, чем создание аналогичного интерфейса в редакторах экранных форм широко распространенных средств разработки ИС (например в Visual Basic, Delphi, Cylix и т.д.).

В решениях второй группы основной моделью, используемой для создания системы, является статическая модель предметной области (схема базы данных в MS Access и ERwin, диаграммы классов в JANUS). При этом данные средства подразумевают предопределенную логику работы интерфейса. За счет этого генерируется не только внешнее представление пользовательского интерфейса, но и программный код, обеспечивающий его работоспособность. Однако интерфейс, создаваемый в рамках этих подходов, однозначно соответствует структуре базы данных, что зачастую оказывается недостаточно удобным для выполнения задач, стоящих перед пользователем.

Генераторы пользовательского интерфейса технологического решения REAL-IT можно отнести ко второй группе подходов, но при этом разработчикам ИС предоставляются средства настройки, позволяющие варьировать структуру создаваемого интерфейса.

2 Моделирование интерфейса в REAL-IT

Выполняя различные задачи, пользователи информационной системы смотрят на данные с различных точек зрения. Например, с точки зрения бухгалтера, важнейшим свойством сотрудника является его зарплата, а для менеджера в первую очередь важны проекты, в которых этот сотрудник участвует. Это означает, что, создавая интерфейс, разработчик должен спроецировать модель данных на точки зрения тех пользователей, для которых она предназначена. Одной модели данных могут соответствовать различные интерфейсы в зависимости от взгляда на систему.

Поэтому помимо модели данных нужна еще информация, определяющая представление этих данных — мы будем называть ее моделью интерфейса. Моделью потому, что речь идет о более крупных и абстрактных единицах, чем отдельные элемен-

ты управления экранных форм (в противном случае нет предмета для генерации). При построении модели интерфейса из модели данных может быть исключена часть элементов (классов, ассоциаций, атрибутов), какие-то многозвенные связи могут быть представлены в виде простых однозвенных и т.д. Таким образом, одной модели данных может соответствовать множество различных моделей интерфейса.

Естественной единицей оконного интерфейса пользователя является окно. Именно в терминах последовательности окон пользователь представляет себе обычно работу с системой. Мы будем выделять в интерфейсе два уровня:

- макроуровень, описывающий набор окон и связей между ними. Связи обозначают возможность вызова одного окна из другого;
- микроуровень, описывающий набор элементов управления каждого окна и свойства этих элементов управления.

Соответственно, мы будем говорить о макромодели интерфейса в целом и о микромоделях отдельных окон.

2.1 Порядок использования модели интерфейса

Возможны следующие стратегии работы с моделью интерфейса:

1. Модель однозначно выводится из модели данных — тогда нет необходимости в ее сохранении как отдельного артефакта[1]. Такой подход реализован, например, в системе JANUS.
2. Модель создается с помощью специализированных средств (например графических редакторов в составе CASE-пакетов).
3. Разработчик создает микромодели для каждого окна, причем окно генерируется сразу после его моделирования. Информация о микромодели либо не сохраняется (как в MS Access), либо сохраняется (как в ERwin). Макромодель отсутствует.

[1]Под артефактами [1] понимаются все продукты проекта, порождаемые или используемые в нем при работе над окончательным продуктом.

4. То же, что и в предыдущем пункте, но при моделировании окна указываются его связи с другими окнами. Информация об этих связях сохраняется, образуя макромодель интерфейса.

Вначале в REAL-IT была реализована вторая стратегия, однако апробация решения показала, что такой подход неудобен для использования — разработчики предпочитают увидеть сгенерированное окно сразу, как только оно было спроектировано. В итоге, в REAL-IT была реализована четвертая стратегия. По сути, это переход от каскадного метода разработки (построили общую макромодель, построили микромодели для отдельных окон) к итеративному (создали, т.е. промоделировали и тут же сгенерировали одно окно, затем другое и т.д.). При этом информация, хранящаяся в модели, не изменилась, изменился только способ наполнения модели. На наш взгляд, имеет смысл сочетать вторую стратегию с четвертой — на начальном этапе создания интерфейса предоставить разработчику возможность создать прототип модели всего интерфейса, после чего уточнять и развивать его, моделируя отдельные окна.

2.2 Диаграммы классов UML

Для хранения информации о модели интерфейса в REAL-IT используются диаграммы классов UML, которые сохраняются в репозитории проекта в CASE-пакете REAL. При этом каждому окну ставится в соответствие отдельный класс, а связи между окнами переходят в ассоциации между классами. Набор классов и ассоциаций описывает макромодель интерфейса. Микромодель окна сохраняется в свойствах класса.

