

УДК 004.434:004.42

*Д. В. Кознов, М. Н. Смирнов, В. А. Дорохов, К. Ю. Романовский***ПОДХОД К АВТОМАТИЧЕСКОМУ ОТСЛЕЖИВАНИЮ
ИЗМЕНЕНИЙ В ПОЛЬЗОВАТЕЛЬСКОЙ ДОКУМЕНТАЦИИ
Web-ПРИЛОЖЕНИЙ^{*)}**

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 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.

Задача обеспечения связи программного кода приложения и его документации хорошо известна и востребована на практике [1–4]. Развитием данной задачи является подход к автоматическому определению тех мест в документации, которые нужно менять при локальных изменениях в коде. Такая задача естественным путем возникает из объединения процессов разработки программного продукта и документации и чрезвычайно актуальна при сопровождении и эволюции программного обеспечения (ПО). Однако в общем виде она не может быть решена полностью автоматически

Кознов Дмитрий Владимирович – кандидат физико-математических наук, доцент кафедры системного программирования математико-механического факультета Санкт-Петербургского государственного университета. Количество опубликованных работ: более 50. Научные направления: визуальное моделирование (UML-подобные языки и программные средства), разработка технической документации, электронное образование. Сайт: <http://www.math.spbu.ru/user/dkoznov>. E-mail: dkoznov@gmail.com.

Смирнов Михаил Николаевич – старший преподаватель кафедры системного программирования математико-механического факультета Санкт-Петербургского государственного университета и одновременно руководитель научно-технического центра по разработке алгоритмоёмких проектов компании ЗАО «ЛАНИТ-ТЕРКОМ». Количество опубликованных работ: 5. Научные направления: технологии разработки электронной документации, информационные системы. E-mail: smnsmn1979@gmail.com.

Дорохов Вадим Александрович – студент V курса математико-механического факультета Санкт-Петербургского государственного университета. Научные направления: технологии разработки электронной документации, информационные системы для комплексной автоматизации бизнеса. E-mail: vadim.dorokhov@gmail.com.

Романовский Константин Юрьевич – кандидат физико-математических наук, старший преподаватель кафедры системного программирования математико-механического факультета Санкт-Петербургского государственного университета и одновременно руководитель научно-технического центра «Системы управления аналитическим оборудованием» компании ЗАО «ЛАНИТ-ТЕРКОМ». Количество опубликованных работ: 15. Научные направления: технологии разработки электронной документации, управление разработкой ПО. E-mail: kromanovsky@yandex.ru.

^{*)} Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 08-01-00716-а).

© Д. В. Кознов, М. Н. Смирнов, В. А. Дорохов, К. Ю. Романовский, 2011

из-за большого семантического разрыва между программным кодом и документацией [3]. Приемлемое для практики решение может быть предложено при уменьшении степени общности задачи и введении дополнительных ограничений.

Один из наиболее очевидных способов сближения кода и документа – использование модельно-ориентированного подхода (Model-Based Development), предполагающего генерацию программного кода приложения (полностью или существенно) по моделям (например, по UML-моделям) и дальнейшую эволюцию приложения посредством изменения моделей и регенерацией кода. В таком случае появляется возможность «привязать» документацию к моделям, а не к самому исходному коду, что значительно удобнее, так как модели содержат более высокоуровневую информацию о приложении. Существует ряд подходов, реализующих эту идею [2, 5, 6], но среди них не удается обнаружить ни одной работы, описывающей реализацию подобной задачи для пользовательской документации Web-приложений.

Web-приложения являются на сегодняшний день одним из самых бурно развивающихся видов ПО. Существует огромное разнообразие Web-приложений — от простейших сайтов-визиток до огромных корпоративных порталов и социальных сетей. В ряде областей Web-приложения уже полностью заменяют настольные (Desktop), и для их моделирования, наряду с UML, создан целый набор подходов – OOHDM [7], UWA [8], WSDM [9], UWE [10], OOWS [11] и др. Среди них заметно выделяется и оказывается по сути стандартом де-факто язык WebML (Web Modeling Language) [12], а также его реализация – CASE-пакет WebRatio [13], который поддерживает автоматическую генерацию кода и интегрирован со средой разработки Eclipse.

В настоящей статье предлагается подход WebMLDoc к решению задачи автоматизированного сопровождения пользовательской документации в процессе эволюции Web-приложений, основанный на модельно-ориентированном подходе. В качестве соответствующей модели выбрана гипертекстовая модель языка WebML, так как она, фактически работает с элементами пользовательского интерфейса и поэтому хорошо отображается на пользовательскую документацию. В то же время эта модель является основной при генерации кода. В качестве средства спецификации документации используется XML-язык DRL [14]. В статье представлены пилотная реализация подхода в среде Eclipse, связывающая между собой два продукта – WebRatio (реализация языка WebML) и DocLine [15] (реализация языка DRL), а также пример, иллюстрирующий работу метода.

1. Обзор.

1.1. Язык WebML и технология WebRatio. Язык WebML [12] позволяет разрабатывать Web-приложения с применением модельно-ориентированного подхода. Для создания отдельного приложения необходимо спроектировать несколько его моделей, по которым будет сгенерирован автоматически исходный код приложения. Каждая из этих моделей описывает характеристики приложения с определенной точки зрения.

Модель данных (Data Model) позволяет задать структуру базы данных приложения и является вариантом классической модели «сущность-связь», используемой во многих подходах и программных средствах.

Гипертекстовая модель (Hypertext Model) служит для описания схемы интерфейса Web-приложения. Самым верхним элементом является *профиль сайта* (Siteviews) – например, профиль администратора, неавторизованного пользователя, авторизованного пользователя. Приложение может состоять из нескольких профилей, а те, в свою очередь, – из областей и страниц. *Область* (Area) – это группа страниц, объединенных общим назначением (например, работа с товарами, техническая поддержка и т. д.).

