

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

АВТОМАТИЗИРОВАННАЯ ПОДДЕРЖКА
ПОЛЬЗОВАТЕЛЬСКОЙ ДОКУМЕНТАЦИИ
WEB-ПРИЛОЖЕНИЙ, РАЗРАБАТЫВАЕМЫХ
В СРЕДЕ WEBRATIO

Дипломная работа студента 544 группы

Дорохова Вадима Александровича

Научный руководитель / подпись /	ст. преподаватель Смирнов М.Н.
Рецензент / подпись /	к. ф.-м.н., доцент Кознов Д.В.
“Допустить к защите” заведующий кафедрой, / подпись /	д.ф.-м.н., проф. Терехов А.Н.

Санкт-Петербург
2012

SAINT PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Software Engineering Chair

COMPUTER-AIDED SUPPORT OF USER MANUALS
FOR WEB-APPLICATIONS BEING DEVELOPED
IN WEBRATIO

by

Vadim Dorokhov

Master's thesis

Supervisor Senior Lect. M. N. Smirnov

Reviewer PhD, Associated Professor
D. V. Koznov

“Approved by” PhD, Professor A. N. Terekhov
Head of Department

Saint Petersburg
2012

Оглавление

Введение	4
Постановка задачи	6
Глава 1. Контекст исследования	7
Язык WebML	7
Описание языка	7
Гипертекстовая модель WebML	9
CASE-пакет WebRatio	11
Технология DocLine и язык DRL	12
Глава 2. Обсуждение проблемы и основные идеи	14
Связь спецификации программного обеспечения с документацией	14
Связь интерфейса и пользовательской документации	15
Преимущества языка WebML для данной задачи	17
Глава 3. Архитектура решения	20
Причины создания нового графического редактора WebMLDoc	20
Модель связи интерфейса с документацией	21
Генерация модели WebMLDoc	23
Управление связями интерфейса с документацией в DocLine	24
Алгоритм определения разделов документации, подлежащих проверке при изменении гипертекстовой модели	25
Глава 4. Особенности реализации и пример	27
Частота определения требующих проверки разделов документации	27
Некорректность работы модуля определения XML Diff	Ошибка! Закладка не определена.
Пример	29
Результаты	36
Список литературы	37

Введение

В современном мире практически каждому человеку ежедневно приходится иметь дело с различными программными продуктами. Кто-то создает сложные информационные системы в специализированных средах разработки, кто-то пишет отчеты в текстовых редакторах, а кто-то смотрит любимый фильм в каком-либо видеоплеере. Однако какие бы задачи не выполняли все эти программные продукты, для удобства работы с ними необходима пользовательская документация, поскольку далеко не все возможности программных продуктов интуитивно понятны. Именно поэтому при разработке программных продуктов на создание пользовательской документации тратятся значительные ресурсы.

Разработка пользовательской документации оказывается сложной задачей, т.к. зачастую документация создается параллельно с разработкой самого приложения, а изменения исходного кода приложения (добавление новых функциональных возможностей, изменение пользовательского интерфейса и пр.) требуют внесения соответствующих изменений и в пользовательскую документацию. При этом одной из серьезных проблем является наличие в документации повторяющихся фрагментов текста, поскольку в этой ситуации одни и те же изменения нужно вносить во много мест в документации. Например, возможна ситуация, когда изменение имени продукта или его версии может потребовать внесения десятков исправлений в различных документах или различных частях одного и того же документа.

Для разработки документации на основе повторного использования был разработан метод DocLine [1, 2], включающий в себя XML-язык разработки документации DRL/PR, графический язык для проектирования повторного использования документации (DRL/GR), процесс разработки документации, а также инструментальный пакет. Данный метод в первую очередь ориентирован на разработку документации семейств программных продуктов, однако он полезен и в разработке документации для отдельных приложений, если она

содержит много повторов.

Пользовательская документация важна не только для настольных (desktop) приложений, но и для активно развивающихся в настоящее время Web-приложений. В связи с непрерывным и стремительным развитием Интернета Web-приложения с каждым годом становятся все более сложными, а их функциональные возможности и «богатство» пользовательского интерфейса все больше приближают их к desktop-приложениям. Соответственно, растет число пользователей Web-приложений.

Для моделирования Web-приложений был создан ряд предметно-ориентированных языков – OOHDM [7], UWA [8], WISDOM [9] и другие. Среди них заметно выделяется и является, по сути, стандартом де-факто язык WebML (Web Modeling Language) [5, 6]. Язык WebML реализован в CASE-пакете WebRatio [5], который поддерживает автоматическую генерацию кода и интегрирован со средой разработки Eclipse.

Из-за необходимости корректировать документацию при изменениях исходного кода приложения возникает естественное желание отслеживать связи между приложением и его документацией. В данной дипломной работе предлагается один из вариантов решения этой проблемы. Рассматриваются Web-приложения, разрабатываемые с применением модельно-ориентированного подхода WebML, интерфейс которых задается гипертекстовой моделью. На этой модели присутствуют как сами экранные формы, так и переходы между ними (гиперссылки), благодаря чему появляется возможность естественной связи элементов этой модели и составных частей документации, поскольку, как правило, пользовательская документация либо описывает некоторые сценарии использования пользовательского интерфейса, либо его отдельные окна. Для разработки документации используются язык DRL/PR и пакет DocLine, а в качестве среды разработки Web-приложений – продукт WebRatio, который реализует подход WebML.

Постановка задачи

Цель данной дипломной работы состоит в разработке метода, позволяющего по изменениям пользовательского интерфейса Web-приложения, создаваемого в среде WebRatio, определить список разделов пользовательской документации в DocLine, требующих корректировки или проверки.

Для достижения этой цели были поставлены следующие задачи.

1. Изучить язык WebML, его гипертекстовую модель, а также CASE-пакет WebRatio.
2. Предложить модель зависимости между пользовательским интерфейсом и пользовательской документацией.
3. Детализировать эту модель для гипертекстовой модели языка WebML, предусматривая следующие возможности.
 - Создавать, изменять и удалять связи элементов модели с разделами документации.
 - Автоматически генерировать список требующих проверки разделов документации при изменении гипертекстовой модели WebML.
 - Вести логирование изменений гипертекстовой модели для дальнейшего внесения исправлений в документацию.
4. Выполнить пилотную реализацию этой модели с поддержкой механизма циклической разработки.
5. Провести апробацию разработанного решения на реальном примере.

Глава 1. Контекст исследования

Язык WebML

Описание языка

Язык WebML позволяет разрабатывать Web-приложения с применением модельно-ориентированного подхода. Для создания отдельного приложения необходимо спроектировать несколько его моделей, а исходный код будет сгенерирован по ним автоматически. Каждая из этих моделей описывает характеристики приложения со своей точки зрения.

Модель данных (data model) позволяет описывать структуру базы данных приложения. Данная модель является вариантом модели сущность-связь. На рисунке 1 показан пример модели данных Web-приложения “Acme”, разработанного на языке WebML в CASE-пакете WebRatio.

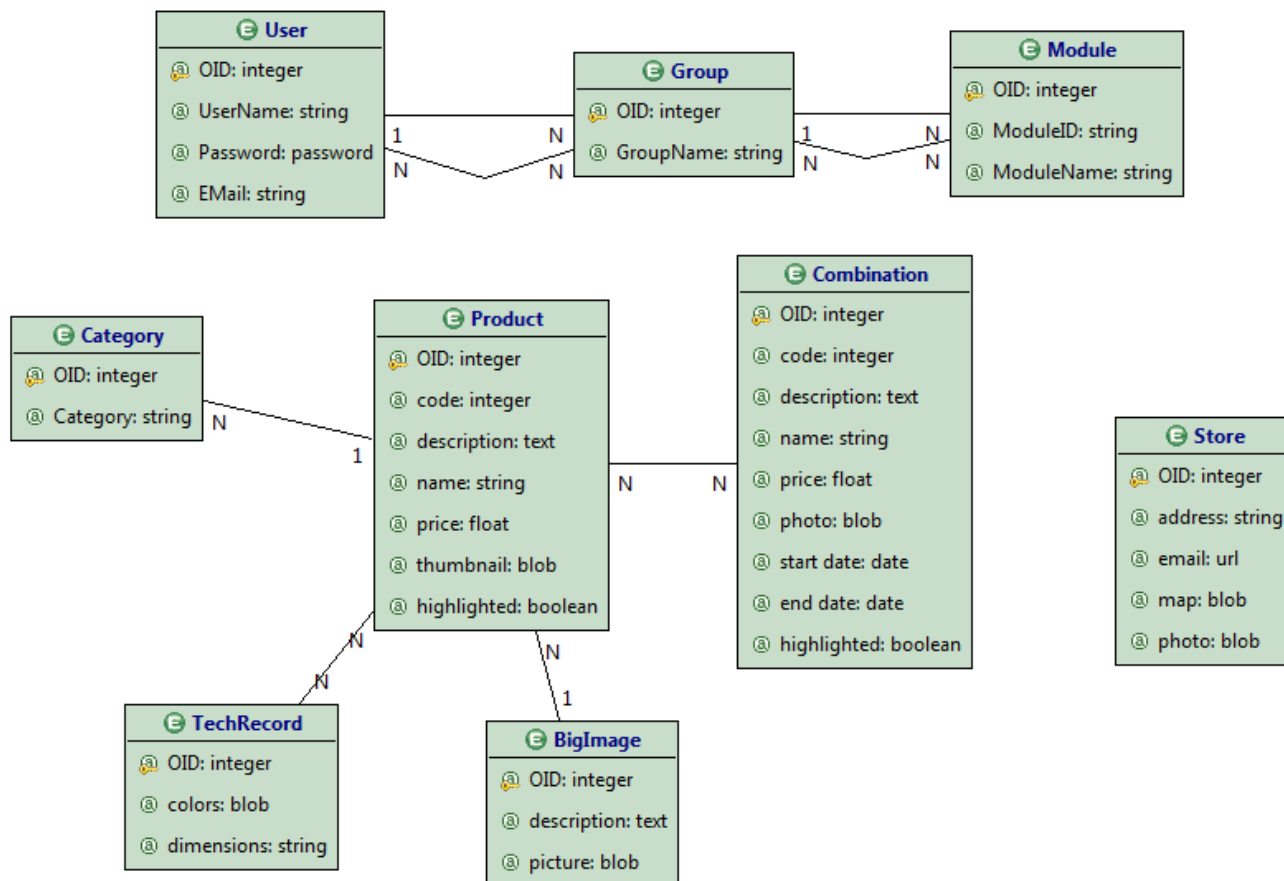


Рис.1. Пример модели данных WebML

Гипертекстовая модель (hypertext model) служит для описания схемы интерфейса Web-приложения. Она позволяет определить страницы, находящиеся на них элементы управления, связь их с базой данных и навигацию между ними. Стоит отметить, что непосредственно внешний вид интерфейса (т.е. цвета, вид отображаемых элементов управления, их размеры) задается с помощью стилей на уровне модельного инструмента, а не в гипертекстовой модели. На рисунке 2 показан пример гипертекстовой модели Web-приложения “Acme”, разработанного на языке WebML в CASE-пакете WebRatio.

