Архитектура среды визуального моделирования QReal

А. Н. Терехов
ant@tercom.ru

Т. А. Брыксин
timofey.bryksin@gmail.com

Ю. В. Литвинов
yurii.litvinov@gmail.com

К. К. Смирнов
lich@math.spbu.ru

Г. А. Никандров
nikandroff@gmail.com

В. Ю. Иванов
seva00@yandex.ru

Е. И. Такун
zhenya.takun@gmail.com

Существует довольно много программных средств визуального моделирования, однако часто появляется необходимость создать новый графический редактор или даже целую CASE-систему для какой-либо специальной предметной области. Разработка таких систем, несмотря на имеющиеся в этой области технологии (Microsoft DSL Tools, Eclipse/GMF и т. д.), продолжает оставаться трудоемкой и сложной задачей. Существует нехватка описаний подходов и принципов реализации таких средств, имеющиеся источники немногочисленны и часто носят общий, поверхностный характер либо полезная информация «растворена» в большом количестве технических деталей реализации. Таким образом, описание опыта разработки средств визуального моделирования и обсуждение различных вариантов их реализации представляется актуальной и важной задачей. В статье описана архитектура системы QReal, являющейся MetaCASE-пакетом и включающей основные типы диаграмм UML 2.1 и некоторые другие

диаграммы. Система поддерживает многопользовательскую работу, является распределенной и многоплатформенной.

Введение

В настоящее время для разработки программного и аппаратного обеспечения довольно активно используется визуальное моделирование. Для автоматизации процесса разработки применяются CASE-средства, в которых разрабатываемая система представляется в виде модели на каком-либо визуальном языке программирования. Широкое распространение получил язык UML [15, 16], для различных предметных областей создаются специализированные языки, такие как IDEF [7], BPMN [10] и др.

Существует множество CASE-пакетов, реализующих различные подмножества или варианты языка UML и другие визуальные языки. Такие пакеты существуют для различных платформ, поддерживают различные процессы и методологии разработки, предназначаются для различных целей[1]. Средства визуального моделирования часто встраиваются в средства разработки (например, различные визуальные редакторы в составе Microsoft Visual Studio, редактор UML-диаграмм в среде NetBeans[2] и т. д.). Тем не менее зачастую возникает необходимость создавать специализированные CASE-средства, предназначенные для какой-либо узкой предметной области [3, 4, 18].

Сложность разработки CASE-пакетов довольно велика, поскольку приходится решать проблемы эффективной реализации редакторов диаграмм, организации хранения и оперативного доступа к большим объёмам данных со сложной структурой, поддержки процесса разработки программного обеспечения, включая многопользовательскую работу, версионирование и т. д. Необходимо также обеспечить «сквозной» характер всей технологии в целом: реализовать генерацию кода по визуальным моделям, поддержать процедуру внесения «ручных» изменений в сгенерированный по моделям программный код, обеспечить согласованность и корректность всех диаграмм на всех этапах разработки.

[1] Списки существующих CASE-пакетов можно найти здесь: http://www.objectsbydesign.com/tools/umltools_byCompany.html или http://en.wikipedia.org/wiki/List_of_UML_tools

[2] http://www.netbeans.org/features/uml/

Существующие технологии и средства разработки специализированных CASE-пакетов, такие как Eclipse GMF [23] и Microsoft DSL TOOLS [22], обладают рядом недостатков, делающих их широкое использование затруднительным: GMF громоздок и довольно сложен в использовании, DSL TOOLS удобен для реализации только несложных языков и «привязан» к среде Visual Studio. Поэтому варианты эффективной реализации перечисленных выше свойств CASE-систем нуждаются в обсуждении, аргументации принятия тех или иных решений. Требуются также различные архитектурные шаблоны. Все это, на наш взгляд, представляет интерес для сообщества разработчиков CASE-систем, разработчиков и пользователей отдельных графических редакторов вне зависимости от того, будут ли они пользоваться представленной в статье системой QReal или нет.

В настоящей статье представлена архитектура системы QReal— MetaCASE-системы для создания визуальных редакторов. Эта система включает в себя также реализацию редакторов для основных диаграмм UML 2.1 и некоторых других видов диаграмм.

Система QReal обладает следующими особенностями.

1. Многоплатформенность — возможность перекомпиляции исходного кода всей системы для разных целевых операционных систем. Для обеспечения кроссплатформенности использовался инструментарий для создания кроссплатформенных приложений Qt [21]. На данный момент поддерживаются платформы Windows и Linux, сборка продукта возможна на всех платформах, поддерживаемых библиотекой Qt (Windows, Mac OS, Linux/X11, embedded Linux, Windows CE и S60, в будущем — QNX и VxWorks).

2. Распределенность:

— возможность удаленного доступа к общему репозиторию;

— наличие системы разграничения прав конкурентного доступа.

3. Реализация генеративного подхода к созданию новых графических редакторов, позволяющего создавать редакторы диаграмм для специализированных визуальных языков, не прибегая к программированию.

4. Распространение всей системы под лицензией GPLv2, т. е. независимость от закрытых, проприетарных библиотек, технологий и средств разработки.

Ниже в статье приводится подробное обсуждение этих особенностей, описание выявленных трудностей и обоснование принятых технических решений.

1. Существующие решения

Системы визуального моделирования делятся на две группы — универсальные и предметно-ориентированные системы [3]. Универсальные CASE-пакеты не имеют какой-либо специализированной ориентации и поддерживают визуальные языки общего назначения (как правило, UML). Такие пакеты являются коробочными и довольно широко представлены на рынке. Наиболее известные из них — Enterprise Architect[3], IBM Rational Rose[4], Microsoft Visio[5], Visual Paradigm[6]. Как правило, универсальные CASE-пакеты имеют открытые программные интерфейсы, позволяющие расширять их функциональность. Чрезмерная универсальность таких средств делает их применение на всех этапах разработки менее выгодным по сравнению с предметно-ориентированными средствами.

