Автоматизированный рефакторинг документации семейств программных продуктов*

Д. В. Кознов
dkoznov@yandex.ru

К. Ю. Романовский
kromanovsky@yandex.ru

Одной из наиболее продуктивных техник в области эволюции семейств программных продуктов (далее — СПП) является рефакторинг, т. е. извлечение и уточнение повторно используемых активов, а также улучшение архитектуры всего СПП таким образом, чтобы поведение существующих продуктов оставалось неизменным. В данной статье идея рефакторинга применяется к документации СПП, поскольку документы так же, как и исходные тексты программ, эволюционируют по мере развития СПП. Сегодня распространены различные подходы к разработке документации, основанные на XML, и XML-спецификации оказываются хорошим объектом для формальных преобразований. В основе исследования лежит технология DocLine, предназначенная для организации адаптивного повторного использования документации. В работе предлагается модель процесса разработки документации, основанная на рефакторинге, набор операций рефакторинга, а также описывается реализация предложенных идей в инструментальном пакете DocLine. Наконец, в статье представлены результаты пилотного проекта, в котором предложенный подход применялся к документации семейства телекоммуникационных систем.

* Исследование выполнено при частичной финансовой поддержке РФФИ (грант 08-01-00716).

Введение

Техническая документация является важной составляющей коммерческого программного обеспечения. Разработка и сопровождение документации — требований, спецификаций, руководств пользователя, учебных материалов и т. п. — являются трудоемкой частью процесса разработки ПО. Как и само ПО, документация может иметь сложную структуру, часто меняться и иметь много версий. Более того, документация зачастую разрабатывается на нескольких языках (например, на английском и русском) и представляется в различных конечных форматах, таких как HTML, PDF, HTML Help и др. Разработка семейств программных продуктов является популярным подходом к промышленной разработке ПО. Еще в 1976 году Парнас заметил, что вместо самостоятельных систем эффективно создавать целые семейства продуктов [25]. Семейство программных продуктов является группой программных систем, обладающих набором общих свойств и разрабатываемых совместно на основе общих, повторно используемых активов [15, 24]. С одной стороны, этот подход позволяет эффективно организовать повторное использование в рамках всего проекта, но, с другой стороны, является сложно организованным процессом. Ведь пока разрабатываются новые продукты, существующие часто требуют сопровождения и расширения. Для решения задач эволюции СПП активно используют идеи рефакторинга архитектуры и общих активов [9, 12, 18, 23, 30].

Документация СПП оказывается еще более сложной, чем документация одиночных приложений, так как содержит множество повторов и изменяется во времени — уточняется, дополняется. В работах [2, 3, 5, 6, 7] мы представили DocLine — технологию разработки документации СПП, основанную на XML и поддерживающую плановое адаптивное повторное использование фрагментов документации. DocLine реализует трехуровневое представление документации — в виде диаграмм (структура повторного использования), XML-спецификаций и целевых документов в форматах PDF, HTML и др. DocLine также предлагает рекомендации по организации процесса разработки документации и поддерживается инструментальным пакетом на базе платформы Eclipse.

Цель данной работы — распространить идею рефакторинга СПП на разработку документации. В самом деле, подходы к разработке документации, основанные на XML (такие, как DocBook [31],

DITA [19] и др.), становятся все более и более популярными, превращая документацию в формальную спецификацию. При добавлении в СПП новых продуктов целевая документация для этих продуктов (в виде PDF-файлов, HTML/WinHelp и проч.) часто должна быть отделена от документации к другим продуктам СПП — заказчики хотят получить только ту документацию, которая описывает заказанное ими ПО (им нет дела, что у нас СПП!). С другой стороны, на уровне внутреннего представления документация СПП может составлять единое целое — ведь она состоит из многочисленных общих разделов, описывающих функциональность, одинаковую во всех продуктах. Таким образом, в этой ситуации, при добавлении в СПП нового продукта и, соответственно, в документацию СПП — нового пакета документов, возникает задача по извлечению и подготовке новых повторно используемых фрагментов текста для включения их в новый пакет документов. При этом целевое представление существующих документов не должно меняться (хотя XML-спецификация меняться может). Мы считаем, что для такого преобразования документации применим термин рефакторинг.

