О средствах разработки проблемно-ориентированных визуальных языков*

А. А. Павлинов
arctyc@mail.ru

Д. В. Кознов
dkoznov@yandex.ru

А. Ф. Перегудов**
peregudov@dip.spb.ru

Д. Ю. Бугайченко
arhangel@tepkom.ru

А. С. Казакова
nastia@oktetlabs.ru

Р. И. Чернятчик
chukovskij@mail.ru

Т. А. Фесенко***
Tatiana.Fesenko@borland.com

А. Н. Иванов
iw@tepkom.ru

DSM (Domain Specific Modeling) — это метод разработки ПО, подразумевающий создание предметно-ориентированных визуальных средств (языков, методов, программных инструментов). Этот метод широко используется при разработке ПО в рамках концепции семейства программных продуктов (product line). В данной работе делается обзор самых зрелых на сегодняшний день DSM-платформ: технологий `Eclipse/Graphical Modeling Framework` (`GMF`) и `Microsoft DSL Tools`, пакетов `MetaEdit+` и `Microsoft Visio 2003`. Выдвигается идея платформенно-независимой DSM-методологии, сформулированы требования, которым она должна удовлетворять. Представляется DSM-средство, созданное нами на основе `Microsoft Visio` для графического проектирования семейства телевещательных систем санкт-петербургской компании «ДИП».

* Работа выполнена при финансовой поддержке МинРосНауки (грант РИ-19.0/002/273.

** Санкт-Петербургский государственный университет кино и телевидения. 191126, ул. Правды, 13, Санкт-Петербург, Россия.

*** Компания "Борланд". 194044, Финляндский пер, д. 4а, Санкт-Петербург, Россия.

Введение

В настоящий момент DSM-подход становится востребован индустрией, поскольку на рынке появляются DSM-платформы, позволяющие произвольной компании-разработчику ПО создавать свои собственные средства визуального моделирования. Эти средства (а) способны максимально точно учесть специфику компании и удовлетворить именно ее нужды и потребности; (б) могут быть разработаны в рамках приемлемых трудозатрат. Такими технологиями являются `Eclipse/Graphical Modeling Framework`[1] (`GMF`), `Microsoft DSL Tools`[2], `MetaEdit+`[3], а также продукт компании Microsoft под названием `Visio 2003`, интегрированный со средой разработки `Microsoft Visual Studio`, и платформа `XMF-Mosaic` компании `Xactium`[4].

Однако, несмотря на активное развитие DSM-подхода, на текущий момент отсутствуют сводные обзоры различных DSM-платформ, нет рекомендаций по использованию той или иной платформы, а сами они, во многом, еще «сырые», для них не накоплена критическая масса успешных промышленных экспериментов.

В данной работе делается обзор самых зрелых на сегодняшний день DSM-платформ: технологий `Eclipse/Graphical Modeling Framework` (`GMF`) и `Microsoft DSL Tools`, пакетов `MetaEdit+` и `Microsoft Visio 2003`. Делается анализ целесообразности использования той или иной платформы в различных условиях, выдвигается идея единой DSM-методологиии, формулируются требования, которым должна удовлетворять такая методология. Представляется и анализируется пример разработки DSM-средства, созданного нами на основе платформы `Microsoft Visio/.Net` для графиче-

[1] http://www.eclipse.org/gmf/.

[2] http://msdn.microsoft.com/vstudio/DSLTools/.

[3] Продукт создан финской компанией MetaCase (http://www.metacase.com/). Описание принципов, положенных в его основу, содержится в [16].

[4] http://albini.xactium.com/web/.

ского проектирования семейства телевещательных систем санкт-петербургской компании «ДИП».

1. О DSM-подходе

1.1. Мотивация

Визуальное моделирование — это подход, используемый при разработке и сопровождении ПО и основанный на создании чертежей программ с помощью, главным образом, различных графовых моделей. Средствами визуального моделирования являются визуальные языки, методы их использования и программные средства, реализующие эти языки и методы [7]. Фактически, предполагается перенести подходы к проектированию сложных инженерных объектов на основе чертежей, используемых в строительстве, машиностроении, энергетики и т. д., на индустрию производства программного обеспечения. Исходно эта идея появилась в рамках структурного анализа (конец 60-х – 80-е годы), была развита в рамках методов проектирования телекоммуникационных систем (70-е – 90-е годы)[5], подхвачена методологиями объектно-ориентированного анализа и проектирования (90-е годы), стандартизована и широко распространена в индустрии в рамках инициатив комитета OMG[6] (с конца 90-х годов по настоящее время).

Тем не менее, до сих пор не существует универсальных методов визуального моделирования ПО, что связано с невидимостью программных продуктов [10] и отсутствием естественной и общепринятой метафоры для визуализации ПО [7], а также с тем, что на данный момент вообще не существует универсального процесса разработки ПО [22]. Как правило, любое средство визуального моделирования требует адаптации и настройки для удовлетворения нужд конкретного процесса, особенно, если есть потребность в автоматизации использования визуальных моделей — генерации целевого кода системы, валидации моделей, отладки исполняемых спецификаций в терминах модели и пр. [5] В частности, язык UML [20] хорошо зарекомендовал себя как средство моделирования, документирования и коммуникаций, которое используется разработчиками на разных стадиях создания ПО. Однако, вопреки постоянному развитию UML, уточнению его исполняемой семантики и метамодели, введению средств расширений языка (профайлов), машинная обработка UML-моделей в общем случае является непростой и дорогостоящей задачей [13]. Анализируя отчеты успешного использования средств поддержки визуального моделирования компании Telelogic AB — одного из мировых лидеров по созданию программных средств в этой области, — можно обратить внимание, что во многих случаях, когда использовалась кодогенерация на основе продуктов этой компании, сами продукты дорабатывались под нужды заказчика[7]. На сегодняшний день многие компании, например Моторола (санкт-петербургское отделение), имеют свои собственные средства визуального моделирования, основанные на стандартных [3]. В качестве еще одного примера можно привести технологию RTST [9], созданную и использующуюся для разработки телекоммуникационных систем в компании ЗАО «ЛАНИТ-ТЕРКОМ», реализующую вариант языка SDL.