Предметно-ориентированные системы предназначены для определённой предметной области, достаточно узкой, чтобы обеспечить наибольшую выгоду от применения визуального моделирования и эффективную генерацию кода. Такие инструменты могут разрабатываться компаниями для своих внутренних нужд, для решения задач именно того процесса, для поддержки которого они создаются. Для облегчения реализации предметно-ориентированных визуальных языков существуют MetaCASE-технологии, такие как Eclipse GMF, Microsoft DSL TOOLS, MetaEdit+ [13]. В современных CASE-системах, таких как GME[7], Rational Software Architect[8], имеются также средства настройки, позволяющие пользователю описывать свои визуальные языки и относительно легко реализовывать их поддержку.

1.1. Инструментарий Microsoft DSL TOOLS

Инструментарий Microsoft DSL TOOLS является частью среды разработки Visual Studio и предназначен для создания встроенных в Visual Studio визуальных редакторов. Процесс создания нового редактора состоит из описания метамодели языка на специальном визуальном языке DSL TOOLS, генерации редактора и внесении

[3] http://www.sparxsystems.com.au/
[4] http://www-01.ibm.com/software/awdtools/developer/rose/
[5] http://office.microsoft.com/ru-ru/visio/FX100487861049.aspx
[6] http://www.visual-paradigm.com/product/vpuml/
[7] http://www.isis.vanderbilt.edu/Projects/gme/
[8] http://www-01.ibm.com/software/awdtools/swarchitect/

в сгенерированный код «ручных» изменений для реализации дополнительной функциональности, такой как валидация моделей. Сохранение «ручных» изменений после перегенерации редактора осуществляется за счёт использования частичных классов (partial classes) .Net.

Язык описания метамоделей состоит из доменных классов (domain classes), которыми задаются отдельные сущности проектируемого языка, отношения ассоциации, агрегирования и наследования (следует отметить, что отношение в DSL TOOLS может иметь атрибуты и представляется, по сути, классом), а также средств задания нотации — геометрических фигур и свойств линий для отношений. Процедуры отрисовки недостающих графических примитивов можно реализовать самостоятельно на C#.

DSL TOOLS хорошо подходит для реализации несложных предметно-ориентированных языков, которые будут использоваться внутри среды Visual Studio, однако независимый графический редактор с помощью DSL TOOLS создать невозможно. Реализация сложных визуальных редакторов с возможностями, не поддерживаемыми DSL TOOLS изначально, как правило, требует больших объёмов кодирования на C#.

1.2. Технология Eclipse GMF

Технология Eclipse GMF (Graphical Modelling Framework) создана на базе среды разработки с открытыми исходными кодами Eclipse. GMF предназначена для быстрой разработки графических редакторов, в основном интегрируемых в Eclipse, и построена на двух широко используемых библиотеках — Eclipse Modeling Framework (EMF) и Graphical Editing Framework (GEF). Архитектура построенного с помощью GMF пакета основана на шаблоне проектирования Model/View/Controller (MVC), где для создания моделей используется EMF, а для создания представлений и контроллеров GEF.

Процесс создания визуальных редакторов в GMF состоит из следующих шагов.

1. Разработка доменной модели — абстрактного синтаксиса разрабатываемого предметно-ориентированного языка. Доменная модель может быть разработана с помощью графического редактора Ecore (входящего в состав GMF), импортирована из XMI[9]-

[9]XML Metadata Interchange, стандарт OMG для обмена метаданными.

документа, из набора аннотированных Java-интерфейсов или из XML-схемы.

2. Разработка графической модели — описания графической нотации разрабатываемого языка.

3. Разработка модели инструментов — описания элементов панели инструментов.

4. Разработка модели соответствия — связи между тремя предыдущими моделями.

5. Создание модели генератора — промежуточного представления разрабатываемого редактора. Эта модель автоматически генерируется по модели соответствия и дополняется «вручную».

6. Генерация целевого визуального редактора.

Данная технология позволяет создавать независимые от Eclipse графические редакторы, однако, на наш взгляд, довольно сложна в использовании и громоздка.

1.3. Rational Software Architect

Пакет IBM Rational Software Architect является развитием широко известной CASE-системы Rational Rose. Этот инструмент представляет собой полноценную среду разработки и предназначен, прежде всего, для проектирования с использованием UML 2.0, имеет развитые средства генерации кода и возвратного проектирования для языков Java и C++. К наиболее интересным особенностям этого инструмента можно отнести следующие:

- автоматический поиск архитектурных паттернов и антипаттернов в моделях;
- возможность определять свои паттерны проектирования и шаблоны генерации кода;
- возможность автоматически следить за целостностью архитектуры разрабатываемого приложения; поддержание стиля кодирования;
- поддержка трассируемости, инструменты для отслеживания связи требований с реализацией;
- поддержка многопользовательской работы, интеграция с системами контроля версий.

Rational Software Architect может использоваться и как платформа для реализации предметно-ориентированных визуальных

http://www.omg.org/technology/documents/formal/xmi.htm

языков. Сам этот CASE-пакет является расширением среды Eclipse и построен с использованием технологии GMF. Кроме того, сам Software Architect имеет средства определения профилей UML, произвольных предметно-ориентированных языков (с версии 7.5), а также возможность использования любых их комбинаций. Но независимый редактор таким способом построить нельзя.

1.4. Технология REAL

CASE-пакет QReal является развитием идей технологии REAL [2, 9], разработанной на кафедре системного программирования математико-механического факультета Санкт-Петербургского государственного университета. REAL является технологией разработки информационных систем и систем реального времени и представляет собой набор взаимосвязанных графических редакторов диаграмм UML 1.4 и SDL-92 [11] — диаграмм, объединенных в единую среду разработки. На его основе был создан набор технологических решений для различных областей, например REAL-IT для автоматической генерации по моделям информационных систем, интенсивно работающих с данными [12].

Решение о разработке нового CASE-пакета было принято по следующим соображениям.

1. Набор визуальных языков, поддерживаемых системой REAL, устарел с появлением UML 2.0.