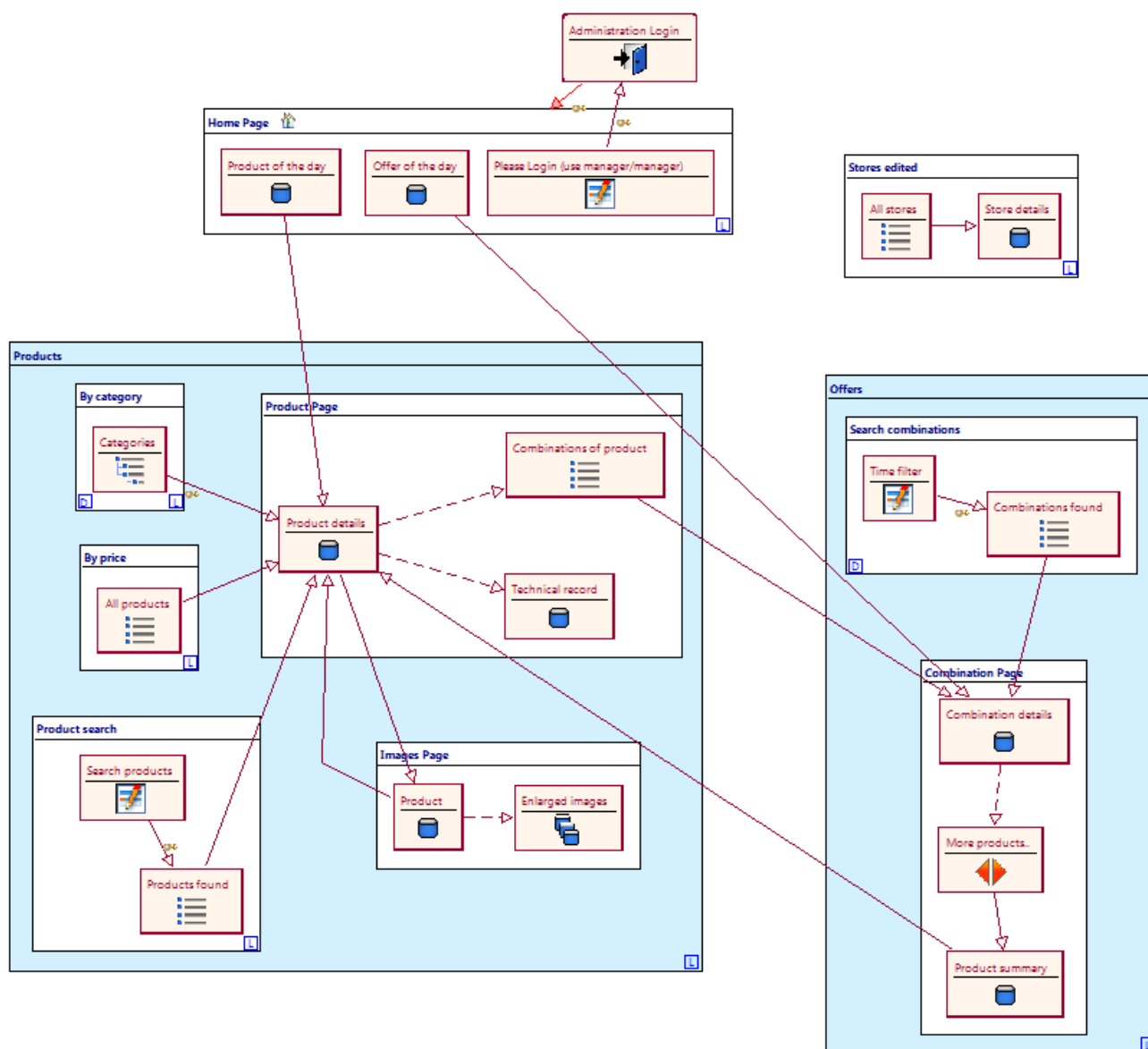


Рис.2. Пример гипертекстовой модели WebML

Модель управления контентом (content management model) расширяет гипертекстовую модель дополнительными конструкциями, такими как операции

и транзакции. Это позволяет задавать поведение экранных форм, осуществлять вызовы предопределенных операций (например, вставка и удаление объектов) и интеграцию с внешними сервисами.

Гипертекстовая модель WebML

Как уже отмечалось, гипертекстовая модель позволяет описывать схему интерфейса и навигацию Web-приложения.

Структура описывается в терминах *страниц* (pages) – контейнеров информации, которая предоставляется пользователю в один момент времени. Страницы состоят из *контент-модулей* (content units) – атомарных элементов, которые используются для представления информации, описанной в модели данных. В WebML существуют разные типы контент-модулей – см. рисунок 3.

Как уже отмечалось, контент-модули группируются в страницы, а те, в свою очередь, могут группироваться в *области* (areas) и *профили сайта* (siteviews) – например, профиль администратора, неавторизованного пользователя, авторизованного пользователя.

Навигация приложения описывается *связями* (links), которые могут соединять страницы и контент-модули. Связи могут передавать данные (тогда они называются контекстными) и/или управление. В отдельный вид выделяют *транспортные связи*, которые используются для передачи только информации, по таким связям нельзя осуществить переход (передачу управления).


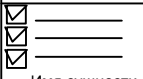

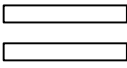
Название	Описание	Нотация
DataUnit	Публикует информацию об одном экземпляре сущности модели данных.	<div>Имя модуля</div>  <div>Имя сущности Selector</div>
MultiDataUnit	Публикует информацию о множестве экземпляров сущностей модели данных, но без возможности выбора какой-либо из них.	<div>Имя модуля</div>  <div>Имя сущности Selector</div>
IndexUnit	Публикует информацию о множестве объектов модели данных с возможностью выбора одной из них или целого набора.	<div>Имя модуля</div>  <div>Имя сущности Selector</div>
Multi-choice IndexUnit	Вариант индекса, в котором каждый элемент ассоциирован с чекбоксом, позволяющим пользователю выбрать несколько объектов.	<div>Имя модуля</div>  <div>Имя сущности Selector</div>
Hierarchical IndexUnit	Вариант индекса, в котором элементы организованы в виде многоуровневого дерева. Позволяет пользователю выбрать один элемент с каждого уровня.	<div>Имя модуля</div>  <div>Имя сущности Selector</div>
ScrollUnit	То же, что Index Unit, но с добавлением возможности прокрутки списка элементов.	<div>Имя модуля</div>  <div>Имя сущности Selector</div>
EntryUnit	Предназначен для ввода данных в Web-систему. Запись в базу данных осуществляется не самим этим модулем: он выдает значения, которые используются как параметры в операциях модели контента.	<div>Имя модуля</div> 

Рис.3. Основные контент-модули языка WebML [3]

Помимо этого, на гипертекстовой модели могут присутствовать операции (operation units), которые используются для обозначения какого-либо процесса, который выполняется в результате перехода по связи. Операции обозначают либо действия с данными, либо выполнение каких-то внешних функций. Основные типы операций представлены и кратко описаны на рисунке 4.

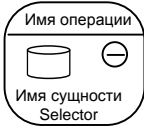


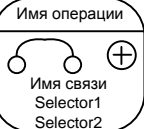
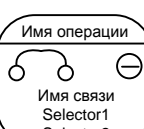

Название	Описание	Нотация
Delete	Удаляет экземпляр сущности.	
Create	Создает экземпляр сущности.	
Modify	Изменяет связь между экземплярами двух сущностей.	
Connect	Создает связь между экземплярами двух сущностей.	
Disconnect	Удаляет связь между экземплярами двух сущностей	
Selector	Делает запрос к схеме данных, но не отображает извлеченную информацию в интерфейсе. Используется как в страницах, так и вне их для вспомогательных запросов.	

Рис.4. Основные операции языка WebML [3]

Операции могут вызываться при передаче управления от одних контент-модулей/страниц другим. Для реализации многовариантных переходов в зависимости от результатов выполнения операции используются так называемые OKLink- и KOLink-ссылки, соответствующие успешному и неуспешному завершению операции.

CASE-пакет WebRatio

CASE-пакет WebRatio – это интегрированный со средой разработки Eclipse инструмент для создания Web-приложений на языке WebML. WebRatio позволяет по моделям WebML генерировать исходный код приложения с использованием стандартов JSP, HTML и XHTML.

Сам процесс разработки Web-приложения в среде WebRatio включает в себя создание нескольких моделей – модели данных, гипертекстовой модели и модели управления контентом, а также дальнейшую генерацию исходного кода по этим моделям. На рисунке 5 показана рабочая область CASE-средства WebRatio, а также часть гипертекстовой модели Web-приложения “Acme”, входящего в основную поставку пакета.

