Поддержка концептуального моделирования при разработке визуальных языков с использованием Microsoft DSL TOOLS

Д. В. Кознов
dkoznov@yandex.ru

А. Н. Иванов
iw@tercom.ru

А. И. Мишкинис
druid-h6@mail.ru

Я. И. Залевский
nexus-kun@mail.ru

Предметно-ориентированное визуальное моделирование (Domain Specific Modeling) — это направление, развивающее и применяющее на практике средства реализации новых визуальных языков. В данной работе мы представляем решение, основанное на продуктах Microsoft Visio, Microsoft DSL TOOLS, Microsoft Word и поддерживающее концептуальное моделирование визуальных языков. Предложенное решение дает возможность создавать модели языков в Visio, которые не содержат реализационных деталей, и которые можно легко обсуждать с широким кругом специалистов. По концептуальной модели сгенерировать реализационную модель в DSL TOOLS, а также каркас документации нового языка в Word. Для поддержания циклической разработки (round-trip engineering) визуального языка реализован механизм переноса изменений без перегенерации из концептуальной модели в реализационную и обратно, а также из концептуальной модели в документацию.

Введение

При всей несомненной полезности универсального языка визуального моделирования UML [13] существуют многочисленные проблемы его применения на практике — сложность, нетривиальность связей между различными видами диаграмм, избыточность и одновременно недостаточность выразительных средств (особенно, когда речь заходит о связи с конкретным языком программирования), трудности моделирования поведения систем [9, 14]. В связи с этим активно развивается предметно-ориентированное визуальное моделирование (Domain Specific Modeling, DSM) — подход, ориентированный на использование специфических диаграмм, удобных именно в данном контексте (в данном коллективе, в данном проекте, данной узкой области и т. д.), допускающий возможность создания новых нотаций прямо по ходу разработки. Мартин Фаулер, известный эксперт и популяризатор UML, рекомендовал смело использовать собственные виды диаграмм, если UML по той или иной причине не устраивает разработчиков [6]. В настоящее время существуют и активно развиваются технологии для спецификации новых графических языков и графических редакторов для них — Eclipse GMF [19], Microsoft DSL TOOLS [16], MetaEdit+ [11], Microsoft Visio [17] и др.

Центральным информационным объектом при разработке нового визуального языка является его метамодель, которая фактически бывает двух видов: 1) абстрактная или концептуальная — см., например, метамодель языка UML [13], и 2) реализационная, задающая схему репозитория CASE-пакета, «вход» для автоматической генерации графического редактора в технологиях типа Microsoft DSL TOOLS и т. д. На концептуальной модели задаются основные сущности предметной области и связи между ними и в то же время отсутствуют технические и реализационные детали, необходимые для эффективной программной реализации языка. Эту модель удобно обсуждать с широким кругом специалистов, уточнять и расширять новый язык. Реализационная же модель может генерироваться по концептуальной и уже дополняться всеми необходимыми реализационными деталями. При этом также оказывается полезной текстовая документация нового языка. Все это позволяет максимально вовлечь в разработку языка тех, для кого язык предназначается, что, в свою очередь, обеспечивает качество языка, а также успех последующего внедрения соответствующих средств его

программной поддержки. Более подробно процесс разработки предметно-ориентированных языков обсуждается в [4].

Концептуальное моделирование активно используется при моделировании схем баз данных [3]. Место концептуального моделирования в разработке DSM-решений обозначено в работе [4]. Концептуальное моделирование поддерживается в технологии Eclipse GMF, однако другое известное средство разработки визуальных средств — Microsoft DSL TOOLS — не имеет таких средств.

В данной работе предложено решение по поддержке концептуального моделирования для Microsoft DSL TOOLS на базе продукта Microsoft Visio. Реализован специальный редактор для создания концептуальных моделей на базе Visio, а также генерация реализованной модели в DSL TOOLS. Реализована и обратная генерация на случай, если разработка DSM-решения начинается с DSL TOOLS. Реализована также генерация каркаса документации визуального языка из Visio в Word. Для поддержки итеративной разработки обоих моделей и документации мы реализовали механизм циклической разработки (round-trip mechanism далее — RT-механизм), с помощью которого можно распространять изменения по моделям без полной перегенерации, тем самым сохраняя сделанные в них ранее «ручные» изменения. Наш RT-механизм отличается от подобного в технологии Eclipse GMF, где также поддержано концептуальное моделирование, так как в последней все происходит фактически в рамках одного продукта, а мы интегрируем три разных продукта. В работе приведен небольшой пример концептуальной и реализационной моделей для языка по разработке оконных приложений для мобильных телефонов.

1. Обзор

1.1. Визуальное моделирование

Визуальное моделирование (visual modeling) является подходом, который применяется для визуализации ПО и/или его предметной области, основываясь на графовой метафоре визуализации, и предполагает визуализацию моделируемого объекта с разных точек зрения, применяясь в различных видах деятельности по разработке и эволюции ПО (например, при анализе, проектировании, реинжиниринге, документировании) [3]. Язык UML, развиваемый международным сообществом с 1997 года, стандартизует различные графи-

ческие нотации, которые удобны для визуализации разного вида ПО и различных его аспектов. Так как ПО является невидимым [3, 7], поэтому необходима общая договоренность о том, как его визуально представлять при разработке, чтобы визуальные модели могли понимать не только их создатели. Опыт других инженерных областей — строительства, машиностроения, электротехники и др. — показывает полезность схематичных изображений сложных инженерных объектов (подробнее о роли чертежей в инженерных науках см. монографию [1]).

