Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра информатики

Система визуального проектирования документации семейств программных продуктов

Дипломная работа студента 542 группы
Семёнова Алексея Александровича

Научный руководитель ст. преп.	…… / подпись /	К.Ю. Романовский
Рецензент к.ф.-м.н., доцент	…… / подпись /	Д.В. Кознов
“Допустить к защите” заведующий кафедрой, д.ф.-м.н., профессор	…… / подпись /	Н. К. Косовский

Санкт-Петербург
2007

Оглавление

Введение

Разработка семейств программных продуктов – популярный и потенциально чрезвычайно эффективный метод разработки ПО, позволяющий повторно использовать различные активы разработки, такие как модели, программные компоненты, требования [1, 2].

Техническая документация продуктов, составляющих семейство, может иметь значительный объём, при этом требования к качеству документации бывают весьма высокими. В таких случаях и разработка, и поддержка этой документации в актуальном состоянии являются чрезвычайно трудоёмкими процессами.

Известно несколько популярных подходов к разработке сложной технической документации. Например, подход единого исходного представления (Single Sourcing) обеспечивает разделение содержания и форматирования [3]. При его использовании возможно на основе единого исходного представления получить комплект документов, таких как руководство пользователя, справочная система и т.п. Другой распространённый подход – DITA (Darwin Information Typing Architecture), предложенный компанией IBM [4]. В рамках этого подхода предлагается строить документацию как набор отдельных блоков текста, так называемых «топиков», каждый из которых может быть использован в любом контексте в произвольном документе.

Документация семейств программных продуктов предоставляет особенно широкое поле возможностей для переиспользования, причём не только на уровне крупных блоков текста – глав или параграфов, – но также и на уровне названий, обозначений и т.п. Связано это в том числе и с тем, что документация обычно имеет структурные элементы, соответствующие программным компонентам, комбинируемым при построении продуктов в семействе. При этом переиспользуемые элементы в каждом конкретном случае нуждаются в адаптации к контексту использования и поэтому должны быть в определённой степени изменчивы, это свойство компонент называется «вариативностью».

Существующие подходы к созданюи технической документации не учитывают особенностей разработки документации в рамках семейств программных продуктов – средства переиспользования в них не предоставляют достаточной гибкости. Фактически основным способом реализации вариативности в них является условное включение блока текста, а более тонкая адаптация фрагмента к контексту невозможна.

В работе [2] представлен метод разработки документации семейств программных продуктов DocLine. Он включает модельно-ориентированный процесс разработки, согласованный с процессом разработки семейств программных продуктов, язык разметки документации DRL/PR (Documentation Reuse Language/Phrase Representation, дальше в тексте мы будем называть его просто DRL) и визуальную нотацию для проектирования документации DRL/GR (Documentation Reuse Language/Graphical Representation). Язык DRL/GR основан на нотации диаграмм возможностей, широко применяемой при разработке семейств программных продуктов [5], и включает диаграммы трёх видов:

- главная диаграмма – диаграмма верхнего уровня, содержит список продуктов семейства и шаблонов документов («информационных продуктов») из документации семейства;
- диаграмма вариативности – отображает структуру информационного продукта: входящие в него информационные элементы и связи между ними;
- диаграмма продукта – показывает структуру после его адаптации соответствующего информационного продукта или информационного элемента.

Разработка специального языка для решения конкретной задачи, синтаксиса визуальной модели и инфраструктуры программных средств для их поддержки обобщается в рамках подхода предметно-ориентированного моделирования (Domain Specific Modeling, DSM). Суть этого подхода – в сужении целевой области применения визуального моделирования (вплоть до отдельных проектов) и достижении, за счёт этого, большей эффективности автоматизированных решений, созданных на его основе [1, 6].

В нашем случае задачей является разработка документации семейств программных продуктов, а языком – DRL. На данный момент существует несколько платформ, предоставляющих средства реализации предметно-ориентированного подхода, в этой работе была выбрана одна из наиболее зрелых среди них – Eclipse GMF [6].

Данная работа проведена в рамках исследовательского проекта, выполняющегося на кафедре системного программирования СПбГУ и посвящённого созданию метода разработки документации семейств программных продуктов DocLine [2]. В этой работе представлены средства поддержки визуального проектирования документации, а реализация редактора и валидатора текста в формате DRL стали предметом дипломной работы Константина Яковлева [7].

1. Постановка задачи

Основная задача данной дипломной работы – создание на основе Eclipse GMF системы визуального проектирования документации семейств программных продуктов, реализующей метод DocLine. Эту задачу можно разбить на несколько подзадач.

1. Изучить технологию Eclipse GMF. Эта задача осложнена тем, что технология относительно нова (разрабатывается с 2005 года), пока что не очень хорошо документирована, опыт её промышленного использования только начинает появляться в таких компаниях, как Borland, GentleWare и IBM.
2. Создать проектную спецификацию графических редакторов DRL.
3. На основе описания DRL создать средствами EMF метамодель DRL.
4. По метамодели с помощью Eclipse GMF создать базовые графические редакторы.
5. Реализовать функциональность редакторов, выходящую за рамки Eclipse GMF.

Должны быть реализованы графические редакторы для диаграмм двух видов:

- редактор главной диаграммы (диаграммы семейства),
- редактор диаграммы вариативности (диаграммы информационного продукта).

2. Обзор литературы

2.1. Семейства программных продуктов

Семейство программных продуктов – это набор программных систем, предназначенных для определённого сегмента рынка и имеющих некоторый контролируемый набор общих характеристик, а также разрабатываемых установленным образом используя общие активы. В качестве переиспользуемых ресурсов при этом выступают не только архитектура, модели и код, но также и требования, процессы, средства разработки, компетенции персонала и т.д. [8].

Этот подход к разработке программных продуктов является относительно новым; среди его сильных сторон отмечают значительное повышение производительности труда, повышение качества продуктов, уменьшение времени вывода на рынок, значительное снижение общей стоимости разработки и др.

Каждый продукт в семействе относительно независим в том смысле, что он может иметь собственного заказчика и свой уникальный набор требований, более того, заказчик может даже не знать о существовании инфраструктуры семейства. Но изготавливается продукт с помощью компонентов, взятых из общих активов и настроенных с помощью механизмов вариативности, заложенных в них при проектировании, с помощью новых компонентов, специфичных для продукта, и собираются они в соответствии с правилами, определёнными для семейства.

В качестве примера семейства программных продуктов можно привести линейку браузеров для мобильных устройств, разрабатываемую в компании Nokia с 1999 года [9]. Использование этого подхода позволило компании к концу 2001 года выпускать до шести версий продуктов в год и предоставлять браузеры для широкого ряда мобильных телефонов с операционными системами Nokia и Symbian, эмулятор браузера в составе инструментария разработки для Windows и компактную версию, не содержащую зависимостей от платформ Nokia для использования такими заказчиками как Samsung и AOL.

Несмотря на многочисленные различия, во всех продуктах из линейки Nokia активно использовались общие компоненты, например, реализация стека протоколов, средства поддержки разметки и скриптовых языков и менеджеры расположения.

В процессе разработки семейства, согласно [8], выделяют три основных вида деятельности:

- разработка общих активов,
- разработка целевых систем,
- управление.

При разработке общих активов принимаются во внимание требования к продуктам, стратегия разработки, производственные ограничения и наличие ресурсов, активов и опыта. Результатами являются архитектура семейства продуктов, спецификация области охвата, метод использования общих активов, привязка требований к архитектуре и, наконец, сами общие активы.

В процессе разработки целевых систем используются данные, полученные при разработке общих активов, а также добавляются требования, выдвигаемые к конкретному продукту. По каждому из требований принимается решение – следует его реализовывать отдельно, в

рамках разработки общих активов или нужная функциональность уже реализована в общих компонентах.

Управление разработкой семейства программных продуктов осуществляется на двух уровнях: техническом и организационном.

Техническое управление контролирует исполнение метода разработки, определённого для семейства, управляет проектами и решает вопросы, связанные с инструментальной поддержкой и конфигурационным управлением.

Организационное управление анализирует рынок, обеспечивает ресурсами, управляет взаимодействиями с внешним миром, разрешает конфликты и осуществляет стратегическое планирование.

Все три вида деятельности имеют выраженный итеративный характер и тесно связаны между собой.

2.2. Диаграммы возможностей

Одна из основных задач разработки семейств программных продуктов – необходимость отображения всех возможных вариаций продуктов семейства.

Одним из самых удачных решений этой проблемы оказалась идея метода «моделирования возможностей» (англ. «feature modeling»), впервые предложенная в 1990 году в работе [10]. Смысл метода состоит в определении сходств и различий, общих и требующих настройки частей в продуктах, составляющих семейство, анализе зависимостей этих частей и упорядочении знаний об этих деталях с помощью согласованной диаграммы, называемой также «моделью возможностей».

Ключевые понятия моделирования возможностей – концепт, возможность, точка вариативности.

Рисунок 1: Пример диаграммы возможностей [2]

Концепт (англ. «concept») – это элемент или структура в предметной области [5].

Возможность (англ. «feature») – некоторое важное свойство концепта, различимое с точки зрения пользователя или разработчика. Возможности позволяют выразить сходства и различия между разными экземплярами реализации концепта, а упорядоченные иерархически на диаграмме возможностей, они отражают его изменчивость.

Иерархическая связь отображает декомпозицию концепта или возможности и называется отношением включения.

Точки вариативности на диаграмме отображают необязательные включения и условия на группы включений. Они могут быть по-разному истолкованы в различных системах, реализующих концепт.

Пример диаграммы возможностей показан на рисунке 1. Телефонный аппарат «Унител» является концептом, обозначающим семейство телефонных аппаратов с различными наборами функций, возможности здесь – выделенные функциональные блоки. Телефонные аппараты в семействе должны поддерживать входящие звонки или исходящие звонки (или и те и другие), причём модуль исходящих звонков обязательно содержит модуль «номеронабиратель». Также в аппарат могут быть включены автоответчик и АОН, который, в свою очередь, реализуется либо в российском стандарте ГОСТ, либо в международном Caller ID.

Диаграммы возможностей в оригинальном варианте могут использоваться только на начальных этапах разработки ПО – при анализе и проектировании, и они не имеют никакой связи с артефактами последующих этапов, такими как программный код или документация [2].

2.3. Предметно-ориентированное моделирование

Использование техник визуального моделирования способствует увеличению эффективности при разработке системы в силу увеличения уровня используемых абстракций. С другой стороны, это же увеличение абстракции может оказаться значительным затруднением при необходимости синхронизации моделей с низкоуровневыми артефактами: нетривиальное автоматизированное использование визуальных моделей является задачей непростой и дорогостоящей [6].

Кроме того, до сих пор в силу ряда естественных причин не существует универсальных методов визуального моделирования ПО. В работе [6] указано, что это связано как с тем, что программные продукты не имеют и не могут иметь естественной общепринятой метафоры для визуализации в силу своей невидимости, так и с тем, что на настоящий момент не существует универсального процесса разработки ПО. Как правило, любое средство визуального моделирования требует настройки и адаптации для нужд конкретного заказчика и конкретного проекта6.

Подход предметно-ориентированного моделирования (далее – DSM-подход) предполагает максимальное сужение целевой области применения визуального моделирования – создавать визуальные языки и средства визуального моделирования для решения конкретной задачи. За счёт этого достигается, с одной стороны, соответствие потребностям заказчика и разработчиков, а с другой – значительное увеличение эффективности автоматических решений [1, 6].

Идея предметно-ориентированного моделирования поддерживается целым рядом специализированных инструментальных средств. В силу относительной новизны идеи и сложности её реализации, на сегодняшний день лишь немногие из них достаточно стабильны и развиты, чтобы считаться готовыми к применению в промышленности [6].

