DocLineFM: среда разработки повторно-используемой документации семейств программных продуктов на базе пакета Adobe FrameMaker*

М. Н. Смирнов
smnsmn1979@gmail.com

Н. Е. Соколов
jentalmeister@gmail.com

К. Ю. Романовский
kromanovsky@yandex.ru

В данной работе представлена пилотная реализация технологии DocLine для среды разработки документации Adobe FrameMaker. Задачей данного эксперимента было использование Adobe FrameMaker как среды разработки документации типа WYSIWYG (What You See is What You Get) и привнесение туда свойства DocLine, которые относятся к адаптивному повторному использованию фрагментов документации. В работе обсуждается полученный результат, а также возникшие проблемы. Работа представляет интерес как пример серьезной программной кастомизации готового коробочного продукта.

Ключевые слова: FrameMaker, DocLine, повторное использование, техническая документация, Eclipse.

Введение

В ходе разработки промышленного программного обеспечения (ПО) требуется создавать значительные пакеты документов, опи-

* Исследование выполнено при частичной финансовой поддержке Российского фонда фундаментальных исследований (грант 08-01-00716-а).

сывающих различные правила работы с ним, — это и краткое описание для быстрого знакомства, и справочник пользователя, и руководство администратора, а также другие виды документов. В случае разработки семейств программных продуктов (СПП) [11] похожие пакеты документов создаются для каждого продукта семейства. Аналогичная задача возникает не только для программных продуктов, но и для бытовой техники — как правило, производители создают семейства приборов, отличающихся отдельными характеристиками.

Традиционное решение — инструкция, охватывающая все семейство, с пометками какой раздел к каким вариантам изделия относится. Минусы такого решения очевидны — пользователю необходимо все время держать в памяти обозначение модели прибора, которое может отличаться от других вариантов лишь одной буквой (т. е. велика вероятность ошибки), кроме того, объем такой инструкции будет больше специализированной, что неблагоприятно влияет на экологию. Для домашней бытовой техники (телевизора и т. п.) цена ошибки не очень велика, а в случае сложной дорогостоящей техники, такой как автомобиль, ошибка может иметь катастрофические последствия — например, в случае ошибочного выбора типа топлива.

Идеальное решение — разработать для каждого продукта собственный пакет документов. Однако делать это «с нуля» для каждого типа продукта достаточно дорого, поэтому целесообразно использовать средства повторного использования. Поскольку продукты все же имеют различия, такое средство должно поддерживать вариативность повторного использования, т. е. позволять адаптировать общие активы к особенностям конкретного контекста (продукта или вида документа). Простейшее средство для таких целей — копирование базовой версии текста с последующими модификациями. Этот подход обеспечивает наибольшую гибкость, но осложняет совместное сопровождение таких документов — после копирования и изменения связь между похожими фрагментами теряется, т. е. внесение исправлений требует изменения целого пакета документов.

Проводя аналогию с разработкой ПО можно сказать, что текст процедуры раскрывается и модифицируется в разных контекстах, поэтому ошибка в исходном варианте потребует поиска и исправле-

ния ошибок уже во всех копиях. Для решения этой проблемы необходимо сохранять связь между оригиналом и копиями и предоставлять механизм управляемой модификации оригинала под потребности контекста.

В научном сообществе документация не так популярна, как другие области программной инженерии, однако есть известные центры, вокруг которых сосредоточены исследования, самый крупный из которых — ассоциация ACM SIGDOC (Association for Computer Machinery's Special Interest Group on the Design of Communication) [9]. В рамках этого сообщества проводятся ежегодные конференции, охватывающие широкий круг вопросов: специализированные языки и средства разработки электронной документации, качество («понятность») текстов, особенности перевода технических документов на иностранные языки, принципы разработки, сопровождения и обеспечения целостности больших пакетов документов и т. д.

В работах [1—6] предлагается метод DocLine, предназначенный для разработки повторно используемой документации СПП, основанный на XML и поддерживающий 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. Базовая версия DocLine встраивается в интегрированную среду разработки Eclipse и содержит графический редактор для создания диаграмм, текстовый редактор для создания текстов документации на языке DRL, а также систему публикации документов в различных форматах, основанную на пакете DocBook.

