Титульный

ОГЛАВЛЕНИЕ

НОРМАТИВНЫЕ ССЫЛКИ

Нормативные ссылки

ОПРЕДЕЛЕНИЯ

Класс является важным понятием объектно-ориентированного программирования. Под классом подразумевается некая сущность, которая задает некоторое общее поведение для объектов.

Объект это некоторая сущность, обладающая определённым состоянием и поведением, имеет заданные значения свойств (атрибутов) и операций над ними (методов).

Писатель это участник редактирования общего текста, пользователь системы совместного редактирования текстов.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ИС — Информационная система.

ПО — Программное обеспечение.

ОЗУ — Оперативное запоминающее устройство.

ОС — Операционная система.

СКВ — Система контроля версиями.

ЭВМ — Электронно-вычислительная машина.

ТЗ — Техническое задание.

БД — База данных.

ВВЕДЕНИЕ

В настоящее время интернет-технологии позволяют обмениваться информацией друг с другом с большой скоростью. Несмотря на высокую развитость технологий, подавляющее большинство подходов к обеспечению синхронизации текста между людьми всё так же остается весьма примитивными: основным способом является простая передача копий файлов.

Данный способ имеет множество недостатков, например каждый раз когда необходимо поделиться новой версией файла, автор передает копию текущего состояния документа и таких копий может накопиться со временем большое количество. Такой подход вносит путаницу и потребность в тщательном контроле над различными версиями файлов.

К счастью, существует альтернативный подход, который заключается в синхронизации текста между пользователями программы в режиме реального времени, который освободит от необходимости пересылки копий файлов, даст возможность редактировать один и тот же текст двум или более людям.

Задача написания и редактирования текстов одновременно несколькими пользователями сегодня является весьма актуальной. Этот факт ярко подтверждается тем, что на сегодняшний день существуют такие интернет-сервисы, как «Google Docs». «Google Docs», или «Документы Google» — это бесплатный онлайн-офис, который включает текстовый процессор, разрабатываемый компанией Google [1].

Несмотря на все преимущества, такие сервисы, как «Google Docs» обладают одним серьезным недостатком: чтобы воспользоваться данной услугой все пользователи должны иметь стабильный доступ к глобальной сети Интернет, а также возможность пользоваться продуктами компании «Google». Однако участники редактирования документа могут находиться в одном помещении (например, коллектив офисных сотрудников) и при этом по каким-либо причинам не иметь доступа к сервису «Google Docs» (в том числе, при нахождении на тех территориях, где доступ к продуктам «Google»

заблокирован, например Китай). В данном случае, существование локальной сети между пользователями не является достаточным условием организации совместного редактирования текста. Такая ситуация может возникнуть в случае, если участники редактирования документа находятся в дороге (в таком случае трудно обеспечить постоянный доступ во Всемирную сеть), или же когда качество предоставления доступа в Интернет является невысоким, в связи с чем связь осуществляется нестабильно.

Описанная выше проблема ставит перед нами необходимость создания такого программного обеспечения, которое бы позволило осуществлять совместное редактирование текстов, но было бы лишено недостатков существующих ныне аналогов.

Следовательно, целью данной работы является реализация системы совместного редактирования текстов в режиме реального времени, работающая в локальной сети. Задачи, которые позволят достигнуть указанной цели, следующие:

1. Изучить существующие программные средства и подходы к организации совместной работы;
2. Спроектировать программное обеспечение, направленное на совместного редактирования текстов;
3. Реализовать программный продукт;
4. Описать созданное ПО.

Объектом данной работы является программное обеспечение, созданное с целью поддержки взаимодействия между людьми, совместно работающими над решением общих задач (программное обеспечение совместной работы). **Предметом** дипломного проекта является программное обеспечение совместной работы для редактирования текста.

Техническое задание на программный продукт включает в себя следующие требования:

1. Реализация программного обеспечения для синхронизации текста в автоматическом режиме без ограничений на стандартные возможности редактора текста;
2. Поддержка модульности исходного кода с целью его максимальной портируемости между различными редакторами;
3. Возможность поиска компьютеров в локальной сети, ожидающих входящие подключения.

В качестве языка программирования выбран язык программирования Python, поскольку он является допустимым языком написания дополнительных плагинов к целому семейству различных популярных бесплатных текстовых редакторов: Vim, Emacs, Sublime Text, gedit, Spyder IDE, LibreOffice Writer. В качестве текстового редактора выбран Sublime Text в связи с предпочтением автора.

Данный дипломный проект направлен на то, чтобы усовершенствовать существующую технологию редактирования документов, создав возможность совместной работы над одним текстом в рамках одной сети, что позволит повысить качество взаимодействия между людьми.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Техническое задание и его анализ

1.1.1 Введение

1.1.1.1 Наименование продукта

В данном документе создаваемая информационная система имеет полное наименование — «Collaboration» (в дальнейшем именуемая просто «программа»). Информационную систему «Collaboration» допускается именовать Система совместного редактирования текстов.

1.1.1.2 Область применения

Предполагается использовать программу в ходе работы над текстовыми документами коллективом офисных сотрудников (например, программистов, либо писателей). При координации мозгового штурма нескольких удаленных пользователей или других организационных мероприятий. А так же при удаленном обучении (консультировании), когда необходим постоянный обмен редактируемого текста между учителем и слушателем с целью повышения интерактивности.

1.1.2 Назначение разработки

Программный проект создается с целью предоставления пользователю возможности кооперативной работы над текстовыми документами.

Создание информационной системы «Collaboration» должно обеспечить повышение эффективности работы офисных сотрудников за счет улучшения качества процесса взаимодействия между ними.

1.1.3 Требования к программе

1.1.3.1 Специальные требования

Программа должна быть реализована в качестве расширения к текстовому редактору Sublime Text. Обладать относительно легкой переносимостью, поскольку в дальнейшем планируется увеличение количества поддерживаемых редакторов.

1.1.3.2 Требования к функциональным характеристикам

Функциональные требования состоят из следующих пунктов:

- Автоматическая или полу-автоматическая синхронизация: пользователь должен как можно меньше времени тратить на синхронизацию с другими пользователями, при этом синхронизация не должна влиять и/или зависеть от стандартного функционала редактора.
- Синхронизация текста должна происходить непрерывно, вне зависимости от действий пользователя для того, чтобы постоянно поддерживать консистентное состояние системы.
- Наличие функциональной возможности поиска ожидающего подключения компьютера в локальной сети, т. е. должна быть возможность подключения двух компьютеров внутри локальной сети без использования пользователями IP адресов в явном виде.

Входные данные

Входными данными при работе через Всемирную сеть является строка подключения (состоящая из IP адреса компьютера и порта), с помощью которой пользователь может инициировать подключение к другому ожидающему компьютеру.

1.1.3.3 Временные характеристики

Работа программы не должна заметно задерживать работу текстового редактора на компьютере с характеристиками: Intel® Core™ 2 Duo 3ГГц, 4 ГБ ОЗУ или на компьютере с аналогичной конфигурацией под управлением ОС Ubuntu 14.04 и Windows 7.

1.1.3.4 Прикладной программный интерфейс

Возможность синхронизации текста должна быть максимально инкапсулирована от API текстового редактора с целью дальнейшего портирования на Emacs, Vim, и др.

1.1.3.5 Пользовательский интерфейс

Возможность инициировать подключение должна предоставляться с помощью стандартных элементов управления текстового редактора. Пользователю должны быть доступны функции подключения, функции поиска ожидающих подключения компьютеров с помощью горячих клавиш.

Также элементы управления должны быть доступны через «палитру команд» (Command Palette) текстового редактора Sublime Text посредством нажатия сочетания «Ctrl + Shift + P».

1.1.3.6 Требования к надежности

Общие требования к надежности подразделяются на два пункта:

1. Устойчивость к проблемам в сети. Если между пользователями передача данных происходит по медленному каналу связи, это не должно стать причиной, по которой синхронизация будет тормозить пользователя, внося какие-либо блокировки в естественный процесс использования редактора текста.

2. Надежность от программных ошибок. В случае возникновения фатальных ошибок всегда, когда это возможно, должен использоваться механизм восстановления.

1.1.3.7 Условия эксплуатации

Условия эксплуатации совпадают с аналогичными условиями эксплуатации ЭВМ, на которой будет запускаться программа. Никаких специальных требований к пользователю не предъявляется кроме знания IP адресов, которые используются в строке подключения.

1.1.3.8 Требования к составу и параметрам технических средств

Минимальный размер оперативной памяти для комфортной работы — 512 МБ. Минимальная мощность процессора эквивалентна мощности процессора Intel Atom N330.

1.1.4 Требования к информационной и программной совместимости

Программа должна корректно поддерживаться системой расширений редактора Sublime Text и работать на всех платформах, на которых работает сам текстовый редактор.

Программа должна быть реализована на языке программирования Python. Разрешается использовать только кроссплатформенные фреймворки и сторонние библиотеки, которые должны поставляться вместе с программой или быть встроенными.

1.1.5 Требования к программной документации

Программная документация требуется в минимальном объеме в виде справочной информации, доступной через стандартное диалоговое окно «Помощь» текстового редактора.

1.1.6 Технико-экономические показатели

Программа должна относится к категории свободного программного обеспечения и не накладывать никаких ограничений на использование, распространение и модификацию.

1.1.7 Стадии и этапы разработки

Нет требований к процессу разработки.

1.1.8 Порядок контроля и приемки

Прием работы предваряет испытание программы на практике.

Испытание состоит из последовательности операций модификации текста одновременно двумя пользователями при нахождении обоих курсоров на одной позиции: одновременная двойная запись, одновременная запись и удаление текста.

1.2 Анализ технического задания

Основным требованием при выборе способа реализации является требования к автоматической или полуавтоматической синхронизации текста, минимизирующего требуемые вмешательства пользователя во время работы программы. Это означает, что требуется найти или разработать соответствующий алгоритм.

Другим важным требованием является надежность ИС:

- Устойчивость к проблемам в сети;
- Надежность от программных ошибок.