2. Процесс добавления нового редактора в систему происходил кодированием на языке C++ (пусть даже и с учетом факта переиспользования компонент), что кажется нам весьма неоптимальным. К тому же добавление нового редактора требовало от разработчиков знания архитектуры CASE-пакета в целом.

3. Реализация графово-графической библиотеки CASE-пакета основана на устаревшей библиотеке MFC, что усложняет задачу сопровождения продукта и перенос на другие платформы.

Была предпринята попытка модификации отдельных модулей REAL (сначала графово-графической библиотеки, а потом и репозитория) с целью устранения указанных недостатков, однако скоро стало ясно, что осуществление подобного рода масштабирования и расширения системы неосуществимо, если возможности для этого не были учтены при изначальном проектировании архитектуры приложения в целом.

1.5. Выводы

Разработка новой CASE-системы общего назначения (поддерживающей язык UML и генерацию в достаточно широкий класс систем, например, Web-приложения) на данный момент представляется нам задачей, вероятность возникновения которой крайне низка, — большое количество довольно качественных систем такого типа представлено на рынке, с различной функциональностью и в разных ценовых категориях. Однако же такие системы не предназначены для разработки пользовательских визуальных языков, редакторов и генераторов к ним.

Поскольку задача создания предметно-ориентированных визуальных языков возникает всё чаще, необходимо иметь возможность быстро создавать CASE-системы для их поддержки. Такие CASE-системы, как нам кажется, не могут быть коробочными, поскольку сами языки и требования к генераторам, редакторам и т. д. могут сильно отличаться не только в разных компаниях, но и в разных проектах в рамках одной компании. Microsoft DSL TOOLS и Eclipse GMF — примеры инструментария для быстрой разработки таких CASE-систем, однако они, по нашему мнению, обладают некоторыми недостатками. Первый из них прост в использовании, но позволяет создавать только графические редакторы, которые являются неотъемлемой частью среды Microsoft Visual Studio. Кроме того, в Microsoft DSL Tools сложно реализуются нестандартные графические нотации. Eclipse GMF существенно более универсален — Rational Software Architect может послужить примером того, что можно сделать с помощью Eclipse GMF, но и существенно более сложен в использовании. Вместе с тем для этого инструментария почти нет доступной документации, и ознакомиться с его функциональными возможностями и особенностями реализации можно, только самостоятельно выполнив значительный проект с использованием этой технологии.

2. Основные концепции QReal

Система QReal имеет клиент-серверную архитектуру и основывается на шаблоне проектирования Model/View. Каждый клиент имеет свою модель, которая обеспечивает доступ к репозиторию, а в роли представлений выступают различные элементы пользовательского интерфейса.

Следует пояснить, почему был выбран именно шаблон Model/ View, а не широко используемый в таких задачах шаблон Model/ View/Controller. Иначе функциональность контроллера брала бы на себя логику графических редакторов, что не очень удобно, поскольку редакторы в системе QReal генерируются, а не программируются «вручную». Разнесение функциональности редакторов по разным модулям усложнило бы генерацию, не предоставив существенных преимуществ. Данное решение детально описано в работе [1].

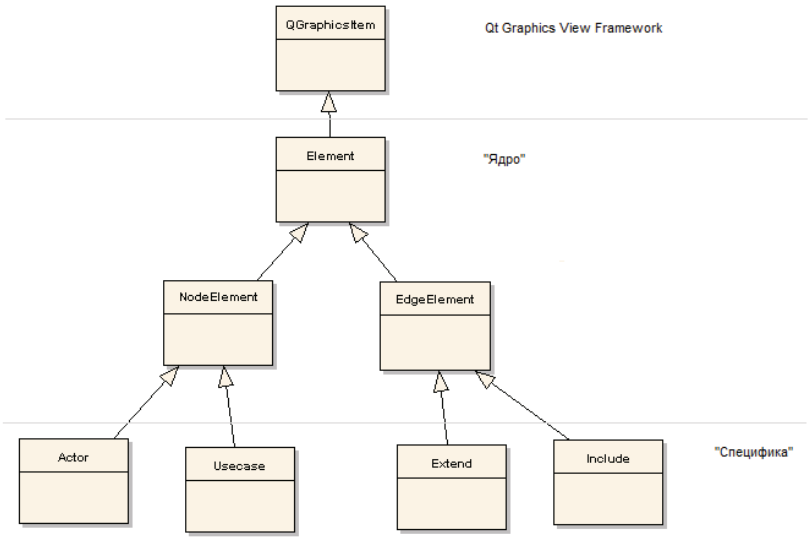

Рис. 1. Архитектура QReal.

Общая архитектура системы изображена на рис. 1. Для обеспечения версионирования и многопользовательской работы используется SVN[10]-сервер. Доступ к нему осуществляется посредством клиентов репозитория, которые хранят свои модели в виде файлов, организованных в рабочую копию системы контроля версий. Генераторы могут быть встроенными в клиентскую часть CASE-пакета (например, генератор XMI) или работать независимо. В таком случае им потребуется компонента, отвечающая за взаимодействие с SVN-репозиторием и представление модели в памяти. Общение с

[10]http://subversion.tigris.org/

ней ведётся по протоколу ICE [19], что позволяет реализовывать генераторы на различных языках программирования.

2.1. Графический интерфейс

Графический интерфейс CASE-пакета обычно содержит вполне определенный набор элементов. В QReal они представлены следующими компонентами (рис. 2).

1. *Инспектор объектов (Object Explorer)* — представляет возможности для работы с деревом объектов текущего проекта. Содержит все элементы в открытом в данный момент репозитории, отсортированные по их типам. Бывает удобен для обзора проекта в целом и при добавлении на текущую диаграмму уже существующего в проекте объекта, например, не принадлежащего никаким диаграммам.

