

О ЗАДАЧЕ СЛИЯНИЯ КАРТ ПАМЯТИ (MIND MAPS) ПРИ КОЛЛЕКТИВНОЙ РАЗРАБОТКЕ

© 2011 г. Д. Кознов, Е. Ларчик, М. Плискин, Н. Артамонов
Санкт-Петербургский государственный университет
198504 Санкт-Петербург, Петергоф г., Университетский просп., 26
E-mail: dkoznov@yandex.ru
Поступила в редакцию 13.06.2011 г.

В данной работе представлено решение задачи слияния карт памяти (Mind Maps) в контексте процесса коллективной Интернет-разработки таких карт. Эта задача актуальна, например, в ситуации, когда один из разработчиков теряет Интернет-соединение, но продолжает изменять карту памяти локально, и его товарищи также меняют ее. Таким образом, при восстановлении Интернет-соединения возникает потребность в слиянии локальной и серверной копий карты памяти. Задача решена на основе известного алгоритма слияния XML-файлов 3DM, модифицированного в соответствии с особенностями нашей задачи: (1) необходимость двух режимов работы – по умолчанию и с предоставлением пользователю возможности самостоятельно разрешать конфликты; (2) необходимость максимального сохранения изменений, сделанных в поддеревьях при разрешении конфликтов Update/Delete; (3) несимметричность сливаемых деревьев (серверная копия считается более приоритетной); (4) необходимость применения edit-скрипта к исходной версии для сохранения истории изменений; (5) наличие уникальных идентификаторов у узлов сливаемых деревьев (то есть более упрощенная процедура идентификации) и потребностью „разносить“ узлы с одним и тем же значением идентификатора, но сильно разным содержанием.

1. ВВЕДЕНИЕ¹

Задача слияния (merge) двух разных версий одного и того же файлового актива является классической задачей конфигурационного управления (configuration management): два или более человек меняют один и тот же актив независимо друг от друга и после этого необходимо объединить сделанные изменения в рамках единой версии актива. Слияние текстовых файлов является решенной задачей (см., например, работу [1]), и соответствующие алгоритмы входят в многочисленные промышленные средства версионного контроля, например, в CVS/Subversion. Однако для более сложных структур данных – XML-файлов, UML-моделей и т.д. – эта задача требует более

сложных решений, так как простое текстовое слияние часто нарушает структуру актива – дерева, графа и пр.

Задача слияния XML-файлов стала актуальна в силу широкого использования данного формата для представления различных данных, совместно используемых в Интернете. Сюда относится и совместная разработка офисных документов (существуют многочисленные внутренние XML-форматы документов – OpenXML, DocBook и др.), и синхронизация XML-данных в условиях ограниченной сетевой пропускной способности (например, для мобильных устройств) и пр.² В настоящее время разработано несколько алгоритмов решения задачи слияния XML-файлов [2–6]. Существуют также практические реализации

¹Работа выполнена при поддержке гранта Российского фонда фундаментальных исследований (РФФИ) № 11-01-00622-а.

²Подробный обзор различных практических задач, где появляется необходимость сливать XML-файлы, приводится в работе [2].

этой задачи: средство с открытым кодом 3DM [7], XML Diff and Merge – свободно распространяемое Java-средство от компании IBM [8], а также коммерческий продукт DeltaXML [9]. Все они позволяют получить разницу между двумя XML-файлами, а также слить их в один файл. Кроме того, существует система контроля изменений документов So6 [10], которая позволяет нескольким пользователям редактировать XML-файлы одновременно. Таким образом, можно сказать, что в общем виде данная проблема решена. Однако конкретные практические задачи формулируют особые требования к слиянию XML-файлов, что влечет необходимость модификации существующих алгоритмов. Одной такой задачей является слияние карт памяти (Mind Maps), представленных в виде XML, в процессе их коллективной разработки Интернет-средствами.

Карты памяти (Mind Maps) – это графическая нотация и метод, предназначенные для работы со знаниями в самых разных областях: в обучении, бизнесе, при написании книг, статей, в научной деятельности, при планировании личных и семейных мероприятий, при психологическом тестировании и т.д. Данный подход был предложен Тони Бьюзеном в конце 70-х годов прошлого века [11] и доказал свою состоятельность. В связи с этим появилось много программных продуктов, поддерживающих карты памяти – Free-mind [12], MindManager [13] и др.³ В последнее время стали активно развиваться Интернет-средства, поддерживающие коллективную работу с картами памяти – Comapping [15], Mindomo [16] и MindMeister [17]. Эти средства хранят карты памяти на сервере, в виде XML-файлов, предоставляя пользователям возможность раздельного редактирования в online-режиме. Однако до настоящего момента в них отсутствовала возможность слияния двух версий одной и той же карты памяти. Данная возможность необходима, например, когда группа пользователей работает с одной картой памяти в online-режиме и один из них теряет подключение к Интернету (например,

³Полный список программных средств поддержки карт памяти можно найти здесь [14].