Первый пункт будет учтен при проектировании системы. Основные проблемы сети это нестабильное подключение и большая латентность. Во время разработки алгоритма следует принимать во внимание данные факторы и

предоставить возможное решение: процедура восстановления после обрыва соединения, процедура проверки непротиворечивости данных и др.

Основной причиной второго пункта требований является большая сложность ПО. Основные источники ошибок данного рода: это ошибки проектирования, ошибки алгоритмизации и ошибки программирования. Следовательно, чтобы обеспечить второй пункт требований, необходимо при проектировании и при разработке «бороться со сложностью ПО»[1].

1.3 Выбор инструментальных средств

В соответствии с ТЗ, использование Python является обязательным, поскольку является единственным поддерживаемым языком программирования для написания расширений к текстовому редактору Sublime Text. Поскольку ПО является сетевым, логично воспользоваться готовым фреймворком, позволяющий написание собственного протокола общения.

Среди всех фреймворков можно выделить несколько поддерживаемых и развивающихся, предоставляющий данные возможности [2]:

- «Twisted» — является самым старым из всех рассматриваемых фреймворков и является до сих пор активно разрабатываемым ПО. Имеет готовую реализацию вспомогательных протоколов, которые можно использовать в своей программе. Является кроссплатформенным.
- «PyEv» — относительно «молодой» фреймворк, использование которого пока не распространенно[2], что может вызвать большие трудности при попытке поиска решений проблем в сети.
- «Asyncore» — фреймворк низкого уровня, предназначенный для разработки высокопроизводительных программ. Сравнительно плохо подходит для прототипирования и в реализации требует большое количество времени и опыта.

---

[1] «Борьба со сложностью» или управление сложностью — это главный технический императив разработки ПО, согласно С. Макконнеллу [11]

[2] Согласно поиску по открытым проектам на сайте github.com и stackoverflow.com

– «Tornado» — веб-фреймворк, являющийся основным конкурентом «Twisted», главной целью которого является хорошее масштабирование на десятки тысяч одновременных подключений. К сожалению, не работает под операционной системой Windows.

Исходя из представленного краткого обзора, можно сделать вывод, что «Twisted» удовлетворяет требованиям ТЗ и является подходящей технологией для реализации системы синхронизации текста в виду его кроссплатформенности, развитости и наличия готовых заготовок, облегчающих разработку нового протокола.

Также, «Twisted» — это событийно-ориентированный фреймворк, который позволяет избавиться от блокировок во время исполнения программы благодаря своей асинхронной природе. Асинхронные программы обладают бóльшей производительностью по сравнению с синхронными[3] в виду того, что во время исполнения блокировки потока выполнения не происходит.

Python обладает большúм количеством библиотек, находящихся в открытом доступе. Функциональная возможность поиска компьютера в локальной сети является задачей, которая уже имеет решение. Поэтому в данном случае целесообразно воспользоваться готовой библиотекой. Согласно поиску по сайту https://pypi.python.org/ — официальному репозиторию Python, имеется библиотека под названием netbeacon, реализующая данный функционал. Netbeacon также удовлетворяет требованиям ТЗ.

1.4 Обзор литературы и существующих подходов к решению

1.4.1 Обзор литературы

Согласно Э.С. Таненбауму, распределенная система — это набор независимых компьютеров, представляющийся их пользователям единой

[3] При сравнении синхронной и асинхронной версий программы с одинаковым количеством потоков исполнения [12].

объединенной системой [3]. Очевидно, система совместного редактирования текстов является распределенной.

В данной книге автор детально описал принципы, концепции и технологии данных систем: связь, процессы, синхронизацию, целостность и репликацию, защиту от сбоев и безопасность, а также общую классификацию распределенных систем. В данном случае систему совместного редактирования текстов можно отнести к классу groupware (в переводе «системы групповой работы») — программ для совместного редактирования документов, проведения телеконференций.

В частности, были рассмотрены следующие информационные системы принадлежащие к данному классу.

Всемирная паутина. Всемирная паутина (англ. World Wide Web) в настоящее время является наиболее важной распределенной системой документов. Её стремительное развитие послужило толчком к резкому повышению интереса к распределенным системам.

Согласно парадигме Всемирной сети, всё является документом и передача и общение между компьютерами происходит с помощью трансляции документов. Все документы хранятся на специальных компьютерах, называемые серверами. Доступ к серверам можно получить с помощью специального ПО — браузера. Браузер выполняет функцию посредника и запрашивает необходимые документы (картинки и вспомогательные сценарии), а так же отвечает за отображение документа клиенту.

Первоочередной задачей в системе совместного редактирования текстов является синхронизация участников редактирования. Данный вопрос может быть решен по-разному.

Долгое время синхронизации данных во всемирной паутине не уделялось большого внимания в основном по двум причинам. Во-первых, жесткая организация паутины в которой серверы не обменивались информацией между собой означает, что синхронизировать практически нечего. Во-вторых, всемирную паутину можно считать системой, предназначенной главным

образом для чтения. Изменения обычно вносятся одним человеком, а в таким условиях трудно ожидать конфликта двойной записи.

Однако всё меняется и постепенно поддержка совместной работы над web-документами стала актуальной. Другими словами, теперь и в Web стали существовать механизмы одновременного обновления документов группой совместно работающих пользователей.

Для решения данной задачи был создан протокол WebDAV (Web Distributed Authoring and Versioning — распределенная подготовка документов в Web с контролем версий), который поддерживается простые средства блокировки разделяемых файлов, а так же создания, удаления, копирования и перемещения документов, находящихся на удаленных web-серверах.

В WebDAV имеются два типа блокировок записи. Блокировка исключительной записи может быть предоставлена клиенту и в течение срока действия запрещает всем остальным клиентам изменять документ. Существует так же и блокировка совместной записи, которая позволяет нескольким клиентам изменять документ одновременно. Поскольку такая блокировка сопровождается дроблением документа, блокировка совместной записи удобна, когда клиенты изменяют разные части одного документа. Однако клиенты должны отслеживать возможные конфликты двойной записи самостоятельно.

Протокол WebDAV может решить задачу совместного редактирования текста группой пользователей, но к сожалению, не в режиме реального времени. К тому же WebDAV для синхронизации параллельного доступа используется примитивный механизм блокировок, который лишает удобства работу пользователей. В случае конфликта двойной записи пользователям придется тратить время на мануальную синхронизацию, выявлять конфликтные строчки и разрешать коллизии.

Существует альтернатива всемирной паутине — IBM Notes. IBM Notes (ранее Lotus Notes) — это распределенная система, ориентированная на базы данных, также относящаяся к классу groupware. Разработана корпорацией Lotus Development. Модель IBM Notes значительно отличается от модели, принятой в

Web. Основным элементом данных является список элементов (заметки), который является структурой данных, привязанной к БД. Изменение и отображение заметок в IBM Notes осуществляется исключительно механизмами баз данных.

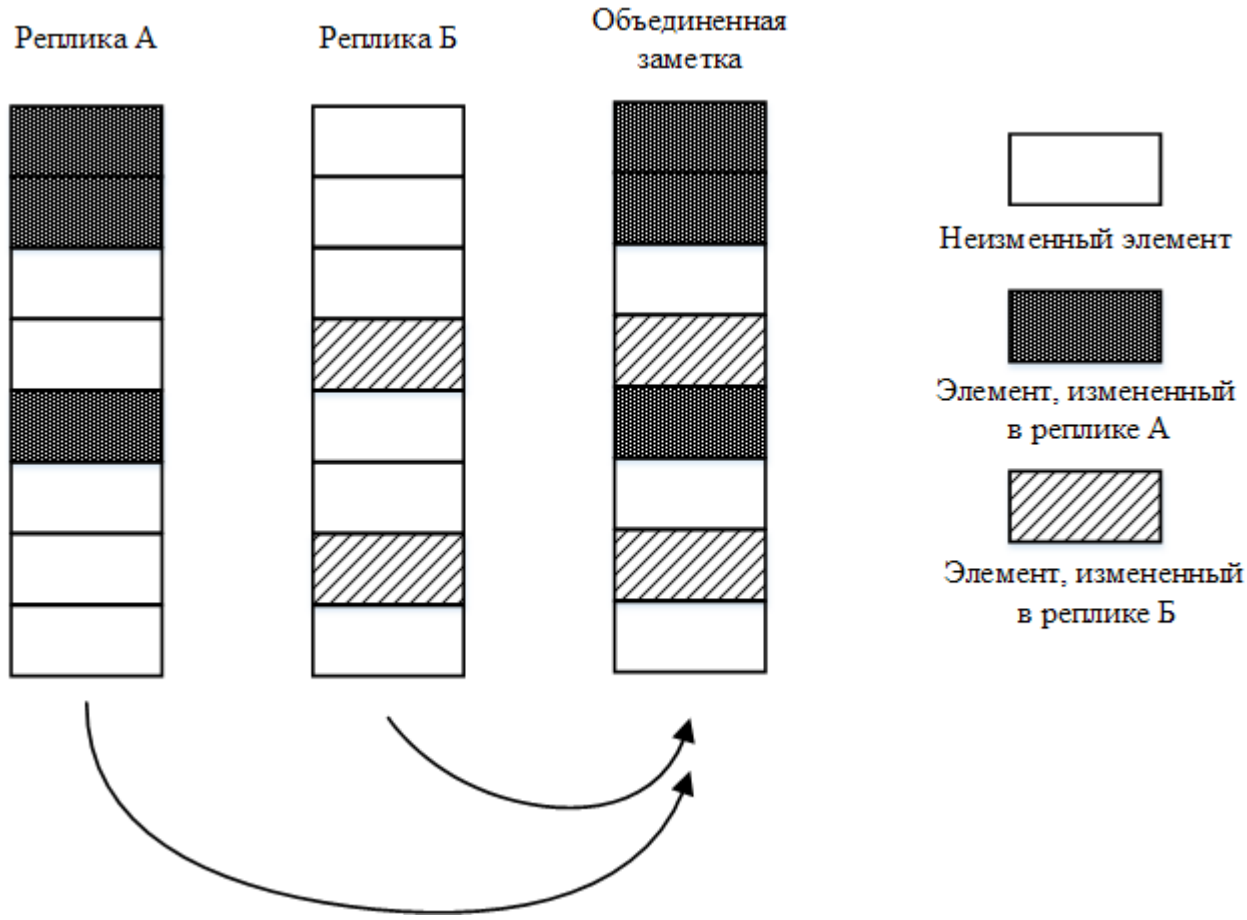

Рисунок 1 — Безопасное объединение двух документов А и Б с конфликтующими идентификаторами

Поддержка синхронизации в IBM Notes минимальна. В сущности, механизм синхронизации сводится к локальным блокировкам. Поддержка блокировки нескольких документов отсутствует.

