САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

На правах рукописи

Романовский
Константин Юрьевич

МЕТОД ПОВТОРНОГО ИСПОЛЬЗОВАНИЯ ДОКУМЕНТАЦИИ СЕМЕЙСТВ ПРОГРАММНЫХ ПРОДУКТОВ

05.13.11 – математическое и программное обеспечение
вычислительных машин, комплексов, систем и сетей

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научный руководитель –
к.ф.-м.н., доцент
Кознов Д.В.

Санкт-Петербург
2010

Содержание

Введение

В современных проектах разработки промышленного программного обеспечения возникает множество задач, не менее важных, чем, собственно, само программирование. Одна из таких задач – это разработка документации. Распространенным классом технических документов является пользовательская документация ПО. Сложность современных программных комплексов (структурная сложность, изменчивость, многоверсионность, многообразие ролей пользователей, обилие функций, задачи администрирования и т.д.), а также необходимость их сопровождения на протяжении многих лет – все это предъявляет высокие требования к документации и к процессу ее разработки. Зачастую для одного программного продукта создается целый набор документов, описывающих его с различных сторон, например, руководство по быстрому старту, подробное руководство пользователя, справочник функций, встроенная справочная система, сайт поддержки, учебные материалы, руководство администратора. Все эти документы должны сопровождаться вместе с ПО, поскольку изменения ПО, затрагивающие функциональность или пользовательский интерфейс, должны быть отражены в документации.

Научная общественность активно занимается вопросами разработки технической документации. Одно из наиболее известных сообществ в этой области – ассоциация ACM SIGDOC (Association for Computer Machinery’s Special Interest Group on the Design of Communication) [53]. В рамках этого сообщества проводятся ежегодные конференции, охватывающие широкий круг вопросов: специализированные языки и средства разработки электронной документации, качество («понятность») текстов, особенности перевода технических документов на иностранные языки, принципы разработки, сопровождения и обеспечения целостности больших пакетов документов и т.д.

Разработчики инструментального ПО также уделяют внимание задачам разработки технической документации. Существует целый спектр программных средств, обеспечивающих разработку как простых, «одиночных» документов, так и масштабных пакетов документов. Простая документация разрабатывается в редакторах общего назначения, таких как Microsoft Word. Преимущество подобных редакторов – в их простоте и удобстве: практически, любой пользователь ПК способен быстро освоить процедуру создания «одиночных» документов в подобных системах. Для более сложного ПО необходимы средства разработки пакетов документов, поддерживающие многоязычность, версионирование документов для различных операционных систем, множественные варианты конечного представления (PDF для печатного руководства, HTML Help для встроенной справочной системы и т.п.). Для таких задач используются специализированные методы и средства, такие как FrameMaker [57] (издательская система компании Adobe), DocBook [51] (open-source стандарт в области разработки документации для Unix/Linux-приложений), DITA [54] (метод разработки сложной модульной документации компании IBM) и другие.

В индустрии разработки ПО активно развивается подход повторного использования [42], основная идея которого состоит в том, что в ПО выделяются фрагменты, которые могут быть использованы неоднократно. Такие фрагменты обосабливаются в отдельные компоненты и затем ПО строится из них. Для разработчиков ПО существует множество подходов организации повторного использования кода, например параграфы в COBOL, функции в процедурных языках, классы и компоненты в объектно-ориентированных языках. Одно время в индустрии считали, что можно разработать набор типовых строительных блоков и потом все новые программные продукты собирать из этих блоков, причем предполагалось, что с такой работой может справиться неспециалист, который будет просто выбирать блоки, нужные для своей задачи. Однако разработка универсальных компонент «на все случаи жизни» оказалась довольно сложным

и дорогостоящим занятием – универсальность ограничивала функциональность, и наоборот – развитая специфическая функциональность сужала сферу применения компонент. В этих условиях важнейшим направлением развития механизмов повторного использования становится поддержка адаптивности повторно-используемых блоков. Адаптивность означает возможность «подстраивать» переиспользуемые компоненты к особенностям конкретного контекста использования без влияния на остальные контексты. Примеры механизмов адаптивности – параметризованные функций в процедурном подходе и наследование с полиморфизмом в объектно-ориентированном подходе. Отметим также универсальный метод адаптивного повторного использования, предложенный Полом Бассетом – метод фреймов [16] – на основе которого Стен Ерзабек разработал язык XVCL [36]. Идея метода фреймов и XVCL применима для организации повторного использования произвольной текстовой информации, однако, фактически, метод фреймов оптимизирован для программ на языке COBOL, а XVCL – для программ на языке Java.

Важной вехой в развитии методов повторного использования была идея Дэвида Парнаса [44] относительно эффективности совместной разработки нескольких схожих программных систем. Эта идея легла в основу подхода разработки *семейств программных продуктов* (далее – СПП), подразумевающего совместную разработку набора продуктов, обладающих общими свойствами, совместно продвигаемых на рынке и создаваемых на основе повторно используемых активов в рамках определенного, заданного процесса разработки [24]. Таким образом универсальность повторно используемых компонент ограничивается рамками СПП, разрабатываемого в одной компанией. При таком ограничении можно достичь приемлемого баланса универсальности и функциональности. В самом деле, повторно используемые компоненты применяются в ограниченном наборе контекстов, при этом их исходные тексты доступны и могут

быть оперативно модифицированы в случае необходимости поддержки новых контекстов.

Вернемся к задаче разработки документации. В больших пакетах пользовательской документации встречается много фрагментов текста, которые используются почти без изменений в различных контекстах. Это дает основание полагать, что в разработке документации также возможно применять повторное использование. Ряд методов разработки документации – DocBook [51], DITA [54] и др. – поддерживают повторное использование фрагментов текста. Однако на практике фрагменты повторяются не полностью идентично, а с некоторыми модификациями, что значительно осложняет поддержку повторного использования. На сегодняшний момент поддерживается лишь простая адаптивность – условное включение фрагмента. Этот механизм можно использовать, например, для обработки незначительных различий поведения продукта в разных операционных системах, однако в целом повторно-используемые фрагменты должны быть идентичны во всех контекстах использования, что сильно ограничивает возможности повторного использования. Кроме того, имеется еще одна особенность – тексты предназначаются в первую очередь для человека, вследствие чего очень важную роль играет понятность документации. Поэтому одну и ту же функциональность часто описывают разными словами. В результате при разработке документации применяют простейший метод повторного использования – копирование и исправление (copy/paste/modify): нужный фрагмент копируется и затем модифицируется под требования контекста. Этот метод хорошо подходит для документов, которые создаются «раз и навсегда» и не требуют дальнейшего сопровождения. Однако в случае, когда сопровождение все-таки требуется, любые изменения (расширения, исправление ошибок и т.п.) необходимо вносить независимо во все копии, поскольку при копировании утрачивается связь между исходным фрагментом и его копией.

В случае документации для СПП для каждого продукта создается стандартный пакет документов и появляется необходимость повторного использования значительных по размерам фрагментов текста между документами для разных продуктов. К сожалению, в рамках существующих подходов к разработке СПП создание и поддержка пользовательской документации ПО не выделяется в отдельную задачу, и подходящие методы и инструментальные средства отсутствуют. В отдельных отчетах об индустриальных внедрениях подхода СПП упоминается о важности повторного использования документации (например, в [35]), однако не предлагается соответствующих методов и средств.

В данной работе предложен новый метод разработки документации семейств программных продуктов DocLine (основные идеи DocLine изложены в работах [8], [9]). Данный метод восполняет разрыв между методами разработки семейств программных продуктов и подходами разработки технической документации. DocLine охватывает весь жизненный цикл разработки документации от проектирования до публикации итоговых документов и поддерживает плановое адаптивное повторное использование. В DocLine явно выделяется исходное и целевое представление документации. Целевое представление – это документы в привычных для пользователя форматах, таких как PDF для печатных документов, HTML для электронных или HTML Help для справочных систем. Целевое представление может быть в любой момент получено из исходного автоматически – эта операция называется публикацией, подобно компиляции исполняемого кода ПО из его исходных текстов.

Для исходного представления документации в DocLine предлагается оригинальный проблемно-ориентированный язык DRL (Documentation Reuse Language) [11]. DRL, подобно другому проблемно-ориентированному языку SDL (Specification and Description Language, [34]), имеет две нотации – графическую (DRL/GR – Graphic Representation) и текстовую (DRL/PR – Phrase Representation). Графическое представление служит для проектирования структуры по-

вторного использования документации. Текстовое представление позволяет описать варианты конфигурирования повторно используемых компонент и, собственно, сами конкретные конфигурации для порождения конечных документов.

Наряду с языком DRL метод DocLine определяет также эталонные модели процесса разработки документации – проактивную и «гибкую». В рамках проактивной модели сначала проводится проектирование схемы повторного использования и разработка адаптивных повторно-используемых компонент, а затем на основе созданной инфраструктуры создаются пакеты документов для конкретных продуктов. В «гибком» процессе сначала создаются требуемые документы, а затем, по мере необходимости, документация подвергается рефакторингу [7], [47]. То есть на основе конкретных документов стоится инфраструктура повторного использования и исходное представление документов перестраивается на использование этой инфраструктуры, при этом целевое представление остается неизменным.

Для практического применения DocLine предложена архитектура пакета инструментальных средств, на основе которой реализована версия пакета, встроенная в интегрированную среду разработки приложений Eclipse. Также была реализована базовая версия пакета на платформе Microsoft.NET. Для порождения документов в различных целевых форматах (PDF и HTML) DocLine интегрирован с популярной технологией DocBook.

Метод DocLine был апробирован на примере пользовательской документации семейства телекоммуникационных систем, разрабатываемого ЗАО «Ланит-Терком». Продукты семейства – цифровые телефонные станции типа «Квант-Е» различного назначения – офисные, сельские, городские, транзитные и др. Объектом апробации стали руководства пользователя для двух разных продуктов семейства (общим объемом около 300 страниц). Были выделены общие для двух продуктов активы, затем сами руководства были переработаны для сов-

местного использования общих активов. Еще одна апробация была выполнена для документации системы поддержки телевизионного вещания компании ДИП [6]. В семейство входят разнообразные устройства, предназначенные для воспроизведения видео, микширования и т.п. В ходе апробации была разработана документация для нескольких устройств с выделением повторно-используемых фрагментов.

Данная работа выполнена в рамках исследовательского проекта кафедры системного программирования Санкт-Петербургского государственного университета под руководством доцента кафедры к.ф.-м.н. Д. Кознова. Д. Кознов предложил использовать визуальное моделирование для проектирования повторного использования документации и выдвинул идею «гибкого» процесса разработки и рефакторинга. К. Романовский создал метод разработки документации, эталонные модели процесса, разработал и специфицировал язык DRL, спроектировал и разработал инструментальные средства поддержки, а также расширил подход рефакторинга и предложил набор автоматизированных операций рефакторинга. Под его руководством на кафедре системного программирования в рамках данного проекта было выполнено несколько дипломных работ.

Автор выражает благодарность научному руководителю и всем участникам этого проекта. Также автор выражает признательность проф. А. Терехову за методическую помощь и конструктивную критику предлагаемых идей; Т. Поповой, Б. Федотову, А. Тиуновой, А. Ежикову за помощь в понимании потребностей предметной области; А. Перегудову и Б. Любимову за содействие в проведении апробации, а также студентам математико-механического факультета, активно участвовавшим в проекте DocLine: А. Семенову, К. Яковлеву, Л. Минчину, К. Вьюшковой, С. Василинцу, И. Чалову, В. Дорохову, А. Голубеву и Н. Соколову. Особую признательность хочется выразить Т. Дроздовой, которая оказала неоценимую помощь при апробации инструментальных средств, вы-

ступив в роли технического писателя и первого пользователя еще «сырого» продукта.

Исследование было поддержано Российским фондом фундаментальных исследований (гранты РФФИ 08-01-00716-а, 08-07-08066-з), Фондом содействия развитию малых форм предприятий в научно-технической сфере (программа СТАРТ), ЗАО «Ланит-Терком», лабораторией СПРИНТ, организованной компаний «Интел» на математико-механическом факультете СПбГУ. На пакет программных средств получено свидетельство о государственной регистрации программы для ЭВМ №2008612676, выданное 28 апреля 2008 г. Федеральной службой по интеллектуальной собственности, патентам и товарным знакам.

Глава 1. Обзор существующих подходов

В этом разделе рассматриваются основные существующие технологии разработки повторно-используемой технической документации, с акцентом на возможности повторного использования. Также затронуты подход повторного использования и разработки СПП, понятие рефакторинга, описывается открытая интегрированная среда разработки Eclipse, в которую встраивается инструментальный пакет DocLine. Наконец, приводится формализм спецификации языков, который мы используем для формального описания DRL.

1.1 Повторное использование

Различают два вида повторного использования – случайное и плановое [42]. Случайное повторное использование предполагает, что разработчики подключают существующие компоненты, когда и если представляется такая возможность. Плановое повторное использование предполагает систематическую разработку компонент, подготовленных для повторного использования и дальнейшее применение таких компонент в разработке. Далее мы будем рассматривать плановое повторное использование, а именно рассмотрим универсальные методы повторного использования произвольной текстовой информации (фреймы Бассета и XVCL), а также диаграммы возможностей – подход управления повторным использованием.

1.1.1 Метод Бассета-Ерзабека

Фреймы Бассета [15] – это универсальный подход к повторному использованию произвольной информации, разработанный в 1980-х годах Полом Бассетом в университете Йорк (г. Торонто, Канада). Пол Бассет проанализировал различные подходы к повторному использованию информации, а также практику их применения, и пришел к выводу, что в различных контекстах переиспользуемые модули должны различаться – такое повторное использование

называется ***адаптивным***. Информация в методе Бассета представляется в виде архетипа и набора дельт. ***Архетип*** – это модель или шаблон, общее в разных информационных объектах, ***дельта*** – это различия объектов.

Метод Бассета позволяет организовывать повторное использование произвольных данных, даже если их формат сам по себе не предусматривает такой возможности. Для этого поверх собственной структуры данных строится специальная разметка, позволяющая выделить архетипы, которые называются фреймами, и описать изменения, требуемые в различных контекстах использования (дельты).

Существует несколько вариантов языка разметки, реализующего метод фреймов. Их объединяет общий набор основных конструкций – фрейм (архетип), точки расширения (предусмотренные позиции, в которых архетип может быть модифицирован), адаптация фрейма (дельта). Последняя реализация языка фреймов – проект XVCL, разрабатываемый группой ученых из Университета Сингапура под руководством Станислава Ерзабека с начала 2000-х годов [36]. В этой реализации предлагается использовать XML как основу языка разметки. XVCL предназначался для организации платформенно-независимого повторного использования в программных приложениях, наиболее впечатляющие успехи были достигнуты для Java-приложений [52]. Язык XVCL использовался также для повторного использования UML-спецификаций [36].

Классической реализацией метода Бассета является продукт Netron Fusion компании Netron[1]. Этот продукт появился в 1980-х годах, предназначался для реструктуризации программ на языке COBOL и имел значительный коммерческий успех. Популярность этого продукта связана с тем, что в оригинальном языке COBOL отсутствуют встроенные средства структуризации кода. Обще-

[1] Netron Fusion Site URL: http://www.netron.com.

принятой практикой программирования на COBOL было создание модулей по несколько тысяч строк кода, имевших изрядно запутанную структуру.

1.1.2 Диаграммы возможностей

При создании сложных систем в индустрии разработки ПО традиционно используются методы и средства визуального моделирования [5]. В частности, при разработке семейств программных продуктов нередко используются UML [14], а также модель возможностей (feature model) [18], [26], [37], [38] и предметно-ориентированные визуальные языки (Domain-Specific Modeling Languages) [49]. Основной источник проблем, возникающих при использовании визуального моделирования при разработке СПП, – необходимость моделировать все возможные различия продуктов семейства (т.е. вариативность семейства). В большинстве подходов интеграция вариативности в модели и визуальный язык приводит к тому, что модели и диаграммы становятся громоздкими и бесполезными.

Наиболее удачно проблема моделирования вариативности семейства продуктов решена в модели возможностей [21], впервые предложенной в работе [37] в 1990 г. группой исследователей из института программной инженерии США во главе с Кио Кангом (Kyo Kang). Основная цель этой модели – в наглядном виде формализовать архетипы и дельты в СПП. Ключевое понятие модели – это возможность (feature), то есть обособленное свойство системы, распознаваемое с точки зрения пользователя или разработчика. Модель возможностей – это набор возможностей и иерархических связей между ними с явно выделенным корнем иерархии, который называется концептом (concept) [26]. Иерархическая связь отображает декомпозицию концепта или возможности и называется отношением включения. Модель возможностей строится в основном на этапе анализа семейства программных продуктов и характеризует семейство в целом.

Основная задача модели возможностей – формализовать общие и различные свойства продуктов семейства. Для этого применяются различные разновидности отношения включения.

Для изображения модели возможностей используются диаграммы возможностей. Фрагмент диаграммы возможностей в классической нотации [26] изображен на Рис. 1.

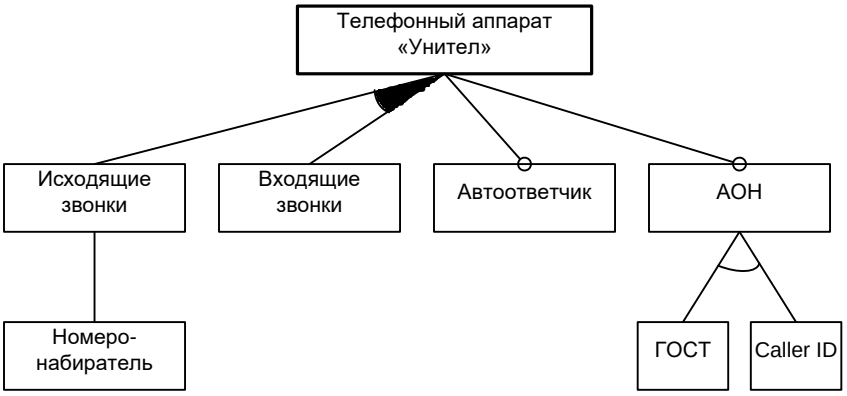


Рис. 1. Пример диаграммы возможностей.

Диаграмма на Рис. 1 описывает семейство телефонных аппаратов «Унител», в которых могут быть реализованы исходящие звонки, входящие звонки, автоответчик и АОН (автоматический определитель номера). Прямоугольником изображается концепт или возможность. Надпись, содержащаяся внутри прямоугольника – это имя соответствующей сущности. Линия, соединяющая концепт с возможностью либо две возможности между собой, изображает обязательное включение (возможность «исходящие звонки» требует наличия возможности «Номеронабиратель»), либо необязательное включение, если линия помечена кружочком со стороны подвозможности (телефонный аппарат может содержать возможность «Автоответчик»).

На практике телефонный аппарат, который не может обслуживать ни исходящие, ни входящие вызовы лишен смысла, так что, по крайней мере, один из этих элементов должен быть представлен в каждом продукте семейства. Для выражения этого факта используется ограничение «произвольное включение»

(OR-группа), для изображения которого на диаграмме линии отношений, на которые накладывается ограничение, соединяются дугой, и угол, ограничиваемый этой дугой, закрашивается (группа включений возможностей «Исходящие звонки» и «Входящие звонки» на Рис. 1).

В ситуации, когда в каждом продукте может быть представлена не более чем одна возможность, используется ограничение «альтернативное включение» (XOR-группа). Для обозначения альтернативного включения линии отношений, на которые накладывается ограничение, соединяются дугой. На Рис. 1 альтернативное включение используется для выражения того факта, что АОН может быть либо российским (ГОСТ), либо европейским (Caller ID).

Любое отношение включения, кроме обязательного включения, является точкой вариации. Точка вариации – это позиция в модели возможностей, которая описывает варианты различной компоновки возможностей в составе различных продуктов семейства.

Существует несколько реализаций диаграмм возможностей (например, Feature Modeling Plug-in[1] и XFeature[2]), но все они являются исследовательскими проектами. Диаграммы возможностей в оригинальном варианте не имеют исполнимой семантики, то есть могут использоваться только на этапах анализа и, в меньшей степени, проектирования ПО и никак не связаны с артефактами последующих этапов, такими как программный код и документация. Различные методы разработки семейств программных продуктов могут определять собственную семантику модели возможностей. Например, она может применяться для конфигурирования продуктов семейства, как предлагается в методе FeaturSEB [31], или как основная модель семейства, вокруг которой интегрируется весь процесс разработки, как в методе DEMRAL [41].

---

[1] http://gsd.uwaterloo.ca/projects/fmp-plugin/.

[2] http://www.pnp-software.com/XFeature/.

1.1.3 Разработка СПП

Разработка семейств программных продуктов завоевывает популярность как в среде исследователей, так и в индустрии [43]. Основная идея, лежащая в основе большинства методов разработки СПП, и впервые предложенная в методе FODA [37], заключается в явном разделении всего процесса на две части – разработку общих активов (domain engineering) и разработку конкретных продуктов семейства (application engineering) [17], [36], [37], [38].

В монографии [30] выделяются две составляющие процесса разработки СПП – разработка семейства продуктов (аналог domain engineering в FODA, см. Рис. 2) и разработка продуктов (аналог application engineering в FODA, см. Рис. 3). Разработка семейства, в основном, выполняется в начале процесса разработки, до массового выпуска продуктов. Тем не менее, допускается возврат к разработке семейства из процесса разработки продуктов, поскольку при добавлении новых продуктов может потребоваться модификация активов семейства.

На этапе анализа семейства необходимо определить, какие продукты могут разрабатываться в рамках семейства, а также выделить общие и вариативные свойства продуктов семейства.

На этапе проектирования семейства нужно решить, как будет разрабатываться семейство и продукты, разработать архитектуру семейства, спроектировать переиспользуемые активы, определить процесс разработки продуктов семейства и спроектировать автоматизацию этого процесса.

Основные задачи этапа реализации семейства – реализовать общие активы семейства, в том числе активы процесса в соответствии с планом автоматизации процесса разработки продуктов.