он находится в дороге и работает на своем ноутбуке) и продолжает работу над картой памяти локально. В это же время его товарищи, также работая с этой картой, изменяют серверную версию. После восстановления Интернет-подключения необходимо слить вместе две версии.

В рамках данной работы известный алгоритм слияния XML-файлов 3DM [3] был применен для решения задачи слияния карт памяти в продукте Comapping [15]. Алгоритм был модифицирован так, чтобы были удовлетворены специфические требования Comapping: обеспечение работы алгоритма по умолчанию для обычных пользователей и, вместе с тем, обеспечение „продвинутых“ пользователей возможностями проанализировать конфликты слияния и разрешить их „вручную“ наиболее предпочтительным способом. В алгоритме 3DM была также изменена процедура идентификации (matching) одинаковых узлов в разных ветках, так как в отличие от 3DM в нашем случае все узлы имеют уникальные идентификаторы. С другой стороны, мы ввели дополнительную метрику идентификации с пороговым значением для определения „похожести“ модифицированных узлов, поскольку, не смотря на одинаковые идентификаторы, пользователи могли перестать отождествлять сильно изменившиеся узлы в локальной и серверной копиях. Также мы внесли изменения в процедуры обработки конфликтов слияния Move/Move, Update/Update, Update/Delete.

Статья организована следующим образом. В разделе 2 делается обзор карт памяти и продукта Comapping. В разделе 3 в общем виде обсуждается задача слияния XML-файлов, вводятся основные термины, а также обсуждается метод определения Q-расстояния между строками, который часто используется при слиянии текстовой информации в узлах сливаемых XML-файлов. В разделе 4 уточняется постановка задачи. В разделе 5 дается обзор имеющихся средств для слияния XML-файлов и обосновывается выбор алгоритма 3DM как наиболее подходящего с учетом специфики нашей задачи. Наконец, в этом разделе были описаны те расширения, которые мы внесли в 3DM-алгоритм.

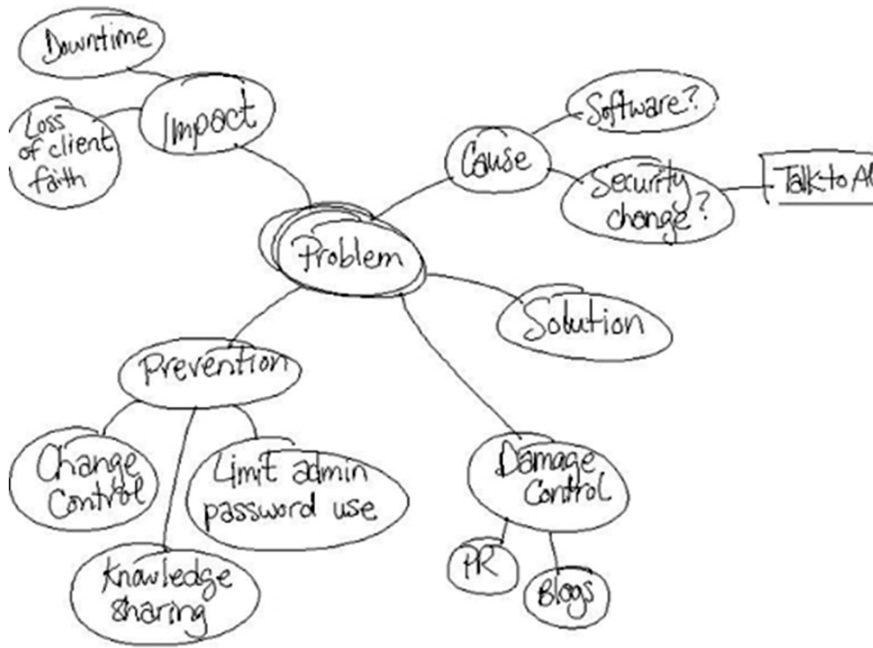


Рис. 1. Пример карты памяти.