Одними из наиболее известных DSM-инструментов являются продукты от Microsoft – Microsoft DSL[1] Tools и Microsoft Visio.

1 DSL – Domain Specific Language, предметно-ориентированный язык.

Инициатива Microsoft Software Factories является попыткой индустриализации процесса разработки ПО. Она состоит в использовании «фабрик» по созданию ПО для ускорения процесса разработки и минимизации затрат. В общем случае под «фабрикой» понимается расширяемая среда разработки (например Microsoft Visual Studio), включающая DSM-средства, средства инструментальной поддержки шаблонов проектирования и т. п. В рамках данной инициативы предоставляется набор средств Microsoft DSL Tools, входящий в Visual Studio SDK[2], 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 генерации кода в создаваемых редакторах.

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 [6, 11].

Пакет Microsoft Visio широко распространён, легок в использовании и может быть гибко настроен. Он представляет собой средство для построения схем и диаграмм различного типа, реализуя различные средства работы с базовыми геометрическими фигурами, текстовыми полями, связанными с этими фигурами; также он предоставляет возможность задавать поведение графических фигур (например, ограничения на изменение размеров по высоте и/или ширине) и т.д. [6]

Кроме этих возможностей пакет Microsoft Visio содержит средства для реализации новых графических языков: можно задавать новые нотации, определять графические свойства новых фигур. Кроме этого, пакет поддерживает встроенный язык сценариев, позволяющий создавать относительно сложные модели поведения графических объектов. Спецификация нового визуального 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 интерфейсы 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 [6].

2.4. Платформа Eclipse GMF

Среда Eclipse – всё более популярная, динамично развивающаяся платформа с открытыми исходными кодами для создания инструментов разработки ПО. На её основе создана развитая интегрированная среда разработки (IDE) для Java, часто при упоминании названия «Eclipse» имеют в виду именно эту IDE. Также для Eclipse существуют средства поддержки разработки на других языках, таких как C++, Perl, Fortran и т.д., однако основным является Java. На текущий момент во многих компаниях Eclipse является корпоративным стандартом для разработки Java-приложений.

Успех этой среды объявняется с одной стороны её бесплатностью, открытостью, модульной структурой, специально спроектированной для расширения, и с другой стороны – существованием развитого сообщества пользователей и разработчиков и поддержкой со стороны таких компаний как IBM, Borland, HP, BEA и Red Hat [12]. С 2004 года ежегодно проводится конференция EclipseCon [13].

2 SDK – Software Development Kit, инструментарий разработчика.

В основе архитектуры Eclipse лежит реализация технологии OSGi[3] – спецификации динамической модульной структуры для Java. Она позволила разделить функциональность системы между модулями («плагинами»), декларирующими свои интерфейсы и зависимости от других плагинов – таким образом, добавление новой функции к Eclipse сводится к реализации соответствующего плагина или набора плагинов. Для использования пользователю нужно будет просто добавить эти плагины к своей среде Eclipse с помощью стандартных мастеров.

Одним из практических результатов данной дипломной работы как раз и является набор плагинов, после установки которых пользователь получит инструменты для работы с документацией в соответствии с подходом DocLine.

Технология Eclipse GMF позволяет в контексте предметно-ориентированного моделирования создавать графические редакторы, основанные на Eclipse.

С технической точки зрения, технология Eclipse Graphical Modeling Framework (GMF) образует «связь» между технологиями Eclipse Modeling Framework (EMF) и Eclipse Graphical Editing Framework (GEF).

Eclipse Modeling Framework – фреймворк и средство генерации Java-кода для построения кода структурированной модели бизнес-объектов с поддержкой редактирования. EMF предоставляет в том числе и простой визуальный редактор моделей в формате упрощённых диаграмм классов UML.

Кроме увеличения эффективности работы относительно реализации кода модели, использование EMF имеет несколько дополнительных преимуществ: сгенерированный код содержит механизм уведомлений об изменениях, возможность сохранения и загрузки в XML-файлы в формате XMI[4] или в формате, определённом XSD[5]-схемой, стандартные средства для валидации и очень эффективные программные интерфейсы для манипулирования объектами модели с помощью рефлексии.

EMF состоит из двух частей: основной фреймворк и EMF.Edit. В основной фреймворк входят библиотеки поддержки времени исполнения и инструменты генерации кода модели. EMF.Edit дополняет основной фреймворк и позволяет интегрировать полученный код с Eclipse: предоставляет средства генерации слоя адаптеров для связи со стандартными вспомогательными инструментами Eclipse и слоя команд для редактирования, а также позволяет сгенерировать простой редактор для Eclipse.

Eclipse Modeling Framework разрабатывается с 2001 года и на данный момент эта технология достигла значительной зрелости. Модели данных, лежащие в основе GEF и GMF, были также разработаны с использованием EMF.GEF позволяет на базе какой-либо модели создать визуальный редактор, обладающий богатым набором стандартных фунций, при этом архитектура будет соответствовать шаблону MVC[6]. GEF включает в себя библиотеку Draw2D, которая содержит инструментарий отображения графических

3 OSGi – Open Services Gateway initiative (сейчас эта расшифровка считается устаревшей), разрабатывается открытой стандартизационной организацией OSGi Alliance (http://www.osgi.org/).

4 XML Metadata Interchange (XMI) – стандарт OMG сериализации в формате XML метаданных, метамодель которых может быть выражена с помощью MOF – в частности, этому требованию удовлетворяет метамодель ECore, используемая в EMF.

5 XML Schema Definition (XSD) – язык описания схемы XML, стандарт W3C-консорциума.

6 Model View Controller (MVC) – архитектура программного обеспечения, в которой модель данных приложения, пользовательский интерфейс и управляющая логика разделены на три отдельных компонента, так, что модификация одного из компонентов оказывает минимальное воздействие на другие компоненты.

элементов, менеджеры расположения графических объектов, механизм событий, палитру стандартных объектов, и пр.Модель данных может использоваться любая, но особенно эффективно использование EMF и GEF в паре – по этой причине и была создана GMF.

В процессе разработки визуального редактора с помощью Eclipse GMF пользователь определяет несколько моделей. Полностью работающий код автоматически генерируется по ним, а также перегенерируется при их изменении, при этом полученный код имеет высокое качество – шаблоны для генерации были тщательно просмотрены и использованы для решения разнообразных задач довольно активным и многочисленным сообществом пользователей GMF.

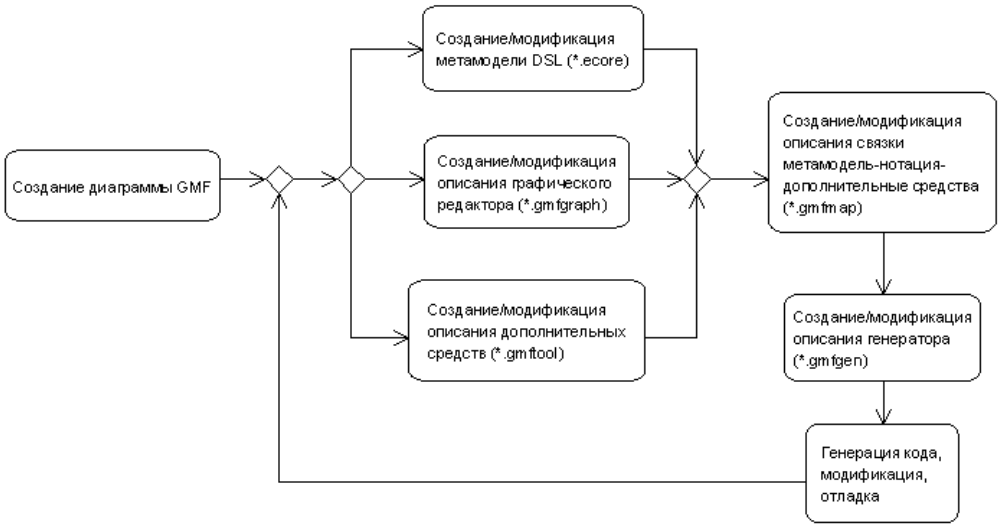

Рисунок 2: Схема разработки с помощью Eclipse GMF [6]

Процесс разработки DSM-пакетов на основе GMF изображен на рисунке 2 и состоит из следующих шагов [6,14]:

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 графических объектов и объектов метамодели, а также вспомогательных средств, описание ограничений на OCL или Java (выходной файл *.gmfmap).
5. Создание описания генератора кода, его модификация (выходной файл *.gmfgen).
6. Генерация кода целевого DSM-средства, запуск отладочного экземпляра Eclipse Application и отладка.
7. Внесение модификаций в исходный код сгенерированного редактора для приведение его в соответствие требованиям, которые невозможно удовлетворить с помощью настройки стандартных средств генерации GMF и для большей специализации его под конкретную задачу.

Разработка редактора на базе каркаса GMF и с помощью ряда моделей обнаружила многие достоинства и недостатки, свойственные этому подходу.

Среди достоинств можно выделить следующие:

- полная интегрированность созданного редактора в Eclipse без необходимости значительных дополнительных усилий позволяет предоставить пользователю целый ряд возможностей:
  - отображение информации в стандартном для пользовательского интерфейса Eclipse виде и с использованием стандартных средств,
  - интеграция инструментов разработки документации со средствами разработки программной системы,
  - возможность использовать существующие мощные средства для побочных задач, связанных с разработкой документации – например, средства версионного контроля;
- многочисленные элементы функциональности, являющиеся элементами каркаса и тоже не потребовавшие дополнительных усилий для реализации – например, поддержка печати, удобный навигатор по диаграмме, сохранение графических данных о диаграмме в отдельный от модели файл;
- удобство разработки, возможность сконцентрироваться на логике, а не на реализации базовых деталей, таких как поддержка стека команд, палитра инструментов, функции рисования.

Недостатки:

- сложность распространения изменений из GMF-моделей в код – итеративность в разработке моделей и кода редактора неизбежна, при этом перенос изменений из модели в код далеко не всегда является тривиальной задачей;
- сложность реализации функциональности, выходящей за рамки стандартов Eclipse GMF – например, внедрение нетривиальной функциональности без нарушения стройности концепций кода системы является непростой задачей, как, например, в случае с реализацией редактора диаграммы вариативности;
- недостаток документации и данных о распространённых практиках применения – очень многие детали приходится узнавать либо экспериментально, либо из кода самого GMF и сгенерированного кода редактора диаграмм.

При разработке редакторов использовалась версия Eclipse 3.3 M6 и GMF 2.0 M6 – одна из последних промежуточных версий на пути к финальной версии, выпуск которой запланирован на конец июня 2007 года. Это позволило использовать новейшие наработки в GMF, такие как значительно улучшенная по сравнению с версией 1.0 архитектура, улучшения в поддержке синхронизации диаграммы и файла модели, поддержка перегенерации файлов описания плагина plugin.xml и plugin.properties и пр.

С другой стороны, платой за новизну была некоторая нестабильность платформы Eclipse GMF и плохая документированность.

2.5. Метод DocLine и язык DRL

Метод разработки документации семейств программных продуктов DocLine предложен на кафедре системного программирования математико-механического факультета СПбГУ. Назначение метода – организация разработки документации способом, максимально

учитывающим особенности документации этого класса программных систем и типичные задачи и проблемы, возникающие при её создании. Одной из важнейших целей, преследуемых авторами метода, является максимизирование повторного использования блоков текста в рамках документации, при этом с сохранением читаемости исходных текстов и удобства для технических писателей.