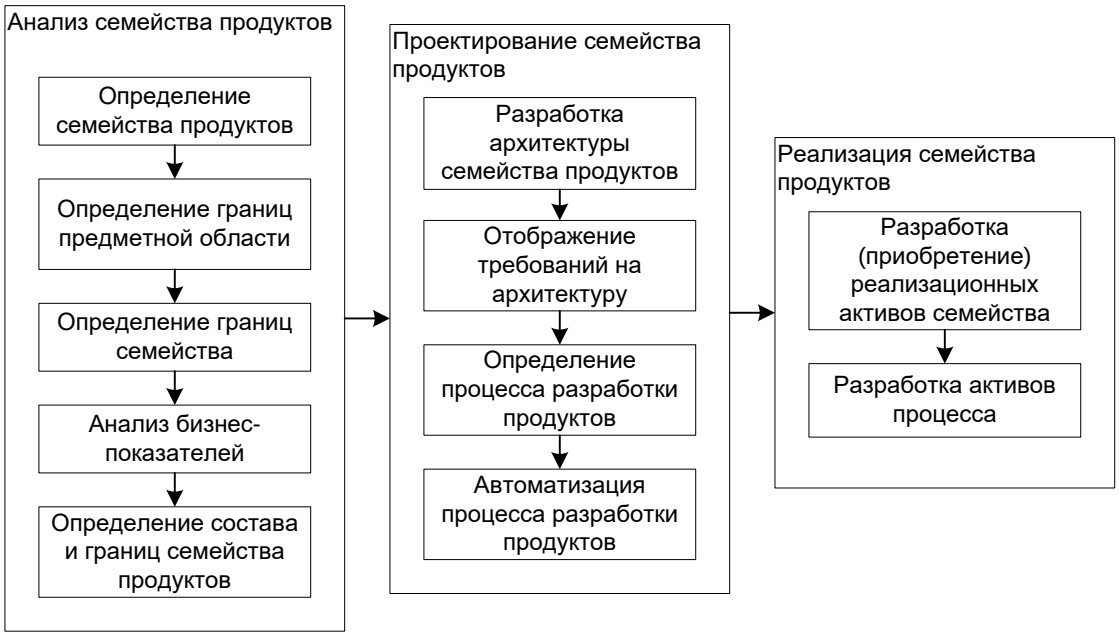


Рис. 2. Процесс разработки СПП [30].

Процесс разработки продукта в рамках семейства изображен на Рис. 3.

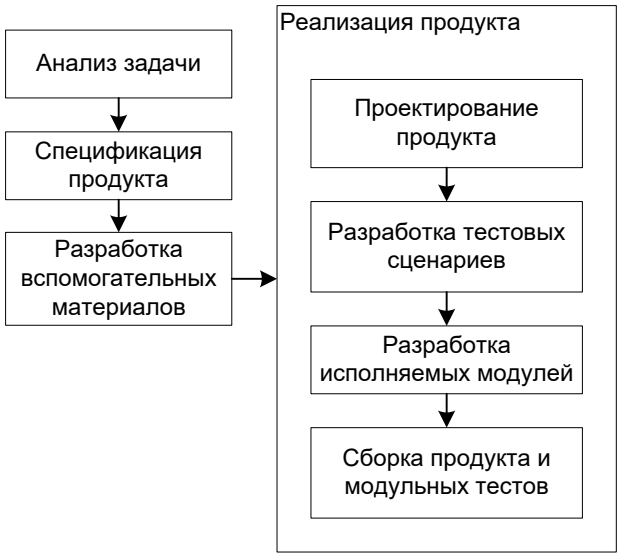


Рис. 3. Процесс разработки продукта в рамках СПП.

Этап анализа задачи необходим для выяснения того, действительно ли поставленная задача может быть решена в рамках разрабатываемого семейства, а также какие именно части задачи могут быть решены в рамках семейства, а какие требуют дополнительной работы и интеграции с семейством.

Основная задача этапа спецификации продукта – определить и формализовать требования к разрабатываемому продукту, а также описать, как именно продукт будет разрабатываться в рамках семейства – какие общие активы могут быть использованы и как потребуется их настроить.

Основная задача этапа разработки вспомогательных материалов – выпустить все необходимые активы продукта, не имеющие отношения к исполняемому коду, в частности, маркетинговые материалы, документацию, упаковку и т.п.

Главная задача этапа реализации продукта – разработать требуемый продукт.

1.1.4 Эталонные модели процесса разработки СПП

Выделяются два типа процессов разработки СПП (также называемых типами эволюции СПП) – проактивные («сверху-вниз», тяжеловесные) [23] и гибкие («снизу-вверх», реактивные, легковесные) [39]. Проактивные процессы предписывают сначала выполнить проектирование семейства, разработать повторно-используемые компоненты, а только затем начинать разработку конкретных продуктов. Некоторые проактивные методы, например [40] предлагают сначала описать и реализовать все возможные конфигурации СПП, а затем получать продукты за счет сочетания и конфигурирования заранее предусмотренных возможностей. «Гибкие» подходы, напротив, предлагают начинать с конкретных продуктов и по мере необходимости выделять повторно используемые активы.

Преимущество проактивных методов в том, что к началу разработки продуктов накапливается обширная база общих активов, а также эталонная архитектура продуктов семейства, что позволяет разрабатывать новые продукты быстрее, качественнее и дешевле. С другой стороны, этот подход связан со зна-

чительными вложениями на начальных этапах разработки и требует длительного времени до выпуска первого продукта.

Преимущество «гибких» подходов заключается в том, что при их использовании не требуется разрабатывать на начальном этапе ничего, кроме документации первого продукта, то есть вложения именно в инфраструктуру семейства минимальны (или вообще отсутствуют). С другой стороны, тщательное предварительное проектирование, характерное для проактивных методов, дает более гибкую архитектуру и в долгосрочной перспективе меньшие расходы на сопровождение и разработку новых продуктов. В конечном итоге выбор метода зависит от потребностей конкретного СПП, оценки его перспектив и наличия ресурсов (материальных, временных и человеческих).

1.2 Разработка технической документации

Зрелые подходы к разработке технической документации обеспечивают отделение содержания документа (контента) от его форматирования. Суть этого принципа состоит в том, что технический писатель при работе над текстом не должен беспокоиться о нюансах полиграфического оформления – достаточно соблюдать логическую структуру текста в виде глав, параграфов и других частей, а правила их форматирования задаются отдельно и могут быть модифицированы независимо от самого текста. Одной из первых реализаций этой идеи была система Дональда Кнута TeX [4]. Затем, с появлением форматов SGML и XML, эта идея стала активно использоваться во вновь создаваемых технологиях и подходах разработки документации (см., например, [51], [54]), а также в различных внутрикорпоративных решениях [29], [32], [12].

Другая тенденция современных подходов к разработке технической документации – использование принципа единого исходного представления (single sourcing), заключающийся в том, что из единого внутреннего представления

текста могут быть сгенерированы различные целевые форматы – HTML, PDF и др. [27], [48], [33].

Рассмотрим несколько подходов подробнее.

1.2.1 DocBook

Технология DocBook появилась в 1991 г. усилиями компаний HaL Computer Systems и O'Reilly&Associates. Ее основная идея – разделение содержания документа и его форматирования, что позволяет создать единое исходное представление документа (single source) и на его основе автоматически генерировать документацию в разных форматах, например, печатная документация (PDF), справочная система (HTMLHelp/WinHelp), электронная документация (HTML) [45].

Технология включает в себя язык, который позволяет выполнить полиграфическую разметку и форматирование текстов документов. Современная версия является XML-языком, описание схемы является открытым, стандартизовано консорциумом OASIS[1] и доступно в нескольких форматах – DTD, XML Schema, Relax NG.

DocBook-документация имеет «монолитную» структуру, что затрудняет повторное использование. Однако поскольку DocBook является XML-языком, для него доступны стандартные для XML средства выделения модулей и их подключения, такие как XInclude. Основной недостаток такого подхода – для указания включаемого фрагмента требуется указывать различные технические параметры, такие как имя файла и т.п. Также этот механизм предполагает, что подключаемые фрагменты используются «as is», без модификации под особенности контекста.

---

[1] Organization for the Advancement of Structured Information Standards Site URL: http://www.oasis-open.org/.

Целый ряд инструментальных средств поддерживает разработку документов DocBook. В первую очередь это пакет XSLT-трансформаций, позволяющий по документам DocBook получить документы в виде HTML, HTMLHelp, FO, PDF, RTF и некоторых других [55]. Также многие коммерческие XML-редакторы (например, XML Spy и Oxygen) предлагают поддержку редактирования DocBook-документов.

В настоящее время DocBook широко используется для разработки документации операционных систем семейства UNIX (FreeBSD, Linux), а также в мире разработки открытого ПО.

1.2.2 DITA

Технология DITA была разработана компанией IBM в 2001 г. для создания модульной технической документации [54]. В отличие от «монолитной» документации DocBook, документация в DITA представляется в виде набора независимых топиков (topic), которые могут иметь различные типы. Таким образом поддерживается повторное использование крупных и логически завершенных фрагментов текста в разных контекстах. В дальнейшем будем называть такое повторное использование ***крупноблочном***.

Язык форматирования документов DITA, также как и DocBook, основан на XML и позволяет не только описывать топики, но полностью задать форматирование текста. DITA позволяет также организовать повторное использование небольших фрагментов текста – отдельных слов, фраз, терминов, фрагментов предложений и т.д.

Оба указанных механизма предполагают, что переиспользуемые фрагменты должны быть написаны так, чтобы их можно было включать в любой контекст без дополнительных изменений. Единственный механизм адаптивного повторного использования, предоставляемый DITA, основан на условном включении фрагментов текста. В DITA версии 1.0 был определен фиксирован-

ный набор переменных, которые можно было использовать при написании условно-включаемого текста (продукт, платформа, аудитория и т.п.). В новой версии 1.2 появилась возможность создавать собственные переменные.

Инструментальные средства DITA содержат пакет преобразований документов DITA в различные выходные форматы. Формат DITA стандартизован международным комитетом OASIS и поддерживается многими XML-редакторами – XML Spy, Oxygen, Frame Maker и др. Стандартные инструменты DITA скорее являются экспериментальными и уступают DocBook в части полиграфического оформления текста. Тем не менее, в ряде крупных компаний, например в IBM, DITA вполне успешно применяется для разработки больших пакетов документов, содержащих много однообразной информации.

1.2.3 FrameMaker

Продукт FrameMaker разрабатывается компанией Adobe[1] и является универсальным текстовым WYSIWIG-редактором [57]. Благодаря стабильности при работе с большими пакетами документов, а также развитой поддержке полиграфии, FrameMaker, фактически, стал стандартом де-факто в разработке технической документации. FrameMaker поддерживает структурное редактирование документов на основе аналога XML-схем, с помощью своего встроенного языка, а также дает возможность использовать DocBook, DITA и другие языки XML-разметки.

FrameMaker позволяет создавать модульную документацию и поддерживает механизм условного включения блоков текста, что позволяет организовать повторное использование документации. Однако отсутствие удобных средств поддержки адаптивного повторного использования ограничивает его использование для семейств программных продуктов.

---

[1] Adobe FrameMaker Site URL: http://www.adobe.com/products/framemaker/.

1.3 Рефакторинг

Рефакторинг – это процесс изменения программной системы, выполняемый таким образом, что внешнее поведение системы не изменяется, но при этом улучшается ее внутренняя структура [28]. Рефакторинг приобрел популярность в «гибких» подходах разработки ПО, поскольку он является альтернативой дорогостоящему предварительному проектированию, обеспечивая постоянные улучшения архитектуры системы с сохранением ее поведения в рамках итеративно-инкрементальной модели процесса разработки ПО, наиболее популярной в настоящее время на практике.

Рефакторинг помогает выполнить такие задачи, как улучшение структуры и внешнего вида программного кода (удаление недостижимого кода, упрощение условных выражений и т.п.), улучшение объектно-ориентированной иерархии (извлечение новых классов, перемещение полей вверх/вниз по иерархии и т.п.). Также есть так называемые «большие рефакторинги», например переход от процедурного к объектно-ориентированному подходу.

Рефакторинг ПО можно выполнять «вручную», однако при этом сложно убедиться в том, что поведение системы остается неизменным. Например, такое действие, как переименование метода, на первый взгляд, кажется тривиальным, однако оно включает в себя поиск и исправление всех вызовов этого метода (включая вызовы через объекты всех классов-потомков) и может затронуть весь исходный код приложения. Имеются инструментальные средства, облегчающие рефакторинг путем автоматизации типичных операций с обеспечением корректности преобразований исходных текстов программ (в данном случае корректность означает сохранение компилируемости кода и его внешнего поведения) – например, Eclipse для языка Java. Такие средства, как интегрированная среда разработки IntelliJ IDEA[1] для языка Java поддерживают «ручной» рефак-

[1] IntelliJ IDEA Site URL: http://www.jetbrains.com/idea/.

торинг, предоставляя средства для автоматизированного регрессионного тестирования выполненных изменений (в том числе включая модульное тестирование).

В рамках разработки СПП появляется важное направление для рефакторинга – повторно-используемые активы и архитектура семейства. Рефакторингу СПП посвящены многие исследования. Так в [19] предлагается метод рефакторинга модели возможностей для расширения множества возможных конфигураций семейства, то есть увеличения потенциала для создания новых продуктов. В [50] авторы предлагают метод декомпозиции приложения на набор возможностей – в результате такого преобразования монолитное приложение преобразуется в набор модулей, которые впоследствии можно использовать для разработки новых продуктов. Работа [25] описывает набор метрик и соответствующий инструментальный пакет для рефакторинга архитектуры СПП. В целом, все перечисленные методы направлены на достижение лучшей вариативности семейства путем выделения общих активов и расширения возможностей их конфигурирования.

1.4 Предметно-ориентированное моделирование

Предметно-ориентированное моделирование (DSM, Domain-Specific Modeling) [30] – подход, который был предложен в конце 1990-х гг. и в настоящее время поддерживается и развивается такими компаниями, как Microsoft, IBM, Borland и др. Его основная идея – ограничение предметной области (вплоть до специфики конкретного проекта разработки ПО) и повышение за счет этого уровня используемых абстракций визуальных языков [49]. DSM-подход является альтернативой стандарту в области визуального моделирования – языку UML, – поскольку последний, являясь универсальным языком, плохо подходит для решения отдельных, частных задач индустрии.

В настоящее время бурно развиваются различные технологи поддержки DSM-подхода. Одна из них, GMF (Graphical Modeling Framework)[1], входит в состав платформы Eclipse и разрабатывается при участии таких компаний, как IBM, Borland, HP, BEA и Red Hat. Eclipse GMF позволяет по метамодели языка и набору описаний сгенерировать репозиторий, графические редакторы, процедуры сохранения/ восстановления моделей и диаграмм. Сгенерированные инструментальные средства интегрируются в платформу Eclipse, обеспечивая привычный для многих разработчиков пользовательский интерфейс и возможность взаимодействия с большим количеством бесплатных и коммерческих модулей расширения.

1.5 Средства формализации текстовых и графических языков

Для современных систем программирования характерно, что у программ есть два основных представления: исходное и целевое. Исходное представление обычно определяется текстовым и/или графическим языком, таким как Java, C++, SDL и т.п. Целевое представление – это или исполнимый код в формате инструкций конкретной ЭВМ и заданной ОС (например, Intel x86 под управлением ОС Windows XP), или байт-код для заданной виртуальной машины (например, для Microsoft.NET Framework или для виртуальной машины Java). В области разработки документации также есть средства, поддерживающие разделение исходного и целевого представления документов. Одна из самых известных таких систем – это TeX [4]. Большинство современных промышленных методов и средств разработки документации (например, DocBook [51], DITA [54], FrameMaker [57]) также разделяют исходное и целевое представление документов. В качестве исходного представления, как правило, использует-

[1] Eclipse GMF URL: http://www.eclipse.org/modeling/gmf/.

ся XML-язык, а целевое представление может быть различным в зависимости от назначения – PDF для печатных документов, HTML для электронных и т.п.

При описании исходного представления современных формальных языков, как правило, выделяют несколько уровней синтаксиса: абстрактный, конкретный и служебный (или сериализационный) [30]. *Абстрактный синтаксис* в общем виде задает список языковых конструкций, их атрибуты и правила сочетания. *Конкретный синтаксис* определяет, как выглядят конструкции языка при написании текстов. *Служебный или сериализационный синтаксис* определяет форму представления конструкций языка, пригодную для долгосрочного хранения и передачи.

Этот подход используется автором при описании языка DRL, поэтому остановимся на нем детальнее. Абстрактный синтаксис DRL формально задает все конструкции языка без привязки к конкретному представлению (текстовому или графическому). Для описания абстрактного синтаксиса используется форма представления грамматик НФБН (Нормальная Форма Бэкуса-Наура) [1]. Ниже представлен фрагмент описания конструкции <Продукт семейства>.

<Продукт семейства> ::

<Идентификатор продукта><Название продукта>

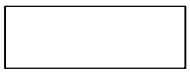
Полная спецификация абстрактного синтаксиса языка DRL представлена в приложении.

Конкретный синтаксис для DRL – это описание графической нотации. Для описания графической нотации используется расширение НФБН, содержащее следующие бинарные инфиксные операторы: *содержит, соединяется с*. Оператор *содержит* означает, что графическое изображение левого аргумента содержит в себе отображение правого элемента. Следующий текст описывает графическую нотацию конструкции <Продукт семейства>.

<Изображение продукта семейства> ::=

<Символ продукта семейства> *содержит* <Имя продукта>

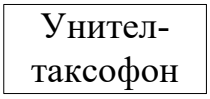
<Символ продукта семейства> ::=



<Имя продукта> ::=

<Текст>

Пример изображения конструкции <Продукт семейства>:



.

Оператор «*соединяется с»* означает, что левый аргумент и правый аргумент в графическом представлении связаны друг с другом, например, набор следующих правил:

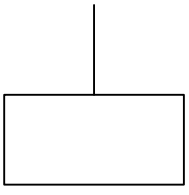
<**Ссылка на элемент**> ::=

<Линия> *соединяется с* <Прямоугольник>

<Линия> ::=

<**Прямоугольник**> ::=

описывает следующую графическую конструкцию:



.

Служебный синтаксис DRL совпадает с конкретным синтаксисом текстовой нотации и представляет собой XML-язык. Для описания служебного син-

таксиса используется язык RelaxNG[1], являющийся одним из стандартов описания схем документов XML. Описание служебного синтаксиса конструкции <Продукт семейства> выглядит следующим образом.

```
<element name=Product>
  <attribute name=id/>
  <attribute name=name/>
</element>
```

Представленный формализм описания языка позволяет однозначно зафиксировать весь набор конструкций языка. Совместно с открытым исходным кодом инструментария поддержки языка, такая спецификация обеспечивает широкие возможности расширения языка как в промышленных условиях применения (как описано в работе [6]), так и в исследовательских целях. Отсутствие подобной полной спецификации, например, для языка моделирования Web-приложений WebML [20] очень сильно затрудняет расширение языка, развитие и создание для него новых программных инструментов.

1.6 Выводы обзора

Существует ряд зрелых подходов к разработке документации, часть из которых поддерживает повторное использование (DITA, DocBook, FrameMaker). Также есть ряд общих методов повторного использования, в которых хорошо проработана адаптивность, но нет возможностей, нужных для разработки документации (фреймы Бассета). Имеются и методы моделирования общих и различных свойств систем, которые могут быть применены к задаче разработки документации (диаграммы возможностей). Однако нет единого метода разработки документации, поддерживающего проектирование сложной документа-

---

[1] Relax NG Site URL: http://relaxng.org/.

ции и адаптивность повторного использования. Для документации СПП эти аспекты являются ключевыми.

При реализации DocLine использовались готовые открытые технологии – DocBook для описания форматирования текстов и публикации документов в целевых форматах и Eclipse GMF для реализации графических средств проектирования структуры сложных пакетов документации. То есть технологию не надо создавать «с чистого листа» и есть возможность сосредоточить основные усилия на главном – на поддержке адаптивного повторного использования документации.

Глава 2. Язык DRL

Схема языка DRL представлена на Рис. 4.

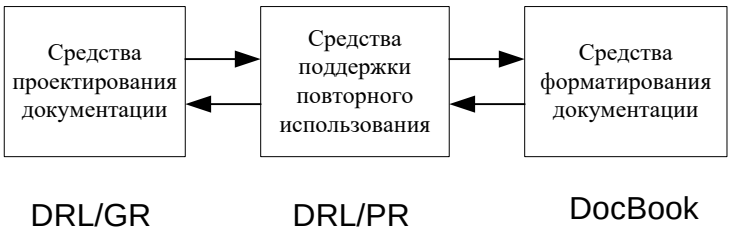


Рис. 4. Схема языка DRL.

DRL/GR предназначен для проектирования документации, то есть задания ее структуры и схемы повторного использования: продуктов семейства, шаблонов документов, связей между шаблонами и продуктами, повторно-используемых компонент и их связей между собой и с шаблонами документов.

DRL/PR предоставляет средства поддержки повторного использования, позволяя детально специфицировать различные аспекты повторного использования фрагментов текста. Также DRL/PR содержит служебный синтаксис для хранения структуры документации, описанной с помощью DRL/GR.

Для реализации форматирования документации (выделения глав, вставки рисунков, таблиц, заголовков и т.д.) DocLine интегрируется с популярной технологией DocBook [51], так что собственных средств форматирования DRL/PR не предоставляет. Такое решение выбрано для облегчения перехода на DocLine для технических писателей, а также для повторного использования подсистем инструментального пакета DocBook, реализующих публикацию документов в различных форматах [55].

Рассмотрим последовательно DRL/GR, DRL/PR и интеграцию DRL с DocBook.

2.1 DRL/GR

Проектирование повторно-используемой документации в DocLine осуществляется с помощью графической нотации DRL/GR, предлагающей следующие виды диаграмм:

- главная диаграмма – содержит список продуктов семейства и состав документации типового продукта;
- диаграмма вариативности – отображает набор повторно используемых фрагментов документации и правила их объединения в документы;
- диаграмма продукта – адаптирует диаграмму вариативности к конкретному продукту СПП.

2.1.1 Главная диаграмма

Описание документации в DRL начинается с создания схемы семейства – списка продуктов семейства и списка документов, составляющих документацию типового продукта.