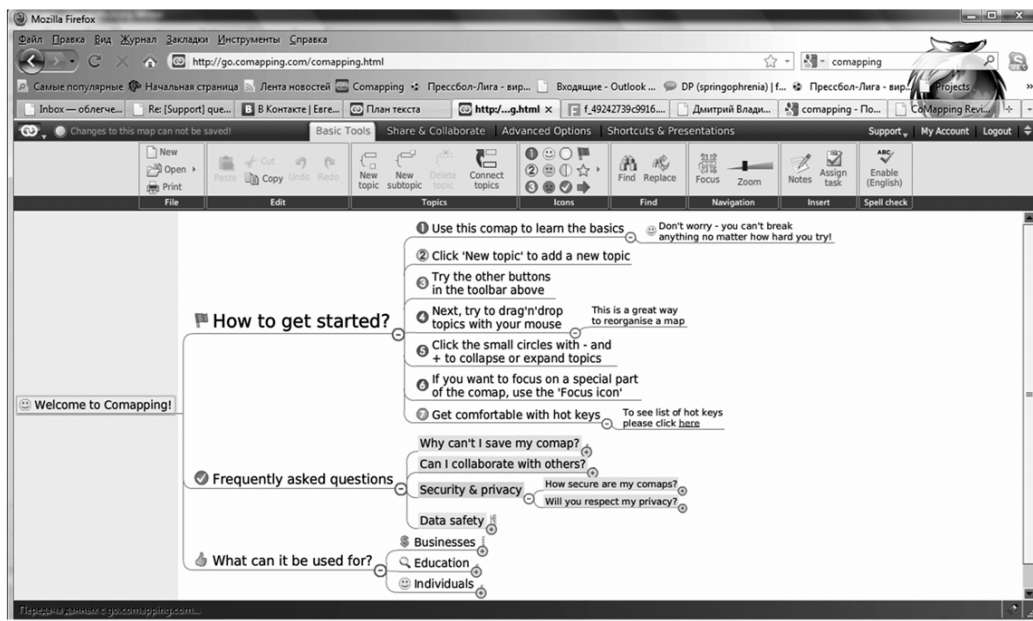


Рис. 2. Пример карты памяти в Comapping.

2. ТИПИЧНЫЕ СРЕДСТВА РАБОТЫ С КАРТАМИ ПАМЯТИ НА ПРИМЕРЕ СОМАРРИНГ

В начале 70-х годов прошлого века английским психологом Тони Бьюзоном была предложена техника работы с информацией, основанная на использовании карт памяти

(Mind Maps) [11]. 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). Карты памяти имеют

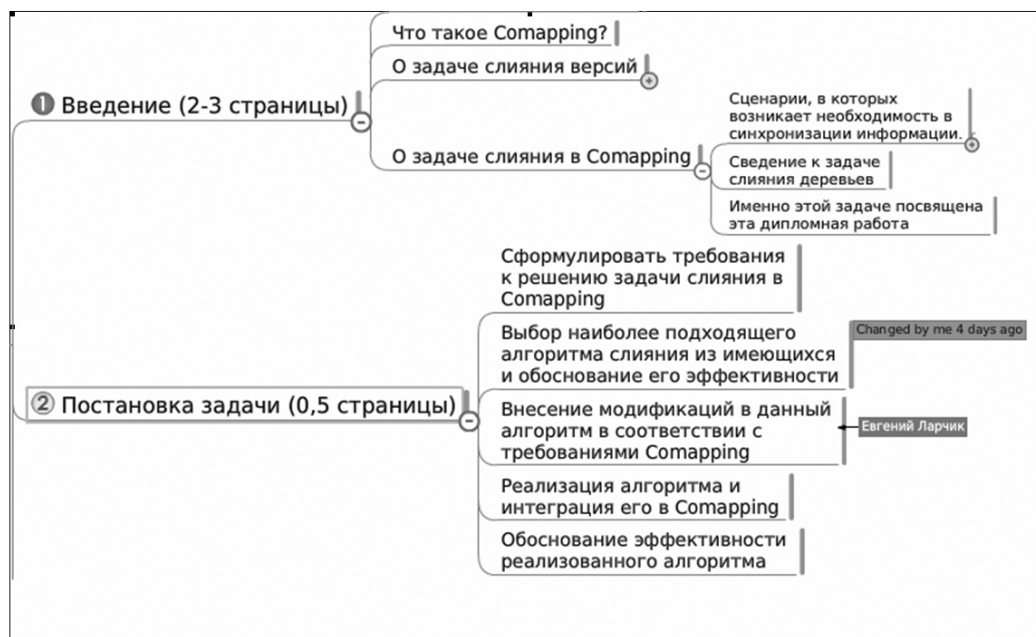


Рис. 3. Совместная работа над картой памяти в Comapping.

