САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Соколов Николай Евгеньевич

Интеграция технологии DocLine в Adobe FrameMaker

Выпускная квалификационная работа

Допустить к защите.
Зав. кафедрой
д. ф.-м. н., профессор …… А. Н. Терехов

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

Рецензент …… К.Ю. Романовский
к. ф.-м. н,
старший преподаватель

Санкт-Петербург
2010

St. Petersburg State University

Faculty of Mathematics and Mechanics

Chair of Software Engineering

Integration DocLine technogogy in Adobe FrameMaker

Graduate paper by

Nikolay Evgenyevich Sokolov

"Admitted to proof"
(Head of the chair , Dr. of Phys. and Math. Sci.)

/Signature/_____/ A. N. Terekhov

Scientific advisor
(Candidate of Phys. and Math. Sci., associate professor)

/Signature/_____/ D. V. Koznov

Reviewer
(Candidate of Phys. and Math. Sci., assistant professor)

/Signature/_____/ K. Y. Romanovsky

St. Petersburg, 2010

Содержание

Введение

Во всех областях в современном мире применяется различное программное обеспечение – от огромных информационных систем до небольших скриптов, автоматизирующих какие-либо действия. И если приложение не пишется одним разработчиком для собственного использования, оно требует наличия документации. Для небольшого продукта документацию можно представить в виде простого текстового файла, прилагающегося вместе с программой. Но для более крупных проектов это будет уже неудобно. Для создания документации можно использовать распространенные текстовые редакторы, такие как Microsoft Word. Последний предоставляет довольно большие возможности для редактирования оформления текста и при этом не требует почти никакого специального обучения – Microsoft Word умеет пользоваться практически любой человек. Но, начиная с какого-то объема документации, Microsoft Word уже не может полноценно использоваться для создания технической документации. И проблема не столько в недостатке какой-либо функциональности, сколько в его непредсказуемой и нестабильной работе при редактировании больших документов.

Профессиональные технические писатели всё чаще используют для создания специализированные языки разметки, такие как TeX [1] или DocBook [14, 17], дающие широкие возможности для редактирования структуры и оформления документации. Самым популярным языком разметки на сегодняшний день является XML (eXtensible Markup Language) [11], позволяющий на своей базе создать язык, удовлетворяющий конкретным нуждам. Примером такого языка может являться XHTML (EXtensible HyperText Markup Language), использующийся для разметки Web-страниц. Также на XML основан широко используемый в среде технических писателей язык DocBook. Он предоставляет широкий набор тэгов для описания структуры и назначения определенных элементов документации, но при этом они не описывает, как в точности будет выглядеть документ – право выбора оставляется за людьми, создающими продукты для работы с документацией в формате DocBook. Технология DocBook активно поддерживается мировым Open-source сообществом, на нем написана документация довольно широко распространенной операционной системы Linux [19].

Однако, технологии, основанные на XML, отходят от концепции WYSIWYG (What You See Is What You Get), позволяющей пользователю в реальном времени следить за внешним видом документации. Компанией Adobe был предложен продукт Adobe FrameMaker [18], одним из основных преимуществ которого является возможность следить в реальном времени за изменениями в представлении документации, как в WYSIWYG-редакторах, но при этом редактировать, основанную на XML структуру документации и “привязанные” к её узлам свойства. Правда всё это накладывает и определенные ограничения – в отличие от DocBook, где описывается только структура документа без оформления, для документа Adobe FrameMaker надо либо задавать отдельные стилевые описания для узлов, либо заранее задать “привязку” представления документов определенного типа к их структуре. В силу своей схожести с языком SGML [9], являющимся предшественником и обобщением языка XML, Adobe FrameMaker позволяет не только экспортировать свои документы в распространенные форматы представления документации, такие как HTML или PDF, но и в XML-файлы, описывающие структуру документа, и обратно импортировать XML-документы.

Если программный продукт имеет лишь одну версию и его дальнейшая разработка не предполагается, можно не задумываться о переиспользовании документации. Но с выходом новой версии продукта требуется и выпуск новой документации, прилагающейся к ней. Самый очевидное решение – переиспользование уже существующей документации к первой версии продукта. Эта мысль кажется хорошей, но лишь до тех пор, пока не будет предпринята попытка воплотить её в жизнь. Стоимость поддержки такой документации может оказаться очень высокой. Почему такое может произойти? Ответ прост – документация к первой версии продукта изначально не планировалась как переиспользуемая. Отсюда можно сделать вывод, что требуется сразу создавать документацию в виде, удобном для переиспользования.

Одной из технологий для создания документации, разработанной с акцентом на возможности повторного использования документации, является технология DocLine [5], разработанная на кафедре системного программирования математико-механического факультета СПбГУ. Она в отличие от других технологий помимо “крупноблочного” переиспользования документации и её составных частей предоставляет “мелкозернистое” переиспользование – переиспользование небольших фрагментов текста, таких как название

продукта, а также адаптивное переиспользование документации, позволяющее учитывать контекст фрагментов при их переиспользовании. Также предоставляется возможность планирования структуры документации и её переиспользования при помощи графической нотации. Существует реализация DocLine для платформы Eclipse. Это продукт имеет обширную функциональность: создание и редактирование документации в графическом и XML-представлениях, их автоматическая синхронизация, рефакторинг существующей документации, а также проверка целостности и корректности документации и её публикация. Но этот продукт может быть не удобен для эффективного применения широким сообществом технических писателей – у них есть свои средства разработки документации, и Eclipse не входит в их число. В связи с этим было целесообразно интегрировать технологию DocLine в одно из популярных средств для редактирования технической документации, как это сделано с другими распространенными XML-технологиями, например DocBook и DITA. Таким средством был выбран Adobe FrameMaker по определенному ряду причин. Во-первых, решающим фактором сыграла широкая распространенность Adobe FrameMaker среди технических писателей. Во-вторых, на выбор Adobe FrameMaker повлияла наличие него структурного режима для редактирования документации, основанного на XML. И, в-третьих, разработчиками Adobe FrameMaker предоставляется API для создания плагинов на C и C++, что дает возможности сильно расширить функциональность продукта.

Постановка задачи

Данная работа выполнялась в контексте проекта разработки плагина DocLine для продукта Adobe FrameMaker. Перед автором были поставлены следующие задачи:

- адаптация языка спецификации документации в технологии DocLine для Adobe FrameMaker
- реализация функциональности импорта/экспорта документации из DRL-файлов;
- увеличение скорости работы плагина;
- обеспечение стабильной работы плагина.

Глава 1. Обзор предметной области

Продукт Adobe FrameMaker

Первая версия FrameMaker-а была выпущена в 1986 году магистром астрофизики колумбийского университета Чарльзом Корфилдом (Charles Corfield). FrameMaker задумывался как WYSIWUG-редактор для технической документации операционной системы SUN-2, что и определило его дальнейшее развитие. Уже первая версия продукта пользовалась популярностью среди технических писателей, и в последующих версиях FrameMaker был портирован на другие операционные системы фирмой, основанной Майклом Корфилдом. Но, начиная с четвертой версии, продукт стал отходить от первоначальной концепции редактора технической документации, что привело компанию Майкла Корфилда на грань разорения. Это позволило компания Adobe купить FrameMaker. Начиная с пятой версии, теперь уже Adobe FrameMaker вернулся на первоначальный путь WYSIWYG-редактора технической документации.

На данный момент выпущено девять версий FrameMaker. Adobe FrameMaker поддерживает все наиболее распространенные операционные системы: Windows, MacOS and Linux. Помимо встроенного бинарного формата документации, Adobe FrameMaker поддерживает работу с популярными основанными на XML форматами технической документации, такими как DITA и DocBook, реализованными в виде плагинов, поставляющихся вместе с Adobe FrameMaker начиная с версии 7.1.

Adobe FrameMaker имеет два типа представления и, соответственно, способа редактирования документации: обычный и структурный.

В обычном режиме техническому писателю предоставляется практически такой же функционал, как и в других WYSIWYG-редакторах.