1.2. Предметно-ориентированное визуальное моделирование

UML-средства визуального моделирования (техники, методы и программные средства) часто требуют значительной адаптации и «подгонки» при практическом использовании в промышленных проектах: контекст использования становится более определенным, общие идеи и подходы можно детализировать, формируя требования к удобному и адекватному инструментарию. Во многих случаях эту задачу удобно решать, используя лишь общие идеи и базовые принципы стандартных средств, не пытаясь их настроить, а создавать свои собственные. В этом и заключается суть DSM-подхода. Дальнейшие детали и обоснования подхода, а также обзор технологий его поддержки можно найти, например, в работах [3, 10]. Ниже представлен ряд определений, используемых в данной статье.

Предметная область (problem domain, domain) — это часть реального мира (люди, знания, бизнес-интересы и проч.), объединенная в одно целое для удовлетворения определенных потребностей рынка [8]. Предметная область может быть как достаточно абстрактной — какое-нибудь сообщество людей, например сообщество разработчиков Linux, — так и очень определенной, с ясно очерченными границами — например какой-нибудь проект по разработке ПО. В данной работе будут рассмотрены предметные области, целиком входящие в какую-либо компанию по разработке программного обеспечения: линейка программных продуктов (product lines), процесс разработки многоверсионного и, соответственно, «долгоживущего» продукта, активы стандартного процесса компании, созданного в компании в рамках внедрения CMM (Defined Process), и т. д. [3, 4].

Предметно-ориентированный язык (Domain Specific Language,

DSL) — это язык, который создается для использования в рамках некоторой предметной области при решении задач именно этой области. Мы будем рассматривать только визуальные предметно-ориентированные языки, используя для их обозначения аббревиатуру DSL.

DSM-решение — это конкретный DSL, метод его использования, а также соответствующие средства инструментальной поддержки, внедренные в конкретный процесс разработки ПО [4].

Метамодель — это формально описанная структура предметной области, ее основные понятия, атрибуты, связи. Также метамодель оказывается основой нового DSL, определяя его конструкции и их взаимосвязи [4].

Концептуальная модель — это метамодель, созданная для разработки будущего DSL и предназначенная для обсуждения DSL с экспертами, будущими пользователями DSL, разработчиками, менеджерами и т. д. [4].

Реализационная модель — это метамодель DSL, предназначенная для автоматической генерации графического редактора. Она оказывается «хребтом» ПО, создаваемого для поддержки нового DSL, в этой метамодели, по сравнению с концептуальной моделью появляется большое количество реализационных деталей [4].

Циклическая разработка (round-trip engineering) — автоматическое поддержание целостности набора изменяемых активов при разработке ПО [15].

RT-механизм — реализация циклической разработки в рамках определенной практической задачи.

1.3. Продукт Microsoft DSL TOOLS

Данный продукт является средой для реализации новых визуальных языков. Его центральным инструментом является редактор метамодели (в наших терминах — реализационной модели), с помощью которой задается метамодель будущего языка. Продукт позволяет автоматически сгенерировать по этой метамодели графический редактор для нового языка в виде набора классов на языке C#. Microsoft DSL TOOLS встроен в среду разработки Microsoft Visual Studio и предназначен для создания графических редакторов, которые также встроены в эту же среду. Более подробно возможности Microsoft DSL TOOLS описаны в работах [3, 5].

1.4. Продукт Microsoft Visio

Данный продукт является средством для разработки различных схем и диаграмм в разнообразных предметных областях — в строительстве, электротехнике, при разработке ПО и т. д. В отличие от таких мощных профессиональных чертежных систем, как, например, AutoCAD [20], данный продукт более «легковесен» и является полупрофессиональным, но зато он обеспечивает нужды большого количества пользователей из разных предметных областей и пользуется успехом на рынке. В составе продукта имеется множество специализированных редакторов, в том числе и UML-редактор. Еще одна важная особенность этого пакета — он является средой для разработки новых графических редакторов: от простых, в разработке которых используются лишь средства настройки палитры и фигур, до достаточно сложных, создаваемых на языке C# с использованием открытого программного интерфейса Visio. Можно по-разному оформлять целевые графические редакторы, созданные на основе Visio — как надстройку к главному пакету или как отдельную ActiveX-компоненту, которую можно встраивать в различные продукты. Более подробно возможности Visio по разработке программных средств поддержки предметно-ориентированных языков описаны в работе [5].

1.5. Концептуальное моделирование в Eclipse GMF

В рамках известного OpenSource проекта Eclipse по созданию многофункциональной среды разработки ПО разрабатывается технология GMF (Graphical Modeling Framework) для реализации DSM-средств [19]. В отличие от Microsoft DSL TOOLS, поддерживающего единственную метамодель, в GMF поддерживается несколько метамоделей, которые на самом деле являются различными частями одной общей: доменная модель (Domain Model) позволяет задать метамодель предметной области, графическая модель (Graphical Definition) предназначается для описания графической нотации нового языка; модель инструментов (Tool Definition) позволяет задать палитру графических объектов, список действий, меню графических объектов и проч.; в модели соответствия (Mapping Model) задается связь предыдущих моделей, а также используемое именно в этом редакторе их подмножество; модель генератора

(Generator Model) генерируется по модели соответствия и позволяет задать дополнительные реализационные детали, необходимые для автоматической генерации графического редактора. Доменная модель является в GMF аналогом нашей концептуальной модели. Она позволяет задать все основные сущности предметной области языка и связи между ними и не содержит реализационной информации. Соответствие между всеми моделями поддерживается в силу наличия модели соответствия, а также обеспечивается общим репозиторием всех моделей. В нашем случае используются три разных программных средства, общий репозиторий отсутствует, задача синхронизации моделей решается другими средствами. Кроме того, в GMF отсутствуют средства документирования визуального языка, подобные созданным нами. Также необходимо отметить, что для того чтобы воспользоваться доменной моделью, необходимо установить на компьютер и Eclipse и GMF, что может оказаться неудобным для аналитиков, — ведь они не занимаются программированием. В нашем решении разработка концептуальной модели требует лишь наличия на компьютере Microsoft Visio. Фактически метамодель Microsoft DSL TOOLS соединяет в себе возможности многочисленных моделей GMF, вследствие чего и оказывается очень неудобной для обсуждения и изменений языка. Поэтому появляется необходимость в концептуальной модели.