следующие достоинства – 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 в процессе создания таких карт и т.д.

Далее мы опишем Web-приложение Comapping – типичный инструмент в этой области. Читателю будет проще познакомиться с общими принципами и проблемами программной разработки карт памяти на основании описания конкретного примера инструмента.⁴

Для представления карт памяти Comapping использует древовидную нотацию, в которой уточнение концепций происходит слева направо (так называемый left-to-right mindmapping), и, соответственно, главный элемент карты памяти находится левее остальных (см. рис. 2). Такая нотация в сочетании с алгоритмом автоматического перераспределения элементов на экране облегчает чтение и понимание карты.

⁴Продукт Comapping является результатом сотрудничества датской компании Area9 и петербургской компании ЗАО „ЛАНИТ-ТЕРКОМ“, а также Санкт-петербургского государственного университета. Исследования по применению продукта в образовательном процессе поддержаны компанией Hewlett-Packard.

Comapping также 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 карты, а также публикацию карт в блогах и на сайтах; печать карты любых размеров; экспортирование карты памяти в форматы PDF, HTML, RTF, SVG, импорт/экспорт в форматы других средств работы с картами памяти (FreeMind, MindManager).

Кроме этого, Comapping фокусируется на предоставлении максимально удобных средств для совместной работы пользователей: несколько пользователей одновременно могут открыть одну и ту же карту для просмотра или редактирования, при этом можно видеть, какие участки карты редактируют в данный момент другими пользователями (рис. 3), можно также общаться во встроенном чате; есть средства для объединения пользователей в группы, а также для изменения прав доступа к карте памяти сразу всем участникам группы; существует механизм email-оповещений об изменениях карты памяти другими пользователями; хранится история изменений карты, которая позволяет отследить, кем и когда была изменена

та или иная часть карты памяти.

Более детально с Comapping, а также с его использованием в образовании, можно ознакомиться в [18, 19].

3. О ЗАДАЧЕ СЛИЯНИЯ XML-ФАЙЛОВ

Задача слияния двух версий одного и того же информационного актива непосредственно связана с задачей синхронизации, которую можно сформулировать так. Предположим, что имеется два набора данных. Тогда *синхронизация (synchronization)* – это процесс их изменения таким образом, чтобы они стали идентичными в тех местах, где они описывают одну и ту же информацию. Конкретные требования к процедуре синхронизации зависят от контекста, в котором решается эта задача. После синхронизации каждый из наборов данных продолжает своё существование. Они могут отличаться один от другого в тех своих частях, которые описывают семантически различную информацию. Задачи синхронизации возникают там, где существуют два и более независимых процесса работы над более-менее разными данными, тем не менее, имеющими общую семантическую часть. Например, в Comapping существует задача синхронизации локальных версий одной и той же карты памяти, находящихся на компьютерах разных пользователей с тем, чтобы быстро отображать изменение, сделанное любым из них, у всех остальных пользователей. Также синхронизацией является задача циклической разработки ПО (Round Trip Engineering) на основе модельно-ориентированного подхода, когда, например, синхронизируются диаграммы классов и соответствующий им программный код – ни то, ни другое нельзя просто сгенерировать повторно, чтобы не потерять уникальную информацию, принципиально не содержащуюся в другом активе [20].

Задача *слияния (merge)* является синхронизацией с тем ограничением, что в итоге обе версии данных становятся идентичными: их разные части объединяются, похожие – сливаются, и в итоге одна из версий перестает существовать. Интерес к задаче слияния XML-файлов возник в начале 2000-х годов и был обусловлен тем, что XML-формат стал широко использоваться

в различных программных средствах для хранения данных. Задача слияния XML-файлов тесно связана с более ранней задачей *нахождения разницы (change detection)* между двумя XML файлами или, в более общем виде, – между двумя деревьями [21], [22], – поскольку любой алгоритм слияния так или иначе сравнивает сливаемые файлы и находит их разницу. Чаще всего эта разница представляется в виде списка операций, которые переводят один из файлов в другой; этот список называют *edit-скриптом (edit-script)* [21]. Важной частью слияния XML-файлов является *идентификация (matching)* – процедура нахождения одних и тех же узлов в разных деревьях [21], [22].

Вот список операций над XML-файлами, которые обычно рассматривают в задаче слияния: *добавить (Insert)* – вставка нового узла в дерево, *удалить (Delete)* – удаление поддерева с корнем в заданном узле, *изменить (Update)* – изменение содержимого узла, *перенести (Move)* – „перевешивание“ поддерева с корнем в заданном узле к новому родителю.

