Лекция № 10. Семейства программных систем и DSM-подход

В этой лекции мы рассказывается о подходе к разработке ПО с помощью организации в компании семейства программных продуктов (software product line). Мы расскажем, каковы этапы создания семейства продуктов и что может являться повторно используемыми активами разработки. Далее, в лекции дается определение DSM-подхода (Domain-Specific Modeling) и обсуждаются его перспективы в контексте product line подхода. Подробно обсуждаются функциональные возможности и структура DSM-пакета. Рассказывается об основных на данный момент DSM-платформах: Eclipse/GMF, Microsoft DSL Tools, MetaEdit+, Microsoft Visio 2003.

Определение семейства программных продуктов. Многочисленные попытки организовать повторное использование различных активов разработки ПО (например, программных компонент, процедур менеджмента, создание стандартных средств разработки), привели мировое сообщество к тому, что повторное использование в программной индустрии не является столь простым делом, как в других, уже устоявшихся промышленных областях и сферах производства услуг. Стало ясно, что нужен специальный контекст, общность интересов и единое понимание целей и задач тех продуктов, в рамках которых происходит повторное использование. Возникла идея совместной разработки нескольких продуктов (product line) в рамках одной компании.

Итак, семейство программных продуктов – это набор продуктов, которые адресуются одному сегменту рынка или выполняют единую миссию. При этом продукты создаются на базе общих, повторно используемых активов, с помощью хорошо определенного метода.

Повторно используемые активы. Повторно используемые активы вовсе не обязательно являются только программными компонентами. Ими могут быть:

- *Требования.* Существенная часть требований к продуктам семейства является общей, что процесс создания требований (requirement engineering) очередной системы семейства.
- *Архитектура.* В рамках семейства продуктов создается архитектура типовой системы. Архитектура конкретной системы создается на ее основе и тем самым существенно экономятся ресурсы на проектирование.
- *Компоненты.* Общая для всех представителей семейства функциональность реализуется в виде повторно используемых программных компонент, из-за чего разработка систем значительно упрощается.
- *Различные модели.* Модели проектирования, результаты анализа производительности и т.д. также повторно используются при создании продуктов данного семейства.
- *Средства разработки.* В рамках семейства продуктов формируется общая технологическая среда разработки ПО – среды разработки (IDEs – Integrated Development Environments), системы управления базами данных (СУБД), средства поддержки планирования, тестирования, конфигурационного управления и т.д.

Кроме того, могут создаваться специальные программные средства, «заточенные» под специфику данного семейства (например, кодогенераторы по визуальным моделям).

- *Процессы.* Здесь речь идет о повторном использовании процедур процесса, каких как конфигурационное управление, проектный менеджмент, планирование, тестирование.
- *Квалификация сотрудников.* Поскольку системы, разрабатываемые в рамках семейства, похожи, а также существует общая инфраструктура разработки (технологии и средства автоматизации), то в такой ситуации легко «передвигать» работников из одного проекта в другой (например, при необходимости «усилить» какой-то проект), или после окончания одного проекта подключать разработчиков к новому проекту или вводить в уже существующий.

Процесс разработки. Итак, компания должна вложиться в создание семейства продуктов. Что это значит? То есть помимо собственно разработки целевых продуктов, необходимо потратить значительные средства (вложить инвестиции) в создание повторно используемых активов, формализовать и запустить процедуры их использования. При этом возврат инвестиций происходит не сразу, а при выпуске значительного количества продуктов. Часто бывает, что, стремясь поскорее вернуть инвестиции, компании не идут на большие расходы, но при этом оказывается, что в долгосрочной перспективе выгод от семейства оказывается меньше. Соотнесение величины инвестиций и сроков их возврата на организацию семейства зависит от конкретной ситуации.

Разработка инфраструктуры семейства (будем для сокращения говорить просто – семейства), с одной стороны, происходит итеративно (например, создается несколько продуктов, на их примере становится ясно, какие повторно используемые активы целесообразно выделить, потом они реализуются, внедряются в уже существующие и новые продукты, улучшаются и модифицируются и т.д.). С другой стороны, основные затраты на создание семейства происходят все таки в начале. Поэтому в разработке семейства можно выделить следующие этапы (см. рис. 10.1):

- анализ;
- проектирование;
- реализация.