В данной статье мы предлагаем модель процесса разработки документации на основе рефакторинга, описываем набор операций рефакторинга и представляем их реализацию в инструментальном пакете DocLine. В статье также обсуждается пилотный проект, в котором предложенный подход применен к документации семейства телекоммуникационных систем. Основные идеи рефакторнга были кратко сформулированы нами в работе [3]. В настоящей статье мы предлагаем более детальное описание этих идей, более подробные примеры, а также модифицированную модель процесса разработки документации на основе рефакторинга.

1. Обзор

1.1. Разработка СПП

Эволюция СПП — важнейший аспект их существования. Изначально появились методы разработки СПП «сверху вниз», подобные DSSA [29], предполагавшие начинать разработку СПП с подробного анализа предметной области, после этого выполнить идентификацию возможностей повторного использования, а затем реализовать общие активы и лишь после этого перейти к разработ-

ке продуктов семейства. Таким образом, организуется крупномасштабное повторное использование. Но применение этого подхода требует существенных инвестиций, предполагает стабильность и объемность рынка, а также значительную протяженность во времени существования данного СПП. Но взамен подход обеспечивает существенное экономическое (и временное) преимущество, которое наступает после того, как будет разработано несколько первых продуктов [14]. В идеальном случае разработка нового продукта в рамках СПП может выглядеть как выбор и конфигурирование готовых общих активов. Подобные методы, однако, несут в себе серьезные риски, поскольку в случае, если количество разрабатываемых продуктов останется небольшим, инвестиции могут не окупиться.

«Легковесные» методы были предложены для минимизации этих рисков. Они предлагают разрабатывать СПП «снизу-вверх», т. е. начинать с разработки первого продукта и переходить к разработке семейства только когда станут понятны перспективы создания новых продуктов [22]. Для облегчения этого перехода из «доноров» (самостоятельных продуктов) извлекаются общие активы, которые и используются в дальнейшей разработке. Этот подход значительно снижает стоимость и время выхода первого продукта, но приносит меньше выгоды с ростом числа разрабатываемых продуктов.

Сторонники обоих подходов сходятся в том, что регулярно в ходе эволюции СПП возникает потребность в изменении архитектуры СПП и общих активов семейства (сценарии эволюции общих активов СПП можно найти в работе [4]), причем эти изменения не должны затрагивать функциональность выпущенных продуктов. Более того, в «легковесных» подходах такие изменения составляют основу процесса разработки.

1.2. Рефакторинг

Рефакторинг — это процесс изменения программной системы, выполняемый таким образом, что внешнее поведение системы не изменяется, но при этом улучшается ее внутренняя структура [20]. Рефакторинг приобрел популярность в «гибких» подходах разработки ПО, поскольку он является альтернативой дорогостоящему предварительному проектированию, обеспечивая постоянные улучшения архитектуры системы с сохранением ее поведения.

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 (например, извлечение новых классов, перемещение полей вверх/вниз по иерархии). Еще есть так называемый «большой рефакторинг», например перепроектирование и переписывание приложения в связи с его переводом с COBOL на Java.

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 (в том числе 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). Такие средства, как интегрированная среда разработки IntelliJ IDEA [32] для языка Java, поддерживают «ручной» рефакторинг, предоставляя средства для автоматизированного регрессионного тестирования выполненных изменений, в том числе включая модульное тестирование.

1.3. Рефакторинг в разработке СПП

Рефакторингу СПП посвящены многие исследования. Так в работе [12] предлагается метод рефакторинга модели возможностей (Feature Model) для расширения множества возможных конфигураций семейства. В работе [30] 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 [18] 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 увеличения их конфигурируемости.