Метод DocLine состоит из трёх частей [2,15]:

- процесс разработки документации;
- язык разработки документации DRL, включающий текстовую и графическую нотации;
- архитектура пакета инструментальных средств.

2.5.1. Процесс разработки документации

Процесс разработки документации, как и процесс разработки семейства в целом, состоит из двух частей – разработки документации семейства и разработки документации продуктов. При необходимости допускается вносить изменения в документацию семейства при разработке документации продуктов – это может потребоваться для тонкой настройки повторного использования документации.

Графическая нотация предназначена для крупноблочного проектирования документации, текстовая – для написания собственно текста. При работе с документацией процесс допускает одновременную работу с этими двумя представлениями – технический писатель может переходить от написания текста к редактированию диаграмм и обратно, без нарушения структуры текста. Возможность синхронизации диаграмм с изменениями в текстовом представлении называется «раундтрип»[7], и средство визуального моделирования должно её корректно поддерживать.

2.5.2. Обзор графической нотации DRL

Графическая нотация DRL (DRL/GR) ориентирована на проектирование структуры документации, а также позволяет получить информацию о её структуре и осуществлять навигацию по её блокам.

В графической нотации визуализируются данные о структуре семейства, структуре информационных продуктов, информационных элементах и об отношениях между ними. Таким образом, отображается только информация о крупных структурных блоках – детали более мелкого («мелкозернистого») повторного использования и текст документации на диаграммах не изображаются.

Всего DRL/GR содержит диаграммы трёх видов, в этой работе были реализованы две из них.

7 Roundtrip (англ.) - поездка в оба конца.

Рисунок 3: Главная диаграмма DRL [2]

Главная диаграмма (рисунок 3) позволяет определить продукты семейства и доступные информационные продукты, а также задать связи между ними.

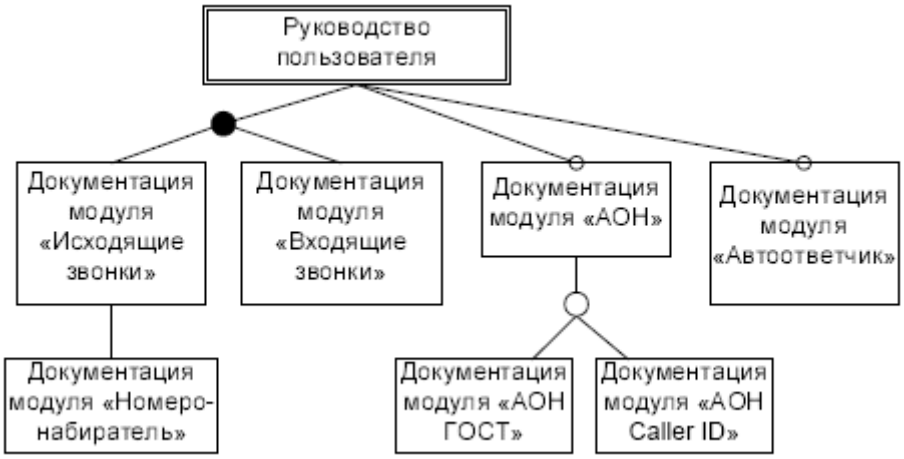

Рисунок 4: Диаграмма вариативности DRL [2]

Диаграмма вариативности отображает структуру информационного продукта или информационного элемента. Связи на этой диаграмме обозначают отношения включения.

На диаграмме вариативности присутствуют узлы двух типов: прямоугольник соответствует информационному элементу (тонкая граница) или информационному продукту (толстая граница), круг – группе ссылок на информационные продукты. Чёрный круг – группа типа OR (допустимо включение непустого подмножества ссылок из группы), белый круг – группа типа XOR (необходимо включение ровно одной ссылки из группы).

Необязательные включения в месте соединения с включаемым информационным элементом отображают пустой кружок. Необязательными могут быть включения как входящие в группу, так и самостоятельные.

Структура отображается в виде дерева: если один и тот же информационный элемент входит в два разных элемента из тех, что показаны на диаграмме, то на ней для этого

информационного элемента будет изображено две копии вершины, обе они будут ссылаться на один и тот же элемент в тексте документации. Соответственно, на диаграмме будут присутствовать и два одинаковых поддерева с корнями в этих вершинах.

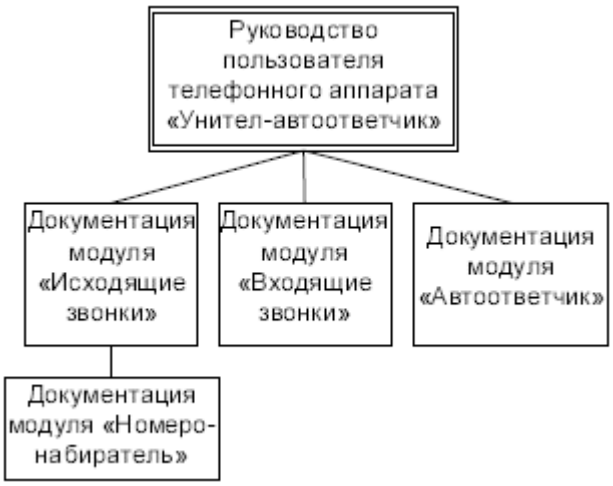

Рисунок 5: Диаграмма продукта DRL [2]

Диаграмма продукта показывает структуру документа после его адаптации. Можно воспринимать её как диаграмму вариативности, на которой разрешили все точки выбора и исключили неиспользованные элементы. Эта диаграмма осталась за рамками данной дипломной работы.

2.5.3. Обзор текстовой нотации DRL

Текстовая нотация DRL (DRL/PR) предназначена для написания текста документации с последующей трансляцией в один из форматов, пригодных для выбранного способа публикации.

Язык DRL/PR основана на XML, его синтаксис задан с помощью схемы Relax NG, разработанной Константином Яковлевым [7]. Элементы DRL/PR бывают двух видов – относящиеся тексту и его разметке и управляющие структурой документации.

Элементы первого вида заимствованы из DocBook – XML-формата разметки, уже многие годы являющегося стандартом де-факто для создания сложной технической документации [16]. XML-представление позволяет эффективно хранить документы этого формата в системах контроля версий, кроме того, в рамках проекта DocBook разрабатываются скрипты и таблицы стилей (XSL – Extended Specification Language) для конвертации документов DocBook во все популярные форматы публикации.

Элементы управления структурой позволяют организовать повторное использование частей документации – как «крупноблочное», так и «мелкозернистое», а также адаптировать фрагменты текста к контексту использования.

Поскольку текст и элементы DocBook на диаграммах не отображаются и поэтому для нас интереса не представляют, далее при разговоре об элементах DRL мы будем подразумевать только элементы управления структурой.

Основная конструкция DRL – семейство продуктов, с её помощью описывается всё, что касается семейства программных продуктов. Семейство продуктов в DRL содержит схему семейства и документацию.

Схема семейства описывает продукты, из которых состоит семейство. Документация в DRL делится на документацию семейства и документацию продукта – т.е. на общую часть и относящуюся к конкретным продуктам.

Документация семейства указывает, какие документы («информационные продукты») могут входить в семейство и как они устроены: из каких частей («информационных элементов») состоят, где допустима настройка повторного использования и какая. Фактически, информационный продукт можно понимать как шаблон документа, который после указания всех параметров и настроек будет готов к включению в документацию конечного продукта.

Документация продукта состоит из набора «специализированных информационных продуктов» и описывает, как именно нужно модифицировать документацию семейства, чтобы получить пакет документов для конкретного продукта семейства. Каждый специализированный информационный продукт соответствует одному информационному продукту и содержит спецификации и настройки, необходимые, чтобы на основе этого информационного продукта получить документ. Также он содержит ссылку на продукт из схемы семейства, к которому полученный документ будет относиться.

Файлы документации в формате DRL являются XML-файлами и делятся на три вида, каждому из которых соответствует свой корневой элемент.

Файл, содержащий информацию о схеме семейства продуктов, в качестве корневого элемента имеет тег ProductLine (этот и все следующие примеры взяты из работы [2]).

```
<ProductLine name="Семейство телефонных аппаратов Унител">
  <Product name="Унител-таксофон" id="payphone"/>
  <Product name="Унител-автоответчик" id="answermachine"/>
  <Product name="Унител-АОН" id="callerid"/>
</ProductLine>
```

Листинг 1: схема семейства продуктов

Каждый из продуктов имеет имя (атрибут name) для восприятия человеком и идентификатор (атрибут id) для ссылок.

Документация семейства хранится в файлах с корневым элементом DocumentationCore.

```
<DocumentationCore>
  <InfProduct id="userguide" name="Руководство пользователя"/>
  <InfElement id="numberid" name="АОН"/>
  <Dictionary id="maindict" name="Основной словарь"/>
  <Directory id="maindir" name="Основной каталог"/>
</DocumentationCore>
```

Листинг 2: документация семейства

Тег InfProduct здесь соответствует объявлению информационного продукта, InfElement – информационного элемента. Теги Dictionary и Directory содержат спецификации соответственно словарей и каталогов и используются для организации мелкозернистого переиспользования. Все элементы имеют идентификатор (атрибут id) и имя (атрибут name), они используются тем же образом, что и в предыдущем примере.

Документация продукта хранится в файле с корневым элементом ProductDocumentation, пример – на листинге 3.

```
<ProductDocumentation productid="callerid">
```

```
  <FinalInfProduct infproductid="userguide">
    <Adapter infelemrefid="CallerID_aon_ref">
      <Insert-After nestid="number_indication_methods">
        Также предусмотрена возможность
        звуковой индикации номера абонента.
      </Insert-After>
    </Adapter>
  </FinalInfProduct>
  <Dictionary id="dict_userguide">
    <!-- ... -->
  </Dictionary>
</ProductDocumentation>
```

Листинг 3: документация продукта

Атрибут productid задаёт привязку к документу из схемы семейства. FinalInfProduct определяет специализированный информационный продукт, идентификатор соответствующего ему информационного продукта задан атрибутом infproductid. Теги Adapter служат для адаптации информационных элементов. В случае, рассмотренном в примере, в информационном элементе с идентификатором CallerID_aon_ref вставляется текст перед точкой расширения с id, равным dict_userguide.

2.6. Выводы

Метод разработки семейств программных продуктов – это современный эффективный метод разработки ПО, основанный на активном и управляемом повторном использовании доступных ресурсов. Проблеме разработки документации семейства, не получавшей до сих пор должного внимания, адресован метод разработки документации DocLine.

Основой метода является предложенный его авторами язык разметки документации DRL, позволяющий организовать высокую степень повторного использования блоков текста. DRL имеет две нотации – текстовую (DRL/PR) и графическую (DRL/GR). Задача построения редактора визуальной модели DRL – пример задачи предметно-ориентированного моделирования. Эта задача может быть разрешена с помощью Eclipse, современной среды создания инструментальных средств поддержки разработки ПО, и Eclipse GMF, технологии разработки визуальных редакторов на базе Eclipse.

3. Архитектура визуальных средств

Один из важнейших результатов данной работы – визуальные редакторы DRL. При их создании использовался стандартный процесс разработки Eclipse GMF, архитектура полученных редакторов основывается на стандартной архитектуре редактора GMF.