Для спецификации схемы семейства в DRL/GR предназначена главная диаграмма. На Рис. 5 показан пример главной диаграммы:

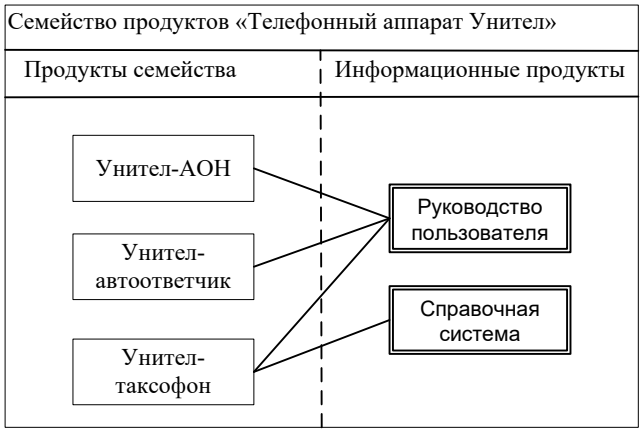


Рис. 5. Главная диаграмма.

Здесь и далее мы будем рассматривать пример семейства телефонных аппаратов «Унител» – это набор телефонных аппаратов различного назначения с разнообразной функциональностью. Главная диаграмма состоит из двух сек-

ций. Слева (секция «Продукты семейства») задается набор продуктов семейства, в данном случае это три вида телефонных аппаратов: «Унител-таксофон» (уличный таксофон, поддерживающий только исходящую связь), «Унител-автоответчик» (домашний аппарат с функцией автоответчика), «Унител-АОН» (домашний аппарат с поддержкой функции АОН). В правой секции («Информационные продукты») определяется состав типового пакета документации, содержащий «Руководство пользователя» и «Справочную систему». Части этого пакета называются информационными продуктами и, фактически, являются шаблонами целевых документов. Для каждого конкретного продукта семейства может требоваться произвольный набор информационных продуктов, что на диаграмме обозначается линией между продуктом и информационным продуктом. В данном случае для всех трех продуктов семейства создается руководство пользователя, а для «Унител-таксофон» создается также и справочная система.

Текстовая нотация DRL/PR является XML-языком и обеспечивает детальную спецификацию повторного использования. Фактически, графическая нотация дает альтернативный взгляд на документацию, позволяющий быстро просмотреть ее структуру. Все, что определяется с помощью визуальной нотации, записывается в конечном виде в XML-представлении. Рассмотрим DRL/PR представление главной диаграммы. Семейство и список продуктов описываются в корневой конструкции <ProductLine/>:

```
<ProductLine name='Телефонный аппарат Унител'>
  <Product  id=pay name=Унител-таксофон/>
  <Product  id=answer name=Унител-автоответчик/>
  <Product  id=aon name=Унител-АОН/>
</ProductLine>
```

Для каждого продукта (<Product/>) указывается имя (name), которое отображается на диаграмме, а также внутренний идентификатор (id), используемый для указания продукта в тексте документации.

Информационные продукты описываются в корневой конструкции <DocumentationCore/>:

```
<DocumentationCore>
  <InfProduct id=guide name='Руководство пользователя' />
  <InfProduct id=help name='Справочная система' />
</DocumentationCore>
```

Для каждого информационного продукта (<InfProduct/>) указывается имя, которое отображается на диаграмме, а также внутренний идентификатор, используемый для указания информационного продукта в тексте документации. Кроме того, информационный продукт может содержать текст, являющийся шаблоном соответствующего документа (содержимое информационного продукта будет рассмотрено далее).

2.1.2 Диаграмма вариативности

Следующий шаг после описания схемы семейства – создание переиспользуемой части документации, то есть шаблонов документов (информационных продуктов) и переиспользуемых строительных блоков, из которых составляются документы. Для описания состава информационного продукта предназначена диаграмма вариативности, основанная на диаграммах возможностей [26] (см. пример на Рис. 6):

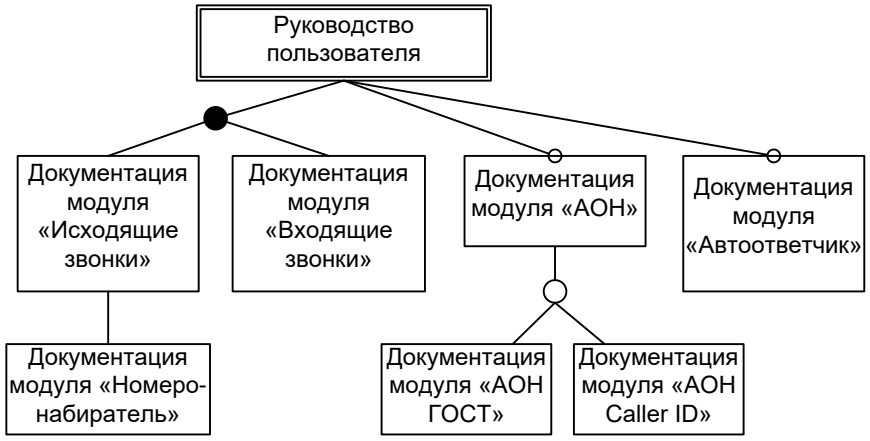


Рис. 6. Диаграмма вариативности.

На этом рисунке показана диаграмма вариативности для информационного продукта «Руководство пользователя». Видно, что он состоит из следующих частей, которые декомпозируются далее:

- документация модуля «Исходящие звонки»,
- документация модуля «Входящие звонки»,
- документация модуля «Автоответчик»,
- документация модуля «АОН».

Такие фрагменты в DRL называются *информационными элементами* и представляют строительные блоки документации – обособленные фрагменты текста, подготовленные к повторному использованию. Это может быть как структурно значимый модуль (глава, секция и т.п.), так и произвольный фрагмент текста. Повторное использование в масштабе информационных элементов называется в DRL *крупноблочным повторным использованием*.

С помощью связей на диаграммах вариативности показывается иерархия включений информационных элементов, причем один и тот же информационный элемент может быть включен в произвольное количество контекстов (отношение аналогичное «каталожному» агрегированию в модели классов UML [2]). В DRL для диаграмм вариативности используется модифицированная нотация Feature Diagrams [26].

На диаграммах вариативности помимо структуры включения информационных продуктов и элементов специфицируется также и простейший вариант адаптивности информационных продуктов – *структурная адаптивность*. Так включение отдельных информационных элементов может быть обязательным или необязательным. На Рис. 6 показано необязательное включение для элемента «Документация модуля «АОН», подразумевающее, что при создании руководства пользователя конкретного продукта семейства на основе информационного продукта «Руководство пользователя», можно включить или исключить документацию модуля «АОН» в соответствии с конфигурацией разрабатываемого телефонного аппарата. Включение элемента «Документация модуля «Номеронабиратель» на Рис. 6 является обязательным, то есть включаемый элемент должен обязательно присутствовать во всех текстах, куда включается элемент «Документация модуля «Исходящие звонки». Также могут задаваться правила включения групп элементов в рамках одного родительского узла: в примере на Рис. 6 указано, что, по крайней мере, один из элементов «Документация модуля «Исходящие звонки» или «Документация модуля «Входящие звонки» должен включаться в любое руководство пользователя, порожденное на основе информационного продукта «Руководство пользователя». Такой тип групп называется OR-группами. Второй тип групп – XOR-группы, из элементов, входящих в такую группу в каждый конкретный контекст может включаться только один (элементы «Документация модуля «АОН ГОСТ» и «Документация модуля «АОН Caller ID» на Рис. 6).

Помимо иерархических связей, на диаграмме вариативности допустимы также семантические связи между произвольными информационными элементами. Семантическая связь не имеет исполнимой семантики и предназначена для указания на произвольную взаимозависимость элементов. Конкретная интерпретация зависит от потребностей проекта. Такие связи удобны для сопровождения документов – с их помощью можно указать, что два информацион-

ных элемента содержат логически связную информацию, поэтому, когда меняется один элемент, нужно изменить и второй.

Для хранения диаграммы вариативности применяется DRL/PR. Для представления информационного элемента используется конструкция <InfElement/>, задающая идентификатор и отображаемое имя информационного элемента. Так информационные элементы на Рис. 6 определяются следующим образом:

```
<DocumentationCore>
  <InfElement id=out name='Документация модуля «Исходящие звонки»' />
  <InfElement id=in name='Документация модуля «Входящие звонки»' />
  <InfElement id=aon name='Документация модуля «АОН»' />
  <InfElement id=ans name='Документация модуля «Автоответчик»' />
  <InfElement id=dia name='Документация модуля «Номеронабиратель»' />
  <InfElement id=gos name='Документация модуля «АОН ГОСТ»' />
  <InfElement id=cid name='Документация модуля «АОН Caller ID»' />
</ DocumentationCore >
```

Однако диаграмма вариативности показывает также и связи между элементами, а именно включение информационных элементов в другие информационные элементы и информационные продукты. Для описания связей используется конструкция <InfElemRef/> – ссылка на информационный элемент. Для объединения ссылок в группы DRL/PR предлагает конструкцию <InfElemRefGroup/>. Информационный продукт «Руководство пользователя», представленный на Рис. 6, выглядит следующим образом:

```
<InfProduct  id=guide name='Руководство пользователя'>
  <InfElemRefGroup id=call_types name='Виды звонков' modifier=OR/>
  <InfElemRef id=out_ref infelemid=out groupid=call_types/>
  <InfElemRef id=in_ref infelemid=in groupid=call_types/>
```

<***InfElemRef*** *id*=aon_ref *infelemid*=aon *optional*=true />

<***InfElemRef*** *id*=ans_ref *infelemid*=ans *optional*=true />

</***InfProduct***>

Для группы указывается идентификатор, имя и модификатор, определяющий тип группы. Для групп, объединяющих взаимоисключающие элементы, следует использовать модификатор XOR, а для групп, объединяющих элементы которые можно использовать в произвольном сочетании используется модификатор OR. Для каждой ссылки на информационный элемент задается идентификатор ссылки, идентификатор информационного элемента, который требуется подключить, а также дополнительные параметры – идентификатор группы для ссылок, объединенных в группы (как out_ref и in_ref) и/или атрибут optional, показывающий, что включение задаваемое ссылкой является необязательным, как для ссылок aon_ref и ans_ref.

Аналогично описывается декомпозиция информационного элемента. Так, например, информационный элемент 'Документация модуля «АОН» описывается следующим образом:

< ***InfElement*** *id*=aon *name*='Документация модуля «АОН»'>

<***InfElemRefGroup*** *id*=aon_types *name*='Виды АОН' *modifier=XOR/*>

<***InfElemRef*** *id*=gos_ref *infelemid*=gos *groupid*=aon_types/>

<***InfElemRef*** *id*=cid_ref *infelemid=cid groupid*=aon_types/>

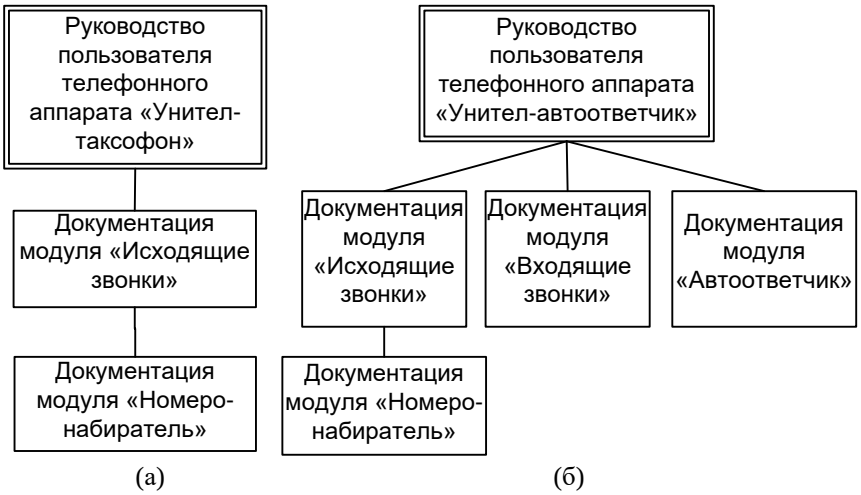
</ ***InfElement***>

2.1.3 Диаграмма продукта

До сих пор мы рассматривали конструкции, описывающие переиспользуемую часть документации. Однако для того, чтобы переиспользуемая документация превратилась в конкретный документ, требуется задать какие информационные продукты для какого продукта семейства актуальны, а также разрешить все неоднозначности. Приведем аналогию из объектно-ориентированного

подхода. Документация семейства – это набор классов. Для работы приложения требуется создать необходимые объекты и задать их атрибуты. Также и в DRL – для порождения конкретных документов необходимо создать документацию продукта.

В DRL/GR документация продукта специфицируется на диаграмме продукта. Пример диаграмм продуктов представлен на Рис. 7: информационный продукт «Руководство пользователя» специализируется для продукта «Унител-таксофон» (Рис. 7, (а)) и для продукта «Унител-автоответчик» (Рис. 7, (б)).



(а) (б)

Рис. 7. Диаграммы продукта.

На этих диаграммах «разрешены» все неопределенности исходной диаграммы вариативности. Например, для «Унител-таксофон» (Рис. 7, (а)) в группе произвольного включения информационных элементов «Документация модуля «Входящие звонки»» и «Документация модуля «Исходящие звонки» выбран только последний информационный элемент. Корень диаграммы продукта – *финальный информационный продукт*, определяющий специализацию информационного продукта для конкретного продукта семейства.

В DocLine-документации выделяются несколько областей видимости: область переиспользуемой части документации (конструкция <DocumentationCore/>), а также область документации конкретного продукта (конструкция <ProductDocumentation />) – для каждого продукта семейства эта

область своя. Главная диаграмма и диаграммы вариативности относятся к области видимости переиспользуемой документации, тогда как диаграмма продукта располагается уже в области видимости документации конкретного продукта.

DRL/PR-представление диаграммы Рис. 7, (а) выглядит следующим образом:

<***ProductDocumentation*** *productid*="pay">
<***FinalInfProduct*** *id*="guide_pay" *infproductid*="guide">
<***Adapter*** *infelemrefid*="out_ref”/>
</***FinalInfProduct***>
</***ProductDocumentation***>

В данном фрагменте указывается, с каким продуктом семейства мы работаем (<ProductDocumentation />), далее указываются специфицируемые финальные информационные продукты (<FinalInfProduct />). Для указания включений информационных элементов, входящих в заданный финальный информационный продукт, используется конструкция адаптер (<Adapter/>) – для фактического включения элемента, для которого на диаграмме вариативности есть неоднозначность, должен быть создан адаптер. Помимо указания включаемых элементов адаптеры позволяют выполнить настройку включаемого элемента – об этом будет рассказано в следующем разделе.

2.2 DRL/PR

DRL/PR обеспечивает возможность детального описания повторно используемых компонент – эти детали осознанно вынесены за пределы DRL/GR для того, чтобы сохранить компактность и наглядность диаграмм. В данном разделе рассматриваются конструкции DRL/PR, которые не имеют непосредственного графического представления.

2.2.1 Адаптивное крупноблочное повторное использование

Для организации повторного использования в реальной документации СПП недостаточно только структурной адаптивности, задаваемой на диаграммах вариативности. Рассмотрим пример фрагмента документации телефонного аппарата. Информационный элемент «Документация модуля «АОН» может содержать такой текст:

После получения вызова телефон запрашивает в сети информацию о вызывающем абоненте и затем отображает на экране номер абонента. (1)

Этот текст подходит для краткой памятки по использованию телефона, а в полном руководстве может быть дополнение:

После получения вызова телефон запрашивает в сети информацию о вызывающем абоненте и затем отображает на экране номер абонента. Для настройки параметров отображения номера вызовите меню и перейдите в раздел Вызовы->Входящие->АОН. (2)

Для обеспечения преобразования примера (1) к виду (2) в DRL имеются конструкции *параметрической адаптивности*. В синтаксисе DRL соответствующий фрагмент текста, подготовленный к повторному использованию с поддержкой параметрической адаптивности, выглядел бы так:

*<**InfElement** id=CallerIdent>*
*После получения вызова телефон запрашивает в сети информацию о звонящем абоненте и затем <**Nest** id=DisplayOptions>* (3)
*отображает на экране номер абонента</**Nest**>.*
*</**InfElement**>*

В нем мы определяем информационный элемент (тег <InfElement/>) и точку расширения (тег <Nest/>). Когда информационный элемент включается в

определенный контекст, каждая точка расширения может быть удалена или дополнена специфичным текстом без необходимости изменения исходного информационного элемента. Если не задано никаких расширений, информационный элемент из примера (3) будет преобразован в текст примера (1). Следующие настройки преобразуют информационный элемент (3) в вид (2):

```
<InfElemRef  id=CallerIdentRef infelemid=CallerIdent>
<Insert-After nestid=DisplayOptions>Для настройки параметров отображения номера вызовите меню и перейдите в раздел Вызовы->Входящие->АОН.
</Insert-After>
</InfElemRef>
```
(4)

В этом фрагменте содержится ссылка на информационный элемент (<InfElemRef/>), определенный в (3), и команда добавления текста к точке расширения (<Insert-After />). Данный фрагмент должен содержаться в контексте информационного продукта «Руководство пользователя», тогда как в контексте информационного продукта «Краткая справка» будет только конструкция <InfElemRef/> без вложенной <Insert-After/>, что при публикации даст вид (1).

В приведенных примерах модификация подключаемого информационного элемента выполняется в точке подключения. Такой подход оправдан, когда информационный элемент используется в нескольких информационных продуктах в рамках одного продукта семейства. Для адаптивного повторного использования фрагмента текста между аналогичными документами для разных продуктов семейства требуется настраивать точки расширения не в момент включения, а в момент публикации, т.е. соответствующие команды должны быть заданы в контексте документации продукта, а именно внутри описания финального информационного продукта. Вернемся к примеру с телефонным аппаратом. Различные типы телефонов (продукты семейства) могут иметь разные оп-

ции индикации вызывающего абонента. Например, телефон для слабовидящих может вместо визуального отображения проговаривать номер абонента. Тогда фрагмент (1) выглядел бы так:

После получения входящего вызова телефон запрашивает в сети информацию о вызывающем абоненте и затем проговаривает номер абонента. (5)

Для того чтобы из (1) получить (5), необходимо выполнить манипуляцию над точкой расширения, определенной в (3), однако эта манипуляция должна быть определена внутри описания соответствующего финального информационного продукта. Для этого в DRL используется конструкция адаптер (<Adapter/>):

<***FinalInfProduct*** *id="UserManual_starblind"*
infproductid="UserManual">
<***Adapter*** *infelemrefid="CallerIdentRef”>*
<***replace-nest*** *nestid=DisplayOptions>проговаривает номер абонента*</***replace-nest***> (6)
</***Adapter***>
</***FinalInfProduct***>

Адаптер позволяет настроить включение любого информационного элемента – для этого в конструкции <Adapter/> предусмотрен атрибут infelemrefid, задающий идентификатор изменяемого включения. В адаптере можно выполнить те же настройки, что и в точке включения. Так в примере (6) используется команда замены текста точки расширения новым текстом (<Replace-Nest/>), приводящая текст (1) к виду (5). Аналогичным образом можно вставить текст до или после точки расширения, для этого предназначены конструкции <Insert-Before/> и <Insert-After/> соответственно – см. пример:

*<**InfElemRef** infelemid=CallerIdent>*
*<**Insert-After** nestid=DisplayOptions>и имя абонента,*
если оно задано в записной книжке телефона (7)
*</ **Insert-After**>*
*</ **InfElemRef** >*

Этот пример расширяет описание набора способов отображения информации о вызывающем абоненте для случая телефона с записной книжкой. Его выполнение приведет текст (1) к следующему виду:

После получения входящего вызова телефон запрашивает в сети информацию о вызывающем абоненте и затем отображает на экране номер абонента и имя абонента, если оно задано в записной книжке телефона. (8)

2.2.2 Адаптивное «мелкозернистое» повторное использование

Точки расширения и операции их модификации позволяют организовать адаптивное повторное использование достаточно крупных фрагментов текста – разделов, абзацев, предложений. Этот механизм можно использовать и для меньших фрагментов (слов или словосочетаний), однако для такого случая в DRL предусмотрен упрощенный механизм «мелкозернистого» повторного использования. Соответствующие конструкции не отображаются на диаграммах, так как для них есть только DRL/PR представление. «Мелкозернистое» повторное использование реализуется в DRL с помощью словарей и каталогов.

Словарь – это набор пар (имя, значение). В произвольной точке документа можно подставить значение элемента словаря, указав его имя. В словаре могут быть такие элементы, как название продукта, версия, набор поддерживаемых операционных систем и т.п. В DRL/PR словари задаются следующим образом:

<***DocumentationCore***>

<**Dictionary** *id="main"*>
<**Entry** *id="Company">ТелеСистемы</***Entry**>
<**Entry** *id="Product">Унител</***Entry**>
</**Dictionary**>
</**DocumentationCore**>

Здесь задается словарь (конструкция <Dictionary />) с двумя элементами (конструкция <Entry />). Для использования словаря имеется конструкция *ссылка на элемент словаря* (<DictRef />):

Производитель телефона
<**DictRef** *dictid="main" entryid="Product"/>– компания*
<**DictRef** *dictid="main" entryid="Company"/>*

После публикации соответствующий фрагмент будет выглядеть так:

Производитель телефона Унител – компания ТелеСистемы.

Однако в случае нашего семейства, название телефона должно быть разным для разных продуктов. Для реализации подобной адаптивности в DRL предусмотрено переопределение словарей в документации продукта:

<**ProductDocumentation** *productid="pay">*
<**Dictionary** *id="main">*
<**Entry** *id="Product">Унител-Таксофон</***Entry**>
</**Dictionary**>
</**DocumentationCore**>

В данном примере словарь переопределяется в контексте документации продукта Унител-таксофон, причем указываются только те элементы, которые действительно нужно изменить, – в данном случае название продукта. При публикации сначала происходит поиск элемента в словарях, определенных в документации конкретного продукта, и только для ненайденных элементов

осуществляется поиск в переиспользуемой части документации. В данном случае фрагмент текста со ссылкой на элемент словаря после публикации выглядит следующим образом:

Производитель телефона Унител-таксофон – компания ТелеСистемы.