Еще одним важным аспектом слияния XML-файлов является *конфликт (conflict)* – ситуация, возникающая при работе алгоритма, когда нет возможности однозначно определить, какой из нескольких возможных вариантов предпочесть. Как указывалось в [3], алгоритм не должен быть слишком „умным“, самостоятельно разрешая все такие ситуации – окончательный ответ здесь должен дать пользователь, проанализировав ситуацию и выбрав наилучший вариант. Конфликты бывают следующих типов: *Update/Update*, *Move/Move*, *Delete/Update*. Конфликт *Update/Update* происходит тогда, когда один и тот же узел был изменён в разных версиях различными способами. Конфликт *Move/Move* происходит тогда, когда одно и то же поддерево было перемещено в разных версиях в разные локации. Конфликт *Delete/Update* происходит тогда, когда в одной из версий поддерево было, удалено, а в другой изменено (т.е. в него был добавлен или перемещён хотя бы один узел, либо был изменён какой-либо из существующих узлов). Отметим, что конфликты могут быть вложенными, то есть когда узел или поддерево, участвующие в конфликте, являются

частью большего поддерева, участвующего в другом конфликте.

Существует две основных парадигмы слияния XML-файлов – *3 way merge* и *2 way merge*. Алгоритмы первого вида используют при слиянии две „разошедшиеся“ версии одного файла, а также исходный файл (в дальнейшем – *базовая версия*), алгоритмы второго вида базовую версию не используют. Результат слияния (в дальнейшем мы будем называть его *целевой версией*) в первом случае получается более качественным, однако *2 way merge* алгоритмы также необходимы в том случае, когда базовая версия недоступна или ее вообще не может быть (например, при слиянии онтологий [23]).

При решении задачи идентификации узлов в рамках алгоритма слияния часто используют специальную метрику – расстояние между текстовыми строками идентифицируемых узлов, и чем меньше это расстояние, тем более похожими считаются строки и, соответственно, содержащие их узлы. Самый простой способ задать эту метрику – определить для каждого алфавитного символа количество его вхождений в каждую из строк, а также подсчитать модуль разности и сложить полученные по всем символам числа. Однако оказывается, что чем больше длина сравниваемых строк, тем выше вероятность назвать похожими строки, которые существенно различаются. Поэтому в [24] предлагается использовать другую метрику – Q-расстояние между строками. Q-расстояние определяется аналогично приведенной выше простой метрике, только подсчитывается количество появлений в строке не отдельных символов, а всевозможных подстрок длины Q. Применяя этот метод, можно использовать различные значения Q в зависимости от размера сравниваемых строк (чем длиннее строки, тем больше должно быть Q). Более формально процесс вычисления Q-расстояния между двумя строками описывается так: для каждой строки строится ассоциативный вектор, в котором всевозможным подстрокам длины Q сопоставляется количество вхождений этой подстроки в исходной строке; Q-расстояние равно модули разности этих векторов. Q-расстояние равно 0, если строки идентичны.

4. УТОЧНЕНИЕ ПОСТАНОВКИ ЗАДАЧИ

Наше решение должно позволить мобильным партнерам разрабатывать карты памяти, при этом партнеры могут терять Интернет-соединение во время работы. При восстановлении Интернет-соединения как раз и возникает задача слияния двух версий – одной, которая образовалась на локальном компьютере, второй – серверной. Мы сформулировали следующие требования к решению задачи слияния двух версий одной карты памяти.

1. Процедура слияния должна иметь полностью автоматический режим для пользователей, которые не могут или не хотят разбираться в ее деталях. Это значит, что должен существовать способ разрешать все конфликты, обнаруженные при слиянии, автоматически, то есть по умолчанию.
2. Процедура слияния должна иметь „продвинутый“ режим использования, то есть информация о конфликтах должна сохраняться и показываться по требованию пользователя, чтобы у него была возможность разрешить конфликты способом, отличным от принятого по умолчанию.
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 сохранения изменённой части.
4. Локальная и серверная копии не являются в нашем случае симметричными – при разрешении конфликтов предпочтение отдается серверной копии. Мы считаем, что пользователи, у которых Интернет-соединение не прерывалось, должны получить минимум сюрпризов после выполнения процедуры слияния. Далее мы будем использовать термин *локальная версия* для обозначения версии карты памяти, с которой пользователь работает локально, после потери соединения с

Интернетом, а термин *серверная версия* – чтобы обозначить версию карты памяти, с которой работают пользователи, сохранившие Интернет-соединение.