Области могут наследоваться. Так же как и страницы, они могут быть ориентировочными (Landmark) – ссылки на такие области будут присутствовать на всех страницах профиля сайта. Область включает в себя набор *страниц* (Pages), которые соответствуют обычным страницам Web-приложения, а также транзакций и операций. Страницы, в свою очередь, состоят из *контент-модулей* (Content units), являющихся атомарными элементами гипертекстовой модели и использующимися для визуального представления информации, описанной в модели данных. В WebML существуют разные типы контент-модулей (DataUnit, IndexUnit, ScrollUnit, EntryUnit и др.), соответствующие отдельным функциональным блокам пользовательского интерфейса. Стоит отметить, что атрибуты внешнего вида интерфейса (т. е. цвета, конкретный вид отображаемых элементов управления, их размеры и пр.) задаются с помощью стилей и свойств непосредственно в пакете WebRatio и в гипертекстовую модель WebML не попадают. Навигация между контент-модулями и страницами описывается с помощью связей (Links). Связи могут передавать данные и/или поток управления в приложении. В отдельный вид модельных сущностей выделяют транспортные связи, которые применяются для передачи только информации, по таким связям нельзя осуществить переход (передачу управления).

Модель управления контентом (Content Management Model) расширяет гипертекстовую модель дополнительными конструкциями – операциями и транзакциями. *Операции* (Operation units) используются для обозначения выполнения какого-либо процесса или его отдельного шага при переходе по связи. Это может быть обработка данных (добавить, удалить, модифицировать некоторый фрагмент базы данных), вызов внешнего сервиса и т. д. Для реализации многовариантных переходов по результатам выполнения операции используются так называемые OKLink- и KOLink-связи, соответствующие успешному и неуспешному завершению операции. *Транзакция* (Transaction) объединяет в себя набор операций, который обладает всеми обычными свойствами транзакций (ACID – Atomicity, Consistency, Isolation, Durability).

Модель запросов (Derivation Model) расширяет модель данных вычислимыми конструкциями и является аналогом представлений (View) в языке SQL.

Язык WebML реализован в пакете WebRatio [13], который по моделям WebML генерирует исходный код приложения с применением стандартов JSP, HTML и XHTML.

Основным достоинством языка WebML для нас является то, что весь целевой код приложения генерируется автоматически по моделям и не требует «ручных» доработок. То есть модели полностью отражают функциональность программного продукта, и работа с моделями, а не с исходным кодом, не приводит к потерям информации.

Важность гипертекстовой модели (далее будем объединять ее с моделью управлением контентом) заключается в том, что она является основной при генерации программного кода Web-приложения.

1.2. Язык DRL и технология DocLine. Для проектирования и разработки документации с акцентом на повторное использование на кафедре системного программирования математико-механического факультета СПбГУ был разработан метод DocLine [15], включающий в себя язык и процесс разработки документации DRL [14], 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.

Внутренне представление документа на языке DRL реализовано на основе XML формата. Все данные хранятся в XML файле. Язык DRL поддерживает модульность документов, структура документа также хранится в XML файле. В совокупности это позволяет легко разбирать структуру документа и извлекать из нее разделы документа.

Ключевой конструкцией языка DRL/PR является *информационный элемент* (InfElement), который применяется для задания разделов документации. Приведем фрагмент документации на DRL, состоящей из нескольких разделов, представленных информационными элементами:

```
<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>
```

1.3. Задача обеспечения трассировки. В работе [16] задача обеспечения трассировки (Traceability) определяется как «возможность отследить (определить и измерить) все шаги, которые привели к определенной точке в процессе, состоящем из цепочки взаимосвязанных событий». Таким образом, решение задачи трассировки позволяет достичь следующих целей:

- провести быстрый анализ процесса, чтобы гарантировать (убедиться), что все, что Вы сделали, есть все то, о чем Вы договорились (хотели) сделать и только то, о чем Вы договорились;
- оценить влияние изменений на стоимость, расписание и технические аспекты проекта;
- ясно понимать, как та или иная отдельная деятельность способствует развитию проекта;
- точно измерить прогресс выполнения работ, отвечая на вопросы, подобные следующим: «существует ли проектный артефакт для каждого требования?», «существует ли сценарий теста для каждого требования?»;
- сравнить прибыль и затраты.

Можно встретить самые разные предметные области, где актуальна задача трассировки — анализ и разработка бизнес-процессов, лабораторные исследования, медицина, промышленное производство, разработка ПО и пр. Например, при анализе и разработке бизнес-процессов под трассировкой понимают обеспечение контроля всевозможных данных в этих процессах с целью определения увеличения/уменьшения эффективности, потерь, простоев и т. д. [17].

В области разработки ПО самым известным вариантом задачи трассировки является отслеживание связей требований с различными артефактами проекта (Requirement traceability) с целью обеспечить в любой момент процесса разработки переход от описания требования к его реализации и тестам [18].

1.4. Трассировка моделей. Еще один вариант задачи трассировки – отслеживание связей моделей ПО (например, UML-моделей) с кодом, требованиями и т. д. Рассмотрим несколько подходов этого направления.

В работе [5] представлено решение задачи трассировки для связи требований с моделями Web-приложения в контексте метода OOWS [11]. По моделям, задающим требования к Web-приложениям, генерируются модели навигации (Navigation Models), описывающие структуру страниц Web-приложения и сценарии переходов между ними. Модель требований представляет собой иерархию задач, которые соответствуют выполнению определенного сценария, например заказ продукта делится на его покупку и поиск. Очевидно, что данная модель представляет собой не дерево, а граф, так как разрешены переходы между задачами разных поддеревьев. Связывание двух этих моделей происходит на основе совпадения атрибутов контента, а также страниц и задач. В результате осуществляется связь между требованиями и их реализацией в более низкоуровневой модели навигации.

В работе [19] рассматривается подход, связывающий бизнес-правила, описывающие требования, с пользовательским интерфейсом ПО. При этом выполняется так называемый анализ задач (Task Analysis), готовящий для бизнес-процесса набор задач, которые необходимо выполнить для его осуществления. Далее строится иерархия этих задач в виде модели задач. Она строится сверху вниз: задача разбивается на подзадачи и т. д., вплоть до мельчайших атомарных действий. Затем происходит выяснение того, каким элементом интерфейса должен пользоваться пользователь в системе, чтобы выполнить ту или иную задачу. Так производится связывание интерфейса с задачами и через них с бизнес-процессом. Для описания интерфейса используется специальный XML-язык. Однако само отслеживание созданных таким образом связей не автоматизировано.