В следствии одновременного редактирования одной и той же заметки двумя пользователями, возникают две различные копии. Если копии изменяются независимо друг от друга, то будут возникать конфликты двойной записи. В системе IBM Notes данные конфликты обнаруживаются и разрешаются следующим образом: в случае, когда это возможно, Notes объединяет копии с отслеживанием составляющих заметку элементов (пример на рисунке 1). В случаях когда автоматическое объединение невозможно, Notes

отмечает неразрешимый конфликт и в этом случае выбирает лишь последнюю измененную заметку.

Таким образом, ни IBM Notes, ни протокол WebDAV не может являться решением задачи синхронизации участников редактирования в режиме реального времени.

Интернет-сервисы. На текущий момент всемирная паутина отождествляется с Интернетом всецело благодаря своей популярности. Во всемирной паутине, в настоящее время, стали популярны интернет-сервисы, предоставляющие возможность совместного редактирования текста. Об этом свидетельствуют такие продукты от известных компаний как «Google Документы», «Zoho Docs» и другие. Данные сервисы предоставляют полноценный текстовый процессор прямо в браузере, возможность общения через чат и одновременное редактирование текста.

На сегодняшний момент, «Google Документы», «Zoho Docs» являются закрытыми проектами, и документация о внутреннем функционировании не находится в публичном доступе.

Однако существует преемник «Apache Wave» ныне закрытого проекта «Google Wave», чья документация выложена в публичный доступ. «Apache Wave» нацелен на совместную работу над документами, позволяя пользователям общаться и редактировать их в режиме реального времени.

Технологическим ядром проекта Wave стала концепция операционального преобразования (англ. «operational transformation») для поддержки целого ряда функциональных возможностей сотрудничества. Впервые концепцию операционального преобразования разработали С. Эллис, С. Гиббс в системе GROVE в 1989 году [4].

Основная идея может быть проиллюстрирована на примере сценария редактирования простого текста следующим образом: дан текстовый документ со строкой «abc», реплицированный на двух компьютерах двух пользователей; а также одновременно произведены две операции над этим текстом, двумя пользователями соответственно:

– вставить символ «x» в позицию 0 (обозначим операцию через O_1);
– удалить символ «c» в позиции 2 (обозначим операцию через O_2).

Предположим, что две операции выполняются в следующем порядке: сначала O_1, и затем O_2 (см. рисунок 2). После выполнения O_1, текст в документе становится «xabc». Для выполнения O_2 после O_1, O_2 должна быть уже преобразована относительно O_1 и предстать в следующем виде: удалить символ «c» в позиции «3» (обозначим операцию через O_2'), где параметр позиции увеличен на единицу, в связи со вставкой символа «x» операцией O_1. Выполнение O_2' на «xabc» должно удалить правильный символ «c», и текст этого документа становится «xab». Если же выполнять операцию O_2 без преобразования, тогда будет неправильно удален символ «b» вместо «c».

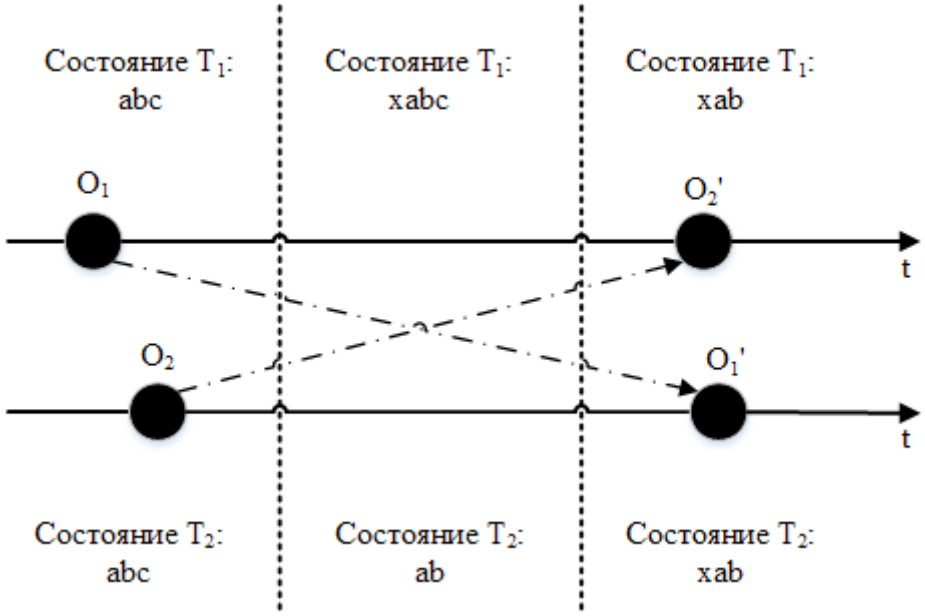

Рисунок 2 — Пример операционального преобразования. На рисунке представлены две временные линии двух пользователей с отмеченными на них операциями O_1, O_2, O_1', O_2'. Текст первого пользователя отмечен как T_1, второго соответственно T_2

Таким образом, основной идеей операционального преобразования является изменение параметров операций редактирования в согласии с эффектами выполнения в предыдущих одновременных операциях, так чтобы преобразованная операция могла сработать корректно и не нарушать согласованность в документе.

Архитектура проекта Wave клиент-серверная. Чтобы обеспечить достаточное время отклика в средах с высокой латентностью, таких как Интернет, синхронизируемые документы копируются в локальное хранилище каждого пользователя, позволяя тем самым проводить операции редактирования локально. При редактировании локальной копии, операции как команды передаются удаленному серверу. Операциональное преобразование гарантирует согласованность данных на всех компьютерах участников группового редактирования. Основное свойство, которое позволяет реализовать удобное использование ПО данного вида — это свойство неблокируемости. Групповое, и одиночное редактирование текста происходит с одним и тем же локальным временем отклика.

К сожалению, операциональное преобразование — это чересчур трудная в реализации концепция. Joseph Gentle, сотрудник Google, высказал свое мнение, что реализация проекта Wave заняла 2 года, и если потребовалось бы переписать весь проект заново, то не смотря на пройденный путь, реализация бы заняла те же 2 года [5].

Таким образом интернет-сервисы предоставляют достаточный функционал, но при этом обладают рядом недостатков. Во-первых, использование интернет-сервисов требует *стабильное* подключение к глобальной сети. Это может стать проблемой, если выход в Интернет осуществляется, например, в общественном месте. Во-вторых, технически, совместная работа может вестись даже по локальной сети, без необходимости Интернета. Латентность в локальной сети очень часто бывает намного ниже, чем в глобальной сети, следовательно, работа через локальную сеть может быть комфортнее для пользователя. В-третьих, не во всех странах интернет-сервисы являются общедоступными (например, Google по политическим причинам заблокирован в Китае). В-четвертых, использование интернет-сервиса ради кооперативной работы автоматически принуждает к безальтернативному использованию встроенного редактора, эффективность работы за которым может слабо сравниться с эффективностью работы за любым современным

редактором аналогичным Sublime Text если речь идет не о простом тексте, а об исходном коде или документе с использованием языка разметки.

Альтернативные подходы к решению задачи.

Совместное редактирование текстов в режиме реального времени может быть реализовано по-разному. В зависимости от реализации, изначальную задачу можно свести к трем различным подзадачам:

- Позволить монопольное редактирование текста различными пользователями, внося различные блокировки на изменение текста. Данный подход представлен технологией WebDAV;
- Предоставить каждому пользователю редактировать свою копию текста. Чтобы совместить правки каждого пользователя, необходимо транслировать все команды-изменения над текстом на сервер-координатор, который и будет хранить результирующую копию. Этот подход представлен в Apache Wave.
- Позволить каждому пользователю редактировать свою копию текста, но результирующую копию получать посредством слияния[4] нескольких версий документа на сервере-координаторе. Этот подход используется во всех современных системах контроля версий таких как Git, Subversion, Darcs и др.

Реализация третьего подхода предполагает наличие возможности слияния двух версий документа.

Данная задача в СКВ Git и др. аналогах решается при помощи специальных утилит diff и patch. Утилита diff создает файл, хранящий построчную разницу между двумя версиями текста (например, между версиями A_1 и A_2), а утилита patch позволяет применить получившиеся изменения к другой версии файла (A_3), тем самым получив результирующую копию [6,7]. Пример использования утилит diff и patch при помощи графической оболочки meld представлен на рисунке 3 ниже.