2. *Инспектор диаграмм (Diagram Explorer)* — отображает все пользовательские диаграммы и их элементы в виде дерева. Один элемент может принадлежать нескольким диаграммам, тогда в ин-

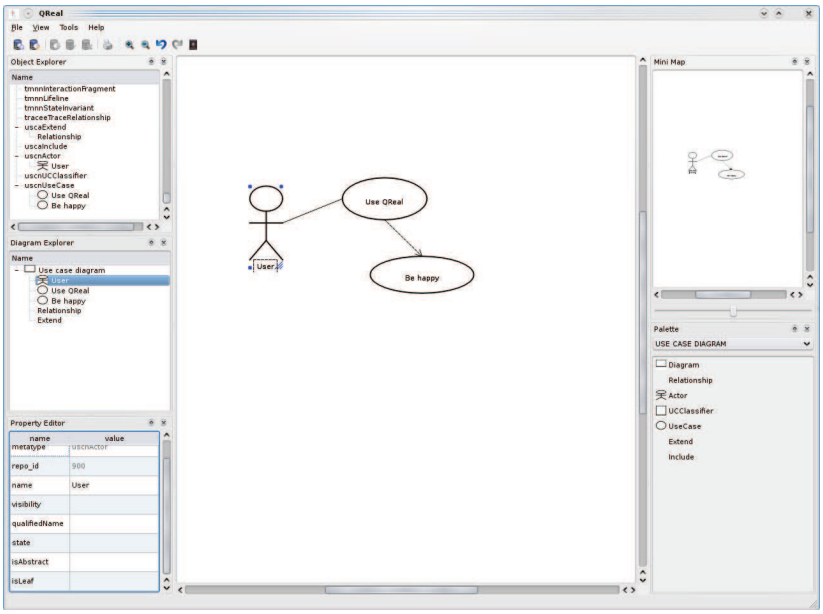

Рис. 2. Главное окно QReal.

спекторе он будет отображаться как потомок всех диаграмм, которым он принадлежит (в инспекторе объектов и в репозитории всем этим «копиям» будет соответствовать один элемент).

3. *Редактор свойств (Property Editor)* — предоставляет информацию о выделенном элементе и позволяет изменять значения его атрибутов.

4. *Палитра компонентов* — отображает все возможные для добавления на диаграмму визуальные элементы.

5. *Меню, панели инструментов* — содержат основные операции над репозиторием, диаграммами, объектами и т. д.

6. *Рабочая область графического редактора диаграмм* — основное рабочее окно CASE-пакета. Здесь осуществляется отображение и модификация создаваемых пользователем диаграмм.

Следует заметить, что реализация подобных элементов графического интерфейса, как правило, является платформо-зависимой задачей, т. е. требует использования системных графических библиотек, что противоречит требованию многоплатформенности. Известно несколько способов решения задач такого рода.

1. Использование переносимости на уровне бинарного кода, например, байт-код Java. Java-приложения могут работать на любом устройстве, для которого существует реализация виртуальной машины.

2. Переносимость приложения на уровне исходных кодов. Это достигается путем использования многоплатформенных инструментариев, инкапсулирующих в себе реализацию базовых графических примитивов для различных операционных систем. Библиотеки подобного рода предоставляют единый интерфейс для работы с ними вне зависимости от используемой операционной системы. Наиболее распространенными готовыми решениями в данной области являются инструментальные средства Qt и GTK+.

Основным достоинством первого подхода является отсутствие необходимости перекомпиляции, что не очень существенно для CASE-систем, поскольку отношение времени компиляции исходных кодов к общему времени работы с продуктом невелико. Достоинством второго подхода является более высокая производительность, что при работе с большими моделями может быть достаточно важно.

В процессе разработки архитектуры и реализации прототипов модулей CASE-пакета было решено использовать инструментарий Qt. Этот выбор показался нам удачным по следующим причинам.

1. Qt — динамически развивающийся проект: только за время работы над QReal было выпущено несколько новых версий этого инструментария, каждая из которых существенно расширяла возможности предыдущей.

2. Если Qt первых версий рассматривалась разработчиками как графическая библиотека, то начиная с версии 4 — это полноценный инструментарий для разработки многоплатформенных приложений. Помимо элементов графического интерфейса он включает в себя классы для работы с сетью, базами данных, XML и многое другое.

3. В Qt с версии 4 входит Qt Model/View Framework, который предоставляет для использования набор готовых реализаций стандартных моделей и представлений, возможность быстро создавать новые.

4. Распространение с версии 4.5 под лицензией LGPL, что даёт возможность использовать Qt в коммерческих приложениях.

5. Перспективы развития инструментария, например портирование QReal средствами Qt в Web-приложение.

2.2. Репозиторий на основе реляционной СУБД

Репозиторий является ответственным за хранение данных и предоставление интерфейсов доступа к ним для других компонент CASE-пакета. В QReal репозиторий организован в соответствии с архитектурой клиент/сервер, где клиентская часть встраивается в сам CASE-пакет, а серверная размещена удалённо. Серверная составляющая может быть реализована:

- на основе стандартных средств версионного контроля (например, CVS, Visual SourceSafe или Subversion);
- в виде хранилища данных с помощью набора плоских файлов;
- как надстройка над реляционной СУБД.

Последний вариант сначала показался нам наиболее удачным, к тому же он уже применялся в системе REAL.