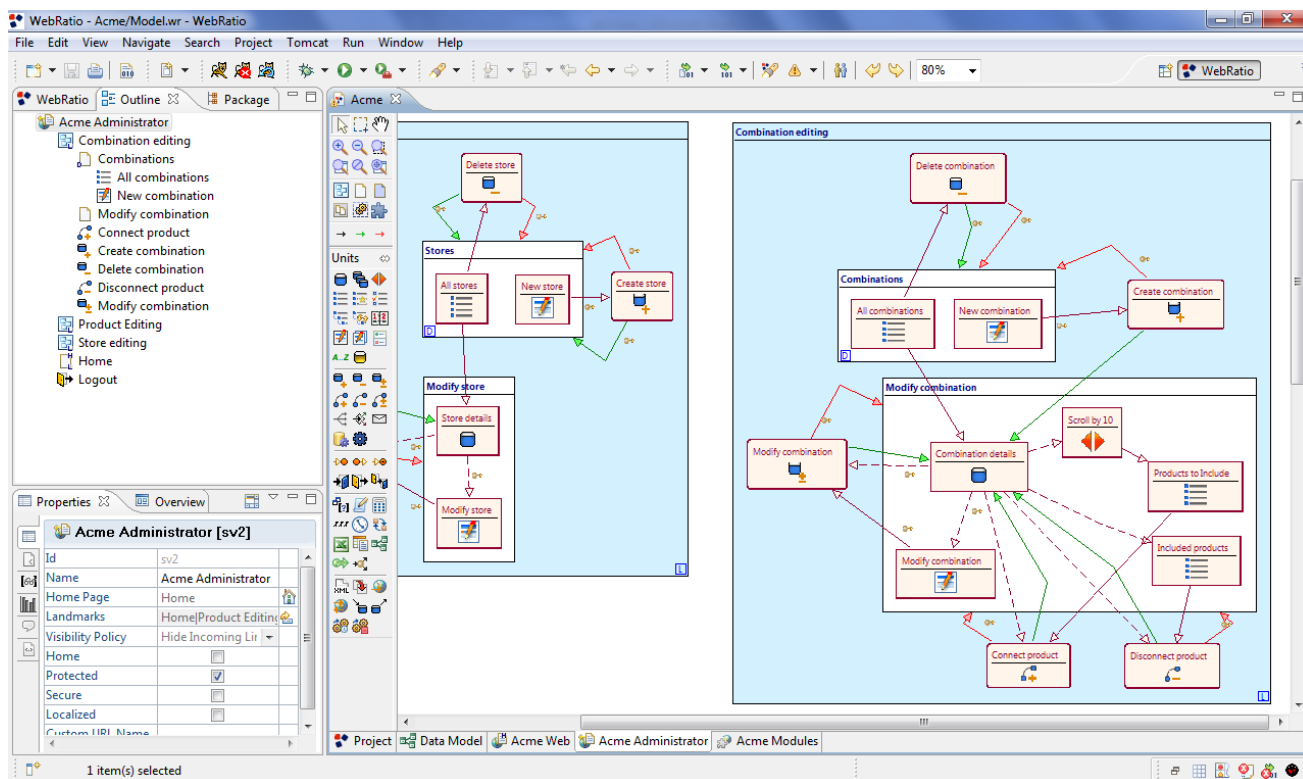


Рис.5. Рабочая область продукта WebRatio

Технология DocLine и язык DRL

Для проектирования и разработки документации на основе повторного использования на кафедре системного программирования математико-механического факультета СПбГУ был разработан метод DocLine, включающий в себя оригинальный язык разработки документации DRL, процесс разработки документации, а также инструментальный пакет. Данный метод охватывает весь жизненный цикл разработки документации от проектирования до публикации итоговых документов и поддерживает плановое адаптивное повторное использование документации.

Язык DRL (Document Reuse Language) имеет две нотации – графическую (DRL/GR) и текстовую (DRL/PR). Графическое представление служит для проектирования структуры повторного использования документации. Текстовое представление позволяет описать в виде XML-представления варианты конфигурирования повторно используемых компонент и конкретные конфигурации для порождения конечных документов.

Глава 2. Обсуждение проблемы и основные идеи

Связь спецификации программного обеспечения с документацией

При разработке программных продуктов, как правило, документация разрабатывается параллельно. Ни для кого не секрет, что эти два процесса взаимосвязаны, поскольку в документации описывается сам программный продукт и правила работы с ним. Таким образом, изменение исходного кода программного продукта часто требует внесения исправлений в документацию.

На рисунке 6 показана взаимосвязь программного продукта и его документации.

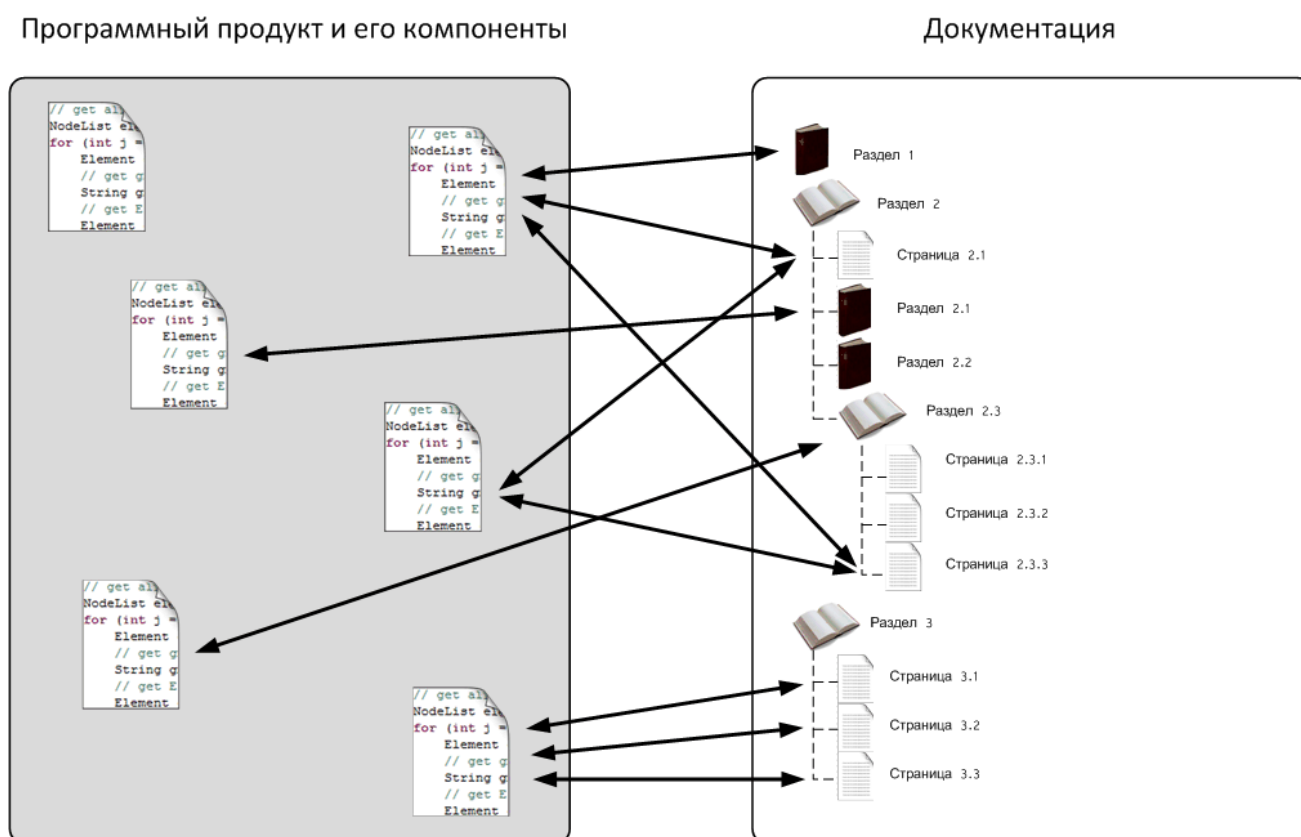


Рис.6. Взаимосвязь программного продукта и его документации

Из рисунка ясно, что связи программного продукта с документацией имеют тип «многие ко многим». Это обусловлено тем, что отдельные компоненты приложения могут быть описаны в разных разделах документации, и наоборот, некоторые разделы документации могут описывать сразу несколько

программных компонентов.

Кроме того, на рисунке показано, что существуют несвязанные ни с одним разделом документации программные компоненты, что соответствует еще незадокументированным фрагментам кода. В свою очередь, несвязанные ни с одной программной компонентой разделы документации могут содержать вспомогательную, не относящуюся непосредственно к коду приложения информацию.

Однако, несмотря на очевидность существования зависимостей между исходным кодом программного продукта и его документацией, выявить и формализовать эти зависимости оказывается не просто. Например, в общем случае совершенно непонятно что и как нужно изменить в документации, если поменялся код какого-либо класса.

Однако для приложений, разрабатываемых на основе модельно-ориентированного подхода, такие связи установить значительно легче. Основная концепция этого подхода — набор высокоуровневых моделей в центре всей разработки с последующей автоматической генерацией программного кода [10]. Все изменения, которые нужно вносить в приложение, вносятся в модели, а затем автоматически переносятся в исходный код программы.

С точки зрения вышеуказанных зависимостей с документацией модельно-ориентированный подход очень удобен, поскольку позволяет «привязать» документацию не к коду «напрямую», а к модели программного продукта, изменения которой значительно проще отслеживать. Создать подобные связи с документацией возможно с помощью соответствующей «раскраски» модели приложения, определив, какие ее элементы с какими разделами документации связаны.

Связь интерфейса и пользовательской документации

Идею связи спецификации программного обеспечения с документацией

можно рассмотреть на более конкретном примере – связь модели пользовательского интерфейса и пользовательской документации приложения.

Чтобы выявить зависимости между пользовательским интерфейсом приложения и его документацией мною была изучена пользовательская документация некоторых реальных программных продуктов (семейство продуктов Microsoft Office, The Bat!, iTunes, ESET NOD32, Notepad++ и др.). Оказалось, в основном, она строится по одному из двух следующих принципов.

1. Описание сценариев работы с приложением. В документации такого типа описывается, какие последовательности элементов интерфейса приложения нужно выбирать, чтобы выполнить то или иное действие.
2. Непосредственное описание пользовательского интерфейса приложения. В документации этого типа присутствует описание отдельно взятых элементов пользовательского интерфейса и их функционала.

На рисунке 7 показан пример пользовательской документации почтового клиента The Bat!, построенной по первому принципу.

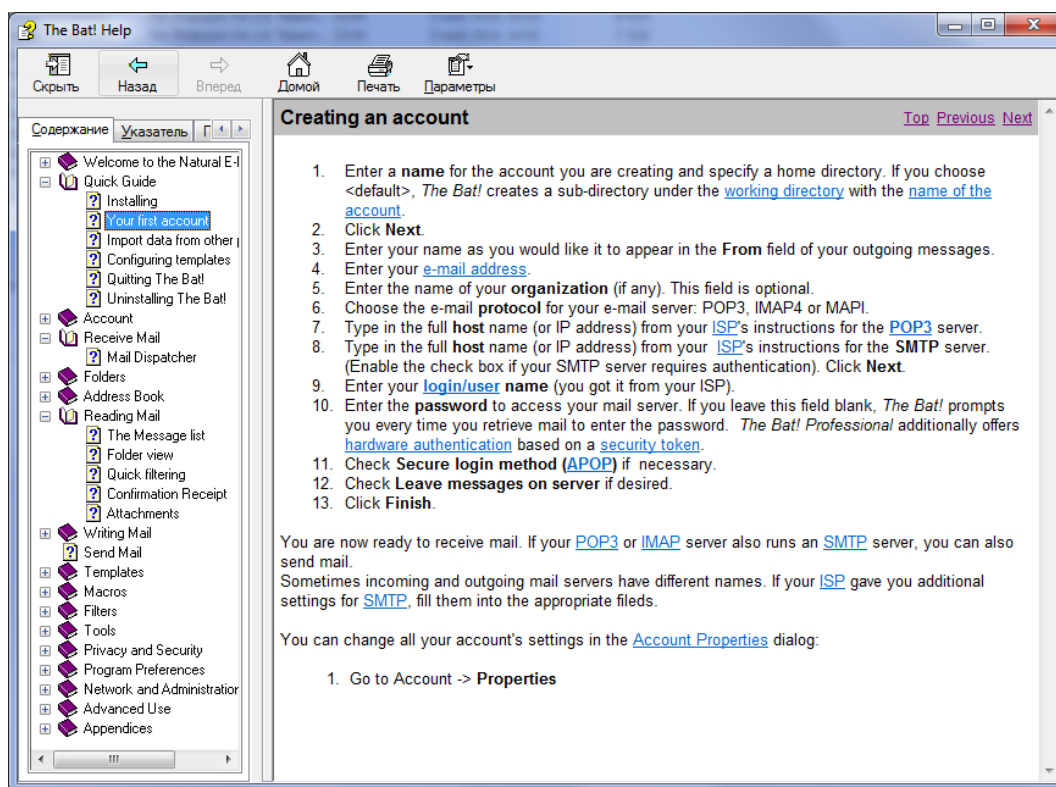


Рис.7. Пример документации, описывающей сценарии работы с приложением.