В [6] представлен подход, позволяющий задавать требования в виде UML-модели классов, автоматически преобразовывая их затем в специальную XML-модель. Утверждается, что далее посредством разработанного набора программных средств можно легко получать доступ к атрибутам требований с тем, чтобы определять их изменения, осуществлять «привязку» к атрибутам других моделей и т. д.

В работе [20] приведено решение задачи отслеживания изменений UML-диаграмм при изменениях документации на примере диаграмм классов UML 2.0. Авторы строят детальный разбор структуры диаграммы классов UML вплоть до мельчайших объектов на диаграмме. Для каждого уровня объектов строится соответствующая иерархия изменений, включающая также степень влияния изменений (как глубоко может затронуть модель данное изменение). И далее, при сравнении текущей и новой измененной моделей производится построение дерева изменений, по которому впоследствии можно определять границы изменений. Авторы представляют также программное средство, реализующее данный подход.

1.5. Отслеживание изменений документации при эволюции ПО. В случае отслеживания связей между кодом приложений и документацией можно выделить следующие варианты задачи трассировки: связь документа с требованиями и программного кода [18], связь кода с документом, описывающим архитектуру приложения [1–4]. Остановимся на этих подходах подробнее.

Работа [18] посвящена обзору методов information retrieval, причем они рассматриваются на примере установления связи между программным кодом и документом с требованиями. Из кода извлекаются имена классов, функций, аргументов функций, и их вхождения ищутся в документации (рассматриваются документы с детальными требованиями). На основе найденных вхождений делаются выводы о связях того или иного

раздела документа с различными сущностями в программном коде. Однако в рамках таких подходов не обсуждается то, как эти связи используются и поддерживаются в целостном состоянии.

Достаточно много работ посвящено задаче автоматизации построения связи между имеющимся legacy-кодом и его документацией для упрощения последующего анализа legacy-кода. В частности, работа [1] посвящена тому, в какую часть имеющейся HTML-документации нужно смотреть, просматривая ту или иную часть COBOL-программы. «Привязка» документации к коду строится автоматически, на основе общей базы знаний, в два этапа. На первом этапе производится синтаксический разбор COBOL-кода и создается дерево синтаксического разбора (AST, Abstract Syntax Tree), узлы которого далее, посредством специального анализа, попадают в базу знаний. На втором этапе из HTML-документов с описанием приложения извлекаются ключевые фразы и прочие атрибуты и также помещаются в базу знаний. Далее анализируются совпадения атрибутов, извлеченных из кода, и атрибутов, извлеченных из документации в базе знаний, и на основании найденных совпадений создается «привязка». В работе [3] рассматривается аналогичный подход с использованием information retrieval метода Latent Semantic Indexing (LSI) для распознавания текстов.

Подход, описанный в [2], предназначен для автоматического построения документов с описаниями кода на основе анализа программного кода. Документация строится путем возвратной инженерии (Reverse Engineering) кода приложения в среде IBM Rational Rose. Автоматически строятся UML-пакеты для каждого фрагмента приложения. Строится диаграмма классов этого фрагмента и создается документация на основе комментариев в коде. Итоговый документ генерируется из данных пакетов.

В работе [4] представлен инструмент NoWeb для повторного использования текстов с описанием программного кода в LaTeX на основе повторного применения самого кода. Документация к классам, методам и т. д. оформляется в блоках, которые можно повторно использовать так же, как повторно использовать конструкции в программном коде, в частности, с помощью наследования. Однако пользовательская документация не является «кодоподобной», она, скорее, «интерфейсоподобная», поэтому такая «привязка» в нашем случае не подходит.

Нам не удалось найти работ, посвященных трассировке программного кода (моделей) и пользовательской документации.

2. Постановка задачи. Пользовательская документация часто создается параллельно с разработкой самого приложения, кроме того, изменения исходного кода приложения (добавление новых функциональных возможностей, изменение пользовательского интерфейса и пр.) требуют соответствующих модификаций и пользовательской документации. Например, возможна ситуация, когда изменение имени продукта или его версии может потребовать внесения десятков исправлений в документации продукта.

Наша цель – разработать подход, реализующий трассировку изменений программного продукта и его пользовательской документации. При этом мы хотим автоматически определять те места в документации, которые нужно изменить при «точечных» изменениях ПО. Сами изменения документации предполагается выполнять «вручную», поскольку полностью автоматизировать процесс эволюции документации по изменениям в коде, очевидно, невозможно.

Однако, если мы хотим связывать код приложения не с детальной документацией, описывающей тот же код, а с пользовательской документацией, которая имеет иной уровень абстракции, то сталкиваемся с проблемой наличия подходящих для трассировки абстракций в программном коде. Функциональность приложения, описанная

в документации как единое целое, в программном коде может не быть локализована в коде (и, как правило, так и есть!), а рассосредоточена и при этом не оформлена в виде отдельных методов, классов и других структурных конструкций используемого языка программирования. Связывать с каким-либо разделом документации такие отдельные фрагменты очень неудобно, так как они могут разрастаться, мигрировать (полностью или частично) из одной части программы в другую, распадаться и т. д. Кроме того, один и тот же программный код может участвовать в реализации различной функциональности.

3. Метод WebMLDoc.

3.1. О структуре пользовательской документации. Можно рассматривать два основных принципа организации пользовательской документации ПО [21, 22] – описание сценариев работы приложения и описание пользовательского интерфейса.

Большое количество разделов в пользовательской документации ПО посвящено сценариям работы с приложением для выполнения определенных функций, например открыть файл, изменить пароль. Каждый такой сценария является цепочкой пользовательских действий с элементами интерфейса, что обеспечивает выполнение системой той функции, которая ему нужна. На рис. 1 представлено описание сценария создания новой учетной записи (Account) в документации для почтового клиента приложения «The Bat!» [23].