Сравнивая подход, используемый QReal, и подход, представленный в работе [5], необходимо отметить некоторые различия. Последний основывается на стандарте объектно-ориентированных баз данных ODMG [20], в то время как в QReal использовался оригинальный подход, направленный на увеличение быстродействия в условиях низкоскоростных сетей. В результате схема БД получилась иной. Основная информация о поддерживаемых типах вме-

сте с уникальными идентификаторами типов хранится в таблице *metatable*. Все существующие в репозитории элементы перечислены в таблице *nametable* вместе со своими уникальными идентификаторами, типом и теми свойствами, которые являются общими для всех элементов (например, имени). Размещение таких свойств в этой таблице позволяет ускорить доступ, особенно для стандартных представлений инструментария Qt. Помимо этого, есть одна большая таблица *diagram*, которая содержит данные о потомках объекта, в частности о содержимом диаграмм. В этой схеме не только диаграммы содержат внутри себя объекты: в частности, пакеты также могут содержать классы в качестве дочерних элементов. В таблице содержится идентификатор родительского элемента, идентификатор дочернего элемента и параметры, отвечающие за отображение объекта на этой диаграмме (координаты, размеры, конфигурацию ломаной линии и т. д.). Помимо этого, как и в реализации репозитория REAL, существуют и таблицы по одной на каждый тип элемента, в которых хранятся свойства, специфичные для каждого типа элементов. Но в QReal все специфичные свойства конкретного элемента находятся в одной таблице, а в REAL они разнесены и по таблицам родителей. Следует отметить, что в реализации REAL более четко проработана реализация целостности репозитория, которая обеспечивалась средствами СУБД, что не было сделано в QReal в связи с переходом на другую систему репозитория.

Также необходимо отметить и некоторые архитектурные особенности построения. В системе REAL доступ к репозиторию из сторонних приложений осуществляется посредством интерфейса COM и средствами VBScript, что привязывает систему к ОС семейства Windows. Программный интерфейс C++ используется только самими редакторами и недоступен другим приложениям. В системе QReal интерфейс на C++ полностью открыт на уровне логической модели и может использоваться другими системами, а в качестве скриптового языка был предложен язык на основе EcmaScript (так называемый JavaScript), однако систему скриптования еще предстоит доделать.

В качестве системы доступа к БД очевидным способом были выбраны средства Qt, что позволило использовать и клиент, и сервер на любой платформе, причем для многих из серверов отпадала необходимость в ODBC-прослойке.

Более подробное описание схемы базы данных можно найти в работе [6].

2.3. Генеративный подход к созданию редакторов

CASE-пакет — это система визуального моделирования, в состав которой входит определенное число графических редакторов. И будь то редакторы бизнес-процессов, схем баз данных, компонент программного обеспечения или структуры организаций, по своему внутреннему устройству они будут довольно похожи.

Редактор инкапсулирует в себе информацию о наборе объектов, допустимых на диаграммах данного типа, и должен быть способен правильно интерпретировать хранящиеся в репозитории значения атрибутов элементов (например, использовать некоторые из них как параметры при отрисовке). Кроме того, редактор должен иметь представление о логических правилах размещения элементов на соответствующих типах диаграмм (например, возможность соединять некоторые элементы ассоциациями, возможность одних элементов быть контейнерами для других и т. д.).

Создавать набор таких редакторов кодированием их «вручную» кажется нам неразумным по следующим причинам:

— с ростом числа редакторов значительно снижается сопровождаемость кода;
— слабая расширяемость — добавление новой функциональности требует изучения (а нередко и существенного рефакторинга) уже существующего кода;
— опасная масштабируемость — создание дополнительного редактора, являющегося типовым для данного CASE-средства, чаще всего будет осуществляться методом Copy/Paste с дальнейшими доработками полученного после копирования текста, что ведёт как к появлению новых ошибок, связанных с неполнотой вносимых правок, так и к размножению уже существующих;
— происходит неизбежное усложнение внутреннего устройства модуля редакторов и, как следствие, усложнение архитектуры CASE-пакета в целом.

В соответствии с этим был предложен следующий подход к автоматическому созданию графических редакторов:

— путем анализа типовых редакторов выделяется базовая функциональность абстрактного редактора, которая кодируется «вручную» на целевом языке высокого уровня (в нашем случае — на C++);

— специфика метамоделей нужных диаграмм (или наиболее полное их подмножество) описывается с помощью специального XML-формата;
— по этим описаниям генерируется C++ код, который в совокупности с базовой функциональностью полностью реализует требуемую функциональность описанных диаграмм.

В CASE-пакете QReal была реализована следующая инфраструктура:

1) «ядро», реализованное «вручную», обеспечивает следующую функциональность абстрактного редактора:

— способность элементов принадлежать диаграммам (в том числе способность одного элемента принадлежать неограниченному числу диаграмм),
— способность получать значения нужных атрибутов из репозитория и отслеживать изменения его содержимого,
— способность элементов отрисовываться в пределах рабочей области диаграммы,
— способность ассоциаций соединять элементы и способность элементов присоединять к себе ассоциации,
— способность абстрактного элемента быть контейнером для других,
— способность элемента реагировать на действия пользователя (например, на выделение курсором мыши, перемещение, изменение размеров) и т. п.;

2) наследуемые от «ядра» классы определяют специфику конкретных типов диаграмм (генерируются автоматически по XML-описаниям):

— набор допустимых для каждого конкретного типа диаграмм элементов,
— графические представления и параметризация их значениями атрибутов этих элементов,
— стиль начертания ассоциаций, форма и тип возможных стрелок,
— логические правила, специфичные для данного типа диаграмм (например, правила соединения элементов ассоциациями) и т. п.

Необходимый для описания метамоделей диаграмм формат должен предоставлять следующие возможности:

— задание графического представления элемента в текстовом виде (для элементов) или способа начертания линий и задания формы и типа стрелок (для ассоциаций);
— задание параметризации статического изображения элемента содержимым репозитория (например, отображение имени элемента);
— описание набора атрибутов элементов: поддержка как основных типов данных (строки, числа, логический тип, перечисление), так и возможность создания пользовательских;
— указание факта наследования элементов;
— описание логики ассоциаций: задание для каждого конца ассоциации набора допустимых для соединения с ним элементов, задание атрибутов ассоциаций (например, множественности или метки) и т. п.

Формат должен основываться на XML и иметь простую структуру. Тогда создание новых редакторов или внесение изменений в уже существующие не требует ни навыков программирования, ни знания внутреннего устройства всего CASE-пакета и может осуществляться пользователями напрямую.

Формат, обладающий описанными характеристиками и используемый в QReal, подробно описывается в работе [8].