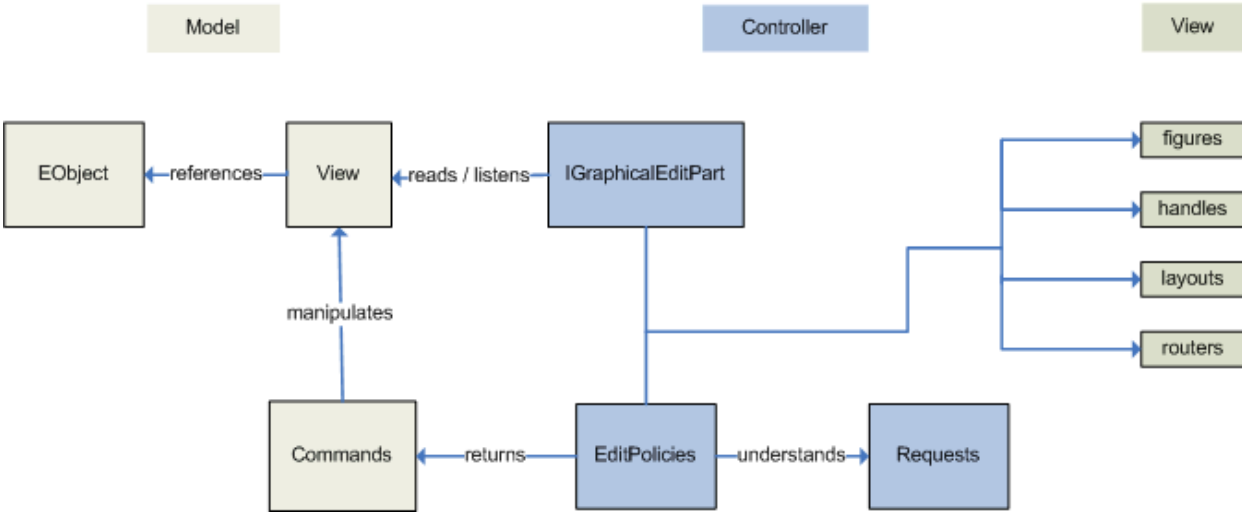

Рисунок 6: Архитектура визуального редактора GMF [18]

На рисунке 6 изображена архитектура шаблонного редактора GMF, она построена по принципу проектирования MVC.

- «Модель» состоит из модели, созданной с помощью EMF, объектов нотационной модели GMF и команд GMF;
- «Контроллер» содержит объекты GraphicalEditPart, они определяют поведение графических объектов на диаграмме с помощью EditPolicies, обрабатывающих события редактора Requests и создающих команды редактирования модели;
- Библиотека Eclipse Draw2D предоставляет средства рисования и размещения объектов для GEF, её объекты образуют слой «представления».

При разработке редакторов потребовались активные модификации сгенерированного кода, лежащего в областях модели и контроллера.

Как упоминалось в обзоре Eclipse GMF, создание редакторов основано на разработке набора моделей и описаний. В данной работе потребовалось определить 8 файлов:

- model.ecore – метамодель данных приложения;
- model.genmodel – настройки генератора кода EMF
- model.gmfgraph – описание графического редактора: все возможные вершины, рёбра и их устройство;
- model.gmftool – описание дополнительных средств, в нашем случае это – перечень всех инструментов создания, с каждым из них ассоциированы нужные иконки;
- productline.gmfmap, infproduct.gmfmap – описания связи графической модели и модели данных редактора, они раздельны для двух созданных редакторов;
- productline.gmfgen, infproduct.gmfgen – настройки генератора кода GMF, также различаются для редакторов.

Первые два файла относятся к EMF, остальные – к GMF, по ним генерируется код редактора на базе GEF. Зависимости между ними были показаны на рисунке 2 – модель данных, описания графического редактора и дополнительных средств самостоятельны, от них зависят описание связки и настройки генераторов.

Таким образом, оба созданные редактора разделяют средства работы с файлами DRL и графические определения вершин, но при этом имеют собственные способы их организации на диаграмме.

Метамодель EMF будет подробно рассмотрена в следующем разделе, а то, что касается наиболее важных из других деталей реализации визуальных редакторов, будет обсуждено в разделе, посвящённом реализации.

3.1. Метамодель EMF

3.1.1. Создание метамодели

В процессе создания EMF-метамодели DRL принимались во внимание два основных обстоятельства:

- она должна использоваться в визуальных редакторах и поэтому отвечать их структуре;
- формат файла сериализации модели должен быть близок к DRL.

Близость формата к DRL необходима для того, чтобы иметь возможность использовать стандартный надёжный код загрузки и сохранения модели, а преобразование между форматом сохранения и DRL производить с помощью XSLT трансформации. Детали этого преобразования будут подробнее рассмотрены в этом разделе ниже.

Существует несколько способов создания модели в EMF:

- создать с помощью собственно редакторов EMF;
- создать файл в формате XMI, в котором EMF хранит модель, вручную – например, с помощью XML-редактора;
- экспортировать XMI-документ с моделью из поддерживающего этот формат средства визуального моделирования, например, Rational Rose;
- создать Java-классы модели вручную и аннотировать методы доступа к полям свойствами модели;
- использовать мастер создания модели по XSD-схеме.

Из всех доступных вариантов, кроме первого, интерес представляет только последний, но использование этого способа имеет существенный недостаток: в результате получается метамодель, сложная для восприятия и для редактирования вручную. Для создания модели DRL автор воспользовался редакторами EMF.

3.1.2. Структура метамодели

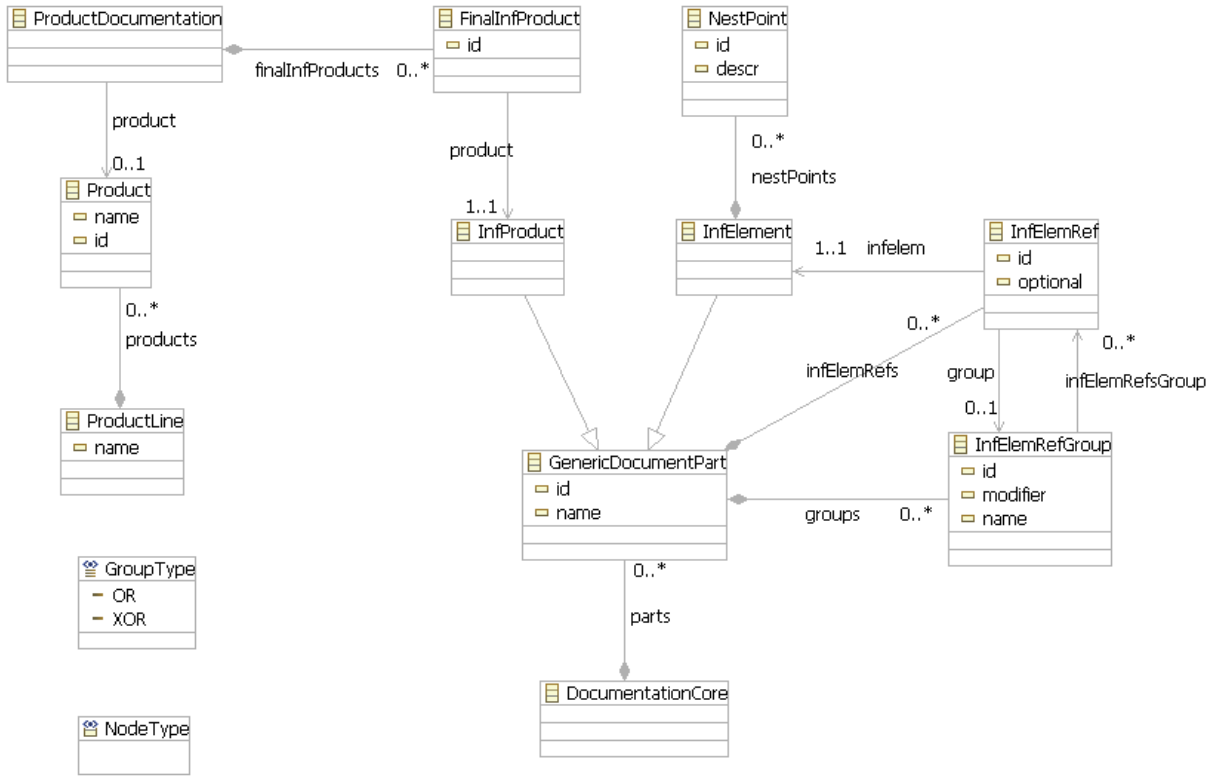

Рисунок 7: EMF-метамодель DRL

Полученная метамодель DRL изображена на рисунке 7.

Для облегчения читаемости на диаграмме не показано, что у всех классов модели есть общий предок – DrlElement, он содержит некоторые общие поля и методы.

Эта метамодель, в основном, повторяет элементы текстового представления DRL/PR. Трём элементам верхнего уровня – DocumentationCore, ProductDocumentation и ProductLine соответствуют одноимённые объекты метамодели, все остальные объекты агрегированы одним из них.

Элементы ProductLine, Product, ProductDocumentation, FinalInfProduct, NestPoint и DocumentationCore полностью соответствуют своим аналогам в DRL/PR – то есть, имеют одинаковые имена и наборы атрибутов. Однако, есть и ряд особенностей модели по сравнению с DRL/PR:

1. InfProduct и InfElement обобщены в GenericDocumentPart – в тексте документации оба они расположены внутри DocumentationCore, имеют имя и идентификатор, агрегируют ссылки на информационные элементы и определения групп ссылок. InfElement, кроме этого, может содержать определения NestPoint. Элемент GenericDocumentPart не имеет аналога в DRL/PR.
2. Между объектом ссылки InfElemRef и объектом группы ссылок InfElemRefGroup существует двусторонняя связь – это сделано для удобства манипулирования моделью. Сериализуется, в соответствии с DRL/PR, только связь от ссылки к группе.

3. GroupType – перечисление возможных типов группы, этот тип имеет поле modifier в InfElemRefGroup. DRL определяет два возможных типа – OR и XOR.
4. NodeType – тип данных, является ссылкой на элемент в дереве DOM. Используется в элементе DrlElement.
5. DrlElement – общий предок для всех элементов диаграммы, кроме специальных – GroupType и NodeType, он не показан на диаграмме. Он ничего не изменяет в формате сериализации модели и нужен для работы со ссылкой элемента на соответствующий узел в дереве документа DRL. Подробнее дерево документа DRL будет обсуждаться в разделе 3.2. DrlElement содержит поле node типа NodeType и определяет операции инициализации элемента дерева и обновления его атрибутов.

3.1.3. Сохранение и загрузка в формате DRL/PR

В соответствии с отношением агрегации метамодель можно разделить на три части:

- ProductLine, Product;
- ProductDocumentation, FinalInfProduct;
- DocumentationCore, все остальные элементы.

Стандартный механизм EMF сериализует объекты в формате XMI. При этом агрегированные элементы записываются как вложенные, а ссылки без агрегации записываются как атрибут-ссылка на id. Таким образом, мы получаем 3 типа файлов, в соответствии с текстовой нотацией DRL.

Сохранённая средствами EMF в формате XMI, модель имеет следующий вид:

```
<v:DocumentationCore
  xmlns:v="http://math.spbu.ru/drl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <parts id="Element1" name="Inf Element 1" xsi:type="v:InfElement">
    <infElemRefs id="ref1" infelem="somefile.drl#Element2"
optional="true"/>
  </parts>
  <parts id="Element2" name="Inf Element 2" xsi:type="v:InfElement"/>
</v:DocumentationCore>
```

Листинг 4: Сериализация модели в XMI

Соответствующий фрагмент DRL должен был бы иметь такой вид:

```
<v:DocumentationCore
  xmlns:v="http://math.spbu.ru/drl">
  <v:InfElement id="Element1" name="Inf Element 1">
    <v:InfElemRef id="ref1" infelem="Element2" optional="true"/>
  </v:InfElement>
  <v:InfElement id="Element2" name="Inf Element 2"/>
</v:DocumentationCore>
```

Листинг 5: Сериализация модели в XMI

Пример показывает, что структура файла соответствует DRL. Для полного соответствия при конвертации из XMI в DRL необходим ряд преобразований

1. Заменить названия ссылок на имена сущностей, на которые они ссылаются. Например, «parts» на «v:InfElement» и «infElemRefs» на «v:InfElemRef».
2. EMF реализует сохранение информации о полиморфизме GenericDocumentPart в атрибуте xsi:type элемента parts. Необходимо его убрать.