Все эти примеры наводят на мысль о некоторой общей ситуации с инструментами визуального моделирования, которая обобщается в рамках подхода предметно-ориентированного моделирования (DSM, Domain-Specific Modeling). Суть этого подхода — в сужении целевой области применения визуального моделирования (вплоть до отдельных проектов) и достижении, за счет этого, большей эффективности автоматизированных решений, созданных на его основе [15]. При этом ключивым аспектом DSM-подхода является создание и наладка (с приемлемыми затратами) в компании инструментальных средств, поддерживающих тот вариант использования визуального моделирования, который оптимален для компании. Выработка самого этого способа бессмысленна вне проектирования и реализации соответствующих инструментов.

При этом можно использовать уже имеющиеся на рынке средства типа пакетов `IBM Rational Rose`, `Borland Together Control Center` и пр., создавая на их основе технологические решения [5]:

[5] Стандарты европейского комитета CCITT, ныне ITU (http://www.itu.int) — языки моделирования SDL (Specification and Description Language) и MSC (Message Sequence Chart).

[6] Международный комитет OMG (www.omg.org) с конца 90-х годов занялся стандартизацией и дальнейшим развитием языков, методов и концепций в области визуального моделирования: универсальный язык моделирования UML, метод проектирования многоплатформенных систем с помощью UML (Unified Modeling Language) — MDA (Model-Driven Architecture), визуальный язык спецификации бизнес-процессов BPMN (Business Process Management Notation).

[7] http://www.telelogic.com/customers/success-stories.cfm.

все подобные средства имеют открытый программный интерфейс, модульную архитектуру; более того, язык UML, который они реализуют, поддерживает механизм расширения (extension-mechanism), позволяющий создавать на базе UML различные языки-диалекты. Однако, как отмечалось в [5], наладка таких пакетов на практике часто оказывается весьма трудоемким процессом — начиная от «доделки» языка UML (непросто даже разобраться в механизме расширения UML) и заканчивая использованием программного интерфейса стандартных пакетов, который часто плохо документирован, содержит ошибки, наконец, попросту неполон, т. е. не предоставляет всех необходимых возможностей.

В этой ситуации оказывается, что во многих случаях проще создать свой собственный небольшой визуальный язык и реализовать свой собственный графический редактор, генератор кода и т. д. (о функциональном составе целевых DSM-средств см. ниже). Тем более что на рынке появились для этого специальные средства (мы называем их DSM-платформами), которые и будут рассмотрены ниже.

Выбор между настройкой стандартных средств визуального моделирования и разработкой собственных, рекоменданции и шаблоны при разработке собственных средств, анализ достоинств DSM-платформ и многое другое — все это должно стать содержанием DSM-методологии.

1.2. Терминология

Дадим несколько определений, которые используются в дальнейшем изложении. *Предметная область* (problem domain, domain) — это часть реального мира (люди, знания, бизнес-интересы и пр.), объединенная в одно целое для удовлетворения определенных потребностей рынка. Предметная область может быть достаточно абстрактной — какое-нибудь открытое сообщество, например, сообщество разработчиков Linux[8] — так и очень определенное, с ясно очерченными границами, например, инфраструктура по разработке семейства продуктов в рамках какой-либо компании[9].

Предметно-ориентированный язык (Domain Specific Language, DSL) — язык, который создается для использования в рамках определенной предметной области. Мы будем рассматривать только визуальные предметно-ориентированные языки, часто используя для их обозначения аббревиатуру DSL.

DSM-средства — это конкретный DSL, метод его использования, а также соответствующие средства инструментальной поддержки.

DSM-пакет (DSM-продукт, DSM-инструменты) — часть DSM-средств, соответствующих программным средствам. DSM-платформа — это инструментальная технология разработки DSM-средств (например, `Eclipse/GMF`, `Microsoft DSL Tools`).

1.3. Структура DSM-пакета

Функциональность DSM-пакетов очень важна, так как их созданием должны заниматься обычные software-компании, которые не специализируются на создании визуальных средств, а имеют свои собственные проекты, в которых планируемый DSM-пакет должен принести определенную пользу. Поэтому создание таких пакетов должно быть им посильно. Это, с одной стороны, достигается путем использования DSM-платформ, реализующих такие трудоемкие компоненты, как графические средства, репозиторий, среду. С другой стороны, функциональность целевого DSM-пакета должна быть четко определена, чтобы не делалось никакой лишней работы. То, что может позволить себе, например, продукт `Borland Together Control Center`[10], часто непосильно для DSM-средств. Итак, DSM-пакет может включать в себя следующие компоненты.

- *Среда* — MDI-приложение, главное меню и панель инструментов (с реализацией общих функций, таких как создание/удаление/открытие/закрытие диаграмм и всего проекта, печать диаграмм, настройка цветов, шрифтов, толщины линий, геометрии и т. д.), несколько рабочих областей (поле для рисования, браузер проекта и т. д.), палитра с элементами графической нотации DSL, поддержка разбиения проекта на страницы, браузер модели и пр. Среды могут существенно различаться по предоставляемому набору функциональности: от сред с поддержкой минимального набора простейших функций до сложных сред с реализацией многопользователь-

[8] http://www.linux.org/.

[9] Мы основываемся здесь на определении предметной области, данном в работе [11].

[10] Один из лидирующих продуктов на рынке средств визуального моделирования, принадлежит компании Borland (http://www.borland.com/us/products/together/).

ской работы с моделями, интеграцией со средствами контроля версий и т. д.

- *Графический редактор* — основную рабочую область для рисования диаграмм с возможностью расположения на ней различных фигур (экземпляров графических конструкций языка), применения к этим фигурам набора операций (наполнения текстом, растягивания-сжатия, передвижения, группировки, разбиения на слои, соединения фигур линиями и т. д.), задания для фигур и линий графических свойств (выбора толщины линий, цвета, свойств шрифтов). Кроме того, редактор может поддерживать выполнение различных функций над диаграммами, таких как переключение между несколькими режимами отображения конструкций, навигацию (через запросы и подсветку найденных сущностей), специфические функции — автоматическую нумерацию элементов, различные варианты автоматического размещения элементов на диаграмме (layout) и т. д.

- *Репозиторий* — единое хранилище информации о визуальных моделях, создаваемых в DSM-пакете, с возможностью быстрого доступа во время работы графического редактора.