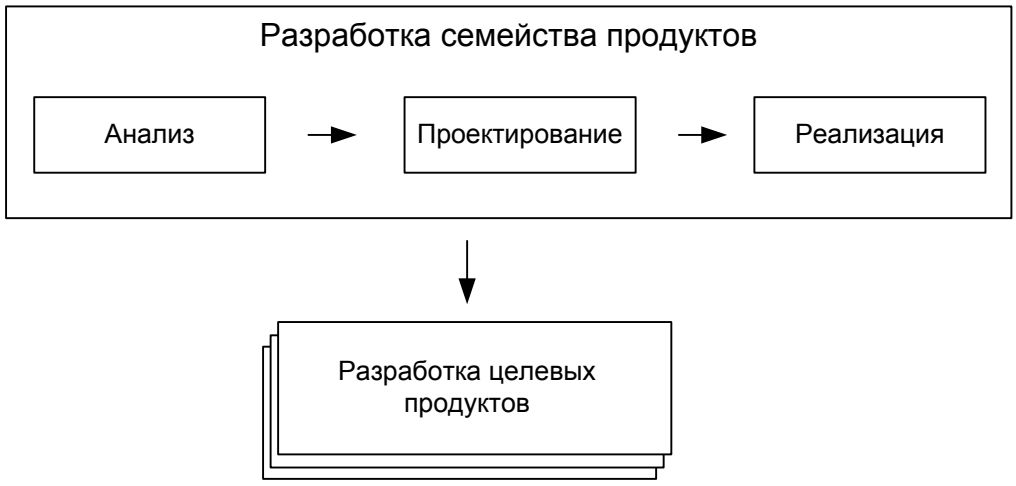

Рис. 10.1. Разработка ПО методом организации семейства продуктов.

Подробнее остановимся на этапах создания инфраструктуры семейства. Общая схема представлена на рис 10.2.

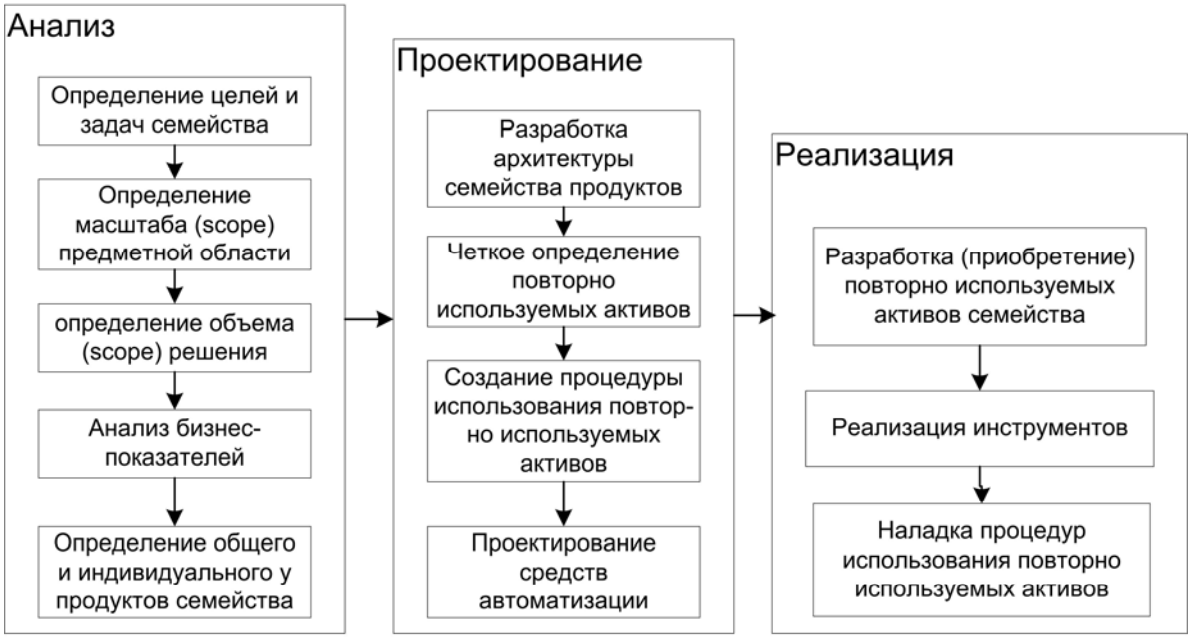

Рис. 10.2.