Иногда в окно требуется вставить целиком содержимое другого окна — например, в JANUS карточка класса-предка вставляется в карточку наследника, а в REAL-IT в окно-карточку могут быть вставлены встроенные списки. При этом вложенное окно перестает быть окном — у него нет рамки, заголовка, некоторых кнопок, отвечающих за его функциональность как отдельного окна (например кнопки “Закрыть”). Тем не менее, у него есть своя микромодель, и его удобно реализовывать как отдельную компоненту, которая генерируется самостоятельно (отдельно от объемлющего окна) и встраивается в объемлющее окно как еди-

ное целое. Мы будем считать такие компоненты также элементами макромодели. При этом отношение вложения для компонент в модели классов отображается как отношение агрегирования. Для удобства изложения будем называть все компоненты интерфейса *экранными формами*. Каждое окно состоит либо из одной экранной формы, либо из нескольких связанных экранных форм. В последнем случае одна из форм считается главной, т.е. она должна агрегировать другие формы.

2.3 Шаблоны экранных форм

Как уже было сказано, модель данных неоднозначно определяет пользовательский интерфейс системы. Экранные формы, визуализирующие фрагмент модели данных, могут использовать какую-то определенную часть информации из модели и поразному представлять ее. Однако при этом интерфейс не должен быть хаотичным, иначе его освоение потребует больших усилий со стороны пользователя, т.е. он должен состоять из взаимосвязанных элементов, причем количество типов этих элементов и видов связей между ними должно быть достаточным, но не слишком большим.

На макроуровне следование этому принципу приводит нас к тому, что каждая экранная форма создается на основе шаблона, параметризуемого, с одной стороны, фрагментом модели данных (набором используемых классов и их связей), а с другой стороны — дополнительными настройками разработчика. Кроме того, шаблон определяет возможные связи данной формы с другими формами.

Набор шаблонов определяет метамодель макроуровня интерфейса, или *макрометамодель*. Именно она описывает структурное разнообразие интерфейса: из экранных форм каких типов он состоит, какими информационными потоками эти формы могут быть связаны между собой, как могут конструироваться окна из отдельных форм. В свою очередь, для каждого типа форм генераторы задают метамодель микроуровня форм этого типа, или *микрометамодель*. Именно она определяет разнообразие визуализации отдельных элементов данных внутри экранной формы.

Макрометамодель интерфейса REAL-IT в настоящий момент содержит три типа форм:

1. список, отображающий множество объектов[2];
2. карточка, позволяющая посмотреть и отредактировать свойства одного объекта;
3. форма-отношение, реализующая связь "многие-ко-многим" между классами модели данных.

Формы этих типов могут быть связаны следующим образом:

1. Список с карточкой для просмотра и редактирования элемента списка. Позволяет пользователю просматривать/редактировать информацию об отдельном объекте, а также создавать новые объекты. В настоящий момент связи этого вида генераторами REAL-IT реализуются только как возможность вызова из окна-списка окна-карточки.
2. Карточка со списком или отношением для связанных объектов. Позволяет из карточки объекта переходить к просмотру/редактированию его связей с другими объектами. Связи этого вида можно инкапсулировать в составное окно, т.е. можно сгенерировать окно, состоящее из карточки и связанных с нею списков.
3. Отдельные элементы одного окна с другим окном — например возможность вызова списка или карточки из фильтра (об этом будет сказано ниже).

Следует отметить, что перечисленный набор типов окон не охватывает все разнообразие экранных форм, которые могут быть полезны при создании пользовательского интерфейса информационной системы. Например, в него не вошли такие типы экранных форм, как статистические таблицы, деревья, позволяющие отображать иерархический набор объектов. Не все виды связей между экранными формами могут быть инкапсулированы в одно окно (например, нельзя создать окно со спис-

[2]Поскольку в REAL-IT используется объектно-ориентированный подход к моделированию данных, то основной единицей данных является объект. При реализации ИС с использованием реляционной СУБД каждому объекту соответствует запись в одной из таблиц базы данных.