- *Генераторы* — средства автоматического создания по графическим моделям программного кода (возможно и других артефактов, например, текстовых документов, табличных отчетов, тестов), импорта/экспорта моделей в различные форматы.

- *Средства проверки корректности моделей* — отладчики модельных спецификаций, валидаторы моделей, средства тестирования в терминах моделей и т. д.

- *Прочие программные средства*, интегрированные с графическими редакторами и предназначенные для смежной деятельности, — например, диалоговый редактор текстовой части графического языка.

DSM-пакет может быть как простым графическим редактором, автоматизирующим создание каких-либо диаграмм, так и сложным программным продуктом, поддерживающим кодогенерацию по моделям, отладку спецификаций в терминах модели, процедуру циклической разработки (round-trip engineering)[11].

Интегрируемость DSM-пакета в среду программирования, используемую разработчиками, является важным условием его успешного применения. Например, реализация отладки визуальных спецификаций требует интеграции с отладчиками нижнего уровня — например, Java- или .Net-отладчиками — с тем, чтобы не создавать для отладочного исполнения UML-моделей свое собственное исполняемое ядро [18]. Естественно, выбор отладчика нижнего уровня во многом диктуется той средой разработки, которая используется в проекте.

Нередко DSM-пакеты становятся частью целевых продуктов, создаваемых в процессе разработки. Например, специально созданный редактор бизнес-процессов (DSM-пакет, специально созданный для данного проекта) может присутствовать в системе документооборота, разрабатываемой в данной компании. Поэтому еще одной важной характеристикой таких DSM-продуктов является их компактность и отчуждаемость (как физическая, так и правовая) от среды разработки. Это требуется не всегда, но иногда оказывается важным.

По этим причинам не может быть единственной, универсальной DSM-платформы, и поэтому нужна общая методология, классифицирующая возможности разных платформ, предоставляющая различные шаблоны и методики создания DSM-средств для разных ситуаций.

2. Обзор DSM-платформ

2.1. Технология Eclipse GMF

Среда `Eclipse` — это кросс-платформенная интегрированная среда разработки программного обеспечения с открытыми исходными кодами. В середине 2006 года вышла первая версия технологии Eclipse Graphical Modeling Framework[12] (GMF), основная задача которой — обеспечить «мост» между двумя другими, широко используемыми и известными технологиями создания средств

[11] Метод, позволяющий изменять как визуальные модели, так и сгенерированный по ним код, с распространением изменений в код и модели соответственно [21].

[12] http://www.eclipse.org/gmf/.

визуального моделирования — Eclipse Modeling Framework (EMF) и Graphical Editing Framework (GEF). Архитектура DSM-пакета строится на основе MVC-шаблона[13]. Для создания уровней представления и контроллеров используется технология GEF, для создания моделей — технология EMF.

Технология GEF состоит из двух частей: модуля `org.eclipse.draw2d` (далее — OED[14]), встраиваемого в `Eclipse`, и самой библиотеки GEF. Модуль OED предоставляет средства программной работы (создания и обработки) графических объектов. В его состав входят менеджеры размещения графических объектов, механизм событий, палитра стандартных объектов, средства их комбинирования для создания более сложных композитных графических объектов, средства установления соединений между графическими объектами, функциональность Drag&Drop, средства работы со слоями изображений и пр. Данный модуль может использоваться автономно, как графическая библиотека.

Графические редакторы, как правило, визуализируют некоторую информацию, существующую и обрабатываемую отдельно (уровень модели шаблона MVC) — например, визуализация параллельных процессов, каких-нибудь сложных структур данных. Эта визуализируемая область в понимании модуля OED представляется как модель объектов — экземпляров Java-классов. Для того чтобы созданный с помощью OED графический редактор мог эффективно с ними взаимодействовать, между ними должна существовать «прослойка», отвечающая за их синхронизацию. Такие «прослойки» создаются при помощи библиотеки GEF. Для создания классов-контроллеров GEF предоставляет набор базовых классов, которые должны быть использованы разработчиками при реализации слоя синхронизации. Важно отметить, что все изменения модели производятся контроллерами не напрямую, а с использованием механизма команд. Это дает возможность для простой реализации возврата изменений.

Графические редакторы на основе GEF можно создавать на базе любых моделей, однако наибольшей эффективности в смысле

[13] MVC (Model View Controller) — известного шаблона проектирования, разделяющего уровень хранения (model) данных, уровень представления данных (view) и промежуточный уровень (controller), связывающий хранение и представление [2].

[14] Эта аббревиатура не является стандартной, а предложена нами для удобства ссылок в тексте статьи на этот модуль.

DSM-подхода можно достичь, используя технологию GEF в паре с EMF.

Технология EMF предназначена для создания приложений, использующих модели сложно устроенных бизнес-данных. Реализация таких моделей (исходный код для их run-time классов) производится с помощью средств генерации EMF по схеме модели, описанной средствами библиотеки Ecore[15] или импортированной из одного из следующих форматов: XMI[16]-документ (как созданный «вручную», так и сгенерированный другими средствами, например, при экспорте UML-модели из пакета IBM Rational Rose[17]), набор аннотированных Java-интерфейсов, XML-схема. Механизм генерации рассчитан на то, что сгенерированный код будет расширяться «вручную» с сохранением пользовательских изменений при последующих повторных генерациях. Сгенерированные таким образом модели имеют ряд важных функций (features): механизм сохранения/загрузки в формате XMI, базовые средства («заглушки») для реализации механизма валидации моделей, удобный программный интерфейс (далее — API[18]) для манипуляции объектами модели. Дополнительно, с помощью входящего в состав EMF модуля EMF.Edit, может быть сгенерирован уровень адаптеров (аналог контроллеров GEF) для связи с элементами пользовательского интерфейса, обеспечивающими редактирование EMF-моделей.

Основными недостатками использования связки GEF/EMF как DSM-платформы является плохая согласованность интерфейсов этих технологий. Кроме того, каждая из них предназначена для более широкого класса задач, чем поддержка DSM-подхода. Так, например, с помощью GEF можно создавать визуальную часть графического редактора и связку графических объектов с модельными, а с помощью EMF — структурированные модели бизнес-объектов с механизмами контроля визуализации и изменений. Каждая из этих технологий создавалась независимо друг от друга, поэтому совместное их использование требует большого количества «ручной» работы. Для преодоления этих проблем был реализован

[15] Ecore — библиотека `Eclipse` для описания метамоделей.