Анализ. Цель этого этапа заключается в том, чтобы определить, *что* за продукты будут разрабатываться в рамках семейства. Здесь очерчиваются границы домена, в самом общем виде определяются планируемые к выпуску продукты (на уровне требований к ним), оцениваются бизнес-перспективы всего рынка, попадающего в домен и планируемых к выпуску продуктов, а также инвестиционные затраты на разработку семейства продуктов. Определяется также, что может быть общего у продуктов данного домена (это прототипы повторно используемых активов), а что нужно будет разрабатывать отдельно для каждого из них.

Проектирование. Основной задачей этого этапа является определить, *как* в рамках семейства будут разрабатываться продукты. Создается архитектура семейства продуктов, в ее рамках определяются и тщательно специфицируются повторно используемые активы. Определяется также процедура создания продуктов семейства на основе повторно используемых активов, включая планы по настройке, доработке и созданию соответствующих средств автоматизации.

Реализация. Здесь реализуется архитектура семейства продуктов, то есть все, обозначенные при проектировании повторно используемые активы, а также создается и налаживается все, что необходимо для процедуры создания продуктов на основе повторно используемых активов.

DSM-подход. Организация семейства программных продуктов требует от компании создания специальной инфраструктуры, которая включает в себя средства разработки, «заточенные» на специфику систем данной предметной области. Эффективность таких средств объясняется тем, что они учитывают индивидуальную специфику конкретного семейства продуктов, а затраты окупаются, поскольку эти средства используются при разработке многих систем в компании. Трудоемкость создания таких средств для компаний-разработчиков ПО все больше снижается, поскольку на рынке появляются соответствующие технологии разработки.

В частности, именно в рамках разработки семейств программных продуктов оказывается востребованным DSM-подход (Domain Specific Modeling). Этот подход предлагает создавать специальные средства визуального моделирования (языки, методы, программные инструменты) для узких областей разработки ПО, в частности, для конкретных семейств программных продуктов. В результате, повышается эффективность использования визуального моделирования, становится возможной (не в академическом или рекламном, а именно в практическом, индустриальном смысле) автоматическая генерация программного кода по диаграммам и другие автоматизированные сервисы визуальных спецификаций.

Затраты на реализацию DSM-средств (графических редакторов, генераторов кода и пр.) снижаются в виду распространения на рынке таких технологий, как Microsoft DSL Tools, Microsoft Visio, MetaEdit+, Eclipse (EMF&GEF).

О терминологии. Дадим несколько определений, которые используются в дальнейшем изложении.

Предметная область (problem domain, domain) – это часть реального мира (люди, знания, бизнес-интересы и другие активы), объединенная в одно целое с целью удовлетворения определенных потребностей рынка. Предметная область может быть достаточно абстрактной – какое-нибудь открытое сообщество, например, сообщество разработчиков Linux[1] – так и очень определенное, с ясно очерченными границами, например, инфраструктура по разработке семейства продуктов в рамках какой-либо компании.

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).

О структуре целевого 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.

