Министерство образования и науки Российской Федерации
Федеральное агентство по образованию Российской Федерации
Государственное образовательное учреждение высшего профессионального образования
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Специальность 010503 «Математическое обеспечение
и администрирование информационных систем»

Кафедра системного программирования

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

_____

«_____»_____2010 г.

ДИПЛОМНАЯ РАБОТА

Задача слияния карт памяти (Mind Maps) в Web-системе Comapping

Исполнитель – студент 545 группы
Ларчик Е. В,

Руководитель – к.ф.-м.н., доцент
Кознов Д.В,

Рецензент
к. т. н. Гребенщиков Н.Н

Санкт-Петербург 2010

Saint-Petersburg State University
Faculty of Mathematics and Mechanics
Department of Software Engineering

Head of the Software Engineering Department,
PhD, Prof. A.N. Terekhov

Diploma thesis

Mind map merge problem in Comapping web application

Larchyk Yauheni

Supervisor:
Associate Prof, Ph.D.
D.V. Koznov

Reviewer:
Assistant Prof., Ph.D.
N. N. Grebenshikov

Saint-Petersburg 2010

Содержание

1. Введение

В начале 70-х годов прошлого века английским психологом Тони Бьюзеном была предложена техника работы с информацией, основанная на использовании карт памяти (Mind Maps) [3]. Карта памяти представляет собой диаграмму с очень простой нотацией. В центре диаграммы находится главный элемент, олицетворяющий собой ключевую идею или концепцию. Этот элемент затем соединяется с другими элементами, поясняющими и детализирующими его, которые располагаются вокруг, и т.д. (рис. 1). Среди преимуществ карт памяти Тони Бьюзен отмечает следующее – лёгкость восприятия и запоминания информации, экономию времени на поиск в тексте ключевых слов (благодаря тому, что они более заметны и связаны между собой ясными и уместными ассоциациями), развитие у человека системного мышления и т.д. Техника карт памяти доказала свою эффективность в случаях, когда возникает необходимость в структурировании больших объёмов информации с целью упрощения дальнейшей работы с ней (например, при проведении мозговых штурмов) [13].

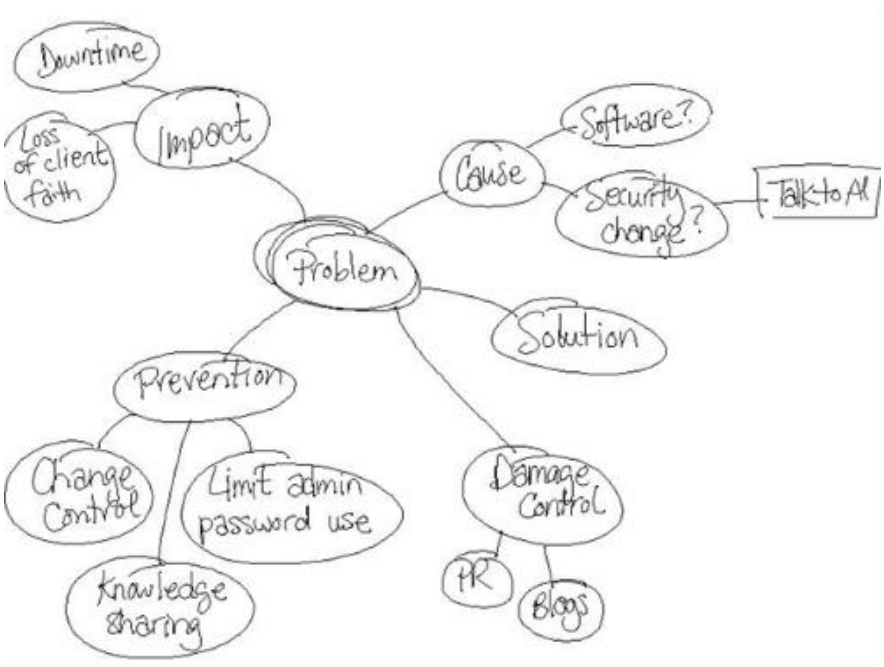

Рис. 1. Пример карты памяти

После создания первых программных продуктов для работы с картами памяти (таких как Freemind[1], MindManager[2]) появилась возможность их раздельного редактирования, что сделало использование карт памяти эффективным при организации совместной работы в различных областях – в бизнесе (управление проектами, планирование, составление документов), в науке (совместное написании книг и статей), в образовании (работа научного руководителя и студента над текстами курсовых и дипломных работ, при проведении экзамена [1]) и т.д. Тем не менее, совместная работа над картами памяти стала по-настоящему удобной лишь с появлением Web-средств, которые хранят данные на специально выделенных для этого серверах. Это позволяет иметь доступ к одной и той же карте памяти с разных компьютеров через Интернет.

Одним из первых таких средств стал продукт Comapping [12]. Comapping реализует всю базовую функциональность для работы с картами памяти (создание, просмотр, редактирование), к которой добавляет функциональность для совместной работы: возможность одновременной работы над картой памяти нескольких пользователей, email-оповещения об изменениях, внесённых другими пользователями, сохранение истории изменений карты памяти. Существует также desktop-версия Comapping, позволяющая сохранять карты памяти локально и работать с ними, не будучи подключённым к Интернет.

Несмотря на то, что Comapping ориентирован на работу в режиме online, когда любое изменение карты сразу же сохраняется на удалённом сервере, существует несколько сценариев, когда редактирование карты памяти происходит в offline-режиме. Например, когда во время работы над картой памяти в Comapping пользователь теряет подключение к сети

1 http://freemind.sourceforge.net
2 http://www.mindjet.com/products/overview

Интернет. В таком случае Comapping позволяет пользователю продолжить свою работу, периодически сохраняя текущее состояние карты памяти локально. Кроме того, как было сказано выше, desktop-версия Comapping позволяет сохранить карту памяти на своём компьютере и продолжить работу над ней в любой удобный момент, даже когда возможности соединиться с Интернетом нет (например, в поезде или самолёте). Между тем, в то время, как пользователь работает над картой памяти в режиме offline, ее могут редактировать другие пользователи в режиме online. Поэтому, когда пользователь вновь подключится к Интернету и захочет сохранить на Comapping-сервере изменения, сделанные им в режиме offline, возникнет необходимость в слиянии двух версий одной и той же карты памяти: offline-версии, которая сохранена на компьютере пользователя, и online-версии, которая сохранена на сервере. Целью данной дипломной работы является решение такой задачи слияния.

Данная задача является актуальной для Web-ориентированных приложений для работы с картами памяти, в том числе и для конкурирующих с Comapping средств Mindomo[3] и MindMeister[4]. Тем не менее, эти средства не предлагают соответвующего решения, хоть и позволяют работать с картами памяти в режиме offline. Mindomo при сохранении карты памяти не проверяет, изменялась ли она другими пользователями, и всегда перезаписывает её «поверх» серверной копии. В MindMeister такая проверка делается, однако если карта изменялась другими пользователями, единственный выбор для пользователя — сохранить изменённую в режиме offline карту под новым именем.

Задача слияния двух версий одного и того же информационного актива напрямую связана с задачей синхронизации, которую в общем случае, следуя [7], можно сформулировать так. Предположим, что имеется

3 http://www.mindomo.com/
4 http://www.mindmeister.com

два набора данных; синхронизация между этими наборами — это процесс их изменения таким образом, чтобы они стали идентичными в той части, где они описывают одну и ту же информацию. Конкретные требования к процессу зависят от контекста, в котором решается эта задача. Задача слияния отличается от задачи синхронизации тем, что после синзронизации одная из версий информационного актива прекращает своё существование. При синхронизации каждая из версий продолжает своё существование и может отличаться от другой в тех своих частях, где они принципиально различны. Задачи слияния и синхронизации информации возникают при создании и использовании средств версионного контроля (CVS[5], Subversion[6]) и других средств совместной работы над различными типами информации (например, работа над картами памяти в Comapping). При этом, информация, которую необходимо синхронизировать и «сливать», может быть самой разнообразной: текстовые файлы, xml-файлы, онтологии, схемы баз данных и пр.

5 http://www.cvs.com
6 http://subversion.tigris.com

2. Постановка задачи

Целью данной дипломной работы является решение задачи слияния карт памяти в Comapping. Для её решения были поставлены следующие задачи:

1) сформулировать требования к решению задачи слияния в Comapping;

2) выбрать наиболее подходящий алгоритм из имеющихся и обосновать его эффективность;

3) адаптировать этот алгоритм к требованиями Comapping;

4) реализовать полученный алгоритм и интегрировать его в Comapping.

3. Обзор продукта Comapping

Comapping — это Web-приложение для работы с картами памяти. Данный продукт является результатом сотрудничества датской компании Area9[7], петербургской компании ЗАО «Ланит-Терком»[8] и математико-механического факультета СПбГУ[9]. Для представления карт памяти Comapping использует древовидную нотацию, в которой уточнение концепций происходит слева направо (так называемый left-to-right mindmapping), соответственно, главный элемент карты памяти находится левее остальных (см. рис.2). Такая нотация в сочетании с алгоритмом автоматического перераспределения элементов на видимой области экрана облегчает чтение и понимание карты памяти [4].

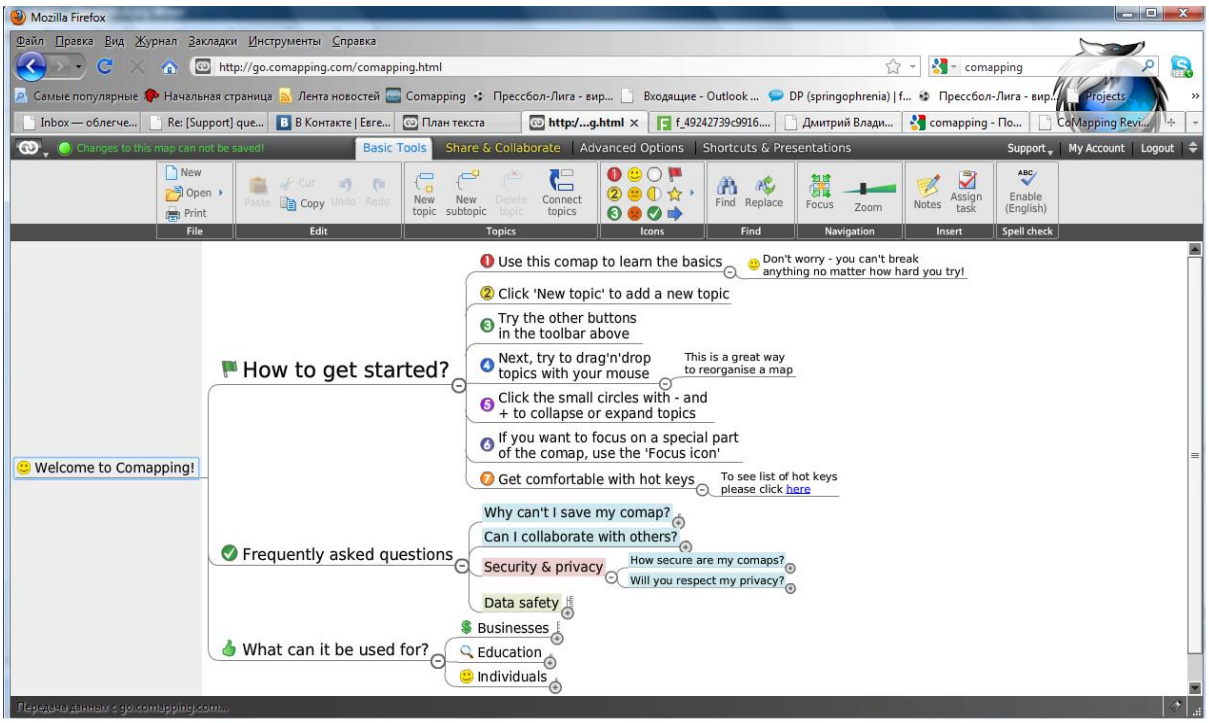

Рис.2. Пример карты памяти в Comapping

Функциональность для работы с картами памяти в Comapping также

7 http://area9.dk
8 http://www.lanit-tercom.com/
9 http://www.math.spbu.ru/

включает в себя следующие возможности:

- связывать с элементами карты памяти файлы и заметки;
- проверять правописание текста в узлах карты памяти;
- осуществлять текстовый поиск и фильтрацию элементов;
- публиковать карту памяти в своём блоге или на сайте;
- организовывать карты памяти в иерархическую структуру, используя папки;
- печатать даже большие по размеру карты памяти, используя настраиваемый механизм печати;
- экспортировать карты памяти в форматы PDF, HTML (рис.3), RTF, SVG, а также осуществлять импорт/экспорт в форматы других средств работы с картами памяти (FreeMind, MindManager).