2. Процесс разработки DSM-решения и концептуальное моделирование

В этом разделе, следуя работе [4], мы коротко изложим модель процесса разработки DSM-решения для случая, когда эта разработка является внутренней для некоторой IT-компании, которой понадобился новый визуальный инструментарий (язык, графический редактор, генератор кода и проч.) для решения собственных специфических задач. Эта модель основана на модели процессов MSF 3.1 [12]. Использование в качестве основы именно этой версии MSF, а не более поздних версий 4.x целесообразно, так как версии 4.x являются фактически шаблонами к продукту Microsoft под названием MS Team Faundation Server и могут быть названы методологией разработки ПО (как версии 3.x и ниже) с большой натяжкой. Рассматриваемая модель, конечно же, не может быть единственно возможной, но она помогает яснее увидеть важность концептуального моделирования при разработке и сопровождении

DSM-решений. Предложенная модель процесса имеет следующие фазы — выработка концепции, планирование, разработка, стабилизация, внедрение, эксплуатация и сопровождение (см. рис. 1). Рассмотрим эти фазы подробнее.

2.1. Выработка концепции

Здесь происходит определение рамок проекта, что имеет первостепенное значение ввиду изменчивости требований к DSM-решению. Необходимо ясно сформулировать ответ на вопрос: что мы создаем? И этот ответ не должен меняться, несмотря на уточнение, добавление и переосмысление требований. «Ползучесть» рамок приводит, в частности, к реализации большого количества демо-функциональности, которая не может быть рабочим инструментом. Важным также является точно определить заказчика DSM-решения. В обычном IT-проекте, следуя MSF, заказчик имеет ряд обязательств. В нашем случае он также отвечает за устойчивость финансирования проекта и за поддержание открытых и доброжелательных отношений между разработчиками и пользователями. Важно, чтобы у него были административные полномочия для того, чтобы влиять и на тех, и на других. Лучше всего, когда заказчик лично заинтересован в проекте, например желает использовать успех проекта для роста по служебной лестнице. Команда и инфраструктура внутреннего проекта также должны быть тщательно определены. Ведь все участники имеют свои рабочие места в разных частях офисов компании и часто — нагрузку, которую с них никто не снимет, несмотря на участие в проекте по разработке DSM-решения. Руководитель проекта должен понять, насколько реально работать с такими людьми, учитывая все особенности их занятости и коммуникаций с ними. Наличие актуальной концептуальной модели, а также документации содействует сохранению рамок проекта. Кроме того, отдельные фрагменты концептуальной модели могут использоваться для многочисленных презентаций — будущим пользователям, начальству и т. д., часто проводимых вначале для обоснования проекта, убеждения нужных людей в его полезности и т. д. Однако тут необходимо понимать, что даже концептуальная модель является слишком сложной для внешних по отношению к проекту людей, поэтому ее фрагменты, используемые для таких презентаций, должны быть специально адаптированы.

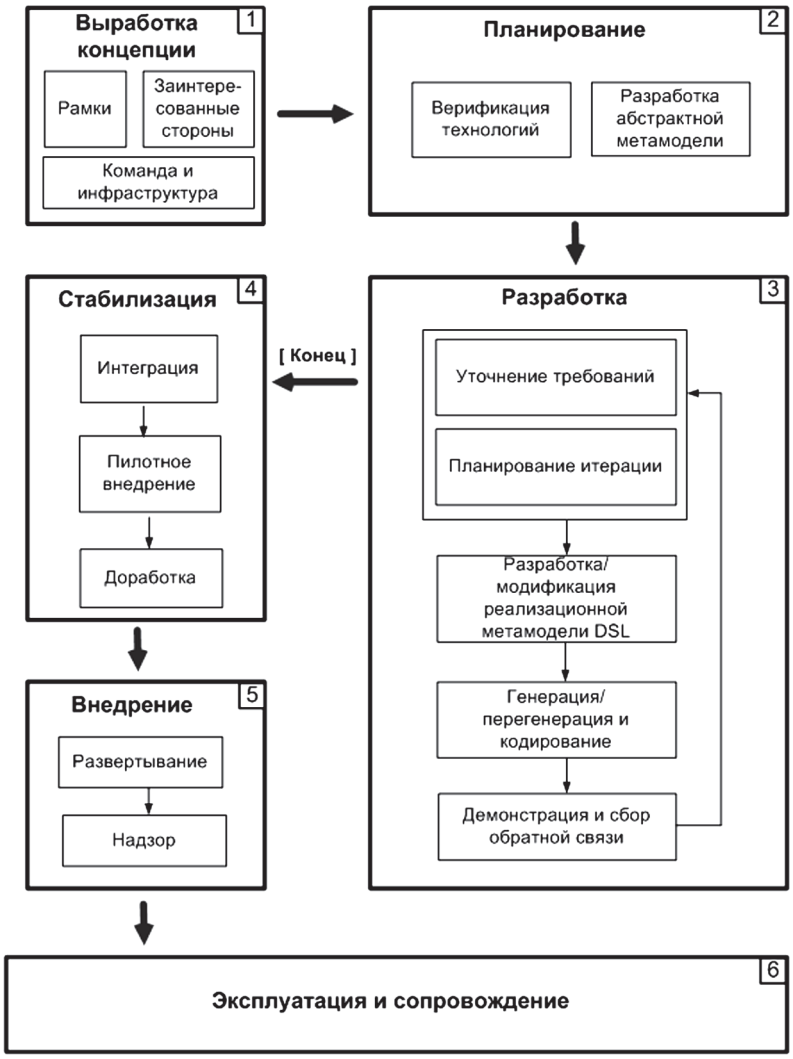

Рис. 1. Модель процесса разработки DSM-решения.