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

То, что может позволить себе, например, продукт Borland Together Control Center[2], часто непосильно для DSM-средств. Итак, DSM-пакет может включать в себя:

- *среду* – 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 многопользовательской работы с моделями, интеграцией со средствами контроля версий и т.д.
- *графический редактор* – основная рабочая область для рисования диаграмм с возможностью расположения на ней различных фигур (экземпляров графических конструкций языка), применения к этим фигурам набора операций (наполнение текстом, растягивание-сжатие, передвижение, группировка, разбиение на слои, соединять друг с другом линиями и т.д.), задания для них различных графических свойств (выбор толщины линий, цвета, свойства 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 – автоматическая 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 программного кода, по моделям (возможно и других артефактов, например, текстовых документов, тестов), например, программного кода, текстовых и табличных отчетов, 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 редакторам, автоматизирующим создание каких-либо диаграмм, так и сложным программным продуктом, поддерживающим кодогенерацию по моделям, отладку спецификаций в терминах модели, поддерживающим процедуру циклической разработки (round-trip engineering).

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. 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

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

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.

Обзор DSM-платформ: 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[3] (GMF), основная задача которой – обеспечить «мост» между двумя другими, широко используемыми и известными технологиями создания средств визуального моделирования – Eclipse Modeling Framework (EMF) и Graphical Editing Framework (GEF). Архитектура DSM-пакета строится на основе MVC-шаблона[4]. 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[5]), встраиваемого в Eclipse, и самой библиотеки GEF. Модуль OED предоставляет средства программной 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 DSM-подхода можно достичь, используя технологию GEF в паре с EMF.

Технология EMF предназначена для создания приложений, использующих структурированные модели бизнес-объектов. Внутренняя реализация моделей (исходный код для их run-time классов) производится с помощью средств генерации EMF по схеме модели, описанной средствами библиотеки Ecore[6], или импортированной из одного из

---

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

[4] MVC (Model View Controller) – это известный шаблон проектирования разделяющий уровень хранения (model) данных, уровень представления данных (view) и промежуточный уровень (controller), связывающий хранение и представление.

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

[6] Ecore – библиотека Eclipse для описания метамоделей.

следующих форматов: XMI[7]-документ (как созданный «вручную», так и сгенерированные другими средствами, например, при экспорте UML-модели из пакета IBM Rational Rose[8]), 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[9]) 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 не зависимо друг от друга, поэтому совместное их использование большого количество «ручной» работы. Для преодоления этих проблем был реализован проект Eclipse GMF, автоматизирующий создание DSM-пакетов на базе библиотек GEF/EMF.

Процесс разработки DSM-пакетов на основе GMF изображен на рис. 10.3 и состоит из следующих шагов:

1. Разработка описания метамодели языка, например, с помощью графического редактора GMF Ecore (выходной файл *.ecore).
2. Разработка описания графического редактора (выходной файл *.gmfgraph).
3. Разработка описания вспомогательных средств – палитра объектов, список действий, меню графических объектов (выходной файл *.gmftool).
4. Разработка описания связки метамодельных и графических объектов, а также вспомогательных средств, описание ограничений на OCL или Java (выходной файл *.gmfmap).
5. Создание описания генератора, его модификация (выходной файл *.gmfgen).
6. Генерация когда целевого DSM-средства, запуск отладочного экземпляра Eclipse Application и отладка.

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

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

[9] API (Application Program Interface) – это общепринятое сокращение для обозначения программного интерфейса компонент или продукта.