5. Поскольку мы предполагаем встроить наше решение в Comapping, то необходимо учесть тот факт, что последний заботится о сохранении истории изменений карты памяти (она позволяет проследить, как, кем и когда изменялась карта памяти, что важно для любого совместно редактируемого документа), и поэтому нельзя сохранить результат слияния, просто удалив предыдущую серверную версию, т.к. иначе история всех предыдущих изменений станет бесполезной. В связи с этим необходимо получить edit-скрипт для перевода серверной версии в целевую.
6. Реализация *3 way merge*, поскольку в нашем случае базовая версия карты памяти доступна для алгоритма слияния, и *3 way merge* гарантирует лучшее качество результата.

Часть этих требований относится к разработке пользовательского интерфейса целевого программного сервиса (п.п. 1, 2), а часть – к модификации алгоритма слияния XML-файлов (п.п. 3–6).

5. АЛГОРИТМ СЛИЯНИЯ XML-ФАЙЛОВ

5.1. Обзор существующих подходов

Итогом интереса к задаче слияния XML-файлов стало появление нескольких различных законченных подходов [2–6], а также ряда программных средств [7–10], решающих данную задачу. После этого многие исследователи и практики высказали мнение, что задачу слияния XML-файлов можно считать решённой, т. к. в большинстве случаев при возникновении необходимости в слиянии достаточно адаптировать под свои требования одно из имеющихся средств [25]. Кратко рассмотрим эти средства.

Технология SO6 [4] является подходом для 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 [10]. Данный подход может использоваться и для слияния деревьев: можно записывать все операции, изменявшие обе версии, а потом синхронизировать эти операции между собой. Однако подобные подходы на практике оказываются довольно трудоемкими, поскольку проще согласовывать результат, а не многочисленные атомарные операции, которые к ним привели – существуют взаимнообратные операции (добавил-удалил одно и то же) и др. случаи, когда последовательность действий легче воспринимать как одно целое. Кроме того, в рамках работы [4] реализованы лишь операции „добавить“ и „удалить“, но нет операции „перенести“.

Подход DeltaXML [5] производит слияние XML-файлов на основе построения 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. Подход поддерживает как 2 way merge так и 3 way merge. Однако он в большей степени ориентирован на XML-данные, а не на XML-документы: основное отличие здесь, как считают авторы, заключается в редком использовании операции „перенести“. Соответственно, DeltaXML не поддерживает эту операцию. Кроме того, в описании подхода нет спецификации самого алгоритма, в частности, промежуточного XML-файла, поскольку DeltaXML является коммерческим продуктом [9]. Поэтому для наших целей этот подход оказывается непригоден.

Подход, описанный в работе [6], предлагает использовать при слиянии XML-файлов следующую технику: сначала алгоритм находит дельту между исходным файлом и одной из версий в виде edit-скрипта, и потом применяет этот edit-скрипт ко второй версии. Главная сложность здесь в том, что контекст выполнения операции мог измениться во второй ветке. Для решения этой проблемы для каждой операции в момент создания edit-скрипта сохраняется еще и её контекст (отпечаток),

который потом помогает определить точное место во второй версии, где она должна быть применена. К сожалению, детали реализации подхода и полную спецификацию алгоритма не удалось найти в свободном доступе, кроме того, подход не рассматривает операцию перемещения, поэтому для наших целей он оказался непригодным.

5.2. 3DM-алгоритм

В работах [2], [3] представлен 3DM-алгоритм для слияния деревьев на основе подхода 3 way merge. В [7] можно найти программное средство с открытым кодом, реализующее данный подход на языке Java. В основе алгоритма 3DM лежит следующая идея. Для каждого узла двух сливаемых деревьев рассматриваются изменения, произошедшие с ним относительно базового дерева, и целевое дерево строится применением к базовому всех этих изменений.

3DM-алгоритм производит слияние XML-файлов в два шага. На первом шаге происходит сопоставление (идентификация) узлов сливаемых деревьев с узлами в базовом дереве. Анализируя эту информацию становится понятно, каким образом изменились сливаемые деревья относительно базового: если узлу в сливаемом дереве не сопоставлен узел в базовом, то он был вставлен и т.д. (алгоритм идентифицирует все возможные операции: вставку/изменение поддерева, удаление/перемещение/копирование поддерева). На втором шаге на основе информации об этих изменениях производится непосредственно слияние деревьев. Обход сливаемых деревьев производится так, что сопоставленные узлы посещаются одновременно. Построение целевого дерева происходит „в ширину“: на основе списков детей сопоставленных узлов в базовом и двух сливаемых деревьев с помощью специальных эвристик строится список детей узла в целевом дереве.