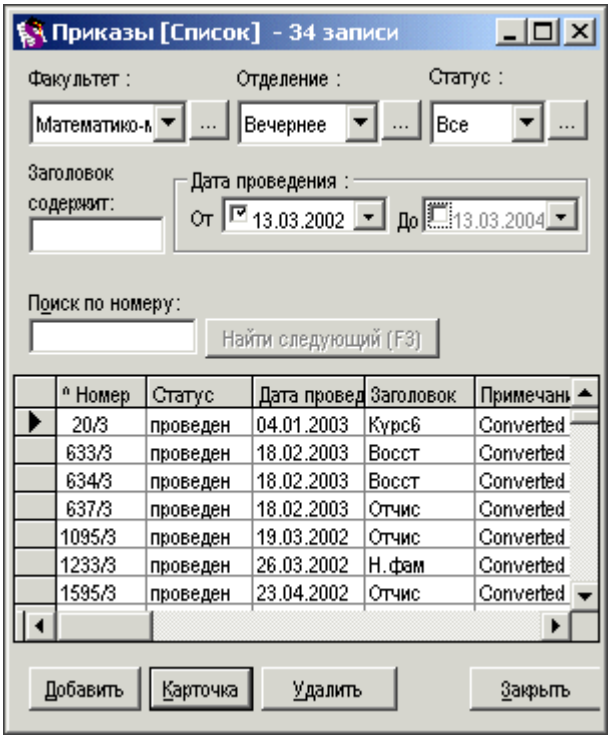

Рис. 1: Список приказов

ком и встроенной в этот список карточкой). Микрометамодели отдельных типов окон также могут быть расширены (например, использованием переключателей (элементы управления типа radiobutton) для выбора одного из значений перечислимого типа).

3 Разработка отдельных типов экранных форм

Опишем более подробно свойства каждого из типов экранных форм, реализованных в REAL-IT.

3.1 Список

Список предоставляет пользователю возможность работать с множеством объектов одного класса (мы будем называть его *основным классом списка*), а также выбирать один или несколько объектов для выполнения над ними каких-либо действий. Каждый элемент списка соответствует одному объекту основного класса, однако при его отображении может использоваться информация о других объектах модели данных. На рисунке 1 приведен пример формы-списка.

Мы считаем, что в общем случае список может предоставлять пользователю следующие возможности:

1. Просмотр элементов списка в виде таблицы.
2. Фильтрацию, т.е. выбор критериев отбора элементов для показа.
3. Подсчет количества элементов.
4. Сортировку элементов.
5. Группировку элементов списка, т.е. разбиение их на непересекающиеся группы по некоторому признаку.
6. Поиск элемента.
7. Нумерацию элементов. При использовании группировок возможна как сквозная нумерация, так и нумерация в пределах одной группы.
8. Выбор набора отображаемых свойств из числа предусмотренных разработчиком, т.е. возможность скрыть ненужные столбцы.
9. Печать и экспорт данных в другие приложения.
10. Переход в карточку для просмотра или редактирования свойств отдельного объекта, а также для добавления в список нового объекта.
11. Редактирование свойств объектов непосредственно из списка.
12. Выбор одного или нескольких объектов.

13. Удаление выбранных объектов.

14. Выполнение над выбранными объектами дополнительных операций, определенных разработчиком (например для студентов — печать различных справок или пересчет академической задолженности).

Некоторые из перечисленных выше возможностей не требуют моделирования, так как их реализацию целиком можно возложить на библиотеки динамической поддержки, а другие являются предметом моделирования и, таким образом, определяются микромоделью. При этом свою специфику накладывает целевая платформа, поскольку именно ею в значительной степени определяется, какие возможности будут предоставлены пользователю и какие из них можно реализовать в динамической поддержке. Например, элемент управления, отображающий список, может поддерживать группировку строчек по значению в произвольном столбце по выбору пользователя (примером такого элемента управления может служить Microsoft Pivot Table), либо поддерживать ее при наличии явного указания этого столбца в коде (как в Microsoft Hierarchical Grid), либо не поддерживать вовсе (как Microsoft Data Grid).

Генерируемые в REAL-IT формы-списки в настоящий момент поддерживают все перечисленные выше возможности, кроме п. 5 (группировка), 7 (нумерация) и 11 (редактирование). Каждая форма содержит область фильтров, область локаторов (полей ввода для поиска отдельного элемента), область списка и область кнопок. Взаимное расположение этих областей определяется шаблоном. В REAL-IT существует два шаблона — для списков, отображаемых в отдельном окне, и для списков, встраиваемых в форму-карточку. При этом область списка присутствует на форме обязательно, а все остальные могут отсутствовать, если при моделировании разработчик не выбрал для них содержания.