В крупных проектах разработки ПО бóльшая часть времени работы технического писателя уходит именно на написание текстов документации, т. е. для него основным инструментом является именно текстовый редактор. Редактор DocLine обеспечивает ряд традиционных для сред разработки ПО сервисов — подсветка синтаксиса, подсказки по завершению конструкций и их параметров, проверка корректности и отображение ошибок с привязкой к их позиции в исходном файле и т. п. Тем не менее текстовый редактор DocLine остается специализированным XML-редактором и не поддерживает режим создания документов WYSIWYG — что-

бы форматировать текст и настраивать повторное использование, приходится писать специальные конструкции, и лишь потом при публикации документа можно увидеть, как выглядит созданный текст. Такая схема работы привычна для программистов, но для технических писателей может быть не очень удобна.

Таким образом, целесообразно поддержать режим редактирования WYSIWYG для DocLine. Реализовать подобную функциональность можно как собственными силами, так и с помощью интеграции в существующий инструмент, поддерживающий требуемые функции. Один из существенных плюсов интеграции — предоставление техническому писателю привычной для него среды (в отличие от Eclipse, который привычен для программистов). Стандарт де-факто для разработки промышленной документации — редактор FrameMaker компании Adobe [7]. FrameMaker ориентирован на редактирование и верстку сложных документов большого объема в различных целевых форматах. Для большинства профессиональных технических писателей FrameMaker — такой же привычный инструмент, как Eclipse для многих Java-программистов. Так, известные технологии разработки промышленной документации DocBook [13] и DITA [12] интегрированы в Adobe FrameMaker, однако они не поддерживают адаптивное повторное использование. Интеграция DocLine в пакет FrameMaker обеспечит прежде всего поддержку адаптивного повторного использования в FrameMaker, т. е. возможности настраивать контент с учетом особенностей конкретного контекста использования.

Однако интеграция в существующий пакет — это всегда риск. С одной стороны, привлекают возможности, которые уже реализованы, с другой стороны, ограничивают существующие механизмы интеграции. Простые функции можно реализовать быстро (в сравнении с разработкой «с нуля») и малыми ресурсами, однако нетривиальные задачи могут не иметь прямого решения в рамках имеющихся механизмов. Одна из задач данной работы — изучить средства интеграции пакета Adobe FrameMaker. В работе представлен адаптированный вариант метода DocLine, интегрированный в среду создания технической документации Adobe FrameMaker.

1. Обзор технологии DocLine

1.1. Основные идеи

Подход DocLine [1] предназначен для разработки и сопровождения технической документации СПП. Одна из отличительных характеристик такой документации — наличие большого количества возможностей повторного использования фрагментов текста 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, то есть специфическую настройку общих фрагментов в каждом отдельном контексте использования, основываясь на идеях Пола Бассета и Станислова Ерзабека [14, 16]. DocLine определяет XML-язык DRL (Documentation Reuse Language) для проектирования и реализации документации [3]. Также DocLine предлагает модель процесса разработки документации и пакет инструментальных средств, интегрированных в среду Eclipse IDE [1].

Далее мы сконцентрируемся на основных свойствах DocLine, необходимых для понимания осуществления рефакторнига документации.

Для поддержки форматирования текста в DocLine используется известный язык DocBook [13], 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.).

1.2. 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 (повторноиспользуемые 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 с различными способами представления.

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 информацию о звонящем абоненте и затем отображает ее на экране. (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 шрифтом выделяется изменяющаяся часть примера):

После получения входящего вызова телефон запрашивает информацию о звонящем абоненте и затем **проговаривает номер абонента** (2)

Для обеспечения преобразования фрагмента (1) к виду (2) с использованием техники адаптивного повторного использования должен быть создан следующий информационный элемент (здесь и далее примеры приводятся в синтаксисе DRL, заголовки конструкций DRL выделяются полужирным шрифтом):

<**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**>. (3)
</**infelement**>

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**> (4)
</**infelemref**>

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**> (5)
</**infelemref**>

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 с различными способами представления. В интерфейсах программных продуктов можно обнаружить много однотипных элементов, например команды пользовательского интер-

фейса. Эти команды по-разному представлены в интерфейсе ПО (в панели инструментов, в меню, всплывающие подсказки и пр.) 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"> Эта команда предназначена для:</attr>
  </entry>
  <entry name=" Save">
    <attr name="name">Сохранение</attr>
    <attr name="icon">Save.bmp</attr>
    <attr name="descr">Эта команда предназначена для :</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.