3. Ссылка на информационный элемент, записанная в формате filename#id, должна быть заменена на id, поскольку в DRL все id являются глобальными.

При загрузке документа для конвертации DRL в XMI нужно выполнить обратные преобразования.

Разработанная схема загрузки и сохранения файла DRL показана на рисунке 8

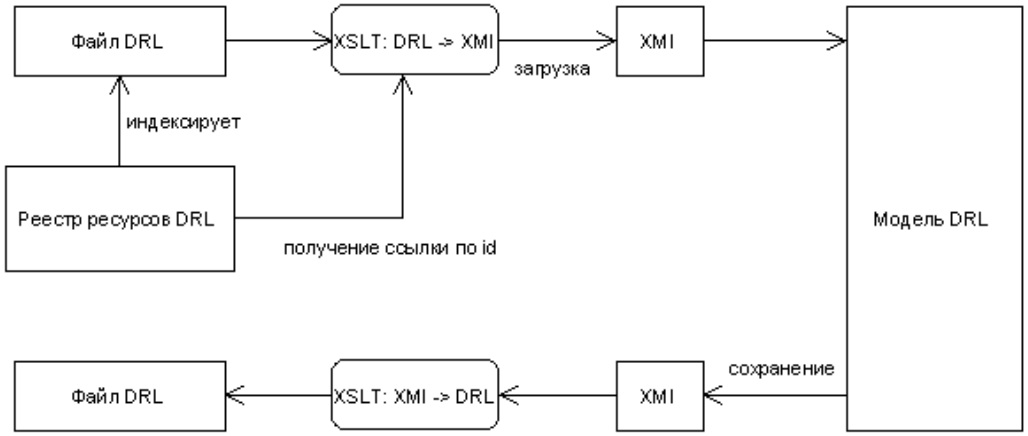

Рисунок 8: Загрузка и сохранение модели DRL

При загрузке DRL-документа сначала с помощью XSLT-трансформации над ним производятся описанные выше преобразования, после чего инициализируется модель. При этом разрешение id происходит с помощью специально разработанного компонента, интегрированного в Eclipse и анализирующего файлы DRL, - реестра ресурсов DRL. Подробнее он будет рассмотрен в разделе, посвящённом интеграции с Eclipse.

При сохранении модели она сначала сериализуется в формат XMI, после чего XSLT-трансформацией приводится к формату DRL.

3.2. Сохранение разметки и текста документа

Описанный в предыдущем пункте механизм был бы достаточен, если бы файлы DRL содержали только элементы управления структурой и вся работа с ними производилась бы исключительно посредством графических редакторов.В нашем случае задача ставится гораздо шире: графическое представление является вспомогательным в том смысле, что в файлах DRL хранится также и текст документации и её разметка; предполагается, что основное время работы над документацией технический писатель проводит именно в текстовом редакторе DRL.

Таким образом, необходимо организовать взаимодействие модели DRL с текстом в DRL-файлах, удовлетворяющее следующим требованиям:

- при загрузке модели должны сохраняться данные о форматировании файла и о расположении элементов и текста, не относящихся к модели;
- редактирование модели должно происходить с учётом введённого вручную текста и затрагивать только те элементы, которые подверглись модификации в редакторе.

Эта задача была решена с помощью введения дополнительной структуры хранения данных, называемой также «деревом документа DRL». В ней сохраняется XML-

представление DRL и работа с моделью одновременно изменяет соответствующие части документа в памяти. Таким образом выполняются оба требования к взаимодействию.

Подробнее детали реализации дерева документа обсуждаются в разделе 4.1.

3.3. Интеграция

Интегрированность получаемого средства визуального моделирования с платформой Eclipse – одно из важных достоинств при разработке с помощью Eclipse GMF.

Практическим результатом исследовательского проекта по созданию системы, поддерживающей метод разработки документации семейств программных продуктов DocLine, стал следующий набор компонентов для Eclipse:

- org.spbu.pldoctoolkit.graph – плагин, содержащий java-код модели DRL, сгенерирован с помощью EMF;
- org.spbu.pldoctoolkit.graph.edit – плагин EMF.Edit, содержит адаптеры для представления и команды для редактирования;
- org.spbu.pldoctoolkit.graph.diagram.infproduct – плагин с кодом диаграммы вариативности;
- org.spbu.pldoctoolkit.graph.diagram.productline – плагин с кодом главной диаграммы;
- org.spbu.pldoctoolkit – общая функциональность: реестр ресурсов DRL, перспектива для разработки документации;
- org.spbu.pldoctoolkit.text – плагин с кодом текстового редактора DRL, разработанного Константином Яковлевым [7];
- org.spbu.pldoctoolkit.feature – компонент, позволяющий сделать все плагины системы видимыми пользователю и управляемыми с помощью стандартного менеджера конфигурации Eclipse.

Перспектива в Eclipse – это конфигурация рабочего места разработчика: набор, размер и расположение окон, список доступных пунктов в главном, контекстных меню, меню создания элементов и меню открытия представления, набор инструментов, видимых разработчику. Число установленных плагинов в рабочей копии Eclipse разработчика может быть очень большим – перспективы позволяют настроить среду для выполнения какой-то конкретной задачи, спрятав все ненужные элементы интерфейса и опции. Возможно настроить несколько перспектив и переключаться между ними при переходе от одной задачи к другой.

Для описываемой системы разработки документации была создана специальная перспектива и включена в состав плагина org.spbu.pldoctoolkit.

Для создания документов DRL были также созданы мастера, интегрирующиеся в список, доступный по нажатию на кнопку «New...». Мастера создания диаграмм доступны в контекстном меню файлов DRL в навигаторе.

В нашей системе важным свойством визуальных редакторов также является их интеграция с редактором текста DRL – именно это позволяет им выполнять навигационную функцию.

Интеграция с текстовым редактором состоит в том, что с любого элемента любой диаграммы возможно перейти к тому месту в тексте документации, где этот элемент был определён. Для открытия именно текстового редактора DRL используется возможность открыть файл в редакторе с известным ID (ID текстового редактора DRL –

org.spbu.pldoctoolkit.editors.DRLEditor); для перехода на нужную строку в файле используется информация, получаемая из реестра ресурсов DRL.

Реестр ресурсов DRL реализован как слушатель событий в системе ресурсов Eclipse. Он создаёт и поддерживает индекс документов DRL с привязкой к проекту Eclipse. Для каждого из найденных объектов индекс содержит его идентификатор и указатель на расположение: имя файла и номер строки.

Реестр ресурсов играет ключевую роль при выполнении многих важных задач в системе, можно выделить те из них, что касаются визуального моделирования:

- найти все элементы определённого типа – используется при отображении главной диаграммы;
- по идентификатору элемента узнать имя файла и номер строки – используется при загрузке текста DRL в модель и при переходе от элемента диаграммы к его определению в тексте.

4. Особенности реализации

В этом разделе мы обсудим существенные детали реализации: сохранение форматирования и не относящегося к DRL содержимого DRL-файлов и детали устройства диаграмм.

4.1. Дерево документа DRL

Необходимости введения дополнительной структуры хранения данных обсуждалась в разделе 3.2. В этом разделе мы рассмотрим детали её реализации и интеграции с графическими редакторами.

На рисунке 9 показана схема загрузки, сохранения и изменения модели разработанных графических редакторов.

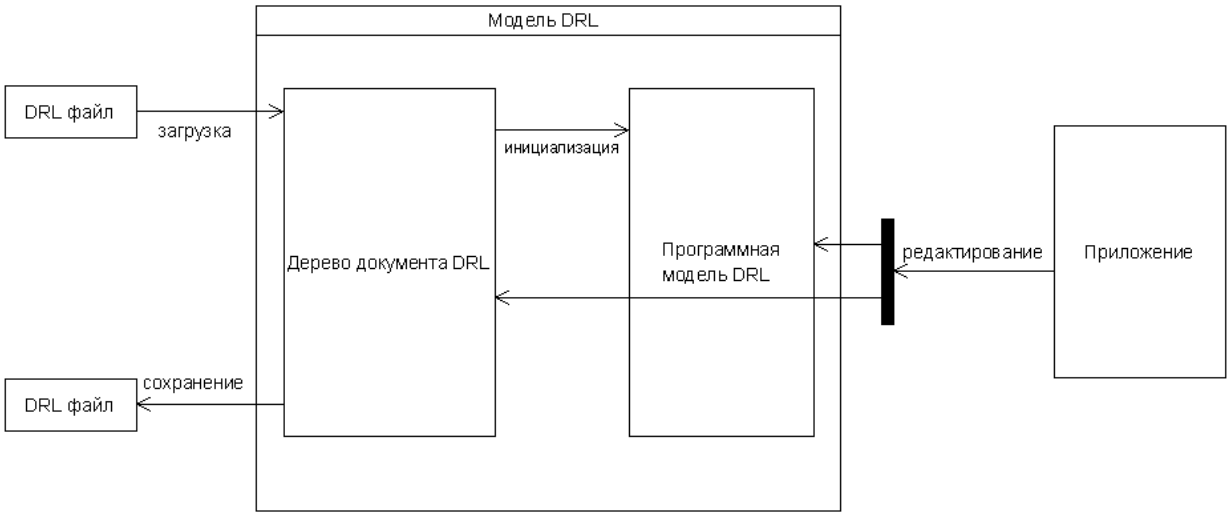

Рисунок 9: Декомпозиция модели DRL

После описанных преобразований в формат XMI, пригодный для чтения стандартными средствами EMF, данные XML сохраняются в структуре, называемой «Дерево документа DRL». Эта структура хранит содержимое документа в виде дерева DOM.

Document Object Model (DOM) – независимый от платформы и языка программирования программный интерфейс для работы с документами в формате XML и HTML. Он определяет логическую структуру документа и способ доступа и манипуляции его частями [17]. Работа с документом с помощью DOM подразумевает, что в память загружена DOM-структура документа; как структура данных она является деревом. Словосочетание «дерево DOM» обозначает эту структуру в памяти, работа с ней производится с помощью интерфейсов DOM API.

Стандартный механизм загрузки модели EMF поддерживает инициализацию из XML-документа с помощью дерева DOM, при этом дерево XML-документа последовательно просматривается вглубь и в соответствии с найденными тегами и атрибутами инициализируются элементы и связи в модели.

Модель DRL инициализируется с помощью построенного DOM-дерева, при этом в алгоритм загрузки были внесены следующие изменения:

- пропуск всех элементов, не являющихся известными открывающими или закрывающими тегами: неизвестных тегов, текстовых данных, комментариев и т.п.;
- при создании элемента модели также инициализируется ссылка на соответствующий ему узел в DOM, эта ссылка является одним из полей в классе DrlElement – общем предке для всех элементов модели.

Изменения модели, производимые из редактора, можно разделить на два вида: изменения в атрибутах элементов модели и структурные изменения модели – создание, удаление, перемещение, копирование элементов и т.п. Изменения атрибутов отражаются в DOM-дереве при синхронизации с моделью непосредственно перед сохранением на диск; редактирование структуры DOM одновременно со структурными изменениями в модели DRL было реализовано с помощью создания и подключения в редакторы собственных команд редактирования модели, наследующих стандартные команды.