2.2. Планирование

Здесь происходит верификация технологий, выбираемых для реализации DSM-решения. Эти технологии должны быть перспективными с точки зрения развития и сопровождения их производителем, чтобы DSM-решение через некоторое время не превратилось в legacy. Здесь также закладываются основы метамодели DSL. Именно на этой фазе разрабатывается концептуальная модель, активно обсуждаясь со всеми заинтересованными сторонами.

2.3. Разработка

Эта фаза состоит из многократно повторяемой типовой итерации. Итеративная разработка DSM-решения является основным способом снятия рисков управления требованиями. Итерация имеет следующую структуру. Сначала, вместе с пользователями и заказчиком решения, производится уточнение требований и концептуальной модели на основе демонстрации результатов предыдущей итерации. Дальше происходит планирование данной итерации. После этого меняется реализационная модель — как по изменениям концептуальной, с использованием RT-механизма, так и для реализации новой функциональности. Далее вносятся изменения в реализацию (путем перегенерации и используя RT-механизм, если используемые средства это позволяют, а также выполняется разработка кода, который не может быть сгенерирован), кроме того, выполняется тестирование. После этого производится демонстрация результатов, а также сбор обратной связи, что важно для снятия риска разработать пакет, который неудобен и/или не отвечает интересам пользователей.

2.4. Стабилизация

Часто DSM-решения имеют специфические требования по интеграции и/или по окружению, в котором они и должны/могут функционировать, поэтому создание и тестирование автономных инсталляционных пакетов является важной задачей. Далее, важным является пилотное внедрение решения — пользователи, наконец, могут попробовать внедрить его в какой-нибудь некритический производственный проект. При этом они, как правило, находят много ошибок, а также могут сформулировать требования к доработке решения. По результатам пилотного внедрения происходит доработ-

ка-исправление найденных ошибок и реализация дополнительной функциональности. При пилотном внедрении важно предоставить пользователям документацию по DSM-решению. Концептуальная модель и документ, ее описывающий, могут быть составной частью такой документации. Здесь, правда, необходимо отметить, что не все пользователи DSM-решения смогут разобраться в концептуальной модели. Опыт показывает, что она все еще достаточно сложна. Тут необходимы дополнительные пояснения разработчиков, а также другие документы. Идеально «перевести» концептуальную модель в текстовое описание языка, тогда пользователи получают что-то наподобие Reference Manual — т. е. справочник по новому языку.

2.5. Внедрение

В рамках этой фазы созданное DSM-решение разворачивается на пользовательских компьютерах. Если системное обслуживание решения требует дополнительных усилий, то разработчики решения передают системным администраторам необходимую информацию. Тут же происходит и обучение, если оно необходимо. Однако мы надеемся на самообучение пользователей, так как они являются программистами и часть из них участвовала в создании решения. Тем не менее ответы на вопросы, проведение презентаций или лекций могут понадобиться. Первое время после внедрения необходим надзор за решением: разработчики должны быть готовы к разным неожиданностям главным образом в виде сложных ошибок, вдруг возникших и парализовавших использование решения. В этом случае разработчики должны оперативно исправить ошибки и выдать пользователям рабочую версию решения. Здесь важна оперативность, чтобы был нанесен минимальный ущерб производственному процессу. Время, в течение которого команда разработчиков еще получает зарплату за участие в проекте, определяется совместно руководителем проекта и заказчиком. После того как они решат, что решение стало достаточно стабильным, команда сокращается до минимума, необходимого для сопровождения и поддержки решения. (О полезности концептуальной модели и ее текстового описания см. выше.)

2.6. Эксплуатация и сопровождение

Здесь ключевым фактором является постоянно действующая команда, которой может быть и один человек — главный автор

DSM-решения, которому придаются, по необходимости, дополнительные ресурсы. Важно, чтобы DSM-решение постоянно «жило»: исправлялись ошибки, реализовывалась новая функциональность и, что особенно важно, DSM-решение должно своевременно переноситься под новые платформы, чтобы не превращаться в legacy. Сопровождение лучше организовать как последовательность новых версий, не допуская замены отдельных бинарных файлов с высылкой обновлений по электронной почте. С другой стороны, тут важно не переусердствовать в формализме — связь авторов и пользователей должна быть живой, открытой, не обременительной ни для одной из сторон. Здесь также важно уделить особое внимание поддержке в актуальном состоянии документации по решению. Иначе, так как свои делают работу для своих, и все это длится годами, а люди постепенно меняются, то в итоге знание о решении рассеивается. Наличие концептуальной модели и документации, автоматически синхронизируемых с реализационной моделью, существенно помогает в документировании DSM-решений. Иначе даже самим авторам через некоторое время бывает трудно вспомнить, зачем в метамодель был введен тот или иной класс, какую информацию отражает та или иная ассоциация.

3. Описание решения

В целом задача нашего решения по поддержанию концептуального моделирования уже описана. Опишем теперь роли каждого из задействованных в решении продукта — Microsoft Visio, Microsoft DSL TOOLS и Microsoft Word. Кратко опишем функциональность решения, разбитую по этим продуктам, а после этого рассмотрим взаимодействие программных инструментов.