Предметом моделирования для списка является определение набора столбцов, фильтров, локаторов и кнопок. Про выбор столбцов и фильтров ниже будет рассказано отдельно. Локатор можно создать только один, выбрав для него один из столбцов.

3.1.1 Определение набора столбцов

Набор столбцов фиксирует свойства объектов основного класса, отображаемые в списке. При этом в числе столбцов могут присутствовать как атрибуты основного класса, так и атрибуты других классов, связанных с основными ассоциациями, а также вычислимые поля (выражения, вычисляемые на основе значений других свойств класса). Для определения набора столбцов используется такой элемент метамодели REAL, как представление (View).

Каждое представление — это вырезка из модели классов, оно содержит набор классов, их атрибутов и ассоциаций между этими классами. У каждого класса или атрибута внутри представления может быть собственное имя (синоним). Один и тот же класс может входить в представление дважды, в таком случае его синонимы должны различаться. Наряду с атрибутами классов в представление могут входить также вычислимые поля.

При генерации списка разработчик может выбрать одно из представлений, содержащих основной класс. Поля выбранного представления образуют набор столбцов списка, таким образом выбранное представление определяет визуализацию объектов в списке. Если требуется отобразить в точности множество атрибутов основного класса, то в качестве представления можно выбрать сам класс.

3.1.2 Моделирование фильтров

Как было сказано выше, список может содержать набор фильтров, используемых для ограничения набора отображаемых объектов. Фильтры бывают двух типов — по атрибуту и по ссылке.

Фильтры по атрибуту позволяют отобрать только те объекты, значение некоторого атрибута которых удовлетворяет указанному условию. Условие определяется в виде пары <операция, значение>, при этом операцию фиксирует разработчик, а значение может ввести пользователь. Набор допустимых операций определяется генератором в зависимости от типа атрибута. Для всех обрабатываемых генератором типов данных допустимо равенство, для упорядоченных типов — операции “больше” и “меньше”, для символьных типов — поиск подстроки. Для

упорядоченных типов, кроме того, существует операция “от..до”, предусматривающая задание допустимого диапазона значений. Фильтр по такой операции содержит не одно, как остальные фильтры, а два поля ввода для указания границ диапазона.

Фильтры по ссылке позволяют отобрать объекты, связанные некоторой ассоциацией с объектом другого класса (будем называть как этот объект, так и его класс, фильтрующими). Объекты, из числа которых может быть выбран фильтрующий объект, мы будем называть допустимыми. Фильтр по ссылке представляется на форме выпадающим списком (элементом управления типа combobox), из которого пользователь может выбрать фильтрующий объект.

Фильтры по ссылке являются наиболее часто используемым видом фильтров — например, в главном списке студентов ИС “Студент” имеется целый ряд таких фильтров, позволяющих отобрать студентов, обучающихся на определенном факультете, курсе, в группе, на кафедре и т.д. При задании фильтра по ссылке разработчик может указать ряд дополнительных настроек:

- Представление. Определяет способ визуализации фильтрующего объекта.
- Способ выбора фильтрующего объекта — из элемента управления “выпадающий список” или из отдельной формы-списка. В последнем случае требуется указать используемую форму (фильтрующий класс должен быть основным классом этой формы).
- Наличие или отсутствие возможности создания новых объектов фильтрующего класса прямо из выпадающего списка.
- Наличие или отсутствие возможности перехода в форму-карточку фильтрующего объекта. Для реализации данной возможности в составе фильтра генерируется кнопка, по нажатию которой пользователь может открыть карточку.

Если фильтрующий класс является абстрактным, то в составе фильтра будет сгенерирован выпадающий список для выбора конкретного класса, позволяющий ограничить список допустимых объектов экземплярами одного из классов-наследников.

Наличие у класса слабого агрегирования [5] рассматривается как “псевдоассоциация”, по которой объекты этого класса могут быть связаны с объектами одного из классов-агрегатов. Соответственно, по этой псевдоассоциации также может быть наложен фильтр по ссылке, состоящий из двух выпадающих списков — для выбора фильтрующего класса из числа классов-агрегатов и для выбора фильтрующих объектов.

Фильтры по ссылке предоставляют возможность множественного выбора, т.е. возможность указать не один, а несколько объектов. При этом условие фильтра считается выполненным для объекта фильтруемого класса, если он связан хотя бы с одним из фильтрующих объектов. Список допустимых объектов всегда дополняется служебным значением “Нет”, означающим отсутствие связи по выбранной ассоциации с какими-либо объектами.