1.4. XML-подходы разработки документации

Многие подходы разработки документации используют принцип разделения содержания и форматирования, который означает, что смысловые конструкции документации, такие как главы, секции, таблицы и т. п., отделены от их форматирования (например, информация о шрифте и размере текста, заголовков). Эта идея появилась задолго до XML и была реализована, например, в издательской системе Дональда Кнута TeX [28]. Современные XML-подходы разработки документации используют эту идею, а также поддерживают концепцию единого исходного представления (single sourcing) — разработку нескольких документов на основе единого исходного текста [26]. Практическая применимость использования подобных технологий широко обсуждается сообществом технических писателей [13]. Преимущества использования XML в средних и крупных компаниях оправдывают стоимость их внедрения (см., например [8]). На практике такие XML-технологии, как DocBook [31] и DITA [19], активно внедряются в промышленности многими крупными компаниями и проектами, включая IBM, PostgreSQL, Adobe, Sun, Cray Inc [16]. Документация Unix-систем и различных оконных оболочек (например, GNOME, KDE) также создается с помощью подобных технологий [17].

2. Подход DocLine

2.1. Основные идеи

Подход DocLine [3] предназначен для разработки и сопровождения технической документации СПП. Одна из отличительных характеристик такой документации — наличие большого количества возможностей повторного использования как в рамках документации одного продукта, так и между похожими документами для различных продуктов. DocLine обеспечивает планирование повторного использования на основе визуального моделирования, позволяя создавать, просматривать и модифицировать схему повторно используемых фрагментов (общих активов) и их связей. DocLine поддерживает *адаптивное* повторное использование, т. е. возможность специфической настройки многократно используемых фрагментов текста в каждом конкретном контексте, где они используются, основываясь на идеях Пола Бассета и Станислова Ерзабека [10, 21].

DocLine определяет XML-язык DRL (Documentation Reuse Language) для проектирования и реализации документации [7]. Также DocLine предлагает модель процесса разработки документации и пакет инструментальных средств, интегрированных в среду Eclipse IDE [3]. Далее мы сконцентрируемся на основных свойствах DocLine, необходимых для понимания рефакторинга документации.

Для поддержки форматирования текста в DocLine используется известный язык DocBook [31], являющийся стандартом де-факто в разработке документации для Linux/Unix-систем. Фактически DRL расширяет DocBook механизмом адаптивного повторного использования. DRL-спецификации сначала транслируются инструментами DocLine в «чистый» DocBook, а затем используются утилиты DocBook для получения целевых документов в различных форматах (PDF, HTML и т. п.).

2.2. Обзор языка DRL

Для описания шаблонов документов, таких как руководство пользователя или справочная система продукта, в DRL используется конструкция «информационный продукт». Конкретные документы для конкретных продуктов семейства (специализированные информационные продукты) получаются за счет настройки (специализации) информационных продуктов. Строительные блоки (повторно используемые фрагменты текста), из которых строится документация (информационные продукты), представлены в DRL конструкцией «информационный элемент». Каждый информационный элемент может включаться в любой другой информационный элемент. Эти включения могут быть необязательными или взаимозависимыми.

DRL предоставляет два механизма адаптивного повторного использования: настраиваемые информационные элементы и каталоги.

Настраиваемые информационные элементы. Рассмотрим документацию семейства телефонных аппаратов с функцией определения номера вызывающего абонента. Описание процедуры обработки входящего звонка может содержать следующий текст:

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

Но различные типы телефонов могут иметь разные средства индикации вызывающего абонента. Например, телефон для слабовидящих может вместо визуального отображения проговаривать номер абонента. Тогда пример (1) выглядел бы так (здесь и далее в примерах фрагментов текста полужирным шрифтам выделяется изменяющаяся часть):

```
После получения входящего вызова телефон запрашивает в
сети информацию о вызывающем абоненте и затем прогова-
ривает номер абонента.
```
(2)

Для обеспечения преобразования фрагмента (1) к виду (2) с использованием техники адаптивного повторного использования должен быть создан следующий информационный элемент:

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

В этом информационном элементе определена точка расширения `DisplayOptions` (тег `<nest/>`). Когда информационный элемент включается в определенный контекст, каждая точка расширения может быть удалена или дополнена специфичным текстом. Если не задано никаких расширений, информационный элемент из примера (3) будет преобразован в текст примера (1). Следующие настройки преобразуют его в (2):

```
<infelemref infelemid=CallerIdent>
<replace-nest nestid=DisplayOptions> проговаривает номер
абонента
</replace-nest>
</infelemref>
```
(4)