Документы Adobe FrameMaker представляются в виде бинарных файлов с закрытой спецификацией. Также документацию можно экспортировать в популярные форматы представления документации, такие как PDF и HTML.

Структурный режим

Поскольку способы представления технической документации основанные на XML с момента своего появления имеют большую популярность, начиная с версии 5.0, FrameMaker

поддерживает структурный режим работы с документацией, позволяющий представить документацию в сходном с XML формате.

Структурный режим работы с документацией Adobe FrameMaker предназначен для работы с документами, имеющими помимо обычного текстового наполнения древовидную структуру. Эта структура может быть задана автоматически, если документ был импортирован из уже существующего XML-документа, или быть задана заранее перед написанием документации в специальном файле.

В структурном режиме работы с документацией Adobe FrameMaker к функциональности обычного WYSIWYG-редактора добавляются механизмы для работы со структурой документации и блоками текста, связанными с ними.

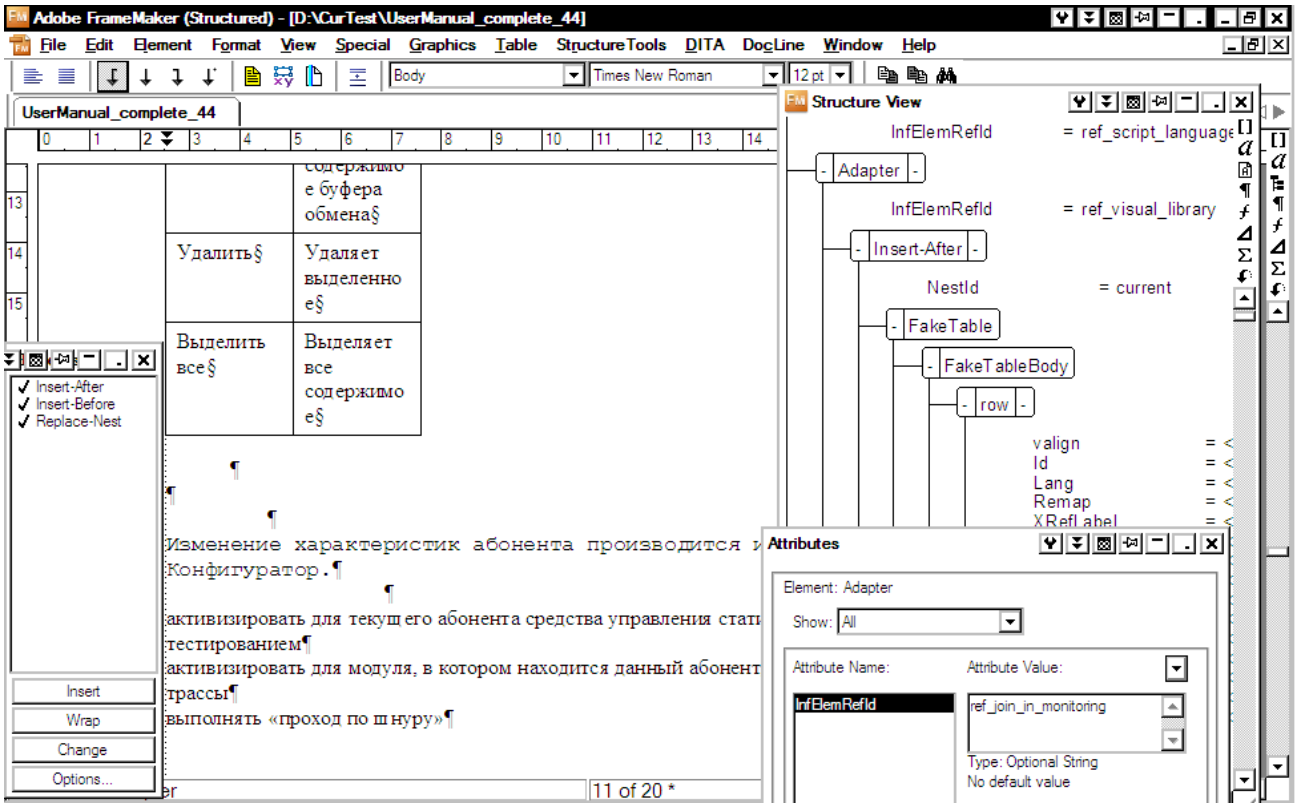

Рис. 1. Интерфейс структурного режима Adobe FrameMaker.

Но такой способ представления документации имеет свои ограничения – документ может иметь структуру, только если он либо был импортирован из XML-файла, который уже имеет структуру, либо принадлежит определенному классу документов, описанному пользователем и которому задано описание возможной структуры его экземпляров. Такой класс документов называется структурным приложением Adobe FrameMaker.

Для того, чтобы новый или уже существующий документ стал структурным, надо импортировать в него описание его структуры – EDD (Element Definition Document). После этого документ станет принадлежать определенному классу документов, имеющих общую структуру. Также новый документ можно сделать изначально структурным, если при его открытии выбрать шаблон документа с уже импортированным EDD, содержащим структуру. Такие шаблоны обычно входят в состав структурных приложений.

Редактирование структуры производится при помощи панелей Structure View и Elements (см. Рис. 1).

Книга

Книга – специальный тип документа Adobe FrameMaker, появившийся в нем с самых первых версий. Он представляет собой документ (структурный или нет), содержащий в листьях ссылки на другие документы Adobe FrameMaker – компоненты книги. Кроме этого книга не может содержать никакой текстовой или графической информации. Книги не могут быть вложены.

Структурные операции над книгой аналогичны структурным операциям над простыми документами Adobe FrameMaker.

Технология DocBook

Одним из распространенных форматов написания документации является DocBook. Основанный на XML, он предоставляет широкий набор тэгов, описывающих структуру или назначение определенных элементов документации, но при этом они не описывают, как будет выглядеть документ – право выбора оставляется за людьми, создающими продукты для работы с документацией в формате DocBook. Это касается и многих других форматов представления документации.

В технологии DocBook нет встроенной поддержки переиспользования документации, но, тем не менее, существуют искусственные методы.

Реализация в Adobe FrameMaker

Поддержка формата DocBook в Adobe FrameMaker реализована в виде структурного приложения XDocBook, соответствующего XML-представлению формата DocBook, структурного приложения DocBook, соответствующего SGML-представлению и плагина,

модифицирующего стандартные функции экспортирования и импортирования XML и SGML-документов.

Технология DITA

DITA – технология для написания технической документации на основе XML, разработанная в компании IBM. Как и DocBook она основана на XML.

Основополагающими для технологии DITA понятиями являются следующие понятия.

- Topic – элемент описывающий какую-то определенную тему. Может иметь один из следующих типов:
    - concept – описание какого-либо понятия;
    - task – описание какой-либо операции;
    - reference – описание справочной информации.
- Map – элемент, связывающий набор топиков и по сути формирующий конечную документацию. Представляет собой коллекцию ссылок на топики.

DITA изначально разрабатывалась с учетом возможности переиспользования документации. Помимо переиспользования с использованием общих файлов, реализованного при помощи карты, в DITA существует способ переиспользования при помощи специального элемента conref, являющегося ссылкой на другие элементы документации.

Реализация в Adobe FrameMaker

Технология DITA в Adobe FrameMaker реализована в виде плагина, который, начиная с версии 8.0, входит в стандартную поставку Adobe FrameMaker. Он предоставляет возможность создания документации и её редактирования. Также он модифицирует стандартные функции для импортирования и экспортирования документации формата DITA.

В отличие от плагина DocBook, основным представлением документации которого является XML-файлы, плагин DITA хранит всю информацию в виде документов Adobe FrameMaker. Также в плагине DITA отказались от использования книг Adobe FrameMaker. DITA Map представляет собой обычный структурный документ Adobe FrameMaker с ссылками на другие документы проекта. Публикация DITA документации также реализуется не с помощью FDK, а встроенными средствами Adobe FrameMaker.