На рисунке 8 показан пример пользовательской документации антивирусной программы ESET NOD32, построенной по второму принципу.

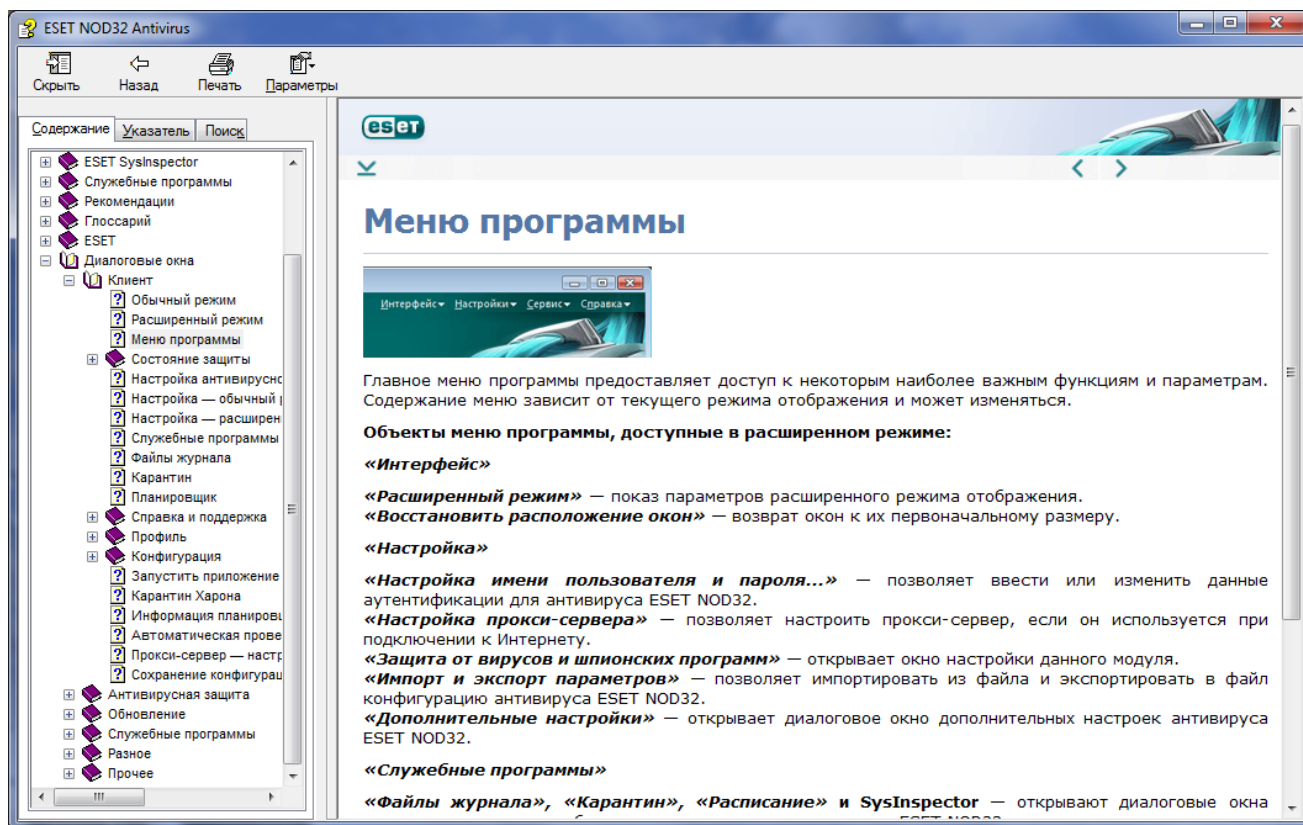


Рис.8. Пример документации, описывающей непосредственно пользовательский интерфейс приложения.

Первый принцип используется значительно шире, однако и непосредственное описание пользовательского интерфейса в документации встречается часто.

Преимущества языка WebML для данной задачи

Как уже отмечалось ранее, язык WebML позволяет создавать Web-приложения с помощью проектирования нескольких моделей, каждая из которых описывает характеристики приложения со своей точки зрения. Важно, что весь целевой код приложения генерируется автоматически по этим моделям и не требует «ручных» доработок. Таким образом, вся разработка приложения сосредоточена на разработке моделей и происходит в пакете WebRatio

Гипертекстовая модель WebML, которая нас интересует, позволяет описывать интерфейс разрабатываемого Web-приложения, а также сценарии работы с ним. Интерфейс Web-приложения задается с помощью отдельных элементов, используемых для представления информации, описанной в модели данных. Такими элементами являются контент-модули (Content Units), состоящие из контент-модулей страницы (Pages), а также области (Areas), состоящие из отдельных страниц. Сценарии же присутствуют на гипертекстовой модели как последовательности элементов интерфейса, соединенных связями (Links).

На рисунке 9 показаны примеры элементов интерфейса и сценария на гипертекстовой модели WebML.

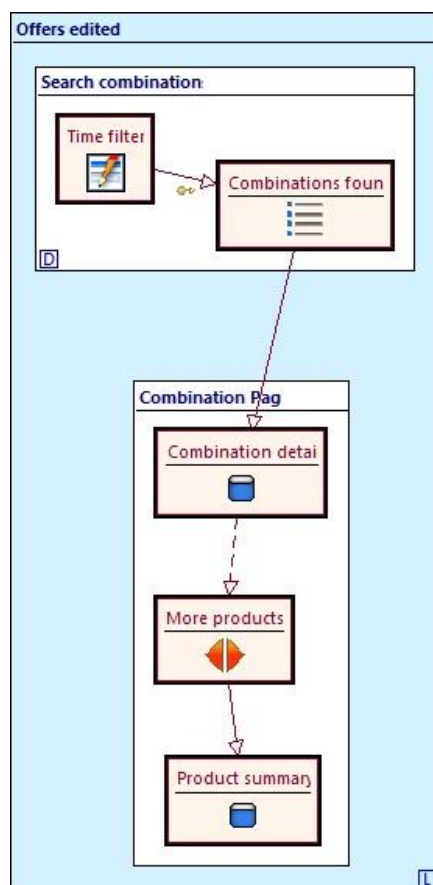


Рис.9 Примеры элементов интерфейса и сценария на гипертекстовой модели WebML

На рисунке показана область с названием “Offers edited”, которая состоит из двух страниц – “Search combinations” и “Combination Page”. Каждая из

страниц, в свою очередь, состоит из нескольких контент-модулей. Все эти элементы являются элементами интерфейса.

Как видно из рисунка, обрамленные толстой черной рамкой контент-модули соединяются друг с другом связями (Links). А вся последовательность этих контент-модулей представляет собой сценарий работы с приложением.

Чтобы связать интерфейс Web-приложения с пользовательской документацией, необходимо к выбранным элементам интерфейса добавить связи с соответствующими разделами документации. В качестве разделов документации в DocLine рассматриваются информационные элементы (InfElements), а в качестве элементов интерфейса Web-приложения, разрабатываемого в среде WebRatio, – все контент-модули (Content Units), страницы (Pages) и области (Areas). Каждый элемент интерфейса можно связать со многими разделами документации, и наоборот, каждый раздел документации может быть связан с несколькими элементами интерфейса.

Из всего вышесказанного можно сделать вывод, что в контексте поставленной задачи язык WebML обладает следующими преимуществами.

1. При разработке Web-приложений на языке WebML используется модельно-ориентированный подход, позволяющий связывать документацию не с исходным кодом, а с моделью приложения.
2. Гипертекстовая модель WebML содержит описания пользовательского интерфейса Web-приложения, а также сценариев работы с ним, что соответствует реальным принципам построения документации.

Глава 3. Архитектура решения

Причины создания нового графического редактора WebMLDoc

Для выполнения цели работы было необходимо расширить функциональность графического редактора гипертекстовой модели WebRatio возможностью создавать связи между элементами пользовательского интерфейса приложения и разделами пользовательской документации в DocLine. Теоретически, это возможно осуществить двумя способами:

- расширить графический редактор WebRatio;
- разработать отдельный графический редактор, импортирующий часть гипертекстовой модели из WebRatio и позволяющий выполнять всю необходимую работу по «раскраске» этой модели и связи с документацией.

Однако, из-за отсутствия API (Application Program Interface, интерфейс программирования приложений) продукта WebRatio, позволяющего расширить его функциональность, а также из-за отсутствия исходных кодов данной среды разработки, изменение графического редактора WebRatio оказалось невозможным. В связи с этим было принято решение о создании отдельного графического редактора WebMLDoc.

К разрабатываемому графическому редактору WebMLDoc были предъявлены следующие требования:

- визуальная схожесть модели WebMLDoc с гипертекстовой моделью в графическом редакторе WebRatio;
- возможность создавать, изменять и удалять связи элементов интерфейса приложения с разделами документации в DocLine;
- связь элементов интерфейса приложения с разделами документации в DocLine должна быть типа «многие ко многим»;
- отсутствие возможности добавлять, изменять или удалять элементы

гипертекстовой модели;

- поддержка механизма циклической разработки, то есть добавление новых элементов в модель WebMLDoc при добавлении их в гипертекстовую модель.

Графический редактор WebMLDoc разрабатывался в среде Eclipse с использованием технологии GMF (Graphical Modeling Framework) [11], предназначенной для разработки графических редакторов диаграмм.

Модель связи интерфейса с документацией

В ходе исследования была разработана следующая модель связи пользовательского интерфейса Web-приложений (гипертекстовой модели WebRatio) и пользовательской документации, разрабатываемой в среде DocLine, – см. рисунок 10.