3.1. Microsoft Visio

Пользуясь средствами расширения Visio, мы создали свой графический редактор для разработки концептуальных моделей визуальных языков. Этот редактор реализует упрощенную нотацию диаграмм классов UML — классы с атрибутами, ассоциации с именами, множественностью, именами ролей и агрегированием, а также наследование. При этом нашей задачей было обеспечить возможность для создания красивых диаграмм, предоставив максимум свобод пользователю и сняв различные ограничения по рас-

положению и размерам графических элементов. В частности, для ассоциации мы реализовали возможность независимого расположения от ее линии имени ассоциации, имен ее ролей и значений множественности, обеспечили также возможность менять размеры прямоугольника класса независимо от размеров текста внутри него, поддержали возможность соединения ассоциаций с произвольной точкой границы прямоугольника-класса. Отметим также, что все именованные сущности концептуальной модели имеют в нашем решении два типа имен — логические, допускающие пробелы и могущие быть русскоязычными, и физические, которые являются C#-идентификаторами. Отсутствие этих возможностей, например, в UML-редакторе, входящем в поставку Visio, сильно сужало возможности по разработке компактных диаграмм, которые удобно обсуждать с различными специалистами. На рис. 2 представлен пример концептуальной модели для небольшого языка проектирования пользовательских интерфейсов приложений для мобильных телефонов, выполненной в нашем редакторе.

Каждая программа, созданная в соответствии с этой метамоделью, включает в себя произвольное количество окон и строго один специальный класс для загрузки/выгрузки приложения. Каждое окно и класс для загрузки/выгрузки имеют строго один входной порт. Количество выходных портов у них может отличаться. Входные/выходные порты необходимы для задания потока управления от одного окна другому. Связь входного и выходного портов задается ассоциацией, соединяющей класс «Входной порт» с классом «Выходной порт». Далее, все окна делятся на две группы: «Окно загрузки», «Окно Да/Нет» и «Окно-сообщение» имеют фиксированное количество выходных портов (1, 2 и 1 соответственно), а «Список» и «Произвольное окно» — произвольное количество выходных портов (но больше единицы!).

3.2. Microsoft DSL TOOLS

Нотация реализационной модели существенно богаче нотации концептуальной модели. Прежде всего, в MS DSL TOOLS имеются классы разных видов — задающие сущности и отношения предметной области (доменные классы, domain classes) и классы-представления, определяющие внешний вид элементов предметной области на диаграммах. Есть также специальные отношения, позволяющие сопоставить доменным классам, их атрибутам и ассоциациям клас-

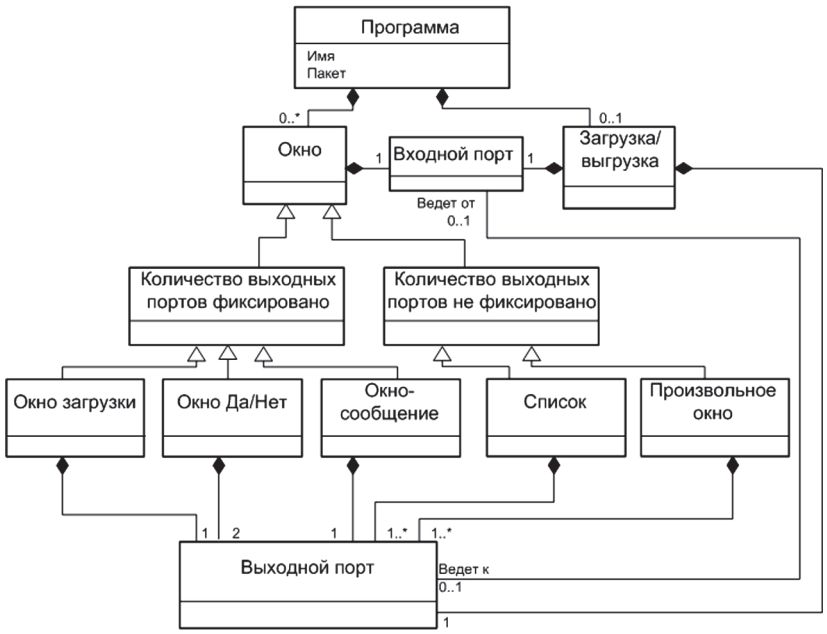

Рис. 2. Пример концептуальной модели.

сы-представления. Классы концептуальной модели генерируются в доменные классы, ассоциации и наследования — в таковые же для доменных классов, а атрибуты — в доменные свойства (domain properties) доменных классов. Кроме доменных свойств у доменных классов есть большое количество других видов свойств, нужных для задания параметров генерации графического редактора. Более того, у доменных свойств есть большое количество различных атрибутов, которые, кроме имени, также не имеют прообраза в концептуальной модели. Более подробно о графической нотации Microsoft DSL TOOLS можно узнать в работе [3]. На рис. 3 представлен фрагмент реализационной модели, сгенерированной по концептуальной модели с рис. 2. Очевидно, что реализационная модель более громоздкая (на рис. 3 поместилась лишь небольшая часть этой модели), содержит много реализационных деталей и существенно менее удобна для обсуждения. Результирующий редактор, созданный на основе этой модели в Microsoft DSL TOOLS, представлен на рис. 4.