[4] Слияние — это процесс объединения изменений, внесенных в две копии файла, путем создания нового файла [7]

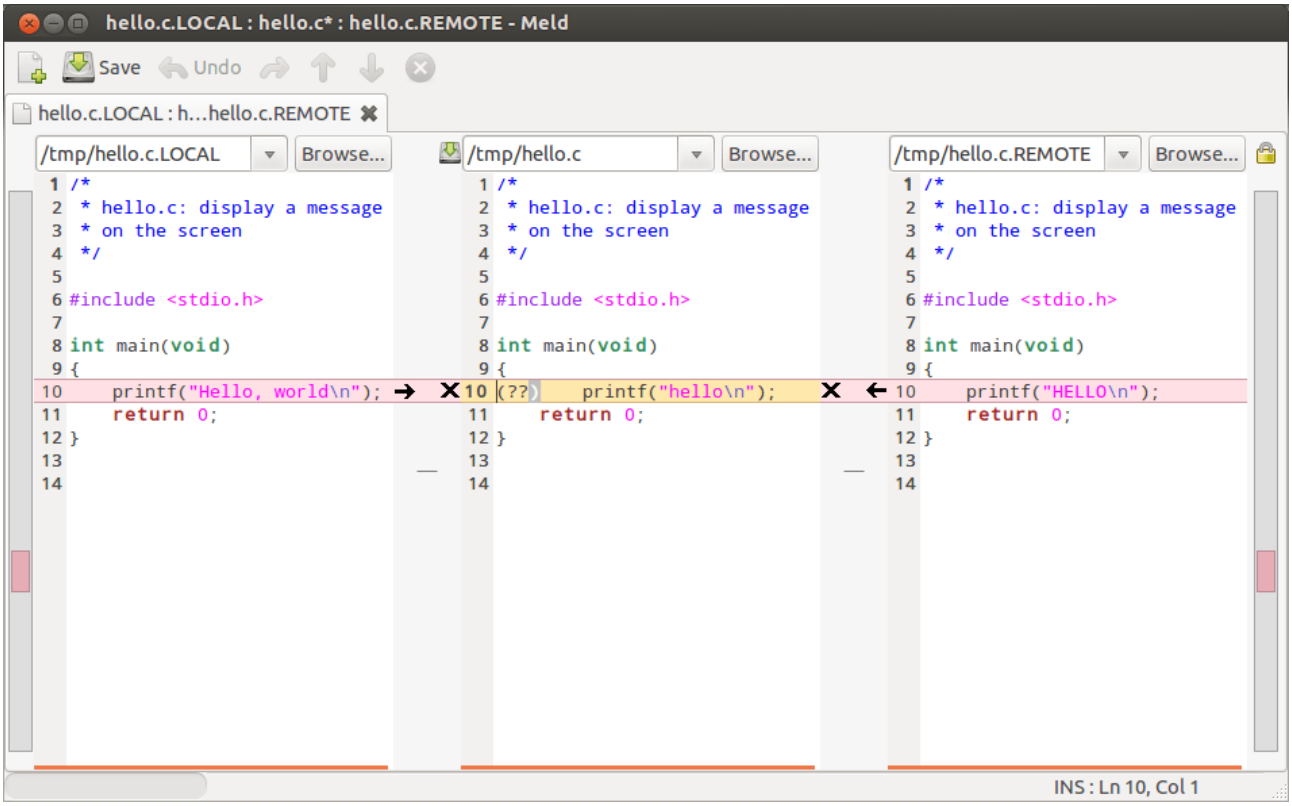

Рисунок 3 — Пример разрешения конфликта двойной записи при помощи программы meld. Две версии одного файла (слева и справа) конфликтуют в 10 строчке

Последний подход используется во всех основных СКВ. СКВ как раз предназначены для координации множества участников редактирования, например при разработке программы редактируются одни и те же файлы разными программистами. Данный подход отличается от предыдущих в следующем:

– Операция слияния производится непосредственно над текстом, а значит обладает независимостью от возможностей текстового редактора. Тем самым позволяя пользоваться различными редакторами.
– Синхронизация текста может производится автоматически за исключением лишь случаев конфликта двойной записи.
– Синхронизация может быть инициирована в любое время, удобное для пользователя.

Перечисленные отличия дают то преимущество, которое необходимо для системы совместного редактирования текстов в режиме реального времени согласно ТЗ. Проблема последнего подхода в том, что во время работы не

учитывается структура файла, которая может быть нарушена в некоторых случаях.

XML является языком разметки. XML документ, в отличии от простого текста обладает определенной структурой (может быть представлен в виде иерархического дерева). Это означает, что применение утилит diff и patch может «сломать» иерархическое дерево, поскольку они работают построчно.

На сегодняшний день, XML является довольно популярным форматом сериализации данных. Поскольку редактируемый документ может быть представлен в формате XML, например формат файла fodt – это дает возможность хранить помимо текстовых данных, данные о форматировании, цвете, врезках и др. Таким образом, задача слияния XML документов является актуальной задачей в контексте синхронизации текста между несколькими пользователями.

Данная проблема синхронизации XML документов решена Т. Линдхолмом в своей работе [8]. В ней представлена технология трехходового слияния XML документов (англ. three-way merge) с вычислительной сложностью $O(n \log n)$. Трехходовое слияние предполагает наличие трех версий документа: общего потомка и двух версий файлов, которые собираются слить в итоговую копию. Общий потомок — это документ, который является общим прообразом для двух текущих версий файлов.

1.4.2 Детальный обзор существующих подходов к синхронизации

1.4.2.1 «Блокирование — изменение — разблокирование»

Стратегия с использованием блокировок текста является самой простой в реализации. В самом простом варианте можно представить как редактирование docx файла, открытого для остальных участников локальной сети. Пока происходит редактирование только одним человеком в каждый момент времени.

«Блокирование — изменение — разблокирование» обладает рядом недостатков и этот подход часто неудобен:

- Блокирование может вызвать проблемы одновременного доступа. Иногда время блокировки могут продолжаться сравнительно долго (либо вовсе всегда, в случае сбоя). Это нарушает доступность файла, которую придется обеспечивать различными техническими ухищрениями такими как блокировка только на определенное количество времени и пр. Это в конечном счете приводит к задержкам и потерянному времени.
- Блокирование может вызвать «пошаговость». Вполне вероятна ситуация, когда в какой-то момент времени есть сразу два писателя, которые редактируют начало разные области текстового файла. И эти изменения несовместны. Очевидно, они могли бы редактировать файл одновременно, но данный подход этого не предусматривает.
- Блокирование не обеспечивает защиту консистентности информации. Блокировки, несомненно являются дорогой операцией для пользователя. Время ожидания отклика обратно пропорционально параметру доступности системы, а значит, чем больше необходимо блокировок – тем хуже для пользователя.

Пускай, есть два файла, в которых есть перекрестные ссылки друг на друга (они зависят друг от друга). Если первый писатель заблокирует первый файл, то это не приведет к блокировке второго файла, который логически зависит от первого. А значит, не смотря на монопольный доступ до ресурса, может случится ситуация, когда сделанные изменения буду несовместимы. Данный подход не обеспечивает защиту от подобных ситуаций, но вместо этого она формирует ложное чувство безопасности у пользователей.

Очевидно, данный подход не может предоставить оптимального решения задачи редактирования текста в режиме реального времени согласно ТЗ.

1.4.2.2 Зеркалирование команд редактирования

Данный подход основан на пересылке всех действий писателя и воспроизведении их на остальных компьютерах. На данный момент, самый популярные алгоритмы данного подхода основаны на концепции операционального преобразования (см. страницу 19).

Для данного подхода критично, в какой последовательности будут производиться вставки и удаления. Если в случае двух человек, это не является проблемой, то во время редактирования втроем, уже необходимо вводить дополнительные блокировки, чтобы обеспечить верную последовательность вставок-удалений, что приводит к задержкам.

Для данного подхода критично, в какой последовательности будут производиться вставки и удаления. Если в случае двух человек, это не является проблемой, то во время редактирования втроем, уже необходимо вводить дополнительные блокировки, чтобы обеспечить верную последовательность вставок-удалений, что приводит к задержкам.

Сохранение истории действий писателя влечет за собой зависимость системы синхронизации от самих действий писателя. В настоящий момент, редакторы обладают богатыми возможностями по модификации текста: поиск/замена, вставка, поддержка функции множественных курсоров, автоформатирование и др. При данном подходе весь данный функционал должен быть поддерживаемым со стороны системы синхронизации текстов, что влечет за собой дополнительную сложность.

Также, в общем случае, модификация текста может быть выполнена не только человеком, но и скриптом или системой контроля версий. Написать реализацию под каждый вид редактирования текста представляется крайне сложной задачей. Поэтому используя данный подход, реализация синхронизации будет сильно зависеть от возможностей самого редактора, что является ограничивающим фактором для портирования приложения.

Таким образом можно сделать вывод, что передача команд не является оптимальным способом решения задачи синхронизации текста в реальном времени.

1.4.2.3 «Копирование — изменение — слияние»

В этой модели каждый участник редактирования обращается к хранилищу и копирует текущую копию файла. После этого писатели редактируют лишь свои копии, независимо от других. В конце концов, личные копии отправляются обратно, изменения объединяются, сливаясь в итоговую версию файла. Можно обозначить ситуации, когда слияние можно провести автоматически, но в ситуациях когда возникают конфликты, может разобраться только лишь человек (если отредактирована одна и та же строка разными писателями по-разному, то автоматически слить изменения разных писателей в одну итоговую копию невозможно).

Судя из названия, данный подход состоит из трех основных шагов:

1. Каждый писатель копирует текущую версию файла себе;
2. Писатели редактируют какие-то фрагменты текста и отправляют свою версию файла в хранилище;
3. Хранилище производит слияние всех текстов в итоговую копию и отправляет всем писателям.

Основной плюс данного подхода заключаются в отсутствии блокировок на редактирование. Каждый писатель независимо ото всех редактирует свою личную копию файла, а поэтому нет необходимости вносить примитивы синхронизации на стадии редактирования текста.

Таким образом, редактирование файла одновременно двумя и более писателями возможно, а синхронизация происходит уже после внесения множества правок. В случае конфликта, система лишь детектирует противоречие. Таким образом каждый писатель подвержен риску конфликта с

другими участниками, а значит время от времени пользователям придется тратить время на разрешение конфликтов.

Легко понять, что чем чаще будут производиться слияния, тем меньше потенциальных конфликтов может произойти, а значит тем меньше времени пользователя будет потрачено на синхронизацию с другими.

На текущий момент, одной из популярных СКВ, использующей данный подход является Subversion. Subversion позволяет пользователям работать параллельно, не тратя время на ожидание друг друга, чем и завоевал свою популярность. Subversion является централизованной системой (синхронизация производится централизованно одним компьютером).