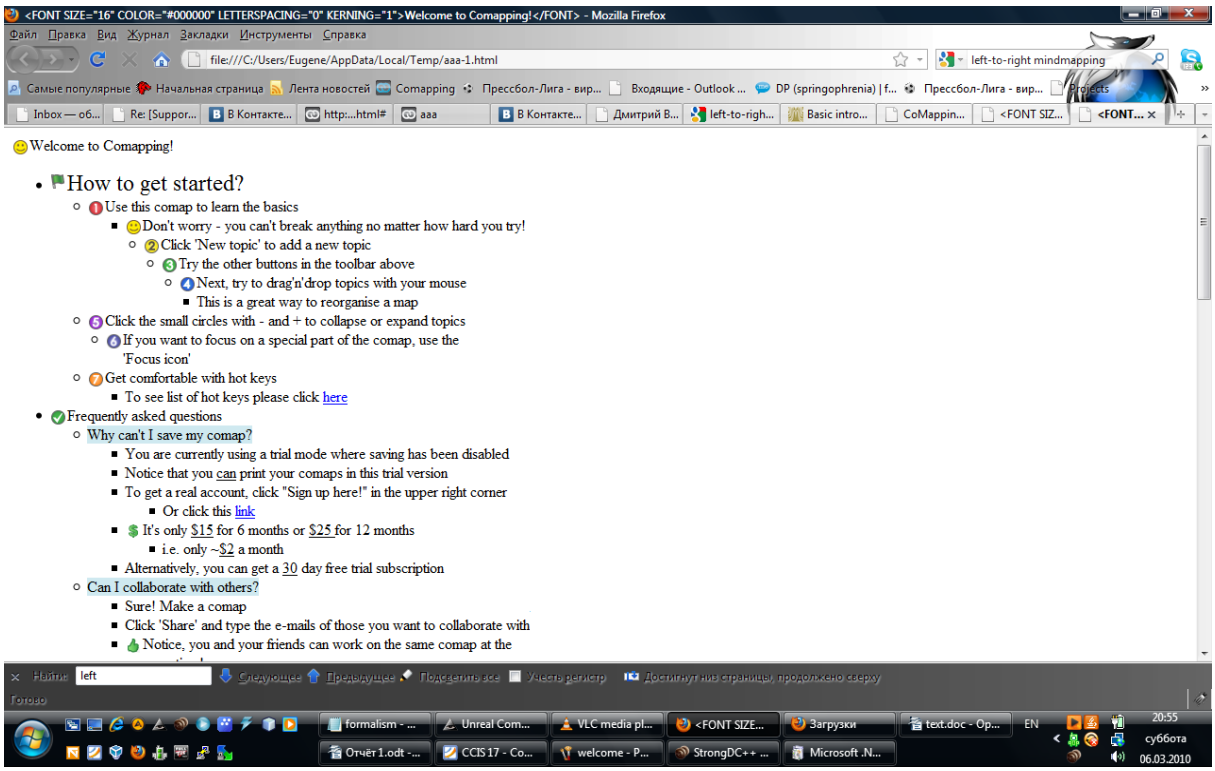

Рис.3. Экспорт карты памяти в формат HTML

Comapping фокусируется на предоставлении максимальных удобств при совместной работе пользователей над картой памяти, что реализуется

с помощью следующей функциональности:

- возможность открыть одну и ту же карту для просмотра или редактирования разным пользователям;
- возможность объединять пользователей в группы и изменять права доступа к карте памяти сразу всем участникам группы;
- возможность редактирования карты памяти несколькими пользователями одновременно (при этом можно видеть, в каком месте карты памяти находятся другие участники (рис. 4), а также общаться во встроенном чате);
- возможность прикреплять к элементам карты памяти задания для пользователей, работающих с ней;
- email-оповещения об изменениях карты памяти другими пользователями;
- хранимая история изменений, которая позволяет отследить, кем и когда была изменена та или иная часть карты памяти.

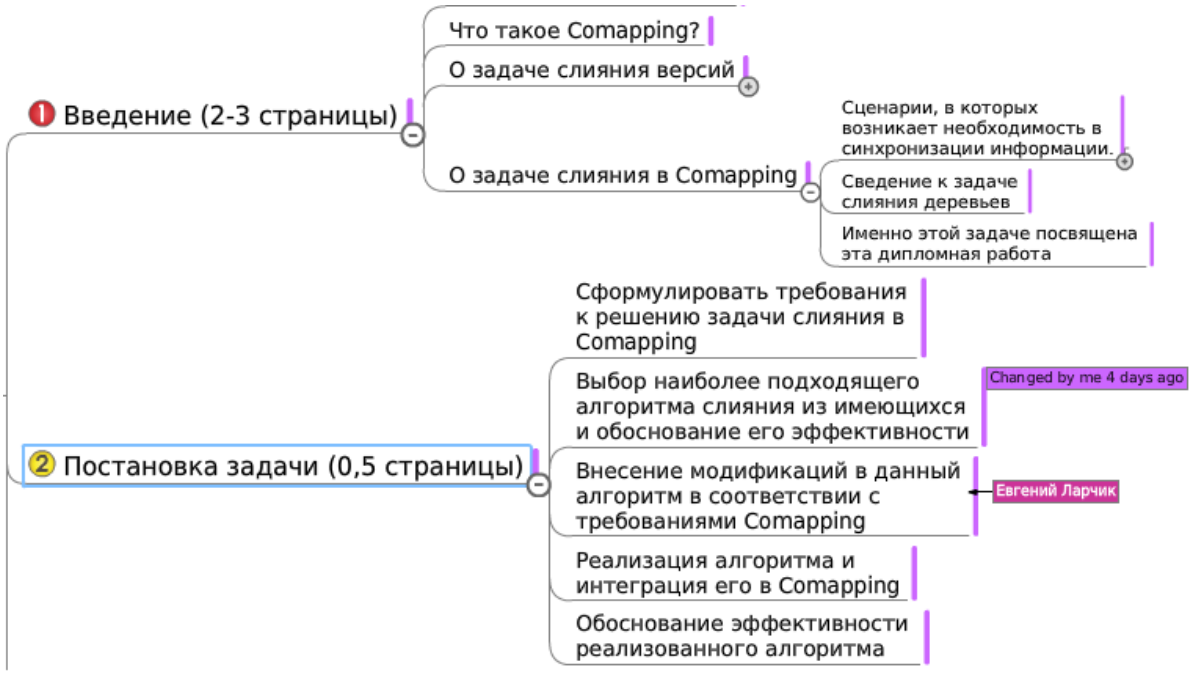

Рис.4. Совместная работа над картой памяти в Comapping

Desktop-версия Comapping требует установки на компьютер пользователя, но позволяет сохранять карты памяти локально и редактировать их, не имея подключения к сети Интернет. Карты памяти сохраняются в файлы формата .comap – внутреннего формата Comapping, основанного на XML.

Продукт Comapping реализован как приложение Adobe Flash[10] на клиентской стороне и как приложение Apache[11]/Neko[12] на серверной стороне. И серверная, и клиентская части написаны на языке программирования haXe[13]. Desktop-версия Comapping разработана с помощью технологии Adobe AIR[14].

10 Adobe Flash (http://en.wikipedia.org/wiki/Adobe_Flash) – мультимедийная платформа, используемая для создания векторной анимации интерактивных приложений (в том числе, игр), а также для интеграции видеороликов в Web-страницы.

11 Apache httpd (http://httpd.apache.org/) – HTTP-сервер с открытым кодом, самый часто используемый HTTP-сервер в Интеренете.

12 Neko (http://nekovm.org/)– «легковесная» виртуальная машина, созданная для одноимённого языка. На сегодняшний день популярна, как одна из платформ, на которых можно создавать серверную часть Web-приложений на языке haXe.

13 haXe (http://haxe.org/) – многоплатформенный язык программирования для создания Web-приложений. Позволяет создавать клиентскую и серверную часть Web-приложений на одном языке программирования.

14 Adobe AIR (http://www.adobe.com/ru/products/air/) – технология, позволяющая на основе обычных броузерных Flash-приложений создавать полноценные desktop-приложения с расширенными возможностями.

4. Требования к решению задачи слияния в Comapping

В начале работы над задачей слияния карт памяти возникла необходимость определить тебования к её решению. Эти требования неочевидны, поскольку у каждого человека своё представление о том, какой результат слияния карт памяти является хорошим. Поэтому в рамках данной дипломной работы автором были проведены беседы с менеджерами, разработчиками и пользователями Comapping с целью выявить и обобщить их мнения и предложения. После анализа собранной информации были сформулированы следующие требования к решению задачи слиянии в Comapping.

1. Процесс слияния должен происходить автоматически, т.е. не требовать по умолчанию каких-либо действий пользователя. Это значит, что все спорные ситуации (конфликты), произошедшие в процессе слияния, должны быть разрешены неким способом по умолчанию. Это требование обусловлено тем, что большинство пользователей Comapping не являются «продвинутыми» IT-пользователями, и потому предложить им готовый результат слияния карт памяти предподчтительнее, чем требовать от них участия в процессе, а значит, и его понимания.

2. Информация о конфликтах и участвующих в них элементах карты памяти должна быть сохранена и показана пользователю, чтобы у него была возможность при необходимости разрешить их способом, отличным от принятого нами по умолчанию. Это функционал для «продвинутых» пользователей, которые хотят контролировать процесс слияния карт памяти.

3. Чтобы предотвратить потерю информации в процессе слияния, конфликтные ситуации, возникающие при удалении какой-либо части карты памяти в одной из её версий и при изменении этой части в другой, должны решаться в пользу неудаления изменённой части.

4. Поскольку Comapping заботится о сохранении истории изменений

карты памяти (она позволяет проследить, как, кем и когда изменялась карта памяти, что важно для любого совместно редактируемого документа), мы не можем сохранить результат слияния online-версии и offline-версии, просто удалив предыдущую, т.к. иначе история всех предыдущих изменений станет бесполезной. Поэтому в нашем решении мы должны получить список операций, переводящий online-версию в результат слияния, и применить к ней эти изменения.