В этом примере определена конструкция `<infelemref/>`, которая содержит ссылку по имени на исходный информационный элемент и команду замены текста точки расширения новым текстом (`<replace-nest/>`). Аналогичным образом можно вставить текст до или после точки расширения, для этого предназначены конструкции `<insert-before/>` и `<insert-after/>` соответственно, например:

```
<infelemref infelemid=CallerIdent>
<insert-after nestid=DisplayOptions>и имя абонента,
если оно задано в записной книжке телефона
</insert-after>
</infelemref>
```
(5)

Результат (5) выглядит так:

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**. (6)

Каталоги. 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 проч.) и, соответственно, в пользовательской документации. 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 набор элементов, заданных 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 Print и Save в каталоге:

```
<directory name="GUICommands">
 <entry name="Print">
   <attr name="name">Печать</attr>
   <attr name="icon">Print.bmp</attr>
   <attr name="descr">Печать документа</attr>
 </entry>
<entry name="Save">
   <attr name="name">Сохранение</attr>
   <attr name="icon">Save.bmp</attr>
   <attr name="descr">Сохранение документа</attr>
 </entry>
</directory>
```

В дополнение к набору элементов каталог содержит ряд *шаб-*

лонов отображения, определяющих то, как компоновать атрибуты для разных представлений каталога в тексте документации. Предложенный 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.

3. Рефакторинг документации

3.1. Модель процесса разработки документации

Как мы обсуждали ранее, имеются две основные модели эволюции СПП: «сверху-вниз» и «снизу-вверх». DocLine поддерживает обе модели, но фокусируется в первую очередь на модели «снизу-вверх», поскольку она чаще используется на практике. При этом документация для следующего продукта создается на основе существующей документации для уже созданных продуктов, поскольку функциональность продуктов существенно пересекается и, значит, соответствующая документация содержит много повторов.

На рис. 1 показан первый шаг процесса разработки документации «снизу-вверх». Сначала разрабатывается документация для первого продукта без учета повторного использования. Это соответствует «гибкому» подходу к разработке ПО, когда вначале делается минимум инфраструктурной работы и основное внимание уделяется тому, что нужно заказчику. 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

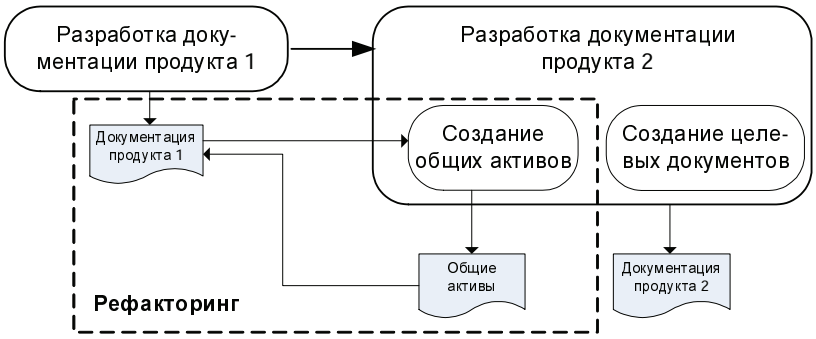

Рис. 1. Модель процесса «снизу-вверх»: начало.

формата позволяют разрабатывать обычную «монолитную» документацию, не расходуя ресурсы на организацию повторного использования). В противном случае потребуется осуществлять преобразование документации к формату DocBook, которое не всегда может быть выполнено автоматически. Далее, во время (или после) разработки второго продукта данного СПП создается документация и для этого продукта. С этой целью анализируются общие места и различия между двумя продуктами, выделяются общие активы в документации первого продукта (фрагменты текста, которые можно вставить либо без изменения, либо с незначительными изменениями), которые оформляются средствами DocLine. Эти общие активы интегрируются в документацию первого продукта вместо прежних фрагментов, а также на их основе создается документация для второго продукта. Рефакторингом здесь является выделение общих активов из документации первого продукта и соответствующее изменение этой документации так, чтобы ее фактическое содержание осталось неизменным. При смысловой неизменности этой документации ее внутренняя XML-структура претерпевает значительные изменения. На рис. 2 показан дальнейший типовой шаг разработки документации для $N+1$ продукта СПП, где $N > 1$. Эта ситуация отличается от первого шага (см. рис. 1) тем, что общие активы уже существуют. Новый продукт может потребовать изменения и доработки повторно используемых компонент СПП (о сценариях эволюции повторно используемого программного кода см., например, [27]), соответственно, потребуется изменить и повторно