[16] XMI (XML Metadata Interchange) — стандарт OMG, описывающий обмен метаданными в формате XML.

[17] Одно из ведущих промышленных средств визуального моделирования принадлежит компании IBM, http://www-306.ibm.com/software/rational/.

[18] API (Application Programming Interface) — это общепринятое сокращение для обозначения программного интерфейса компоненты или продукта.

проект Eclipse GMF, автоматизирующий создание DSM-пакетов на базе библиотек GEF/EMF.

Процесс разработки DSM-пакетов на основе GMF изображен на рис. 1 и состоит из следующих шагов:

1) Разработки описания метамодели языка, например, с помощью графического редактора GMF Ecore (выходной файл `*.ecore`).

2) Разработки описания графического редактора (выходной файл `*.gmfgraph`).

3) Разработки описания вспомогательных средств — палитры объектов, списка действий, меню графических объектов (выходной файл `*.gmftool`).

4) Разработки описания связки метамодельных и графических объектов, а также вспомогательных средств, описание ограничений на OCL или Java (выходной файл `*.gmfmap`).

5) Создания описания генератора, его модификация (выходной файл `*.gmfgen`).

6) Генерации когда целевого DSM-средства, запуск отладочного экземпляра Eclipse Application и отладка.

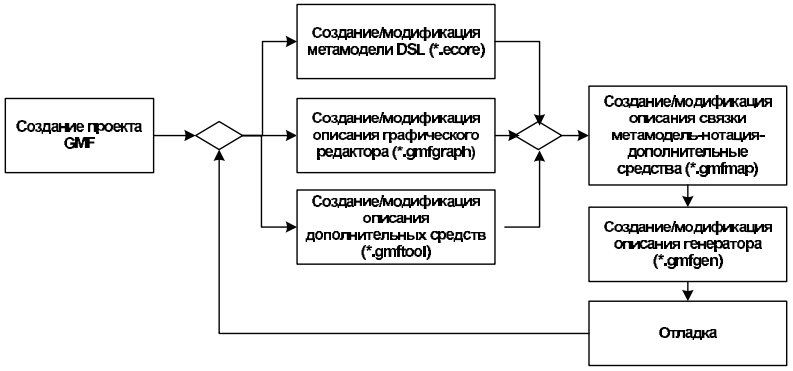

Рис. 1. Процесс разработки редактора с помощью технологии GMF

2.2. Microsoft DSL Tools

Инициатива Microsoft под названием Software Factories [9] является попыткой индустриализации процесса разработки ПО. Суть этой инициативы заключается в создании и использовании «фабрик» по созданию ПО для ускорения процесса разработки, придания ему большей гибкости и минимизации затрат. В общем случае под фабрикой понимается комплекс средств на базе расширяемой среды разработки (например, `Microsoft Visual Studio`), включающий в себя DSM-средства, инструментальную поддержку шаблонов проектирования, средства поддержки повторного использования компонентов и другие средства, облегчающие применения лучших практик для разработки определенного класса программных средств, в частности, семейств программных продуктов. В рамках данной инициативы предоставляется набор средств `Microsoft DSL Tools`, входящий в `Visual Studio SDK`, для реализации DSM-пакетов в среде `Visual Studio 2005`.

В состав платформы `Microsoft DSL Tools` входят: мастер проектов, создающий пустой проект нового DSL, средства описания и генерации моделей языков и их графических редакторов, а также средства поддержки кодогенерации в создаваемых редакторах.

Для описания модели предметной области, которая будет являться и метамоделью создаваемого языка, предлагается специальный редактор классов, позволяющий создавать классы объектов предметной области, отношения включения, наследования и ссылки, а также роли отношений с указанием множественности. На рис. 2 приводится часть диаграммы классов, описывающих предметную область «семья».

Для описания целевого графического редактора предлагается специальная секция диаграммы классов, в которой определяются графические объекты, использующиеся для отображения объектов модели предметной области, и их привязка к классам модели. Дополнительно могут быть заданы настройки палитры объектов и окна навигатора по модели.

По описанной выше модели средствами `Microsoft DSL Tools` генерируются набор реализационных классов модели предметной области, классов, реализующих палитру объектов и навигатор модели, а также XML-описание графического редактора. Полученные артефакты являются достаточными для генерации полноценного графического редактора. Новый редактор может быть открыт в ре-

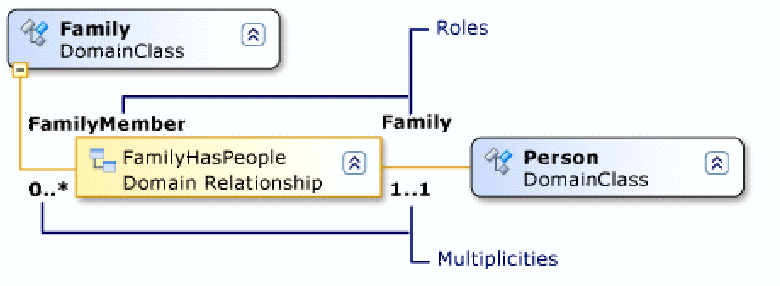

Рис. 2. Пример диаграммы классов предметной области

жиме отладки в новом экземпляре `Microsoft Visual Studio` или собран в инсталляционный пакет.

Дополнительная функциональность, такая как процедуры валидации моделей, создается программистом в ручном режиме в виде частичных классов (partial classes) .Net, расширяющих сгенерированные классы. Использование частичных классов гарантирует, что последующие изменения модели предметной области и перегенерация кода не уничтожат дополнительный код, добавленный программистом.

Процесс разработки редактора может быть цикличным (итерация цикла представлена на рис. 2): изменение модели и настроек, генерация кода, добавление кода программистом, отладка, повтор цикла.

Созданный таким образом редактор предоставляет пользователям средства создания и редактирования моделей, механизм сохранения моделей, процедуры валидации и возможность генерации различных артефактов (исходного кода, отчетов, конфигурационных файлов и т. д.) по моделям. Генерация артефактов осуществляется по шаблонам. Генерационные шаблоны создаются пользователями редактора на специальном языке разметки, содержащем директивы генерационному процессору, управляющие конструкции и включения кода на C# для извлечения данных из моделей.