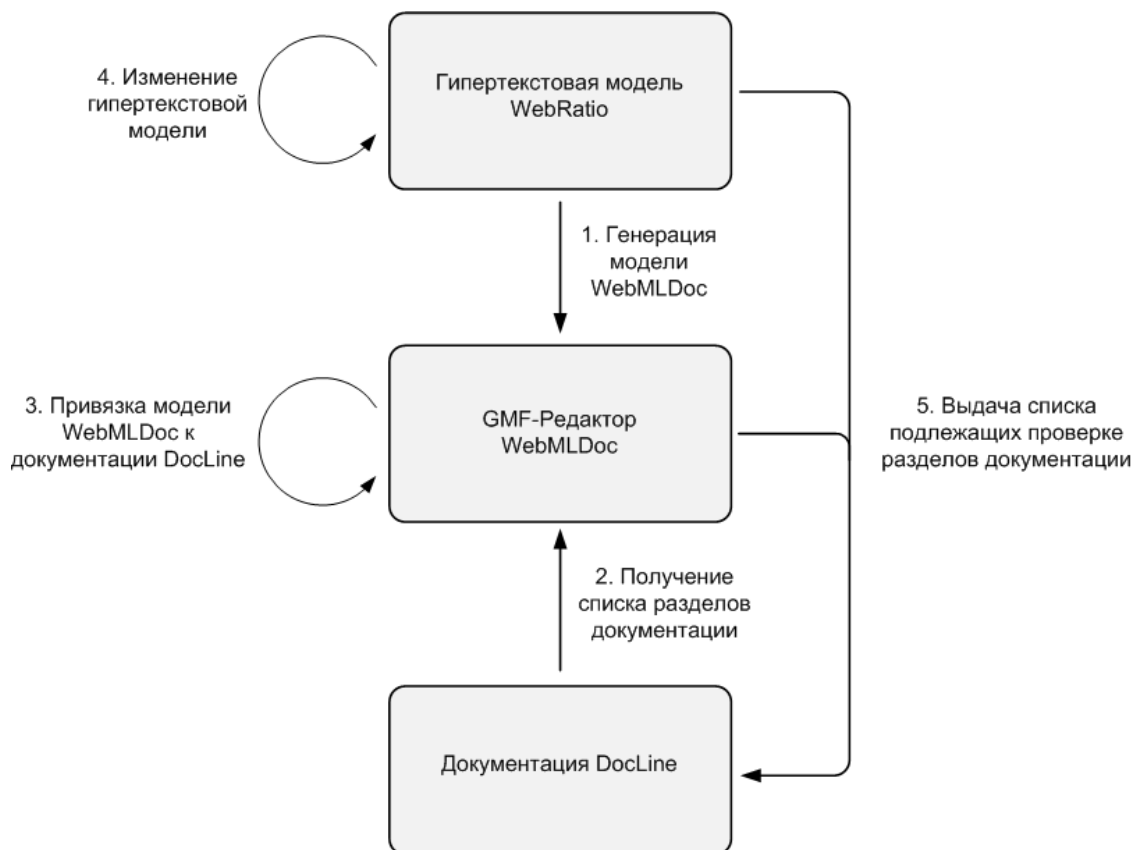


Рис.10. Модель связи пользовательского интерфейса Web-приложений, разрабатываемых в среде WebRatio, с пользовательской документацией в DocLine

На *шаге 1* из гипертекстовой модели WebRatio генерируется модель графического редактора WebMLDoc.

Шаг 1 выполняется автоматически при каждом открытии графического редактора WebMLDoc или при вызове команды “Update Model”. Автоматическая регенерация схемы позволяет модели WebMLDoc быть актуальной при каждом ее открытии, вне зависимости от изменений, внесенных в исходную гипертекстовую модель WebRatio.

Для создания связей между элементами интерфейса и документацией в DocLine, необходимо знать, между какими элементами эта связь устанавливается. Именно для этого на *шаге 2* графический редактор получает список разделов документации, к которым эта связь может быть добавлена.

Шаг 3 – это автоматическое копирование уже существующих связей с документацией в сгенерированную модель, а также дальнейшей раскраски получившейся модели. Раскраску модели WebMLDoc осуществляет разработчик, по своему выбору определяя, какие элементы интерфейса с какими разделами документации связаны. На данном этапе могут не только добавляться новые связи с документацией, но также редактироваться или удаляться уже существующие.

На *шаге 4* происходит внесение изменений в гипертекстовую модель WebRatio. В данной дипломной работе реализована поддержка механизма циклической разработки, поэтому гипертекстовую модель можно менять в любой момент разработки.

На *шаге 5* определяется, как была изменена гипертекстовая модель WebML, и с учетом существующей раскраски модели WebMLDoc пользователю предлагается список возможных изменений в документации. Шаг 5 автоматически вызывается при открытии редактора WebMLDoc, однако помимо этого разработчик может запустить эту проверку в любое время, чтобы получить актуальный список изменений. Все изменения, полученные на данном этапе,

протоколируются, и в дальнейшем могут быть исправлены разработчиком.

Генерация модели WebMLDoc

Модель любого графического редактора, созданного с помощью технологии GMF(Graphical Modeling Framework), хранится в двух файлах. В первом из них хранится структура данной модели, т.е. какие элементы присутствуют в модели, кто чьим потомком является, между какими элементами существуют связи и т. д. Во втором файле хранится расположение элементов модели, т.е. их координаты на диаграмме.

Для визуальной схожести модели WebMLDoc с гипертекстовой моделью WebML нам необходимо корректно генерировать оба этих файла.

Генерация структуры модели WebMLDoc выполняется с помощью специально созданной XSL-трансформации (eXtensible Stylesheet Language Transformations) [12], обрабатываемой процессором Saxon [13] и дальнейшей обработки полученного результата. Обработка применяется для корректировки полученной модели – изменения форматов идентификаторов, а также для выделения координат элементов модели в отдельный файл.

Структура модели WebMLDoc хранится в файлах с расширением “webml”. Отдельный файл соответствует одному профилю сайта (siteview). Также в этом файле хранятся связи элементов интерфейса с документацией, о которых рассказано далее.

Для корректного размещения элементов модели WebMLDoc на диаграмме применяется следующий алгоритм.

1. Координаты всех элементов модели WebMLDoc копируются из гипертекстовой модели в отдельный файл с именем “siteview%№%.coords”, где %№% – это номер обрабатываемого профиля сайта.

2. Для каждого профиля сайта (siteview) из ранее созданной модели WebMLDoc (если она есть) копируются координаты элементов. Полученные координаты также записываются в файл “siteview%№%.coords”, при этом сохраненные на *шаге 1* координаты переписываются. Если у какого-то элемента не было координат в редакторе WebMLDoc (например, если этот элемент только что добавлен), то его координаты не перезаписываются и используются стандартные из гипертекстовой модели, полученные на *шаге 1*.
3. Полученный список координат применяется к модели WebMLDoc.

Данный алгоритм, с одной стороны, позволяет модели WebMLDoc визуально выглядеть так же, как гипертекстовая модель WebML, а с другой стороны предоставляет пользователю возможность самому расставлять элементы модели на диаграмме так, как ему это удобно. А новые элементы, добавленные в WebRatio, будут появляться в модели WebMLDoc на тех же местах, на которых они расположены в редакторе WebRatio.

Расположение всех элементов модели, т.е. полученные по вышеописанному алгоритму координаты, а также автоматически сгенерированные параметры отображения хранятся в файлах с расширением “webml_diagram”. Отдельный файл соответствует одному профилю сайта (siteview).

Управление связями интерфейса с документацией в DocLine

При каждом открытии графического редактора WebMLDoc его модель генерируется заново по гипертекстовой модели WebRatio. В сгенерированной с помощью XSL-трансформации модели отсутствуют связи элементов интерфейса с документацией, поэтому актуальным становится вопрос о сохранении существующих связей интерфейса с документацией при каждом открытии редактора WebMLDoc.

Данный вопрос решается следующим способом. При каждом открытии редактора WebMLDoc создается резервная копия существующих моделей

WebMLDoc. А затем связи интерфейса с документацией из резервной копии добавляются в только что сгенерированную модель.

Стоит отметить, что в графическом редакторе WebRatio запрещено менять идентификаторы элементов. Поэтому связи копируются по идентификаторам элементов, к которым они относятся. Это позволяет добиться следующих результатов.

- Если элемент был удален в графическом редакторе WebRatio, то относящаяся к нему связь с документацией также будет удалена в редакторе WebMLDoc.
- Если элемент был изменен в графическом редакторе WebRatio (изменено его имя или любой другой атрибут), то связь с документацией у этого элемента все равно останется.

Таким образом, решается вопрос автоматического удаления неактуальных и сохранения существующих связей интерфейса с документацией.

Добавление новых, а также удаление или редактирование старых связей осуществляется пользователем с помощью стандартных средств графического редактора WebMLDoc.

Алгоритм определения разделов документации, подлежащих проверке при изменении гипертекстовой модели

Поскольку пользовательская документация напрямую связана с пользовательским интерфейсом приложения, изменение последнего может потребовать внесения большого количества исправлений в документацию. Следующий алгоритм позволяет автоматизировать определение разделов документации, подлежащих проверке при изменении гипертекстовой модели в WebRatio.

1. Алгоритм запускается автоматически при открытии редактора WebMLDoc, при вызове команды “Update Model” или при вызове команды “Check

Documentation” (подробнее о запуске алгоритма в этих случаях см. главу 4).

2. Гипертекстовая модель, хранящаяся на жестком диске в иерархичной структуре файлов и папок, собирается в один файл.
3. На основе собранной гипертекстовой модели и ее резервной копии, сохраненной при предыдущем запуске алгоритма, вычисляются изменения, внесенные в модель разработчиком с момента последнего запуска этого алгоритма.
4. Для каждого измененного элемента в модели WebMLDoc выбираются разделы документации, с которыми он был связан, и добавляются в список разделов документации, требующих проверки.
5. Полученный список протоколируется и предоставляется пользователю.
6. Сохраняется новая резервная копия гипертекстовой модели.

На *шаге 3* для определения внесенных в гипертекстовую модель изменений применяется библиотека JExamXML [14], предназначенная для вычисления XML Diff. Это бесплатная библиотека, позволяющая вычислять разницу двух XML-файлов и работающая через интерфейс командной строки или из Java-приложения с использованием специального API.

На *шаге 5* все изменения заносятся в специальный журнал, который может быть открыт пользователем позже в любой момент. В журнале присутствует возможность отмечать разделы документации, которые были проверены и исправлены.