2. Пакет Adobe FrameMaker

FrameMaker был выпущен в 1986 г. как WYSIWUG-редактор для технической документации операционной системы SUN-2. Затем продукт был перенесен на другие платформы и получил широкое распространение среди технических писателей во всем мире. Современные версии FrameMaker (сейчас он принадлежит компании Adobe) наряду со стандартным WYSIWYG-режимом редактирования поддерживают также и структурный режим, в котором текст привязывается к строго определенной иерархической структуре. Наиболее популярный сегодня формат такой структуры — XML.

Для удобного редактирования текстов, имеющих специфическую структуру, в Adobe FrameMaker предусмотрены механизмы расширения, которые мы рассмотрим далее. Отметим, что внутренний формат документов Adobe FrameMaker не является XML-форматом, т. е. для работы с существующими XML-документами требуется реализация прямого и обратного преобразований (импорта и экспорта).

2.1. Механизмы расширения

Adobe FrameMaker предлагает два механизма расширения — структурные приложения (Structured Applications) и инструментарий разработки расширений (FDK, Frame Developer Kit).

Структурные приложения позволяют описать собственный вариант структуры документа, тогда как FDK позволяет создавать подключаемые модули, расширяющие или дополняющие функции среды FrameMaker. Структурное приложение — базовый механизм для поддержки специфичных форматов документации — их внутренней структуры и визуального представления. Этот механизм позволяет достаточно быстро перенести в Adobe FrameMaker суще-

ствующую XML-структуру документации или создать новую XML-структуру. Основной компонент структурного приложения — шаблон документа, содержащий описание структуры документа EDD (Element Definition Document).

Для структурных приложений, работающих с внешними XML-файлами, задается также и структура XML-документа в формате XML DTD (Data Type Definitions). Для описания отображения XML-документа во внутренний формат FrameMaker и обратно служат правила чтения/записи. В соответствии с этим отображением при импортировании и экспортировании производится требующаяся пользователю трансляция элементов одного типа в другие. Также для структурного приложения можно указать каскадную таблицу стилей (CSS) для XML-представления документации — в соответствии с указанными стилями документ будет отображаться в редакторе.

Инструментарий разработки расширений (FDK) используется, когда штатных средств Adobe FrameMaker и структурного приложения недостаточно — в такой ситуации необходимо создать с помощью FDK собственный модуль расширения (плагин) Adobe FrameMaker, как правило на C/C++. FDK состоит из следующих компонентов: FrameMaker Development Environment (FDE), FrameMaker API, Import/Export API. FDE включает в себя собственную систему типов (для обеспечения кроссплатформенности встроенные типы языка C недоступны для использования) и набор библиотек для работы с окружением, например с файловой системой, независимо от целевой операционной системы, что позволяет создавать кросплатформенные плагины. FrameMaker API предназначен для обеспечения взаимодействия плагина со средой Adobe FrameMaker, а также для создания графического интерфейса. Import/Export API позволяет модифицировать ход трансляции документа, представленного в виде XML-файла в документ Adobe FrameMaker и наоборот.

Далее мы остановимся на описании того, как две известные технологии разработки технической документации — DocBook и DITA — интегрированы в пакет FrameMaker.

2.2. Интеграция технологии DocBook

DocBook — технология разработки документации, предложенная в 1991 г. компаниями HaL Computer Systems и O'Reilly&Associates. Ее основная идея — разделение содержания документа и его форматирования, что позволяет создать единое исходное представление документа (single source) и на его основе автоматически генерировать документацию в разных форматах, например печатная документация (PDF), справочная система (HTMLHelp/WinHelp), электронная документация (HTML) [11].

Технология DocBook включает в себя язык, который позволяет выполнить полиграфическую разметку и форматирование текстов документов. Современная версия является XML-языком, описание схемы является открытым и стандартизовано консорциумом OASIS.