2.3. MetaEdit+

Технология `MetaEdit+` финской компании MetaCase является комплексом средств, сочетающим в себе средства создания различ-

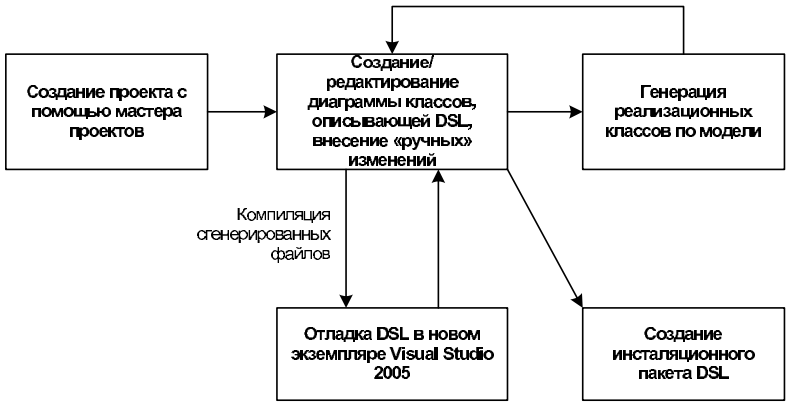

Рис. 3. Процесс разработки редактора с помощью `Microsoft DSL Tools`

ных DSL, а также соответствующие им DSM-пакеты. Программные средства технологии разделяется на следующие подсистемы (см. рис. 2, 4):

- Method Workbench — средство проектирования различных DSL;
- MetaEdit+ — многопользовательская среда, предоставляющая конечным пользователям средства для работы с DSL, как созданными с помощью Method Workbench, так и с входящими в стандартную поставку системы.

Method Workbench предлагает описывать DSL с помощью языка метамоделирования GOPPRR[19] [15]. С его помощью описываются абстрактный и конкретный синтаксис[20], а также семантика DSL.

Для создания абстрактного синтаксиса нового DSL язык GOPPRR предполагает определение типов объектов предметной обла-

[19] Graph, Object, Property, Port, Relationship, and Role.

[20] Согласно [14], мы рассматриваем синтаксис DSL в следующих измерениях: конкретный (concrete syntax) — правила изображения символов языка, из которых строятся визуальные модели; абстрактный (abstract syntax) — структура визуальных спецификаций, в рамках которой тщательно классифицированы все графические символы, определены все их атрибуты и связи друг с другом; служебный (serialization syntax) — способ хранение визуальных моделей. Семантика и прагматика имеют стандартное для языков программирования значение.

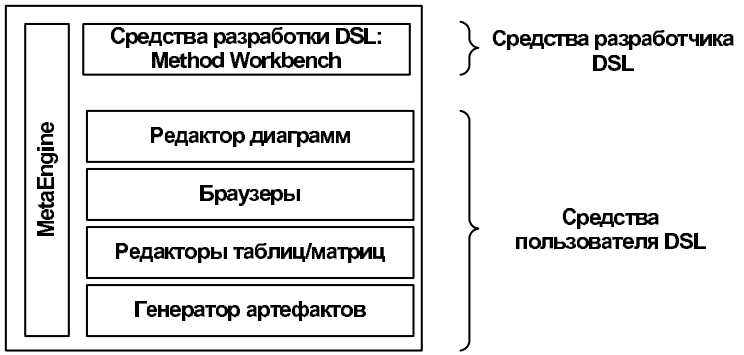

Рис. 4. Схема разделения функциональности MetaEdit+ по ролям пользователей

сти (Objects), которые визуализируются с помощью данного DSL, типов отношений (Relationships), связывающих эти объекты, а также ролей (Roles), которые могут играть объекты в отношениях. Типы объектов, связей и ролей могут иметь различные наборы свойств (Properties), а также визуальные символы, которые будут использоваться на диаграммах для их отображения. Для разных элементов языка могут задаваться также семантические и синтаксические ограничения, используя конструкцию языка GOPPRR под названием порт (Port). Эти ограничения могут описывать возможные сценарии соединения диаграммных объектов, которые также могут иметь свои наборы свойств и символы для отображений. Для связи созданных объектов и уточнения семантики языка используется понятие графа (Graph). При определении графа используются дополнительные средства, такие как связки «отношение-роль-порт-объект», указания множественности и явные семантические ограничения. Также на уровне графа могут быть заданы правила декомпозиции и разбиения (правила переходов между различными графами-нотациями для общих типов объектов и отношений) сущностей. Символы всех объектов создаются и редактируются с помощью встроенного графического редактора. Также с помощью Method Workbench могут быть отредактированы диалоговые формы элементов языка и шаблоны для генерации артефактов по моделям.

Подсистема `MetaEdit+` является настраиваемой средой под-

держки предметно-ориентированных языков, которая конфигурируется «на лету» описанием DSL, созданным с помощью Method Workbench. Среда предоставляет конечным пользователям стандартный набор средств, присущий средствам поддержки DSL, среди которых многопользовательский репозиторий для хранения информации о моделях, графический редактор, позволяющий создавать и редактировать модели, браузер репозитория, набор редакторов свойств объектов языка, средства импорта/экспорта моделей, а также генератор артефактов.

2.4. Пакет Microsoft Visio

Пакет `Microsoft Visio 2003` представляет собой средство для построения схем и диаграмм различного типа, реализуя различные средства работы с базовыми геометрическими фигурами (границы, заливка, вращение/отражение при построении и т. д.), текстовыми полями, связанными с этими фигурами, предоставляет возможность задавать поведение графических фигур (например, ограничения на изменение размеров по высоте и или ширине) и т. д. Кроме того, в состав поставки пакета входят различные типы специальных диаграмм — карт, электрических схем, планов зданий, топологии вычислительных сетей, схем бизнес-процессов, спецификаций ПО и т. д. Для каждой из этих областей существуют свои специализированные пакеты и средства, но, во-первых, они стоят дорого, во-вторых, требуют специальных (и часто немалых) усилий по освоению. Пакет `Microsoft Visio` очень известен, легок в использовании и может быть гибко настроен. Он подходит для создания небольших диаграмм в очень разных областях (и таких пользователей очень много), но для глубокого, профессионального моделирования лучше использовать специализированные средства.