Глава 4. Особенности реализации и пример

Частота определения требующих проверки разделов документации

При разработке алгоритма определения разделов документации, требующих проверки при изменении интерфейса, возник вопрос, как часто этот алгоритм запускать. Достаточно ли вызова алгоритма только по требованию пользователя?

Рассмотрим пример. Разработчик создает Web-приложение в среде WebRatio и параллельно создает документацию в DocLine. Процесс разработки итеративный, т.е. разработчик может изменять гипертекстовую модель WebRatio постепенно, изменяя на каждой итерации отдельные элементы гипертекстовой модели. В конце каждой итерации разработчику предлагается список разделов документации, требующих проверки. Предположим, что в графическом редакторе WebMLDoc уже созданы связи между интерфейсом и документацией. Пусть на n -й итерации пользователь запускает вышеуказанный алгоритм с целью узнать, какие разделы документации ему нужно исправить. Алгоритм выдает список требующих проверки разделов и сохраняет резервную копию гипертекстовой модели, чтобы по ней в дальнейшем определять изменения. После этого начинается $(n+1)$ -я итерация, на которой разработчик выполняет три действия.

1. Он добавляет в WebRatio новую страницу для данного Web-приложения.
2. Открывает редактор WebMLDoc и связывает эту страницу с каким-либо разделом документации.
3. Возвращается в редактор WebRatio и меняет у страницы имя.

Таким образом, с момента последней проверки документации была добавлена новая страница, а затем изменено ее имя.

Поскольку между измененной страницей и документацией установлена

связь (второе действие (n+1)-й итерации), то необходимо оповестить разработчика о том, что имя страницы изменено, и следует проверить соответствующие разделы документации. Однако, поскольку алгоритм сравнивает гипертекстовую модель с ее резервной копией, сохраненной при последнем запуске алгоритма (конец n-й итерации), данные изменения останутся незамеченными, т.к. в резервной копии добавленной страницы еще не было. Поэтому будет сказано только о добавлении новой страницы, и ни слова об изменении ее имени. Таким образом, мы потеряем важное действие и не оповестим разработчика о требуемых изменениях в документации.

Из этого следует, что недостаточно вызывать алгоритм только по требованию пользователя, и нужны дополнительные вызовы алгоритма при возникновении определенных событий. Таким событием является обновление модели в графическом редакторе WebMLDoc, осуществляемое автоматически при каждом открытии редактора или вручную по вызову специальной команды «Update model». Вызов алгоритма проверки документации при обновлении модели помогает избежать вышеописанной ситуации, поскольку добавление элемента будет отслежено еще до связывания этого элемента с документацией, а любые последующие изменения будут запротоколированы.

Проблемы при выборе библиотеки для выполнения XML diff

Как уже говорилось ранее, чтобы определить изменения, внесенные в гипертекстовую модель, используется библиотека JExamXML.

Изначально, в данном проекте использовалась библиотека XOperator [15], разработанная компанией Living Pages Research GmbH. Данный продукт может работать как command-line приложение, или может быть использован внутри Java-проекта с помощью Java XOperator API. Тестирование данного модуля как command-line приложения ошибок не выявило. Поэтому библиотека XOperator была использована в коде проекта через API. Для определения отдельных изменений в гипертекстовой модели был реализован алгоритм, позволяющий

разбирать результат работы XOperator и выделять отдельные изменения.

Однако при дальнейшем тестировании библиотеки было выяснено, что через API она работает не так, как из командной строки, и к тому же некорректно. К примеру, в листинге 1 показан код XML-файла до внесения изменений и после них.

<pre><root> <element id="id1" name="elem1"/> <element id="id2" name="elem2"/> <element id="id3" name="elem3"/> </root></pre>	<pre><root> <element id="id2" name="elem2"/> <element id="id3" name="elem3"/> </root></pre>
До изменений	После изменений

Листинг 1. Пример XML-файла до и после внесения изменений

В данном примере изменения заключаются только в удалении элемента с `id="id1"`. Однако результат работы XOperator окажется другим – будет сообщено, что удалены все три элемента, а затем добавлены второй и третий. В общем случае, сообщается, что были удалены все элементы того же уровня, что и первый встреченный удаленный элемент, следующие за ним, а затем они же добавлены.

В худшем случае может получиться, что при удалении одного элемента будет сообщено, что удалена вся модель, а потом вся модель без одного элемента добавлена, что категорически не соответствует действительности.

Из-за некорректной работы библиотеки было решено отказаться от ее использования, и использовать аналогичную библиотеку JExamXML.

Пример

Разработанный метод был опробован на Web-приложении Asme, являющемся частью стандартной поставки WebRatio.

Asme – это интернет магазин с поддержкой двух типов профилей: профиль обычного пользователя (покупателя) и профиль администратора сайта, который может изменять информацию о товарах и предложениях интернет магазина. Доступ к профилю администратора осуществляется через форму входа,

расположенную в правой верхней части главной страницы сайта (см. рисунок 11).

The screenshot shows a login form titled "Please Login (use manager/manager)". It contains two input fields: "username" and "password". Below these fields is an "Enter" button. The form is located in the top right corner of the page, with navigation links "Home Page Stores" visible above it.

Рис.11. Форма входа в профиль администратора сайта

В магазине Асте представлены различные товары, есть возможность их сортировки по цене, категории, в которой они представлены, а также реализован поиск товара по ключевым словам. Кроме этого, в интернет магазине Асте есть специальные предложения (offers), имеющие конечный срок действия. На рисунке 12 показан пользовательский интерфейс данного интернет магазина при работе с представленными продуктами.

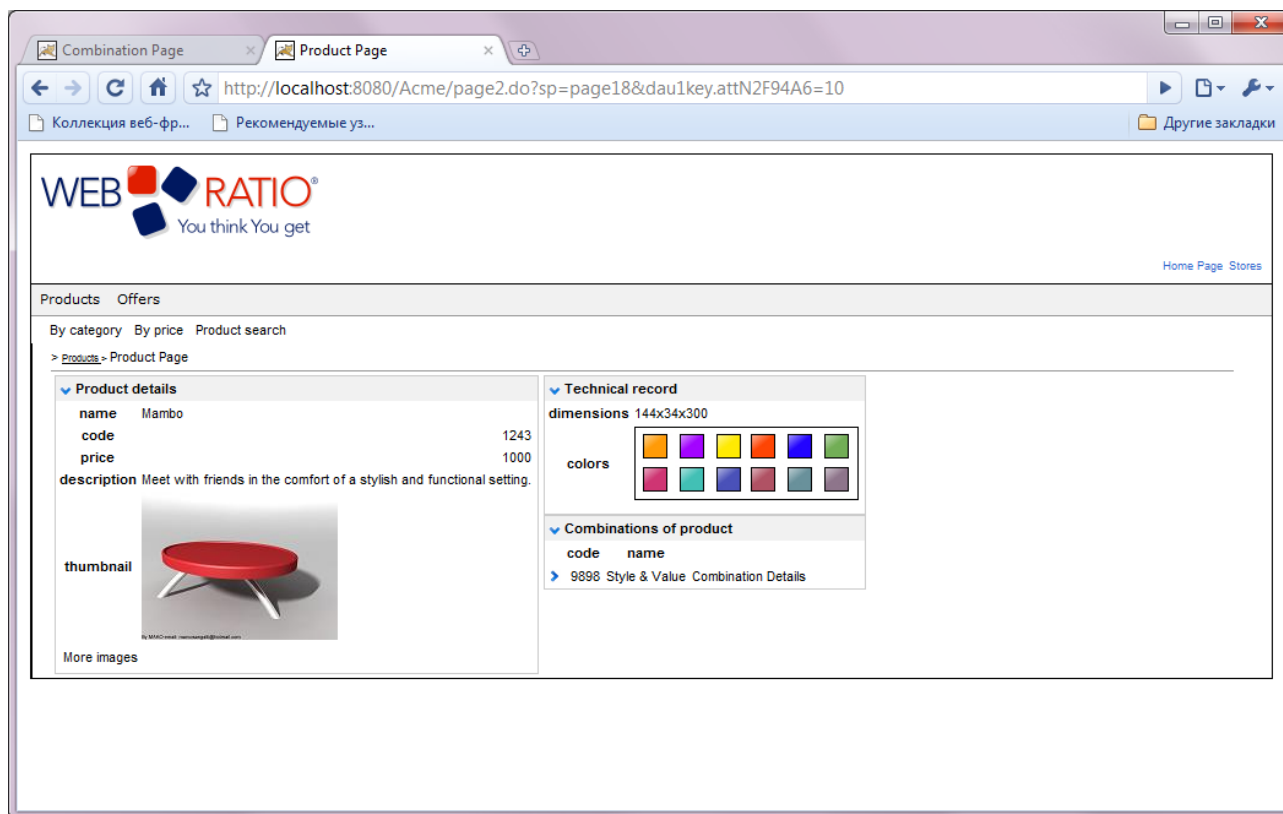


Рис.12. Пользовательский интерфейс Web-приложения Асте

Web-приложение Acme было создано в модельно-ориентированной среде разработки WebRatio. Его исходный код был сгенерирован автоматически по нескольким моделям, в том числе гипертекстовой модели, модели данных, и модели управления контентом. На рисунке 13 показана гипертекстовая модель данного Web-приложения.