Технология DocLine

Во многих методах переиспользования документации применяется только переиспользование текстовых фрагментов без адаптации их к конкретному контексту, что сильно сужает их область применения, например, из-за существования в языке таких явлений, как склонения существительных или наличие разных лиц у глагола. Игнорирование их может привести к нечитаемости текста и возможно даже изменению его содержания. Отсюда появилась идея использовать адаптивное переиспользование документации – переиспользование фрагментов документации с учетом их контекста.

Технология DocLine состоит из описания процесса разработки документации, языка DRL и набора инструментов для работы с документацией.

Язык DRL имеет две нотации: DRL/GR, описывающую общее представление документации и "крупноблочное" переиспользование, и DRL/PR, использующего DocBook для написания текста и предоставляющего возможности для более точной настройки переиспользования документации и "мелкозернистого" переиспользования.

Язык DRL/PR

Основные конструкции

Для определения списка продуктов семейства и состава документации типового продукта в языке DRL/PR существуют элементы *<ProductLine>* с *<Product>* для списка продуктов семейства:

<ProductLine name="Телефонный аппарат Унител"*>*

*<Product id="pay" name="*Унител-таксофон"*/>*

*<Product id="answer" name="*Унител-автоответчик"*/>*

*<Product id="aon" name="*Унител-АОН"*/>*

</ProductLine>

и *<DocumentationCore>* с *<InfProduct>* для состава документации

<DocumentationCore>

<InfProduct id="guide" name="Руководство пользователя" */>*

<InfProduct id="help" name="Справочная система" */>*

</DocumentationCore>

Информационные продукты состоят из набора повторно используемых фрагментов документации. Составляющими информационных продуктов являются информационные элементы (*<InfElement>*). Они образуют иерархическую структуру, что в сочетании с возможностью опционального включения реализует механизм структурной адаптивности – простейший вариант “крупноблочного” переиспользования документации.

В языке DRL/PR состав информационных продуктов задается элементами *<DocumentationCore>* с *<InfElement>*

<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 >

и *<InfProduct>* с *<InfElemrefGroup>* и *<InfElemRef>* для задания “горизонтальных” связей между информационными элементами

*<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>.

За разрешение всех неоднозначностей в конечной документации отвечают элементы <ProductDocumentation> и <FinalInfProduct>

<ProductDocumentation productid="pay">

<FinalInfProduct id="guide_pay" *infproductid=*"guide">

<Adapter infelemrefid="out_ref”*/>*

</FinalInfProduct>

</ProductDocumentation>.

Адаптивное “крупноблочное” повторное использование

Для организации повторного использования в реальной документации семейств программных продуктов недостаточно только структурной адаптивности, задаваемой элементами, описанными ранее. Так, например, если понадобится иметь в одной части документации или в другой её версии дополненный в определенном месте текст из первой документации, то этих средств будет недостаточно.

“Крупноблочное” переиспользование документации реализуется при помощи выделения в исходной документации определенного текста при помощи точки расширения (<nest>) и последующей замене его в конечных версиях продукта на требующийся текст или элемент языка DRL/PR. Пример такого выделения показан ниже:

*<**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**>*

Изменение точки расширения производится в момент обращения к информационному элементу (<InfElemRef>) и может быть трех типов:

- добавление текста перед точкой расширения (<Insert-Before>),
- добавление текста после точки расширения (<Insert-After>),
- замена точки расширения (<Replace-Nest>).

Пример такой замены:

<***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**>*

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 продуктах 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. Это достигается при помощи элемента <Adapter>, который как и ссылка на информационный элемент может включать в себя преобразования текста внутри точек включения тех же трех типов:

<***FinalInfProduct*** *id="UserManual_starblind" infproductid="UserManual">*

<***Adapter*** *infelemrefid="CallerIdentRef”>*

<***replace-nest*** *nestid=DisplayOptions>проговаривает номер абонента</**replace-nest**>*

*</**Adapter**>*

*</**FinalInfProduct**>*

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:

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

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>
```

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">Сохраннеие активного документа 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 словаря необходимо извлечь значение, так как каждый шаблом может ссылаться на единственный каталог.

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.

Инструмент для Eclipse

В рамках проекта DocLine ранее были разработаны следующие технические средства.

- Графический редактор DRL/GR, поддерживающий все три вида диаграмм: главной диаграммы, диаграммы вариативности и диаграммы продукта [4, 6, 7].
- Текстовый редактор DRL/PR, основанный на XML-редакторе Eclipse и поддерживающий редактирование документации, её импортирование и базовые операции рефакторинга [2, 3, 7, 12].
- Менеджер циклической разработки, связывающий редакторы DRL/GR и DRL/PR и обеспечивающий возможность round-trip разработки документации. Так, например, после описания переиспользования в виде диаграмм на языке DRL/GR можно уточнить документацию на языке DRL/PR в текстовом редакторе, а затем снова изменить её в графическом редакторе, не потеряв при этом никакой информации.
- Менеджер публикации, позволяющий конвертировать документацию в DocBook и проверять её корректность. Проверка корректности включает в себя проверку на соответствие языку DRL/PR, проверку ссылочной целостности и проверку документа DocBook, полученного после конвертации, средствами самого ж DocBook.
- Модуль генерации, предназначенный для экспортирования документации DocBook, полученной с помощью менеджера публикации в конечные форматы: HTML и Pdf.

Обзор средств интеграции с Adobe FrameMaker

Структурное приложение

Базовым средством для структурирования документации в Adobe FrameMaker является создание структурного приложения – набора файлов для определения представления и поведения определенного класса документов. Такой способ предоставляет возможность относительно быстро перенести структуру документации в XML-формате или создать новую структуру. При этом сразу появится возможность импортирования и экспортирования документации из XML и в XML соответственно, но как-либо модифицировать этот процесс будет невозможно. Это и является основным недостатком данного метода.

Компоненты структурного приложения задаются в специальном файле – structapps.fm, находящемся в директории со структурными приложениями.

Основным элементом структурного приложения является шаблон документа данного типа. С помощью него создаются новые документы с уже существующей структурой. Шаблон - пустой или частично заполненный документ, в который импортировано описание структуры из EDD (Element Definition Document), созданного заранее. EDD – документ, содержащий описание структуры определенного класса документов. Эту структуру можно из него импортировать в другие документы, что включит их в класс, описываемый в EDD.

Вторым важным, но не обязательным, файлом является XML DTD-файл (Data Type Definitions), описывающий, как и EDD, структуру документов, но только для XML. Этот файл, как и все последующие, нужен исключительно в случае если требуется проводить импортирование или экспортирование документации из/в XML соответственно. DTD требуется для валидации документации при импортировании и экспортировании а также для корректной обработки других файлов структурного приложения. DTD-файл можно сгенерировать из EDD-файла средствами Adobe FrameMaker.

Правила чтения/записи нужны для установления отображения между элементами структуры в Adobe FrameMaker и XML. В соответствии с этим отображением при импортировании и экспортировании производится требующаяся пользователю трансляция элементов одного типа в другие. Правила чтения/записи являются единственным способом модификации поведения Adobe FrameMaker при импортировании и экспортировании документации с использованием только структурного приложения.

Также для структурного приложения можно указать каскадную таблицу стилей (CSS) для XML-представления документации. Она не несет никакой функциональной нагрузки и влияет только на представление документации в Adobe FrameMaker. Как и DTD-файл, может быть сгенерирована из EDD.

В структурном приложении можно указать подсказки для Adobe FrameMaker (DOCTYPE), которые позволяют отфильтровать список выбора структурных приложений для конкретного документа. DOCTYPE содержит в себе имя какого либо элемента структуры, который

характерен для данного типа документов, и если Adobe FrameMaker “встречает” в документе элемент с таким именем, то он может предположить, что это документ соответствующего типа.

FDK

Если пользователю требуется большая функциональность для работы с документацией, чем предоставляемая Adobe FrameMaker и его структурными приложениями, то имеется возможность написания плагинов к Adobe FrameMaker при помощи предоставляемого разработчиками API. Плагины пишутся на C/C++ с использованием библиотек FDK (FrameMaker Developers Kit).

Далее будут рассмотрены составные части FDK.

FrameMaker Development Environment

FDE включает в себя собственную систему типов, при этом встроенные типы языка C недоступны для использования.

Также FDE включает в себя средства для работы с окружением плагина таким, как файловая система, с возможностью абстрагироваться от конкретной операционной системы, что позволяет писать действительно кроссплатформенные приложения.

FrameMaker API

FrameMaker API предназначен для обеспечения взаимодействия плагина со средой Adobe FrameMaker а также для создания графического интерфейса плагинов.

Import/Export API

Import/Export API позволяет модифицировать ход трансляции документа, представленного в виде XML-файла в документ Adobe FrameMaker и наоборот.

Глава 2. Проект DocLineFM

В рамках исследовательского проекта по разработке инструментов для работы с технологией DocLine было принято решение создать плагин для Adobe FrameMaker, поддерживающий следующие функции:

- открытие проекта с документацией;
- создание нового проекта и его составных частей;
- импортирование проекта из DRL-документации;
- экспортирование проекта в DRL-документацию;
- публикация проекта в Pdf и HTML;
- закрытие проекта.

Задача была поставлена группе человек.

Помимо описанной функциональности требовалось адаптировать язык DRL/PR для использования в Adobe FrameMaker, создать интерфейс плагина и задать оформление документации.

Представление DRL-документации в Adobe FrameMaker

При выборе способа представления документации было предложено два варианта.

Первый вариант – это представление документации в обычном для Adobe FrameMaker виде – книга с набором документов в качестве её компонентов. Такой подход, например, применяется в плагине для DocBook. Главное преимущество такого варианта – он удобен для пользователей, знакомых с Adobe FrameMaker. Но он также и имеет существенный недостаток – функция обновления книг является довольно медленной операцией, особенно на больших объемах документации, и это может снизить скорость работы плагина.