Кроме этих возможностей пакет `Microsoft Visio` содержит средства для реализации новых графических языков: можно задавать новые нотации (stencils), определять графические свойства новых фигур (shapesheet tables). Среда поддерживает встроенный скриптовый язык, позволяющий создавать довольно сложные модели поведения графических объектов. Спецификация нового языка сохраняется в виде специального шаблона и подключается к списку доступных шаблонов, предлагаемых пользователю при создании нового Visio-проекта.

Строго говоря, пакет не является DSM-платформой, так как не содержит многих важных возможностей, например, средств для создания репозиториев. Однако на базе этого пакета создано очень много различных решений в силу его простоты, доступности, наличия хорошего API и средств интеграции с `Microsoft Visul Studio` (через специальную библиотеку Visio SDK). Кроме того, этот пакет можно надстроить недостающими средствами, как это предложено в [8].

2.5. Сравнение DSM-платформ

Обычно, реальная практическая задача по созданию DSM-средств диктует достаточно жесткие условия и ограничения на их средства разработки и среду эксплуатации. Например, если целевой проект (то есть те программные системы, для разработки которых предназначаются данные DSM-средства) разрабатывается на Java, то предпочтительнее использовать средства `Eclipse/GMF`, если в `Microsoft Developer Studio` — то `Microsoft DSL Tools`, если нужно средство графического проектирования вне контекста разработки ПО или на стыке (например, легкий графический пакет для инженеров, участвующих в создании программно-аппаратной системы), то здесь удобен пакет `Microsoft Visio`, позволяющий создавать DSM-средства, минимально «привязанные» к исполняемой платформе (.Net или Java) и, значит, имеющие простую схему инсталляционного процесса, низкие требования по вычислительным ресурсам. В этой ситуации можно воспользоваться также `MetaEdit+`.

Нужно отметить, что технологии `Eclipse/GMF` и `Microsoft DSL Tools` очень сильно связаны с платформами разработки — `Eclipse` и `Microsoft Visual Studio` соответственно. Особенно технология `Microsoft DSL Tools`, которая требует для работы не только многочисленных библиотек периода исполнения, подобно `Eclipse/GMF`, но также и среды разработки (`Microsoft Visual Studio`), а также является составной частью концепции разработки ПО под названием Software Factory [14]. Обе технологии — `Eclipse/GMF` и `Microsoft DSL Tools` — предоставляют богатые возможности по доводке программного кода DSM-средств «вручную» с сохранением результатов при повторной генерации. Эти технологии целесообразно использовать, если вы работаете, соответственно, в `Eclipse` или `Microsoft Visual Studio`.

Характеризуя рассмотренные выше DSM-платформы, можно выделить следующие случаи их использования, помимо платформенных предпочтений.

Если требуется создать многофункциональные средства визуального моделирования, то лучше использовать `Eclipse/GMF`. Например, с использованием этой технологии создается известный пакет `Borland Together Control Center`, который, конечно, не попадает в разряд DSM-средств, являясь мощным промышленным пакетом визуального моделирования (точнее, целым семейством таких продуктов). Но часто DSM-средства оказываются весьма трудоемки по разработке, так как требуют богатой функциональности и могут заказываться большими компаниями типа Siemens, Motorola и т. д. для внутреннего использования.

Если создается целое семейство визуальных языков и средств их поддержки, то лучше использовать технологию `MetaEdit+`. Кроме того, эта технология является наиболее зрелой в плане создания DSM-средств без программирования.

Если же нужен «легкий» графический редактор, имеющий специфическую нотацию и несложные правила поведения фигур, ту лучше использовать пакет `Microsoft Visio`. В этом случае часто оказывается, что можно обойтись только средствами `Microsoft Visio` и избежать использования библиотеки Visio SDK, а также программирования в среде .Net. Например, таким образом был создан редактор MSC-диаграмм, описанный в [4] и достаточно полно реализующий эту графическую нотацию[21].

Наконец, скажем несколько слов о степени готовности и перспективности DSM-платформ. На настоящий момент наиболее зрелыми являются платформы `Eclipse/GMF` и `MetaEdit+`, которые, к тому же, активно развиваются. Платформа `Microsoft DSL Tools` еще совсем молодая и годится пока лишь для экспериментов. Пакет `Microsoft Visio`, фактически, законсервирован как DSM-платформа и поддерживается только как графический пакет.

В таблице приведены сводные свойства описанных выше DSM-платформ.

[21] MSC (Message Sequence Chard) — стандарт комитета ITU для визуального моделирования сценариев работы телекоммуникационных систем.

Сводные свойства DSM-платформ

DSM-платформа		Eclipse GMF	Microsoft DSL Tools	MetaEdit+	MS Visio
Разработка целевого DSM-пакета	Средства описания DSL-метамодели.	EMF-модель, сериализуемая в XMI, возможен импорт аннотированных Java-интерфейсов, XML-Schema модели классов Eclipse UML2.	Диаграмма классов специального вида, создаваемая с помощью редактора VisualStudio 2005.	Создание GOPPRR-описания DSL-метамодели встроенными средствами. Грань между процессом описания DSL-метамодели и соответствующего редактора не четкая.	Нет
	Средства описания графического DSL-редактора.	Графический редактор задается в специальном формате. Возможно задание ограничений и методов валидации на OCL и Java.	XML-описание графического редактора, генерируемое по диаграмме классов.		Есть средства описания графических объектов, есть встроенный язык для описания поведения графических объектов.
	Средства описания дополнительной функциональности DSM-пакета.	Описание вспомогательных средств (палитра объектов, список действий, меню графических объектов).	Средства описания поведения навигатора модели; средства описания палитры визуальных объектов.	Есть средства для описания палитры визуальных объектов, для проектирования диалогов ввода и редактирования свойств модельных объектов, для описания декомпозиции и разбиения.	Нет
	Средства автоматизации процесса разработки целевого DSM-пакета.	Автоматическая генерация графического редактора средствами EMF; автоматическая генерация редактора диаграмм и палитры средствами GMF.	Автоматическая генерация реализационных классов метамодели языка, автоматическая генерация XML-описания графического редактора, «ручная» реализация процедур валидации моделей на основе предлагаемой архитектуры.	Автоматическая генерация метаописания языка (хранятся в репозитории) и графического редактора для работы с ним. DSM-пакет создается "на лету": MetaEdit+ динамически настраивает среду на поддержку конретного DSL, описание которого выбрано из репозитория.	Нет
Характеристики целевого DSM-пакета	Среда целевого DSM-пакета	Eclipse IDE	Microsoft Visual Studio 2005	MetaEdit+	MS Visio
	Графический DSL-редактор.	Диаграммный редактор основывается на GMF Runtime	Встроенный редактор Microsoft Visual Studio 2005	Встроенный редактор MetaEdit+	MS Visio
	Готовые решения для генерации артефактов по моделям.	Нет	Механизм генерации артефактов по шаблонам на ASP-подобном языке	Механизм генерации артефактов по шаблонам на собственном языке Report Definition Language	Создание отчетов в форматах Excel, HTML, XML по войствам фигур.
	Репозиторий	Есть (фактически, в его роли выступает EMF)	Существует богатое API для доступа к хранимым объектам, которое поддерживает транзакции, Undo/Redo и пр.	Многопользовательский репозиторий среды MetaEdit+.	Нет
	Отчуждаемость целевого DSM-пакета от DSM-платформы	Нет	Нет	Нет	Нет