Очевидно, что основной недостаток данного подхода заключается в том, что результирующая система будет использовать полудуплексный способ связи[5]. Каждый пользователь может привносить изменения в свою локальную копию, при этом, ранее второго пункта алгоритма, пользователи не знают, конфликтуют ли текущие изменения с централизованной копией. Это заставляет участников время от времени разрешать конфликты, тратя свое время на синхронизацию.

Второй основной недостаток, это жесткие требования к стабильности сети: для редактирования в режиме реального времени данный подход «не прощает» задержки соединения.

Следовательно, данная стратегия так же не обеспечивает все требования ТЗ.

[5] Полудуплексный способ связи предполагает, что устройство в один момент времени может либо передавать, либо принимать информацию.

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Используемые алгоритмы и их анализ

2.1.1 Модифицированная стратегия «копирование — изменение — слияние»

Очевидно, стратегия «копирование — изменение — слияние» не полностью подходит для системы совместного редактирования в режиме реального времени и требуется доработка. Но прежде чем начать описывать содержательную часть, необходимо ввести три определения.

Определения.

Текст — это последовательность букв в котором все переносы кодируются специальным (составным) символом «\n». Текст можно вычитать друг и друга, получая при этом некоторый объект-дельту.

Дельта между текстами T_1 и T_2 — это совокупность удалений, вставок и замен, которые необходимо воспроизвести, чтобы текст T_1 превратился в текст T_2. Например: пусть текст T_1 это «Жираф», а T_2 это «жираф». Тогда разница между ними будет: удалить первый символ, вставить букву «ж» перед «и». Дельты можно применять к тексту, воспроизводя совокупность удалений и вставок.

Очевидно, что разница между двумя текстами не единственна.

Дельта может быть представлена в формате UniDiff [6] (см. таблицу 1, стр.33). В первой строке между символами «@@» указывается предполагаемые номера строк нахождения. Если перед строкой стоит знак минус, это значит что данную строку необходимо удалить, если стоит знак плюс, то вставить. Если не стоит никакого знака, то данная строчка должна найтись в файле (т. е. она выполняет функцию контекста).

Каждая дельта состоит из фрагментов. Фрагмент представляет одну операцию вставки, удаления. Каждый фрагмент обладает контекстом — строка

текста перед или после, которая должна совпасть с исходным текстом, к которому применяется дельта. Например, в таблице два фрагмента «-и сосала сушку», «+и сосал сушку» предваряются строкой контекста «по шоссе».

Конфликт — это особая ситуация, которая возникает при проведении операции слияния двух версий файла при которой, существует противоречие между сделанными правками. Конфликт возникает, когда в одной версии файла строка K_1 заменяется на K_2. А в другой версии файла эта же строка заменяется на K_3. В результате возникает неразрешимое противоречие при попытке .

Анализ. Если рассмотреть «копирование — изменение — слияние», можно выделить основные моменты:

- Редактируется локальная копия. Это позволяет избавиться от ожидания других писателей.
- Централизованный сервер хранит версию файла, которая считается «правильной» в том смысле, что все правки, сделанные над «правильной» версией файла являются окончательными и не могут быть отменены. Это позволяет каждому участнику подключиться к редактированию в любой момент времени.
- Слияние позволяет собрать воедино все изменения, произведенные над файлом (без необходимости записи всей истории операций редактирования писателя).

Для того, чтобы обеспечить требования задачи, необходимо задаться целью держать локальную копию файла и централизованную как можно «ближе». В данном случае «ближе» понимается в смысле расстояния Левенштейна. Для этого, вне зависимости от действий писателя, будет производится синхронизация текущего состояния текста с остальными писателями (либо центральным координатором) с каким-то периодом[6] T.

Поскольку отправлять весь файл в хранилище через период T является дорогостоящей операцией, потребуется, расчет дельты Δ уже на компьютерах

[6] Этого можно добиться, запустив отдельный поток, отвечающий всецело за синхронизацию.

писателей. Для этого необходимо хранить последнюю версию текста S, считающейся «верной» (в Subversion правильная версия — централизованная итоговая копия, которая получается сливанием). Например, если редактирование текста только началось и файл пуст, то «правильная» копия текста у всех писателей это пустая строка.

В каждый момент времени $t = Tn$ при $n \in \mathbb{N}$, каждый писатель имеет текст S_{t-T} который является одинаковым и по смыслу является слитой версией на момент времени $t - T$.

Не обращая пока внимания на обработку ошибок, которые могут возникнуть во время работы, опишем основные шаги алгоритма. Для каждого писателя:

1. В момент времени t запускается процедура синхронизации;
2. Считается дельта $\Delta_{t,t-T}$ между текущим текстом C_t и S_{t-T};
3. Полученный результат сериализуется и отправляется в хранилище;
4. Присваивается $S_t = C_t$.

Таким образом, обозначена процедура, которая гарантирует, что при редактировании текста, об этом будут уведомлено хранилище, при этом хранилище обязано воссоздать текст C_t применяя дельту к исходному тексту S_{t-T}. Остается обозначить, каким образом принимать дельты, чтобы не нарушить консистентность текста S во всей системе:

1. Хранилище принимает дельту $\Delta_{t,t-T}$;
2. Применяет дельту к S_{t-T}, получая $C_t = S_t = S_{t-T} + \Delta_{t,t-T}$;
3. Если предыдущие шаги выполнены без конфликтов двойной записи, отправляет всем остальным участникам $\Delta_{t,t-T}$.

Необходимо обязать каждого писателя принять дельту $\Delta_{t,t-T}$ от хранилища, поскольку все изменения в хранилище являются итоговыми и не могут быть отменены. Тогда для писателя процедура применения дельты от хранилища:

1. Попытаться применить дельту к текущему тексту $C_{t+\delta}$ (δ отвечает за время задержки между текущим временем и временем t);
2. Если возникает конфликт, необходимо вернуться (в истории изменений текста) на первое состояние, при котором $\Delta_{t,t-T}$ может примениться и воссоздать состояние $S_t = S_{t-T} + \Delta_{t,t-T}$.
    a. Сохраним текущее состояние текста как $C_{t+\delta}$ для того, чтобы не потерять уже сделанные изменения, которые произошли за время δ.
    b. Будем отменять («откатывать») изменения начиная с последнего, двигаясь обратно во времени, пока не сможет примениться $\Delta_{t,t-T}$ без конфликтов. Обозначим найденное состояние как Z.
    c. Присвоим $S_t = Z + \Delta_{t,t-T}$, получив тем самым то состояние, которое было в хранилище на момент времени t.
    d. Чтобы не потерять сделанные изменения за время δ, посчитаем дельту между Z и $C_{t+\delta}$. Обозначим эту дельту как ζ.

Очевидно, что дельта ζ не сможет примениться к состоянию Z без конфликтов, потому что контекст, необходимый для вставок и удалений не будет совпадать, из-за примененной $\Delta_{t,t-T}$.

Таким образом, чтобы правильно обрабатывать ситуации конфликта, необходим алгоритм, который бы мог применить дельту с нарушением контекста. К счастью, такой алгоритм существует и называется Fuzzy Patch.

2.1.2 Алгоритм Fuzzy Patch

Описание Fuzzy Patch алгоритма лучше начать с примера ситуации, которая часто возникает у программистов при работе с СКВ.

Во время работы с системой контроля версий Git, типична ситуация, когда при слиянии двух веток[7], возникает конфликт, который может разрешить

[7] Ветка — это термин, использующийся в СКВ для обозначения текущей истории изменений, которые привели к тому состоянию файла, которое можно наблюдать. Здесь важен тот факт, что разные версии одного и того же файла имеют разные истории. Если есть две ветки изменений, то к этому можно отнестись как к двум

лишь человек. Такое получается, когда дельты начинают применять к версии текста, которая не является прямым прообразом данного текста. Приведем пример (см. таблицу 1).

Таблица 1 — Пример применения дельты в СКВ Git, когда нарушен контекст дельты

Дельта	Текст	Результат применения дельты к тексту

`@@ -1,3 +1,3 @@`
`-Шла Саша`
`+Шел Паша`
`по шоссе`
`-и сосала сушку`
`+и сосал сушку`
	`Шла Саша`
	`по шоссе`
	`и сосала сушку`
		`Шел Паша`
		`по шоссе`
		`и сосал сушку`

	`Шла Маша`
	`по шоссе`
	`и сосала сушку`
		`<<<<<<<`
		`Шла Маша`
		`=======`
		`Шел Паша`
		`>>>>>>>`
		`по шоссе`
		`и сосал сушку`

Как видно из примера, во втором случае дельта не может быть применена полностью, потому что нарушен контекст. Результат применения дельты ко второму тексту получился не таким, каким он предполагался и максимум, который смогла сделать СКВ — это указать возможное место строки «Шел Паша» — это первая строка (между строками с символами < и > указывается конфликт двойной записи где две конфликтующие строки отделены символами =).

Такое решение основывается на том, что последующий контекст совпал, но точное решение СКВ не может сделать и оставляет ситуацию как есть.

Среди известных программ, реализующих данное поведение — это GNU Patch. GNU Patch использует простой алгоритм нечеткого поиска. Согласно документации:

« ... программа может определить, если дельта не может быть полностью применена без ошибок и в таких случаях ищет корректное место для каждого фрагмента дельты в отдельности. Вначале фрагмент дельты ищется в

альтернативным реальностям в которых существуют две версии одного файла. Слияние веток – это объединение изменений в одну ветку. Без конфликтов сделать это не всегда удается.

предполагаемой окрестности. Если контекст не совпадает производится вторая попытка, сужая контекст таким образом, что отбрасываются одна строка контекста до и одна строка после фрагмента дельты. Если по прежнему не получается применить дельту в предполагаемой окрестности, то отбрасываются по две строки и производится вновь поиск» [9].

Как можно понять, такой подход является относительно ненадежным. Даже незначительные изменения между двумя версиями файла могут привести к тому, что дельта не сможет примениться.

В приведенном выше примере, если заменить слово «Саша» на «Паша», то это приведет к конфликту, несмотря на то, что контекст совпал (начало файла — пустая строка). Интуитивно понятно, что при редактировании текста в режиме реального времени такие ситуации не должны происходить в принципе.