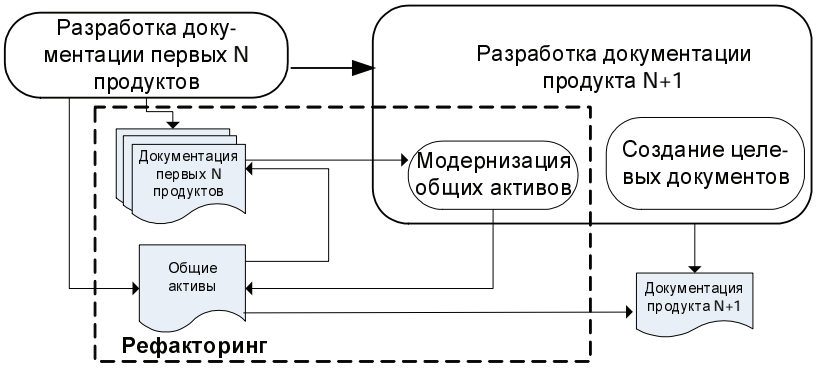

Рис. 2. Модель процесса «снизу-вверх»: типовой шаг.

используемые активы существующей документации. Эти изменившиеся активы будут использованы при конструировании документации для $N+1$ продукта. Рефакторинг обеспечивает здесь модернизацию существующей документации для первых N продуктов с сохранением их фактического содержания в неизменном виде при переработке их XML-представлений.

Отметим, что при разработке документации для следующего продукта существующие и вновь созданные фрагменты текста, которые не используются повторно (или используются идентично), должны быть преобразованы в единый крупный информационный элемент. DRL-спецификация декомпозируется глубже, если информационный элемент различается в контексте различных продуктов семейства или же если есть иные существенные причины декомпозиции, например параллельное редактирование документации несколькими техническими писателями. DRL не предназначен для обычной декомпозиции текста на главы, разделы, подразделы и т. д. — с его помощью задается именно структура повторно используемых фрагментов текста.

3.2. Операции рефакторинга

Опишем, как идеи рефакторинга реализуются в преобразования XML-спецификаций. Первая группа операций предназначена для извлечения новых общих активов.

Импорт документации из формата DocBook. Монолитный документ в формате DocBook импортируется в DocLine. Для этого создается минимально необходимый набор DRL-конструкций, а именно:

- информационный элемент, содержащий полный текст исходного DocBook-документа;
- информационный продукт, содержащий ссылку на вышеуказанный информационный элемент и определяющий единственный возможный тип документа;
- 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).

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 документация в формате DocBook импортируется в DocLine и технический писатель намерен извлечь из нее несколько новых информационных продуктов.

Расщепление информационного элемента. Существующий информационный элемент разделяется на два новых. Все ссылки и адаптеры соответствующим образом модифицируются.

Извлечение информационного элемента. Фрагмент DRL-спецификации выделяется в самостоятельный информационный элемент. Затем в том месте, где он находился, вставляется ссылка на вновь созданный информационный элемент. Если извлекаемый фрагмент уже содержал точки расширения, то для всех финальных информационных продуктов создаются новые адаптеры для сохранения всех манипуляций с этими точками расширения.

Следующие операции предназначены для «настройки» общих активов, т. е. для расширения их вариативности.

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 конечного продукта СПП.

Выделение в Insert After/Insert Before. Начальный/хвостовой фрагмент текста точки расширения извлекается и помещается во все существующие ссылки на исходный информационный элемент в виде конструкции `Insert-After`/`Insert Before`. Если указанный фрагмент не находится внутри какой-либо точки расширения, а содержится внутри какого-либо информационного элемента, то создается новая пустая точка расширения перед/после этого фрагмента.