3. Опыт использования DSM-платформы Microsoft Visio 2003/.Net

Мы использовали эту платформу для разработки графических средств проектирования аппаратуры семейства систем телевещания (ТВ). На основе пакета `Microsoft Visio` был создан графический редактор принципиальных схем для таких систем, реализован генератор Excel-отчетов по диаграммам, а также генератор загрузочной конфигурации ПО целевой системы. Репозиторием редактора активно пользовался не только он сам, но также и другая программная компонента — модуль описания и учета аппаратуры семейства ТВ систем. Этот проект подробно описан нами в работе [6].

Для успешной реализации этого проекта нам пришлось создать дополнительный комплекс средств для поддержки репозитория, а также создать архитектуру на основе MVC-шаблона (подробно данный комплекс средств описан в [8]). Отметим некоторые трудности, с которыми мы столкнулись при использовании пакета `Microsoft Visio`:

- невозможность использовать стандартный для `Microsoft Developer Studio` путь создания инсталляционных пакетов;
- слабая объектная ориентация модели графических фигур (Shape Sheets), в силу чего ее почти невозможно надстраивать и расширять, а также затруднительно активно использовать через открытый API (мало операций над данными);
- трудности с заданием сложных фигур: сложные фигуры создаются только через группы, у таких групп оказываются проблемы с производительностью при перерисовке; также значительные трудности с автоматическим выравниваем (layout) линий в соответствии со специфическими требованиями графических нотаций;
- трудности с созданием и изменением диаграмм из кода вне `Visio`, через API;
- трудности с заданием корректного поведения вложенных фигур — например, сложное состояние в диаграммах состояний и переходах UML, включающее в себя несколько простых, к которым извне, через границу сложного состояния, идут линии-переходы.

Данная DSM-платформа принесла нам следующие выгоды при разработке и внедрении графических средств проектирования:

- целевой DSM-пакет обладает почти полной функциональностью исходного пакета `Visio` (главное меню, многостраничные диаграммы, возможности работы с базовыми свойствами геометрических фигур, многостраничная печать и т. д.), что очень удобно;
- возможно быстро создавать демо-версии и обсуждать их с заказчиком (в частности, создать графическую нотацию и приготовить демонстрационный пример оказалось возможным всего за 2 часа);
- компактность и отчуждаемость целевого продукта — целевой инсталляционный пакет имел объем около 3-х Mb, для его установки требуются .Net Framework 2.0 (объемом примерно 20 Mb) который свободно загружается с сайта компании Microsoft, а также `Microsoft Office 2003`, включая `Microsoft Visio 2003`;
- широкая распространенность `Microsoft Visio` в академической, научной и производственной сферах, так что решения на его основе легко понимаются и принимаются.

4. Требования к платформенно-независимой DSM-методологии

Нам кажется, что методологическая поддержка DSM-подхода крайне важна — иначе он вряд ли будет широко применяться на практике. Естественно, что компании-разработчики ПО не могут владеть всей нужной информацией для выбора необходимой платформы, им трудно оценить стоимость разработки, внедрения и поддержки DSM-средств, часто страдает процедура выявления требований к таким средствам. Относительно последнего важно отметить, что требования должны быть определены предельно точно, так как любая неточность, расплывчатость, функциональность «про запас» или «для приличия» сильно повышают стоимость и способны (да еще при отсутствии опыта создания таких средств и неточных оценках) сделать проект нереальным.

Платформенная независимость методологии означает, что она не ограничивается какой-то одной платформой, одним сообществом — например, `Microsoft Visual Studio` и `DSL Tools`. В ней должна содержаться информация по всем известным DSM-платформам, вдобавок, проанализированная, надлежащим образом обобщенная и структурированная. К сожалению, на настоящий момент информацию о DSM-платформах можно найти только в специальных сообществах — форумах по `Eclipse`, `Visual Studio` и т. д.

Отметим теперь набор теоретических знаний и практических навыков, которыми должна обладать команда, разрабатывающая DSM-средство (часть таких знаний также должны быть представлены в DSM-методологии):

- хорошее понимание целей и задач визуального моделирования, его сильных и слабых сторон;
- глубокое знание UML, знакомство с такими понятиями, как classifier, ассоциации, стереотипы, профайлы; знание языка OCL, стандарта XMI и т. д.;
- владение навыками метамоделирования — т. е. умение создавать формальные спецификации визуальных языков;
- владение различными видами автоматической обработки визуальных спецификаций — валидации моделей, генерации кода по диаграммам, возвратной инженерии, отладке приложения в терминах модели и т. д. [18];
- навык работы со стандартными средствами поддержки визуального моделирования — продуктами `IBM Rational Rose`, `Borland Together Control Center` и т. д.;
- понимание специфики процесса использования визуального моделирования: особенности разработки и использования разовых и «долгоживущих» моделей [19], сохранения целостности различных UML-моделей [12], поддержки согласованности диаграмм и кода при итеративной разработке (round-trip engineering) [21] и т. д.
- наличие обширного практического навыка работы в средах программирования, в которые встроены DSM-платформы: если это `Microsoft DSL Tools`, то нужно иметь большой опыт

работы в `Microsoft Visual Studio`, если это `Eclipse/GMF` — то нужно иметь опыт программирования на Java и профессионально разбираться в среде `Ecplipse`.