Фильтры могут накладываться как на основной класс, так и на любой из классов, входящих в представление списка (а фильтры по атрибуту — и на вычислимые поля представления). Кроме того, фильтры могут накладываться на другие фильтры — поскольку в каждом фильтре по ссылке определяется список допустимых объектов, то набор объектов в этом списке также можно ограничить с помощью других фильтров (как по атрибуту, так и по ссылке).

Использование ограничений на данные. Ограничения на ассоциации между классами, описанные в модели данных, используются при генерации списков следующим образом: код формы запрещает устанавливать такой набор фильтров на класс, которому не может удовлетворять ни один объект этого класса вследствие наличия ограничений.

Например, пусть имеется список студентов с фильтрами по факультету и специальности, и при этом факультет и специальность связаны ассоциацией “многие-ко-многим”, определяющей, каким специальностям учат на каждом факультете. Пусть в модели данных задано ограничение, согласно которому студент может учиться только на одной из тех специальностей, которым учат на его факультете. Тогда список возможных значений в фильтре по специальности будет зависеть от списка выбранных значений в фильтре по факультету таким образом, что в него бу-

дут включены все те, и только те специальности, которым учат хотя бы на одном из выбранных факультетов.

Реализация этого запрета производится следующим образом: список допустимых объектов в фильтре по ограниченной ассоциации строится на основе списка выбранных объектов в фильтрах, соответствующих контексту этой ассоциации [2] таким образом, чтобы существование хотя бы одного объекта в отфильтрованном списке не противоречило ограничению.

Таким образом, если у нас имеется несколько фильтров на один класс, то множество допустимых объектов в некоторых из них может меняться в зависимости от выбранных объектов в других фильтрах, т.е. имеет место зависимость между фильтрами. При этом циклические зависимости не допускаются — в случае их обнаружения генератор сообщает об ошибке.

Скрытые фильтры. Фильтры могут быть видимыми для пользователя системы и невидимыми (скрытыми). Скрытые фильтры позволяют ограничивать набор отображаемых объектов программно из других элементов приложения. Они используются, главным образом, в списках, встроенных в формы-карточки. При этом список используется для отображения множества объектов, связанных выбранной разработчиком ассоциацией с объектом, представленным в карточке. В этом случае в карточке генерируется код, устанавливающий соответствующие фильтры во всех встроенных в нее списках.

3.2 Карточка

Карточка предназначена для просмотра и редактирования детальной информации об отдельном объекте (экземпляре класса модели данных). К такой информации относятся значения атрибутов объекта, а также информация о его связях с другими объектами. Кроме того, из карточки можно выполнять действия над объектом, которые указаны в модели данных как методы его класса. На рисунке 2 приведен пример формы-карточки.

Не всегда требуется показывать пользователю все свойства объекта (атрибуты, ассоциации, методы). Кроме того, отдельное свойство может быть представлено по-разному. При большом количестве свойств может быть полезным разбиение их на