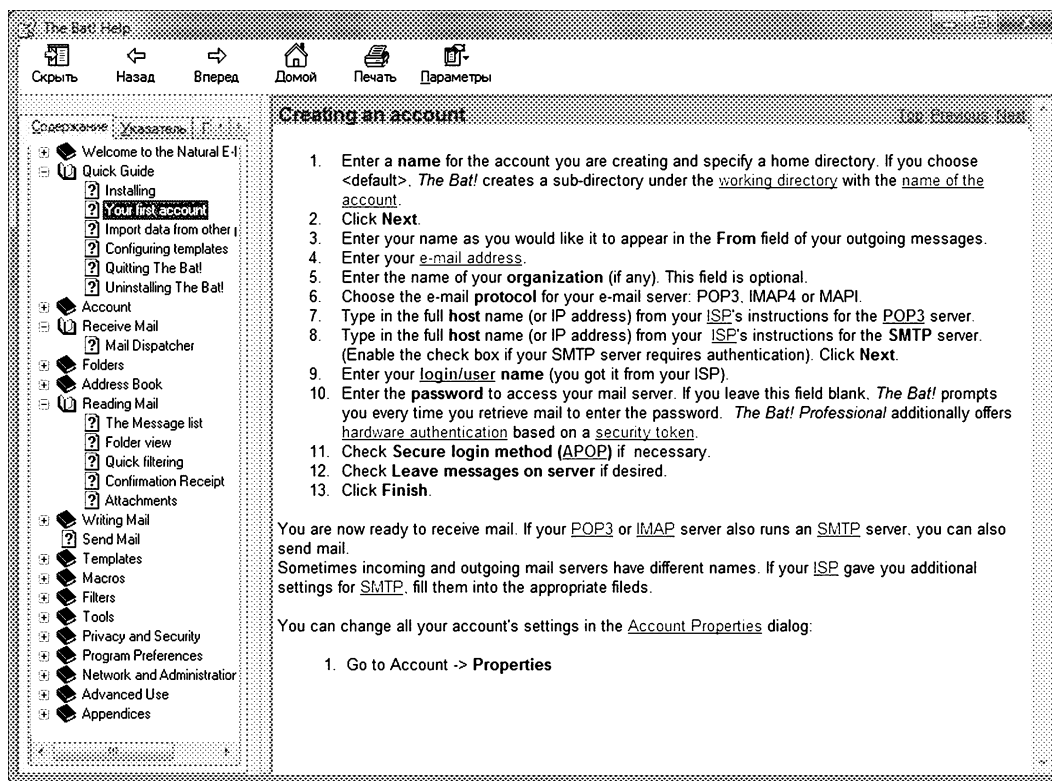


Рис. 1. Пример документации, описывающей сценарии работы с приложением

Видно, что в данном фрагменте документации описываются многочисленные диалоговые окна программы «TheBat!», связанные с созданием учетной записи. Часто сюда накладываются еще и различные варианты доступа к команде пользовательского интерфейса, участвующего в сценарии, – из пункта меню, из тулбара, с помощью комбинации «горячи» клавиш и т. д. И это все – различные элементы пользовательского интерфейса, связанные с разделом документации, описывающим данный сценарий. Более того, справки к линейкам продуктов содержат групповые описания сходных функций сразу для нескольких продуктов, задействуя еще больше окон и элементов управления пользовательского интерфейса. Например, в документации Microsoft Office команда «Open» сразу описывается для нескольких продуктов – Microsoft Word, Excel, Access и PowerPoint. При этом у данной команды есть особенности для каждого из продуктов. Таким образом, количество элементов пользовательского интерфейса, связанных с данным топиком, возрастает еще больше.

Еще одним видом пользовательской документации является непосредственное описание пользовательского интерфейса приложения – всех рабочих областей, всех окон, элементов меню и т. д. Каждый элемент интерфейса рассматривается подробно, полностью, включая все входящие в него другие элементы. На рис. 2 приведен пример пользовательской документации антивирусной программы ESET NOD32 [24], построенной по этому принципу.

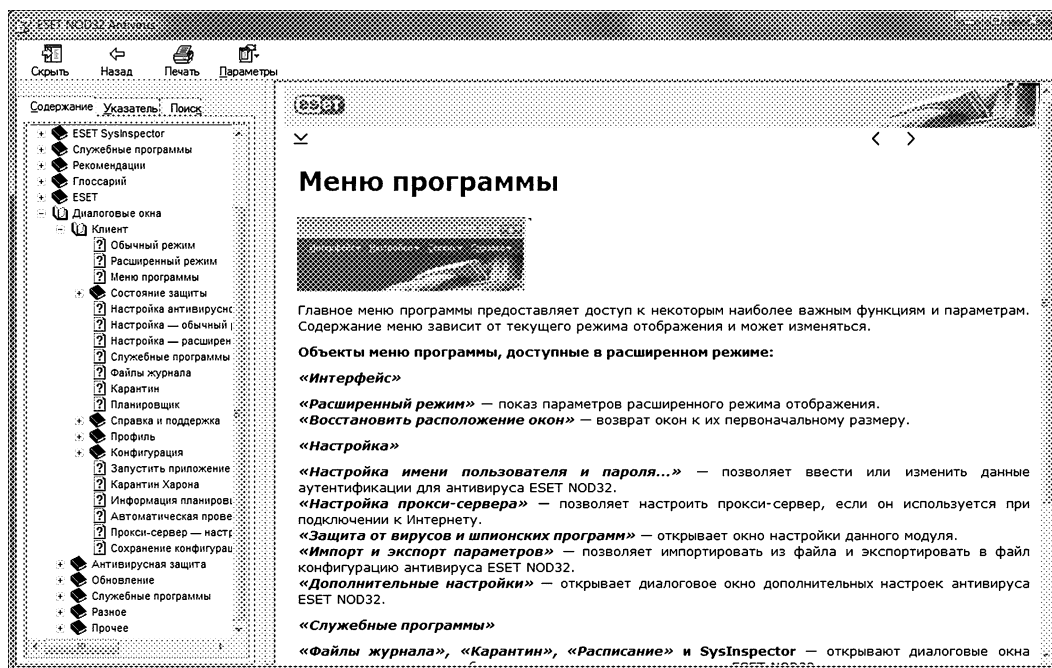


Рис. 2. Пример документации, описывающей непосредственно пользовательский интерфейс приложения

Документация для конкретного программного продукта может сочетать в себе оба способа. Но в любом случае большая часть пользовательской документации так или иначе описывает пользовательский интерфейс, так как пользователю нужно уметь работать с продуктом, а сделать это можно исключительно через интерфейс. Конечно,

остаются еще и другая информация – различного вида введения, описание общих принципов работы приложения, а также программных интерфейсов, языков программирования и пр., но это гораздо меньше.