К тому же, такое поведение может породить ошибки такого плана: если в файле находятся два одинаковых кусочка текста и при этом кусочек, к которому по идее должна примениться дельта, был удален. Программа GNU Patch посчитает неудаленный отрывок текста за совпавший контекст, даже если он находится в противоположном конце файла.

Fuzzy Patch алгоритм решает эту проблему: для этого хранятся фрагменты дельты не построчно, а посимвольно. Для каждого символа есть его контекст. Это позволит применять несколько фрагментов дельты на одной строке.

Данная ситуация может натолкнуть на мысль, что вместо процедуры отбрасывания контекстных строк лучше для каждого символа в дельте (обладающего контекстом) найти место в тексте для которого минимально расстояние Левенштейна: количество изменений которые надо допустить в тексте, чтобы он совпал с контекстом дельты. Назовем эти изменения ошибками.

Действительно, расстояние Левенштейна характеризует «похожесть» одной строки на другую. Интуитивно, если исходный текст к которому

применяется дельта будет похож на контекст дельты, то расстояние Левенштейна будет небольшим.

Таким образом можно предложить наивный алгоритм, который бы мог применить дельту с нарушением контекста: допустить факт того, что в контексте одна ошибка, пройти весь текст, попытавшись применить его с одной ошибкой, и, если не получилось, то допустить 2 ошибки и так далее.

2.1.2.1 Алгоритм нечеткого поиска фрагментов дельты в тексте

На сегодняшний момент алгоритмы быстрого нечеткого поиска в достаточной степени изучены. Один из таких алгоритмов — это двоичный алгоритм поиска подстроки, или что тоже самое Shift-or.

Данный алгоритм поиска подстроки, использует тот факт, что в современных компьютерах битовый сдвиг и побитовое ИЛИ являются атомарными операциями. Вычислительная сложность — $O(|n|\cdot|m|)$ где $|n|$ это длина искомой подстроки, $|m|$ — это длина строки, в которой ведется поиск. Полное описание алгоритма представлено в [10].

При нечетком поиске подстроки в строке существует два основных параметра, отвечающие за точность найденного фрагмента: близость к ожидаемой области и количество допустимых ошибок в подстроке.

В качестве метрики похожести для нечеткого поиска, используется следующее определение:

$$s = \left(\frac{e}{p}\right)\frac{1}{c} + \left(\frac{d}{t}\right)\frac{1}{1-c}, \quad (1)$$

где s — это минимизируемый параметр, интуитивно характеризующий степень похожести; e — количество ошибок (расстояние Левенштейна); p — длина подстроки; d — расстояние (в символах) найденной подстроки от предполагаемого места; t — длина всего текста; c — весовой параметр.

Данное определение интуитивно позволяет учитывать не только количество ошибок (в контексте), но и сдвиг (исчисляемый в символах) от предполагаемого идеального места нахождения искомого фрагмента дельты.

Константа c вводит соотношение между учитываемыми ошибками и сдвигом.

Еще одна переменная, которая может быть настроена это размер текста t. Минимум переменной t можно ограничить для небольших файлов (из-за того, что изменения в нескольких символах могут привести слагаемое $\frac{d}{t}$ к сравнительно большому числу) или, аналогично максимум для больших файлов.

При нечетком поиске всегда есть компромисс между местом нахождения подстроки и количеством допущенных ошибок. Опишем данный компромисс, рассмотрев как ошибки и сдвиг соотносятся при нечетком поиске.

Самый простой алгоритм поиска подстроки в строке будет следующий: сканировать весь текст в поисках точного вхождения. Если подстрока не найдена, то сканирование продолжиться с допущением одной ошибки. Если подстрока с одной ошибкой не найдена, то допустить две ошибки и т. д. пока число ошибок не будет равно на один меньше, чем длина искомой подстроки (потому что в таком случае она найдется в любом месте строки).

Данная стратегия поведения приводит к прямоугольной сетке, содержащей места возможных соответствий (см. таблицу 2).

Таблица 2 — Пример нечеткого поиска с допущением нескольких ошибок

Количество допущенных ошибок e	Строка поиска
	`'Twas brillig, and the slithy toves.Did gyre`
0	`1`
1	`111  1`
2	`11111  111  1111  111`

В приведенном выше примере в строке ищется образец «the». Единица обозначает начало найденного образца в строке. Есть одно точное совпадение

(первая строка содержит одну единицу), 4 неточных совпадений с одной ошибкой и 15 совпадений с двумя ошибками. При допущении количества ошибок, равного длине образца, подстрока заведомо найдется в любой строке, а потому данная ситуация лишена смысла.

Допустим, что перед нечетким поиском, примерное место нахождения образца в строке известно. Тогда с уменьшением допустимых ошибок, будет увеличиваться окрестность места предполагаемого нахождения подстроки в строке. Пример поиска подстроки «was» в окрестности 25-го символа (прочерком указываются символы, которые не участвовали в поиске, нулем указываются просмотренные символы):

Таблица 3 — Пример нечеткого поиска со знанием примерного местонахождения образца

Количество допущенных ошибок e	Строка поиска
	'Twas brillig, and the slithy toves.Did gyre
0	--1000000000000000000000000000000000000000
1	--------------000000000000000000000000----
2	------------------------0000--------------

В приведенном примере выше видно, что в каждой строке существуют границы слева и справа, дальше которых не имеет смысла искать, потому что все найденные вхождения вне этих границ будут заведомо с бóльшим параметромs. Эти границы можно вывести из формулы (1), выделив d при данных e и s:

$$d = t\left(s - \frac{e}{pc}\right)(1 - c)\,, \qquad (2)$$

Таким образом, для того, чтобы найти образец в строке с помощью нечеткого поиска, необходимо ограничить s каким-то числом s_f, рассчитать для каждого найденного образца в предполагаемой окрестности (т. е. последовательно, для каждого $e = 1,2,3,\ldots$, ограничить область поиска левой и правой границей) $s^{(i)}$ по формуле (1). Выбрать лучшее вхождение с минимальным $s^{(i)}$.

Поскольку априорной информации о том, каким числом s_f надо ограничить во время работы программы нет. То можно действовать следующим образом: найти первое вхождение подстроки, начав искать с начала строки. Для найденного первого образца посчитать s по формуле (1) и взять это значение как s_f.

2.1.2.2 Применение дельты с допущением ошибок

С помощью алгоритма поиска Shift-or находится необходимая подстрока в тексте. Далее, необходимо воспроизвести последовательность удалений и вставок таких, чтобы расстояние Левенштейна между исходным текстом и измененным было минимально.

Для этого приведем пример. Предположим, что есть два писателя и первый писатель модифицирует единственное слово в документе, а затем передает дельту второму, чтобы он применил её к своему тексту. Тогда может случится следующая ситуация. Пусть для первого писателя первые две версии текста будут V_1 и V_2 (представлены в таблице 4).

Таблица 4 — Различные версии текста для первого и второго писателей

Версия текста	Текст
V_1	`The computer's old processor chip.`
V_2	`The computer's new processor chip.`
V_1	`The server's odd coprocessor chip.`

Тогда дельта между V_1 и V_2 (начиная с 8 символа):

```
-puter's old process
+puter's new process
```

В этом случае, если второй писатель имеет изначально текст **V_1**, то искомая подстрока (в ходе применения данной дельты к тексту **V_1**) для первого фрагмента дельты будет: «`puter's old process`». Данная подстрока найдется в тексте начиная с восьмого символа с шестью допустимыми ошибками (см. таблицу 5): «`er's odd coprocesso`».

Таблица 5 — Результат нечеткого поиска для второго писателя

Искомая подстрока h для первого фрагмента	`puter's old process`
Текст $\mathbf{V_1}$ в котором производится поиск	`The server's odd coprocessor chip.`
Результат поиска ($e = 6$)	`--------100000000000000000000000`
Найденный фрагмент H в тексте $\mathbf{V_1}$	`er's odd coprocesso`

Как видно из примера, нельзя просто заменить три буквы начиная с 8 позиции на «`new`». Для того чтобы, провести замену, необходимо сопоставить индексы символов между заменяемой подстрокойи строкой в которой проводят замену (например, апостроф в h имеет индекс 6, а в H индекс 3). Часть букв остается на своих местах, а часть заменяется. Чтобы построить такое отображение, достаточно построить посимвольную разницу (дельту) между двумя строками H и h (зачеркиванием указаны символы, подлежащие удалению, а подчеркиванием указаны вставляемые символы, символы остающиеся без изменений никак не выделены), пример в таблице 6.

Таблица 6 — Посимвольная разница между h и H

Искомая подстрока h для первого фрагменты дельты Δ	`puter's old process`
Найденный фрагмент H в тексте $\mathbf{V_1}$	`er's odd coprocesso`
Посимвольная разница между h и H	~~put~~er's o~~l~~d co~~p~~rocesso

Из таблицы 6 видно, что часть текста остается неизменной и меняются лишь индексы. Следовательно, имея данное отображение можно утверждать: чтобы применить дельту Δ второму писателю, надо удалить «`odd`» и вставить на его место «`new`». $\mathbf{V_2}$ второго писателя после применения дельты будет: «`The server's new coprocessor chip`». Что и требовалось в итоге.

В конце важно обязательно сохранить разницу для дальнейших фрагментов дельты между предполагаемым местом нахождения и реально найденным местом (в данном случае 9), потому что следующая дельта будет иметь скорректированую предполагаемую позицию. В данном случае предполагаемое место нахождения равно 8, найденное место равно 9, следовательно следующий фрагмент необходимо искать со сдвигом равным единице.

2.1.3 Ограничения модифицированного алгоритма

В ходе дипломной работы была реализована модифицированная стратегия «копирование — изменение — слияние» и проведено несколько испытаний согласно ТЗ. Несмотря на то, что внешне программа выполняла свою функцию, была отмечена нестабильность работы — при тестировании приложения количество конфликтов достигало до 20 раз в минуту, что заставляло тормозить интерфейс пользователя. Алгоритм в текущем виде не проходил тестирование и требует доработки.