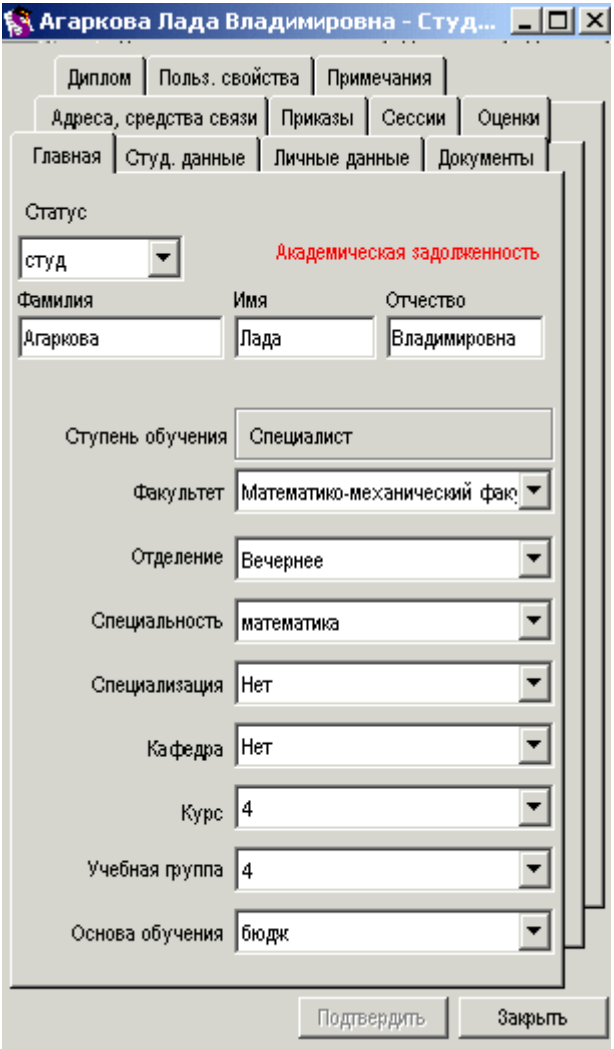

Рис. 2: Карточка студента

отдельные группы, отображающиеся совместно. Исходя из этих соображений, карточка в REAL-IT содержит закладки, на которых располагаются элементы управления, позволяющие пользователю работать с отдельными свойствами объекта. Каждое свойство представлено отдельным элементом управления или группой элементов, по следующему принципу:

- Способ отображения атрибута определяется исходя из его типа. Например, поле ввода для строкового атрибута, переключатель для логического, поле выбора для перечислимого и т.д. Если для данного типа поля генератором предусмотрены альтернативные варианты визуализации (например многострочное поле ввода или элемент управления RTF-текст для текстовых полей), разработчику предоставляется выбор одного из них. Кроме того, разработчик может использовать нестандартный тип элемента управления для отображения атрибута, указав его в соответствующем пользовательском свойстве REAL для данного атрибута.

- Объекты, связанные с данным по ассоциациям с множественностью "многие-к-одному" ("многие" со стороны данного объекта) или "один-к-одному", отображаются с помощью полей выбора. Каждое поле выбора соответствует роли объекта в одной ассоциации. Генерация этих полей аналогична генерации фильтров по ссылке, рассматривавшихся при описании форм-списков, и разработчик может указать все те же дополнительные свойства отображения такого поля.

- Список объектов, связанных с данным по ассоциации с множественностью "один-ко-многим", представляется с помощью отдельной экранной формы — списка или отношения, в которой устанавливаются соответствующие фильтры. При этом в карточке располагается либо сама форма, либо кнопка, по которой данная форма открывается в отдельном окне. Первый вариант удобен при отображении множественных свойств объекта, т.е. объектов, связанных с данным отношением агрегирования (например список телефонов конкретного человека). Второй способ обычно используется, если связанных объектов много.

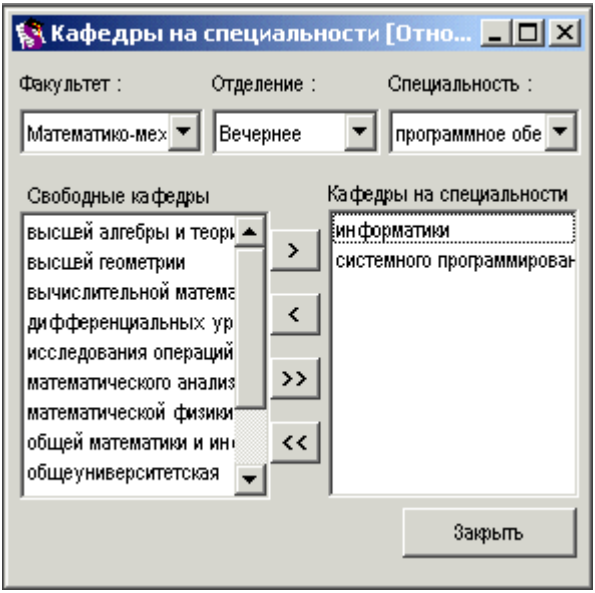

Рис. 3: Отношение “Кафедры на специальности”

- Для вызова метода объекта на форму добавляется соответствующая кнопка.

В соответствии с вышесказанным, модель карточки для некоторого класса объектов представляет собой лес, состоящий из деревьев высоты 2. Корнями этих деревьев являются закладки, листьями — отдельные свойства объектов класса. Все деревья, а также листья одного дерева образуют линейную последовательность, определяющую порядок следования закладок и порядок следования элементов управления на каждой закладке. В процессе моделирования разработчик может добавлять и удалять закладки, распределять свойства по закладкам или удалять свойства из карточки. Каждый узел имеет собственный за-

головок, который разработчик может изменить (по умолчанию в качестве заголовков для свойств используются их логические имена в модели REAL, а заголовки закладок устанавливаются генератором).