Объявление ссылки на информационный элемент необязательной, и наоборот. Указанная обязательная ссылка на информационный элемент (*infelemref*) объявляется необязательной. Это означает, что в новых финальных информационных продуктах можно 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 операции в адаптерах могут быть удалены.

Преобразование в условный блок. Выделенный фрагмент текста помечается как условный блок (*conditional*) с логическим условием, заданным техническим писателем. Во всех существующих финальных информационных продуктах, содержащих указанный фрагмент, условие устанавливается истинным, чтобы гарантировать неизменность существующих конечных документов.

Изменение правил включения ссылок. Речь идет об изменении правила включения по умолчанию необязательных ссылок на информационный элемент. В зависимости от особенностей проекта необязательные ссылки могут по умолчанию включаться во все контексты использования (или исключаться из них), 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 значений по умолчанию все адаптеры должны быть обновлены, чтобы гарантировать неизменность существующих конечных документов.

Следующие операции предназначены для облегчения использования конструкций, поддерживающих «мелкозернистое» повторное использование, — словарей и каталогов.

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 пользователь имеет возможность отказаться от замены тех или иных вхождений.

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.

Копирование/перемещение элемента. Речь идет о копировании/перемещении элемента словаря/каталога 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.

Переименование. Эта операция обеспечивает 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. После переименования все ссылки на переименованные элементы автоматически обновляются.

3.3. Пример

Цель этого примера — показать, что операции рефакторинга документации требуют нелокальных модификаций исходного текста для того, чтобы сохранить конечные документы неизменными, и следовательно, для их выполнения необходимы специальные средства автоматизации. Ниже приводится фрагмент исходного текста

DRL-документации (указанные конструкции могут располагаться в различных файлах):

```
<infproduct id=PhoneManual>
   <infelemref id=InOutRef infelemid=InOut/>
</infproduct>

<infelement id=InOut> <section><title>Исходящие звонки</title>
   Ваш телефон поддерживает следующие способы набора
   номера абонента:
   <nest id=DialOptions>набор на цифровой клавиатуре
   </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=OfficePhone infproductid=PhoneManual>
   <adapter infelemrefid=InOutRef>
      <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 `PhoneManual`, 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 `OfficePhone`, который модифицирует информационный продукт `PhoneManual`, 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, границы которого выделены полужирным шрифтом в примере выше. Если выполнить операцию извлечения информационного элемента для этого фрагмента, то 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 абонента: <nest id=DialOptions>набор на
     цифровой клавиатуре</nest>.
  </section>
  <infelemref id=IncomingCallsRef 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 *OfficePhone* (измененный текст выделен полужирным шрифтом):

```
<finalinfproduct name=OfficePhone infproductid=PhoneManual>
   <adapter infelemrefid=InOutRef>
      <insert-after nestid=DialOptions>
         или выбор из адресной книги
</insert-after>
   </adapter>
   <adapter infelemrefid=IncomingCallsRef>
```

```
      <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 выделеной точкой расширения были перенесены из существующего адаптера в новый, описывающий специализацию вновь созданного информационного элемента.

3.4. Инструментальная поддержка

Операции рефакторинга, определенные выше, реализованы в инструментальном пакете DocLine [3]. Пакет инструментов DocLine представляет собой набор модулей расширения инструментальной среды Eclipse IDE (рис. 3). Поддержка рефакторинга документации реализована в текстовом редакторе DRL/PR в виде библиотеки, реализующей базовые функции, такие как разбор DRL-файлов (с поддержкой многофайловой документации), генерация DRL-файлов и поиск DRL-конструкций в дереве разбора. На основе представленной библиотеки реализованы собственно сами операции рефакторинга.

4. Апробация

Процесс рефакторинга документации СПП, описанный в данной статье, был использован при разработке документации семейства телекоммуникационных систем [1]. Это семейство включает автоматические телефонные станции (АТС) различного назначения: офисные, междугородные, транзитные и т. д. Для нашего эксперимента мы выбрали два продукта семейства — стандартную городскую АТС (далее — ГАТС) и специализированную АТС (далее — САТС). Мы решили перенести в DocLine руководства пользователя для рабочего места оператора этих продуктов.