Также во время разработки была отмечена большая сложность отладки приложения в следствии его алгоритмической сложности, в том числе особую трудность вызвали «откатывания» внесенных изменений во время разрешения конфликта двойной записи.

2.1.4 Вторая модификация стратегии «копирование — изменение — слияние»

Чтобы решить проблемы первого модифицированного алгоритма было разработано оригинальное решение.

Основная проблема модифицированного алгоритма состояла в неправильном подходе к доступу «к общему» ресурсу — тексту, который считается «верным» (или синхронизированным). Другими словами требовалась взаимное исключение. В системах, состоящих из множества процессов, для

программирования взаимного исключения обычно проще всего использовать критические области. Когда процесс нуждается в том, чтобы обновить совместно используемые структуры данных, он сначала входит в критическую область. При взаимном исключении гарантируется, что ни один из процессов не использует одновременно с ним общие структуры данных. В однопроцессорных системах критические области защищаются семафорами, мониторами и другими примитивами синхронизации.

Взаимные исключения в распределенных системах обычно являются реализациями трех основных алгоритмов [3]:

- Централизованный алгоритм;
- Распределенный алгоритм;
- Алгоритм макерного кольца.

Централизованный алгоритм предполагает, что каждый раз, когда процесс собирается войти в критическую область, посылается сообщение координатору с запросом, в какую кри-тическую область данный процесс собирается войти, и запрашивается разрешение на это. Если ни один из процессов в данный момент не находится в данной критической области, координатор посылает ответ с разрешением на доступ.

Распределенный алгоритм работает следующим образом. Когда процесс собирается войти в критическую область, он создает сообщение, содержащее имя критической области, свой номер и текущее время. Затем он отсылает это сообщение всем процессам, концептуально включая самого себя. Вместо отдельных сообщений может быть использована доступная надежная групповая связь. Когда процесс получает сообщение с запросом от другого процесса, действие, которое оно производит, зависит от его связи с той критической областью, имя которой указано в сообщении. После посылки сообщения-запроса на доступ в критическую область процесс приостанавливается и ожидает, что кто-нибудь даст ему разрешение на доступ. После того как все разрешения получены, он может войти в критическую область.

Алгоритм макерного кольца предполагает, что программно создается логическое кольцо, в котором каждому процессу назначается положение в кольце, как показано на рисунке 4.

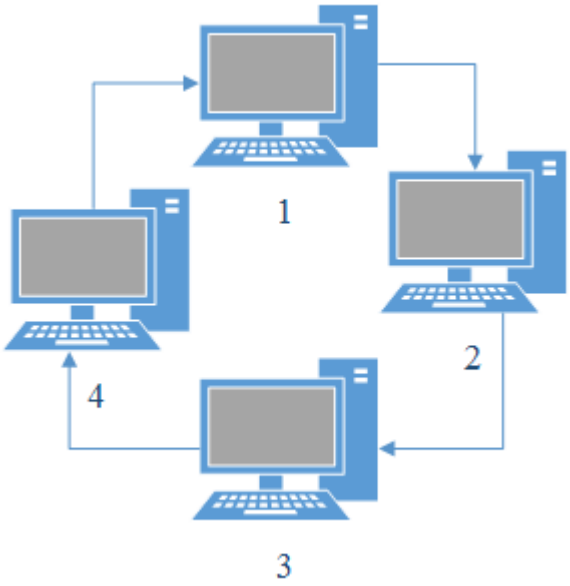

Рисунок 4 — Пример маркерного кольца, состоящего из четырех компьютеров

При инициализации кольца процесс 1 получает маркер. Маркер циркулирует по кольцу. Он передается от процесса k процессу k+1. Когда процесс получает маркер от своего соседа, он проверяет, не нужно ли ему войти в критическую область. Если это так, он входит в критическую область, выполняет всю необ-ходимую работу и покидает область. После выхода он передает маркер дальше. Входить в другую критическую область, используя тот же самый маркер, запрещается.

Централизованный алгоритм наиболее прост и эффективен. На то, чтобы войти в критическую область, ему достаточно трех сообщений — запрос, разрешение навход и сообщение о выходе. Но его использование в случае редактирования общего текста затруднено. Действительно, если количество одновременно работающих пользователей будет 3 и более, каждый пользователь прежде чем сможет внести свои правки, должен будет подождать свою очередь.

Распределенный алгоритм требует наличия полной упорядоченности событий в системе. Это вносит дополнительную сложность в реализации.

Алгоритм маркерного кольца так же как и централизованный алгоритм является простым в реализации и с первого взгляда его использование не дает никакого «выигрыша» по сравнению с централизованным алгоритмом, но это не так.

Конструктивное улучшение первого модифицированного алгоритма.

В качестве улучшения модфицированного алгоритма предлагается комбинированный алгоритм, который сочетает в себе централизованный подход и алгоритм маркерного кольца. Рассмотрим одного писателя P и координатора K. Предположим, что на компьтере писателя есть две версии текста: текущее состояние текста C_t (тот, который видит и редактирует пользователь) и текст, являющийся последней синхронизированной версией S_t между координатором и писателем. Другими словами, синхронизованная версия совпадает на компьютере координатора и писателя. Опишем основные шаги процедуры синхронизации текста C_t между координатором K и писателем P.

Для писателя P (см. вспомогательный рисунок 5):

1. Если имеется маркер: посчитать дельту Δ между C_t и S_t.
2. Полученный результат отправить координатору K и отдать маркер.

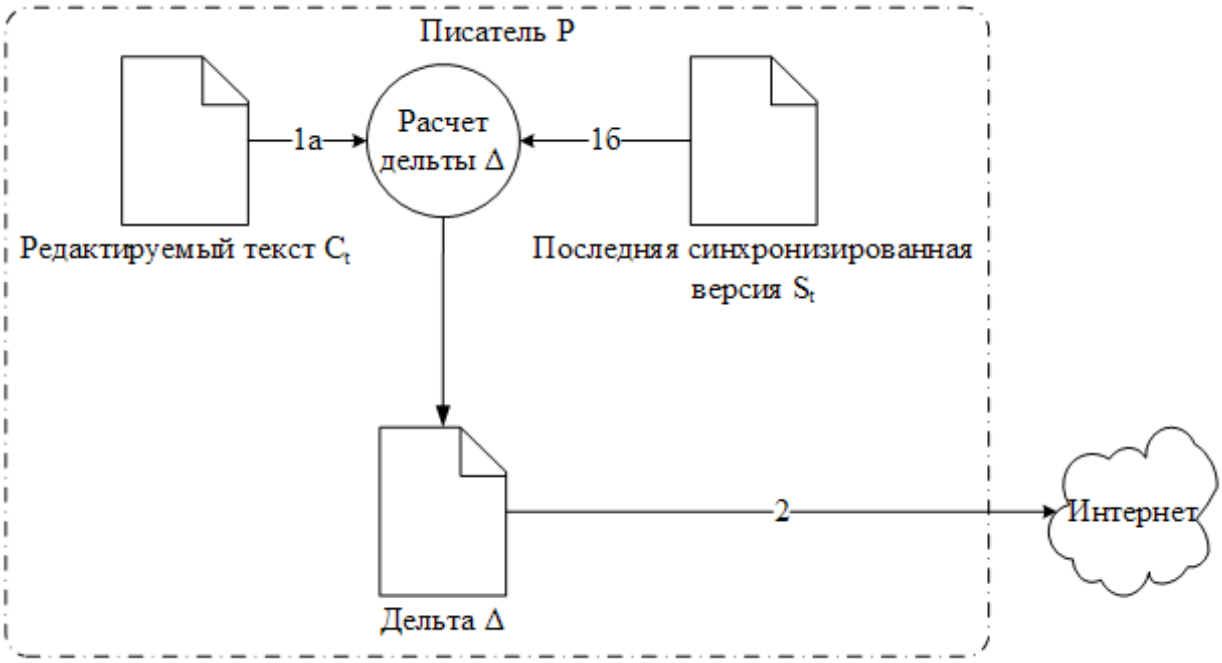

Рисунок 5 — Первая часть исправленного алгоритма (для писателя P)

Координатор K в свою очередь (см. вспомогательный рисунок 6):

1. Если имеется маркер: принять дельту Δ.
2. Применить дельту Δ к своему тексту $\boldsymbol{S_t}$, получив в результате $\boldsymbol{S_{t+1}}$.
3. Применить дельту Δ к своему тексту $\boldsymbol{C_t}$ с учетом ошибок контекста (см. раздел 2.1.2.2, на стр. 38), получив в результате $\boldsymbol{C_{t+1}}$.

4. Посчитать дельту $\boldsymbol{\Delta}$ между $\boldsymbol{C_{t+1}}$ и $\boldsymbol{S_{t+1}}$ и отправить писателю P вместе с маркером.