Выделение исполняемых операций в классы команд – распространённый шаблон проектирования, в соответствии с которым при исполнении команды в её объекте может быть сохранена необходимая информация о контексте, которая позже может быть использована для отмены или повторения команды. Таким образом, храня в очереди объекты выполненных или готовых к выполнению команд, можно реализовать последовательность операций отмены и повторения на уровне редактора.

EMF.Edit и GMF имеют собственные реализации программных интерфейсов поддержки редактирования с использованием команд.

Были созданы следующие классы команд:

- AddCommandWrapper – наследует команду EMF.Edit AddCommand, создаёт для нового элемента модели узел в DOM-дереве и вставляет его после узла элемента, указанного параметром команды как сосед;
- MoveCommandWrapper – наследует команду EMF.Edit MoveCommand, при изменении порядка следования ссылок на элементы модели родительской в коллекции так же перемещает и узлы в DOM-дереве;
- RemoveCommandWrapper – наследует команду EMF.Edit RemoveCommand, удаляет узел дерева, соответствующий удаляемому элементу модели;
- DrlElementCreateCommand – наследует команду GMF CreateElementCommand, аналог AddCommandWrapper;
- DrlElementCreateRelationshipCommand – наследует DrlElementCreateCommand, добавляет функциональность, нужную для создания объектов ссылок (InfElemRef);
- DrlElementDestroyCommand – наследует команду GMF DestroyElementCommand, аналог RemoveCommandWrapper.

На основе команд DrlElementCreateCommand и DrlElementCreateRelationshipCommand были также созданы специализированные команды для создания объектов диаграмм.

Все команды реализованы по одной и той же схеме: при выполнении команды в её объекте сохраняется ссылка на изменяемый элемент и, если необходимо, на узел в DOM-дереве, соседствующий с узлом изменяемого элемента. Эта информация используется при выполнении отмены и повтора команды.

При запросе на сохранение модели вначале происходит синхронизация атрибутов узлов DOM-дерева со значениями атрибутов модели DRL, а затем содержимое DOM-дерева с помощью описанных в предыдущем разделе трансформаций сериализуется в DRL-файл.

4.2. Главная диаграмма

Разберём устройство главной диаграммы. Пример главной диаграммы показан на рисунке 10.

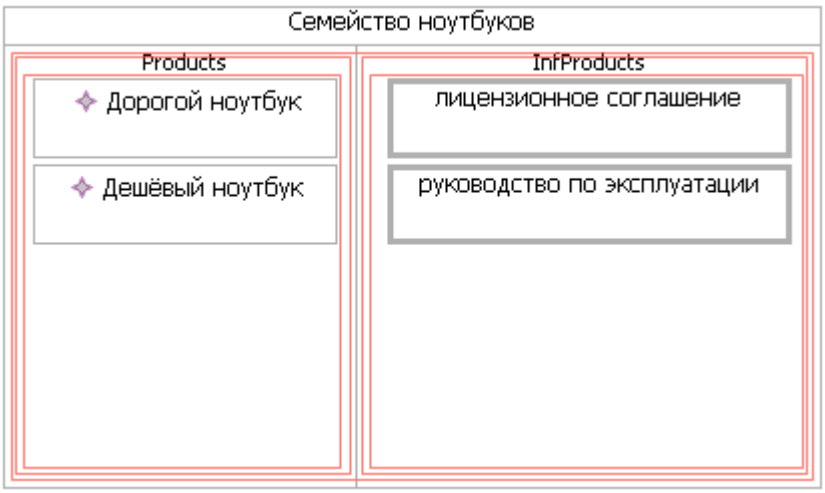

Рисунок 10: Устройство главной диаграммы (отмечены границы отсеков)

На диаграмме расположен единственный элемент верхнего уровня, соответствующий элементу ProductLine в одном из файлов документации DRL – невозможно ни удалить его, ни создать новые.

«Отсек» («Compartment») – элемент диаграммы, являющийся контейнером для расположения других элементов. Хотя фактически в документе, описывающем схему семейства, содержится только один элемент ProductLine и вложенные в него Product, организация нужного внешнего вида диаграммы потребовала создания пяти отсеков, в двух из которых расположены элементы, отвечающие продуктам и информационным продуктам, а в остальных – по одному невидимому элементу, имеющему размер, равный размеру отсека и нужному только для того, чтобы разместить в нём набор других отсеков.

Одной из сложностей реализации стало то, что эту структуру оказалось невозможно корректно выразить с помощью моделей GMF – без модификаций генерируемый по ним код не только не работает, но даже не компилируется. После углублённого изучения архитектуры и структуры кода проблему удалось решить – вручную внести необходимые правки. Их же необходимо вносить после каждой перегенерации кода.

Список информационных продуктов в правой части диаграммы не является содержимым какого-то определённого файла, в отличие от списка продуктов в левой части. Это перечень всех информационных продуктов, найденных в проекте Eclipse, которому принадлежит диаграмма – для построения этого списка используется реестр ресурсов DRL.

4.3. Диаграмма вариативности

Рисунок 11: Устройство диаграммы вариативности

Диаграмма вариативности отображает структуру информационного элемента или информационного продукта, в соответствии с графической нотацией DRL.

Создание элементов на диаграмме возможно одним из двух способов:

- либо создать новый информационный элемент или группу ссылок на информационные элементы, выбрав в панели инструментов нужный инструмент создания и щёлкнув им по элементу на диаграмме, который должен стать родительским;
- либо добавить ссылку на существующий информационный элемент, соответствующий пункт есть в контекстном меню информационного продукта и каждого информационного элемента на диаграмме.

При удаление группы ссылок на диаграмме происходит удаление объекта группы ссылок из модели и удаление всех входящих в неё ссылок на информационные элементы.

При удалении информационного элемента удаляется ссылка на него из включавшего объекта, сам элемент остаётся в модели и может быть позже включён в другой элемент из числа тех, что изображены на диаграмме.

Корневой элемент – информационный продукт или информационный элемент, изображённый в корне дерева – не может быть удалён с диаграммы.

Если один и тот же информационный элемент включён в несколько других информационных элементов, изображённых на диаграмме, то ему будет соответствовать соответствующее число копий графических элементов диаграммы и поддеревьев, отражающих его структуру. Таким образом, изображённый на диаграмме граф всегда является деревом. При редактировании одной из копий (изменение имени, добавление и удаление зависимого информационного элемента или группы ссылок) изменения сразу же отражаются и на остальных копиях.

Организация описанной древесной структуры диаграммы потребовала принципиальных изменений в алгоритме синхронизации с моделью.

В описаниях алгоритмов присутствует термин «дескриптор ребра». В стандартном случае это структура, содержащая ссылку на элементы модели, соответствующие вершинам начала и конца ребра на диаграмме.

Стандартный алгоритм синхронизации выглядит следующим образом:

1. Построить список всех элементов модели, которые должны быть изображены на диаграмме.
2. На основе этого списка удалить лишние вершины диаграммы вместе со входящими и исходящими рёбрами, создать недостающие вершины.
3. Построение списка дескрипторов рёбер диаграммы: для каждой вершины на диаграмме взять соответствующий элемент модели и добавить в список дескрипторы для исходящих рёбер.
4. Построение соответствия между элементами модели и вершинами диаграммы: обойти все вершины диаграммы, пары элемент модели - вершина диаграммы занести в хеш-таблицу.
5. Удалить рёбра, которым не соответствует ни один дескриптор.
6. Создать рёбра для дескрипторов, для которых не нашлось соответствия, вершины начала и конца получаются из таблицы, построенной на шаге 4.

Синхронизация рёбер здесь основана на таблице соответствий между элементами модели и вершинами диаграммы. В случае диаграммы вариативности такого соответствия нет: одному элементу модели может соответствовать несколько вершин.

Решение проблемы синхронизации основано на том факте, что диаграмма является деревом, поэтому каждый элемент диаграммы, кроме корневого, необходимо имеет входящее ребро, а вершина, не имеющая входящего ребра, является только что созданной.

К дескриптору ребра был добавлен указатель на вершину-начало ребра.

Для выделения корневого элемента диаграммы при её создании к метаданным соответствующей вершины добавляется строчка «root». Это позволяет найти этот элемент на первом шаге алгоритма синхронизации.

Теперь опишем модифицированный алгоритм:

1. Найти на диаграмме корневой элемент.
2. Построить список всех элементов модели, которые должны быть изображены на диаграмме, с помощью рекурсивного обхода «вширь» начиная с корневого элемента, при этом элементы, которые должны быть отображены несколько раз, столько же раз должны войти в этот список.
3. На основе этого списка удалить лишние вершины диаграммы вместе со входящими и исходящими рёбрами, создать недостающие вершины.
4. Построить список дескрипторов рёбер диаграммы тем же способом, что и в стандартном алгоритме, но в дескриптор дополнительно записывается вершина-начало ребра.
5. Удалить рёбра, которым не соответствует ни один дескриптор.-
6. Построить список новых вершин на диаграмме.

7. Создать рёбра для дескрипторов, для которых не нашлось соответствия, ссылка на вершину-начало содержится в дескрипторе, вершина-конец берётся из списка новых вершин и при этом из него исключается.

4.4. Синхронизация визуального представления и текста документации

Раундтрип между визуальным и текстовым представлением является важной характеристикой системы, позволяющей писателю пользоваться текстовым или графическим представлением поочерёдно, в зависимости от текущей задачи.

Каждой диаграмме соответствует файл, в котором хранится описывающая её графическая информация – в этом файле в формате XMI сериализована нотационная модель GMF. Поскольку файлы модели могут быть изменены в том числе с редакторов вне Eclipse, требуется определять изменения, сделанные вне визуального редактора, и обрабатывать их.

Диаграмма синхронизируется с моделью каждый раз при открытии её в редакторе – для главной диаграммы этот процесс тривиален, а для диаграммы вариативности был описан выше.

Для обнаружения внешних изменений редактор диаграмм регистрирует обработчик событий в файловой системе Eclipse.

При каждой активации редактора с открытой диаграммой происходит проверка, не был ли изменён файл модели и если он был изменён, то дальнейшие действия зависят от того, есть ли на диаграмме несохранённые пользовательские изменения:

- если изменений нет, то она синхронизируется с моделью;
- иначе пользователю показывают диалоговое окно, предлагающее отменить изменения и синхронизировать диаграмму с моделью либо продолжить работу.

5. Пример

В качестве примера документации семейства программных продуктов, разработанной в соответствии с подходом DocLine, мы рассмотрим специально разработанную документацию для гипотетического семейства средств защиты компьютера от сетевых угроз «ИнтерБез». Мы рассмотрим только часть этой документации, непосредственно касающуюся визуальных моделей.