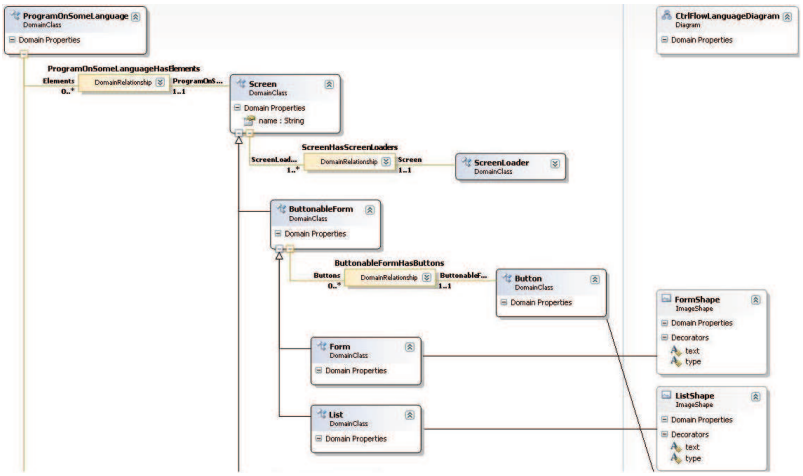

Рис. 3. Пример реализационной модели.

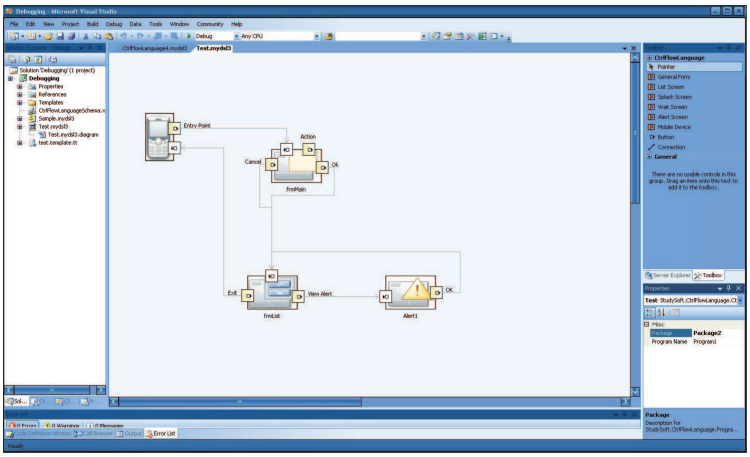

Рис. 4. Пример сгенерированного с помощью Microsoft DSL TOOLS графического редактора.

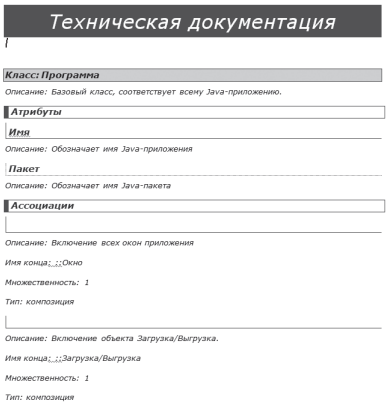

Рис. 5. Фрагмент сгенерированной документации.

3.3. Microsoft Word

Продукт Microsoft Word всем хорошо известен и не нуждается в представлении. Мы используем его для разработки текстового описания нового языка — всех его классов, атрибутов и ассоциаций. Документ, описывающий визуальный язык, состоит из блоков, описывающих классы, заголовка документа и произвольного текста в начале документа, между блоками и в конце документа. Весь этот сопутствующий текст сохраняется при изменениях концептуальной модели. Фрагмент документа с описанием класса «Программа» из приведенного выше примера представлен на рис. 5.

Подобные блоки автоматически генерируются нашим программным продуктом для всех классов, присутствующих в концептуальной модели.

3.4. Взаимодействие программных инструментов

Схема взаимодействия программных средств, интегрируемых нашим решением, — Visio, DSL TOOLS, Word, — представлена на рис. 6. На этом рисунке сплошными стрелками обозначена возможность генерации одной спецификации по другой. Пунктирные стрелки показывают маршруты переноса изменений, и в данном случае они совпадают со сплошными стрелками.

Изменения в визуальном языке лучше вносить в концептуаль-

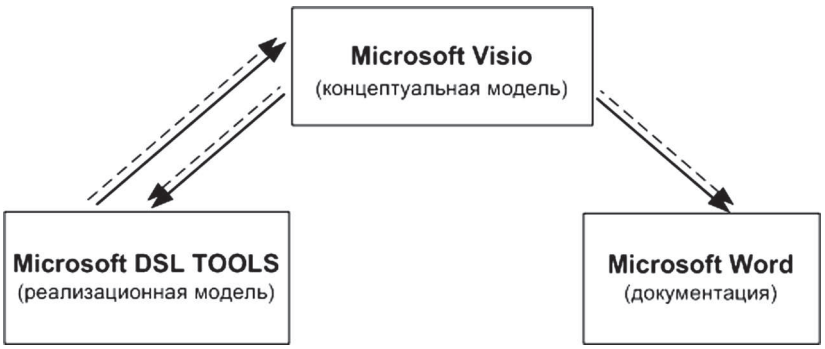

Рис. 6. Схема взаимодействия продуктов Visio, DSL TOOLS, Word.

ную модель, чтобы ее и дальше можно было использовать для обсуждения и улучшения данного языка. Далее эти изменения должны попасть в реализационную модель и после этого — в программные средства поддержки нового языка. Должна также измениться и документация. Но при реализации программных средств может измениться реализационная модель в той ее части, которая пересекается с концептуальной моделью. Например, может оказаться, что некоторые конструкции неудобны для реализации и требуют переделки. Вся эта переделка осуществляется в Microsoft DSL TOOLS, и соответственно изменяется концептуальная модель, а по ней и документация, чтобы последние не потеряли актуальность и могли использоваться в проекте и дальше. Мы не допускаем изменения концептуальной модели через изменение документации — на рис. 6 это отражено как отсутствие пары стрелок от прямоугольника Microsoft Word к прямоугольнику Microsoft Visio. Но это является вполне естественным решением. С другой стороны, при изменении концептуальной модели сохраняется весь текст документации (кроме описания удаленных элементов) и тот текст, который расположен вне таблиц с описанием элементов модели, т. е. является произвольным текстом документа (например, его название, авторы, номер версии, какие-то комментарии после блока описания того или иного класса). Мы также поддерживаем синхронизацию имен элементов концептуальной модели в тексте документации, которые могут быть изменены пользователем в концептуальной модели. Таким образом, при изменении какого-либо одного из этих трех ар-