3.2. Связываем документацию с моделями. Мы предлагаем связывать документацию не с кодом приложения, а с его моделью. С точки зрения целей нашего исследования, это оказывается эффективным в том случае, если по модели генерируется исходный код приложения, который не меняется потом «руками». То есть визуальное моделирование используется в режиме языка программирования, согласно классификации Мартина Фаулера [25]. Таким образом, все изменения, которые нужно вносить в приложение, вносятся в модели, а затем автоматически переносятся в исходный код программы.

Однако возникает задача найти наиболее подходящую для нашего случая модель, которая будет максимально близка к абстракциям пользовательской документации. Коль скоро определено, что документация сильно связана с пользовательским интерфейсом, то нам нужны подходящие модели интерфейсов программных продуктов.

Во-первых, выбираем Web-приложения интенсивной обработки данных (Data Intensive Systems) – всевозможные сайты и системы, отображающие через Web содержимое некоторой базы данных [12]. Для нас эти приложения интересны тем, что в основном они определяются пользовательским интерфейсом, так как их бизнес-логика не очень сложна *) . Число таких приложений огромно, количество их пользователей также велико, и таким образом существует потребность в понятных Web-справках и других видах документации для подобных для них. Эти приложения также активно эволюционируют, так что стоит задача обновления их пользовательской документации.

Во-вторых, выбираем гипертекстовую модель языка WebML, которая позволяет задавать схему интерфейса и одновременно – навигацию между его элементами, т. е. содержит информацию о сценариях. Эта модель достаточно широко используется на практике при генерации программного кода для Web-приложений, в отличие от других моделей пользовательских интерфейсов, предназначенных для desktop-приложений [26, 27].

В качестве средства разработки документации предлагается применить подход DocLine [15], имеющий формальное внутреннее представление документации в виде XML-языка DRL, позволяющего разбивать документацию на модули, что облегчает организацию связи документации с кодом. Стоит отметить, что DRL интересен нам еще и потому, что ближайшей перспективой WebMLDoc является создание на его основе средств управления повторным использованием документации в семействах Web-приложений. В отличие от других XML-языков разработки документации DRL содержит развитые средства поддержки повторного использования разделов документов.

3.3. Сценарий использования подхода. Схема работы нашего подхода выглядит так:

1. Производится «привязка» конструкций гипертекстовой модели приложения на языке WebML к разделам пользовательской документации на языке DRL. Мы не определяем, как именно была создана документация на DRL, рассматривая ее существование просто как объективный факт.

*) Такие приложения, конечно же, определяются еще и схемой базы данных, но ее спецификация существенно более компактна, чем спецификация пользовательского интерфейса, и схема в значительно меньшей степени подвержена изменчивости.

2. После внесения изменений в гипертекстовую модель Web-приложения мы определяем разницу новой и прежней моделей.
3. После этого, используя «привязку» гипертекстовой модели к набору топики документации, мы устанавливаем, какие топики нужно изменить – меняются те топики, в которых данный элемент пользовательского интерфейса задействован.
4. Технический писатель меняет выданный ему список топики документации.

3.4. Мета-модель. Для того чтобы осуществить привязку гипертекстовых моделей к разделам документации на языке DRL, мы предлагаем промежуточный язык, который называем так же, как и весь наш подход, – WebMLDoc. Он является упрощенным подмножеством гипертекстовой модели WebML с включенными в него средствами группировки элементов интерфейса в сценарии и связывания последних с топики в DRL. Этот язык понадобился для реализации функциональности по связыванию двух спецификаций – в WebML и DRL, – а также из-за того, что нам пришлось создавать отдельный графический редактор из-за проблем с интеграцией с продуктом WebRatio.

На рис. 3 представлена мета-модель языка WebMLDoc. Проект в WebMLDoc (класс WebMLDocProject) состоит из сущностей (класс EntityBox), представляющих соответствующие элементы гипертекстовой модели WebML, и их связей (класс ConnectionLine). Сущности могут быть вложенными и бывают либо верхнеуровневыми (потомки класса HighLevelWebMLEntity), либо «листовым» (потомки класса LeafWebMLEntity). Листовые могут входить в сценарии (класс Scenario). Мы связываем разделы пользовательской документации (информационные элементы языка DRL, представленные в мета-модели классом DRLInfElement) с набором Web-страниц нашего приложения, если данные разделы посвящены этим страницам, либо с последовательностью контент-модулей и переходов, реализующих некоторый пользовательский сценарий (например, открытие файла), который описывается в этом разделе документации. Обе связи являются связями «многие-ко-многим».

4. Реализация.

4.1. Инструментарий. Для реализации предложенного подхода был разработан прототип программного продукта WebMLDoc. Он является графическим редактором для «привязки» гипертекстовой модели WebML к разделам документации в DRL. WebMLDoc связывает два пакета – WebRatio и DocLine. Отдельный редактор понадобился потому, что продукт WebRatio не имеет открытых средств для создания надстроек с целью «раскрашивать» гипертекстовую модель прямо в редакторе гипертекстовой модели в WebMLDoc.

На рис. 4 представлено окно редактора WebMLDoc (в качестве примера использовалась WebML модель Acme из официальной поставки продукта WebRatio). На рис. 5 показано, как выглядит та же модель в WebRatio. Можно заметить сходство данных спецификаций. Мы старались максимально достичь этого, делая внешний вид нашего редактора как можно более похожим на WebRatio, а также максимально сохраняя визуальные атрибуты модели (прежде всего координаты графических элементов). Это сделано для того, чтобы наш редактор был максимально «незаметным» для пользователей WebRatio.

4.2. Особенности реализации. Графический редактор WebMLDoc разрабатывался в среде Eclipse с использованием технологии GMF (Graphical Modeling Framework) [28], предназначенной для создания графических редакторов. Этот выбор обусловлен тем, что и пакет WebRatio разработан с помощью этих же средств.