5. Обзор существующих решений

Карта памяти в Comapping представляет собой упорядоченное дерево, в узлах которого находится информация об элементах карты (текст, заметки, иконки, прикреплённые файлы). Поэтому задачу слияния карт памяти можно рассматривать как задачу о слиянии деревьев. Такая задача является широкоизвестной, однако она чаще всего она рассматривается в контексте слияния XML- и XHTML-файлов [4, 6, 7, 9]. В этом случае стандартные методы, применяемые для слияния текстовых файлов в системах контроля версий, оказываются непригодными, поскольку они не учитывают древовидную структуру XML- и XHTML-файлов и нарушают её даже в простейших случаях – пример представлен на рис. 5. Поэтому возникает необходимость в алгоритмах слияния, работающих с деревьями.

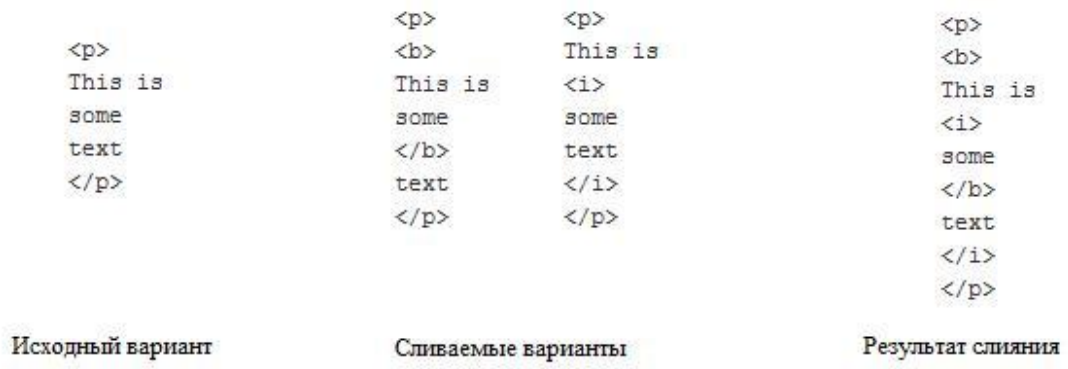

Рис. 5. Пример недопустимости построчного слияния XHTML-файлов

Интерес к задаче слияния XML-файлов возник в начале 2000-х годов и был обусловлен тем, что формат XML стал использоваться различными программными средствами для хранения данных (например, Microsoft Office[15], OpenOffice[16]). Задача слияния XML-файлов тесно связана с более ранней задачей нахождения разницы между двумя XML-файлами (change

15 http://office.microsoft.com/ru-ru/default.aspx
16 http://www.openoffice.org/

detection), поскольку любой алгоритм слияния так или иначе сравнивают сливаемые файлы и находят их разницу. Чаще всего эта разница представляется в виде списка операций, переводящих один из файлов в другой (этот список называют edit script) [5].

Итогом интереса к задаче слияния стало появление нескольких различных законченных подходов [7, 6, 4]. После этого многие исследователи и практики высказали мнение, что задачу слияния XML-файлов можно считать решённой[17], т. к. в большинстве случаев при возникновении необходимости в слиянии достаточно адаптировать под свои требования одно из предложенных средств с открытым кодом.

Опишем существующие подходы к решению задачи слияния. Существует два основных подхода — так называемые 3 way merge [4, 6, 7] и 2 way merge [6, 9]. Алгоритмы, использующие первый из них, рассматривают изменения двух сливаемых деревьев относительно базовой версии — общего предка для обоих сливаемых деревьев, и пытаются применить все эти изменения к этой базовой версии. Алгоритмы, использующие второй подход, применяются в случаях, когда изменения сливаемых деревьев относительно базовой версии дерева проследить невозможно либо когда общего предка для сливаемых деревьев нет вообще (например, когда сливаем деревья, полученные совершенно различными способами, но возможно имеющие какую-то общую часть). В случаях, когда информация о базовом дереве может быть получена, использование первого подхода предпочтительнее, так как это позволяет избежать ложных конфликтов, отсечь которые невозможно, не имея такой информации. Например, если в одной из веток содержимое узла было изменено, а в другой – нет. Так же это позволяет избежать включения в результат слияния излишней информации. Например, когда в одной из веток

17 http://useless-factor.blogspot.com/2008/01/matching-diffing-and-merging-xml.html
http://stackoverflow.com/questions/2222548/3-way-xml-merge-algorithm

поддерево было удалено, а во второй — нет [7].

Рассмотрим подробнее различные алгоритмы слияния деревьев. При этом результат слияния будем называть *merged-деревом*, общего предка сливаемых деревьев — *base-деревом*, а сливаемые деревья — *ветками*.

5.1. Технология So6

В работе [4] описывается технология So6, которая сливает деревья на основе подхода Operation Transformation (OT). Этот подход предполагает, что имеются две изменяемые копии одного и того же артефакта (текста, дерева и т.д.) и нужно транслировать операции, изменяющие любую из этих копий, на другую копию. Поскольку предполагается, что копии могут изменяться параллельно, то контекст, в котором операция была применена в первый раз, может отличаться от того контекста, в котором её нужно применить на другой копии. Поэтому возникает необходимость преобразовывать операцию перед ее выполнением на другой копии в зависимости от того, как эта копия была изменена локально. Для этого описываются функции преобразования, которые для всевозможных пар примитивных операций определяют, каким образом должно измениться выполнение первой операции при условии, что до неё была выполнена вторая операция. Эти функции, очевидно должны быть коммутативными, поскольку для двух параллельно выполненных на разных копиях операций результат слияния не должен различаться в зависимости от того, с какой из операций мы начинаем распространение – получившиеся в итоге две копии должны быть идентичны. So6 накладывает ещё одно ограничение на процесс синхронизации: в каждый момент времени на всех копиях транслируется и преобразовывается максимум одна операция. Таким образом, процесс синхронизации можно описать следующим образом: 1) изменение одной из копий генерирует операцию, которая должна быть применена ко второй копии; 2) при получении операции на второй копии

мы получаем список операций, изменявших эту копию локально, т.е. не примененных к первой копии; 3) преобразовываем полученную операцию относительно каждой из этих локальных операций; 4) результат применяем ко второй копии.

Технология SO6 является детальной моделью, в которой подробно описан механизм синхронизации деревьев. В частности, авторы предлагают точные спецификации функций преобразования, а также алгоритмов и используемых структур данных. Существует также открытая реализация подхода на языке Java, которая является частью проекта LibreSource[18].

К недостаткам данного подхода следует отнести следующие.

1) So6 рассматривает только две примитивные операции над деревьями: вставку (insert) и удаление (delete) поддерева. То есть в работе отсутствует поддержка операций перемещения поддерева (move) и изменения узла (change). Это значит, что авторами описаны только 2x2=4 из 4x4=16 функций преобразования. В тоже время написание функций преобразования, корректных относительно свойства коммутативности, и проверка их правильности, по признанию авторов, является сложной и трудоёмкой задачей.

2) OT-подход можно адаптировать к задаче слияния в Comapping: для этого, например, можно все операции, изменявшие одно из сливаемых деревьев, преобразовать относительно операций, изменявших второе дерево. Однако данный подход нацелен на синхронизацию деревьев в режиме реального времени, и, очевидно, показывает лучшие результаты, когда синхронизируются небольшие изменения деревьев.

18 http://dev.libresource.org/
LibreSource – Web-портал, на котором собраны средства (с открытым кодом) для совместной работы нескольких человек над своими проектами

5.2. Средство DeltaXML

В работе [6] предлагается подход к слиянию XML-файлов на основе использования промежуточного XML-файла, который содержит оба сливаемых файла и организован так, что легко определить, какие данные являются общими для сливаемых файлов, а какие — уникальны для каждого из файлов. Тапким образом, в отличии от других методов, данный подход при слиянии порождает не edit script, а текстовую дельту. Кроме того, подход нацелен на слияние XML-данных, а не XML-документов. Как говорят авторы, в первом случае нет особой необходимости поддерживать операцию move, так как по их опыту структура XML-файлов с данными не склонна часто меняться на практике, а встречающиеся редкие случаи модно промоделировать композицией операций delete/add. Подход может быть использован как при 2 way merge, так и при 3 way merge. В качестве средства, использующего данный подход, предлагается коммерческий пакет DeltaXML[19].

В случае 2 way merge из двух сливаемых файлов DeltaXML строит так называемый delta-файл. Этот файл содержит все данные из обоих файлов; общие данные двух сливаемых файлов присутствуют в delta-файле в единственном экземпляре; простой XSL-скрипт позволяет получить из delta-файла каждый из двух сливаемых файлов. Затем специальный XSL-скрипт строит из delta-файла merged-файл, который является результатом слияния (рис. 6).

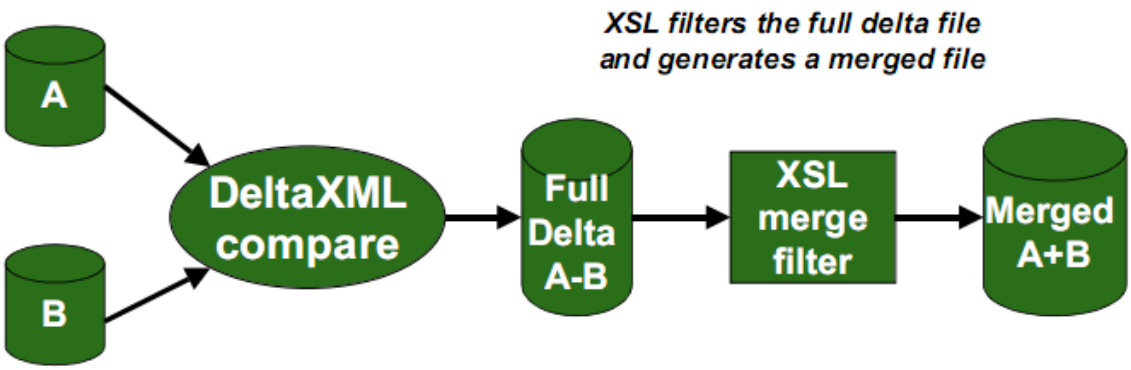

Рис. 6. Сценарий выполнения 2 way merge в DeltaXML

В случае 3 way merge DeltaXML строит два delta-файла, сравнивая каждый из сливаемых файлов с базовым. Затем из этих файлов XSL-преобразованием строится два аннотационных файла, которые содержат информацию об изменениях сливаемых файлов относительно базового. После этого аннотационные файлы сливаются с использование 2 way merge (рис. 7).

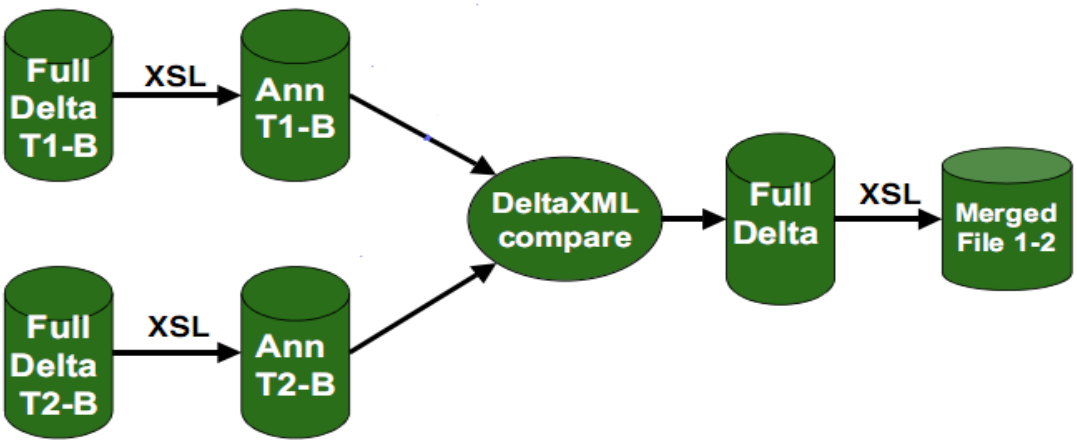

Рис. 7. Сценарий выполнения 3 way merge в DeltaXML

Данный подход не поддерживает move-операцию и DeltaXML, являясь коммерческим продуктом, скрывает детали реализации построения delta-файлов, а также их преобразования в аннотационные и merged-файлы, недоступны. Для нашей задачи открытость исходного алгоритма критически важна, поскольку его необходимо изменять в соответствии с требованиями Comapping. В нашем случае также необходима поддержка move-операции.