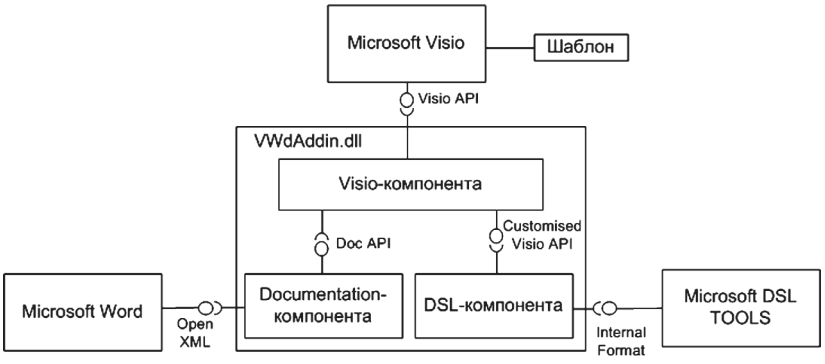

Рис. 7. Архитектура решения.

тефактов остальные два нужно синхронизировать, чтобы они не потеряли актуальность. Опыт показывает, что «вручную» данная работа обычно не выполняется регулярно — созданная на первых этапах концептуальная модель быстро теряет актуальность, так как последующие изменения вносятся «напрямую» в реализационную модель, документация также оказывается неактуальной, и в итоге вся разработка сосредотачивается вокруг реализационной модели.

4. Архитектура и состав пакета, особенности реализации

4.1. Архитектура

Мы реализовали наше решение на языке C#. Кроме того, для редактора концептуальной модели с помощью механизма шаблонов Visio нами были заданы специальные настройки, с помощью которых мы определили палитру графических элементов нашего редактора, а также задали необходимые им параметры — внешний вид, правила растяжения по высоте, ширине, точки соединения линий с границами и проч. Архитектура решения представлена на рис. 7.

Откомпилированный C#-код решения содержится в файле VWdAddin.dll. Связь с Microsoft Visio осуществляется через программный интерфейс этого продукта — Visio API. Связь с продуктом Microsoft Word мы производим через обработку файлов до-

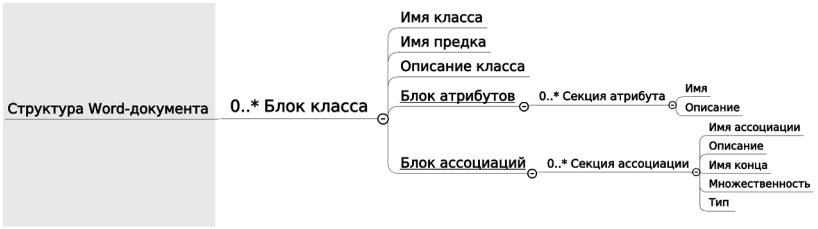

Рис. 8. Структура документа в Microsoft Word.

кументации, использующих в качестве формата хранение Open XML — открытый формат для представления Word-документов, появившийся в версии Microsoft Word 2007 [18]. Доступ к продукту Microsoft DSL TOOLS, подобно доступу к Word, мы осуществляем через обработку файлов с DSL TOOLS спецификациями (содержащими реализационную модель) — на рис. 7 этот интерфейс мы назвали InternalFormat.

Теперь рассмотрим подробнее составные части VWdAddin.dll. Visio-компонента является программной реализацией редактора концептуальной модели и содержит все реализованные нами диалоги нашего Visio-редактора, а также обрабатывает события, которые случаются при работе пользователя с редактором. Эта компонента также реализует интерфейс Customized Visio API для доступа DSL-компоненты к внутреннему представлению концептуальной модели при синхронизации. Кроме того, данная компонента реализует генерацию концептуальной модели по реализационной.

DSL-компонента реализует генерацию реализационной модели по концептуальной, а также синхронизацию концептуальной и реализационной моделей.

Documentation-компонента обеспечивает генерацию схемы документации по концептуальной модели, а также валидацию этой схемы при сохранении документа и изменение схемы при изменениях концептуальной модели. Она реализует интерфейс Doc API для синхронизации документации по изменившейся концептуальной модели. Этот интерфейс является программной «оберткой» Open XML-формата для доступа к документу в соответствии с разметкой, представленной на рис. 8.

4.2. Состав пакета

Для корректной работы нашего решения на компьютере должны быть установлены следующие продукты.

1. Microsoft.Net Framework 3.0 — необходим, так как решение создано с помощью Microsoft Visual Studio 2005, на языке C#.
2. Microsoft Visio 2003/2007 — для разработки концептуальной модели. Решение добавляет наш редактор концептуальной модели как Add-in к Visio.
3. Microsoft Visual Studio 2005 с DSL TOOLS — для разработки реализационной модели. Если реализационная модель не разрабатывается, то Visual Studio и DSL TOOLS не требуется. Независимость от Visual Studio позволяет использовать Visio и наше решение аналитикам, которые не занимаются разработкой и у которых поэтому отсутствует Visual Studio. Созданную концептуальную модель можно передать разработчикам позднее.
4. Microsoft Word 2007 — для разработки документации к создаваемому визуальному языку.

Для удобства установки решения мы создали инсталляционный пакет. Правила и процесс установки подробно описаны в работе [2].