Генерация XML-представления модели WebMLDoc выполняется с помощью специально созданной XSL-трансформации (eXtensible Stylesheet Language Transformations),

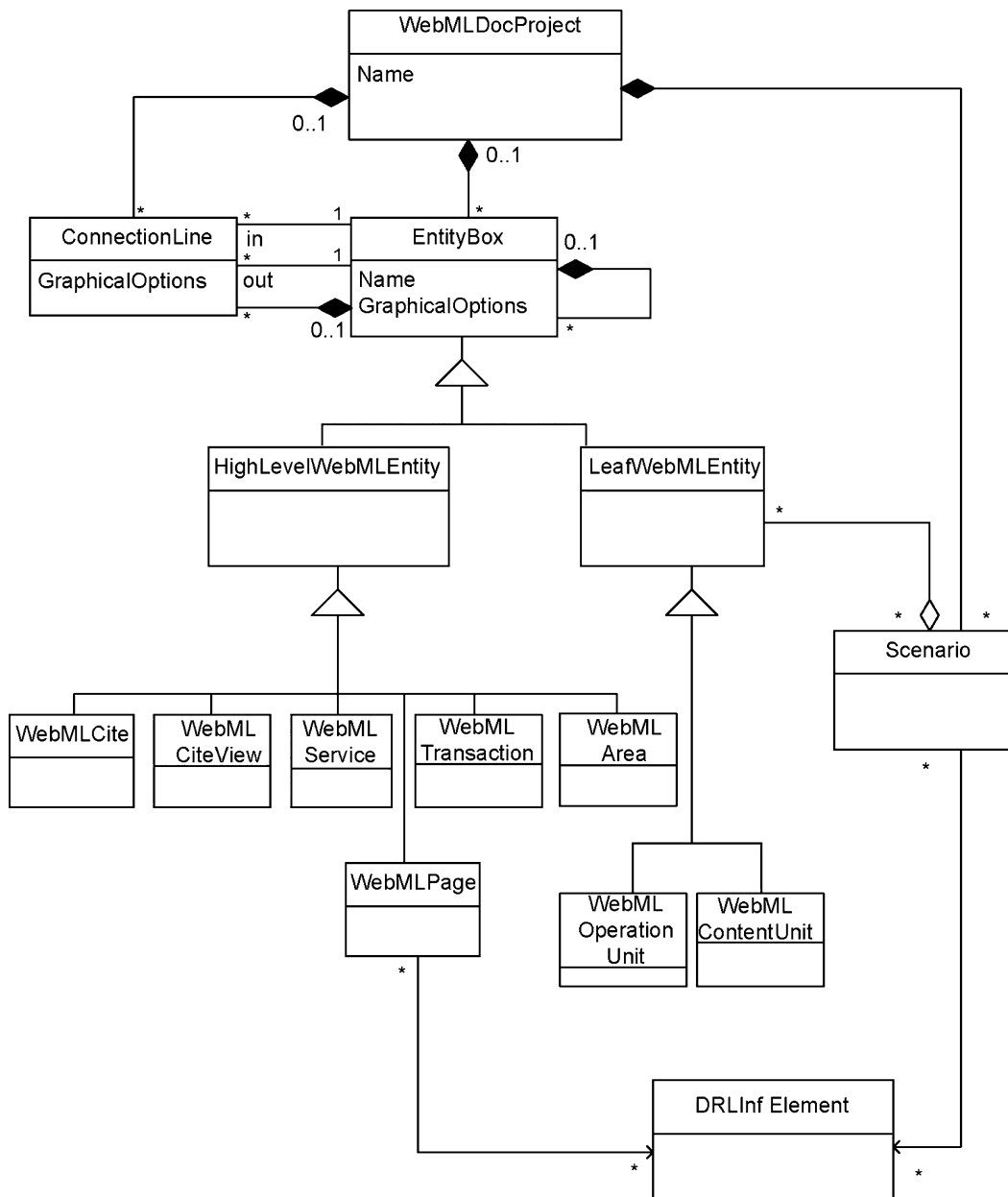


Рис. 3. Мета модель языка WebMLDoc

обрабатываемой процессором Saxon [29].

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.

При каждом открытии графического редактора WebMLDoc его модель генерируется заново по гипертекстовой модели WebRatio. В сгенерированной с помощью