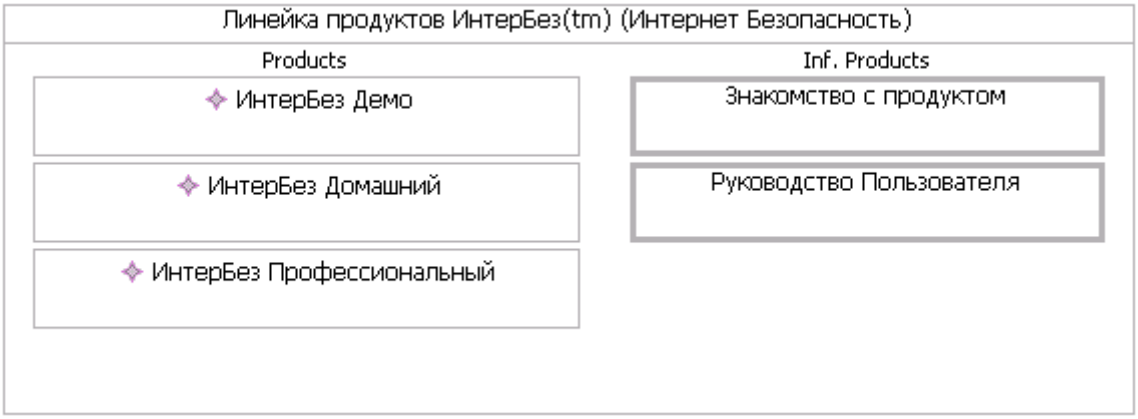

Рисунок 12: Диаграмма семейства линейки продуктов ИнтерБез

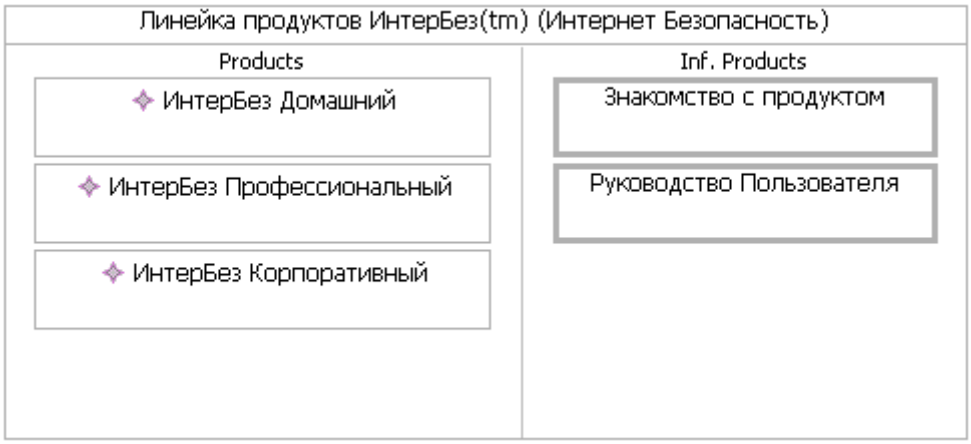

Рисунок 13: Главная диаграмма семейства ИнтерБез

Семейство включает в себя три продукта: «ИнтерБез Домашний», «ИнтерБез Профессиональный» и «ИнтерБез Корпоративный». Различия между ними состоят в функциональности – в «Домашней» версии она крайне урезана; «Корпоративная» версия поддерживает получение обновлений с сервера, расположенного в локальной сети.

В качестве информационных продуктов представлены «Знакомство с продуктом» – короткий обзор системы, и «Руководство пользователя» – полная документация на систему, содержащая данные о работе функционального блока только если этот блок входит в эту версию продукта.

Файл DRL, в котором описано семейство продуктов, выглядит следующим образом:

```
<d:ProductLine name="Линейка продуктов ИнтерБез(tm) (Интернет
Безопасность)" xmlns:d="http://math.spbu.ru/drl">
 <d:Product id="demo" name="ИнтерБез Домашний"/>
 <d:Product id="home" name="ИнтерБез Профессиональный"/>
 <d:Product id="professional" name="ИнтерБез Корпоративный"/>
```

```
</d:ProductLine>
```

Листинг 6: пример файла с описанием семейства

Комплекс средств включает в себя брандмауэр, антиспам и антивирус, а также центр управления для централизованного управления продуктом. Центр управления и брандмауэр являются обязательными компонентами; антивирус и антиспам обязательными не являются, но в системе должен присутствовать хотя бы один из них.

Комплекс поддерживает две схемы обновления по сети: с сайта компании, в случае версий «Домашний» или «Профессиональный», или с выделенного сервера, находящегося в локальной сети, как в версии «Корпоративный». Будем считать, что в продукте может быть реализована только одна из этих схем.

На рисунке 13 показана схема документации семейств программных продуктов ИнтерБез.

Рисунок 13: Диаграмма вариативности руководства пользователя

Рисунок 13: Диаграмма вариативности руководства пользователя для семейства ИнтерБез

Описанные выше ограничения касаются и структуры документации. Ссылки на описания антиспама и антивируса соединены в группу OR – это и означает, что в конечном итоге в документе должна присутствовать хотя бы одна из этих частей. Ссылки на описание обновления с локального сервера или через сервер централизованных обновлений объединены в группу XOR, что делает их взаимоисключающими.

Текст DRL, отвечающей этой диаграмме, хранится в 2х разных файлах. В одном из них описан информационный продукт «Руководство Пользователя», а в другом – набор общих частей документации.

Файл с описанием информационного продукта «Руководства пользователя» и описание общих для проекта информационных элементов вынесен как приложение в диплому.

После проектирования и наполнения текстом документация может быть опубликована в одном из форматов pdf или html – система публикации является частью дипломной работы Константина Яковлева [7]. На рисунке 14 представлен фрагмент документации к версии «Профессиональный» в формате pdf.

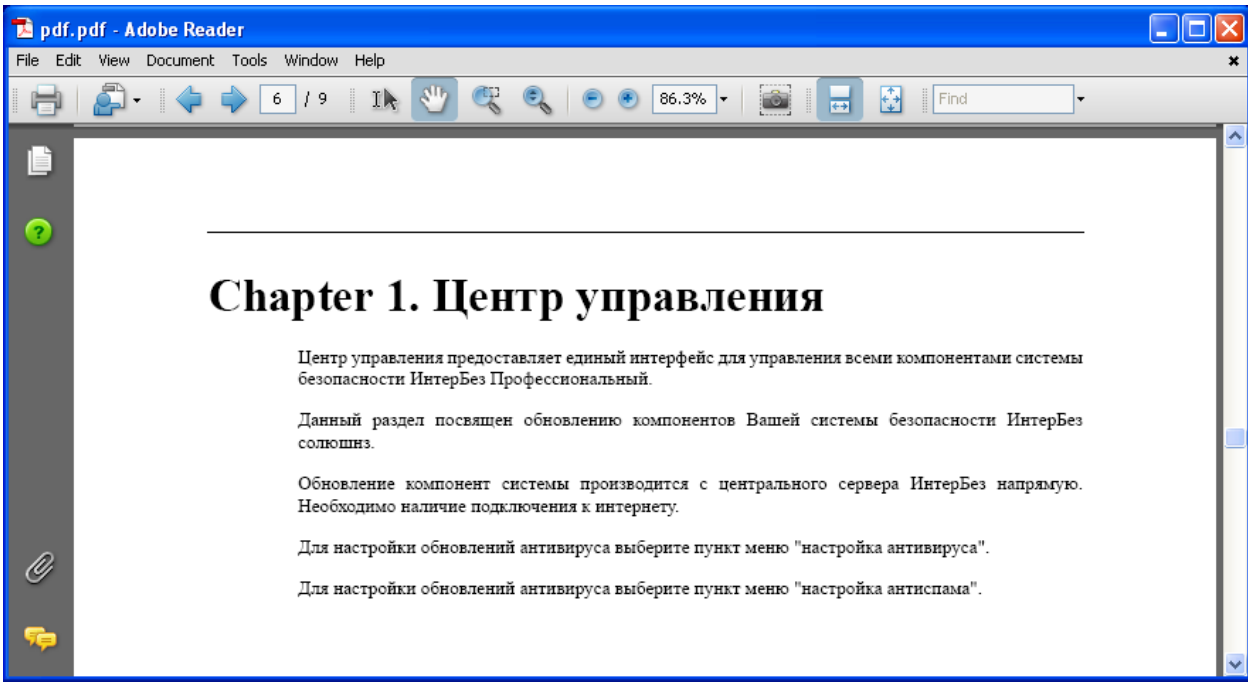

Рисунок 14: Документация в формате pdf

Таким образом, разработанная система поддерживает полный цикл работы с документацией семейств программных продуктов: проектирование, создание, редактирование и публикацию.

6. Заключение

В рамках этой работы были достигнуты следующие результаты:

- разработана спецификация графического редактора DRL,
- спроектирована метамодель DRL с помощью Eclipse EMF,
- реализован механизм загрузки и сохранения между метамоделью DRL и файлами документации DRL,
- реализованы графические редакторы диаграммы семейства и диаграммы информационного продукта,
- реализована интеграция графических редакторов с текстовым XML-редактором DRL [7],
- разработан пример, демонстрирующий возможности созданной системы.

7. Литература

[1] Clements, P., Northrop, L. Software Product Lines: Practices and Patterns. - Boston, MA: Addison-Wesley, 2002. - 608 p.

[2] Романовский К.Ю. Метод разработки документации семейств программных продуктов. // Системное программирование. Вып. 2. Сб. статей / Под ред. А.Н.Терехова, Д.Ю.Булычева. СПб.: Изд-во СПбГУ, 2007.

[3] Clark D. Rhetoric of Present Single-Sourcing Methodologies. // SIGDOC'02, October 20-23, 2002, Toronto, Ontario, Canada. - 2002.- P. 20-25

[4] OASIS DITA Technical Committee official site. // http://www.oasis-open.org/committees/dita

[5] Czarnecki K., Eisenecker U., Generative Programming: Methods, Tools and Applications. - Reading, Mass.: Addison Wesley Longman, 2000. - 864 p.

[6] Павлинов А., Кознов Д., Перегудов, Бугайченко Д., Казакова А., Чернятчик Р., Фесенко Т., Иванов А. Комплекс средств для реализации предметно-ориентированных визуальных языков.

[7] Яковлев К. Создание среды разработки документации для семейств программных продуктов, Дипломная работа, СПбГУ, Математико-Механический факультет, кафедра системного программирования, 2007.

[8] Clements, P., Northrop, L. A Framework for Software Product Line Practice. - Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. http://www.sei.cmu.edu/productlines/framework.html, 2006.

[9] Ari Jaaksi. Developing Mobile Browsers in a Product Line. // IEEE Software, Jul./Aug. 2002, pp. 73-80.

[10] Kang K., Cohen S., Hess J., Novak J., et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study. // Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990

[11] Microsoft DSL Tools project official site // http://msdn.microsoft.com/vstudio/DSLTools/

[12] Eclipse project official site. // http://www.eclipse.org

[13] EclipseCon official site. // http://www.eclipsecon.org/

[14] Shatalin A., Tikhomirov A. GMF Architecture Overview. Technical Report, EclipseCon, 10 Sept. 2006

[15] Романовский К.Ю., Кознов Д.В. Язык DRL для проектирования и разработки документации семейств программных продуктов. // Вестник С.-Петербург. ун-та. Сер. 10.2007. Вып. 4. С.