Рассмотрим следующую конструкцию «мелкозернистого» повторного использования – каталог. В интерфейсах программных продуктов можно обнаружить много однотипных элементов, например, команды пользовательского интерфейса. Эти команды по-разному представлены в интерфейсе ПО (в панели инструментов, в меню, всплывающие подсказки и пр.) и, соответственно, в пользовательской документации. Так при описании панели инструментов эти команды описываются пиктограммами с названием и текстом всплывающей подсказки, при описании меню можно видеть название команды и комбинацию горячих клавиш.

Для удобства задания подобных списков в DocLine вводится понятие каталога. Каталог содержит элементы, заданные набором атрибутов, например, каталог команд содержит название, пиктограмму, описание, сочетание горячих клавиш, текст всплывающей подсказки, список побочных эффектов, правила использования и т.п. Покажем, как может выглядеть описание двух команд «Напечатать» и «Сохранить Как» в каталоге:

```
<directory name="GUICommands">
    <entry name="Print">
      <attr name="name">Напечатать</attr>
      <attr name="icon">Print.bmp</attr>
      <attr name="descr">Печать активного документа</attr>
    </entry>
    <entry name=" SaveAs">
```

```
        <attr name="name">Сохранить как</attr>
        <attr name="icon">SaveAs.bmp</attr>
        <attr name="descr">Сохранение с новым именем</attr>
      </entry>
    </directory>
```

В дополнение к набору элементов каталог содержит ряд шаблонов отображения, определяющих то, как компоновать атрибуты для разных представлений каталога в тексте документации. Представленный ниже шаблон задает представление для описания панели инструментов и включает пиктограмму и название команды:

```
<DirTemplate name="toolbar_short" directory="GUICommands">
  <fig>Images/<attrref name='icon'/></fig><attrref name="name"/>
</DirTemplate>
```

Шаблон отображения содержит текст и ссылки на атрибуты элемента (<attrref/>). Когда технический писатель включает элемент каталога в целевой контекст, то требуется указать идентификатор элемента и соответствующий шаблон представления. Затем содержимое шаблона будет помещено в заданную точку и все ссылки на атрибуты будут заменены их значениями для указанного элемента. Используя различные шаблоны представления можно организовывать различные формы отображения одних и тех же элементов – в сокращенной форме, в полной форме и т.п. Пример ссылки на элемент каталога в тексте документации:

```
<dirref templateid= "toolbar_short" entryid=SaveAs / >
```

Ссылка на элемент каталога задается конструкцией <dirref />, в качестве атрибутов указываются идентификатор шаблона и записи в каталоге. Следует отметить, что в данной конструкции не указывается собственно из какого сло-

варя необходимо извлечь значение, так как каждый шаблон может ссылаться на единственный каталог.

2.2.3 Условные блоки

Для простейших случаев адаптивного повторного использования достаточно конструкции условного включения фрагмента текста. В зависимости от значения заданного условия такой фрагмент включается или не включается в документацию конкретного продукта. В DRL для этих целей служит конструкция <Conditional/ >.

Дисплей телефона имеет
*<**Conditional** condition= GreenBL> зеленую</**Conditional**>*
*<**Conditional** condition=OrangeBL> оранжевую</**Conditional**>подсветку.*

В документации продукта можно задать значение условия с помощью конструкции <SetValue />. Пример приведен ниже.

*<**SetValue** id=GreenBL value=True/>*
*<**SetValue** id=OrangeBL value=False/>*

Такой механизм может быть полезен в случае, когда, например, в разных партиях подсветка может иметь разный цвет, при том, что остальные характеристики телефона неизменны.

2.3 Интеграция языка DRL с форматом DocBook

В современных технических документах используется множество приемов форматирования текста – заголовки, главы, таблицы, списки, рисунки, сноски, ссылки и т.п. Для поддержки форматирования текста DocLine интегрируется с технологией DocBook, которая широко используется для разработки документации свободного ПО, но не содержит средств адаптивного повторного использования. В тексте на DRL могут использоваться конструкции DocBook и наобо-

рот, внутри конструкций DocBook могут использоваться конструкции DRL, например ссылка на элемент словаря или ссылка на информационный элемент.

Так, в частности, благодаря открытости и интеграции с DocBook, DocLine можно использовать и для разработки документации по ГОСТ ЕСПД[1], что может потребоваться при внедрении метода в российских компаниях. Это потребует доработки шаблонов публикации документов, предлагаемых DocBook. Подобная доработка, только в меньшем объеме, выполнялась в процессе апробации DocLine на документации семейства телекоммуникационных систем [10].

[1] ГОСТ 19 ЕСПД – Единая Система Программной Документации.

Глава 3. Процесс разработки документации

Процессы разработки СПП подразделяются на 2 типа: проактивные («сверху-вниз») и «гибкие» («снизу-вверх») [23] [39]. В DocLine поддерживаются оба типа процесса. В проактивном варианте процесса создание документации начинается с тщательного проектирования документации с планированием повторного использования, затем разрабатываются общие активы. Только после этого создается собственно документация первого продукта.

В «гибком» процессе разработка документации начинается с создания документации для первого продукта. Затем, когда появляются новые продукты и возникает необходимость разработать документацию для них, выполняется выделение общих фрагментов и разработка документации очередного продукта на основе полученных общих активов.

На практике многие компании, разрабатывающие ПО, выделяют довольно скромные ресурсы на создание документации. В таких условиях, а также когда разработка семейства начинается с одного продукта (например, в рамках легковесного процесса [39]), «гибкий» процесс более применим, поскольку требует на старте существенно меньших вложений и дает первый результат раньше.

Независимо от выбранной модели процесса, перед тем как приступить к разработке повторно-используемой документации с помощью DocLine, нужно выполнить оценку целесообразности применения такого подхода. Рассмотрим критерии целесообразности, а затем разберем предложенные модели процесса более подробно.

3.1 Целесообразность применения DocLine

Использование метода DocLine требует обучения технических писателей, причем для тонкой настройки повторного использования от технического писателя потребуется навык написания XML-документов [6]. Для технических пи-

сателей, имеющих опыт работы с такими технологиями как DITA и DocBook, обучение не составит большого труда, поскольку они также основаны на XML-технологии. Кроме того, организация, применяющая предлагаемый метод, должна быть достаточно стабильной и иметь долгосрочные планы, связанные с разработкой семейства программных продуктов, поскольку в краткосрочной перспективе стоимость повторно-используемой документации возрастает, как и в случае с любым методом планируемого повторного использования общих активов [58]. Таким образом, перед принятием решения об использовании метода, необходимо оценить также и экономическую целесообразность его применения.

Благодаря поддержке гибкого процесса разработки, метод применим практически в любых условиях, поскольку позволяет на начальных этапах не делать вложений в повторное использование и по сложности и стоимости не превосходит применение популярного DocBook. В ряде случаев использование DocLine дает существенные преимущества по сравнению с традиционными методами разработки документации. Рассмотрим эти случаи.

- Документация различных продуктов семейства имеет много общего, при этом переиспользуемые фрагменты имеют различия. В случае отсутствия различий вполне эффективно будут работать традиционные методы разработки документации.
- Регулярно разрабатываются новые версии существующих продуктов или новые продукты семейства. Если же документация разрабатывается один раз и исправления в нее вноситься не должны, то традиционные методы разработки документации в сочетании с простейшим методом повторного использования копирование-вставка вполне удовлетворят потребности разработчика.

Отдельно отметим, что в случае, когда ошибки в документации недопустимы (например, в случае mission-critical систем), поддержка повторного ис-

пользования, предоставляемая DocLine, позволяет снизить вероятность размножения ошибки, характерного для повторного использования методом копирования-вставки-модификации.

3.2 Проактивный процесс

Независимо от модели процесса в работе технических писателей можно выделить два вида деятельности – разработка повторно-используемой документации и разработка документации конкретных продуктов. В проактивном варианте процесса создание документации начинается с разработки повторно-используемой документации (см. Рис. 8).

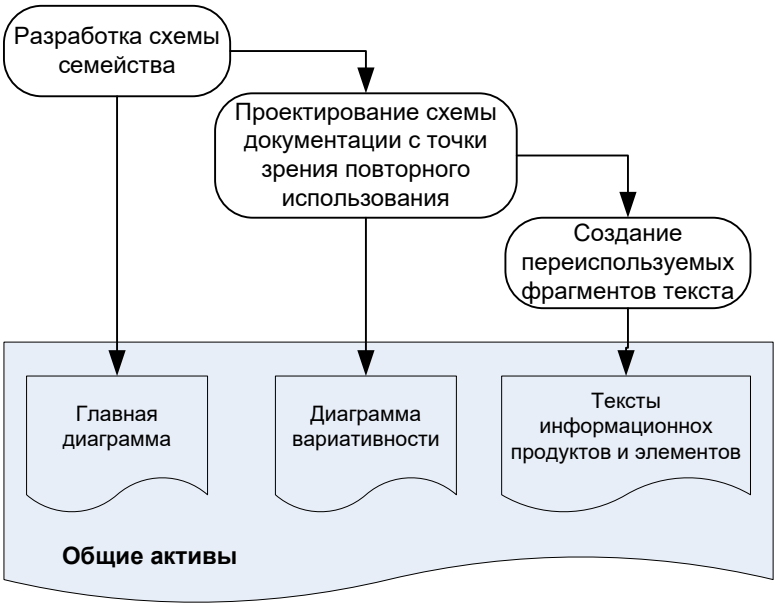


Рис. 8. Схема процесса разработки повторно-используемой документации.

Когда общие активы документации созданы, на их основе порождается документация конкретных продуктов (см. Рис. 9).

При добавлении к семейству второго и последующих часть общих активов может потребовать модификации. Также может потребоваться создать новые общие активы на основе документации существующих продуктов. Однако от добавления нового продукта, существующие не должны меняться. Это правило распространяется на все общие активы, включая код (о сценариях эволюции

повторно используемого программного кода см. работу [46]) и документацию. Таким образом, вместе с созданием и модификацией общих активов необходимо внести корректирующие изменения в документацию существующих продуктов. Такое выделение общих активов из документации первого продукта и соответствующее изменение этой документации так, чтобы ее фактическое содержание осталось неизменным, является рефакторингом документации. При смысловой неизменности этой документации ее внутреннее DRL-представление претерпевает значительные изменения. Схема процесса добавления второго и последующих продуктов показа на Рис. 10.

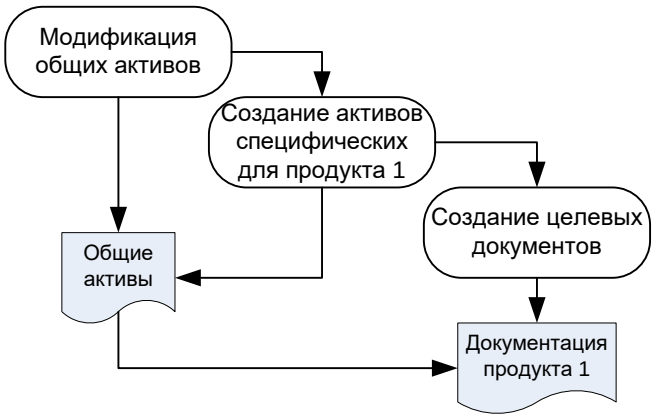


Рис. 9. Схема процесса разработки документации одного продукта.

При использовании проактивного подхода первый отчуждаемый результат требует довольно большой подготовительной работы. Такой процесс целесообразно применять в ситуации, когда заранее известно, какие продукты будут в

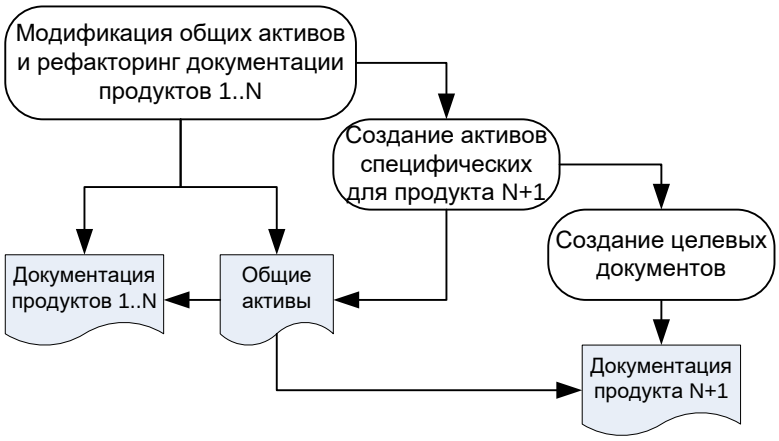


Рис. 10. Схема процесса разработки документации второго и последующих продуктов.

семействе, а также имеется достаточно ресурсов (людей, времени, денег и т.п.) для глубокого анализа возможностей повторного использования.

3.3 «Гибкий» процесс

В случае «гибкого» процесса сначала разрабатывается документация для первого продукта без учета повторного использования. Это соответствует гибкому подходу к разработке ПО, когда в начале делается необходимый минимум инфраструктурной работы и основное внимание уделяется тому, что нужно заказчику ПО. С точки зрения последующего перехода к повторно-используемой документации, на этом этапе накладывается единственное ограничение – документацию желательно изначально разрабатывать с помощью DocLine или DocBook. Оба формата позволяют разрабатывать обычную монолитную документацию, не расходуя ресурсы на организацию повторного использования. Допускается использовать и иные средства разработки документации, но в таком случае потребуется осуществлять преобразование документации к формату DocBook, которое не всегда может быть выполнено автоматически.

Далее, во время (или после) разработки второго и последующих продуктов семейства создается документация и для них. Для этого анализируются общие места и различия между новым продуктом и существующими, выделяются об-

щие активы в документации существующих продуктов (фрагменты текста, которые можно переиспользовать либо без изменения, либо с незначительными изменениями), которые оформляются средствами DocLine. Эти общие активы интегрируются в документацию существующих продуктов вместо прежних фрагментов, при этом внешний вид документации существующих продуктов остается неизменным – как и в проактивном процессе тут применяется рефакторинг документации.

Отметим, что при разработке документации для очередного продукта существующие и вновь созданные фрагменты текста, которые используются однократно (или используются идентично единым блоком во всех контекстах), должны быть оформлены в виде единого информационного элемента. DRL-спецификацию целесообразно декомпозировать детальнее, если информационный элемент различается в различных контекстах использования или же если есть иные существенные причины для декомпозиции, например параллельное редактирование документации несколькими техническими писателями. Чтобы сохранить визуальные модели понятными, не стоит использовать DRL для обычной декомпозиции текста на главы, разделы, подразделы и т.д. – в противном случае структура повторного использования потеряется среди большого количества структурных элементов текста.

3.4 Операции рефакторинга

В гибком процессе разработки документации, предлагаемом DocLine, рефакторинг играет очень важную роль. Однако и в проактивном процессе для рефакторинга также найдется применение. Сам по себе рефакторинг – это процедура, которая может выполняться и без специальной поддержки, однако опыт сред разработки ПО показывает, что специальные средства рефакторинга существенно повышают его эффективность. В DocLine предлагается ряд типовых операций рефакторинга, позволяющих преобразовывать исходное представле-

ние документации с целью улучшения возможностей повторного использования или оптимизации внутренней структуры документации. Рассмотрим несколько групп операций.

3.4.1 Создание общих активов

Следующие операции помогают созданию общих активов на базе существующего текста. Как правило, они используются при преобразовании монолитной документации для повторного использования, а также при присоединении к существующему семейству документации нового продукта.

1. *Импорт документации DocBook*. Монолитный документ в формате DocBook импортируется в DocLine. После выполнения данной операции документация остается монолитной, однако вся дальнейшая работа с ней может выполняться уже средствами DocLine. Для этого создается минимально необходимый набор DRL-конструкций, а именно:
   - информационный элемент, содержащий полный текст исходного DocBook-документа;
   - информационный продукт, определяющий единственный тип документа и состоящий из ссылки на вышеуказанный информационный элемент;
   - финальный информационный продукт, описывающий необходимые адаптации информационного продукта для получения документации конкретного продукта СПП (в данном случае адаптации не требуются, поскольку речь идет об идентичной копии исходного документа).
2. *Извлечение информационного продукта.* Выбранный документ (или фрагмент) выделяется в отдельный информационный продукт. Затем создается финальный информационный продукт, использующий вновь созданный элемент для порождения точной копии исходного документа. Эта операция обычно используется, когда существующая монолитная документация им-

портируется в DocLine и технический писатель разделяет ее на несколько информационных продуктов.

3. *Извлечение информационного элемента*. Фрагмент DRL-спецификации выделяется в отдельный информационный элемент. Затем в той позиции, где он находился, вставляется ссылка на вновь созданный информационный элемент. Если извлекаемый фрагмент содержал точки расширения, то для всех финальных информационных продуктов создаются новые адаптеры для сохранения всех манипуляций с этими точками расширения.
4. *Расщепление информационного элемента*. Существующий информационный элемент разделяется на два новых. Все ссылки и адаптеры, относящиеся к исходному элементу, соответствующим образом модифицируются.

3.4.2 Настройка общих активов

Следующие операции предназначены для улучшения адаптивности общих активов, т.е. для подготовки фрагментов текста к повторному использованию в различных контекстах.

5. *Преобразование в точку расширения*. Фрагмент текста внутри информационного элемента преобразуется в точку расширения (окружается конструкцией DRL <nest/>): после этого данный фрагмент может быть удален или дополнен независимо для каждого контекста использования (различных документов для одного продукта или похожих документов для разных продуктов).
6. *Выделение в Insert-After/Insert-Before*. Начальный/хвостовой фрагмент текста точки расширения извлекается и помещается во все существующие ссылки на исходный информационный элемент в составе конструкции Insert-After/Insert-Before. Если указанный фрагмент не находится внутри точки расширения, но находится внутри информационного элемента, то создается новая пустая точка расширения в позиции этого фрагмента.

7. *Объявление ссылки на информационный элемент необязательной и наоборот*. Указанная обязательная ссылка на информационный элемент (infelemref) декларируется как необязательная. Это означает, что в новых финальных информационных продуктах можно будет удалять указанную ссылку. Вместе с тем, существующие документы должны остаться неизменными, так что во всех существующих финальных информационных продуктах, содержащих упомянутый информационный элемент, создается адаптер, форсирующий включение необязательного информационного элемента. Обратная операция (объявление ссылки на информационный элемент обязательной) возможна, только если во всех существующих контекстах необязательная ссылка фактически включалась, при этом после объявления ссылки обязательной соответствующие адаптеры могут быть удалены.
8. *Преобразование в условный блок*. Выделенный фрагмент текста помечается как условный блок (conditional) с заданным логическим условием. Во всех существующих финальных информационных продуктах, содержащих указанный фрагмент, условие устанавливается истинным, чтобы гарантировать неизменность существующих конечных документов.
9. *Изменение правила по умолчанию для включения необязательных ссылок на информационный элемент*. В зависимости от особенностей проекта необязательные ссылки на информационные элементы могут по умолчанию включаться или исключаться во всех контекстах использования, где явно не указывается требуемое поведение. Так, если необязательные ссылки по умолчанию включены, то везде, где в адаптере отсутствуют команды, явно их исключающие, они будут входить в соответствующий контекст, и наоборот, если по умолчанию такие ссылки исключаются, то везде, где нет явных команд их включения, они будут опущены. При изменении зна-

чений по умолчанию все адаптеры должны быть обновлены, чтобы гарантировать неизменность существующих конечных документов.

3.4.3 Настройка «мелкозернистого» повторного использования

Следующие операции предназначены для оптимизации конструкций, поддерживающих «мелкозернистое» повторное использование – словарей и каталогов.

10. *Извлечение в словарь*. Выделенный фрагмент текста (обычно отдельное слово или словосочетание) выносится в словарь и его вхождение заменяется ссылкой на новый элемент словаря. Затем документация автоматически просматривается в поисках других вхождений того же элемента и найденные вхождения заменяются ссылкой на элемент словаря, при этом технический писатель имеет возможность отказаться от замены тех или иных вхождений.
11. *Извлечение в каталог*. На основе выделенного фрагмента создается новый элемент каталога. Затем выделенный текст заменяется ссылкой на новый элемент каталога с использованием выбранного (или вновь созданного) шаблона представления элемента каталога.
12. *Копирование/перемещение элемента словаря/каталога из переиспользуемой документации в документацию конкретного продукта или наоборот*. Выбранный элемент словаря или каталога копируется или перемещается из общих активов в документацию конкретного продукта и наоборот. Перемещение элемента словаря из общих активов в документацию продукта применяется когда требуется устанавливать различные значения этого элемента в различных продуктах. Обратное перемещение предназначено для того, чтобы получить возможность использовать элемент словаря не только в данном продукте, но и в других продуктах.

3.4.4 Переименование

Следующая операция обеспечивают переименование различных структурных элементов документации. Вот список элементов документации, которые могут быть переименованы: информационный элемент, информационный продукт, словарь, каталог, элемент словаря или каталога, точка расширения, ссылка на информационный элемент, шаблон представления элементов каталога.

13. *Переименование.* После переименования все ссылки на переименованные элементы автоматически обновляются.

3.4.5 Пример применения рефакторинга

Рассмотрим пример выполнения операции рефакторинга. Цель этого примера – показать, что операции рефакторинга требуют нелокальных модификаций исходного текста документации для того, чтобы сохранить конечные документы неизменными, следовательно, для их выполнения необходимы специальные средства автоматизации. Ниже приводится фрагмент исходного текста DRL-документации (указанные конструкции могут располагаться в различных файлах):

<infproduct id="phone_manual">
<infelemref id="InOut_ref" infelemid="InOut"/>
</infproduct>
<infelement id=InOut>
<section><title>Исходящие звонки</title>
Ваш телефон поддерживает следующие способы набора номера:
<nest id=”DialOptions”>набор на цифровой лавиатуре</nest>.
</section>
<section><title>Входящие звонки</title>
После получения входящего вызова телефон запрашивает в сети информацию о звонящем абоненте и затем

<nest id=”DisplayOptions”>отображает на
экране номер абонента</nest>.
</section>
</infelement>
<finalinfproduct name="office_phone" infproductid = "phone_manual">
<adapter infelemrefid = "InOut_ref">
<insert-after nestid = "DialOptions">
или выбор из адресной книги
</insert-after>
<insert-after nestid = "DisplayOptions">
и имя абонента, если оно задано в записной книжке телефона
</insert-after>
</adapter>
</finalinfproduct>

Этот фрагмент содержит информационный продукт phone_manual, являющийся шаблоном руководства пользователя телефонного аппарата. Он содержит ссылку на информационный элемент InOut, описывающий порядок осуществления исходящих звонков и приема входящих. Также в примере имеется специализация базового руководства пользователя – финальный информационный продукт office_phone, который модифицирует информационный продукт phone_manual, определяя руководство пользователя офисного телефона.