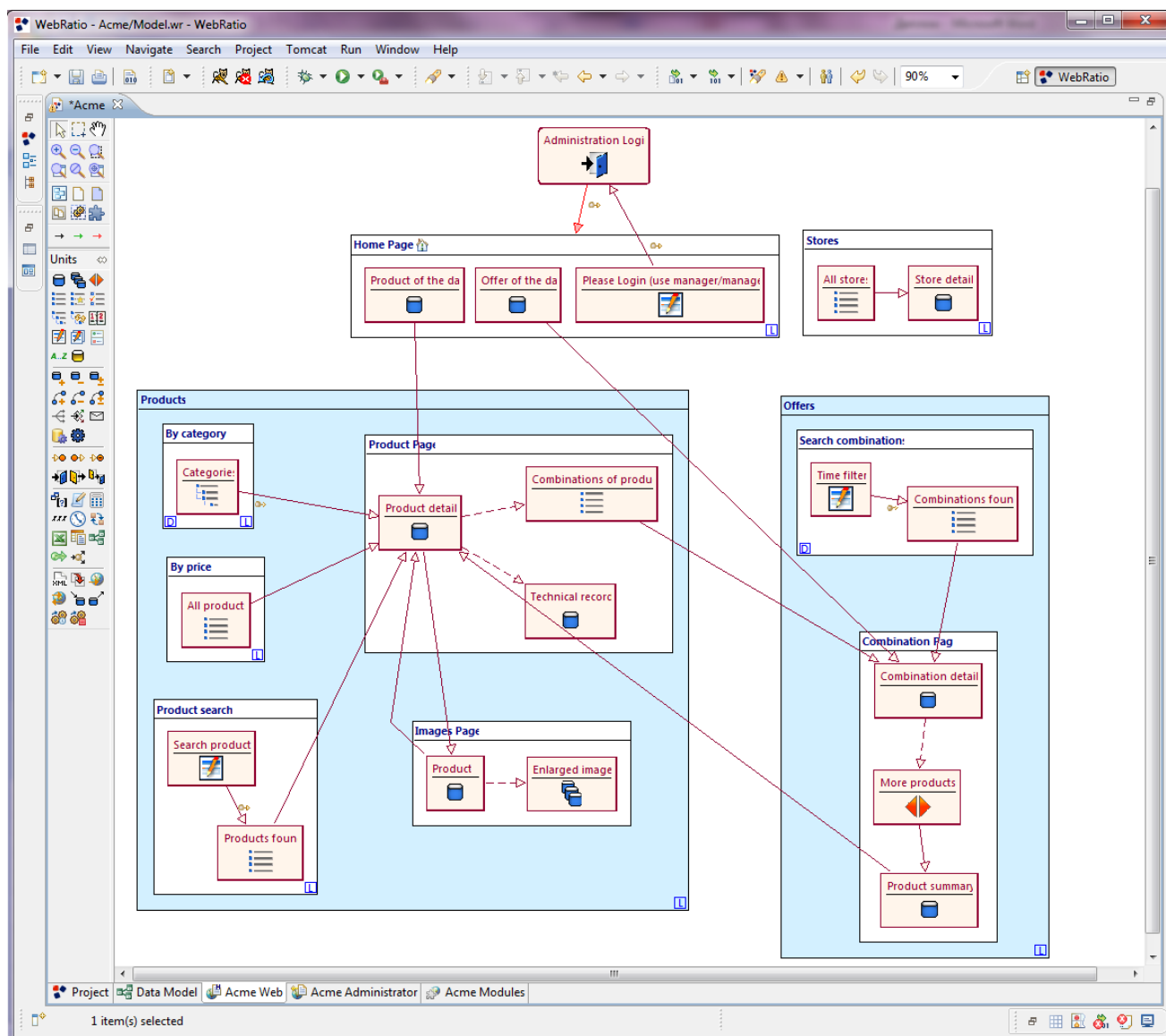


Рис.13 Гипертекстовая модель Web-приложения Acme

Для апробации разработанного в данной дипломной работе метода с помощью технологии DocLine была разработана пользовательская документация Web-приложения Acme. Один из разделов этой документации, описывающий работу с продуктами, изображен на рисунке 14.

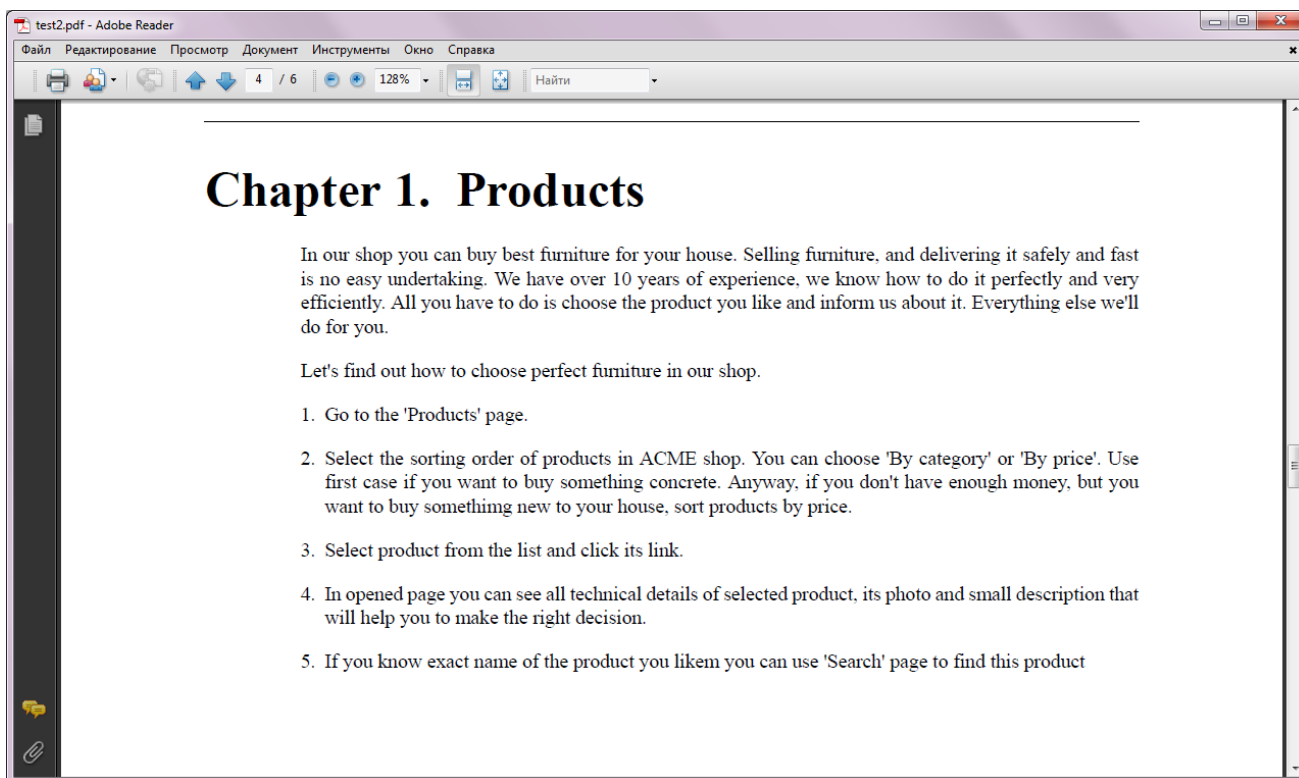


Рис.14. Пользовательская документация Web-приложения Асте

Показанный на рисунке 14 раздел документации в технологии DocLine представляется XML-элементом InfElement с идентификатором "products_chapter".

Далее в разработанном графическом редакторе WebMLDoc была осуществлена «привязка» элементов гипертекстовой модели к разделам документации (InfElement). На рисунке 15 показан графический редактор WebMLDoc с привязкой гипертекстовой модели к документации.

Желтые прямоугольники на диаграмме соответствует идентификаторам разделов документации, к которым привязаны соответствующие элементы интерфейса. Так, например, на рисунке показано, что область "Products" (Area "Products") связана с разделом документации с ID="products_chapter", а страницы "By category" и "By price" – с разделами "by_category_infelem" и "by_price_infelem".

Несмотря на то, что в данном примере элементы гипертекстовой модели привязаны не более чем к одному разделу пользовательской документации, эта

связь имеет тип «многие ко многим». То есть каждый элемент гипертекстовой модели можно связать с несколькими разделами документации, и, наоборот, к каждому разделу документации можно привязать несколько элементов гипертекстовой модели.

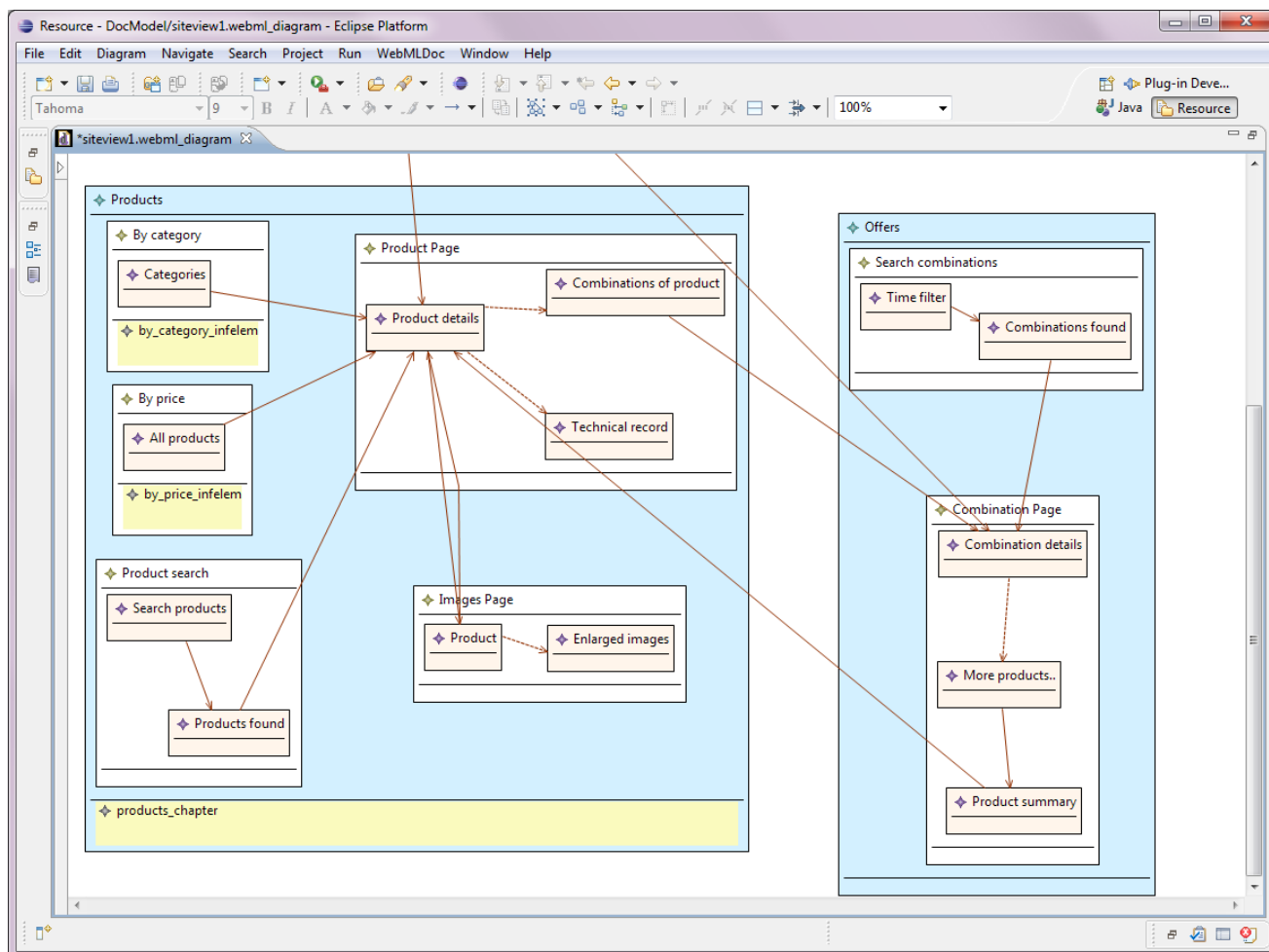


Рис.15. «Привязка» гипертекстовой модели к пользовательской документации

Предположим, администратор интернет магазина решил, что нужно убрать возможность сортировки товара по цене, а оставить только сортировку по категориям. Для этого он из гипертекстовой модели удалил страницу “By price”.