Наш опыт показывает, что собрать команду, владеющую такими знаниями (эдакое соединение университетских ученых и хороших программистов), — не просто. Опыт внедрения в программную индустрию наукоемких инновационных технологий [1], в частности, формальных методов верификации и тестирования [17], заставляет искать новые подходы к популяризации DSM-подхода. Таким подходом может стать DSM-методология — ведь программисты хорошо обучаемы, главной проблемой часто оказывается наличие информации и быстрый доступ к ней. Теперь сформулируем требования к той информации, которую должна содержать DSM-методология:

- различные шаблоны (например пакет `Micrisoft DSL Tools`, снабжен многочисленными шаблонами языков); полезны различные архитектурные шаблоны целевых DSM-пакетов, различные стратегии создания кодогенераторов и т. д.
- нужна информация о разных свойств DSM-платформ — отчуждаемость от целевой платформы разработки, трудоемкость использования, диапазон функциональных характеристик целевых DSM-пакетов;
- необходимы шаблоны процессов разработки, внедрения и эксплуатации DSM-пакетов; это очень важно, поскольку часто, будучи созданными, такие решения оказываются без надлежащей поддержки и сопровождения [18]; компаниям часто трудно оценить реальные ресурсы и организационные усилия для успешной разработки и эксплуатации таких средств;
- нужна также хорошо разработанная терминология;
- необходимы описания различных успешных (и не очень успешных) проектов, в рамках которых DSM-средства создавались и использовались.

Заключение

В качестве дальнейших интересных тем исследования в рамках создания DSM-методологии выделим сравнительный анализ языков и средств описания метамоделей и графических редакторов. Интересно также понять, в каком формате может существовать DSM-методология, как эволюционизировать.

Для того чтобы наметить контуры и структуру DSM-методологии, создать ее первую версию (то есть наполнить эту структуру содержанием), необходимы дополнительные эксперименты, более глубокий опыт по разработке DSM-пакетов, а также анализ и обобщение этого опыта.

Список литературы

[1] *Баранцев А. В., Кулямин В.В., Омельченко В. А., Петренко О. Л.* Проблемы внедрения наукоемких технологий. http://www.software-testing.ru/lib/ispras/.

[2] *Гамма Э., Хельм З., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования: паттерны проектирования (пер. с англ.) СПб.: Питер, 2004. 366 с.

[3] *Дробинцев П. Д.* Интегрированная технология обеспечения качества программных продуктов с помощью верификации и тестирования. Канд. дис. СПГПУ. 2006. 238 с.

[4] *Замышляев А. Н.* Визуализация недетерминированных сценариев ПО при возвратном проектировании. Дипломная работа, СПбГУ. 2005. 34 с.

[5] *Кознов Д. В.* Визуальное моделирование компонентного ПО. Канд. дис. СПбГУ. 2000. 82 с.

[6] *Кознов Д. В., Перегудов А. Ф., Бугайченко Д. Ю.* и др. Визуальная среда проектирования систем телевизионного вещания // Наст. сборник. С. 142–168.

[7] *Кознов Д. В.* Языки визуального моделирования: проектирование и визуализация программного обеспечения. Учебное пособие. СПб.: Изд-во СПбГУ, 2004. 143 с.

[8] *Павлинов А. А., Кознов Д. В., Перегудов А. Ф.* и др. Комплекс средств для реализации предметно-ориентированных визуальных языков // Принято к печати в журнале «Вестник СПбГУ». 2007.

[9] *Парфенов В. В., Терехов А. Н.* RTST — технология программирования встроенных систем реального времени // Системная информатика. Вып. 5: Архитектурные, формальные и программные модели. Новосибирск, 1997. С. 228–256.

[10] *Brooks F.* No Silver Bullet // Information Proceeding of the IFIP 10th World Computing Conference. 1986. P. 1069–1076.

[11] *Czarnecki K., Eisenecker U. W.* Generative programming: Methods, Tools, and Applications. Addison-Wesley, 2000. 832 p.

[12] *Elaasar M., Briand L.* An Overview of UML Consistency Management // Technical Report SCE-04-18, Carleton University, Department of Systems and Computer Engineering. 2004.

[13] *France R. B., Ghosh S., Dinh-Trong T., Solberg A.* Model-driven development using UML 2.0: promises and pitfalls // Computer. Vol. 39. Issue 2. 2006. P. 59–66.

[14] *Greenfield J., Short K.* et al. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. John Wiley & Sons, 2004. 666 p.

[15] Journal of Visual Languages and Computing. Preface. No. 15. 2004. P. 207–209.

[16] *Kelly, S., Lyytinen, K., and Rossi, M.,* MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE Environment // Proc. CAiSE'96, 8th Intl. Conference on Advanced Information Systems Engineering, LNCS 1080. 1996. P. 1–21.

[17] Knight J.C., DeJong C.L., Gibble M.S., Nakano L.G., Why Are Formal Methods Not Used More Widely? // Fourth NASA Formal Methods Workshop (http://www.cs.virginia.edu/ jck/). 1997.

[18] Koznov D., Kartachev M., Zvereva V., Gagarsky R., Barsov A. Roundtrip engineering of reactive systems // Proc. 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA). 2004. P. 343–346.

[19] *Koznov D.* Visual Modeling in Software Management // Proc. 2nd International Workshop «New Models of Business: Managerial Aspects and Enabling Technology». 2002. P. 161–169.

[20] OMG. UML 2.0 Infrastructure Specification. September, 2004. http://www.omg.org.

[21] *Sendall S., Kuster J.* Taming Model Round-Trip Engineering // Proc. Workshop on Best Practices for Model-Driven Software Development (part of 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, Applications). 2004.

[22] *Sommerville I.* Software Engineering. 6th edition. Addison-Wesley, 2001. 693 p.