В настоящее время DocBook широко используется для разработки документации операционных систем семейства UNIX (FreeBSD, Linux), а также в мире разработки открытого ПО. Поддержка формата XML DocBook в Adobe FrameMaker реализована в виде структурного приложения XDocBook и плагина, модифицирующего стандартные функции экспортирования и импортирования XML-документов.

2.3. Интеграция технологии DITA

Технология DITA была предложена компанией IBM в 2001 г. для разработки модульной технической документации. Документация в DITA представляется в виде набора независимых топиков (topic), которые могут иметь типы. Таким образом поддерживается повторное использование крупных и логически завершенных фрагментов текста в разных контекстах.

Язык форматирования документов DITA, также как и DocBook, основан на XML и позволяет не только описывать топики, но и полностью задать форматирование текста. DITA позволяет также организовать повторное использование небольших фрагментов текста — отдельных слов, фраз, терминов, фрагментов предложений и т. д. Инструментальные средства DITA содержат пакет преобразований документов DITA в различные выходные форматы. Формат DITA стандартизован международным комитетом OASIS.

Стандартные инструменты DITA скорее являются экспериментальными и уступают DocBook в части полиграфического оформления текста. Тем не менее в ряде крупных компаний, например в IBM, DITA вполне успешно применяется для разработки больших пакетов документов, содержащих много однообразной информации.

Технология DITA в Adobe FrameMaker реализована в виде плагина, который входит в стандартную поставку, начиная с версии 8. Он предоставляет возможность импорта, создания документации, её редактирования и экспорта. В отличие от плагина DocBook, основным представлением документации которого являются 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 реализуется встроенными средствами Adobe FrameMaker без использования внешних инструментальных средств DITA.

3. Проекции языка DRL в формат FrameMaker

Проанализировав структуру похожих решений для Adobe FrameMaker (плагины для работы с документами в форматах DocBook и DITA), мы решили все элементы документации DocLine отобразить на документы FrameMaker, а для удобства навигации по проекту создать FrameMaker-книгу (аналог проекта в средах разработки типа Visual Studio), включающую в себя все составные части документации.

Язык DRL/PR, на котором разрабатывается документация в DocLine, допускает практически произвольно комбинировать конструкции обычного DocBook и собственные. То есть внутри любой конструкции DocBook могут быть DRL-конструкции и наоборот. Основное ограничение здесь заключается в том, что итоговые документы, полученные в результате публикации (напомним, что DRL-документы с DocBook-вставками конвертируются для дальнейшей обработки в «чистый» DocBook), должны быть корректными документами DocBook, тогда как каждый исходный документ сам по себе может быть лишь частично корректным.

Для описания такого «смешения» двух XML-языков в DocLine используется схема Relax NG [15] — в ней есть конструкция, идентифицирующая произвольный элемент указанного пространства имен. По этой схеме проверяется лишь частичная корректность, а полная корректность проверяется путем публикации итоговых документов и их проверки.

В EDD (языке описания схем документов Adobe FrameMaker) аналогичная конструкция отсутствует, т. е. необходимо явно описать все возможные сочетания элементов. В DocBook имеются сотни конструкций, и описание всех возможных сочетаний этих конструкций с DRL-конструкциями является очень трудоемкой задачей и существенно осложняет сопровождение полученного решения: даже при незначительных изменениях форматов DocBook или DRL потребуется нетривиальная модификация EDD-схемы. В связи с этим мы ограничились лишь некоторыми, наиболее распространенными, «контейнерными» конструкциями DocBook, которые, как правило применяются в переиспользуемых элементах: Para, Preface, Chapter, ListItem, book, Section, row и entry.

Еще одна проблема отображения XML DRL в Adobe FrameMaker — поддержка фрагментов таблиц. В DRL-элементе может содержаться один ряд таблицы или одна ячейка, которые впоследствии будут вставлены в контекст (в таблицу). Однако FrameMaker не допускает создания ряда вне таблицы или ячейки вне ряда. Для решения этой проблемы мы добавили в модификацию DRL для FrameMaker вспомогательные контейнеры для рядов (FakeTable) и ячеек (FakeRow).