При этом изменится пользовательский интерфейс Web-приложения – исчезнет возможность выбора сортировки по цене.

Для того чтобы узнать, какие разделы документации нужно исправить после данных изменений, достаточно просто выбрать пункт “Show changes log” из меню “WebMLDoc” графического редактора WebMLDoc. При этом сначала

будут определены изменения гипертекстовой модели, а потом в зависимости от них и существующих связей с документацией предложены разделы документации, которые нужно исправить. Все изменения будут запротоколированы и показаны пользователю. В результате удаления сортировки по цене в журнале появится следующая информация (см. рисунок 16).

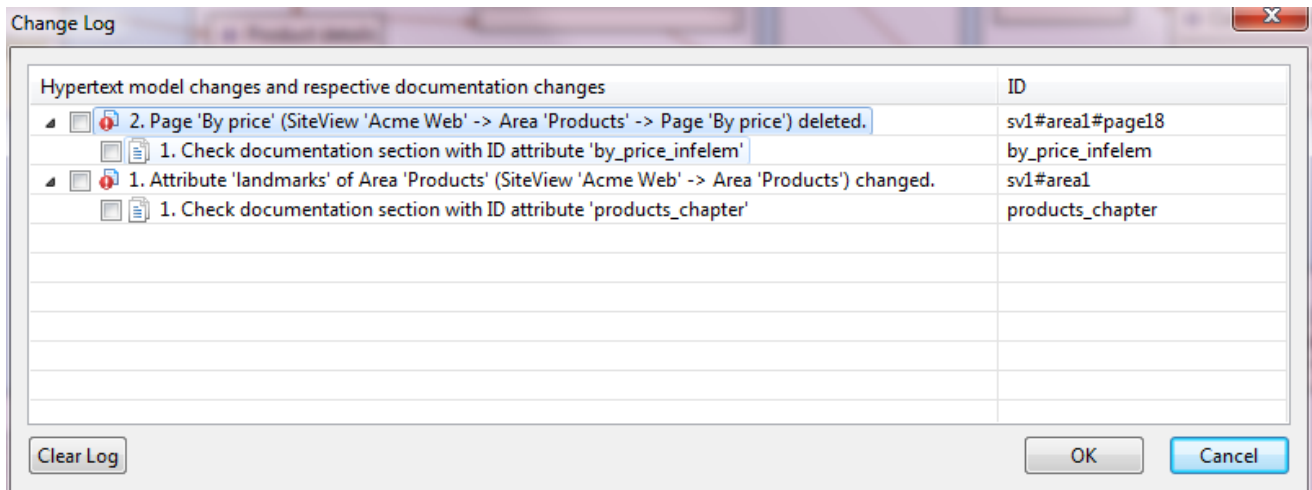


Рис.16. Журнал изменений

Лог имеет двухуровневую древовидную структуру. На первом уровне находятся изменения гипертекстовой модели, а их потомками на втором уровне являются исправления, которые необходимо внести в документацию. Рядом с каждым элементом стоит checkbox, с помощью которого можно отмечать уже исправленные разделы документации. Кроме этого, есть возможность очистки лога.

Как видно из рисунка, для исправления документации необходимо проверить корректность раздела “by_price_infelem”, поскольку соответствующий элемент гипертекстовой модели был удален, а также исправить раздел “products_chapter”.

После внесения изменений в документацию раздел Products будет выглядеть так, как показано на рисунке 17.

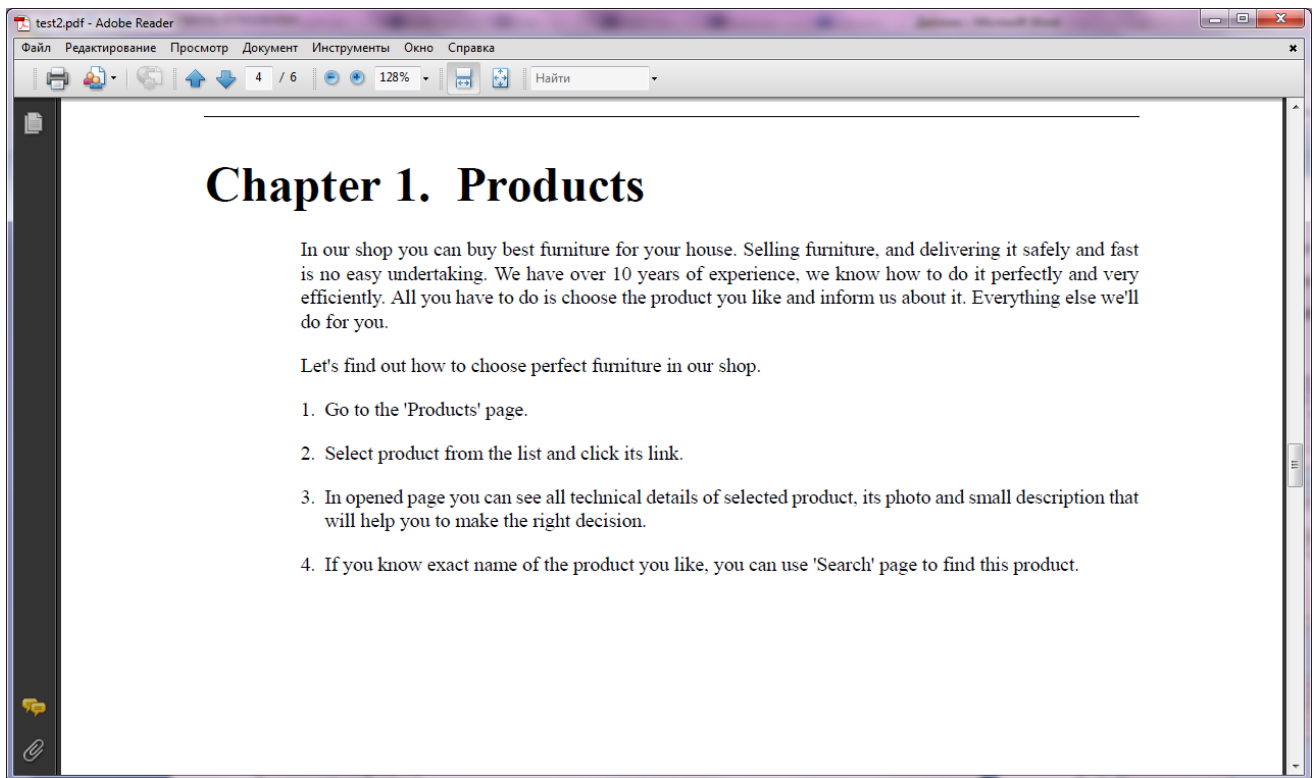


Рис.17. Исправленный раздел документации “Products”

Результаты

В рамках данной дипломной работы были достигнуты следующие основные результаты.

- Изучен язык WebML и CASE-пакет WebRatio.
- Выявлены зависимости между элементами пользовательского интерфейса программных продуктов и разделами пользовательской документации.
- Спроектирована модель связи пользовательского интерфейса Web-приложения, разрабатываемого в среде WebRatio, с пользовательской документацией в DocLine.
- На основе платформы Eclipse GMF разработан графический редактор, позволяющий устанавливать связь между элементами пользовательского интерфейса Web-приложения, разрабатываемого в среде WebRatio, и разделами пользовательской документации в DocLine.
- Разработаны и реализованы алгоритмы взаимодействия и синхронизации разработанной модели WebMLDoc с гипертекстовой моделью WebML, а также алгоритм определения разделов документации в DocLine, требующих проверки или корректировки при изменении гипертекстовой модели WebML (использованы некоммерческие Java-библиотеки JExamXML и Saxon 9 Home Edition).
- Разработанное решение опробовано на Web-приложении Asme, являющемся частью стандартной поставки WebRatio.

Список литературы

1. *К.Ю.Романовский, Д.В.Кознов*. Язык DRL для проектирования и разработки документации семейств программных продуктов. Вестник СПбГУ. Сер. 10, 2007. Вып. 4. С. 1–16.
2. *К.Ю.Романовский*. Метод разработки документации семейств программных продуктов. Системное программирование. Вып. 2, Изд-во С.-Петерб. ун-та, 2006. С. 191–218.
3. *А.Н.Иванов, Д.В.Кознов, М.Г.Тыжгеев*. Расширение языка WebML средствами моделирования интерфейсов полнофункциональных Web-приложений, интенсивно работающих с данными. Системное программирование. Вып. 4, Изд-во С.-Петерб. ун-та, 2009. С. 31–50.
4. *О.А.Евтифеева*. Язык декомпозиции гипертекстовых моделей. Дипломная работа, СПбГУ, 2008. С. 1–65.
5. WebML User Guide. WebRatio 4.3. 2006. 90 p.
6. *Rossi G., Pastor O., Schwabe D., et. al.* Web Engineering: Modeling and Implementing Web Applications. Springer, 2007. P. 464.
7. *Daniel Schwabe, Gustavo Rossi*. An Object Oriented Approach to Web-Based Application Design. Theory and Practice of Object Systems. Vol. 4, Issue 4 (October 1998). P. 207–225.
8. *UWA Consortium*. The UWA Approach to Modeling Ubiquitous Web Applications. IST Mobile & Wireless Telecommunications Summit, Greece, June, 2002. P. 1–6.
9. *O.M.F. De Troyer, C.J. Leune*. WSDM: A User Centered Design Method for Web Sites. Proceedings of the 7th International World Wide Web Conference, Elsevier, 1998. P. 85–94.
10. OMG Model Driven Architecture
<http://www.omg.org/mda/>
11. Eclipse Graphical Modeling Framework
<http://www.eclipse.org/modeling/gmf/>
12. XSL Transformations (XSLT)

<http://www.w3.org/TR/xslt>

13.Saxon 9 Java API

<http://www.saxonica.com/documentation/javadoc/index.html>

14.JExamXML Java API

<http://www.a7soft.com/jexamxml/index.html>

15.XOperator Java API

[http://www.living-
pages.de/de/projects/xop/doc/api/com/ercato/core/xml/XOperator.html](http://www.living-pages.de/de/projects/xop/doc/api/com/ercato/core/xml/XOperator.html)