Алгоритм 3DM был выбран нами в качестве базового в силу следующих причин. Во-первых, он использует подход 3 way merge, во-вторых, он рассматривает все операции над деревьями, которые могут быть произведены в Comapping, и, в-третьих, он является открытым. Однако, так как наша задача обладает рядом специфических

требований, то оказалось необходимым изменить выбранный нами 3DM алгоритм. Ниже эти изменения детально описаны.

5.3. Изменение процедуры идентификации

Поскольку каждый узел карты памяти в Comapping имеет уникальный идентификатор (id), то процедура идентификации по сравнению с 3DM-алгоритмом упрощается: у нас при объединении версий сопоставляются друг другу только узлы с одинаковыми id. С другой стороны, id узлов карты памяти не видны пользователям карты, поэтому в процессе редактирования локальной и серверной копий пользователи могут 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 процедуре идентификации мы дополнительно проверяем „похожесть“ сопоставленных по id узлов с помощью определённой метрики. Идея идентификации с метрикой при слиянии различных структур данных не является новой и подробно обсуждается, например, в контексте слияния онтологий и баз данных [23].

Предлагаемая здесь метрика „похожести“ узлов определяется тремя следующими параметрами: местоположением узла в ветке (изменилось ли оно относительно местоположения узла в базовой версии), сходством содержимого узлов (задается числом в интервале $[0, 1]$) и „похожестью“ списков детей узлов (задается числом 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 [24]. Мы используем различные значения Q в зависимости от средней длины сравниваемых слов: Q равно 2, если длина строки меньше 50 символов, Q равно 4, если длина от 50 до 150 символов, Q равно 6, если длина от 150 до 500 символов, и Q равно 8, если длина больше 500 символов. „Похожесть“ списка детей узлов мы определяем как $2 \cdot M/N$, где M – это число пар

детей, совпадающих по id, N – общее количество детей у обоих узлов. Пустые списки детей считаем похожими со значением 1.

В зависимости от значения первого параметра мы определяем нижние пределы для двух других параметров, и если „похожесть“ содержимого узла, и „похожесть“ списка детей опускается ниже этих пределов, то узлы не идентифицируются как одинаковые. Значения пределов следующие: если узел был перемещён в ветке, то пороговое значение определяется равным 0.3 для обоих параметров, если не был перемещён – то 0.1. Данные коэффициенты были выбраны 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 положительной идентификации. В целом коэффициенты должны быть занижены, поскольку наша цель – идентифицировать отрицательно только очевидно „разошедшиеся“ узлы.

5.4. Неравнозначность локальной и серверной копий

ЗДМ-алгоритм не делает различий между двумя сливаемыми файлами, поэтому он часто в процессе своей работы меняет их местами, когда это ему удобно. В нашем случае этот неприемлемо. Поэтому, в частности, симметричные конфликты (*Move/Move*, *Update/Update*), а также ситуацию спорного порядка узлов мы разрешаем всегда в пользу серверной версии. Таким образом, в случае конфликта *Update/Update* узлу присваивается его содержимое из серверной копии, а в случаях, когда для двух узлов невозможно определить, в каком порядке они должны следовать в списке детей, то они сохраняют такой же порядок следования, как в серверной версии.

5.5. Разрешение конфликта *Move/Move*

ЗДМ-алгоритм разрешает конфликт *Move/Move* слиянием поддеревьев из двух версий и копированием результата в обе локации при порождении целевой версии.

Однако при вложенных конфликтах такой подход ведёт, во-первых, к множественному раскопированию одних и тех же узлов, а во-вторых, к дублированию конфликтов (дублируется, как минимум, каждый конфликт в скопированном поддереве). Поэтому мы разрешаем этот конфликт по умолчанию помещением „слитого“ поддерева только в одну локацию – ту, в которой он оказался в серверной версии. Однако у пользователя остаётся возможность разрешить его другим способом.

5.6. Обработка конфликта *Update/Delete*

Как уже говорилось выше, наш механизм синхронизации двух версий карты памяти должен избегать потерь изменений, поэтому при разрешении конфликта *Update/Delete* мы, в отличие от ЗДМ-алгоритма, по умолчанию оставляем поддерево в целевой версии. Пользователь, опять таки, может выбрать и вторую альтернативу разрешения данного конфликта.