[16] The DocBook Project. // http://docbook.sourceforge.net

[17] Document Object Model (DOM) Level 2 Core Specification // http://www.w3.org/TR/DOM-Level-2-Core/

[18] Antony Hunter, Mohammed Mostafa. Extending your DSM by leveraging the GMF Runtime. // EclipseCon 2007, http://www.eclipsecon.org/2007/

Преобразование из DRL в XMI

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:v="http://math.spbu.ru/drl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:resolver="java:org.spbu.pldoctoolkit.graph.util.IdUtil"
    xmlns:saxon="http://icl.com/saxon"
    extension-element-prefixes="resolver saxon"
    version="2.0">

    <xsl:param name="project-name"/>

    <xsl:template match="v:Product">
        <products>
            <xsl:apply-templates select="node() | attribute() | text() |
comment()"/>
        </products>
    </xsl:template>

    <xsl:template match="v:InfElement | v:InfProduct">
        <xsl:variable name="type" select="name()"/>
        <xsl:element name="parts">
            <xsl:apply-templates select="attribute()"/>
            <xsl:attribute name="xsi:type"><xsl:value-of
select="$type"/></xsl:attribute>
            <xsl:apply-templates select="node() | text() | comment()"/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="v:InfElemRef">
        <infElemRefs>
            <xsl:apply-templates select="node() | attribute() | text() |
comment()"/>
        </infElemRefs>
    </xsl:template>

    <xsl:template match="v:InfElemRefGroup">
        <groups>
            <xsl:apply-templates select="node() | attribute() | text() |
comment()"/>
        </groups>
    </xsl:template>

    <!-- infelem id reference -->
    <xsl:template match="@infelemid">
        <xsl:variable name="id"><xsl:value-of select="."/></xsl:variable>
        <xsl:attribute name="infelem">
            <xsl:value-of select="resolver:idToUriString($project-name, $id)"/>
        </xsl:attribute>
    </xsl:template>

    <!-- InfElemRef -> group reference -->
    <xsl:template match="@groupid">
        <xsl:variable name="id"><xsl:value-of select="."/></xsl:variable>
        <xsl:choose>
            <xsl:when test="$id = ''">
                <xsl:attribute name="group"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:attribute name="group">
                    <xsl:value-of select="concat('#', $id)"/>
                </xsl:attribute>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:template>
```

```
    <xsl:template match="node() | attribute() | text() | comment()">
        <xsl:copy>
            <xsl:apply-templates select="node() | attribute() | text() |
comment()"/>
        </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

Преобразование из XMI в DRL

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:v="http://math.spbu.ru/drl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:resolver="java:org.spbu.pldoctoolkit.graph.util.IdUtil"
    version="2.0"
    extension-element-prefixes="resolver"
    exclude-result-prefixes="xsi">

    <xsl:output indent="yes"/>

    <xsl:template match="products">
        <v:Product>
            <xsl:apply-templates select="node() | attribute() | text() |
comment()"/>
        </v:Product>
    </xsl:template>

    <xsl:template match="parts">
        <xsl:variable name="type"><xsl:value-of
select="@xsi:type"/></xsl:variable>
        <xsl:element name="{$type}">
            <xsl:copy-of select="@*[not(name()='xsi:type')]"/>
            <xsl:apply-templates select="text() | comment() | node()"/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="infElemRefs">
        <v:InfElemRef>
            <xsl:apply-templates select="node() | attribute() | text() |
comment()"/>
        </v:InfElemRef>
    </xsl:template>

    <xsl:template match="groups">
        <v:InfElemRefGroup>
            <xsl:apply-templates select="node() | attribute() | text() |
comment()"/>
        </v:InfElemRefGroup>
    </xsl:template>

    <!-- infelem id reference -->
    <xsl:template match="@infelem">
        <xsl:variable name="uri"><xsl:value-of select="."/></xsl:variable>
        <xsl:attribute name="infelemid">
            <xsl:value-of select="resolver:uriStringToId($uri)"/>
        </xsl:attribute>
    </xsl:template>

    <!-- InfElemRef -> group reference -->
    <xsl:template match="@group">
        <xsl:variable name="uri"><xsl:value-of select="."/></xsl:variable>
        <xsl:attribute name="groupid">
            <xsl:value-of select="resolver:uriStringToId($uri)"/>
        </xsl:attribute>
    </xsl:template>

    <xsl:template match="node() | attribute() | text() | comment()">
        <xsl:copy>
            <xsl:apply-templates select="node() | attribute() | text() |
comment()"/>
        </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

Приложение 1. Пример файла с общими элементами DRL

```
<?xml version="1.0" encoding="UTF-8"?>
<d:DocumentationCore xmlns:d="http://math.spbu.ru/drl"
xmlns="http://docbook.org/ns/docbook">
 <d:InfElement id="about_company" name="О компании">
       <para>
             Компания <d:DictRef dictid="company" entryid="name"/> занимается
разработкой комплексных
             средств защиты от сетевых угроз с 1990 года и на данный момент
является безусловным лидером
             в этой области.
       </para>
 </d:InfElement>

 <d:InfElement id="preface" name="Вступление">
       <preface>
             <title>Вступление</title>
             <para>
                   Благодарим за приобретение <d:DictRef dictid="main"
entryid="productname"/>!
             </para>
             <d:InfElemRef id="ref_about_company" infelemid="about_company"/>
       </preface>
 </d:InfElement>

 <d:InfElement id="firewall" name="Брандмауэр">
       <chapter>
             <title>Брандмауэр</title>
             <d:InfElemRef id="ref_firewall_description"
infelemid="firewall_description"/>
             <d:InfElemRef id="ref_firewall_config" infelemid="firewall_config"
optional="true"/>
       </chapter>
 </d:InfElement>

 <d:InfElement id="firewall_description" name="Что такое брандмауэр">
       <para>
             Брандмауэр (firewall) - перегородка из огнеупорного материала,
             возводимая на пути распространения пожара. Также данный термин
             стал использоваться для обозначения аппаратных и программных
             средств сетевой защиты (сетевой экран). Пожалуй, такое название
             было выбрано из маркетинговых соображений: "Пусть за стеной
             бушует пожар или беснуются варвары, но вам за надежной защитой
             ничего не грозит". В действительности сетевой экран походит не
             на сплошную стену, а на таможенный пост, где в соответствии с
             предписаниями проверяется багаж путешественников. Если груз к
             ввозу или вывозу разрешен, его пропускают. Если нет - извините.
             Причем решения принимаются на основе адресов
получателя/отправителя,
             а не содержимого груза. Так что доверенный отправитель может
             отправить своему визави не только поздравительную открытку, но и
             грамм триста тротилового эквивалента. Таможня "посылочку"
             пропустит, если связь разрешена в предписании.
       </para>
 </d:InfElement>

 <d:InfElement id="firewall_config" name="Конфигурирование брандмауэра">
       <para>
             Конфигурация брандмауэра полностью автоматическая.
             В случае возникновения трудностей обращайтесь в нашу Службу
Поддержки!
       </para>
 </d:InfElement>
```

```
 <d:InfElement id="antivirus" name="Анти-вирус">
       <chapter>
             <title>Антивирус</title>
             <d:InfElemRef id="ref_antivirus_description"
infelemid="antivirus_description"/>
             <d:InfElemRef id="ref_antivirus_config"
infelemid="antivirus_config" optional="true"/>
             <para>
                   Действия при обнаружении вируса или другой угрозы.
                   Доступны следующие виды действий:
                   <itemizedlist mark="opencircle">
                         <listitem><para>удаление</para></listitem>
                         <listitem><para>просмотр журнала
событий</para></listitem>
                         <d:Nest id="extended_actions"/>
                   </itemizedlist>
             </para>
       </chapter>
 </d:InfElement>

 <d:InfElement id="antivirus_description" name="Что такое анти-вирус">
       <para>
             Антивирусная программа (антивирус) — программа для обнаружения и
удаления компьютерных вирусов и других вредоносных программ, предотвращения их
распространения, а также лечения программ зараженных ими.
       </para>
       <para>
             Первые, наиболее простые антивирусные программы появились почти
сразу после появления вирусов. Сейчас разработкой антивирусов занимаются крупные
компании. Как и у создателей вирусов, в этой сфере также сформировались
оригинальные приемы - но уже для поиска и борьбы с вирусами. Современные
антивирусные программы могут обнаруживать десятки тысяч вирусов.
             Основные задачи современных антивирусных программ:
             <itemizedlist mark="opencircle">
                   <listitem><para>Сканирование файлов и программ в режиме
реального времени.</para></listitem>
                   <listitem><para>Сканирование компьютера по
требованию.</para></listitem>
                   <listitem><para>Сканирование интернет-
трафика.</para></listitem>
                   <listitem><para>Сканирование электронной
почты.</para></listitem>
                   <listitem><para>Защита от атак враждебных веб-
узлов.</para></listitem>
                   <listitem><para>Восстановление поврежденных файлов
(лечение).</para></listitem>
             </itemizedlist>
       </para>
 </d:InfElement>

 <d:InfElement id="antivirus_config" name="Конфигурирование антивируса">
       <para>
             Конфигурация антивируса полностью автоматическая. В случае
возникновения трудностей обращайтесь в нашу
             Службу поддержки!
       </para>
 </d:InfElement>

 <d:InfElement id="antispam" name="Анти-спам">
       <chapter>
             <title>Анти-спам</title>
             <para>
                   Компонент Анти-спам встраивается в установленный на Вашем
компьютере
                   почтовый клиент и контролирует все поступающие почтовые
сообщения на предмет спама.
                   Все письма, содержащие спам, помечаются специальным
заголовком.
```

```
                    Предусмотрена также возможность настройки Анти-спама на
обработку спама
                    (автоматическое удаление, помещение в специальную папку и
т.д.).
              </para>
        </chapter>
 </d:InfElement>

 <d:InfElement id="control_center" name="Центр управления">
        <chapter>
              <title>Центр управления</title>
              <para>
                    Центр управления предоставляет единый интерфейс для
управления всеми
                    компонентами системы безопасности <d:DictRef dictid="main"
entryid="productname"/>.
              </para>
              <d:InfElemRef id="ref_components_update"
infelemid="components_update"/>
        </chapter>
 </d:InfElement>

 <d:InfElement id="components_update" name="Обновление компонент">
        <para>
              Данный раздел посвящен обновлению компонентов Вашей системы
безопасности <d:DictRef dictid="company" entryid="name"/>.
        </para>
        <d:InfElemRefGroup id="update_methods" modifier="XOR" name="способы
обновления"/>
        <d:InfElemRef groupid="update_methods" id="ref_update_local"
infelemid="update_local" optional="true"/>
        <d:InfElemRef groupid="update_methods" id="ref_update_inet"
infelemid="update_inet" optional="true"/>
        <d:Nest id="av-update">
              <para>
                    Для настройки обновлений антивируса выберите пункт меню
"настройка антивируса".
              </para>
        </d:Nest>
        <d:Nest id="as-update">
              <para>
                    Для настройки обновлений антивируса выберите пункт меню
"настройка антиспама".
              </para>
        </d:Nest>
 </d:InfElement>

 <d:InfElement id="update_inet" name="Централизованные обновления">
        <para>
              Обновление компонент системы производится с центрального сервера
ИнтерБез напрямую.
              Необходимо наличие подключения к интернету.
        </para>
 </d:InfElement>

 <d:InfElement id="update_local" name="Обновления с сервера компании">
        <para>
              Обновление компонент системы производится с сервера локальной сети
Вашей компании.
              Подключение к интернету не требуется.
        </para>
 </d:InfElement>

 <d:Dictionary id="company" name="Глобальный словарь">
        <d:Entry id="name">ИнтерБез солюшнз</d:Entry>
 </d:Dictionary>
</d:DocumentationCore>
```

Приложение 2. Пример файла с описанием информационного продукта

```
<?xml version="1.0" encoding="UTF-8"?>
<d:DocumentationCore xmlns:d="http://math.spbu.ru/drl"
xmlns="http://docbook.org/ns/docbook">
 <d:InfProduct id="UserManual" name="Руководство Пользователя">
        <book>
               <bookinfo>
                     <title>Руководство Пользователя <d:DictRef dictid="main"
entryid="productname"/></title>
               </bookinfo>
               <d:InfElemRef id="ref_preface" infelemid="preface"/>
               <d:InfElemRef id="ref_control_center" infelemid="control_center"/>
               <d:InfElemRef id="ref_firewall" infelemid="firewall"/>

               <d:InfElemRefGroup id="optional_components" modifier="OR"
name="опции"/>
               <d:InfElemRef groupid="optional_components" id="ref_antivirus"
infelemid="antivirus" optional="true"/>
               <d:InfElemRef groupid="optional_components" id="ref_antispam"
infelemid="antispam" optional="true"/>
        </book>
 </d:InfProduct>
</d:DocumentationCore>
```