5.3. Алгоритм слияния с использованием отпечатков контекста

Подход, описанный в работе [9], предлагает использовать при слиянии XML-файлов следующую технику: сначала алгоритм находит

дельту между base-деревом и одной из веток (список операций, переводящих одно дерево в другое), и потом применяет эту дельту ко второй ветке. Таким образом, дельта применяется не к тому дереву, для которого она была посчитана. Главная сложность такого подхода заключается в том, что контекст выполнения операции мог измениться во второй ветке. Предлагается решать эту задачу с помощью так называемых отпечатков контекста операций. То есть для каждой операции сохраняется её отпечаток, который представляет собой значения некоторой хеш-функции от узлов, находящихся в некоторой окрестности узла, к которому была применена операция. Эти отпечатки помогают определить точное место во второй ветке, в котором необходимо произвести операцию, даже если оно отличается от места в первой ветке, в котором она была произведена изначально. Для этого рассматривается окрестность первоначального места выполнения операции и с помощью отпечатка операции ищется наиболее вероятное место выполнения операции во второй ветке.

Подход представляет собой описание теоретической модели, в которой подробно описаны шаги алгоритма слияния. Модель была реализована и апробирована, однако детали реализации авторами умалчиваются. К сожалению, реализацию не удалось найти и в свободном доступе в Интернет.

К недостаткам данного подхода следует отнести то, что он не рассматривает операцию перемещения поддерева, а также то, что он является 2 way merge алгоритмом.

5.4. Подход 3DM

В работе [7] представлен подход 3DM – алгоритм и программное средство с открытым кодом для слияния деревьев на основе подхода 3 way merge. В основе 3DM лежит следующая идея. Для каждого узла двух

сливаемых деревьев рассматриваются изменения, произошедшие с ним относительно base-дерева, и merged-дерево строится применением к base-дереву всех этих изменений. При этом, во-первых, предлагается рассматривать только локальные изменения узла, т.е. изменения его содержимого и изменения в списке его детей. Для определения изменений, произошедших с узлами в ветке, в алгоритме используется matching (процесс сопоставления узлов двух деревьев [10]) между base-деревом и веткой. При этом идентифицируются следующие операции — вставка узла, удаление узла, изменение содержимого узла, перемещение узла к новому родителю и копирование узла. Во-вторых, все изменения, произошедшиее в поддереве в одной из веток, алгоритм применяет ко всем копиям этого поддерева во второй ветке, если таких несколько.

Подход 3DM включает в себя подробное описание алгоритма слияния карт памяти с доказательством его свойств, а также реализацию подхода в виде средства на языке Java.

Алгоритм, использующийся в 3DM, обладает следующими достоинствами:

1) использует подход 3 way merge (как было сказано выше, 3 way merge даёт лучшие результаты при слиянии, чем 2 way merge);

2) рассматривает все операции, которые могут быть произведены с узлом дерева в Comapping (изменение и вставка узла, удаление и перемещение поддерева);

3) является открытым (доступные его реализации на Java[20] и haXe[21]).

Поэтому данный алгоритм представляется наиболее подходящим для нашей задачи и был выбран в качестве базового.

Опишем алгоритм слияния 3DM. Он включает в себя два шага. На первом шаге происходит matching узлов base-дерева и узлов в каждой из

20 http://tdm.berlios.de/3dm/doc/index.html

21 Реализация алгоритма на языке haXe была выполнена программистом Comapping Николаем Артамоновым

веток. На втором шаге происходит непосредственное слияние деревьев на основе информации, полученной на первом шаге.

5.4.1. Matching в 3DM

Matching между двумя деревьями T1 и T2 в 3DM определяется как набор пар связанных между собой узлов (n, m), где узел n из T1, узел m из T2. Автор алгоритма рассматривает 3 типа связей: content match (узлы сопоставляются, т.к. у них одинаковое или мало отличающиеся содержимое), structure match (узлы соспоставляются, т.к. у них одинаковые или малоотличающиеся списки детей) и full match (связь по содержимому и структуре одновременно).

Каждому узлу ветки сопоставляется максимум один узел из base-дерева. Если узлу в base-дереве соспоставлен только один узел в ветке, то связь между ними автоматически full match. Однако узлу в base-дереве может быть сопоставлено более одного узла в ветке, но тогда ровно один из этих узлов сопоставляется ему со связью full match, и он называется первичной копией узла из base-дерева в ветке.

Два узла из разных веток называются *партнёрами*, если для них существует узел из base-дерева, с которым они входят в matching.

Идеальный matching получается, если с помощью истории изменений отслеживаются все изменения каждого узла. Однако, поскольку автор ставит своей целью создать универсальный алгоритм слияния деревьев, то предполагается, что истории изменений дерева нет. Поэтому вводится понятие достаточного matching. Matching является достаточным, если при его использовании в 3 way merge алгоритме результат будет таким же, как и при использовании идеального matching. Алгоритм, использующийся в 3DM строит достаточный matching между base-деревом и веткой.

5.4.2. Слияние деревьев в 3DM

Идея самого алгоритма слияния основывается на последовательном построении merged-дерева путём одновременного обхода узлов обеих веток таким образом, что сопоставленные на предыдущем этапе узлы посещаются одновременно. Поэтому для merged-дерева и каждой из веток заводится свой курсор, который в каждый момент времени указывает на какой-то узел в дереве. Курсоры веток также могут указывать на специальный нулевой узел. Узлы, на которые указывают курсоры, назовём *текущими*. Параллельно с построением merged-дерева ведётся учёт конфликтов, произошеших в процессе слияния, и строится список операций переводящих base-дерево в merged-дерево (так называемый edit script [5]).

В началае работы алгоритма merged-дерево состоит только из корня и курсоры всех трёх деревьев указывают на их корни. Каждый новый шаг алгоритма строит список детей текущего узла в merged-дереве. Для этого используется информация о списках детей текущих узлов в ветках.

Рассмотрим подробнее шаги, выполняющиеся на каждой итерации алгоритма. В начале итерации курсоры веток либо указывают на пару партнёров, либо один из них указывает на нулевой узел (когда в одной из веток узел был вставлен или удалён). Курсор merged-дерева указывает на узел, для которого мы строим список детей. В качестве примера рассмотрим первую итерацию алгоритма для деревьев, изображённых на рисунке 8.1. Текущими узлами веток являются их корни.

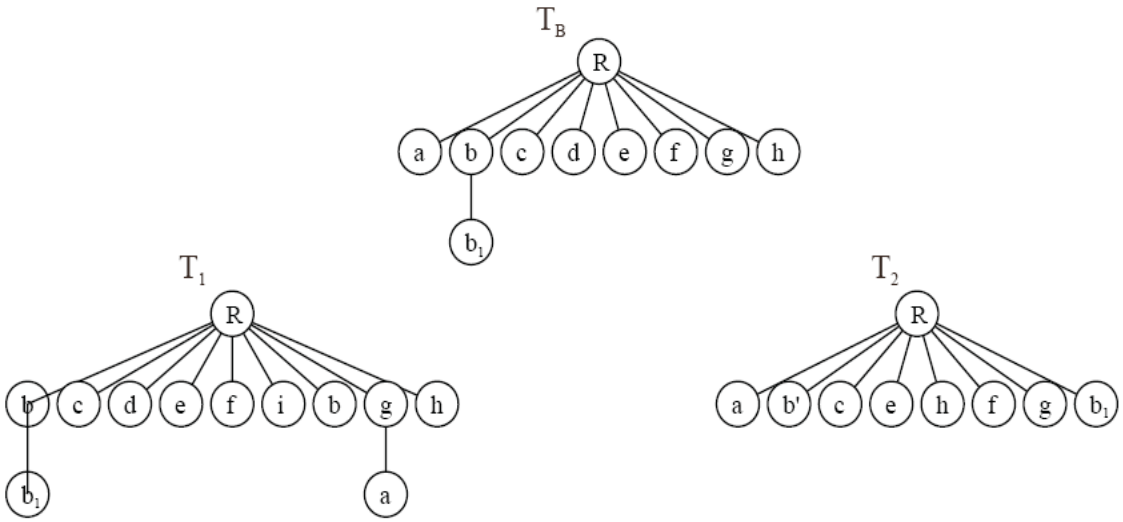

Рис. 8.1. Базовое дерево Tb и ветки T1, T2 в начале работы алгоритма

Шаг 1. Построение merge-списков для текущих узлов веток

Целью этого шага является представления списка детей текущих узлов веток в удобном для дальнейших действий виде. Для этого вводится специальная структура данных — merge-список. Такой список строится для каждой из двух веток. Merge-список является списком записей, каждая из которых содержит главный узел и некоторое числа дополнительных узлов (так называемых hangon-узлов). Узел ветки является главным, если он не изменил своего родителя по сравнению с base-деревом. Узел является hangon-узлом, если в ветке он был перемещён или вставлен. Запись может быть заблокированной или нет. Merge-список начинается специальной записью **α** и заканчивается специальной записью **ω**. Пример merge-списков можно увидеть на рис. 8.2: hangon-узлы составлены в стек над главным узлом записи, заблокированные записи подчёркнуты.

Будем говорить, что заблокированная запись является left-locked, если и предыдущая запись в merge list также заблокирована, right-locked – если заблокирована последующая запись. Merge-список узла строится по следующему правилу. Для каждого его потомка (назовём его child):

1) если child был вставлен в ветке, то он добавляется, как hangon

к последней записи в merge-списке, и эта запись становится left-locked (т. е. блокируется сама запись и предыдущая);

2) если child был перемещён в ветке, то производятся аналогичные действия;
3) если child – являтеся непервичной копией узла из base-дерева, то он добавляется как hangon к последней записи в merge-списке и эта запись становится left-locked и right-locked;
4) иначе в merge list добавляется новая запись, главным узлом которой является child.

Merge-списки для корня merged-дерева в нашем примере представлены на рисунке 8.2.

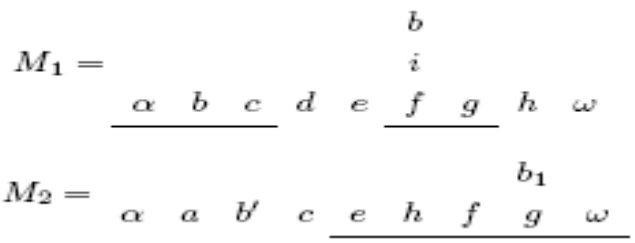

Рис. 8.2. Шаг1. Merge-списки для корней веток

Шаг 2. Проверка удалений и перемещений узлов

На этом шаге в двух merge-списках находятся те записи, главные узлы которых не должны появиться в итоговом списке детей в merged-дереве. Это те узлы, которые в одной из веток не поменяли своего положения, а в другой — были удалены или перемещёны к новому родителю. При необходимости после удаления записи её hangon-узлы перемещаются к предыдущей.

При удалении проверяется, не изменилось ли поддерево с корнем в удалённом узле относительно base-дерева, и если изменилось, то фиксируется конфликт Update/Delete.