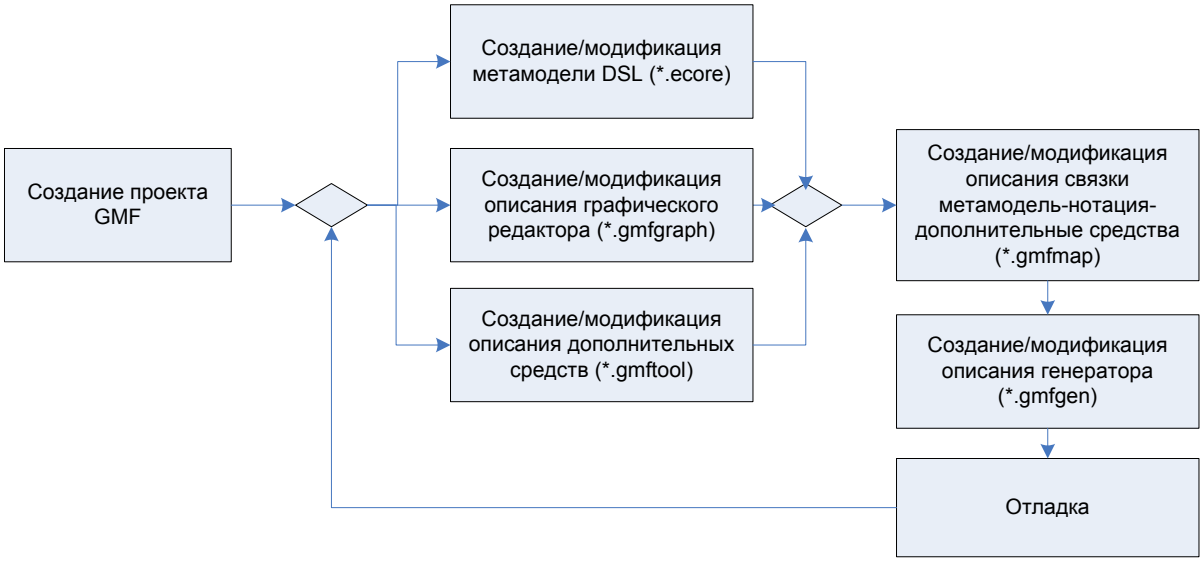

Рис. 10.3.

Обзор DSM-платформ: технология Microsoft DSL Tools. Инициатива Microsoft под названием Software Factories является попыткой индустриализации процесса разработки ПО. Суть 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. 10.4 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.

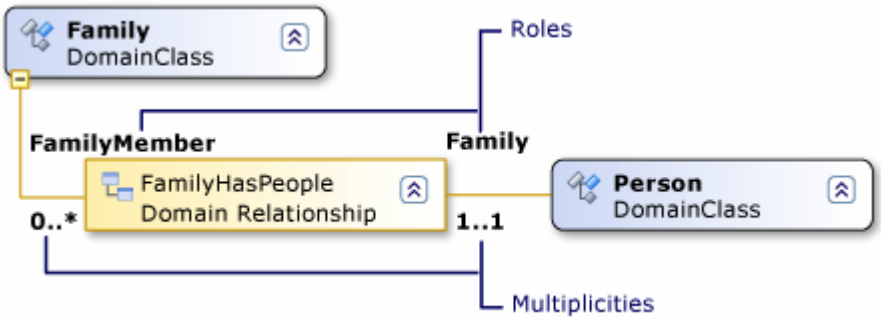

Рис. 10.4

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 режиме 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 дополнительного кода, добавленного программистом.

Процесс разработки редактора может быть цикличным (итерация цикла представлена на рис. 3): изменение модели и настроек, генерация кода, добавление кода программистом, отладка, повтор цикла.

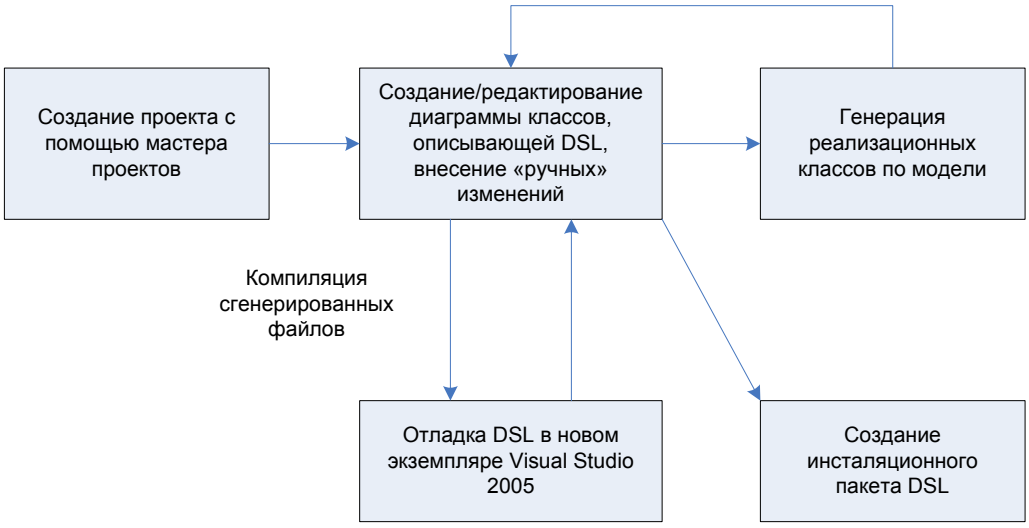

Рис. 10.5.