3.3 Форма — отношение

При моделировании данных могут встречаться различные виды ассоциаций, как бинарные ("один-ко-многим", "один-к-одному", "многие-ко-многим"), так и тернарные (связывающие более двух классов[3]). При переходе от модели сущность-связь к реляционной модели ассоциации "многие-ко-многим" и тернарные ассоциации обычно реализуются отдельными таблицами.

Модель классов UML позволяет моделировать ассоциации "многие-ко-многим", но не содержит специальных средств для моделирования тернарных ассоциаций. При создании модели данных в технологическом решении REAL-IT ассоциации "многие-ко-многим" и тернарные ассоциации предлагается моделировать отдельными классами (мы будем называть их классами-отношениями или просто отношениями). Для выделения классов-отношений метамодель расширена свойством класса "Является отношением"[4], которое может принимать значение "Истина" или "Ложь". Если значения этого свойства не установлены разработчиком, то класс считается отношением, если у него нет ни одного атрибута, он не наследуется от другого класса, и у него есть две или более ассоциаций "многие-к-одному".

Поскольку отношения реализованы классами, то для них можно создать такой же интерфейс, как и для обычных классов, т.е. набор списков и карточек. Однако подобный интерфейс не всегда будет удобен для пользователя ИС. Поскольку эти классы являются вспомогательными и соответствуют связям, а не объектам предметной области, то создание для них списков и

[3]Формально корректней было бы говорить не о классах, связанных ассоциацией, а о ролях ассоциации, соответствующих классам. Один класс может иметь несколько различных ролей в одной ассоциации, например, класс "Сотрудник" может играть обе роли в ассоциации "начальник-подчиненный". Однако чтобы не загромождать текст и облегчить его понимание, мы будем говорить о связанных классах, имея в виду именно соответствующие роли. Генераторы REAL-IT корректно обрабатывают множественное участие класса в ассоциации.

[4]Для расширения метамодели используются пользовательские свойства CASE-пакета REAL.

карточек может вызвать у пользователя непонимание.

Можно выделить два способа представления отношений "многие-ко-многим", часто встречающихся в интерфейсах информационных систем:

1. **Табличный.** Отношение представляется в виде таблицы: строки таблицы соответствуют объектам одного класса, столбцы — объектам второго класса, в ячейках отражается наличие или отсутствие ассоциации между соответствующей парой объектов.

2. **Списочный.** Фиксируется объект одного из классов, а все объекты второго класса отображаются в двух списках — списке объектов, связанных ассоциацией с зафиксированным, и списке объектов, которые могут быть с ним связаны. Перемещение объекта из одного списка в другой означает создание или удаление экземпляра ассоциации.

Достоинством первого способа является его наглядность. Кроме того, если отношение содержит какие-либо собственные свойства, их можно отображать в ячейках таблицы. Недостатком этого способа является то, что площадь отображаемой таблицы растет квадратично от количества объектов и при большом количестве объектов просматривать таблицу становится затруднительно.

Достоинством второго способа является то, что с отношением удобно работать даже при большом количестве объектов. Кроме того, этот способ можно обобщить для случая тернарных ассоциаций — если ассоциация связывает объекты N классов, то можно зафиксировать объекты $N-1$ класса, а объекты оставшегося класса отобразить в списках. Недостатком этого способа является его асимметричность, поскольку один из классов, входящих в отношение, является выделенным и отображается в виде списков, а не фиксируется, как остальные. Это означает, что в большинстве случаев приходится создавать несколько форм для одного отношения — по одной на каждый из входящих в отношение классов.

В REAL-IT в настоящий момент реализован второй способ представления отношений. При моделировании формы разработчик указывает, какой из классов отображается в списках. Как

и при моделировании списковых форм, можно указать представление для списков, а также набор дополнительных фильтров для всех классов, входящих в отношение. На рисунке 3 приведен пример такой формы.

Заключение

Описанный в работе подход реализован в генераторах пользовательского интерфейса, входящих в состав технологического решения REAL-IT. Эти генераторы успешно использовались при создании ряда промышленных информационных систем. В таблице приведена статистика использования генераторов при создании ИС "СТУДЕНТ", предназначенной для автоматизации управления учебным процессом вуза.

Тип формы	Полностью сгенерировано	С "ручными" изменениями[5]	Сопровождается "вручную"[6]	Всего
Список	69	16	6	91
Карточка	37	21	16	74
Отношение	17	1	4	22
Другое[7]	0	0	6	6
Всего	123	38	32	193

Видно, что генераторы в той или иной мере использовались при создании более чем 96% экранных форм, при этом более 60% экранных форм не требовали "ручной" доработки после генерации.