После этого шага в обоих merge-списках остаётся равное число

записей (см. рис. 8.3)

$$M_1^D = \begin{array}{cccccccc} & & & & b & & & \\ & & & & i & & & \\ \underline{\alpha} & \underline{b} & \underline{c} & e & \underline{f} & \underline{g} & h & \omega \end{array}$$

$$M_2^D = \begin{array}{cccccccc} & & & & & & b_1 & \\ \alpha & b' & c & \underline{e} & \underline{h} & \underline{f} & \underline{g} & \underline{\omega} \end{array}$$

Рис. 8.3. Merge-списки после шага 2

Шаг 3. Построение парного merge-списка детей

На этом шаге мы объединяем merge-списки текущих узлов веток в список пар. Этот список фактически соответствует списку детей текущего узла в merged-дереве, т. к. каждая пара в этом списке является либо парой партнёров, либо парой, один из узлов которой является нулевым. Во время построения этого списка для каждого узла, попавшего в какой-либо из merge-списков, находится его партнёр в другой ветке и при необходимости фиксируется конфликт Move/Move (если оказывается, что оба парнёра были перемещены и при этом в разные локации). Список пар для нашего примера представлен на рисунке 8.4.

$$\mathrm{M} = \begin{bmatrix} b & c & e & h & f & i & b & g & b_1 \\ b' & c & e & h & f & \cdot & b' & g & b_1 \end{bmatrix}$$

Рис. 8.4. Шаг3. Список пар

Шаг 4. Построение списка детей текущего узла в merged-дереве и рекурсивный вызов

На этом шаге каждая пара из списка, построенного на предыдущем шаге, добавляет к текущему узлу merged-дерева новый узел. Его содержимое определяется содержимым узлов входяших в пару узлов. При необходимости фиксируется конфликт Update/Update. Курсор merged-

дерева перемещается на добавленный узел, а курсоры веток — на соответсвующие узлы из пары. После этого процедура построения детей запускается рекурсивно.

6. Модификация 3DM-алгоритма в соответствии с требованиями Comapping

Несмотря на то, что базовый алгоритм, описанный выше, позиционируется самим автором как универсальный алгоритм слияния деревьев, основным случаем его применения является слияние деревьев, полученных из xml- и xhtml-файлов. Такие файлы содержат размеченный текст, поэтому для узлов полученных деревьев важен, например, порядок следования их детей, поскольку правильный порядок обеспечивает корректность структуры документа, а также содержательность текста. Поэтому базовый алгоритм внимательно следит за контекстом, в который попадает вставляемый или перемещаемый узел дерева (это реализуется с помощью использования блокировок в merge-списке). При слиянии карт памяти правильный порядок детей не является критически важным, но специфика Comapping накладывает свои ограничения, что приводит к необходимости модификаций базового алгоритма, которые описаны ниже.

6.1. Изменения в matching

Поскольку каждый узел карты памяти в Comapping имеет уникальный идентификатор (id), то процесс matching упрощается, т.к. сопоставляются друг другу только узлы с одинаковыми id. С другой стороны, id узлов карты памяти не видны пользователю, поэтому в процессе редактирования он может перестать связывать старое и новое содержимое узла, хотя эта связь остаётся в виде неизменного id. В таких случаях сопоставлять две версии узла друг другу не имеет смысла, несмотря на то, что у них одинаковые id. Поэтому в процессе matching мы дополнительно проверяем «похожесть» сопоставленных по id узлов. Эта «похожесть» измеряется с помощью определённой метрики. Идея matching с метрикой при слиянии онтологий и баз данных подробно обсуждается в

работе [10].

Предлагаемая здесь метрика «похожести» узлов определяется тремя следующими параметрами: местоположением узла в ветке (изменилось ли оно относительно местоположения узла в base-дереве), сходством содержимого узлов (число в интервале [0, 1]) и «похожестью» списков детей узлов (число в интервале [0, 1]).

Сходство содержимого в Comapping определяется схожестью плоского текста в узлах (двух строк без учёта регистра, форматирования, иконок и т.д.). Для сравнения текста в узлах мы используем алгоритм нахождения Q-расстояния между строками [11]. Этот алгоритм заключается в следующем: каждой из двух строк сопоставляется ассоциативный вектор, в котором всевозможным подстрокам длины Q сопоставляется количество раз, которое они встречаются в исходной строке. Q-расстояние равно модулю разности этих векторов. Q-расстояние равно 0, если строки равны. Для простоты дополним строки Q – 1 специальным символом (который не встречается ни в одной из строк) в конце, тогда сумма координат ассоциативного вектора для строки равна длине строки, и в таком случае, максимум Q-расстояния равен сумме длин двух строк. Таким образом, похожесть контента узлов мы определяем как Q-расстояние между двумя строками, делённое на сумму длин строк.

«Похожесть» списка детей узлов мы определяем как 2 * M / N, где M – это число пар детей, совпадающих по id, N – общее кол-во детей у обоих узлов. Пустые списки детей считаем похожими со значением 1.

В зависимости от значения третьего параметра мы определяем нижние пределы для двух других параметров: если и «похожесть» контента, и «похожесть» узлов опускается ниже этих пределов, то узлы не должны попадать в matching. Значения пределов следующие: если узел был перемещён в ветке, то 0.3 для обоих параметров, если не был перемещён, то 0.1 для обоих параметров. Данные коэффициенты были выбраны

достаточно приблизительно, но с учетом следующих соображений:

- в первом случае пределы должны быть выше, поскольку, перемещая узел, пользователь, как минимум, перестаёт связывать его с прежним местоположением, и поэтому имеет смысл требовать большей схожести сопоставляемых узлов для включении их в matching;
- в целом коэффициенты должны быть явно занижены, поскольку пока наша цель — убрать из matching только очевидно разные узлы.

Однако, при последующих улучшениях алгоритма вопрос о точном значении пределов, безусловно, должен быть рассмотрен более детально.

6.2. Неравнозначность online- и offline- деревьев

Алгоритм, используемый в 3DM, не делает различий между двумя сливаемыми ветками. В то же время в Comapping ветки чётко разделены: есть offline-дерево, которое было сохранено локально, и online-дерево, которое представляет собой текущее состояние карты памяти на сервере. Это разделение проявляется, в частности, в том, что симметричные кофнликты (Update/Update, Move/Move, ситуации спорного порядка узлов) мы разрешаем всегда в пользу online-дерева. Причина такого подхода состоит в том, что во время процесса слияния над картой памяти могут работать другие пользователи в режиме online, и ситуация, когда сделанные ими только что изменения вдруг исчезнут (в случае, если они конфликтуют с локальными изменениями карты памяти), неприемлема. Кроме того, при таком подходе пользователю, производящему слияние, удаётся сразу понять, каким образом были изменены конфликтные узлы в online-дереве (про offline-дерево он и так знает).

6.3. Изменения в Edit script

Edit script – это список операций, переводящих одно дерево в другое [5]. Базовый алгоритм порождает edit script, переводящий base-дерево в merged-дерево. Однако согласно требованию 4 к решению задачи слияния в Comapping, алгоритм слияния должен порождать edit script, переводящий online-дерево в merged-дерево. Это можно сделать двумя способами. Во-первых, можно не записывать в edit script те изменения, которые переводят узлы из их состояний в base-дереве в их состояния в online-дереве. Во-вторых, можно запустить алгоритм повторно, используя в качестве base-дерева и одной из веток online-дерево, а в качестве другой ветки merged-дерево — результатом работы алгоритма станет merged-дерево, а в edit script попадут операции, переводящие online-дерево в merged-дерево. Но поскольку существующая реализация алгоритма позволяет в момент добавления операций в edit script определять, изменениями в какой из веток оно вызвано, то реализован первый вариант.

6.4. Обработка конфликта Update/Delete

Конфликт Update/Delete возникает в случаях, когда в одной из веток поддерево было удалено, в другой — в него была добавлена новая информация (например, вставлен или перемещён новый узел или был изменён один из узлов поддерева). Согласно требованию 3 к решению задачи слияния в Comapping механизм синхронизации двух версий карты памяти должен избегать потерь изменений, поэтому конфликт Update/Delete в адаптированной версии алгоритма, в отличие от базовой, следует разрешать, оставляя поддерево в merged-дереве.

6.5. Обработка конфликта Move/Move

Конфликт Move/Move возникает в случаях, когда поддерево в разных ветках было перемещено в разные локации. В базовой версии алгоритма

этот конфликт разрешается копированием поддерева в обе эти локации. Однако такой подход ведёт в случаях вложенных конфликтов, во-первых, к множественному раскопированию одних и тех же узлов, а во-вторых, к дублированию конфликтов (дублируется как минимум каждый конфликт в скопированном поддереве). Это приводит к наличию в merged-версии карты памяти лишних узлов и усложнению последующего разрешения конфликтов пользователем. Поэтому в адаптированной версии алгоритма этот конфликт разрешается по умолчанию помещением поддерева в локацию, в которую он был перемешён в online-дереве (это симметричный конфликт, поэтому разрешаем его в пользу именно online-дерева). Однако, конфликт фиксируется, и у пользователя остаётся возможность разрешить его другим способом.

7. Особенности реализации решения

Механизм слияния карт памяти является частью клиентской части продукта Comapping. Клиентская часть является Flash-приложением на языке haXe. Выбор языка haXe обусловлен его удобством для создания Web-приложений, использующих платформу Adobe Flash. Например, по сравнению с языком ActionScript[22], который также используется для создания таких приложений, haXe поддержан более быстрым компилятором и является многоплатформенным (на haXe можно писать приложения для JavaScript, Flash, PHP, виртуальной машины Neko и др.), т. е. на нём можно создавать и клиентскую (Flash, JavaScript[23]), и серверную (PHP[24] или Neko) часть Web-приложения.

Реализация представленного в данной работе механизма слияния карт памяти основывается на архитектуре, представленной на рис. 9.

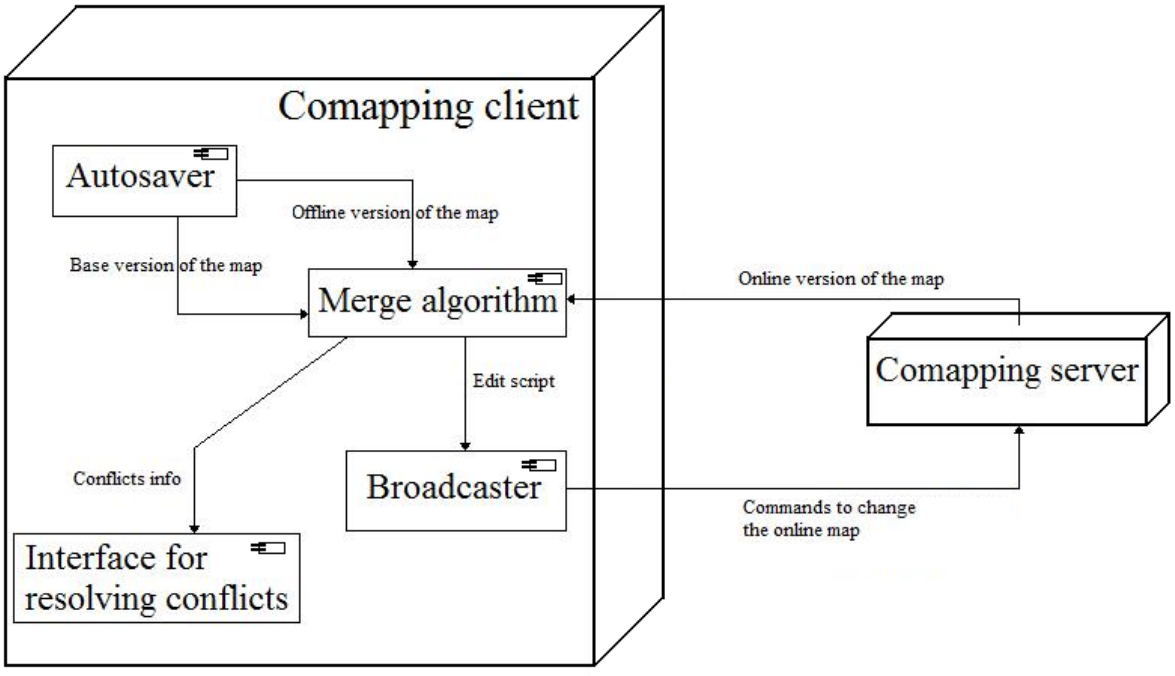

Рис. 9. Архитектура механизма слияния карт памяти

22 http://en.wikipedia.org/wiki/ActionScript
23 http://en.wikipedia.org/wiki/JavaScript
24 http://php.net/

Эта архитектура состоит из следующих компонент:

- Autosaver – механизм сохранения карт памяти локально на компьютере пользователя;
- Merge algortihm — реализация предложенного в работе алгоритма слияния деревьев;
- Broadcaster – механизм трансляции изменений online-версии карты памяти на сервер Comapping;
- Interface for resolving conflicts – пользовательский интерфейс для разрешения возникших при слиянии конфликтов.