5.7. Изменения в *edit-скрипте*

ЗДМ-алгоритм порождает *edit-скрипт*, переводящий исходную версию карты в целевую. Нам такой подход не годится, как было указано выше. Достичь нужного нам решения можно двумя способами. Во-первых, можно не записывать в *edit-скрипт* те изменения, которые соответствуют изменениям узлов из исходной версии в серверную. Во-вторых, можно запустить алгоритм повторно, используя в качестве исходной версии серверную версию, а в качестве локальной версии – целевую. Результатами работы такого повторного запуска станет целевой дерево, а в *edit-скрипт* попадут операции, переводящие серверную версию в целевую. Но поскольку средства *Comparing* легко позволяют в момент добавления операций в *edit-скрипт* определять, изменениями в какой из версий оно вызвано (локальной или серверной), то был выбран первый вариант.

6. ЗАКЛЮЧЕНИЕ

В данной работе представлено решение задачи слияния карт памяти в процессе

групповой разработки через Интернет. Известный алгоритм слияния XML-файлов 3DM адаптирован к решению данной задачи и встроен в продукт Comapping, предназначенный для коллективной работы с картами памяти через Интернет. С представленным решением, как и с продуктом в целом, можно ознакомиться на сайте Comapping.⁵

СПИСОК ЛИТЕРАТУРЫ

1. *Johnson M.K.* Diff, patch, and friends // Linux Journal, August 1996.
2. *Lindholm T.* 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.
3. *Lindholm T.* A three-way merge for XML documents. ACM Symposium on Document Engineering. 2004. P. 1–10.
4. *Oster G., Skaf-Molli H., Molli P., Naja-Jazzar H.*: Supporting Collaborative Writing of XML Documents. ICEIS (4) 2007. P. 335–341.
5. *La Fontaine R.* Merging XML files: a new approach providing intelligent merge of XML data sets // Proceedings of XML Europe 2002 Barcelona Spain. 21 p.
6. *Rönnau S., Christian Pauli C., Borghoff U.M.*: Merging changes in XML documents using reliable context fingerprints. ACM Symposium on Document Engineering 2008. P. 52–61.
7. 3DMJavaImplementation: <http://www.cs.hut.fi/~ctl/3dm/>
8. XML Diff and Merge: <http://www.alphaworks.ibm.com/tech/xmldiffmerge>
9. DeltaXML: <http://www.deltaxml.com/>
10. SO6 toolset: <http://dev.libresource.org/>
11. *Busen T.* The Mind Map Book. Penguin Books, 1996. 320 p.
12. FreeMind <http://freemind.sourceforge.net>
13. Mind Manager <http://www.mindjet.com/products/mindmanager-9-win/overview>
14. Mind Maps Tools <http://www.mindjet.com/products/overview>
15. Comapping www.comapping.com
16. Mindomo <http://www.mindomo.com/>
17. Mindmeister <http://www.mindmeister.com>
18. *Koznov D., Pliskin M.* Computer-Supported Collaborative Learning with Mind-Maps. T. Margaria and B. Steffen (Eds.): ISoLA 2008, CCIS V. 17. 2008. Springer-Verlag, Berlin Heidelberg. 2008. P. 478–489.
19. *Кознов Д.В.* 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.
20. *Hettel T., Lawley M., Raymond K.* Model Synchronisation: Definitions for Round-Trip Engineering. A. Vallecillo, J. Gray, A. Pierantonio (Eds.): ICMT 2008, LNCS 5063, Springer-Verlag, Berlin, Heidelberg. 2008. P. 31–45.
21. *Chawathe S.S., Rajaraman A., Garcia-Molina H., Widom J.* Change detection in hierarchically structured information // ACM SIGMOD International Conference on Management of Data (SIGMOD). 1996. P. 493–504.
22. *Sudarsah S. Chawathe, Hector Garcia-Molina.* Meaningful Change Detection in Structured Data // Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM Press. 1997. P. 26–37.
23. *Shvaiko P., Euzenat J.* A Survey of Schema-based Matching Approaches. Journal on Data Semantics IV, 2005. P. 146–171.
24. *Ukkonen E.* Approximate string matching with q-gram sand maximal matches. Theoretical Computer Science. V. 92. N 1. 1992. P. 191–211.
25. <http://useless-factor.blogspot.com/2008/01/matching-diffing-and-merging-xml.html>, <http://stackoverflow.com/questions/2222548/3-way-xml-merge-algorithm>

⁵Имеется бесплатная тридцатидневная полнофункциональная версия продукта.