Предположим, что в ходе развития семейства появилась необходимость выпуска телефонного аппарата, который не поддерживает входящие звонки (например, это может быть актуально для таксофона). В таком случае целесообразно выделить описание входящих звонков в отдельный информационный элемент – это фрагмент, выделенный полужирным курсовом в примере выше. Если выполнить операцию извлечения информационного элемента для этого

фрагмента, мы получим следующие изменения. Информационный элемент InOut будет выглядеть так (измененный текст выделен полужирным шрифтом):

```
<infelement id=InOut>
  <section><title>Исходящие звонки</title>
    Ваш телефон поддерживает следующие способы набора номера:
    <nest id=”DialOptions”>набор на цифровой клавиатуре</nest>.
  </section>
  <infelemref id="IncomingCalls_ref" infelemid="IncomingCalls"/>
</infelement>
```

На основе выделенного фрагмента текста будет создан новый информационный элемент:

```
<InfElement id="IncomingCalls">
  <section><title>Входящие звонки</title>
      После получения входящего вызова телефон запрашивает
      в сети информацию  о звонящем абоненте и затем
      <nest id=”DisplayOptions”>отображает на экране номер
                                  абонента</nest>.
  </section>
</InfElement>
```

Наконец, рассмотрим изменения финального информационного продукта office_phone (измененный текст выделен полужирным шрифтом):

```
<finalinfproduct name="office_phone" infproductid = "phone_manual">
  <adapter infelemrefid = "InOut_ref">
    <insert-after nestid = "DialOptions">
        или выбор из адресной книги
    </insert-after>
  </adapter>
```

<adapter infelemrefid = "IncomingCalls_ref">

<insert-after nestid = "DisplayOptions">

и имя абонента, если оно задано в записной книжке телефона

</insert-after>

</adapter>

</finalinfproduct>

Как видно по тексту примера, манипуляции с выделенной точкой расширения были перенесены из существующего адаптера в новый, описывающий специализацию вновь созданного информационного элемента.

Глава 4. Инструментальный пакет

4.1 Архитектура инструментального пакета

Архитектура средств инструментальной поддержки DocLine представлена на Рис. 11. Графический редактор DRL/GR – это визуальный редактор, обеспечивающий проектирование и просмотр структуры документации в терминах DRL/GR. Редактор поддерживает работу с тремя видами диаграмм DRL/GR – главной диаграммой, диаграммой вариативности и диаграммой продукта.

Текстовый редактор DRL/PR – это текстовый XML-редактор, поддерживающий редактирование текстов на DRL/PR, импорт документации из внешних источников, а также рефакторинг документации.

Менеджер циклической разработки реализует автоматическую roundtrip-процедуру [13], а именно выполняет интеграцию текстового и графического редакторов, поддерживает документацию в целостном состоянии.

Менеджер публикации обеспечивает преобразование DRL-текстов в формат DocBook. Модуль диагностики ошибок выполняет проверку корректности текстов на DRL с многоступенчатой диагностикой ошибок (проверка осуществляется в несколько этапов – проверка соответствия синтаксису языка DRL, затем проверка ссылочной целостности и, наконец, проверка корректности сгенерированного текста на языке DocBook). Модуль трансляции выполняет преобразование DRL-текстов в DocBook, а также производит проверку корректности сгенерированных DocBook-текстов по запросу модуля диагностики.

Модуль генерации, основываясь на инструментальных средствах DocBook, выполняет генерацию конечных документов в целевых форматах.

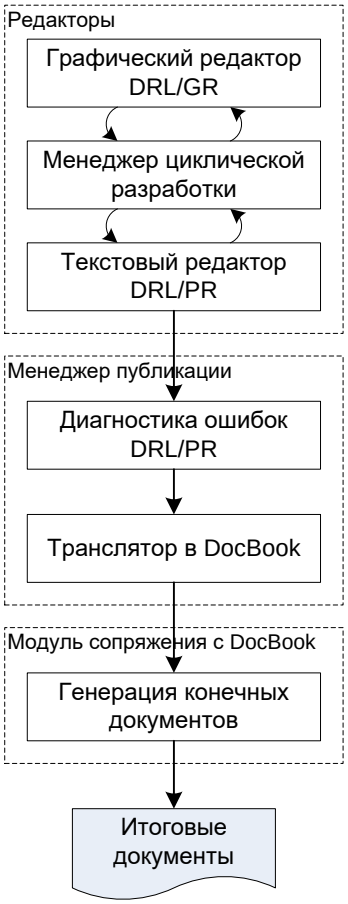


Рис. 11. Архитектура инструментального пакета DocLine.

4.2 Текущий статус разработки

Первая экспериментальная версия инструментального пакета DocLine была реализована на платформе Microsoft.NET в виде отдельного экранного приложения (на Рис. 12 изображен снимок главного окна приложения). В этой версии были реализованы менеджер публикации и модуль сопряжения с DocBook.

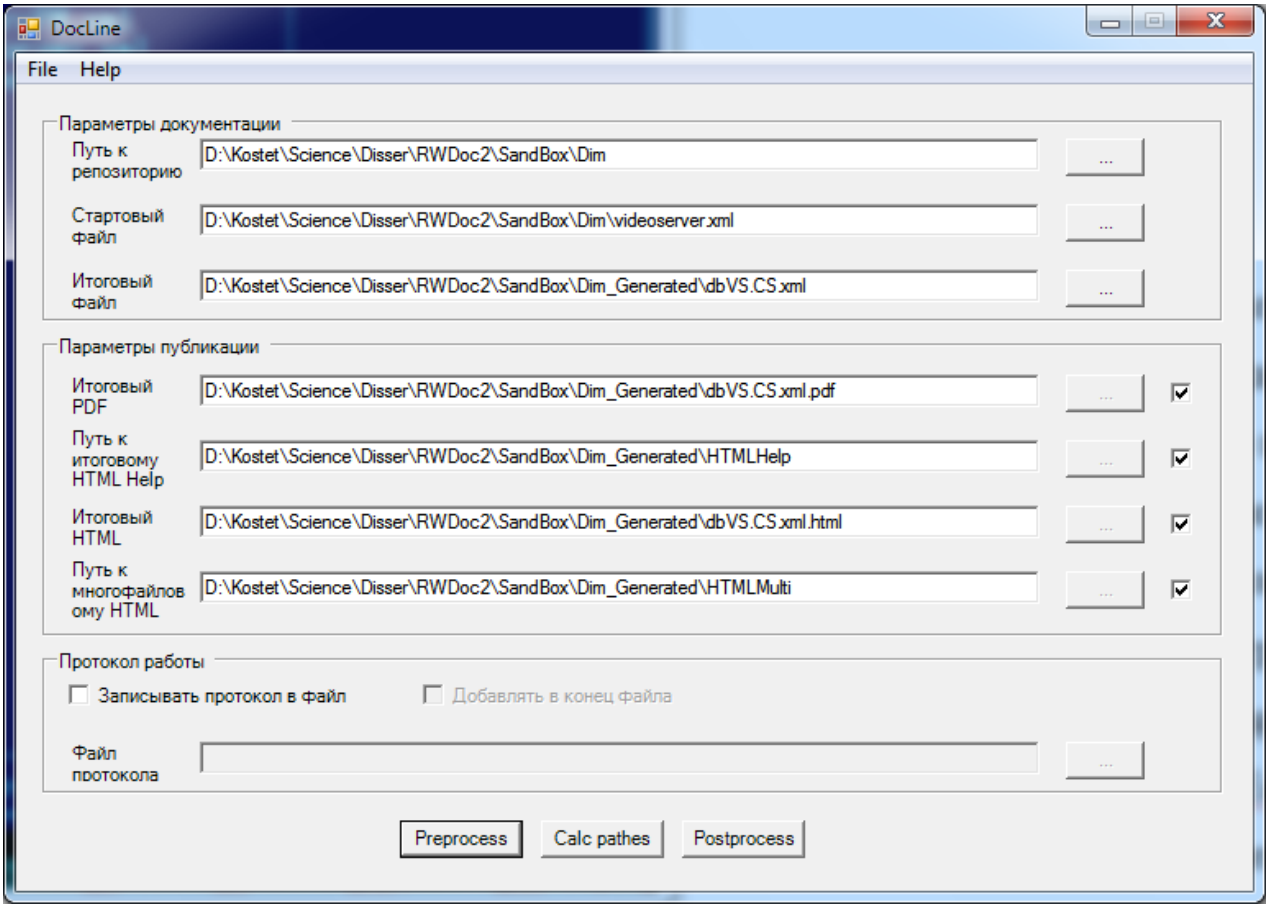


Рис. 12. Главное окно .NET-версии инструментального пакета.

Затем было принято решение перенести дальнейшую реализацию инструментария на открытые Java-технологии. Текущая версия DocLine имеет открытую расширяемую архитектуру и доступна для использования и доработки. Исходный код можно получить в открытом доступе[1].

В Java-версии все инструментальные средства встроены в открытую интегрированную среду разработки Eclipse [56] в виде модулей расширения (plug-ins). Графический редактор реализован с помощью технологии Eclipse GMF, предназначенной для создания инструментария проблемно-ориентированных языков. Для целевых документов поддерживаются форматы HTML и PDF.

[1] URL: http://code.google.com/p/pldoctoolkit/.

Также в инструментальный пакет входят вспомогательные модули, обеспечивающие интеграцию в среду разработки Eclipse.

DocLine позволяет располагать в отдельных файлах различные элементы документации, такие как информационные продукты, информационные элементы, финальные информационные продукты, словари и каталоги. Вместе с тем, в среде Eclipse есть стандартные возможности интеграции с системами контроля версий, такими как Perforce, Microsoft Visual SourceSafe, Subversion и другими. Таким образом, DocLine интегрируется с системами контроля версий, что необходимо в любом промышленном проекте разработки ПО, а также позволяет организовать коллективную работу над документацией.

Рассмотрим более подробно компоненты инструментального пакета.

4.3 Графический редактор и менеджер циклической разработки

Графический редактор DocLine состоит из трех редакторов диаграмм – главной диаграммы, диаграммы вариативности и диаграммы продукта. Также в состав редактора входит подсистема синхронизации диаграмм с текстовым представлением, позволяющая при открытии диаграммы отразить на ней все изменения в тексте, которые были выполнены после последнего ее сохранения.

Графический редактор DocLine создан с помощью технологии Eclipse GMF. Для этого была разработана метамодель языка DRL, предназначенная для автоматизированной генерации графических редакторов (в терминах Eclipse GMF это EMF-модель1). Фрагмент EMF-модели DRL изображен на Рис. 13.

На основе EMF-модели и нескольких вспомогательных моделей (модели графических символов, модели инструментов и связующей модели) был сгенерирован код редакторов, а также функции сохранения и восстановления диаграмм и моделей.

---

[1] Eclipse Modeling Framework URL: http://www.eclipse.org/modeling/emf/.

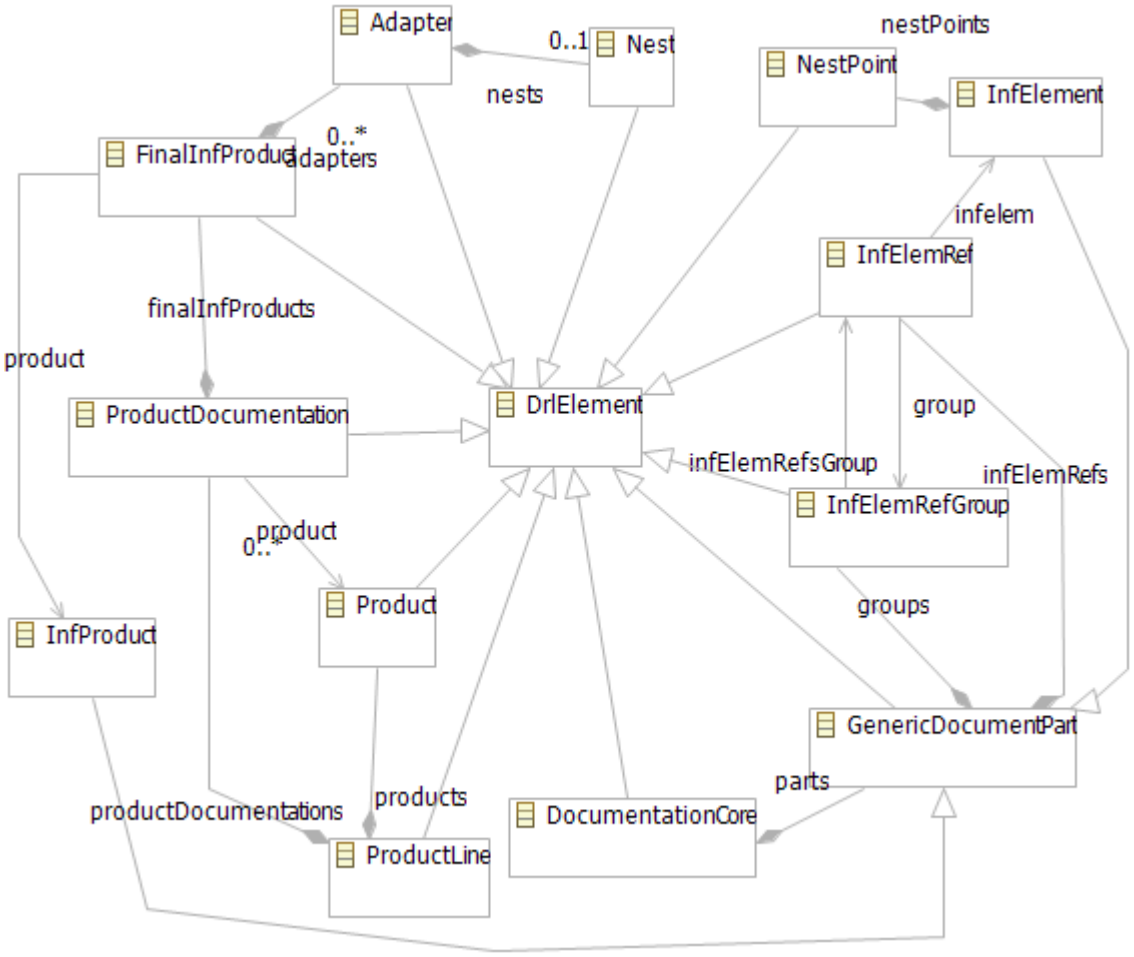


Рис. 13. Фрагмент EMF-модели языка DRL.

В случае DocLine полученное сериализационное представление модели должно соответствовать языку DRL/PR, поскольку в DocLine предусмотрено дальнейшее «ручное» редактирование DRL-спецификации с целью настройки повторного использования и наполнения документации текстом. Eclipse GMF поддерживает раздельное сохранение графической информации (координаты и размеры графических символов) и модельной информации, однако формат сохраняемой модельной информации содержит ряд служебных данных, затрудняющих дальнейшее ручное редактирование. Кроме того, элементы, изображенные на одной диаграмме, могут физически храниться в различных файлах – это также сделано для удобства дальнейшей работы технического писателя над документацией. Для решения этих проблем было предложено использовать до-

полнительное XSLT-преобразование, которое при сохранении диаграммы преобразует ее представление к формату DRL, а при чтении наоборот, добавляет к DRL-спецификации необходимые служебные данные.

Графический редактор интегрирован с текстовым – пользователю предоставляется возможность перейти от элемента диаграммы к его текстовому представлению.

На Рис. 14 показан снимок экрана редактора с открытой диаграммой вариативности.

В текущей версии реализации графического редактора DocLine поддерживается редактирование главной диаграммы и диаграмм вариативности. Для диаграммы продукта в настоящее время поддерживается только режим просмотра.

Поскольку DRL-спецификации имеют два различных представления (DRL/GR и DRL/PR), то рано или поздно возникает ситуация, когда изменения в одном представлении необходимо отразить в другом представлении. В DocLine для этих целей служит менеджер циклической разработки, являющийся, фактически, частью графического редактора. Основная идея, лежащая в его основе – единая модель DRL, на которую есть два взгляда – DRL/GR и DRL/PR. В качестве такой основной модели была взят DRL/PR. DRL/GR-спецификации могут быть восстановлены по имеющейся DRL/PR-спецификации, при этом потеряются только сведения о размерах и расположении графических элементов. В связи с этим, случаи противоречивых изменений двух представлений редки и могут быть обработаны вручную. Более часто встречаются ситуации, когда изменения внесены в одно из представлений и должны быть отображены в другом. В случае, когда изменения вносятся в DRL/GR-представление, эти изменения автоматически отражаются в соответствующих конструкциях DRL/PR в момент сохранения диаграммы. В случае, когда изменяется DRL/PR-представление, эти изменения будут отображены при очередном открытии DRL/GR-диаграммы, которую они затрагивают.

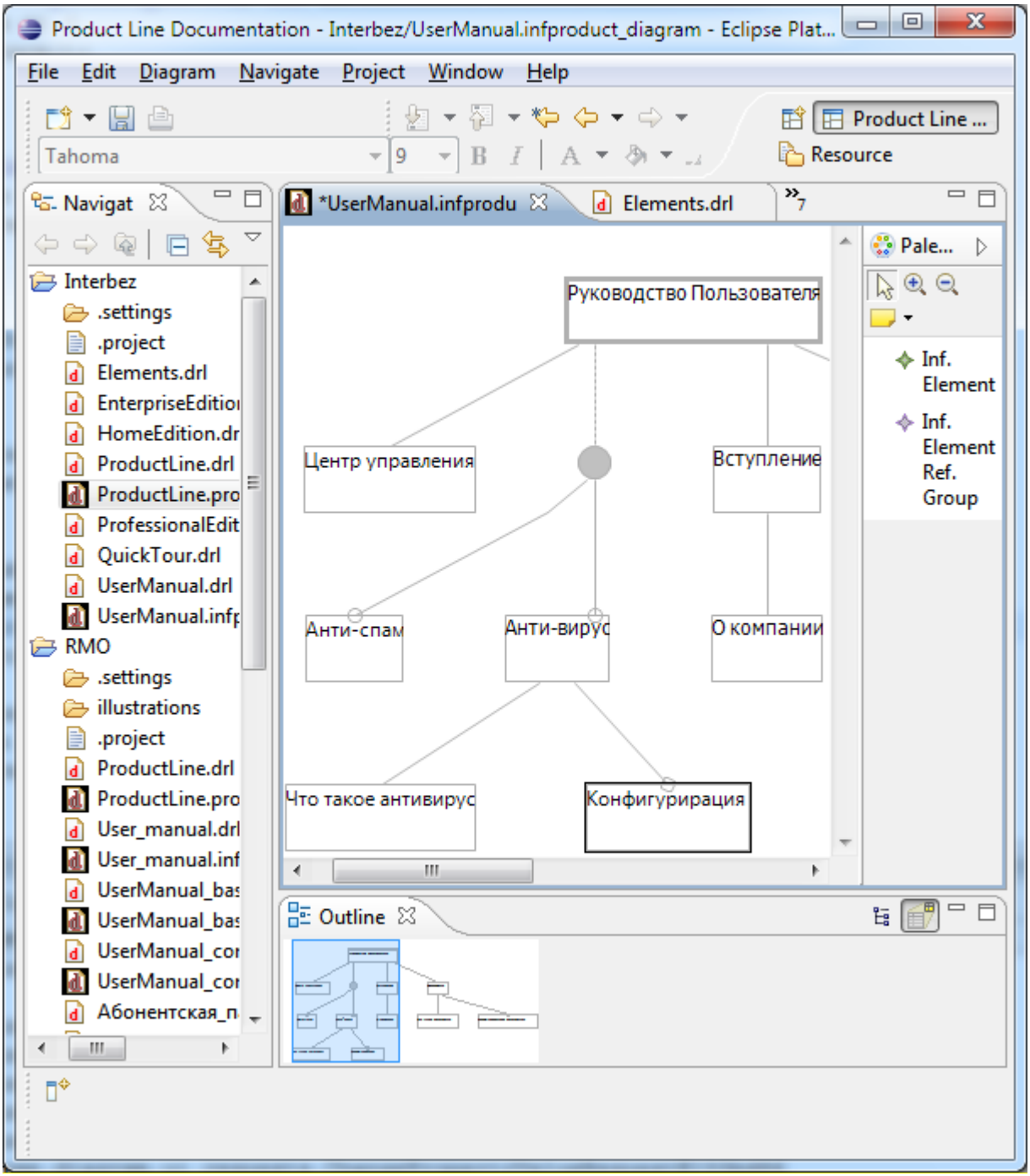


Рис. 14. Графический редактор DocLine:

диаграмма вариативности.

4.4 Текстовый редактор

Текстовый редактор расширяет функциональность стандартного XML-редактора с открытым исходным кодом, входящего в платформу Eclipse, адаптируя его с учетом потребностей DocLine. На Рис. 15 показан снимок экрана с открытым DRL-документом.