В данном случае под компонентой, следуя [2], мы будем понимать независимую часть системы, реализующую определённую функциональность. При этом каждая из обозначенных выше компонент реализуется по-разному. Так, например, Autosaver представлен одним классом, Merge algorithm – это целый пакет классов, а Broadcaster — всего несколько функций.

7.1. Компонента Autosaver

Поскольку браузерные Flash-приложения запускаются в режиме «песочницы» (sandbox), то у них, в отличие от desktop-приложений, нет возможности создавать произвольные файлы на компьютере пользователя. Такие файлы могут понадобится, например, для хранения настроек программы. Однако платформа Flash позволяет хранить некоторые данные пользователя локально в специальных файлах — так называемых Local Shared Objects (наподобие cookies в браузерах). При этом разрешение на использование этих файлов даёт сам пользователь, он также определяет их максимальный размер. Именно в такие файлы Autosaver сохраняет текущее состояние карты памяти. Карта памяти сохраняется сериализованной в формат comap – внутренний формат Comapping,

основанный на XML.

Autosaver был частью Comapping и до появления механизма слияния карт памяти. Однако реализация этого механизма потребовала внесения изменений в Autosaver. Так, теперь первый раз сохранение карты памяти происходит в момент, когда пользователь теряет подключение к Интернету (ранее проверка подключения пользователя к Интернету производилась раз в минуту, и потому первое сохранение происходило не сразу, но сейчас это важно, поскольку необходимо корректно сохранить base-версию карты памяти). В этот момент карта памяти сохраняется в двух экземплярах. Первый из них впоследствии не перезаписывается – этот экземпляр впоследствии используется как base-версия карты памяти, второй перезаписывается каждую минуту — это offline-версия карты памяти.

7.2. Компонента Merge algorithm

Реализация на haXe 3DM-алгоритма, выполненная Николаем Артамоновым, была независимой от контекста (то есть от типа содержимого узлов дерева), в котором алгоритм применяется. Реализация модифицированного алгоритма сохранила эту особенность – все классы в ней также являются параметризованными относительно типа содержимого узлов в сливаемых деревьях. В Comapping этим типом является тип TopicData. TopicData содержит в себе информацию о содержимом узла, а также позволяет задавать древовидную структуру карты памяти, поскольку включает в себя список ссылок на другие объекты типа TopicData. Именно объектами этого типа являются три версии карты памяти, поступающие на вход алгоритму: base-версия и offline-версия, получаемые с помощью компоненты Autosaver, а также online-версия, скачиваемая с сервера. Поэтому первым шагом модифицированного алгоритма является преобразование этих объектов в параметризованные типом TopicData деревья, использующиеся в алгоритме далее.

Связь алгоритма с контекстом описывается с помощью следующих функций:

1) isEqualById – проверяет равенство содержимого узлов по их id и используется при matching;
2) isEqual – проверяет равенство содержимого узлов и используется при слиянии;
3) merge – пробует слить содержимое двух узлов, если не получается, возвращает null.

Алгоритм требует, чтобы эти функции были реализованы для объектов типа содержимого узла деревьев. Соответственно, автором данной работы эти функции реализованы для объектов типа TopicData.

Результатом работы алгоритма являются merged-дерево, edit script, переводящий online-дерево в merged-дерево, и информация о конфликтах, произошедших в процессе слияния.

7.3. Компонента Broadcaster

Edit script содержит список операций, переводящих online-дерево в merged-дерево (вставок и удалений поддерервьев, изменений узлов). Однако для того, чтобы применить эти операции к online-версии карты памяти, они должны быть преобразованы в объекты типа BroadcastCommand, описывающие в Comapping изменения карты памяти, и эти команды в свою очередь должны быть протранслированы на сервер. Этой работой занимается специально созданная для этого компонента Broadcaster.

7.4. Компонента Interface for resolving conflicts

Последним этапом работы механизма слияния является отображение возникших конфликтов и предоставление пользователю возможности их разрешить. Интересной особенностью языка haXe являются

параметризованные перечисления (enumerations). Благодаря им мы можем объединить все типы конфликтов в одно перечисление, в котором для каждого типа конфликта хранятся те параметры, которые необходимы для работы с ним. Для конфликтов типа Update/Update хранится id конфликтного узла карты памяти, а также его содежимое в online- и offline-версиях карты памяти; для конфликтов типа Update/Delete хранится id корня конфликтного поддерева, а также булевский параметр, определяющий в какой из версий карты памяти (online или offline) поддерево было удалено; для конфликтов типа Move/Move хранится id корня конфликтного поддерева, а также информация об альтернативном местоположении этого поддерева. Таким образом, информацию о конфликтах хранится в хеш-таблице (реализована в виде класса IntHash), где id конфликта сопоставляется параметризованный объект из перечисления.

После того, как получена информация о конфликтах, запускается процедура addConflicts, находящая узлы карты памяти, участвующие в конфликтах, и сохраняющая в них id соответствующих конфликтов, что позволяет при отрисовке узла получать конфликт, в котором он участвует. При разрешении конфликтов, информация о них удаляется как из hash-таблицы, так и из узлов, в них участвующих.

Непосредственно интерфейс работы пользователя над разрешением конфликтов детально описан в Приложении 1. Он реализован стандартными средствами Comapping для работы с контекстным меню, с отображением узлов карты памяти и изменением её структуры в online-режиме.

8. Апробация решения

Перед внедрением в новую версию продукта Comapping предложенное решение было оттестировано. Кроме обычного тестирования было создано несколько пакетов тестов, имитирующих следующие часто встречающиеся на практике случаи.

1. Отсутствие изменений online-версии карты памяти.

Такой случай использования механизма слияния представляется одним из наиболее частых, поскольку активная совместная работа над картой памяти происходит обычнов первые дни после создания карты. Затем правки вносятся гораздо реже, и потому велика вероятность, что за время, проведённое пользователем в режиме offline, карта памяти не будет изменена другими пользователями. Кроме того, нередки ситуации, когда при наличии ноутбука пользователь в режиме online только открывает карту памяти с целью продолжить работу над ней в режиме offline – при встречах в кафе или других местах, где наличие Интернета не гарантировано.

В таких случаях задача механизма слияния состоит в том, чтобы правильно «перенести» offline-изменения в online-версию карты памяти. Правильность работы механизма слияния для таких случаев была проверена для 3-х карт памяти небольшого размера (в том числе для карты, состоящей только из одного корневого узла), в каждую из которых было внесено от 5 до 15 изменений.

2. Неконфликтующие изменения online- и offline-версий.

Часто пользователи работают над различными частями карты памяти, и поэтому внесённые ими изменения не конфликтуют друг с другом. В таких случаях корректным результатом является карата памяти, в которой присутствуют как все online-изменения, так и все offline-изменения, а

также отсутствуют конфликты.

Корректность работы предложенного решения в таких случаях была проверена для 3 карт памяти (приблизительно по 50 узлов в каждой), в каждую из версий (online и offline) которых вносилось 5-10 неконфликтующих изменений.

3. Конфликтующие изменения online- и offline-версий.

В таком случае все конфликты должны быть разрешены по умолчанию, но при этом показаны пользователю. Для того, чтобы проверить корректность результата слияния для таких случаев, были предложены 3 карты памяти, в online- и offline-версии которых было внесено по 5 изменений, конфликтующих между собой. В том числе были рассмотрены случаи вложенных конфликтов.

4. Редактирование в режиме offline большой карты памяти (более 1000 узлов).

Отдельным случаем использования механизма слияния является слияние больших карт памяти, поскольку в таких случаях важна не только корректность результата, но и малое время работы механизма слияния.

В рамках исследования слияния больших карт памяти было рассмотрено 2 карты памяти — в 1000 и 2000 узлов (карты памяти бо́льшего размера рассматривать не имеет смысла, т. к. для них Comapping не гарантирует корректной работы). Для каждой из этих двух карт памяти в online- и offline-версиях было внесено по 15 изменений, пять из которых являлись конфликтующими. В обоих случаях процесс слияния завершился корректно и занял не более трёх секунд.

Таким образом, механизм слияния карт памяти в Comapping корректно справился со всеми предложенными примерами и не содержит явных ошибок Тем не менее, после накопления достаточно большой

статистики реального использования предложеннного решения необходимо провести дополнительные исследования корректности работы и удобства использования алгоритма. В частности, должен быть рассмотрен вопрос удобства работы с конфликтами на больших картах памяти. Однако эти задачи выходят за рамки данной дипломной работы.

9. Заключение

В рамках данной дипломной работы были получены следующие результаты:

- сформулирована задача слияния карт памяти после их рассинхронизации в Web-средстве Comapping, а также требования к её решению, обоснована связь данной задачи с задачей слияния деревьев и xml-файлов;
- сделан обзор существующих подходов к слиянию деревьев и xml-файлов и выбран алгоритм, наиболее полно отвечающий специфике Comapping (алгоритм Танкреда Линдхольма, используемый в средстве 3DM);
- выбранный алгоритм был изменён, адаптирован к требованиям Comapping и реализован на языке haXe;
- были реализованы также другие части механизма синхронизации карт памяти в Comapping: инфраструктура, необходимая для внедрения алгоритма, и интерфейс для пользовательской работы над разрешением конфликтов, произошедших в процессе слияния карт памяти;
- механизм синхронизации был внедрён в online-версию Comapping.

Дальнейшими направлениями работы являются внедрение механизма синхронизации карт памяти в desktop-версию Comapping, более формальное описание изменённого алгоритма слияния деревьев и его свойств, а также более детальное исследование пригодности реализованного решения с учётом статистики его использования.

10. Список использованнных источников

[1] Д.В.Кознов. Методика обучения программной инженерии на основе карт памяти // Системное программирование. / Вып. 3, под ред. А.Н.Терехова и Д.Ю.Булычева. СПб.: Изд. СПбГУ, 2008. С. 121-140.

[2] Д.В.Кознов. Основы визуального моделирования. Интернет-университет информационных технологий; БИНОМ. Лаборатория Знаний, 2008. 246 с.

[3] Tony Busen. The Mind Map Book. Penguin Books, 1996. 320p.