2.3.1. Генерация редакторов по метамодели

Процесс генерации редакторов в системе QReal организован так: созданные описания метамоделей подаются на вход специальной утилите-генератору, которая осуществляет анализ предоставленных ей XML-документов, в результате чего для каждой сущности (как для элемента, так и для ассоциации) определяется список свойств с указанием их типа, список родительских элементов (сущность может наследовать свойства и логику родительских элементов), отображаемое имя сущности и т. д.

Дополнительно сущности, представляющие вершины графа логической модели, имеют секцию описания их графического представления (в первых версиях для его задания использовался язык SVG). Это описание сохранялось в виде файла на диске и использовалось как для отрисовки элементов в рабочей области редактора диаграмм, так и для создания иконок для палитры компонентов, инспектора объектов и диаграмм. Изображения, описанные на языке SVG, являются статичными, поэтому для их параметризации со-

держимым репозитория (например, для отрисовки имени элемента) используется специальная XML-секция. Для этого используются расширенные соответствующим образом HTML-теги, поскольку инструментарий Qt предоставляет классы для интерпретации и отрисовки основных тегов HTML [17]. Таким образом, обеспечивается форматирование подставляемых из репозитория значений. В ходе разбора соответствующих секций XML-описаний генератор заменяет теги параметризации соответствующим C++ кодом для получения нужных значений атрибутов элемента, при этом теги форматирования текста сохраняются. Во время выполнения полученного кода произойдет обращение к репозиторию, и значение атрибута будет отображено в рабочей области редактора поверх статичного SVG-изображения элемента.

Здесь следует указать на то, что использование формата SVG, как и любого другого графического формата общего назначения, позволяет реализовать только самые простые редакторы, поскольку в развитом CASE-средстве графические элементы могут обладать достаточно сложным поведением, не выразимым ни статическим, ни параметризованным изображением. Представляется, что CASE-средства должны иметь специализированный формат задания графических элементов. В системе QReal от использования формата SVG в результате пришлось отказаться, хотя общий принцип наличия секции с графическим представлением элементов в XML-описаниях был сохранён. Самым простым примером невыразимого в SVG аспекта отрисовки элементов являются порты.

Порты — это области графического отображения элемента, к которым возможно присоединение ассоциаций. Формат (а следовательно, и генератор) поддерживает задание портов двух типов — точка и отрезок (в случае последнего ассоциация может быть прикреплена к любой точке отрезка). Вся функциональность по обработке присоединения к портам и адекватного отображения ассоциаций при изменении размера элемента, а также при его движении реализована в ядре редактора, в генерируемых классах конкретных элементов достаточно лишь указать тип и расположение порта.

Для сущностей, представляющих рёбра графа модели, для каждого из концов ассоциации указывается набор типов элементов, к которым она может быть прикреплена, тип начертания линии ассоциации при ее визуальном отображении, формы и стиля закраски стрелок, например, задание обычных стрелок и стрелок в виде ромба (как закрашенные, так и нет для обоих вариантов).

После разбора XML-описаний метамоделей производится построение набора свойств сущностей (с учетом наследования элементов), списка возможных для присоединения элементов ассоциаций (для каждого из концов), проверяется ссылочная целостность, выполняются другие проверки семантической корректности описаний редакторов.

В ходе заключительной фазы происходит генерация необходимых артефактов, а именно создаются:

- классы на языке C++ для всех элементов и ассоциаций диаграмм;
- файлы с описаниями графического представления элементов;
- внутренние средства доступа к значениям атрибутов элементов в репозитории из любых модулей проекта;
- дополнительный код для присоединения генерируемых классов к проекту; в него входят фабрика объектов для создания экземпляров описанных типов элементов и ассоциаций, а также служебные файлы — файл ресурсов проекта и файл, осуществляющий включение генерируемых файлов с классами описанных сущностей в процесс сборки проекта.

Подробнее генератор редакторов описан в работе [1].

3. Дальнейшее развитие QReal

3.1. Смена способа хранения данных

После создания и апробации первого работающего прототипа системы QReal были сформированы основные замечания к нему.

1. Использование реляционной СУБД в качестве сервера репозитория затрудняет оповещение подключенных клиентов об изменении хранимых данных.
2. Состав хранимых на сервере данных определялся схемой текущей базы данных, которая генерировалась по описаниям метамоделей. В результате если несколько клиентов имели бы разный набор редакторов (или, что еще хуже, разные версии одних и тех же редакторов), то работа с ними сопровождалась бы большим количеством ошибок, содержимое репозитория с большой вероятностью было бы испорчено.
3. Неэффективными оказались некоторые типы запросов (упреждающая загрузка, отложенное чтение) и кэширования.

Все эти проблемы могли бы быть устранены либо расширением клиента репозитория (реализацией механизма опроса, хранением дополнительной информации о редакторах и проверкой ее в клиенте перед началом работы и т. п.), либо введением некой программной «прослойки» между клиентом репозитория и СУБД, в которую можно было бы вынести всю необходимую дополнительную функциональность.

Анализ этих и некоторых других замечаний к существующему решению дал понять, что, хотя выбранный при создании прототипа системы подход построения репозитория позволил в сжатые сроки получить большую частью требуемой функциональности, в перспективе потребуется его существенная доработка.

В итоге было принято решение о реализации собственного сервера репозитория, представляющего собой простую объектно-ориентированную СУБД. При этом клиент репозитория должен инкапсулировать в себе всю требуемую информацию для осуществления взаимодействия с удаленным сервером и предоставлять всем остальным компонентам системы доступ к данным посредством фиксированного API.

Первоначально общение между клиентом и сервером репозитория осуществлялось с помощью текстового протокола поверх TCP/IP, однако с появлением первых генераторов CASE-пакета была поставлена задача поддержки протокола более высокого уровня. Это объяснялось тем, что первые генераторы писались на языке C# под платформу .NET, а клиент репозитория — на языке C++ с обширным использованием возможностей инструментария Qt. Поэтому удачным, на наш взгляд, решением проблемы организации взаимодействия этих компонент явилось внедрение высокоуровневого RPC-протокола взаимодействия клиента и сервера репозитория — таким образом, для каждого языка/платформы этот интерфейс необходимо реализовать только один раз. В качестве такого протокола был выбран ZeroC ICE [19].