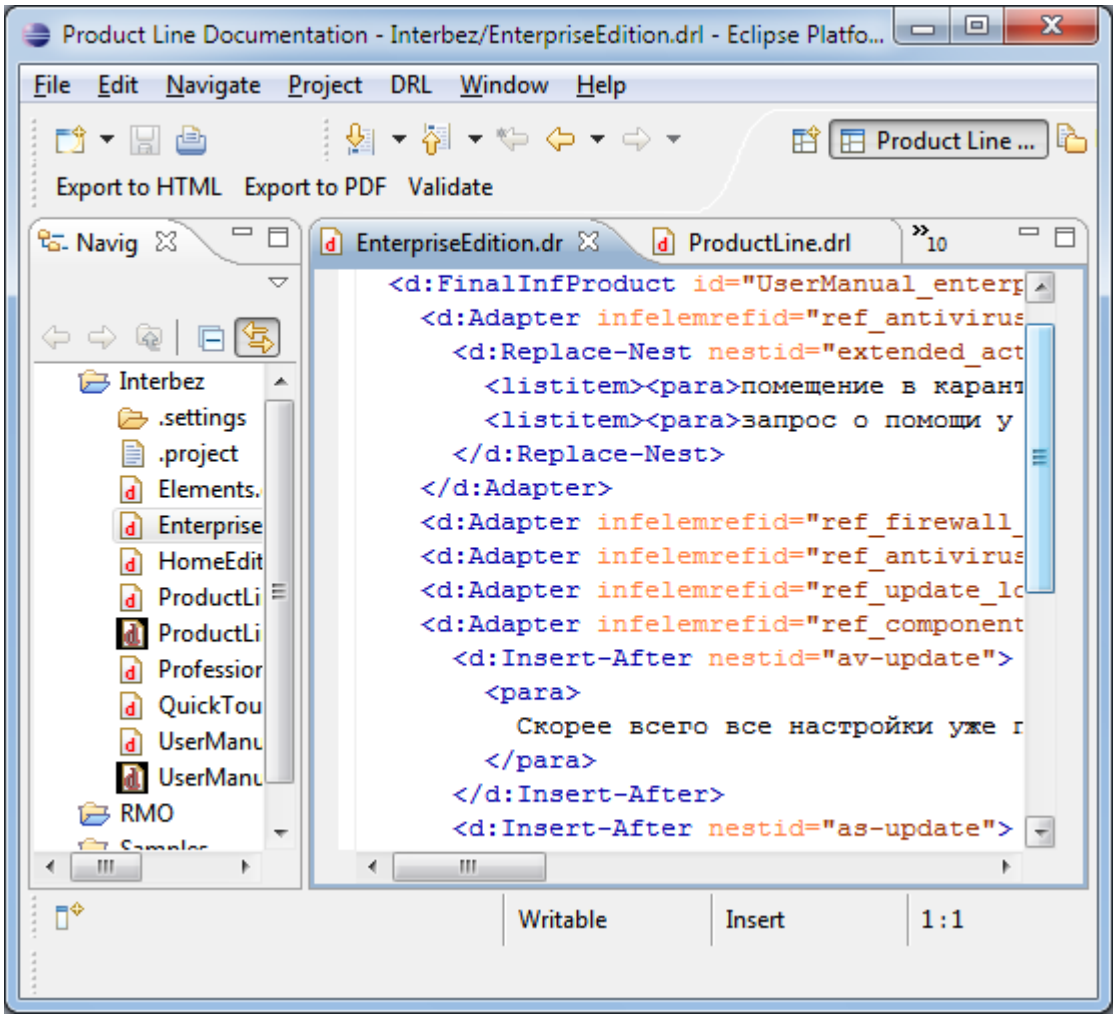


Рис. 15. Текстовый редактор DocLine: DRL-документ.

В редакторе реализованы следующие функции:

- подсветка синтаксиса конструкций,
- автоматическое завершение начатых конструкций,
- подсказка и автоматическая вставка атрибутов конструкций,
- вставка шаблонов конструкций,

- автоматическое закрытие открытого XML-тега.

4.5 Рефакторинг

Поддержка рефакторинга разделена на два архитектурных слоя – библиотека базовых функций и собственно сами операции рефакторинга. Библиотека базовых функций реализует чтение и запись DRL-документации с учетом ее многофайловой структуры, построение дерева разбора и операции поиска и обхода конструкций в дереве разбора и др. На основе этой библиотеки реализованы следующие операции рефакторинга:

- импорт документации DocBook,
- извлечение информационного элемента,
- расщепление информационного элемента,
- преобразование в точку расширения,
- преобразование в Insert-After/Before,
- объявление ссылки на информационный элемент обязательной и наоборот,
- извлечение в словарь,
- извлечение в каталог.

4.6 Публикация конечных документов и проверка корректности

Для преобразования DRL-документации к виду целевых документов применяется процесс публикации, включающий генерацию DocBook-документа и последующую публикацию итоговых документов средствами инструментального пакета DocBook [55]. Общая схема процесса публикации показана на Рис. 16. На первом шаге из набора DRL-документов создается единый DocBook-документ. В качестве точки старта задается финальный информационный продукт – именно для него и выполняется публикация целевого документа. Далее средствами инструментального пакета DocBook выполняется публикация ито-

гового документа в целевом формате. В текущей реализации DocLine поддерживаются форматы HTML для электронной документации и PDF для печатной.

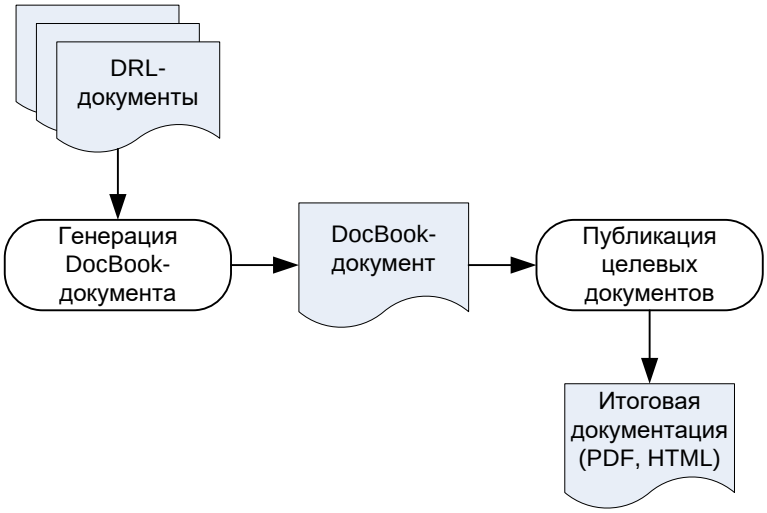


Рис. 16. Схема процесса публикации итоговых документов.

Поскольку DocLine допускает «ручное» редактирование DRL-документов, в результате ошибки технического писателя разрабатываемая документация может стать синтаксически некорректной (подразумевается корректность с точки зрения языков разметки DRL/PR и DocBook, проверка синтаксиса и орфографии естественного языка лежит за рамками данной работы). Для DRL/PR существует схема в формате Relax NG, по которой можно проверить корректность DRL-спецификации. Однако тут есть существенная проблема – дело в том, что внутри DRL-конструкций встречаются конструкции DocBook, причем в рамках каждого модуля соответствующий фрагмент DocBook-текста может быть некорректен, тогда как при сборке итогового документа он будет соединен с другими фрагментами и вместе получится корректный DocBook-документ. И наоборот – каждый фрагмент может быть сам по себе корректен, но при соединении с другими может получиться ошибочный текст. В результате проверка корректности отдельного файла, которую можно выполнить по схеме документа, не позволяет обоснованно судить о корректности документа-

ции в целом. Для решения этой проблемы в DocLine реализован многоступенчатый механизм проверки корректности – см. Рис. 17.

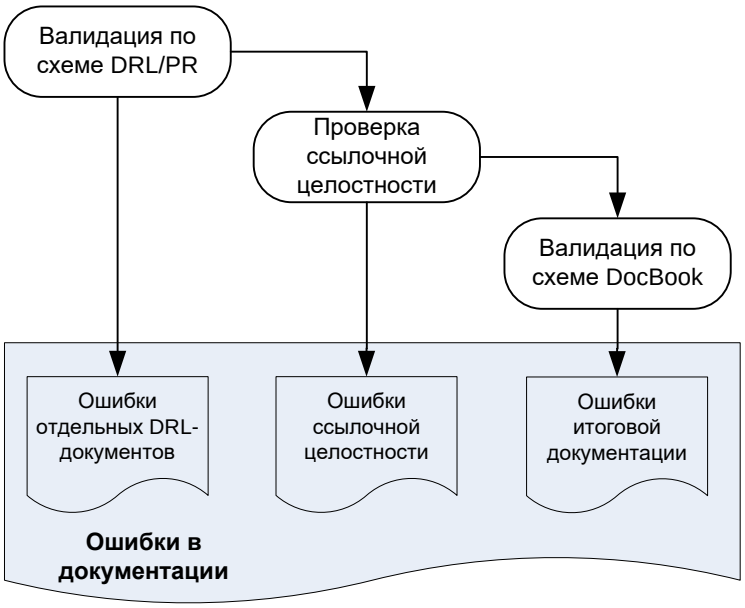


Рис. 17. Схема процесса многоступенчатой валидации DRL-текстов.

Первый этап валидации – проверка текущего документа по схеме DRL. Этот шаг выполняется при каждом сохранении DRL-документа и позволяет проверить корректность на уровне конструкций DRL/PR, не погружаясь в текст, расположенный внутри DRL-конструкций. Следующий этап выполняется в процессе генерации DocBook-документа по набору DRL-документов – тут проверяется ссылочная целостность документации, т.е. что для всех ссылок существуют элементы, на которые идет ссылка. Наконец, полученный документ проверяется по схеме DocBook. Проверка ссылочной целостности и валидация по схеме DocBook выполняются в процессе публикации, или же по специальному запросу пользователя (командой панели инструментов DocLine). Все найденные ошибки выводятся в соответствующей панели интегрированной среды Eclipse с привязкой к исходному тексту. Далее уже средствами Eclipse можно перемещаться от описания ошибки к соответствующей позиции в DRL-файле.

Глава 5. Апробация

5.1 Объект апробации

Для апробации DocLine было выбрано промышленное семейство телекоммуникационных систем, выпускаемых компанией ЗАО «Ланит-Терком». Базовый продукт семейства – программное обеспечение электронной автоматической телефонной станции типа «Квант-Е» (далее – станция или АТС). На основе базового продукта разработан ряд модификаций, включая сельские, офисные, городские, транзитные АТС, а также специализированные станции (например, с поддержкой IP-телефонии). К настоящему моменту в телефонных сетях России установлены сотни экземпляров представленного семейства1. Апробация проводилась для двух продуктов семейства – базовой городской телефонной станции (далее - ГАТС), а также станции специального назначения (далее в тексте - САТС). Апробация проводилась на примере руководства пользователя для одной из подсистем рассматриваемого ПО – рабочего места оператора станции (далее – РМО), обеспечивающего поддержку технического обслуживания и эксплуатации станции.

САТС является, фактически, «вырезкой» из ГАТС, которая была доработана под индивидуальные требования заказчика. Относительно документации заказчик выдвинул требование, чтобы в его комплекте были описаны только те функции, которые вошли в его комплект поставки, а все остальные, общие, были из описания убраны. Так, из описания РМО всего семейства станций появился документ, описывающий только РМО САТС. Именно к двум этим документам и была применена технология DocLine.

---

[1] Более подробно о телефонных станциях «Квант-Е» см. [3].

5.2 Ход апробации

Основной задачей апробации было выполнить перенос существующей документации семейства телефонных станций «Квант-Е», созданной в Adobe FrameMaker, в DocLine с организацией повторного использования документации. Как следует из описания семейства, в этой документации должны быть повторы текста.

В апробации участвовали технический писатель и консультант по семейству телефонных станций. До начала апробации технический писатель не был знаком ни с пакетом DocLine, ни с технологией DocBook, ни с XML, ни с исходным семейством. Руководил апробацией автор данной диссертационной работы, он же разрешал технические вопросы, связанные с эксплуатацией программного продукта DocLine.

В ходе апробации выполнялись следующие работы:

- изучение и анализ предметной области,
- планирование повторного использования,
- выделение и спецификация переиспользуемых компонент,
- задание форматирования документов средствами DocBook.

Работа проводилась итеративно, т.е. в разные моменты апробации происходил возврат к тем или иным видам деятельности. Например, изучение и анализ предметной области производились в течение всего проекта, по мере необходимости.

В целом трудозатраты на проект оказались следующими: 20 дней работы технического писателя, 2 дня работы консультанта по семейству телефонных станций, 2 дня работы автора DocLine.

Рассмотрим отдельные работы, выполненные в ходе апробации.

5.2.1 Изучение и анализ предметной области

Эта часть проекта оказалась очень важной по следующим причинам. Технический писатель, организующий повторное использование, должен хорошо разбираться в функциональности описываемых продуктов – существенно лучше, чем при написании обычных документов. В последнем случае технические писатели часто подходят к написанию документации формально, не пытаясь анализировать детально специфику продуктов, а описывают отдельные окна приложений и их свойства. Такую же склонность продемонстрировал и наш технический писатель, и потребовалось немало сил, чтобы убедить его в необходимости детально изучить описываемые продукты. Поиск похожих и почти похожих фрагментов текста можно производить синтаксически, но гораздо эффективнее понимать, чем продукты отличаются, и подходить к изменениям в документах от изменений в функциональности. В целом изучение и анализ предметной области занял 3 дня работы технического писателя и 1 день работы консультанта по семейству телефонных станций.

5.2.2 Планирование повторного использования

Для планирования повторного использования в DocLine предназначен DRL/GR. На Рис. 18 представлена главная диаграмма для семейства АТС, на которой отображены продукты семейства (левая секция) и единственный имеющийся у нас информационный продукт.

Рис. 18. Главная диаграмма семейства АТС.

Затем технический писатель с помощью диаграмм вариативности начал изображать общую структуру документов – главы, разделы, подразделы т.д. и хотел уже на этой картине отмечать повторно используемые компоненты. Однако такой подход был отброшен, поскольку диаграммы получались слишком большие, что затрудняло их изучение. После того, как на диаграмме вариативности (Рис. 19) были представлены разделы первого уровня, дальнейшая декомпозиция проводилась только для тех разделов и подразделов, внутри которых присутствовала вариативность.

Результаты этого этапа – диаграммы на Рис. 18, Рис. 19, Рис. 20. Таким образом, мы выявили варианты крупноблочного повторного использования (т.е. какие разделы присутствуют/отсутствуют в документации для разных продуктов), а также, в каких разделах есть незначительные отличия. Эта часть апробации заняла 5 дней работы технического писателя, 1 день работы консультанта по семейству телефонных станций и половину дня работы специалиста по DocLine.

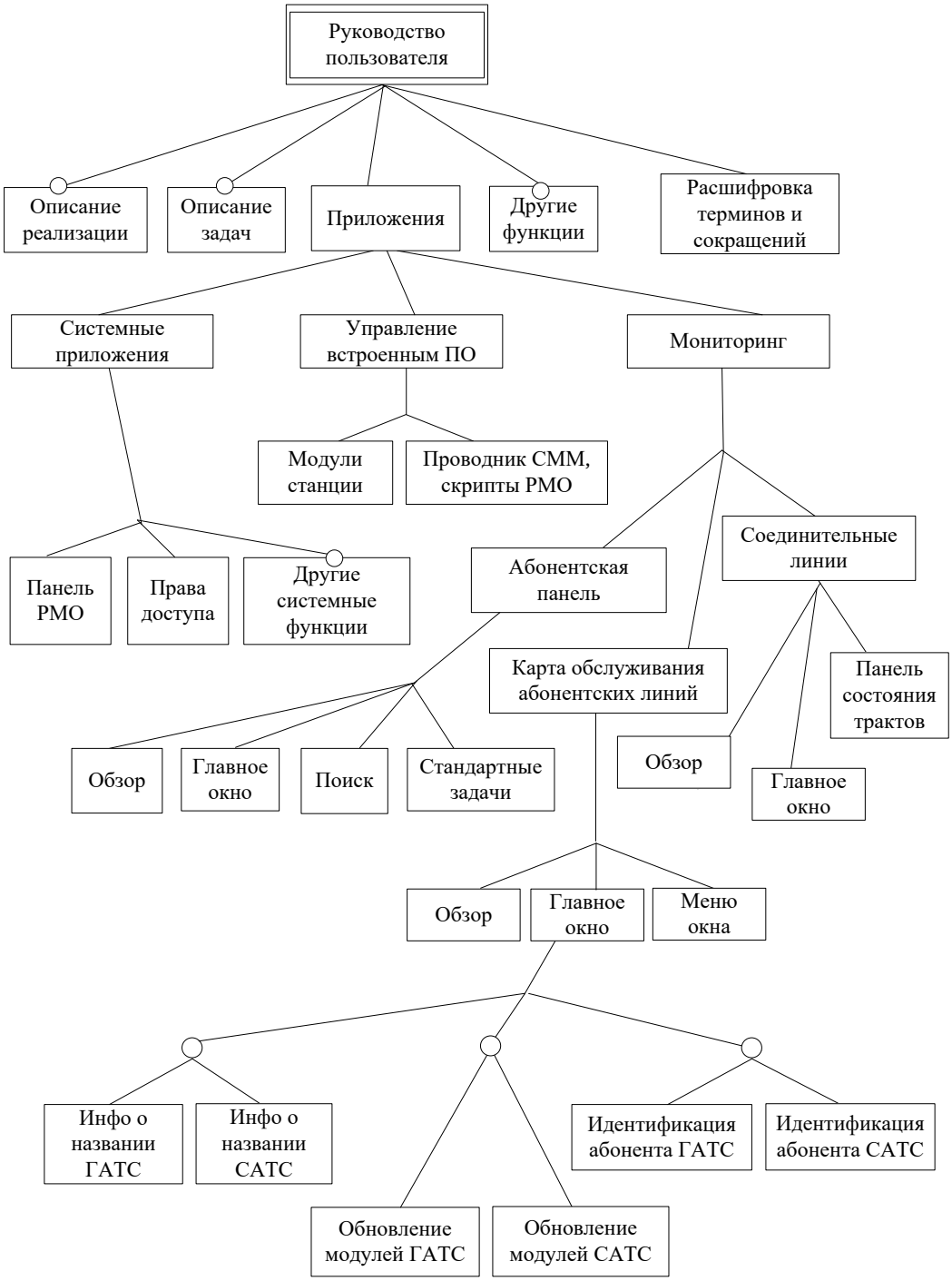


Рис. 19. Диаграмма вариативности для руководства пользователя.

5.2.3 Выделение и спецификация переиспользуемых компонент

Для разделов документации, в которых для ГАТС и САТС имелись отличия (такие разделы были найдены при планировании повторного использования) средствами DRL/PR была проведена настройка адаптивности. Они были

выделены в информационные элементы, в них вставлялись необходимые точки расширения, создавались специализированные информационные элементы – те, которые входили в конечные документы. Эта деятельность была тесно связана с анализом и изучением предметной области и оказалась самой трудоемкой, заняв 8 дней работы технического писателя и 1 день работы специалиста по DocBook. Работа была существенно облегчена благодаря средствам рефакторинга, имеющимся в DocLine, автоматизировавшим такие операции как выделение информационного элемента, извлечение текста в точку расширения, выделение элемента в словарь и т.п.

5.2.4 Задание форматирования документов средствами DocBook

Требовалось, чтобы внешний вид документации после применения DocLine был максимально близок к исходному. Поэтому нужно было задать необходимые свойства форматирования средствами DocBook и настроить подходящим образом утилиты публикации DocBook. Это заняло половину дня работы специалиста по DocLine, так как выяснилось, что тонкая настройка DocBook достаточно сложна для технического писателя, не имеющего ранее опыта работы с XML-технологиями. Затем технический писатель выполнил разметку текста документации средствами DocBook, на это затратилось 4 дня.

5.3 Особенности использования DocLine

Рассмотрим, какой оказалась вариативность текстов в ходе апробации, и как удалось ее выразить средствами DocLine.

5.3.1 Вариативность продуктов семейства

Вариативность текстов прямо следует из вариативности функциональности продуктов, которые описываются этими текстами. Как уже говорилось выше, исторически САТС разрабатывалась как ограниченная версия ГАТС и уже в процессе эволюции появились разнообразные отличия в функциях, реализован-

ных в обеих системах. Таким образом, в нашем случае можно выделить базовый продукт и его документацию (ГАТС) и продукт-потомок (САТС). Рассмотрим следующие виды вариаций функциональности (и документации), характерные для такого подхода.

- *Удаление* – из базовой версии системы удаляется функциональность, которая не требуется пользователям нового продукта. Соответственно описание этой функциональности удаляется из документации. Это может быть отдельный фрагмент текста, целая глава или набор глав. К сожалению, нередко текст, который требуется удалить, распределен по различным разделам документации – где-то удаляется глава, где-то отдельные абзацы, элементы списка или строчки таблиц и т.п. Так происходит потому, что описание удаляемой функциональности часто не локализовано в тексте, а распределено по разным его местам.
- *Добавление* – в базовую версию добавляется новая функциональность, и ее описание должно быть внесено в пользовательскую документацию. Как и в случае с удалением, изменения текста могут затрагивать целый ряд разделов документации. Эта операция не характерна для рассматриваемого случая, так как в САТС не произошло добавления каких-либо существенных новых функциональных возможностей по сравнению с ГАТС.
- *Изменение* – значительно меняется какое-либо функциональное свойство, что для документации выражается в требовании описать измененные атрибуты этого свойства в виде блока подряд идущего текста либо изменения рассредоточены в разных фрагментах документа.
- *Локальное изменение* – меняется лишь отдельный атрибут функционального свойства, например количество обслуживаемых абонентов. Для пользовательской документации такое изменение функциональности превратится в небольшое локальное изменение текста. Например, при изменении количе-

ства обслуживаемых абонентов будет изменено число в сводной таблице функциональных возможностей.

Есть также особый вид изменений, не затрагивающий функциональность – это *«параллельные места»*: одна и та же функциональность описывается в разных документах различными словами. «Параллельные места», в целях унификации документов, нужно по возможности убирать, выбирая единый способ описания одной и той же информации. Эта рекомендация не относится к ситуации, когда технический писатель осознанно выбирает различные способы изложения информации, например, ориентируясь на подготовку читателя или степень детальности соответствующего документа.

Еще раз подчеркнем, что удаление, добавление и изменение особенны тем, что некоторая информация (описание одной функциональной возможности) не локализована в тексте, поэтому ее добавление или удаление требует особой внимательности. Ошибки здесь могут привести к потере корректности текста.

В Табл. 1. приведены процентные соотношения указанных типов изменений в документации РМО ГАТС и САТС.