[4] Pascal Moli Gérard Oster and Hala Naja-Jazzar. Supporting Collborative Writing of XML Documents. INRIA, 2006. 8 p.

[5] Sudarsah S. Chawathe, Hector Garcia-Molina. Meaningful Change Detection in Structured Data. In Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data. ACM Press, 1997. P. 26–37.

[6] R. la Fontaine. Merging XML files: a new approach providing intelligent merge of XML data sets. In Proceedings of XML Europe 2002 Barcelona Spain. 21 p.

[7] Tancred Lindholm. A 3-way Merging Algorithm for Synchronizing Ordered Trees — the 3DM merging and differencing tool for XML, Master's Thesis, 2005, Helsinki University of Technology. 205 p.

[8] D. Koznov, M. Pliskin. Computer-Supported Collaborative Learning with Mind-Maps. T. Margaria and B. Steffen (Eds.): ISoLA 2008, CCIS Vol. 17, 2008. Springer-Verlag, Berlin Heidelberg, 2008. P. 478-489.

[9] Sebastian Rönnau, Christian Pauli, Uwe M. Borghoff. Merging changes in XML documents using reliable context fingerprints. In Proceeding of the eighth ACM symposium on Document engineering 2008. 10 p.

[10] Pavel Shvaiko, Jérôme Euzenat. A Survey of Schema-based Matching Approaches. Journal on Data Semantics IV, 2005. P. 146-171.

[11] Ukkonen E. Approximate string matching with q-gram sand maximal matches. Theoretical Computer Science, vol. 92, no. 1, 1992. P. 191-211.

[12] http://www.comapping.com

[13] http://www.mind-mapping.co.uk/mind-maps-ideas.htm

Приложение 1. Пользовательский интерфейс механизма слияния карт памяти в Comapping

Базовая функциональность

На отключение от сети Интернет клиентское приложение Comapping реагирует появлением красного индикатора в верхнем левом углу окна браузера, благодаря которому пользователь понимает, что изменения карты памяти больше не сохраняются на сервер. В момент отключения текущее состояние карты сохраняется на компьютере пользователя в двух экземплярах. Первый из них представляет собой base-версию карты, второй экземпляр — это offline-версия карты памяти, она автоматически обновляется каждую минуту вплоть до закрытия Интернет-браузера.

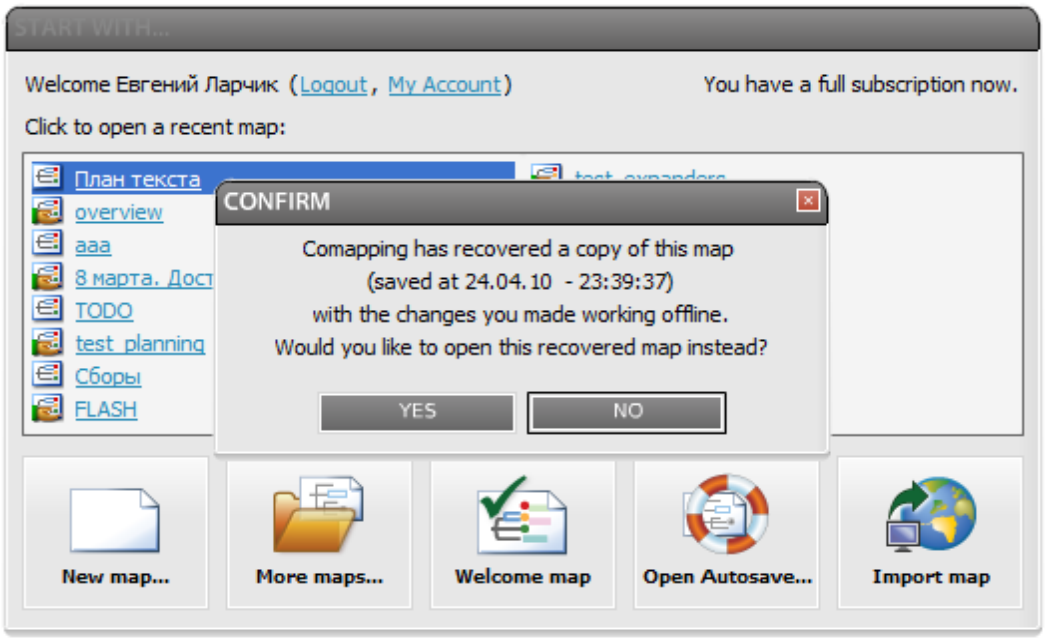

Рис. 10. Оповещение о существовании локальной версии карты памяти

При следующем запуске Comapping в режиме online (в режиме offline Commaping не запустить!) пользователь может просмотреть изменения карты памяти, сделанные им в режиме offline. Это можно сделать двумя

способами. Во-первых, при нажатии в стартовом диалоге Comapping кнопки Open Autosave (см. рис. 10) откроется список карт памяти, когда-либо сохранённых локально, среди которых просто нужно найти нужную и открыть её. Во-вторых, при попытке открыть серверную версию карты пользователь будет оповещён о том, что на его компьютере имеется сохранённая копия, и ему будет предложено посмотреть её — если пользователь откажется от этой возможности, то будет открыта текущая серверная версия карты памяти – см. рис. 10, сообщение Confirm.

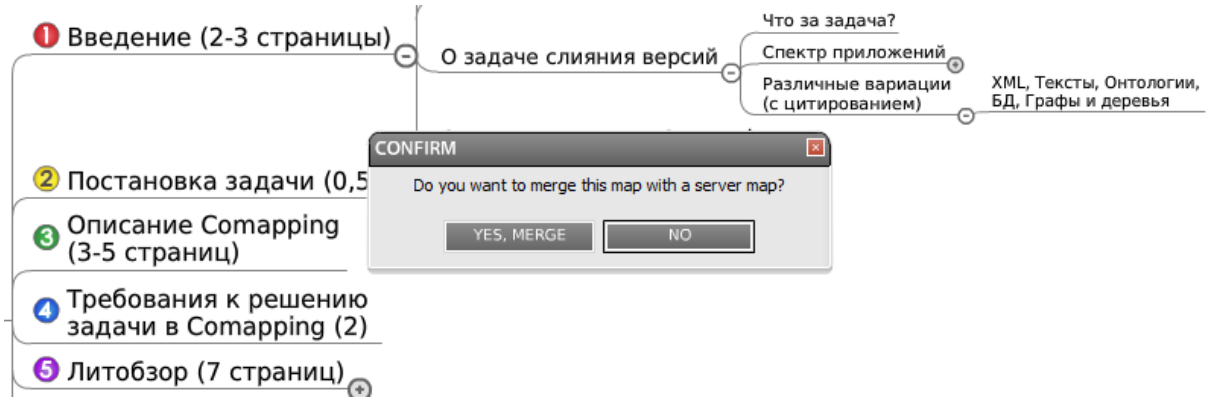

Рис. 11. Предложение о синхронизации локальной
и серверной версий карты памяти

При открытии сохранённой локально версии пользователю будет сразу же предложено слить эту версию с той, которая находится на сервере – см. рис. 11, сообщение confirm. Пользователь может отказаться от слияния – тогда он может продолжить работу над картой памяти в только в режиме read-only. Если же пользователь соглашается, то запускается алгоритм слияния, описанный выше. После окончания работы алгоритма в браузере пользователя открывается итоговая, изменённая серверная версия карта памяти, на которой отображается информация о случивших конфликтах. Все дальнейшие изменения карты памяти, которые будет выполнять пользователь, сохраняются на сервере, то есть работа над картой происходит в обычном режиме.

Организация работы пользователя с конфликтами

Корни поддеревьев, участвующих в конфликтах, отмечаются специальным знаком конфликта (красный восклицательный знак), а в их контекстном меню появляются специальные пункты для его разрешения. Также на карте памяти присутствует кнопка Hide Conflicts, с помощью которой можно разрешить все оставшиеся конфликты разом (рис. 12).

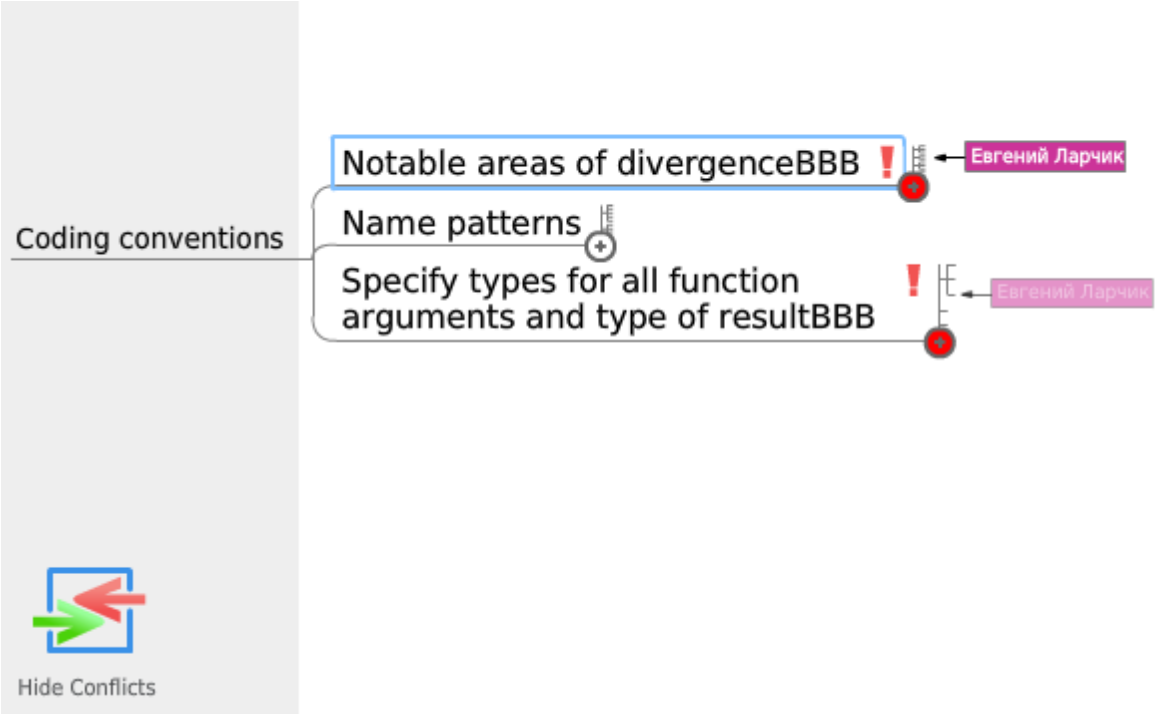

Рис. 12. Показ конфликтов после слияния карт памяти в Comapping

Пользователю могут быть показаны конфликты одного из трёх типов:

- Update/Update (один и тот же узел был изменён в ветках различными способами),
- Move/Move (одно и то же поддерево было перемещено в ветках в разные локации),
- Update/Delete (в одной из веток поддерево было, удалено, а в другой изменено, т.е. в него был добавлен или перемещён хотя бы один узел либо был изменён какой-нибудь из узлов поддерева).

Рассмотрим подробнее типы конфликтов и возможные варианты их разрешения пользователем.

Конфликт Update/Update по умолчанию разрешается в пользу online-версии карты памяти. Однако пользователю предоставляется возможность заменить содержимое узла на то, какое он имел в offline-версии. Рассмотрим сценарий разрешения конфликта Update/Update с помощью примера, изображённого на рис. 13.