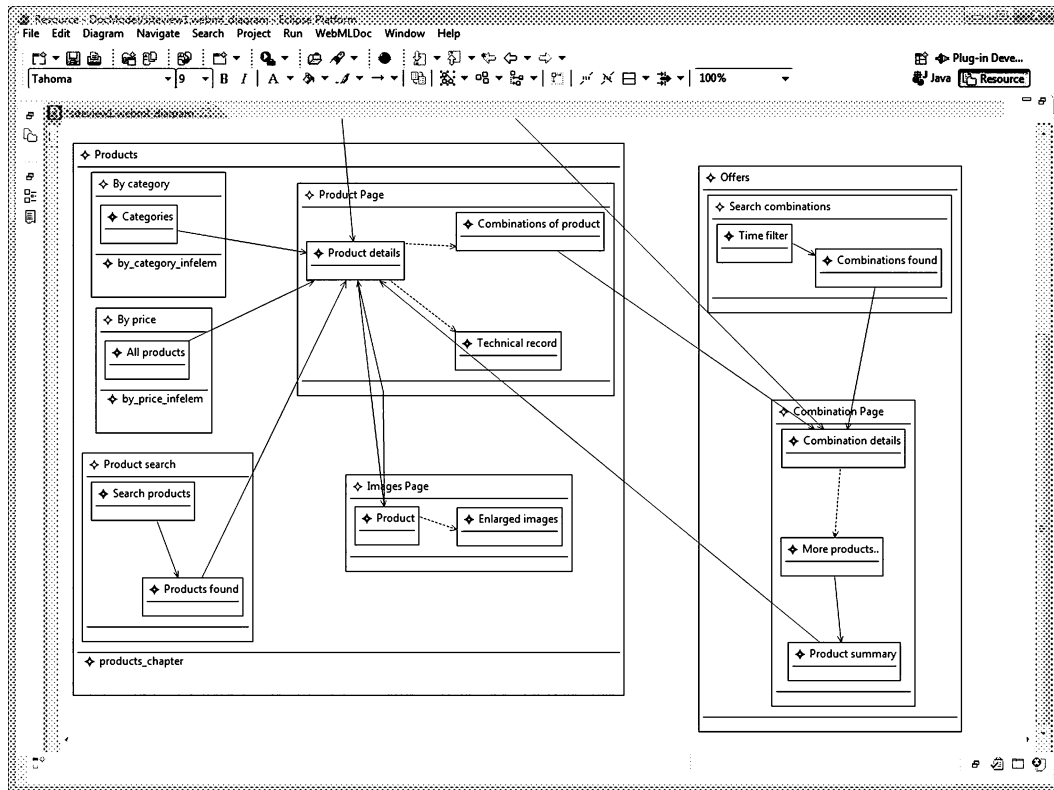


Рис. 4. Модель WebMLDoc для приложения Асме

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.

Для определения внесенных в гипертекстовую модель изменений (XML Diff) была применена библиотека JExamXML [30]. Это бесплатная библиотека, позволяющая вычислять разницу двух XML-файлов и работающая через интерфейс командной строки или из Java-приложения с использованием специального программного интерфейса.

5. Пример. В качестве примера работы метода рассмотрим работу с Web-приложением Асме, являющемся частью стандартной поставки WebRatio.

Асме – 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.

Магазин Асме позволяет просматривать различные товары, сортировать их по цене, по категории, в которой они представлены, а также осуществлять поиск товара по ключевым словам. Кроме этого, в нем есть специальные предложения (Offers), имеющие конечный срок действия.

Исходный код магазина Асме был сгенерирован автоматически по нескольким моделям, в том числе гипертекстовой модели, модели данных, и модели управления контентом. На рис. 5 показана гипертекстовая модель данного Web-приложения.

В рамках работы с помощью технологии DocLine была разработана пользовательская документация Web-приложения Асме.

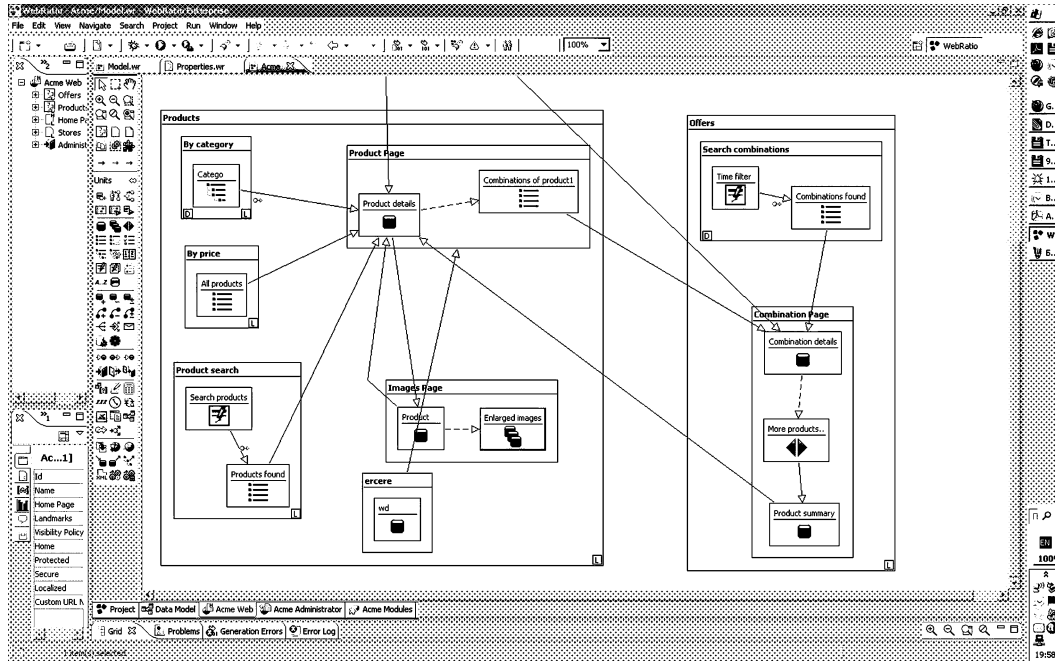


Рис. 5. Гипертекстовая модель Web-приложения Acme

В графическом редакторе WebMLDoc была осуществлена «привязка» элементов гипертекстовой модели к разделам документации, задаваемых в DRL конструкцией InElement. Далее был реализован следующий сценарий для демонстрации работы подхода и инструментария.

1. Администратор Интернет-магазина убрал возможность сортировки товара по цене, а оставил 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.

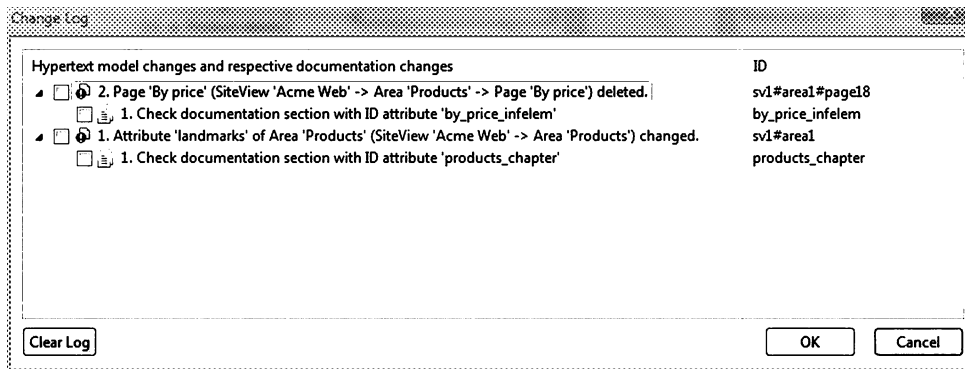


Рис. 6. Пример журнала изменений в WebMLDoc

2. Все изменения запротоколированы и показаны пользователю. Для просмотра изменений в графическом редакторе WebMLDoc достаточно просмотреть лог. В результате удаления сортировки по цене в журнале изменений появится следующая информация (рис. 6). 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».