В ходе анализа мы обнаружили, что исторически САТС появилась как вариант ГАТС с урезанной функциональностью. Однако в процессе развития САТС некоторые исходные функции были модифицированы, а также появились новые, так что соответствующим образом менялось и руководство пользователя.

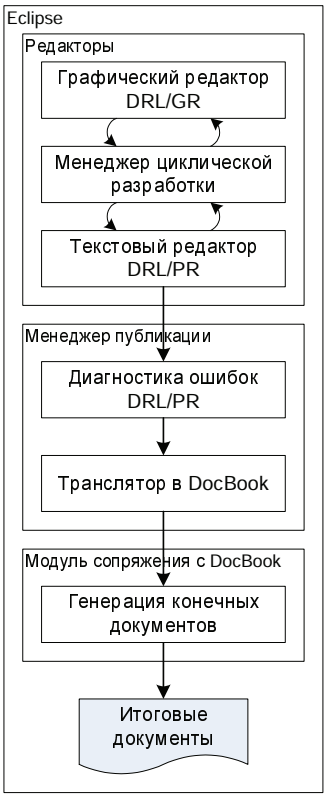

Рис. 3. Инструментальный пакет DocLine: схема пакета.

В первую очередь мы преобразовали рассматриваемую документацию в формат DocBook (в нашем эксперименте это было сделано «вручную», однако в некоторых случаях этот процесс может быть автоматизирован). Затем мы начали выделять в тексте конструкции DRL. Мы обнаружили ряд часто используемых терминов и словосочетаний и создали словари и каталоги для того, чтобы обеспечить их унифицированное использование во всей документации. Далее мы нашли несколько фрагментов текста, которые были похожи в обоих документах, и создали на их основе информационные элементы. Затем эти информационные элементы были пре-

образованы для использования в обоих документах (были созданы необходимые точки расширения и адаптеры).

В нашем эксперименте мы воспользовались следующими операциями: импорт документации DocBook, извлечение информационного элемента, преобразование фрагмента в точку расширения, извлечение текста в конструкции Insert After/Insert Before, объявление ссылки на информационный элемент необязательной, извлечение в словарь, извлечение в каталог. Эти операции позволили нам построить удобную внутреннюю структуру документации и обеспечили повторное использование фрагментов текста двумя пакетами документации с сохранением неизменным внешнего вида конечных документов.

Мы обнаружили, что поиск кандидатов на извлечение в общие активы — достаточно сложная задача. Рассмотрим, например, операцию извлечения информационного элемента. Как найти то, что необходимо извлекать? Вероятные кандидаты — это похожие, но не идентичные фрагменты текста, а их поиск в паре документов суммарным объемом более 300 страниц оказался очень сложной задачей. Отсюда следует, что есть потребность в специализированном инструментарии, который технические писатели могли бы использовать вместе с инструментом рефакторинга для облегчения поиска возможных общих активов.

Заключение

Подход рефакторинга документации СПП, предложенный в данной статье, предназначен для преобразования монолитной документации одного или нескольких продуктов в повторно используемую документацию СПП с явно выделенными обоими активами. Также этот подход может использоваться для создания документации новых продуктов семейства на основе существующих. Основная причина, по которой это следует делать, — необходимость внесения большого количества точечных изменений в документацию.

В дальнейших исследованиях мы планируем обеспечить автоматизированный поиск фрагментов текста для рефакторинга: фрагментов для выделения в информационные элементы, слов для внесения в словари и стандартных словосочетаний для внесения в каталоги. Многообещающим выглядит подход выявления похожих фрагментов исходного кода (source code clones detection) [11], так

как его можно расширить для поиска «полиморфных» похожих фрагментов.

Также представляют интерес управление изменениями документации на основе управления вариативностью СПП. Ведь фактически описания продуктов семейства отличаются там, где они описывают различную функциональность.