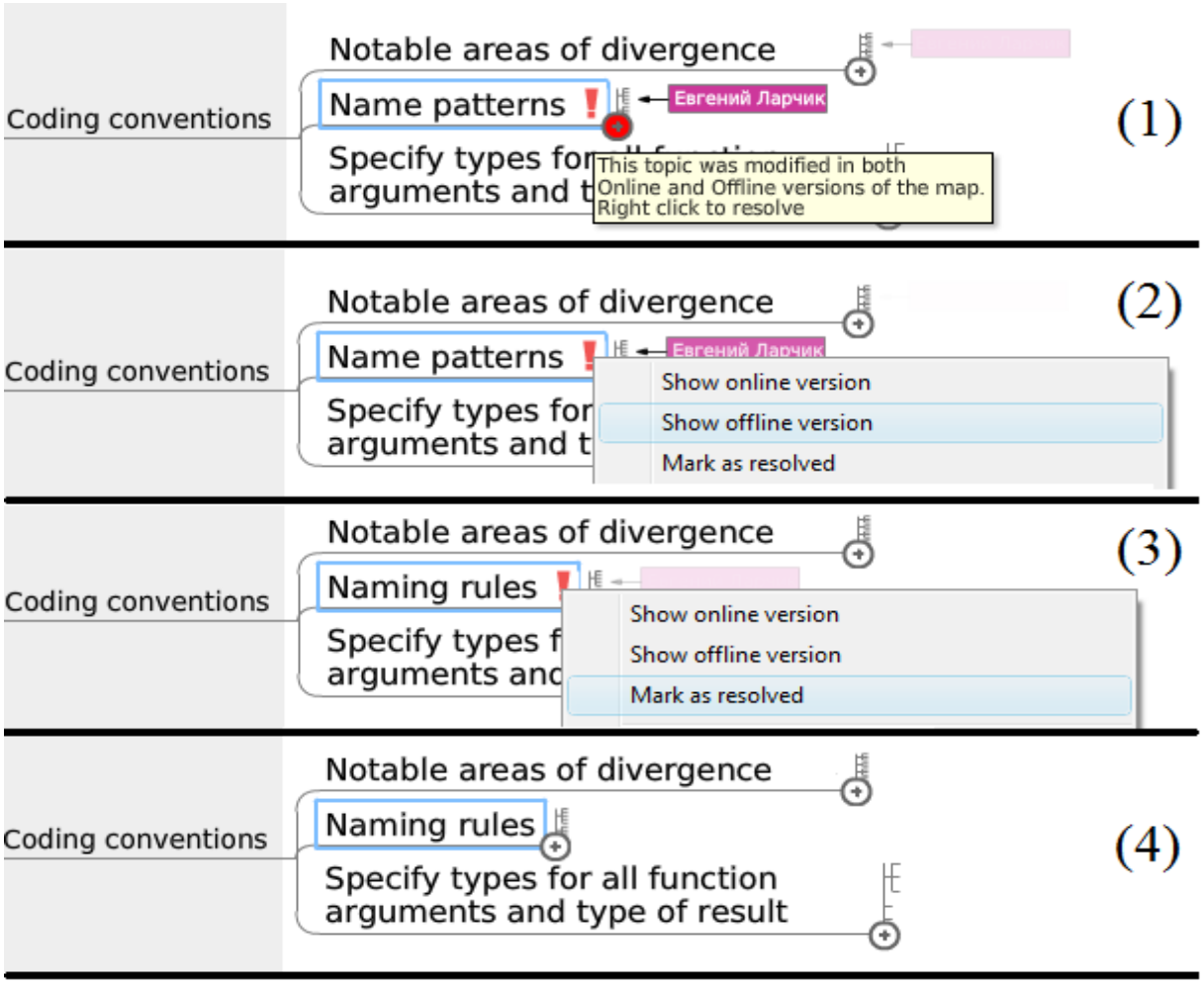

Рис. 13. Пример разрешения конфликта Update/Update:
(1) показ описания конфликта; (2) выбор опции перехода к offline-содержимому узла;
(3) выбор опции разрешения конфликта; (4) итог работы с конфликтом.

На рис.13, (1) представлено описание конфликта, которое появляется при наведении курсором мышки на знак конфликта, а также предложение

воспользоваться контекстным меню для его разрешения. Варианты разрешения конфликта представлены на рис. 13, (2). У пользователя есть возможность увидеть online- и offline-версии содержимого узла, а так же разрешить конфликт, сохранив текущий вариант содержимого как правильный. Поскольку по умолчанию конфликт был разрешён в пользу online-дерева, то выбираем показ offline-содержимого узла. На рис. 13, (3) видно, что содержимое узла изменилось. Теперь мы разрешам конфликт, используя соответсвующий пункт меню. На рис. 13, (4) знака конфликта рядом с узлом уже нет.

Конфликт Update/Delete по умолчанию разрешается в пользу сохранения поддерева в merged-дереве. Пользователю даётся возможность либо удалить поддерево, либо оставить всё, как есть (рис. 14).

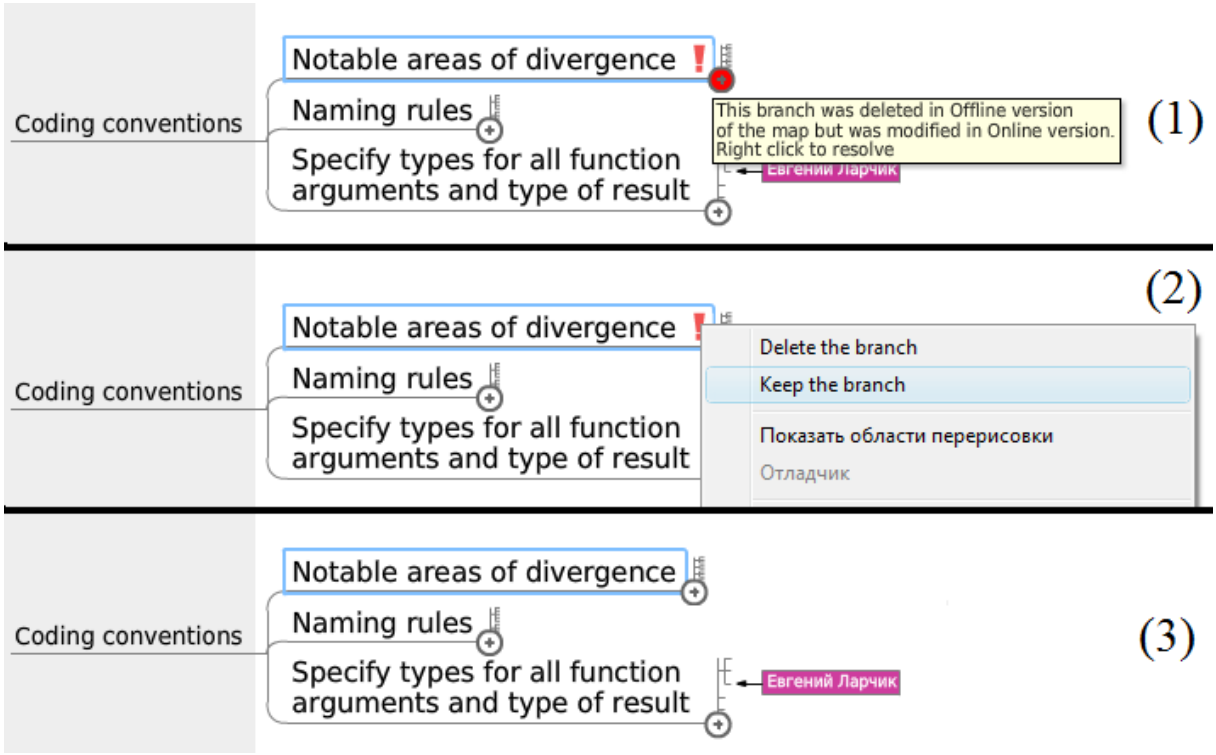

Рис. 14. Пример разрешения конфликта Update/Delete: (1) показ описания конфликта; (2) выбор опции сохранения конфликтного поддерева; (3) итог работы с конфликтом.

Конфликт Move/Move по умолчанию разрешается в пользу online версии карты памяти, т.е. поддерево в merged-дереве находится в той локации, в которую оно было перемещено в online-версии. У пользователя

есть возможность переместить поддерево в альтернативную локацию и, при необходимости, вернуть его обратно (рис. 15)

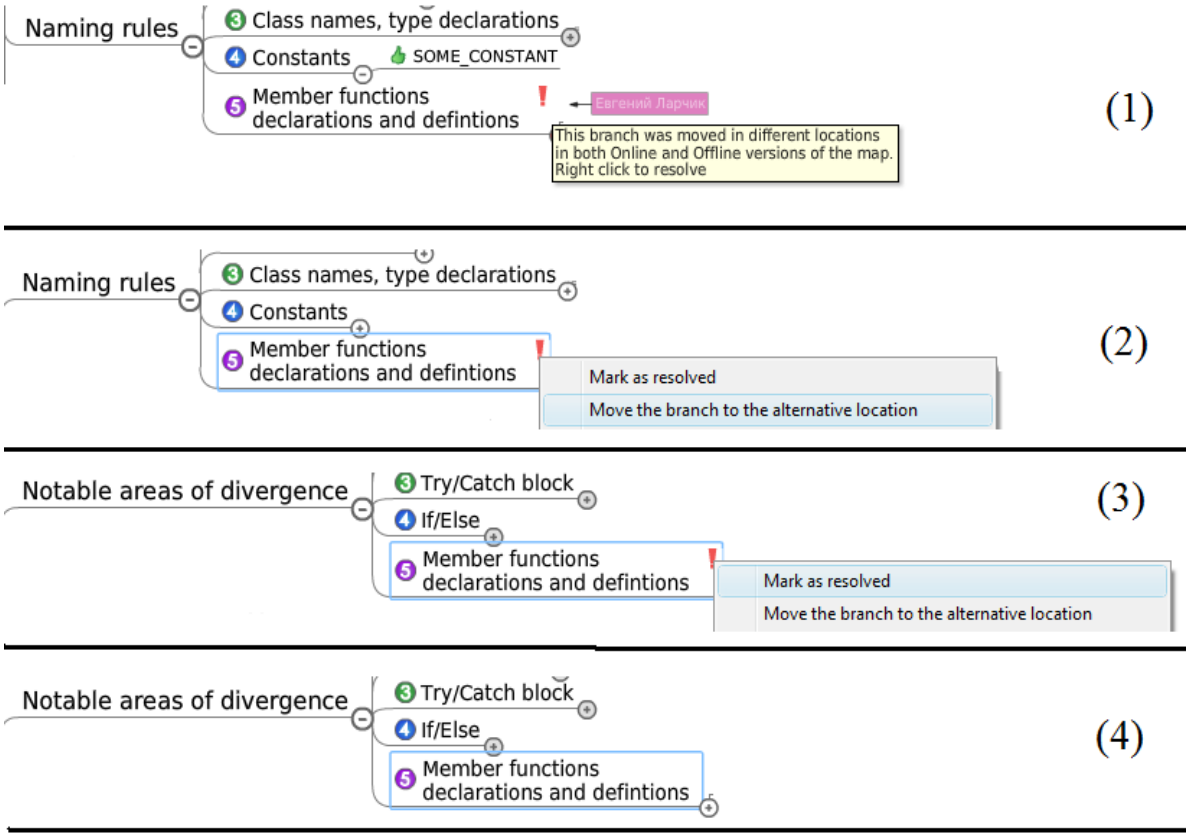

Рис. 15. Пример разрешения конфликта Move/Move:

(1) показ описания конфликта; (2) выбор опции перемещения поддерева к альтернативному родителю; (3) выбор опции разрешения конфликта; (4) итог работы с конфликтом.