3. Пользователем вносятся изменения в документацию.

Заключение. В качестве следующего шага по развитию подхода WebMLDoc мы планируем слияние парадигмы линеек программных продуктов (Product Lines) [31] и метода DocLine в рамках одной технологии и единого инструментария. При этом будет использоваться одна и та же диаграмма вариативности как для программного продукта, так и для документации. Такой подход позволит в едином месте один раз сконфигурировать product line проект, состоящий из продукта и его документации. В дальнейшем также единообразно будут отслеживаться изменения конфигурации проекта.

В результате развития подхода предполагается получить единый инструмент, позволяющий параллельно разрабатывать как линейки программных продуктов (Web-приложений интенсивной обработки данных), так и документацию. Инструмент должен объединять в себе модуль разработки семейств программных продуктов на основе модельно-ориентированного подхода и модуль разработки документации для семейств программных продуктов.

Литература

1. *Putrycz E., Kark A. W.* Connecting Legacy Code, Business Rules and Documentation // RuleML 2008, LNCS 5321. Berlin; Heidelberg: Springer-Verlag, 2008. P. 17–30.
2. *Pierce R., Tilley S.* Automatically Connecting Documentation to Code with Rose // SIGDOC'02 October 20–23. Toronto, Ontario, Canada: ACM 1-58113-543-2/02/0010, 2002. 7 p.
3. *Wang X., Lai G., Liu C.* Recovering Relationships between Documentation and Source Code Based on the Characteristics of Software Engineering // Electronic Notes in Theoretical Computer Science. Elsevier B.V. 2009. Vol. 243. P. 121–137.
4. *Childs B., Sametinger J.* Literate Programming and Documentation Reuse // Proc. Fourth Intern. Conference on Software Reuse. 1996. P. 205–214.
5. *Valderas P., Pelechano V.* Introducing Requirements Traceability Support in Model-Driven Development of Web Applications // Information and Software Technology J. 2009. Vol. 51, Issue 4. P. 749–768.
6. *Hollings D., Sharpey-Schafer K., Kelleher J.* Representing Software Traceability using UML and XTM with an investigation into Traceability Patterns. Honours Project. 2005.
7. *Schwabe D., Rossi G.* An Object Oriented Approach to Web-Based Application Design. Theory and Practice of Object Systems. 1998. Vol. 4, Issue 4. 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. *De Troyer O. M. F., Leune C. J.* WSDM: A User Centered Design Method for Web Sites // Proc. of the 7th Intern. World Wide Web Conference. Elsevier. 1998. P. 85–94.
10. *Knapp A., Koch N., Zhang G., Hassler H.* Modeling Business Processes in Web Applications with ArgoUWE // UML 2004, LNCS 3273. Berlin; Heidelberg: Springer-Verlag, 2004. P. 69–83.
11. *Fons J., Pelechano V., Albert M., Pastor O.* Development of Web Applications from Web Enhanced Conceptual Schemas // Proc. of the 22th Intern. Conference on Conceptual Modeling (ER 2003). Chicago, IL, USA, October 13–16, 2003. LNCS. Vol. 2813, Berlin: Springer, 2003. P. 232–245.
12. *Rossi G., Pastor O., Schwabe D.* et. al. Web Engineering: Modeling and Implementing Web Applications. Berlin: Springer, 2007. 464 p.
13. WebRation toolkit. URL: <http://www.webratio.com>.
14. *Романовский К. Ю., Кознов Д. В.* Язык DRL для проектирования и разработки документации семейств программных продуктов // Вестн. С.-Петерб. ун-та. Сер. 10. 2007. Вып. 4. С. 1–16.
15. *Кознов Д. В., Романовский К. Ю.* DocLine: метод разработки документации семейств программных продуктов // Программирование. 2008. № 4. С. 1–13.

16. GS1 Standards Document, GS1 Global Traceability Standard. 2009.
17. Do Organisations Need Traceability? Executive Summary. Theta Technologies & Affiliated Resellers. URL: www.informationleader.com/Web/.
18. *Abadi A., Nisenson M., Simionovici Y.* A Traceability Technique for Specifications // Proc. of the 16th IEEE Intern. Conf. on Program Comprehension. 2008. P. 103–112.
19. *Sousa K., Mendona H., Vanderdonck J., Pimenta M. S.* Supporting Requirements in a Traceability Approach between Business Process and User Interfaces // Proc. IHC 08 of the VIII Brazilian Symposium on Human Factors in Computing Systems. 2008. P. 272–275.
20. *Briand L. C., Labiche Y., Yue T.* Automated Traceability Analysis for UML Model Refinements // Information and Software Technology J. 2009. Vol. 51, Issue 2, February. P. 512–527.
21. IEEE Std 1063-2001, IEEE Standard for Software User Documentation.
22. Единая система программной документации (ЕСПД). ГОСТы серии 19. URL: <http://standards.narod.ru/gosts/gost19/gost19.htm>.
23. TheBat Documentation. URL: <http://allbat.info/>.
24. ESET MOD32 Documentation. URL: <http://www.esetnod32.ru/>.
25. *Фаулер М.* UML. Основы. 3-е изд. СПб.: Символ-Плюс, 2006. 192 с.
26. *Da Silva P. P.* User Interface Declarative Models and Development Environments: A Survey // LNCS. 2000. Vol. 1946. P. 207–226.
27. *Ivanov A., Koznov D.* REAL-IT: Model-Based User Interface Development Environment // Proc. of IEEE/NASA ISoLA 2005 Workshop on Leveraging Applications of Formal Methods, Verification, and Validation. Loyola College Graduate Center Columbia, Maryland, USA, 23–24 September 2005. P. 31–41.
28. Eclipse Graphical Modeling Framework. URL: <http://www.eclipse.org/modeling/gmf/>.
29. Saxon 9 Java API. URL: <http://www.saxonica.com/documentation/javadoc/index.html>.
30. JExamXML Java API. URL: <http://www.a7soft.com/jexamxml/index.html>.
31. A Framework for Software Product Line Practice. Version 5.0. URL: <http://www.sei.cmu.edu/productlines/>.

Статья рекомендована к печати проф. А. Н. Тереховым.

Статья принята к печати 10 марта 2011 г.