Второй вариант – представление документации по примеру DITA-плагина. В DITA выделен специальный файл – карта, являющийся обычным документом Adobe FrameMaker, но хранящим в себе ссылки на другие документы. Такой файл в каком-то роде является аналогом книги, и в DITA-плагине его применение просто обусловлено наличием такого

файла в DITA. Использование такого файла могло бы сократить время работы плагина, но при этом сильно усложнило бы его реализацию.

В результате сравнения скорости работы плагинов DITA и DocBook было выявлено, что различия в производительности на документации приблизительно равного объема составляли около пятнадцати секунд и, следовательно, выбирать структуры надо было по другим параметрам. Предоставление в виде книги с набором документов было признано более удачным, поскольку оно более удобно для использования техническими писателями, работающим с Adobe FrameMaker.

Также, в соответствии с выбранным алгоритмом, было принято решение сделать компонентами книги элементы третьего уровня в иерархии адаптированного языка DRL:

- InfElement;
- InfProduct;
- FinalInfProduct;
- Dictionary;
- Directory;
- DirTemplate.

Структурное приложение DocLine

Структурное приложение DocLine состоит из:

- шаблона документа, для создания новых документов со структурой DRL;
- шаблона книги, для создания новых книг со структурой DRL;
- DTD-файлы, для валидации документации в Adobe FrameMaker и её корректного отображения в XML;
- правила чтения/записи, для отображения документации в Adobe FrameMaker с XML-документацией.

Архитектура плагина

Плагин DocLine для Adobe FrameMaker можно разбить на 4 модуля, которым соответствуют 3 библиотеки (две C-библиотеки и одна Java-библиотека) и набор XSL-файлов. Взаимосвязь компонентов показана на Рис. 2.

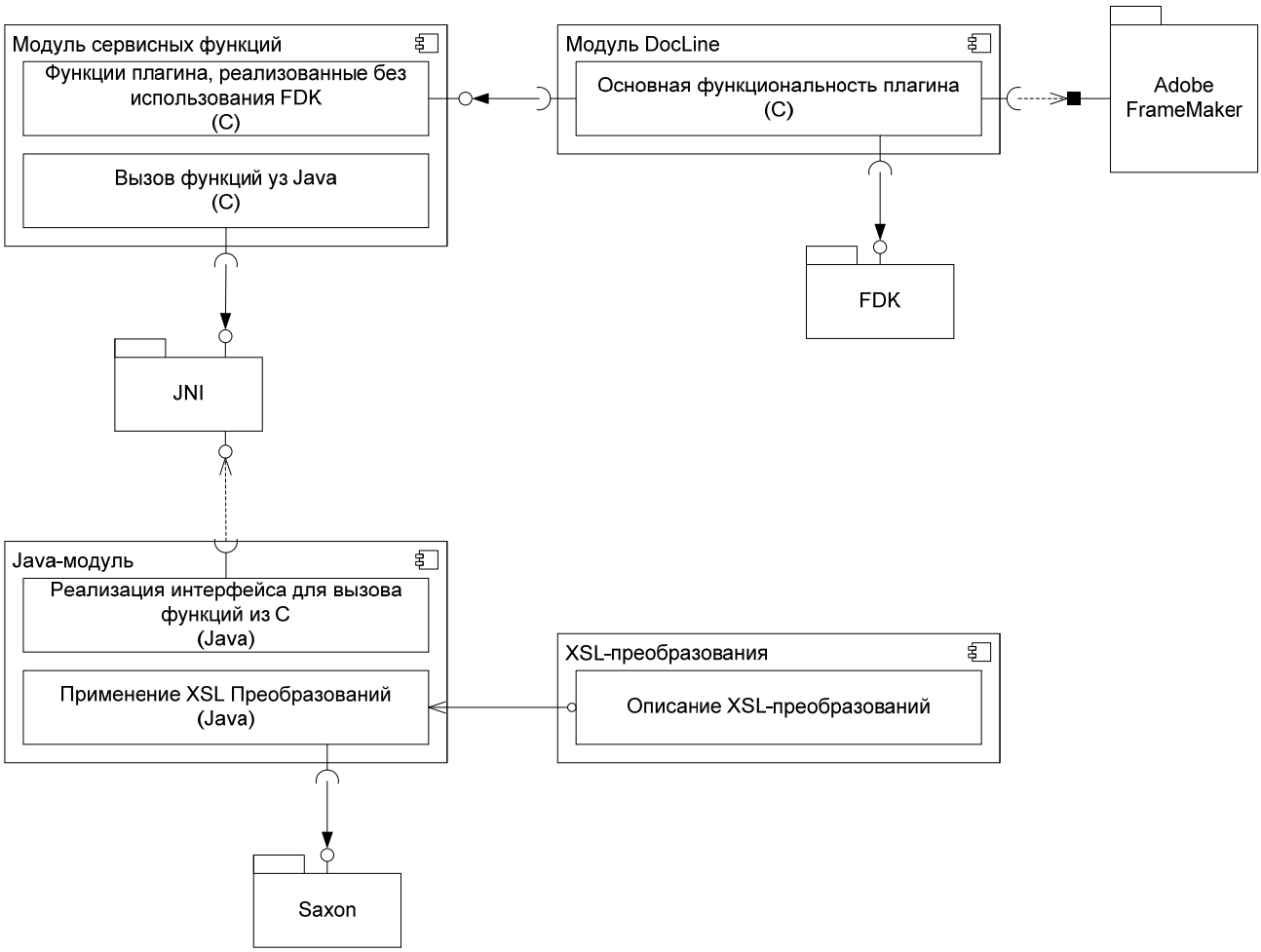

Рис. 2. Общая архитектура плагина.

Рассмотрим каждый модуль.

Модуль DocLine

Данный модуль является основным модулем плагина. Именно в нем происходит взаимодействие с Adobe FrameMaker. Именно здесь используются библиотеки FDK, что с одной стороны позволяет получить доступ к функциям Adobe FrameMaker для создания интерфейса и выполнения операций над документами, а с другой стороны не позволяет использовать типы и функции C, что в некоторых случаях требуется.

Модуль содержит следующие функции:

- инициализация интерфейса плагина;
- привязка конкретных функций к событиям элементов интерфейса;
- создание нового проекта DocLineFM и его компонент;
- импортирование/экспортирование проекта DocLineFM (за исключением вызова Java-функций);
- закрытие проекта.

Модуль сервисных функций

Основное отличие модуля сервисных функций (СФ) от первого модуля – отсутствие FDK.

Модуль включает в себя вызов Java-методов и для других функций плагина, таких как импортирование и экспортирование, и некоторые функции, реализация которых требовала функций и типов языка C. Так, например, из-за отсутствия поддержки UTF-8 в Adobe FrameMaker невозможно реализовать текстуальную коррекцию файлов документации, представленных в виде XML-файлов с кодировкой UTF-8

Модуль можно условно разбить на две части – часть, включающую в себя вспомогательные функции плагина, реализованные на “чистом” C и вызов Java-методов.

Связь Java-C реализована посредством JNI(Java Native Interface)[13]. В данной части модуля происходит только инициализация/деинициализация JVM, вызов функций JNI и обработка их результатов. Стоит отметить, что в качестве параметров для инициализации JVM передаются путь к publishutil.jar и максимальный и минимальный размер выделяемой оперативной памяти. Каждый из них накладывает свои ограничения. Файл publishutil.jar должен лежать только в директории fminit/docline корневой папки Adobe FrameMaker. А статическое задание ограничений по памяти ведет к тому, что при недостатке свободной оперативной памяти JVM не инициализируется. В то же время, если выделять небольшое максимальное количество памяти, на больших документациях не сработают Java-методы.

Java-модуль

Поскольку ранее уже были написаны функции для публикации документации на Java в виде плагина к Eclipse , было принято решение переиспользовать их. Отсюда и появился этот модуль.

Его, как и предыдущий модуль, можно разделить на две части – часть, реализующую интерфейсы JNI, и часть, выполняющую основную функциональность – публикацию документации и выполнение XSL-преобразований.

XSL-преобразования

XSL-преобразования (XSLT) можно выделить в отдельный модуль, поскольку с помощью них выполняется довольно большой объем действий, и изменение XSL-файлов никак не требует изменения остальных частей плагина. При помощи XSL-преобразований производится публикация документации, представленной в виде DRL-файлов, подготовка документации к импортированию и приведение документации к “нормальному” виду после экспортирования.

Описание функциональности плагина

В этом разделе будут рассмотрен интерфейс плагина и наиболее распространенные сценарии его использования.

Доступ к функциям плагина DocLineFM осуществляется через элементы меню DocLine. На Рис.3 показана структура этого меню.

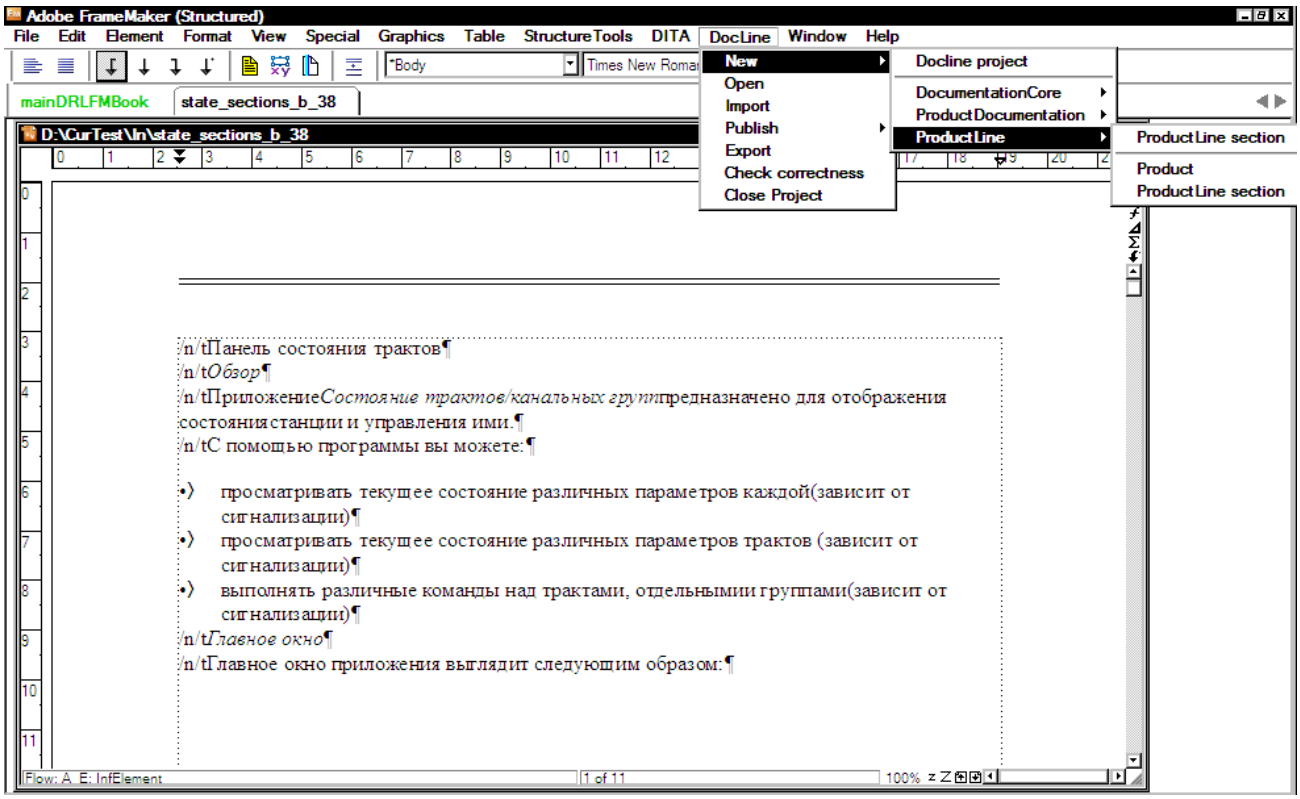

Рис. 3. Интерфейс плагина DocLine.

Подменю New объединяет в себе функции создания новых компонентов документации и создание самой документации. При создании новой документации в выбранной директории создается главная книга проекта с уже существующим элементом верхнего уровня в структуре – DocLine. Следующие три подменю меню New отвечают за создание элементов второго уровня (разделов документации) и элементов третьего уровня (компонент документации), которые могут находиться внутри них. При создании компонент документации пользователю будет предложен список подходящих разделов документации, среди которых он должен будет выбрать тот, в который надо поместить элемент. Если же в проекте нет требующихся элементов второго уровня пользователю предварительно будет показан диалог создания раздела.

Пункт меню Open отвечает за открытие документации. После выбора этого пункта пользователю будет показан диалог выбора директории с проектом.

Пункт меню Import вызывает функцию импортирования DRL-документации из директории, которую выберет пользователь. Также пользователю будет предложен выбор директории, в которую будет сохранен проект DocLineFM.

Пункт меню Export экспортирует открытый проект в документацию, представленную в виде DRL-файлов, в выбранную директорию.

Подменю Publish предназначено для публикации проекта и содержит пункты меню, соответствующие конечным форматам: Pdf и HTML.

Пункт меню Close закрывает открытый проект.