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.

Обзор DSM-платформ: пакет MetaEdit+. Технология MetaEdit+ финской компании MetaCase является комплексом средств, сочетающим в себе средства создания различных DSL, а также соответствующие им DSM-пакеты. Программные средства технологии разделяется на следующие подсистемы (см. рис.10.6):

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

Подсистема Method Workbench предлагает описывать DSLs с помощью языка метамоделирования GOPPRR[10]. С его помощью описываются абстрактный и конкретный синтаксис, а также семантика DSL.

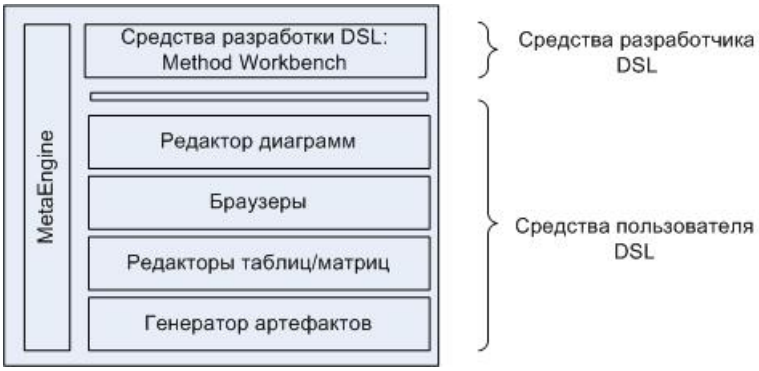

Рис. 10.6.

Для создания абстрактного синтаксиса нового DSL язык GOPPRR предполагает определение типов объектов предметной области (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 синтаксических 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 поддержки 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

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

редакторов свойств объектов языка, средства импорта/экспорта моделей, а также генератор артефактов.

Обзор DSM-платформ: пакет Microsoft Visio 2003. 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 не малых) 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, однако компания Microsoft предоставляет специальную библиотеку – Visio SDK, – предоставляющую API для создания программных модулей в рамках платформы .Net, расширяющих функциональность Microsoft Visio. Благодаря этому пакет может использоваться для создания DSM-средств, а также позволяет реализовывать надстройки, увеличивающие его возможности как DSM-платформы, что и предлагается в данной статье.

Основная литература

1. Jack Greenfield, Keith Short et al. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. – John Wiley & Sons. 2004. – 666 p.
2. Павлинов А., Кознов Д., Перегудов, Бугайченко Д., Казакова А., Чернятчик Р., Фесенко Т., Иванов А. О средствах разработки проблемно-ориентированных визуальных языков // Готовится к печати в сборнике «Системное программирование» – вып. 2, 2006 г., Изд-во СПб ун-та.

Дополнительная литература

3. A Framework for Software Product Line Practice. Version 4.2. http://www.sei.cmu.edu/productlines/framework.html
4. J.-P. Tolvanen, S. Kelly, J. Gray, K. Lyytinen. – Proceedings of OOPSLA workshop on Domain-Specific Visual Languages, Tampa Bay, Florida, USA; Technical Reports, TR-26, University of Jyvaskyla, Finland, 2001. – 101 p.
5. Journal of Visual Languages and Computing. 15 (2004). Preface. – P. 207–209.
6. K.Czarnecki, U.W.Eisenecker. Generative programming: Methods, Tools, and Applications. – Addison-Wesley 2000. – 832 p.
7. Clements, P. & Northrop, L. Software Product Lines: Practices and Patterns. – Boston, MA: Addison-Wesley, 2002.