Характер изменений	**%**
Удаление	32
Добавление	0
Изменение	5
Локальное изменение	26
«Параллельные места»	37

Табл. 1. Процентное соотношение типов изменений.

5.3.2 Схема вариативности документации

Диаграмма вариативности для документации семейства АТС показана на Рис. 19. Фактически, диаграмма вариативности показывает общее дерево разде-

лов для руководств пользователя двух продуктов семейства. Проработку диаграммы вариативности «в глубину» целесообразно ограничить и отображать только информационные элементы, которые варьируются крупноблочно (т.е. входят или нет целиком) или содержат внутри себя мелкие отличия для двух документов. Остальные разделы (см., например, многие разделы первого уровня на Рис. 19) также выносятся в информационные элементы, но далее не раскрываются в DRL в иерархию, хотя размечаются средствами DocBook с целью задания требуемого форматирования. Таким образом, «листьями» на диаграммах вариативности являются разделы, которые идентичны во всех контекстах применения, либо разделы, которые варьируются, но не включают других разделов – вообще, либо варьирующихся.

В документации раздел «Главное окно» почти идентичен для обоих типов станции, но содержит несколько фрагментов, которые различаются лишь порядком расположения. Для описания такой вариативности было решено выделить «перемещающиеся» фрагменты в информационные элементы и объединить в XOR-группы – каждая такая группа описывает вхождения одного фрагмента в различные точки объемлющего блока.

На Рис. 20 представлена диаграмма продукта для РМО САТС. Таким образом, эта диаграмма является «вырезкой» для САТС из общей модели документации (модели вариативности).

5.3.3 Настройка адаптивности

Рассмотрим пример адаптивности в документации семейства АТС. В руководствах пользователя РМО ГАТС и САТС есть несколько общих подразделов («Абонентская панель», «Карта обслуживания абонентских линий», «Соединительные линии»), которые содержат ряд отличий, не разрешаемых на уровне включения или исключения целых информационных элементов. В DocLine для

формализации таких отличий используются точки расширения и механизм адаптации.

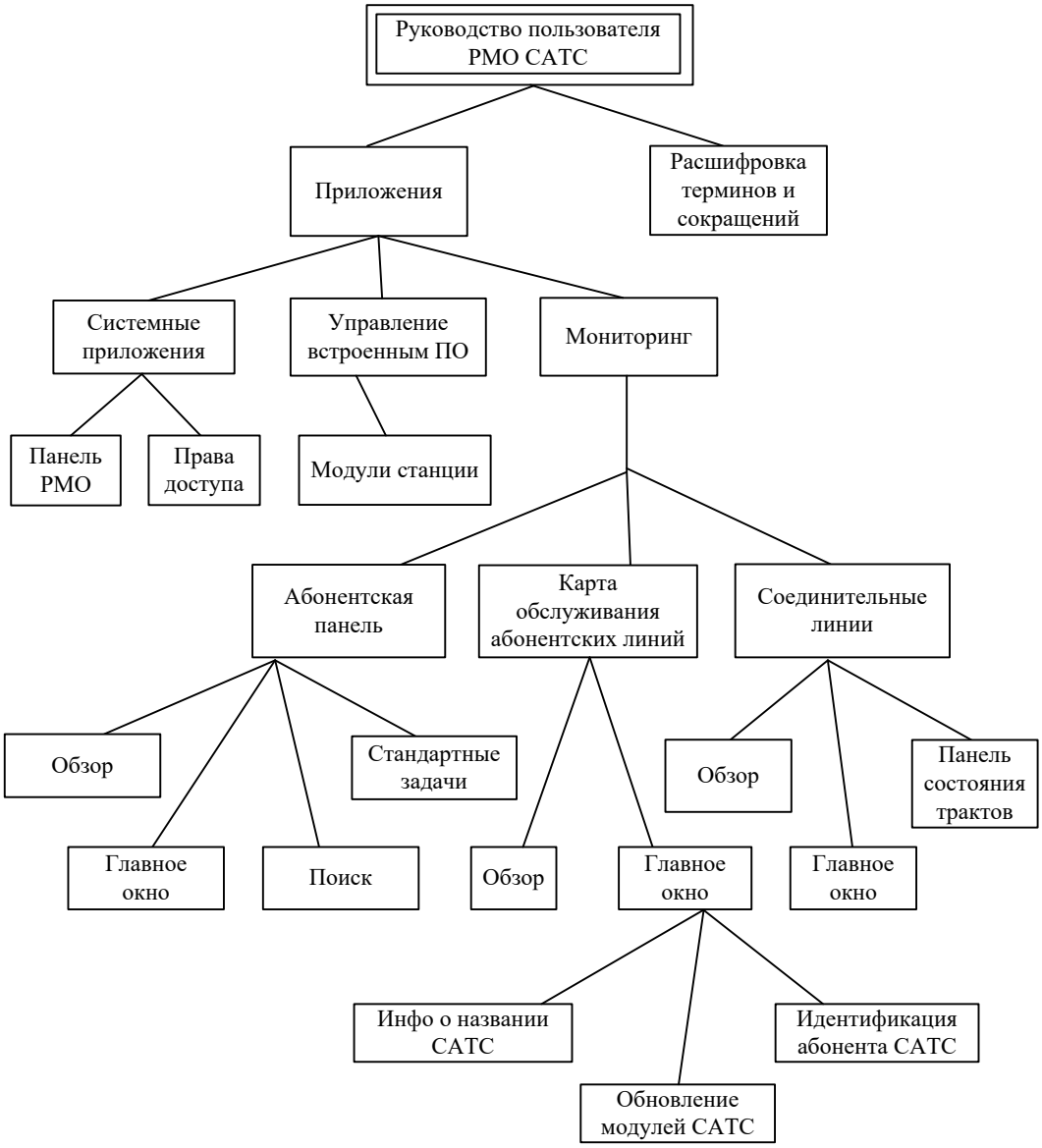


Рис. 20. Диаграмма продукта для руководства пользователя РМО САТС.

В Табл. 2 показан фрагмент документации из раздела «Карта обслуживания абонентских линий», описывающий управление автоматическим обновлением состояний абонентских линий. Содержательная разница между реализацией этой функции для САТС и ГАТС состоит в том, что для САТС необходимо обеспечивать более частые обновления, при этом в ГАТС обновляется состояние только явно запрошенных абонентских модулей, а в САТС – всех мо-

дулей, которые были доступны при последнем обмене данными со станцией. Различающиеся детали выделены полужирным курсивом.

РМО ГАТС	**РМО САТС**
Чтобы активизировать автоматическое обновление состояний абонентских линий необходимо нажать на правую часть этой кнопки и в появившемся меню выбрать необходимый период времени обновления (из представленных: 5, 10 или 30 секунд). При этом будут обновляться состояния абонентов, которые ***заказаны в списке опрашиваемых модулей***.	Чтобы активизировать автоматическое обновление состояний абонентских линий необходимо нажать на правую часть этой кнопки и в появившемся меню выбрать необходимый период времени обновления (из представленных: ***1***, 5, 10 или 30 секунд). При этом будут обновляться состояния ***только тех*** абонентов, которые ***были доступны при последней загрузке данных из станции***.

Табл. 2. Пример локальных изменений в тексте.

Фактически, фрагменты очень похожи, но не идентичны. В исходной документации версия для САТС была получена копированием и последующим исправлением фрагмента для ГАТС. Рассмотрим, как такие «точечные» изменения можно описать с помощью DRL. Фрагмент текста, описывающий автоматическое обновление, будет выглядеть следующим образом:

<InfElement id=AutoUpdate>

Чтобы активизировать автоматическое обновление состояний абонентских линий необходимо нажать на правую часть этой кнопки и в появившемся меню выбрать необходимый период времени обновления (из представленных:
<Nest id=Period>*5, 10 или 30 секунд****</Nest>****). При этом будут обновляться состояния*
<Nest id=Condition /> *абонентов, которые*
<Nest id=Constraint>заказаны в списке опрашиваемых модулей</Nest>.

</InfElement>

Все позиции, где допустимы изменения, отмечены точками расширения (конструкция <nest/>).

В документацию РМО ГАТС указанный элемент будет включен «как есть», а для документации РМО САТС требуется описать ряд изменений:

<Adapter infelemrefid=AutoUpdate_ref>

*<Insert-Before nestid=Period>**1,** </Insert-Before>*

*<Insert-Before nestid=Condition>**только тех**</Insert-Before>*

*<Replace-Nest nestid=Constraint>**были доступны при последней загрузке данных из станции**</d:Replace-Nest>*

</Adapter>

Изменения описываются с помощью конструкции <Adapter/>, в которой указываются точки расширения, требующие модификации и, собственно, сами модификации – в данном случае добавление данных о периоде обновления, и изменение описания множества обновляемых модулей.

Как видно из примера, DRL позволяет использовать в нескольких документах похожие, но не идентичные фрагменты текста – это и есть реализация адаптивности повторного использования (то есть возможности адаптироваться под требования контекста). В данном случае с помощью механизма адаптации DocLine формализовано изменение вида «локальное изменение».

5.4 Анализ результатов апробации

Первый результат апробации заключается в том, что в реальной документации продуктов семейства действительно много повторов, которые варьируются в деталях (см. вышеприведенные примеры). Таким образом, адаптивное повторное использование документации действительно оправдано и механизмы адаптивности (точки расширения и адаптеры) сыграли существенную роль в успехе апробации – без них в большинстве случаев повторное использование было бы невозможно, даже в рассмотренной ситуации, когда один из продуктов семейства был изначально построен как «вырезка» из базового продукта.

Относительно применимости языка DRL по результатам данной апробации можно сказать следующее. DRL/GR оказался востребован, особенно диаграммы вариативности. Именно с их помощью происходило планирование повторного использования, а также обсуждение его результатов между техническим писа-

телем, специалистом по DocLine и консультантом по семейству телефонных станций. Это особенно важно на первых порах, пока технический писатель еще не овладел до конца приемами практического применения DocBook. Вся дальнейшая организация повторного использования происходит на основе диаграмм вариативности. Диаграммы продуктов были полезны, хотя и в меньшей степени, главная диаграмма в нашем случае оказалась небольшой и малосодержательной.

Из DRL/PR активно применялись информационные элементы, точки расширения и адаптеры. Для унификации терминологии были задействованы словари. Каталоги практически не потребовались, поскольку в документации не было массово встречающихся однородных объектов.

Таким образом, использование DocLine предполагает высокие требования к квалификации технических писателей – необходимо, чтобы у них были способности к системному мышлению и широкий кругозор в рамках данного проекта для того, чтобы они могли организовывать повторное использование, не только основываясь на синтаксическом подобии отдельных фрагментов текста, но и исходя из свойств описываемого ПО. Требуются также ресурсы для организации и осуществления дополнительных коммуникаций между техническими писателями и другими участниками проекта (менеджерами, разработчиками, архитекторами) для эффективной передачи знаний о продуктах семейства – об их схожих свойствах и различиях.

Программные средства DocLine показали себя достаточно удобными для эффективной разработки документации. Основные сложности возникали с редактированием текста на XML и настройками DocBook – в следующих версиях программного пакета планируется упростить такие работы.

В целом выяснилось, что применение DocLine при наличии модели вариативности практически ничем не отличается по трудоемкости от применения, например, DocBook, однако при этом DocLine добавляет средства планирова-

ния и механизм адаптивного повторного использования. Основная сложность применения DocLine – необходимость техническому писателю изучить основы XML. Основные предположения об эффективности DocLine относятся к возможности применения подобных XML-технологий при разработке технической документации в принципе. Преимущества применения XML в средних и крупных компаниях перевешивают затраты на его внедрение (см., например, [12]). В западных университетах специально готовят технических писателей (специальность «технические коммуникации»), в частности учат их XML-технологиям. Такие XML-технологии как DocBook и DITA активно применяются в индустрии разработки ПО[1]. В то же время, многие технические писатели и в России используют для верстки книг продукт TeX, который намного сложнее, чем DocLine.

Затраты на внедрение DocLine оправданы, если требуется разработать ряд похожих документов, и, главное, есть процесс их эволюции – вносятся исправления и доработки, а также дополнения. В такой ситуации структура повторного использования, предлагаемая DocLine, позволяет внести изменения в одно место – во все остальные они распространятся автоматически. В случае же, когда документов мало и они статичны (разрабатываются один раз и навсегда), или же нет высоких требований к качеству документации, применение DocLine не даст преимуществ перед DocBook, однако сохранит возможность организации адаптивного повторного использования документации в дальнейшем, если на каком-либо этапе развития СПП ситуация изменится.

---

[1] Список промышленных компаний, использующих DocBook: http://wiki.docbook.org/topic/WhoUsesDocBook.

Глава 6. Сравнения и соотнесения

DocLine базируется на методе фреймов Бассета-Ерзабека [15], [36], фактически предлагая его модификацию, адаптированную для разработки документации. По сравнению с оригинальным методом Бассета [15] и его расширением [36], в DocLine поддерживается форматирование документации средствами DocBook, многоступенчатая проверка корректности текстов, а также ряд конструкций, специфичных для документации, таких как словарь и каталог.

Еще одно нововведение DocLine – явное разделение проектирования повторно-используемой документации и порождения на ее основе документации конкретного продукта. Так в методе Бассета-Ерзабека при подключении повторно-используемого актива необходимо в точке включения задать его адаптацию к особенностям контекста. В DocLine можно обозначить факт использования общего актива в том или ином модуле (в частности, этот модуль также может быть общим активом!) и выполнить его предварительную настройку, а окончательно описать все особенности использования можно уже с помощью адаптера при создании документации конкретного продукта.

Для поддержки визуального проектирования схемы повторного использования документации используется модифицированная нотация диаграмм возможностей [26]. Визуальные средства проектирования повторного использования отсутствуют в исходном методе Бассета-Ерзабека. С другой стороны, диаграммы возможностей сами по себе не имеют исполнимой семантики, в то время как DocLine, предлагает вполне конкретную интерпретацию символов диаграмм, реализующую часть документации СПП. Чтобы визуальная модель оставалась «обозримой», в DocLine предлагается использовать ее именно для структуры повторного использования, а не для структуры всей документации (хотя в ряде случаев целесообразно отдельные модули выносить на уровень ви-

зуальной модели, например для выделения фрагментов редактируемых различными техническими писателями).

Главное отличие DocLine от технологий разработки документации DocBook, DITA, FrameMaker – это поддержка модульного адаптивного повторного использования. DocLine расширяет DocBook, добавляя туда эти возможности, а также средства визуального проектирования схемы повторного использования. DITA в оригинале предполагает модульную организацию исходного представления документации, удобную для повторного использования, однако не поддерживает адаптивность повторного использования и не имеет средств визуального проектирования. То же самое верно и для Adobe FrameMaker – в базовом варианте адаптивность не поддерживается, однако с помощью расширений ее можно реализовать, в частности одно из направлений дальнейшего развития DocLine – разработка версии инструментального пакета для Adobe FrameMaker.

Основное преимущество существующих подходов перед DocLine – это реализованная поддержка WYSIWYG-редактирования[1] текста документации. Однако на практике выясняется, что для использования даже простейших возможностей повторного использования, предлагаемых такими средствами, WYSIWYG-представления уже не достаточно, и приходится так или иначе «погружаться» в служебное представление документации – как правило, это означает текстовое редактирование XML-текста или диалоговое редактирование многочисленных свойств конструкций. В перспективе для DocLine также можно реализовать режим WYSIWYG-редактирования (экспериментальная реализация под Adobe FrameMaker уже имеется, однако выходит за рамки данной

---

[1] WYSIWYG (What You See Is What You Get) – разновидность редакторов, позволяющих непосредственно во время создания документа видеть его на экране таким же, как он будет выглядеть в напечатанном виде.

диссертационной работы). Для упрощения работы с XML в Eclipse-версии инструментального пакета DocLine предлагаются шаблоны основных конструкций DRL/PR и автоматическое завершение начатых фрагментов конструкций DRL/PR и DocBook.

В

Табл. 3 в сводной форме отмечены основные отличия DocLine от существующих подходов. Сравнение производится по следующим критериям.

- Повторное использование – означает поддержку повторного использования контента.
- Адаптивное повторное использование – поддержка возможности настройки общих активов в соответствии с требованиями контекста использования.
- Визуальная схема повторного использования – наличие средств проектирования схемы повторного использования с помощью техник визуального моделирования.
- Разделение точек подключения и настройки общего актива – возможность задавать настройки общих активов непосредственно при создании документации конечного продукта, а не в момент создания ссылки на соответствующий модуль.
- Модульная документация – поддержка разбиения документации на обособленные модули.
- Разделение контента и формата – возможность при написании текстов оперировать абстрактными понятиями, такими как «глава», «заголовок» и т.п., не погружаясь в особенности настройки отображения (шрифты, размер символов, расположение и т.п.) соответствующих элементов. Настройки отображения в свою очередь могут быть заданы отдельно для всего документа.

- Единое исходное представление – поддержка порождения нескольких целевых документов на основе единого содержимого, но в различных форматах (HTML, PDF и др.).
- Форматирование документации – наличие команд, позволяющих форматировать документацию – создавать заголовки, списки, иллюстрации, колонтитулы и т.п.
- WYSIWYG-редактирование – поддержка редактирования документа в таком виде, как его будет видеть читатель, т.е. без необходимости манипулировать тегами, атрибутами и т.п.
- Открытый формат – формат представления документации специфицирован и допускает «ручное» создание текстов и разработку сторонних редакторов.
- Инструментарий с открытым исходным кодом – имеется инструментальный пакет с открытым исходным кодом с возможностью модификации и расширения.

В сообществе технических писателей активно обсуждается применимость подходов разработки документации, поддерживающих те или иные средства повторного использования [22]. Основная проблема таких подходов состоит в том, что документация может стать однообразной, тогда как при ручном написании текстов одна и та же информация может быть задана с разной степенью детализации или с разных точек зрения. Однако повторное использование – это лишь техническое средство в руках технического писателя, а качество документации во многом зависит от квалификации ее автора. Действительно, даже без специальной поддержки повторного использования можно воспользоваться копированием и вставкой нужных фрагментов текста. С другой стороны, DocLine поддерживает различные средства настройки повторно-используемых фрагментов – это параметрическая адаптивность для настройки модулей документации и каталоги для «мелкозернистого» повторного использования. Благо-

даря этим средствам повторно-используемые фрагменты могут быть представлены в нужном виде для каждого контекста использования.

	DocLine	DocBook	DITA	TeX	Frame Maker	Подход Бассета-Ерзабека
Повторное использование	Да	Крупные фрагменты	Да	Крупные фрагменты	Да	Да
Адаптивное повторное использование	Да	Нет	Нет	Нет	Нет	Да
Визуальная схема повторного использования	Да	Нет	Нет	Нет	Нет	Нет
Разделение точек подключения и настройки общего актива	Да	Нет	Нет	Нет	Нет	Нет
Модульная документация	Да	Только средствами XML	Да	Да	Да	Да
Разделение контента и формата	Да	Да	Да	Да	Да	Нет
Единое исходное представление	Да	Да	Да	Нет	Да	Нет
Форматирование документации	Да	Да	Слабо	Да	Да	Нет
WYSIWYG-редактирование	Экспериментально	Базовые конструкции	Базовые конструкции	Ограниченно	Да	Нет
Открытый формат	Да	Да	Да	Да	Нет	Да
Инструментарий с открытым исходным кодом	Да	Да	Да	Да	Нет	Частично

Табл. 3. Сравнение DocLine с существующими подходами.

Заключение

В ходе выполнения данного диссертационного исследования достигнуты следующие результаты.

- Создан и формально специфицирован новый язык разработки документации DRL, включающий возможности визуального проектирования документации, средства для задания адаптивного повторного использования фрагментов документации в XML-формате, средства задания форматирования текста (интеграция с форматом DocBook).
- Предложены две эталонные модели процесса разработки документации СПП: проактивная («сверху-вниз») и «гибкая» («снизу-вверх»).
- Разработаны операции рефакторинга документации, позволяющие создавать и настраивать общие активы в процессе эволюции документации.
- Предложена архитектура пакета инструментальных средств для разработки документации СПП, выполнена реализация пакета на платформах Microsoft.NET (первая версия) и Java/Eclipse (вторая версия).
- Проведена апробация метода и инструментальных средств на документации реальных промышленных продуктов: .NET-версия применена для разработки документации семейства телевещательных систем (ООО «Фирма ДИП»), Eclipse-версия – для разработки документации ПО семейства телефонных станций (ЗАО «Ланит-Терком»).

В дальнейшем планируется развивать DocLine в следующих направлениях.

- Разработка специализированных шаблонов итоговых документов, например, для создания документации в соответствии с ГОСТ ЕСПД.
- Интеграция с существующими средствами разработки СПП.
- Разработка интеллектуальных операций рефакторинга, помогающих техническому писателю найти общие активы.

Список литературы

[1] Ахо А. В., Лам М. С., Сети Р., Ульман Д. Д.. Компиляторы: принципы, технологии и инструментарий, 2 издание. М.: Вильямс, 2008. 1184 с.

[2] Буч Г., Рамбо Д., Якобсон А. Язык UML. Руководство пользователя. Второе издание. М.: ДМК-пресс. 2006. 496 с.

[3] Кияев В.И., Кищенко Д.М., Окомин И.С. Опыт усовершенствования и стандартизации процесса создания ПО цифровых телефонных станций // Системное программирование. Вып. 2 / Под ред. А.Н. Терехова, Д.Ю. Булычева. СПб.: Изд-во С.-Петерб. ун-та, 2006. С. 219–239.

[4] Кнут Д.. Все про TeX. М.: Вильямс, 2003. 560 с.

[5] Кознов, Д.В. Основы визуального моделирования. М.: Изд-во Интернет-университета информационных технологий, ИНТУИТ.ру, БИНОМ, Лаборатория знаний. 2008. 248 с.

[6] Кознов Д.В., Перегудов А.Ф., Романовский К.Ю., Кашин А, Тимофеев А. Опыт использования UML при создании технической документации. // Системное программирование. Вып. 1. Сб. статей / Под ред. А.Н. Терехова, Д.Ю. Булычева. СПб.: Изд-во СПбГУ, 2005. С. 18–36.