3.2. Замена формата SVG

Еще одной проблемой созданного прототипа было масштабирование графических изображений элементов на диаграммах. Для задания представлений элементов использовался формат SVG, что позволяло быстро создавать векторные изображения и обрабатывать их стандартными средствами модуля QtSvg (в Qt реализована

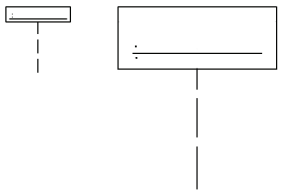

Рис. 3. Элемент до (слева)
и после (справа) масштабирования.

поддержка SVG 1.2 Tiny[11]). Однако изображения векторной графики при масштабировании изменяют свои размеры и толщину линий пропорционально растяжению всей фигуры, что приводило к ситуациям, проиллюстрированным на рис. 3.

Пользоваться такими фигурами было неудобно, а так как формат SVG не предоставляет средств для его расширения, была поставлена задача создания собственного языка описания графических представлений элементов на замену SVG (и, соответственно, реализация средств отрисовки изображений, описанных с помощью этого языка). Основными требованиями к этому языку стала поддержка дифференцированного масштабирования элементов — возможность при описании элемента задавать тип масштабирования составляющих его частей. В результате был создан язык SDF (stencil description format) и средства его отрисовки, которые поддерживают два вида масштабирования.

Абсолютное масштабирование. Размер графического примитива задается в явном виде и при масштабировании не меняется — пример представлен на рис. 4.

Процентное масштабирование. Размер графического примитива задается в процентах от общего размера элемента (сюда же входит и «обычное» пропорциональное масштабирование) — пример представлен на рис. 5.

Кроме того, были созданы средства конвертации SVG-описаний в формат SDF для преобразования большого числа уже имеющихся SVG-изображений элементов. К тому же мы получили гибкий формат, который можно быстро изменять под свои дальнейшие нужды.

[11] http://www.w3.org/TR/SVGMobile12/

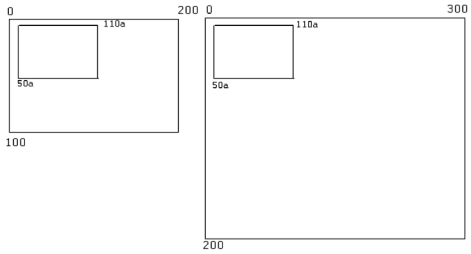

Рис. 4. Пример абсолютного масштабирования.

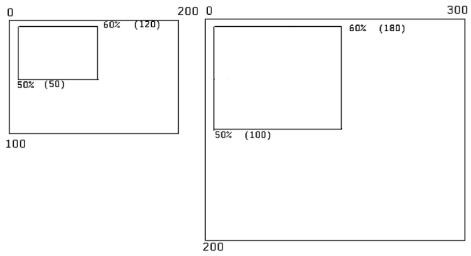

Рис. 5. Пример процентного масштабирования.

3.3. Выделение логической модели

В результате работы над QReal возникло более ясное понимание того, как модели должны храниться в репозитории и отображаться в пользовательском интерфейсе. Проблема, общая для всех CASE-систем, состоит в том, что было бы удобно иметь хорошо выраженную логическую модель разрабатываемой системы, с которой удобно работать генераторам кода, и при этом иметь возможность в процессе моделирования рассматривать моделируемую систему с различных точек зрения. К примеру, модель некоторой системы может содержать диаграммы конечных автоматов, описывающие поведение каких-либо классов, и диаграммы последовательности, которые описывают некоторые сценарии взаимодействия объектов этих классов. Естественно, такая модель может содержать избыточную информацию, которую необходимо автоматически поддерживать согласованной.

Поэтому имеет смысл явно разделить модель на логическую и графическую. При этом одной логической сущности может соответствовать несколько (или ни одного) графических представлений. Представления имеют атрибуты, относящиеся непосредственно к графике, как то: положение, размеры, форма, цвет и т. д. Все существенные с точки зрения логики модели атрибуты (поля, методы, ассоциации и т. д.) хранятся в логической сущности и только отображаются представлениями. Это полезно для поддержания консистентности, поскольку, меняя существенные атрибуты одного представления логической сущности, мы меняем саму логическую сущность, автоматически меняя тем самым остальные её представления.

Сущности логической и графической модели взаимосвязаны следующим образом.

1. Элемент графической модели может иметь один прототип — логическую сущность («обычные» элементы) или не иметь прототипа — специальные графические сущности, предназначенные для группировки графических элементов, аннотации диаграмм, рисования и т. д. Пример такого представления, не имеющего логического прообраза, — диаграмма (как элемент графической модели). Она может рассматриваться как способ группировки графических элементов и не должна быть «видна» генераторам (которые, как правило, используют для группировки пакеты).
2. Один элемент логической модели может иметь ноль или больше представлений. Представления могут не только иметь разные параметры отображения, но и вообще задаваться разными фигурами и иметь разный набор «существенных» атрибутов. Элемент логической модели можно представить как объединение существенных атрибутов всех его возможных представлений (существенными атрибутами представления здесь называются атрибуты логической сущности, с которыми пользователь может взаимодействовать через представление).
3. Связи между элементами логической модели довольно просты — отношение «родитель/сын» (точнее, «контейнер—элемент») и ссылки на другие объекты или ассоциации. В графической модели связи между элементами могут отражать способ визуальной группировки элементов и могут быть довольно сложными — «родитель/сын» (один элемент может на-

ходиться внутри другого и, например, перемещаться по диаграмме как единое целое), «встроенный» элемент, имеющий фиксированную внутри элемента-родителя позицию (например, метод в классе), и прочие виды связей, которые могут задаваться, например, графическими грамматиками [3].