Глава 3. Архитектурные решения

Адаптация языка DRL/PR для Adobe FrameMaker

Язык DRL/PR допускает произвольный уровень вложенности конструкций DocBook и собственных конструкций друг в друга. XML позволяет как описать такое явление, например, в DTD или RELAX NG-схеме, так и реализовать его, причем описание не является обязательным. Но для структурного документа Adobe FrameMaker обязательно должно быть задано описание структуры документа, но язык его описания не позволяет реализовывать бесконечное число уровней вложенности подходящее для данного случая.

В самом простом случае бесконечное число уровней вложенности реализовать довольно просто – элементы структуры могут рекурсивно содержать друг друга. Но в случае с DRL/PR требуется, чтобы внутри любого элемента DocBook мог содержаться элемент “чистого” DRL/PR. В RELAX NG-схеме, а именно в ней изначально был описан DRL/PR, есть конструкция, описывающая любой элемент из указанного пространства имен. В EDD Adobe FrameMaker нет такой конструкции и следовательно единственным выходом является добавление в правила для каждого элемента DocBook элементов “чистого” DRL/PR и наоборот – добавление в каждый элемент DRL/PR все возможные элементы DocBook в EDD. В силу трудоемкости такой операции было принято решение применить это только к набору наиболее распространенных в плане переиспользования элементов DocBook . Такими элементами являются: Para, Preface, Chapter, ListItem, book, Section, row и entry.

Ещё одним важным изменением языка DRL/PR в его версии для Adobe FrameMaker по сравнению с исходным языком было добавление вспомогательных таблиц (элемент FakeTable) и строк таблиц (элемент FakeRow). Дело в том, что Adobe FrameMaker не позволяет находиться элементу, с типом, помеченным, как ячейка таблица или как ряд таблицы вне ряда или таблицы соответственно. Но как может находиться строка или ячейка вне таблицы? В DRL/PR такой случай возможен. Если в таблице хранятся изменяющиеся с версией данные, то нет смысла менять таблицу целиком – достаточно просто заменить нужные ячейки. Но тогда в Replace-Nest и появятся ячейки без рядов и таблиц.

Также есть ещё набор незначительных изменений в атрибутах элементов. Новые атрибуты были добавлены для хранения дополнительной информации об исходных DRL/PR-документах, таких как имя файла. Это сделано в первую очередь для того, чтобы после экспортирования была получена исходная документации с точностью до внесенных во время работы в Adobe FrameMaker изменений.

Импортирование

Импортирование документации – процесс преобразования документации, представленной в виде набора DRL-файлов в документы Adobe FrameMaker.

Для реализации импортирования было предложено несколько алгоритмов, но отличались они в основном тем, когда собственно выполнять импортирование средствами FDK и над какими файлами выполнять импортирование. Есть три варианта, какие файлы импортировать: один файл, являющийся “слитым” из исходных файлов, исходные файлы или файлы, разбитые по компонентам документации: InfElement, InfProduct, FinalInfProduct, Product, Dictionary, Directory, DirTemplate

Рассмотрим достоинства и недостатки каждого из вариантов.

При первом алгоритме все импортируемые файлы объединяются в один с корневым элементом DocLine. При этом все элементы, бывшие ранее “верхними” стали “детьми” нового верхнего элемента. Таким образом после объединения будет получен документ, являющийся структурно идентичным (с точностью до вспомогательных таблиц) документу,

который требуется получить в Adobe FrameMaker после операции импортирования. Но при таком методе возникает ряд проблем.

Первая проблема – это невозможность выполнения операции слияния при помощи XSL-преобразований. XSL-преобразования ограничены только изменением структуры документа и при обходе во время их выполнения нельзя рассмотреть несколько файлов, поскольку они не объединены общей древовидной структурой. Следовательно, объединение файлов пришлось бы делать при помощи существующего внешнего XML-парсера, что влекло бы либо усложнение Java-составляющей плагина, либо при помощи дополнительных библиотек для C, привнесло бы свои трудности в связи с использованием FDK.

Первые два уровня документации внутри Adobe FrameMaker согласно разработанному представлению документации вынесены в структуру основной книги. Следовательно, при выполнении импортирования требуется при помощи правил чтения/записи задать выделение этих элементов в книгу. Тогда импортирование будет выполняться с созданием книги. Но эта книга будет содержать в структуре только элемент первого уровня, в то время как остальные будут находиться в отдельных файлах, указанных в виде компонент книги. Такое представление документации не соответствует выбранному и поэтому потребуется переносить элементы второго уровня структуры в книгу, а соответствующие файлы компонент разбивать на отдельные файлы по элементам третьего уровня и добавлять в виде компонентов к перенесенным элементам второго уровня. При этом разбитие файлов будет сопровождаться переносом структуры, что является довольно трудоемкой операцией.

Импортирование с книгой влечет за собой и третью проблему. Дело в том, что при реализации импортирования при помощи FDK, а не Import/Export API, невозможно задать программно имя книги, являющейся результатом импортирования – пользователю будет показан диалог сохранения файла, с помощью которого он выберет имя для конечной книги. А поскольку имя главной книги фиксировано, то получается, что пользователь вводит информацию, которая далее нигде не используется. Эта причина и стала основной, повлекшей отказ от данного алгоритма.

Вариант с импортированием исходных файлов немного лучше предыдущего. Если с помощью правил чтения/записи задать создание книги из разделов документации, то после

импортирования будут получены книга для каждого исходного документа и набор документов Adobe FrameMaker, являющихся компонентами этих книг. У этих документов “корневой” элемент – компонент документации, что означает, что это именно те файлы, которые должны быть компонентами в конечной книге. Поэтому не придется копировать структуру документов. Но у этого метода сохраняется другой недостаток предыдущего – запрос у пользователя имени промежуточного файла. Причем в данном случае ситуация ещё хуже – количество запросов будет равно количеству исходных файлов, и запросы будут происходить на протяжении практически всего процесса импортирования, что будет требовать постоянного присутствия пользователя. Во время импортирования большой документации такой недостаток очень существенен.

Алгоритм с разбиением исходных файлов не обладает ни одним из приведенных выше недостатков, и, как следствие, был выбран именно он.