[7] Кознов Д.В., Романовский К.Ю. Автоматизированный рефакторинг документации семейств программных продуктов // Системное программирование. Вып. 4 / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд-во С.-Петерб. ун-та, 2009. С. 128–150.

[8] Кознов Д.В., Романовский К.Ю. DocLine: метод разработки документации семейств программных продуктов // Программирование. 2008. №4. С. 1–13.

[9] Романовский К.Ю. Метод разработки документации семейств программных продуктов // Системное программирование. Вып. 2 / Под ред. А. Н. Терехова,

Д. Ю. Булычева. СПб.: Изд-во С.-Петерб. ун-та, 2007. С. 191–218.

[10] Романовский К.Ю. Разработка повторно-используемой документации семейства телефонных станций средствами технологии DocLine // Вестн. С.-Петерб. ун-та. Сер. 10: Прикладная математика, информатика, процессы управления. 2009. Вып. 2. С. 166–180.

[11] Романовский К.Ю., Кознов Д.В. Язык DRL для проектирования и разработки документации семейств программных продуктов // Вестн. С.-Петерб. ун-та. Сер. 10: Прикладная математика, информатика, процессы управления. 2007. Вып. 4. С. 110–122.

[12] Albing B. Combining Human-Authored and Machine-Generated Software Product Documentation // Proceedings of the IEEE International Professional Communication Conference, 2003. 21-24 Sept. 2003. P. 6–11.

[13] Assmann U. Automatic Roundtrip Engineering // SC 2003, Workshop on Software Composition. Warsaw, Poland, April 2003. Electronic Notes in Theoretical Computer Science (ENTCS) 82, No. 5. 2003. P. 1–9.

[14] Atkinson C., Bayer J., Bunse C., et al. Component-Based Product Line Engineering with UML. Addison-Wesley Professional, 2001. 464 p.

[15] Bassett P. Framing software reuse - lessons from real world. Prentice Hall, 1996. 365 p.

[16] Bassett P. Frame-Based Software Engineering, IEEE Software, July, 1987. P. 9–16.

[17] Batory D., Lofaso B., Smaragdakis Y. JST: Tools for Implementing Domain-Specific Languages. // Proc. 5th Int. Conf. on Software Reuse, Victoria, BC, Canada. 1988. P. 143–153.

[18] Bayer J., DeBaud J.M., Flege O., Knauber P., Laqua R., Muthig D., Schmid K. and Widen T. PuLSE: A Methodology to Develop Software Product Lines, Proc. Symposium on Software Reusability, SSR'99, Los Angeles, May 1999. P. 122–132.

[19] Calheiros F., Borba P., Soares S., Nepomuceno V., Vander Alv.: Product Line Variability Refactoring Tool. 1st Workshop on Refactoring Tools, Berlin. 2007.

[20] Ceri S., Fraternali P., Bongio A. Web Modeling Language (WebML): a modeling language for designing Web sites // Computer Networks. 2000. Vol. 33, N 1–6. P. 137–157.

[21] Chen L., Babar M. A., Ali N.. Variability Management in Software Product Lines: A Systematic Review // Proceedings of 13th International Software Product Line Conference (SPLC 2009), Aug 24-28, 2009, San Francisco, CA.

[22] Clark D. Rhetoric of Present Single-Sourcing Methodologies // Proceedings of the 20th ACM annual international conference on Computer documentation, Toronto, Ontario, Canada. 2002. P. 20–25.

[23] Clements P. Being Proactive Pays Off. // IEEE Software July/August 2002. P. 28–30.

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

[25] Critchlow M., Dodd K., Chou J., van der Hoek A. Refactoring product line architectures // IWR: Achievements, Challenges, and Effects. 2003. P. 23–26.

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

[27] Dunn M.. Single-Source Publishing with XML // IEEE IT Professional, January/February 2003. P. 51–54.

[28] Fowler M., et al. Refactoring: Improving the Design of Existing Code. Addison-Wesley. 1999.

[29] Fraley, Liz. Beyond Theory: Making Single-Sourcing Actually Work // Proceedings of the 21st Annual International Conference on Computer Documentation. New York: ACM Press, 2003. P. 52–59.

[30] Greenfield J., Short K., Cook S., Kent S. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Indianapolis, Indiana: Wiley Publishing, Inc. 2004. 696 p.

[31] Griss M., Favaro J., d' Alessandro M. Integrating Feature Modeling with RSEB. // IEEE Proceedings of Fifth International Conference on Software Reuse, 1998. P. 76–85.

[32] Haramundanis K., Rowland L.. Experience paper: a content reuse documentation design experience // Proceedings of the 25th annual ACM international conference on Design of communication, El Paso, Texas, USA. 2007. P. 229–233.

[33] Houser R. Single-sourcing Online Help and Training Manuals // Proceedings of IEEE International Professional Communication Conference. 2005. P. 404–413

[34] ITU-T Recommendation Z.100: Specification and description language (SDL). 1999. 244 p.

[35] Jaaksi A. Developing Mobile Browsers in a Product Line // IEEE Software July/August 2002. P. 73–80.

[36] Jarzabek S.; Bassett P.; Hongyu Zhang; Weishan Zhang. XVCL: XML-based variant configuration language // Proc. of 25th International Conference on Software Engineering, 3-10 May 2003. P. 810–811.

[37] 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.

[38] Kang K., Lee J., Donohoe P. Feature-Oriented Product Line Engineering // IEEE Software July/August 2002. P. 58–65.

[39] Krueger C. Eliminating the Adoption Barrier // IEEE Software July/August 2002. P. 30–31.

[40] Krueger C. New Methods in Software Product Line Practice // Communications Of The ACM. December 2006/Vol. 49, No. 12. P. 37–40.

[41] Lee K., Kang K., Lee J. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering // C. Gacek, ed., Proc. 7th Int'l Conf. Software Reuse, Springer Lecture Notes in Computer Science, vol. 2319, Heidelberg, Germany, 2002. P. 62–77.

[42] McConnell S. Rapid Development. Microsoft Press, 1996. 680 p.

[43] Northrop L. SEI's Software Product Line Tenets // IEEE Software July/August 2002. P. 32–40.

[44] Parnas D. On the Design and Development of Program Families // IEEE Transactions on Software Engineering, March 1976. P. 1–9.

[45] Rockley A., Kostur P., Manning S. Managing Enterprise Content: A Unified Content Strategy. Berkeley, CA: New Riders Press, 2002. 592 p.

[46] Romanovsky K. Generation-Based Software Product-Line Evolution: A Case Study. // Proceedings of 2nd International Workshop New Models of Business: Managerial Aspects and Enabling Technology. Edited by N. Krivulin, 2002. P. 178–186.

[47] Romanovsky K., Koznov D., Minchin L.: Refactoring the Documentation of Software Product Lines // Proceedings of 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques CEE-SET 2008, Brno (Czech Republic), October 13-15, 2008. P. 157–170.

[48] Steele K. The road to single-sourcing: a case study // Proceedings of IEEE International Professional Communication Conference, 2001. P. 141–149.

[49] Tolvanen J., Kelly S. Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences. // Proceedings of Software Product Line Conference'2005. Lecture notes in computer science vol. 3714, 2005. P. 198–209.

[50] Trujillo S., Batory D., Diaz O.: Feature Refactoring a Multi-Representation Program into a Product Line. // Proc. of the 5th Int. Conf. on Generative Programming and Component Engineering, 2006. P. 191–200.

[51] Walsh N., Muellner L. DocBook: The Definitive Guide. O'Reilly, 1999. 644 p.

[52] Yang J., Jarzabek S. Applying a Generative Technique for Enhanced Reuse on J2EE Platform, 4th Int. Conf. on Generative Programming and Component Engineering, GPCE'05, Sep 29 - Oct 1, 2005, Tallinn, Estonia. P. 237–255.

[53] Association for Computing Machinery's Special Interest Group on the Design of Communication. URL: http://www.sigdoc.org/.

[54] Day D., Priestley M., Schell, David A. Introduction to the Darwin Information Typing Architecture – Toward portable technical information. URL: http://www-106.ibm.com/developerworks/xml/library/x-dita1/.

[55] The DocBook project Site. URL: http://docbook.sourceforge.net/.

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

[57] Marques M. Single-sourcing with FrameMaker. // TECHWR-L Magazine Online. http://www.techwr-l.com/techwhirl/magazine/technical/singlesourcing.html.

[58] Northrop L., Clements P et al. A Framework for Software Product Line Practice. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2007. URL: http://www.sei.cmu.edu/productlines/.

Приложение: Синтаксис языка DRL

Абстрактный синтаксис DRL

Абстрактный синтаксис DRL задается с помощью НФБН.

<DRL-документация> ::

<Семейство продуктов>

<Документация семейства>

<Документация продукта>*

<Семейство продуктов> ::

<Название семейства>

<Набор продуктов>

<Название семейства> ::

<Плоский текст>

< Набор продуктов> ::

<Продукт семейства>*

<Продукт семейства> ::

<Название продукта>

<Идентификатор продукта>

<Название продукта> ::

<Плоский текст>

<Идентификатор продукта> ::

<Плоский текст>

<Документация семейства> ::

<Информационный продукт>+

<Информационный элемент>*

<Словарь>*

<Каталог>*

<Информационный продукт> ::

<Имя информационного продукта>

<Идентификатор информационного продукта>

{<Текст XML>*

{<Ссылка на информационный элемент>

| <Группа ссылок на информационный элемент> }*

<Ссылка на элемент словаря>*

<Ссылка на элемент каталога>*

<Условный блок>* }+

< Имя информационного продукта> ::

<Плоский текст>

< Идентификатор информационного продукта> ::

<Плоский текст>

<Информационный элемент> ::

<Имя информационного элемента>

<Идентификатор информационного элемента>

{<Текст XML>*

<Точка расширения>*

{<Ссылка на информационный элемент>

| <Группа ссылок на информационный элемент> }*

<Ссылка на элемент словаря>*

<Ссылка на элемент каталога>*

<Условный блок>* }+

< Имя информационного элемента> ::

<Плоский текст>

< Идентификатор информационного элемента> ::

<Плоский текст>

<Точка расширения> ::

<Идентификатор точки расширения>

[<Описание точки расширения>]

<Текст XML>

<Идентификатор точки расширения> ::

<Плоский текст>

<Описание точки расширения> ::

<Плоский текст>

<Ссылка на информационный элемент> ::

[<Идентификатор ссылки на информационный элемент>]

<Идентификатор информационного элемента>

[<Необязательный>]

< Идентификатор ссылки на информационный элемент> ::

<Плоский текст>

<Группа ссылок на информационный элемент> ::

<Идентификатор группы>

<Модификатор группы>

<Ссылка на информационный элемент>

<Ссылка на информационный элемент>+

< Идентификатор группы> ::

<Плоский текст>

<Модификатор группы> ::

<ИЛИ> | <Исключающее ИЛИ>

<Документация продукта> ::

<Ссылка на продукт семейства>

<Специализированный информационный продукт>+

<Словарь>*

<Каталог>*

< Специализированный информационный продукт> ::

<Идентификатор специализированного информационного продукта>

<Ссылка на информационный продукт>

<Присваивание значения псевдопеременной**>***

<Адаптер>*

<**Идентификатор специализированного информационного продукта**> ::

<Плоский текст>

<Ссылка на информационный продукт> ::

<Идентификатор информационного продукта>

<Ссылка на продукт семейства> ::

<Идентификатор продукта семейства>

<Адаптер> ::

<Идентификатор ссылки на информационный элемент>

<Команда адаптации информационного элемента>*

<Команда адаптации информационного элемента> ::

<Идентификатор точки расширения>

{{<Вставить-до> |

<Вставить-после> |

<Вставить-вместо>}

<Текст XML>}

<Словарь> ::

<Идентификатор словаря>

<Элемент словаря>+

<Идентификатор словаря> ::

<Плоский текст>

<Элемент словаря> ::

<Идентификатор элемента словаря>

<Значение элемента словаря>

<Идентификатор элемента словаря> ::

<Плоский текст**>**

<Значение элемента словаря> ::

<Текст XML>

<Ссылка на элемент словаря> ::

<Идентификатор словаря>

<Идентификатор элемента словаря>

<Каталог> ::

<Идентификатор каталога>

<Элемент каталога>+

<Шаблон элемента каталога>+

<Идентификатор каталога> ::

<Плоский текст>

<Элемент каталога> ::

<Идентификатор элемента каталога>

<Атрибут элемента каталога>+

<Атрибут элемента каталога> ::

<Идентификатор атрибута элемента каталога>

<Значение атрибута элемента каталога>

<Идентификатор атрибута элемента каталога> ::

<Плоский текст>

<Значение атрибута элемента каталога> ::

<Текст XML>

<Шаблон элемента каталога> ::

<Идентификатор шаблона элемента каталога>

<Идентификатор каталога>

{<Текст XML>*

<Ссылка на атрибут элемента каталога>*}+

<Ссылка на атрибут элемента каталога> ::

<Плоский текст>

<Ссылка на элемент каталога> ::

<Идентификатор шаблона элемента каталога>

<Идентификатор элемента каталога>

<Условный блок> ::

<Условие>

<Текст XML>

<Условие> ::

<Плоский текст>

<Присваивание значения псевдопеременной> ::

<Имя переменной>

<Плоский текст>

<Имя переменной> ::

<Плоский текст>

<Использование значения псевдопеременной> ::

<Имя переменной>

<Текст XML>

Конкретный синтаксис DRL/GR

Конкретный синтаксис DRL/GR определяется в терминах расширенной НФБН.

<Диаграмма семейства> ::=

<Прямоугольник> содержит

{<Заголовок диаграммы семейства>

<Горизонтальный разделитель>

<Секция продуктов семейства>

<Вертикальный пунктирный разделитель>

<Секция информационных продуктов>}

<Заголовок диаграммы семейства> ::=

Семейство продуктов <кавычка><Название семейства><кавычка>

<**Кавычка**> ::=

“

<**Название семейства**> ::=

<Плоский текст>

<**Горизонтальный разделитель**> ::=

_____

<**Секция продуктов семейства**> ::=

<Изображение продукта семейства>*

<**Вертикальный пунктирный разделитель**> ::=

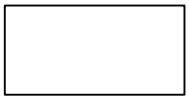
<**Секция информационных продуктов**> ::=

<Изображение информационного продукта>

<Изображение продукта семейства> ::=

<Символ продукта семейства> содержит <Имя продукта>

<Символ продукта семейства> ::=



<Имя продукта> ::=

<Плоский текст>

<Изображение информационного продукта> ::=

<Символ информационного продукта> содержит <Имя информационного продукта>

<Символ информационного продукта> ::=

<Имя информационного продукта> ::=

<Плоский текст>

<**Диаграмма вариативности**> ::=

<Корень диаграммы вариативности> соединяется с

<Включение информационного элемента или группы>*

<**Корень диаграммы вариативности**> ::=

<Изображение информационного продукта> |

<Изображение информационного элемента>

<**Включение информационного элемента или группы**> ::=

<Включение информационного элемента> |

<Включение группы>

<**Включение информационного элемента**> ::=

(<Изображение включения> соединяется с <Узел диаграммы вариативности>) |

<**Включение группы**> ::=

(<Изображение обязательного включения > соединяется с <Группа включений>)

<**Узел диаграммы вариативности**> ::=

<Изображение информационного элемента> соединяется с

< Включение информационного элемента или группы>*

<**Изображение информационного элемента**> ::=

<Символ информационного элемента> содержит <Имя информационного элемента >

<Символ информационного элемента> ::=

<Имя информационного элемента> ::=

<Плоский текст>

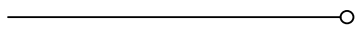
<**Изображение включения**> ::=

<Изображение обязательного включения> |

<Изображение необязательного включения> |

<**Изображение обязательного включения**> ::=

<**Изображение необязательного включения**> ::=|



<**Группа включений**> ::=

<Символ группы> соединяется с

<Включение информационного элемента>+

<**Символ группы**> ::=

<Символ OR-группы> |

<Символ XOR-группы>

<**Символ XOR-группы**> ::=

<**Символ OR-группы**> ::=

<**Диаграмма продукта**> ::=

<Корень диаграммы продукта> соединяется с

<Обязательное включение информационного элемента>*

<**Обязательное включение информационного элемента**> ::=

<Изображение обязательного включения> соединяется с

<Узел диаграммы продукта>

<**Узел диаграммы продукта**> ::=

<Символ информационного элемента> соединяется с

<Обязательное включение информационного элемента>*

Конкретный синтаксис DRL/PR и служебный синтаксис

Конкретный синтаксис DRL/PR, который также является и служебным синтаксисом для всего языка DRL, задается с помощью схемы Relax NG.

```
<grammar>
 <start>
  <choice>
   <ref name="ProductLine"/>
   <ref name="DocumentationCore"/>
   <ref name="ProductDocumentation"/>
  </choice>
 </start>
 <define name="ProductLine">
  <element name="ProductLine">
   <attribute name="name"/>
   <zeroOrMore>
    <element name="Product">
     <attribute name="id"/>
     <attribute name="name"/>
    </element>
   </zeroOrMore>
  </element>
 </define>
 <define name="ProductDocumentation">
  <element name="ProductDocumentation">
   <attribute name="productid"/>
   <zeroOrMore>
    <choice>
     <ref name="FinalInfProduct"/>
     <ref name="Dictionary"/>
     <ref name="Directory"/>
     <ref name="DirTemplate"/>
    </choice>
   </zeroOrMore>
  </element>
 </define>
 <define name="DocumentationCore">
  <element name="DocumentationCore">
   <zeroOrMore>
    <choice>
     <ref name="InfProduct"/>
     <ref name="InfElement"/>
     <ref name="Dictionary"/>
     <ref name="Directory"/>
     <ref name="DirTemplate"/>
    </choice>
   </zeroOrMore>
  </element>
 </define>
 <define name="InfProduct">
  <element name="InfProduct">
   <attribute name="id"/>
   <attribute name="name"/>
   <ref name="DocbookOrCommonDrl"/>
  </element>
 </define>
 <define name="InfElement">
  <element name="InfElement">
   <attribute name="id"/>
   <attribute name="name"/>
   <ref name="DocbookOrCommonDrl"/>
  </element>
 </define>
 <define name="Dictionary">
  <element name="Dictionary">
   <attribute name="id"/>
   <attribute name="name"/>
   <zeroOrMore>
    <element name="Entry">
     <attribute name="id"/>
     <text/>
    </element>
   </zeroOrMore>
  </element>
 </define>
 <define name="FinalInfProduct">
  <element name="FinalInfProduct">
   <attribute name="id"/>
   <attribute name="infproductid"/>
   <zeroOrMore>
    <choice>
     <element name="SetValue">
      <attribute name="id"/>
      <attribute name="value"/>
      <empty/>
     </element>
     <element name="Adapter">
      <attribute name="infelemrefid"/>
      <zeroOrMore>
       <choice>
```

```
            <element name="Replace-Nest">
              <attribute name="nestid"/>
              <ref name="JustDocbook"/>
            </element>
            <element name="Insert-Before">
              <attribute name="nestid"/>
              <ref name="JustDocbook"/>
            </element>
            <element name="Insert-After">
              <attribute name="nestid"/>
              <ref name="JustDocbook"/>
            </element>
          </choice>
        </zeroOrMore>
      </element>
    </choice>
  </zeroOrMore>
 </element>
</define>
<define name="Directory">
 <element name="Directory">
  <attribute name="id"/>
  <attribute name="name"/>
  <zeroOrMore>
   <element name="Entry">
    <attribute name="id"/>
    <zeroOrMore>
     <element name="Attr">
      <attribute name="id"/>
      <text/>
     </element>
    </zeroOrMore>
   </element>
  </zeroOrMore>
 </element>
</define>
<define name="DirTemplate">
 <element name="DirTemplate">
  <attribute name="id"/>
  <attribute name="directoryid"/>
  <ref name="DirTemplateContents"/>
 </element>
</define>
<define name="DirTemplateContents">
 <zeroOrMore>
  <choice>
   <text/>
   <element name="AttrRef">
    <attribute name="attrid"/>
   </element>
   <element>
    <nsName ns="docbook.org/ns/docbook"/>
    <zeroOrMore>
     <attribute>
      <anyName/>
     </attribute>
    </zeroOrMore>
    <ref name="DirTemplateContents"/>
   </element>
  </choice>
 </zeroOrMore>
</define>
<define name="DocbookOrCommonDrl">
 <zeroOrMore>
  <choice>
   <text/>
   <element name="DirRef">
    <attribute name="templateid"/>
    <attribute name="entryid"/>
   </element>
   <element name="InfElemRef">
    <attribute name="id"/>
    <attribute name="infelemid"/>
    <optional>
     <attribute name="groupid"/>
    </optional>
    <optional>
     <attribute name="optional">
      <choice>
       <value>true</value>
       <value>false</value>
      </choice>
     </attribute>
    </optional>
   </element>
   <element name="InfElemRefGroup">
    <attribute name="id"/>
    <attribute name="name"/>
    <attribute name="modifier">
     <choice>
      <value>XOR</value>
      <value>OR</value>
     </choice>
    </attribute>
   </element>
   <element name="DictRef">
    <attribute name="dictid"/>
    <attribute name="entryid"/>
   </element>
   <element name="Conditional">
    <attribute name="condition"/>
    <ref name="DocbookOrCommonDrl"/>
   </element>
   <element name="Nest">
    <attribute name="id"/>
    <optional>
```

```
          <attribute name="descr"/>
        </optional>
        <ref name="JustDocbook"/>
      </element>
      <element>
        <nsName ns="docbook.org/ns/docbook"/>
        <zeroOrMore>
          <attribute>
            <anyName/>
          </attribute>
        </zeroOrMore>
        <ref name="DocbookOrCommonDrl"/>
      </element>
    </choice>
  </zeroOrMore>
 </define>
 <define name="JustDocbook">
  <zeroOrMore>
    <choice>
      <text/>
      <element>
        <nsName ns="docbook.org/ns/docbook"/>
        <zeroOrMore>
          <attribute>
            <anyName/>
          </attribute>
        </zeroOrMore>
        <ref name="JustDocbook"/>
      </element>
    </choice>
  </zeroOrMore>
 </define>
</grammar>
```