Дальнейшее развитие предлагаемого решения, нацеленное на увеличение процента сгенерированных форм, подразумевает расширение используемой метамодели интерфейса. Такое расширение может происходить в следующих направлениях:

- использование информации о динамике поведения системы;
- добавление новых типов экранных форм (например статистических таблиц или списков, имеющих иерархическую структуру);
- расширение возможностей создания составных форм (например составления окна из списка и связанной с ним карточки);

- расширение набора поддерживаемых типов данных.

Другим направлением развития предложенного подхода может служить его адаптация для моделирования интерфейса WEB-приложений. Такая адаптация предполагает определение набора требований и ограничений, специфических для WEB-приложений, анализ метамодели интерфейса REAL-IT с точки зрения этих требований, а также сравнение данного подхода с другими средствами и языками моделирования WEB-интерфейсов, в первую очередь тех, которые основаны на использовании модели данных, в частности, с WebML [11], HDM [13], RMM [14] и др.

Список литературы

[1] Буч Г., Рамбо Д., Якобсон А. Унифицированный процесс разработки программного обеспечения. — СПб.: Питер, 2002. — 496 с.

[2] Иванов А.Н. Графический язык описания ограничений на диаграммы классов UML (готовится к выходу) // Программирование. — 2004. — № 3.

[3] Иванов А.Н. Механизмы поддержки циклической разработки ИС в рамках модельно-ориентированного подхода // Наст. сборник. — С. 101-123.

[4] Иванов А.Н. Технологическое решение REAL-IT: создание информационных систем на основе визуального моделирования // Наст. сборник. — С. 89-100.

[5] Романовский К.Ю., Кузнецов С.В., Кознов Д.В. Объектно-ориентированный подход и диаграммы классов UML // Объектно-ориентированное визуальное моделирование. — СПб.: 1999. — С. 21-56.

[6] Терехов А.Н. и др. Real: методология и CASE-средство разработки информационных систем и программного обеспечения систем реального времени // Программирование. — 1999. — № 5. — С. 44-51.

[7] Baeg J., Fukazawa Y. A Dialog-Oriented User Interface Generation Mechanism // Proc. of the 3rd Asia-Pacific Software Engineering Conference. — 1996.

[8] Balzert H. From OOA to GUIs — the JANUS System // Journal of Object-Oriented Programming — 1996. — Vol. 8. № 9. — P. 43-47.

[9] Barclay P. J. e.a. Teallach — a flexible user-interface development environment for object database applications // Journal of Visual Languages & Computing — 2003. — Vol. 14. I 1. — P. 47-77.

[10] Browne T.P. e.a. Using declarative descriptions to model user interfaces with MASTERMIND // Formal Methods in Human-Computer Interaction. — Springer-Verlag. — 1997. — P. 93-120.

[11] Ceri. S., Fraternali. P., Bongio A. Web Modeling Language (WebML): a modeling language for designing Web sites // Computer Networks. — 2000. — Vol. 33. № 1-6. — P. 137-157.

[12] da Silva P. User Interface Declarative Models and Development Environments: A Survey // LNCS. — 2000. — Vol. 1946. — P. 207-226.

[13] Garzotto F., Paolini P. HDM — A Model-Based Approach to Hypertext Application Design // ACM Transaction on Information Systems. — 1993. — Vol. 11. № 1. — P. 1-26.

[14] Isakowitz T., Sthor E. A., Balasubramanian P. RMM: a methodology for structured hypermedia design // Comm. of the ACM. — 1995. — Vol. 38. № 8. — P. 34-44.

[15] Janssen C., Weisbecker A., Ziegler J. Generating User Interfaces from Data Models and Dialogue Net Specifications // Proc. of the InterCHI'93, New York. — 1993. — P. 418-423.

[16] Myers B., Hudson S., Pausch R. Past, Present and Future of User Interface Software Tools // ACM Transactions on Computer-Human Interaction. — 2000. — Vol. 7. № 1. — P. 3-28.

[17] Puerta A. The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development // Proc. of the Conference on Computer-Aided Design of User Interfaces. — 1996. — P. 19-36.

[18] Rumbaugh J., Jacobson I., Booch G. The Unified Modeling Language Reference Manual. — Boston: Addison-Wesley. — 1999. — 576 p.

[19] Shirogane J., Fukazawa Y. Method of User-Customizable GUI Generation and Its Evaluation // Proc. 3rd Asia-Pacific Software Engineering Conference. — 1998.