4. Описание функциональности

Реализованный модуль расширения DocLineFM позволяет в рамках стандартного пакета Adobe FrameMaker (в варианте для семейства операционных систем Microsoft Windows) работать с DocLine-документацией (см. рис. 1).

В частности, поддержаны следующие функции:

- навигация по структуре документа (Structure View — иерархическое отображение документа в правой части снимка экрана на рис. 1);
- выбор допустимых в текущей позиции курсора конструкций (список в левой нижней части снимка экрана на рис. 1);

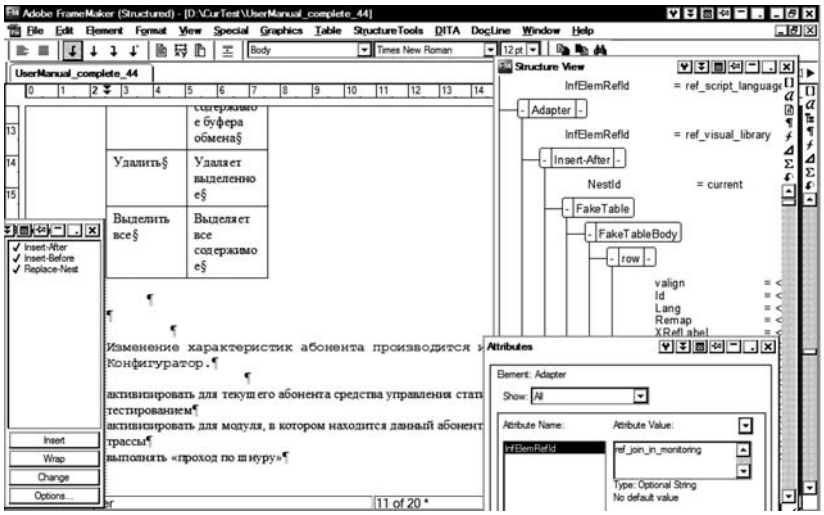

Рис. 1. Интерфейс плагина DocLineFM в среде Adobe FrameMaker: структурный режим

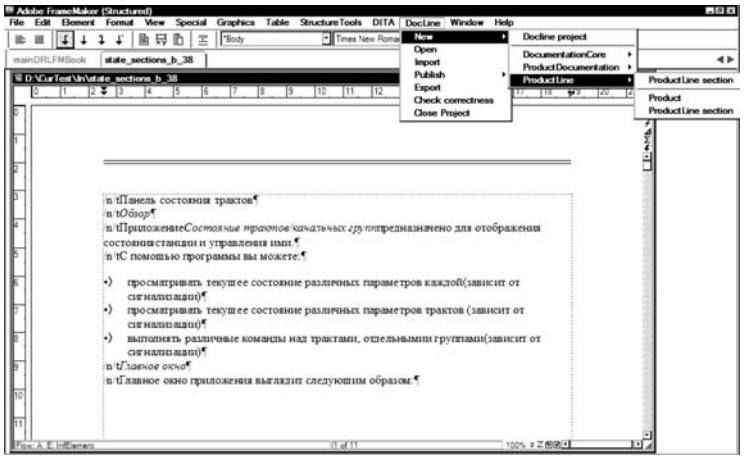

Рис. 2. Интерфейс плагина DocLineFM в среде Adobe FrameMaker: специализированное меню

- базовая WYSIWYG-функциональность (в частности, форматируются списки, заголовки, таблицы, выделение текста курсивом и полужирным шрифтом).

Кроме стандартных возможностей редактора, реализованы следующие функции, специфичные для DocLine:

- импорт существующей документации DocLine в Adobe FrameMaker;
- экспорт документации из FrameMaker в DocLine;
- публикация итоговой документации в форматах PDF и HTML по проекту DocLineFM;
- специализированное выпадающее меню, встроенное в оболочку Adobe FrameMaker, облегчающее работу с DocLine FM-документацией (рис. 2): создание и редактирование документации, вставка типовых конструкций, импорт и экспорт DocLine-документации, публикация итогового пакета документов и проверка корректности исходных текстов документации.