4. Хранить в репозитории необходимо отдельно логическую и графическую части модели. Выделение графических представлений в отдельные сущности репозитория полезно для организации многопользовательской работы, поскольку в этом случае изменение графических свойств какого-либо представления может быть осуществлено независимо от логической сущности, а значит, с меньшей вероятностью приведёт к конфликту.
5. В пользовательском интерфейсе должны отображаться обе части модели (например, в виде двух отдельных деревьев), при этом должна быть обеспечена синхронность вносимых изменений.

3.4. Метаредактор

Долгое время XML-описания редакторов в системе QReal писались «вручную», однако сами XML-описания можно рассматривать как целевой текстовый язык для генерации из визуальной модели. Для этого в QReal был создан ещё один графический редактор (метаредактор, т. е. редактор графических редакторов). По сути, метаредактор предоставляет пользователям визуальный язык, являющийся подмножеством стандартного языка MOF (Meta Object Facility) [14], на котором описываются все метамодели, например метамодель UML. Практика показывает, что достаточно реализовать лишь небольшой набор элементов MOF:

— классы (задают множество элементов конкретного типа);
— ассоциации между ними (например, для задания отношения наследования);
— атрибуты классов (например, используемые для задания свойств элементов заданного типа).

На данный момент нам кажется, что этих сущностей достаточно, чтобы описать метамодели всех возможных редакторов, поддерживаемых нашим CASE-средством. Метаредактор был создан в

достаточно короткое время, что подтвердило эффективность предлагаемого генеративного подхода к созданию редакторов.

Заключение

В статье описаны подходы к созданию среды визуального моделирования, опыт их апробации и сформированная на их основе архитектура такой среды. Представленная архитектура была реализована на уровне прототипа в пакете QReal, разработанном на кафедре системного программирования математико-механического факультета СПбГУ.

В качестве основных проблем, возникающих при реализации систем такого класса, были выделены следующие:

- выбор способа хранения создаваемых пользователями моделей, с учетом требований по обеспечению многопользовательской работы, поддержки версионирования и ссылочной целостности. Реляционные базы данных, представляющиеся наиболее естественными кандидатами на эту роль, оказываются недостаточно эффективны в данном случае в силу указанных в работе специфических требований;
- отделение логической модели от ее графического представления на диаграммах;
- разработка набора графических примитивов и способа конструирования из них элементов диаграмм, позволяющих достаточно полно реализовывать различные диаграммы.

В настоящий момент работа над QReal продолжается, соответственно продолжается и поиск эффективных решений возникающих при этом проблем, в том числе и обсуждаемых в данной статье.

Список литературы

[1] *Брыксин Т. А.* Model/View-архитектура CASE-пакета REAL-MV. Дипломная работа. СПбГУ. 2007. 42 с. http://unreal.tepkom.ru/trac/attachment/wiki/Diplomes/ Bryksin_Diploma.doc

[2] *Иванов А., Кознов Д., Лебедев А., Мурашева Т., Парфенов В., Терехов А.* Объектно-ориентированное расширение технологии RTST // Записки семинара кафедры системного программирования, CASE-средства RTST++. Изд-во СПбГУ, 1998. С. 17–36.

[3] *Кознов Д. В.* Основы визуального моделирования. Лаборатория знаний, Интернет-университет информационных технологий; БИНОМ, 2008. 280 с.

[4] *Кознов Д. В., Ольхович Л. Б.* Визуальные языки проектов // Системное программирование. Вып. 1: Сб. статей / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб., 2004. С. 148–167.

[5] *Кондратьев А.М.* CASE-средства и объектные базы данных // Объектно-ориентированное визуальное моделирование / Под ред. А. Н. Терехова. СПб.: Изд-во СПбГУ, 1999. С. 57–78.

[6] *Никандров Г.* Реализация кроссплатформенной архитектуры CASE-пакета. Дипломная работа. СПбГУ. 2007. 60 с. http://unreal.tepkom.ru/trac/attachment/wiki/Diplomes/Nikandrov_Diploma.pdf

[7] Семейство стандартов Integration DEFinition, http://www.idef.com/

[8] *Симонова А. А.* Подход к разработке CASE-пакетов. Дипломная работа. СПбГУ. 2007. http://unreal.tepkom.ru/trac/attachment/wiki/Diplomes/Simonova_Diploma.doc

[9] *Терехов А. Н., Романовский К. Ю., Кознов Д. В., Долгов П. С., Иванов А. Н.* REAL: методология и CASE-средство для разработки систем реального времени и информационных систем // Программирование. 1999. № 5. С. 44–52.

[10] Business Process Modeling Notation. Final Notation Specification dtc/06-02-01, OMG, 2006. http://www.omg.org/spec/BPMN/1.2/PDF

[11] ITU-T Recommendation Z.100: Specification and Description Language. 1993. 204 p.

[12] *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.

[13] *Kelly S., Lyytinen K., Rossi M.* MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment // Proceedings of the 8th International Conference on Advances Information System Engineering. 1996. P. 1–21.

[14] Meta Object Facility (MOF) Core Specification, Version 2.0, Object Management Group. 2006. http://www.omg.org/spec/MOF/2.0/

[15] OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.2, OMG, 2009. http://www.omg.org/spec/UML/2.2/Infrastructure/PDF

[16] OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.2, OMG, 2009. http://www.omg.org/spec/UML/2.2/Superstructure/PDF

[17] Supported HTML Subset, Qt Documentation. Nokia Corporation, 2009. http://doc.trolltech.com/4.5/richtext-html-subset.html

[18] Thomas G. Muth, Functional Structures in Networks. Springer, 2005.

[19] The Internet Communications Engine, ZeroC Inc., 2009. http://www.zeroc.com/ice.html

[20] The Object Data Management Standard: ODMG 3.0 / Ed. by R. G. G. Cattell, Douglas K. Barry, Mark Berler et al. Morgan Kaufmann, 2000. 256 p.
[21] http://www.qtsoftware.com/
[22] http://msdn.microsoft.com/en-us/library/bb126235.aspx
[23] http://www.eclipse.org/modeling/gmf/