Еще одна область для дальнейших исследований — это определение прагматики рефакторинга, т. е. рекомендаций по проведению рефакторинга документации. Для рефакторинга программного кода имеются различные соглашения о кодировании, правила построения объектно-ориентированных иерархий, готовые рекомендации по применению тех или иных операций рефакторинга и т. п. [20]. Мы планируем разработать набор сходных рекомендаций для нашего случая. Вот интересный частный вопрос прагматики — сохранение баланса между универсальностью документации и сложностью ее структуры. Наконец, мы планируем выполнить более масштабные промышленные эксперименты.

Список литературы

[1] *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 СПбГУ, 2006. С. 219–239.

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

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

[4] *Попова Т., Кознов Д., Тиунова А., Романовский К.* Эволюция общих активов в задачах реинжиниринга // Системное программирование / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд-во СПбГУ, 2004. С. 184–199.

[5] *Романовский К. Ю.* Метод разработки документации семейств программных продуктов // Системное программирование. Вып. 2 / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд-во СПбГУ, 2007. С. 191–218.

[6] *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.

[7] *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.

[8] *Albing B.* Combining Human-Authored and Machine-Generated Software Product Documentation // Professional Communication Conference. IEEE Press. 2003. P. 6–11.

[9] *Alves V., Gheyi R., Massoni, T., Kulesza* et al. Refactoring Product Lines // Proceedings of the 5th International Conference on Generative Programming and Component Engineering. Portland, Oregon, USA, 2006. P. 201–210.

[10] *Bassett P.* Framing Software Reuse – Lessons From the Real World. Yourdon Press, Prentice Hall, 1997.

[11] *Burd E., Bailey J.* Evaluating Clone Detection Tools for Use During Preventative Maintenance // Proc. 2nd IEEE International Workshop on Source Code Analysis and Manipulation (SCAM) Montreal, Canada, Oct. 2002. P. 36–43.

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

[13] *Clark D.* Rhetoric of Present Single-Sourcing Methodologies // SIGDOC'02, Toronto, Ontario, Canada, 2002.

[14] *Clements P.* Being Proactive Pays Off // IEEE Software, July/August, 2002. P. 28–31.

[15] *Clements P., Northrop L.* Software Product Lines: Practices and Patterns. Addison-Wesley. Boston, 2002.

[16] Companies Using DITA: http://dita.xml.org/deployments

[17] Companies Using DocBook: http://wiki.docbook.org/topic/WhoUsesDocBook

[18] *Critchlow M., Dodd K., Chou J., van der Hoek A.* Refactoring Product Line Architectures // IWR: Achievements, Challenges, and Effects. 2003. P. 23–26.

[19] *Day D., Priestley M., Schell, David A.* Introduction to the Darwin Information Typing Architecture — Toward Portable Technical Information. http://www-106.ibm.com/developerworks/xml/library/x-dita1/

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

[21] *Jarzabek S., Bassett P., Hongyu Zhang, Weishan Zhang.* XVCL: XML-Based Variant Configuration Language // Proc. of 25th Intern. Conf. on Software Engineering. 2003. P. 810–811.

[22] *Krueger C.* Eliminating the Adoption Barrier // IEEE Software. July/August 2002. P. 29–31.

[23] *Liu J., Batory D., Lengauer C.* Feature Oriented Refactoring of Legacy Applications // Proceedings of the 28th International Conference on Software Engineering. ACM Press, 2006. P. 112–121.

[24] *Northrop L., Clements P.* A Framework for Software Product Line Practice, Version 5.0. 2008. http://www.sei.cmu.edu/productlines/framework.html

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

[26] *Rockley A., Kostur P., Manning S.* Managing Enterprise Content: A Unified Content Strategy. New Riders, 2002.

[27] *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» / Ed. by N. Krivulin. St. Petersburg, 2002. P. 178–186.

[28] TeX user group: http://www.tug.org

[29] *Tracz W.* Collected Overview Reports from the DSSA Project, Technical Report, Loral Federal Systems, Owego, 1994.

[30] *Trujillo S., Batory D., Daz O.* Feature Refactoring a Multi-Representation Program into a Product Line // Proc. of the 5th Int. Conf. on Generative Programming and Component Engineering. 2006.

[31] *Walsh N., Muellner L.* DocBook: The Definitive Guide. O'Reilly, 1999.

[32] http://www.jetbrains.com/idea/index.html