На Рис. 4 приведена диаграмма (в нотации BPMN [8]), описывающая алгоритм импортирования документации. Также на ней показано соответствие определенных шагов алгоритма модулям плагина.

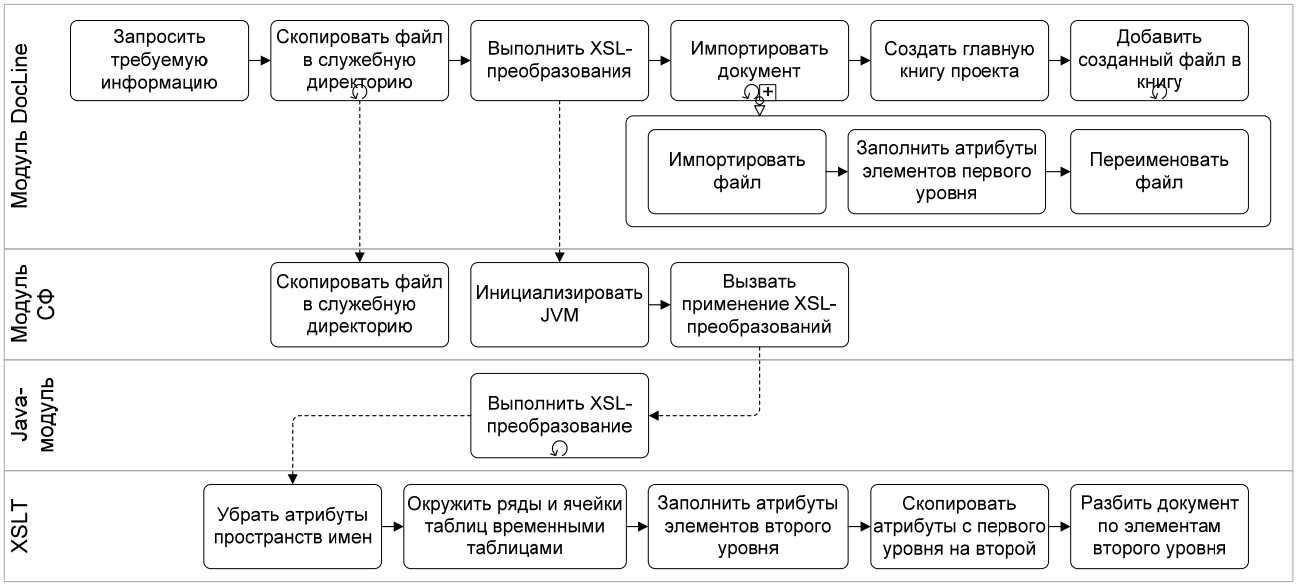

Рис. 4. Алгоритм импортирования документации.

Рассмотрим более подробно некоторые шаги алгоритма.

Перед началом импортирования у пользователя запрашиваются директория с документацией для импортирования и директория, в которой будет храниться новый проект.

Файлы документации перед выполнением каких-либо операций сначала копируются в промежуточную директорию. Но, поскольку FrameMaker некорректно обрабатывает кириллические имена файлов, функция копирования файлов была реализована с использованием функций языка C и, следовательно, перемещена в модуль сервисных функций.

Основной задачей XSL-преобразования является приведение файлов документации к виду, пригодному для импортирования и преобразованию их к модифицированному языку DRL/PR. На этом шаге заполняются значениями вспомогательные атрибуты. Атрибуту parenttype компоненты документации присваивается имя его предка. Атрибуту filename присваивается имя файла. Также копируются значения атрибутов разделов документации в атрибуты компонент. XSL преобразование также выполняет ещё два важных действия – удаление префиксов пространств имен (Adobe FrameMaker не понимает их) и разбиение файлов по компонентам документации. Также на этапе XSL преобразований строки и ячейки, лежащие вне таблиц “оборачиваются” вспомогательными таблицами (FakeTable) и строками (FakeRow).

На этапе добавления импортированных книг в основную книгу проекта происходит простой просмотр директории и все сгенерированные файлы добавляются в место, однозначно определяемое скопированным ранее на второй уровень атрибутом id и его типом, записанным в атрибуте parenttype. Если место для добавления не найдено, то в книге создается новый раздел с именем, записанным в атрибуте parenttype, и в него копируются необходимые атрибуты из верхнего элемента рассматриваемого файла.

Экспортирование

Экспортирование документации – процесс преобразования документации, представленной в виде проекта DocLineFm в набор DRL/PR-файлов.

Как в функции импортирования здесь могли возникнуть сомнения, когда требуется экспортировать книгу и требуется ли экспортировать всю книгу или предварительно разбить

книгу на части и затем экспортировать их. Рассмотрим самый простой случай – экспортирование книги целиком. На выходе будет получен XML-файл, структура которого полностью дублирует структуру исходной книги, и с точки зрения производительности намного лучше разбить этот файл при помощи XSL-преобразований на нужные файлы, чем копировать структуру из основной книги в промежуточные файлы.

На Рис. 5 представлен алгоритм экспортирования документации.

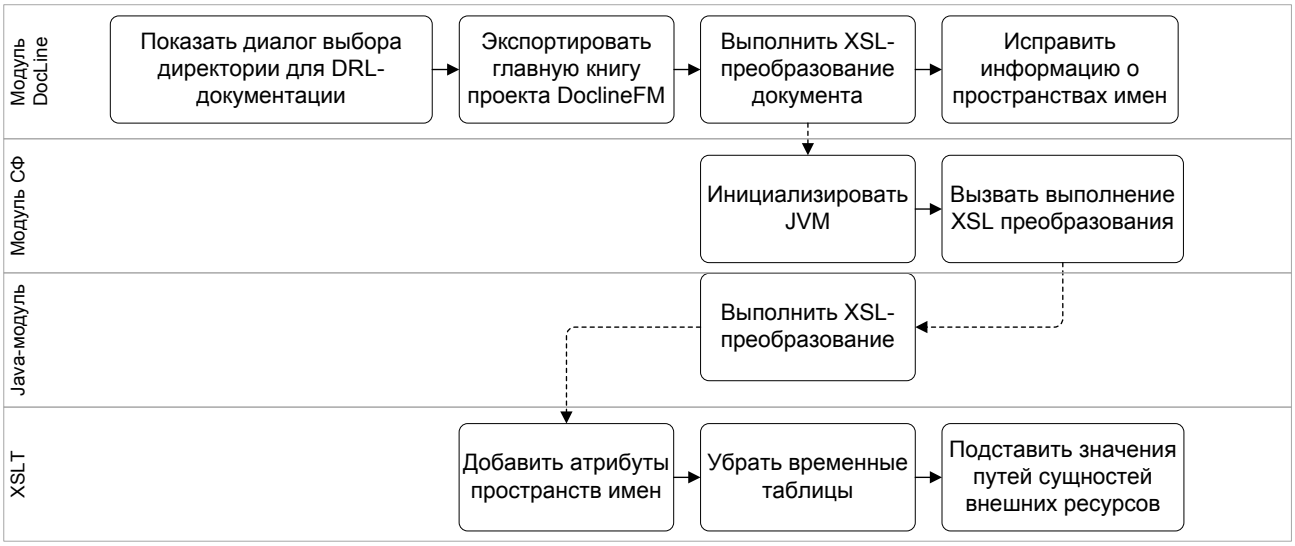

Рис. 5. Алгоритм экспортирования документации.

При экспортировании файла, представленного книгой, Adobe FrameMaker создает не один XML-файл, а по одному XML-файлу для каждой компоненты книги и один для самой книги. В файле, соответствующем книге содержится структура книги и на месте компонент – ссылки на XML-файлы компонент, представленные в виде сущностей XML (XML Entity). Такое представление документации неудобно для пользователя. Решением данной проблемы является применение, как и в импортировании к файлу книги XSL-преобразования, которое помимо этого разбивает файл на нужные, переименовывает его согласно значению атрибута filename, убирает дополнительные данные и структуры, созданные при импортировании, и добавляет пространства имен.

Глава 4. Повышение скорости работы и обеспечение стабильности

Весной 2009-го года была представлена первая версия плагина. Она разрабатывалась группой участников исследовательского проекта DocLine.

Помимо отсутствия части функциональности у первой версии были серьезные недостатки в скорости и стабильности работы.

Повышение скорости работы плагина

В первой версии наибольшие проблемы с производительностью наблюдались у функций импортирования, экспортирования и публикации. Причем, поскольку медленная работа публикации была обусловлена наличием внутри неё функции экспортирования, требовалось оптимизировать только первые две функции.

Скорость работы функций плагина в основном была повышена за счет изменения алгоритмов.

Наиболее трудоемкие операции при работе с Adobe FrameMaker – это импортирование и экспортирование документов, обновление книг и структурное копирование содержимого одного документа в другой. Следовательно, требуется минимизировать их использование.

Стоит сразу заметить, что импортирование, экспортирование и обход структуры основаны на обходе дерева, и это значит, что для применения этих операций к нескольким файлам потребуется приблизительно столько же времени, сколько и для применения к файлу, являющемуся их объединением. Также стоит отметить, что из-за обновления книги во время импортирования с созданием книги оно работает намного медленнее импортирования без такового.

В первой версии плагина использовалось:

- В импортировании:
    - Импортирование с созданием книги – число раз равное количеству импортируемых документов

    - Копирование структуры – число раз равное количеству импортируемых документов
    - Обновление книг – одно общее обновление (здесь не учитываются обновления книг при импортировании с созданием книги)
- В экспортировании:
    - Экспортирование – число раз, равное количеству элементов разделов в книге
    - Копирование структуры – число раз, равное количеству разделов в книге
    - Обновления книг нет

В конечной версии плагина:

- В импортировании:
    - Импортирование с созданием книги – один раз
    - Копирования структуры нет
    - Обновление книг – одно общее обновление
- В экспортировании:
    - Экспортирование – один раз
    - Копирования структуры нет
    - Обновления книг нет

Тестирование документации проводилось на документации объемом приблизительно 1.5 млн. символов на компьютере со следующими характеристиками: процессор Intel Core2Duo 3.00 GHz, 2Гб DDR2 оперативной памяти с частотой 800МГц.

	Первая версия	Вторая версия
Импортирование	10 минут	2 минуты
Экспортирование	8 минут	1 минута
Публикация	8.5 минут	1.5 минут

Таким образом, было достигнуто увеличение производительности более чем в пять раз.

Повышение стабильности работы плагина

Был проведен набор тестов для выявления ошибок. В ходе анализа ошибок выяснилось, что основной причиной нестабильной работы являлось использование переменных-массивов с невыделенной ранее памятью. Такое может произойти только в двух случаях: память была изначально не выделена и память была выделена, но впоследствии удалена.

Практически все ошибки были второго типа. Их также можно разделить на два типа.

Первый тип – это удаление строки, передаваемой в функцию, как фактический параметр.

Пример такой ошибки приведен ниже:

```
void function1(char *ch)
{
	free(ch);
}
 void function0(int len)
{
	char * ch = (char*)malloc(len*sizeof(char));
	function1(ch);
	free(ch);  //Здесь произойдет ошибка
}
```

В С строки, как и массивы, являются указателями на первый элемент. Поэтому, передавая в function1 строку, мы передаем её по ссылке, а не по значению. То есть внутри function1 все операции над строкой затрагивают также и строку в function0. Из-за этого, удалив выделенную ей память в function1, мы не можем обращаться к ней в function0 после этого.

Второй тип – это использование после удаления в скопированной по ссылке переменной.

Ниже приведен пример такой ошибки:

```
void function(int len)
{
	char *ch, *chCopy;
	ch = (char*)malloc(len*sizeof(char));
	chCopy = ch;
	free(chCopy);
	free(ch); //Здесь произойдет ошибка
}
```

Поскольку строки – указатели, в chCopy будет скопирована ссылка на первый элемент той же самой строки, и это, как и в прошлом случае, приведет к невозможности использовать оригинал после освобождения памяти для копии.

Также небольшой процент ошибок состоял в том, что не при выделении памяти не учитывался конечный пустой символ строки ‘\0’.

Помимо исправления ошибок с использованием невыделенной памяти, были также исправлены ошибки, вызывающие утечку памяти – отсутствие освобождения памяти после её использования. С помощью средств Visual Studio[20] и специализированных программ, таких как Deleaker[15] и DevPartner Studio[16], был проведен анализ плагина и выявлены утечки памяти, которые были исправлены.

Таким образом большинство ошибок было вызвано недостаточным опытом программирования на C и работой с динамическим выделяемой памятью в нем.

В настоящее время плагин работает стабильно.

Заключение

В ходе выполненной работы были достигнуты следующие результаты:

- изучены основы языка C, получен опыт работы с динамическим управлением памятью в C;
- изучены основы языка XSL-преобразований, а также язык описания XML-схем DTD;
- создан плагин DocLine для Adobe FrameMaker:
    - адаптирован язык DRL/PR для Adobe FrameMaker;
    - реализован импорт/экспорт документации из DRL-файлов;
    - достигнута приемлемая скорость работы при данной архитектуре;
    - обеспечена стабильная работа плагина;
- с помощью плагина создана документация к плагину.

Список литературы

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

[2] Кознов Д.В., Романовский К.Ю. Автоматизированный рефакторинг документации семейств программных продуктов. Системное программирование. Вып. 4, под ред. А.Н.Терехова и Д.Ю.Булычева. СПб.: Изд. СПбГУ, 2009.

[3] Лебедкова Т. А.: Расширение метода рефакторинга документации семейств программных продуктов, Дипломная работа, Санкт-Петербургский Государственный Университет, Математико-Механический факультет, кафедра системного программирования, 2010.

[4] Романовский К.Ю. Метод повторного использования документации семейств программных продуктов. Кандидатская диссертация, СПБГУ, 2010. 110 с.

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

[6] Семенов А. А.: Система визуального проектирования документации семейств программных продуктов, Дипломная работа, Санкт-Петербургский Государственный Университет, Математико-Механический факультет, кафедра системного программирования, 2007.

[7] Яковлев К. С.: Создание среды разработки документации для семейств программных продуктов, Дипломная работа, Санкт-Петербургский Государственный Университет, Математико-Механический факультет, кафедра системного программирования, 2007.

[8] Bruce Silver BPMN Method and Style: A levels-based methodology for BPM process modeling and improvement using BPMN 2.0, Cody-Cassidy Press, 2009, 236 p.

[9] Charles F. Goldfarb The SGML Handbook, Oxford University Press, USA, 1991, 688 p.

[10] Elliotte Rusty Harold XML Bible. IDG Books Worldwide, Inc., 1999. 996 p.

[11] Eric van der Vlist RELAX NG. O'Reilly Media. 2004. 512 p.

[12] Konstantin Romanovsky, Dmitry Koznov, Leonid Minchin Refactoring the Documentation of Software Product Lines. Central and East European Conference on Software Engineering Techniques CEE-SET 2008, Brno., 2008.

[13] Sheng Liang Java Native Interface: Programmer's Guide and Specification. Sun Microsystems, Inc. 2002. 318 p.

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

[15] Сайт продукта Deleaker URL: http://www.deleaker.com.

[16] Сайт продукта DevPartner Studio: http://microfocus.com/products/micro-focus-developer/devpartner/devpartner-studio-professional-edition.aspx.

[17] Сайт проекта DocBook. URL: http://www.docbook.org.

[18] Сайт Adobe FrameMaker. URL: http://www.adobe.com/products/framemaker.

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

[20] MSDN. Microsoft Visual Studio URL: http://msdn.microsoft.com/ru-ru/vstudio/default.aspx.

Приложение 1. Глоссарий

XML (eXtensible Markup Language) – расширяемый язык разметки – текстовый формат, предназначенный для хранения структурированных данных. Представляет собой набор вложенных друг в друга элементов, имеющих определенные свойства (атрибуты) и хранящих какие-либо данные.

XSL преобразование (XSL Transformation, XSLT) – язык преобразования XML-документов, основанный на самом XML. Позволяет преобразовывать любые XML-документы в один или несколько других XML-документов.

SGML (Standard Generalized Markup Language) – ещё один язык разметки структурированных документов. В настоящее время практически не используется.

DTD (Data Type Definitions) – язык описания структуры XML и SGML-документов. Основан на отличном от XML синтаксисе. Позволяет задать возможных “детей” текущего XML-узла на основе регулярных выражений и его возможные атрибуты. По DTD производится валидация XML и SGML-документов. В настоящее время происходит уход от использования этого языка из-за появления новых, основанных на XML, более совершенных языков описания структуры XML-документов.

Документ Adobe FrameMaker – бинарный файл, хранящий в себе информацию о содержимом документа, его представлении внутри Adobe FrameMaker. Существует два типа документов – структурные и нет, соответствующие двум типам работы Adobe FrameMaker. При структурном режиме работы помимо обычного инструментария WYSIWYG(What You See Is What You Get)-редактора пользователю предоставляются инструменты для работы со структурой структурированной документации.

Книга Adobe FrameMaker – структурный документ Adobe FrameMaker особого типа. Представляет собой, как и все другие документы Structured Adobe FrameMaker, деревянную структуру, но в листьях, у которой находятся ссылки на другие документы Adobe FrameMaker.

EDD (Element Definition Document) – документ, описывающий структуру документов Structured Adobe FrameMaker. Является аналогом DTD XML-документов для документов Adobe FrameMaker. Но в отличие от него также позволяет привязывать к отдельным компонентам структуры свойства, относящиеся к представлению текстовой информации, находящейся в узлах соответствующих EDD документов. Но EDD также и имеет ограничения,

которых нет в EDD, такие как то, что каждый узел структуры дожжен быть представлен в дереве соответствующего документа, а не может быть переменной, использующейся в других узлах (как ENTITY в DTD). Также EDD не поддерживает получение элементов из определенного пространства имен XML.

Шаблон документа – пустой или частично заполненный документ Structured Adobe FrameMaker с заданным на основе EDD определением структуры. Шаблоны документов используются для создания новых структурных документов Adobe FrameMaker.

Правила чтения/записи – набор правил, описанных на специальном языке и задающих соответствие между элементами, определенными в DTD для XML-файлов и описанными в EDD для соответствующих им документов Adobe FrameMaker.

Структурное приложение Adobe FrameMaker – набор файлов Adobe FrameMaker, задающие отображение определенного класса документов внутри Adobe FrameMaker и правила импортирования/экспортирования документов в XML. Структурное приложение Adobe FrameMaker состоит из одного базового документа – шаблона документов данного класса. Но оно может также содержать DTD файл, правила чтения записи, библиотеки для изменения поведения Adobe FrameMaker при импортировании и экспортировании документов данного класса и CSS файлы. Также для структурного приложения в специальном документе может быть определен ряд дополнительных свойств.

FDK (FrameMaker API) – набор написанных на языке C библиотек для создания плагинов к Adobe FrameMaker. Содержит собственный набор типов и собственный набор функций, оперирующий ими.