4.3. Реализация RT-механизма

Основной особенностью реализации нашего решения является реализация RT-механизма. Остановимся на этом подробнее. Так как мы интегрируем три независимых программных средства и у нас нет общего репозитория и общей модели, то для переноса изменений без перегенерации нам необходимы средства для идентификации одних и тех же элементов в концептуальной, реализационной моделях и в документации. При их изменениях в одной спецификации соответственно должны меняться их «дублеры» в других спецификациях. В нашем решении все классы, ассоциации, наследования и атрибуты имеют глобальные уникальные идентификаторы (GUIDs), которые хранятся в свойствах этих элементов в Visio и в DSL TOOLS, а в документации они хранятся в XML-тегах внутреннего представления документа. Наличие GUID позволяет находить одни и те же элементы в разных моделях даже в том случае, если пользователь изменил их имена.

Перенос изменений из Visio-модели в DSL-модель (реализован в DSL-компоненте) происходит следующим образом. При работе пользователя в Visio, в редакторе концептуальной модели, GUID всех удаляемых элементов записываются в специальный список — историю удалений, — а изменяемые заносятся в другой список — историю изменений. Эти списки модифицируются в том числе и при использовании пользователем операции Undo (но до первого сохранения!). При сохранении выполненных изменений в редакторе концептуальной модели запускается алгоритм синхронизации Visio-модели с DSL-моделью. Сначала создается копия последней синхронизированной DSL-модели с тем, чтобы была возможность эффективно реализовать операцию Undo, которую после сохранения Visio-модели может инициировать пользователь. Далее, элементы Visio-модели, которых нет в DSL-модели, создаются, потом удаляются все элементы, записанные в истории удалений, а элементы из истории изменений синхронизируются. Список удаленных элементов необходим, чтобы не реализовывать дополнительный проход по DSL-модели.

Перенос изменений из DSL-модели в Visio-модель (реализован в DSL-компоненте) происходит следующим образом. Сравниваются текущая версия DSL-модели и последняя, синхронизированная с Visio-моделью. Элементы, которых нет в прежней копии, создаются, элементы, которых нет в новой модели, удаляются, элементы, существующие в обеих схемах, синхронизируются.

Перенос изменений из концептуальной модели в документацию (реализован в Documentation-компоненте) происходит так. Определяется разница между новой концептуальной моделью и XML-схемой документации. При этом используется дополнительная разметка XML-схемы, созданная нами в соответствии с моделью документа, представленного на рис. 8. На основе идентификации по GUID, списка измененных и удаленных элементов, XML-схема модифицируется.

Заключение

Наше решение реализует законченный объем функциональности. Оно прошло тестирование разработчиков и готово для пилотного использования. В дальнейшем мы планируем провести тщательное тестирование на больших моделях и сложных алгоритмах итеративной разработки, выяснив надежность нашего RT-

механизма. Кроме того, мы планируем также использовать это решение для создания адекватных формальных моделей циклической разработки.

Список литературы

[1] *Джонс Дж. К.* Инженерное и художественное конструирование / Пер. с англ. М.: Мир, 1976.

[2] *Залевский Я. И.* Концептуальное моделирование при разработке DSL-средств: Диплом. СПбГУ, 2008. 31 с.

[3] *Кознов Д. В.* Основы визуального моделирования: Учебное пособие. М.: Интернет-университет информационных Технологий; БИНОМ. Лаборатория знаний, 2008. 246 с.

[4] *Кознов Д. В.* Разработка и сопровождение DSM-решений на основе MSF // Системное программирование. Вып. 3 / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд-во СПбГУ, 2008. С. 80–96.

[5] *Павлинов А., Кознов Д., Перегудов А., Бугайченко Д.* и др. О средствах разработки проблемно-ориентированных визуальных языков // Системное программирование. Вып. 2 / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд. СПбГУ, 2006. С. 121–147.

[6] *Фаулер М., Скотт К.* UML. Основы / Пер. с англ. СПб.: Символ, 2006, 184 с.

[7] *Brooks F.* No Silver Bullet // Information Proceeding of the IFIP 10th World Computing Conference, 1986. P. 1069–1076. (Русский перевод: *Брукс Ф.* Мифический человеко-месяц, или как создаются программные системы. СПб.: Символ, 2000).

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

[9] *Dori D.* Why Significant UML Changes is Unlikely // Communication of the ACM. November. Vol. 45, N 11. 2002. P. 82–85.

[10] *Greenfield J., Short K. et al.* Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. John Wiley & Sons. 2004. 666 p. (Русский перевод: *Гринфилд Д.* и др. Фабрики разработки программ (Software Factories): потоковая сборка типовых приложений, моделирование, структуры и инструменты. Изд-во «Вильямс», 2006. 592 с.)

[11] *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.

[12] Microsoft Solutions Framework, Process Model, Version 3.1. 2002. http://www.microsoft.com/msf (Русский перевод: Microsoft Solutions Framework. Модель процессов, версия 3.1. 41 с.).

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

[14] *Robert B.* France, Sudipto Ghosh, Trung T. Dinh-Trong, Arnor Solberg. Model-Driven Development Using UML 2.0: Promises and Pitfalls // IEEE Computer 39(2). 2006. P. 59–66.

[15] *Sendall S., Kuster J. M.* Taming Model Round-Trip Engineering // Proceedings of Workshop on Best Practices for Model-Driven Software Development, Vancouver, Canada 2004.

[16] http://msdn.microsoft.com/en-us/library/bb126235.aspx

[17] http://office.microsoft.com/ru-ru/visio/default.aspx

[18] http://openxmldeveloper.org/

[19] http://www.eclipse.org/modeling/gmf/

[20] http://www.autodesk.ru