5. Особенности реализации

Пакет средств разработки расширений (FDK) появился в ранних версиях пакета FrameMaker более 10 лет назад и с тех пор лишь увеличивался в соответствии с новыми возможностями FrameMaker. Поскольку сам FrameMaker разработан на C/C++, то и средства расширения предназначались в первую очередь для программистов на этих же языках. Более того, для обеспечения переносимости на различные платформы был избран «минималистический» подход в смысле использования средств языка — классы не используются, т. е. фактически внутренняя структура Adobe FrameMaker «обернута» в C-интерфейс (множество функций со структурными параметрами). Кроме того, используется собственная система типов, а стандартные типы языка C запрещены.

Таким образом, разработка расширений с помощью FDK на сегодняшний день существенно уступает большинству механизмов расширения более современных систем, в том числе механизму расширения среды Eclipse — как из-за необходимости применения сложного для разработчиков языка C (в работе [8] приводится статистика, что разработка приложений на C/C++ в 4–5 раз более трудоемка, чем программирование на C#/Java, при этом в про-

граммах на C/C++ может быть в 3 и более раз больше критических дефектов), так и вследствие жестких ограничений, лишающих преимуществ использования современных интегрированных сред разработки. В итоге разработка ведется в низкоуровневом окружении, что существенно удорожает процесс и увеличивает вероятность ошибок.

Документация FDK на первый взгляд выглядит достаточно подробной, но плохо структурирована. Тем не менее в целом FDK обеспечивает доступ практически ко всей функциональности FrameMaker, доступной пользователям, и позволяет создавать функциональные расширения, хотя трудоемкость этого процесса достаточно высока.

В нашем случае для упрощения реализации мы отказались от поддержки переносимости и сосредоточили усилия на семействе операционных систем Microsoft Windows. В частности, мы переиспользовали часть реализации DocLine для Eclipse, а именно Java-модули публикации и проверки корректности. Сами по себе эти модули являются переносимыми, но интерфейс JNI для вызова Java-кода из приложений на C хоть и поддерживается на различных операционных системах, но его реализация не является универсальной, так что программы, использующие его, необходимо подстраивать для каждой платформы.

Заключение

Благодаря реализации указанных функций работа с документацией DocLine становится более привычной для технических писателей, так как реализована в стиле, традиционном для Adobe FrameMaker — одного из самых популярных инструментов создания промышленной технической документации. С другой стороны, для технических писателей, привыкших к самостоятельному редактированию XML-текстов (в частности, для программистов, исполняющих функции технического писателя), сохраняется возможность использования стандартного пакета инструментов DocLine, что позволяет работать ближе к структуре документации.

Фактически DocLine-расширение Adobe FrameMaker добавило к стандартным средствам DocLine базовую поддержку WYSIWYG-режима и инструмент иерархического просмотра проекта докумен-

тации. В дальнейшем мы планируем специально исследовать вопрос о том, насколько удобны для технических писателей созданные нами средства.

Список литературы

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

[2] *Кознов Д. В., Романовский К. Ю.* Автоматизированный рефакторинг документации семейств программных продуктов // Системное программирование. Вып. 4: Сб. статей / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд-во СПбГУ, 2009. С. 127–149.

[3] *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. Вып. 4. СПб.: Изд-во СПбГУ, 2007. С. 110–122.

[4] *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. Вып. 2. СПб.: Изд-во СПбГУ, 2009. С. 166–180.

[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, 2006. С. 191–218.

[6] *Смирнов М. Н., Кознов Д. В., Дорохов В. А., Романовский К. Ю.* Программная среда WebMLDoc для автоматизированного отслеживания изменений в пользовательской документации Web-приложений // Системное программирование. Вып. 5: Сб. статей / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд-во СПбГУ, 2010. С. 32–51.

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

[8] *Ушаков Д., Салищев С.* Использование языков и сред управляемого исполнения для системного программирования // Системное программирование. Вып. 4 / под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд-во С.-Петерб. ун-та, 2009. С. 197–216.

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

[10] *Bassett P.* Framing Software Reuse — Lessons from Real World. Prentice Hall, 1996. 365 p.

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

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

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

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

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

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

[17] Saxon 9 Java API. URL: http:// www.saxonica.com/ documentation/ javadoc/ index.html