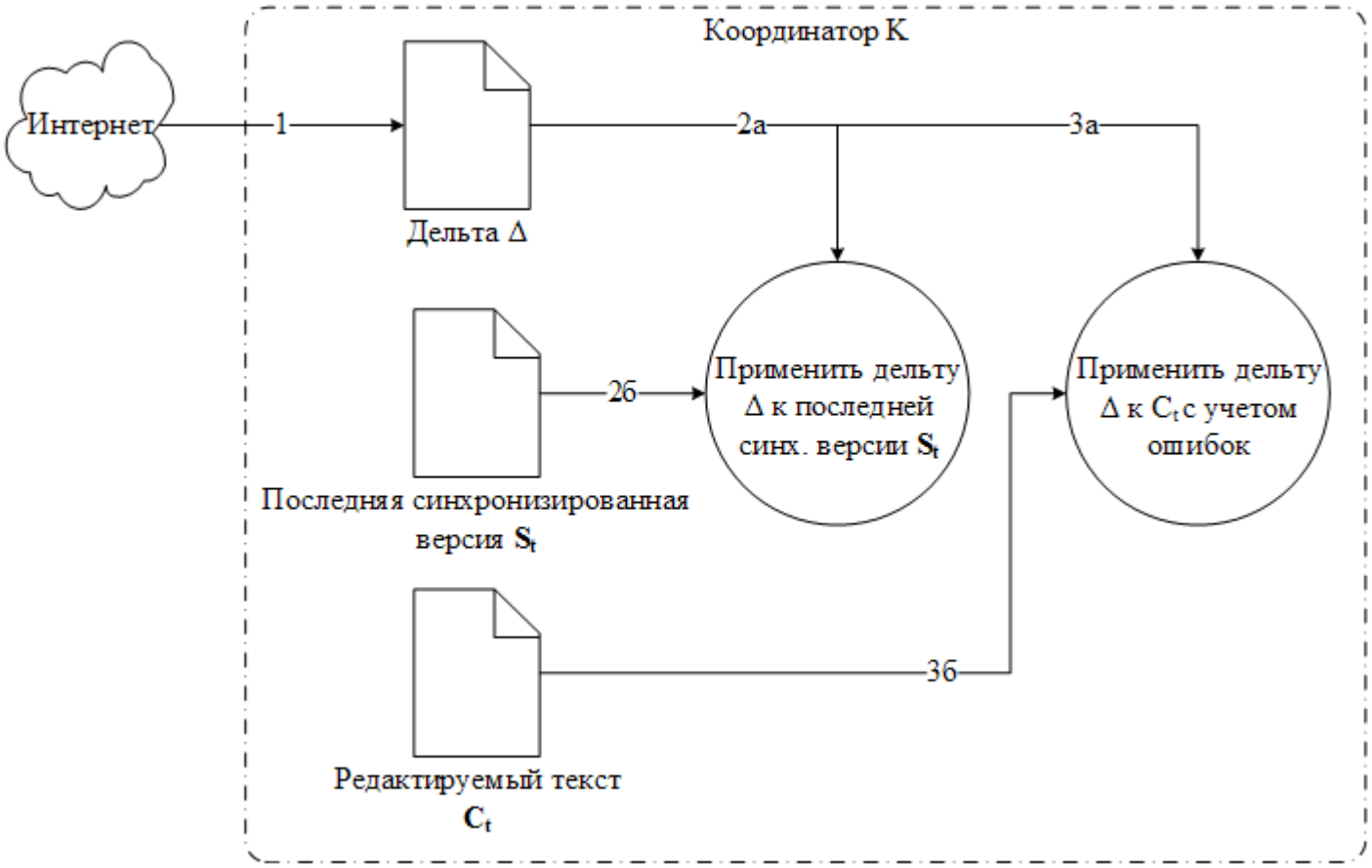

Рисунок 6 — Вторая часть исправленного алгоритма (для координатора K)

Как можно увидеть, поскольку $\boldsymbol{S_t} = S_t$, то $\boldsymbol{S_{t+1}} = S_{t+1}$. Это значит, что между координатором и писателем гарантируется наличие одинаковой синхронизированной копии. Следовательно, можно считать, что два удаленных процесса разделяют общий ресурс S_t так, если бы они выполнялись на одной машине и исключительный доступ до которого определялся маркером.

Данная процедура синхронизирует текст в одностороннем порядке. Чтобы получить изменения от координатора, необходимо запустить процедуру в обратную сторону, заменив местами P и K.

Концептуальное отличие данного алгоритма и первой моодификации состоит в наличии определенной последовательности работы писателя P и координатора K. Другими словами, существует маркер в логическом кольце

длины 2, состоящим из P и K. Благодаря тому, что глобально система стремится «сохранить» $\boldsymbol{S_t} = S_t$ и передаваемые дельты всегда применяются без конфликтов над текстом $\boldsymbol{S_t}$. Алгоритму не требуется дорогостоящая операция «откатывания» изменений при разрешении конфликтов двойной записи как в модифицированном варианте, поскольку в этом случае нет необходимости отменять внесенные изменения. Поскольку именно разрешение конфликтов двойной записи было узким местом модифицированного алгоритма, можно считать, что данный подход является более лучшим.

2.1.4.1 Масштабирование. Топология сети

Приведенная выше схема показывает синхронизацию лишь между двумя сторонами, либо пользователь и сервер, либо пара пользователей. Вторая модифицированная стратегия синхронизации может масштабироваться на неограниченное количество пользователей. В этом случае, если одновременно работает n писателей и один координатор, то координатор входит в n логических маркерных колец, образуя топологию «звезда» (см. рисунок 7). При этом текст координатора $\boldsymbol{C_t}$ для каждого кольца является общим. Когда первый писатель изменяет свой документ, текст координатора $\boldsymbol{C_t}$ обновляется и эти изменения становятся доступными всем остальным писателям.

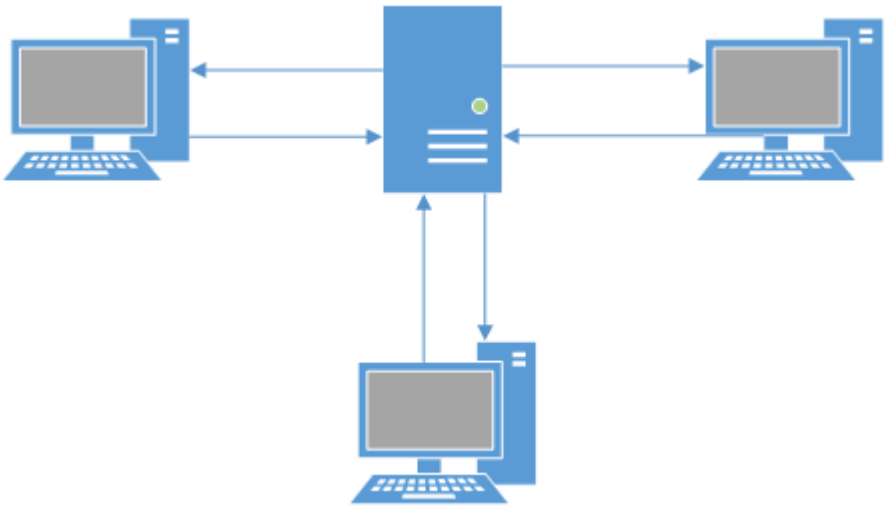

Рисунок 7 — Топология сети при одновременной работе трёх человек. В центре указан координатор

2.1.4.2 Ограничения и дальнейшая разработка

Одним из ограничений исправленного подхода, является то, что в каждый момент времени происходит передача информации лишь между координатором и одним из писателей. Это может стать проблемой в случае значительных задержек в связи. Одно из возможных путей модернизации алгоритма, это наличие возможности отправлять непрерывный поток обновлений в каждом из направлений, не дожидаясь наличия маркера.

Еще одно из направлений для улучшения — это добавление функциональной возможности отслеживать, какой пользователь был ответственен за какие правки общего документа. В настоящее время правки от всех пользователей объединяются в единую версию на координаторе без указания принадлежности. Наличие данной возможности поможет реализовать множество СКВ-подобных функций: визуально разделять правки разных пользователей, а также потенциально позволит отменять правки определенного писателя.

2.2 Анализ разработанного решения

2.2.1 Характеристики разработанного решения

Результаты тестирования производительности представлены в таблице 7.

Тестирование показало, что даже на машинах пятилетней давности можно работать над обычными текстовыми файлами количество символов которых не превосходит 50 тыс.

Следует учитывать, что все приведенные результаты получены при минимальной загруженности компьютера т. к. был запущен лишь один пользовательский процесс Sublime Text.

Таблица 7 — Результаты производительности

---	Конфигурация 1	Конфигурация 2
Процессор	Intel Celeron CPU 1007U 1.5ГГц	Core 2 Duo 3 ГГЦ
ОЗУ	4 ГБ	4 ГБ
Операционная система	Windows 7	Elementary OS Freya
Производительность при редактировании текста		
Текст объемом 44КБ (42796 символа)	Комфортная работа без задержек	
Текст объемом 88КБ (85592 символа)	Большие задержки при редактировании	Комфортная работа без задержек
Текст объемом 100КБ (97263 символа)	Работа с файлом невозможна	Большие задержки при редактировании

2.2.2 Соответствие техническому заданию

В соответствии с требованиями технического задания система совместного редактирования текстов «Collaboration» реализована на языке программирования Python в виде плагина к текстовому редактору Sublime Text.

Разработанная ИС позволяет одновременное редактирование текста двум и более пользователям.

Все три пункта функциональных требований выполнены. Был разработан специальный алгоритм, синхронизирующий текст между пользователями полностью автоматически.

Был проведен комплексный подход при разработке алгоритма, учтены и выполнены требования к надежности: алгоритм оказался устойчив относительно больших задержек в сети.

На основании приведенных результатов тестирования можно сделать вывод о том, что требования к производительности также выполнены.

Предусмотрены сочетания клавиш для основного функционала, а так же реализована возможность поиска координатора в локальной сети.

Несмотря на сложность алгоритма, система синхронизации работает без нареканий и проходит серию испытаний указанной в ТЗ без ошибок.

В итоге можно сделать вывод, что разработанная система совместного редактирования текстов удовлетворяет требованиям технического задания.

2.2.3 Возможное применение

Sublime Text является редактором исходного кода. Поэтому данный программный проект рассчитан в основном на программистов. Использование самого плагина предполагает редактирование как любого программного кода, так и обычного текста (имеется поддержка unicode). Поскольку Sublime Text обладает широкой поддержкой системы компьютерной верстки TeX, то плагин может быть востребован среди пользователей TeX для совместного написания статей. Таким образом основной целевой аудиторией являются команды писателей и программистов, работающих удаленно.

Одной из вторичных целей при разработке была возможность предоставить пользователями различных редакторов одинаковые возможности совместного редактирования. Другими словами, не ограничивать пользователей выбором лишь одного редактора. Поэтому разработка плагина велась с учетом его дальнейшего портирования на целое семейство бесплатных программ таких как gedit, Vim, Emacs и др.

БИБЛИОГРАФИЯ

1. Документы Google [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Google_Docs (дата обращения: 18.05.2015).

2. «A clean, lightweight alternative to Python’s twisted?» [Электронный ресурс]. URL: http://stackoverflow.com/questions/1824418/a-clean-lightweight-alternative-to-pythons-twisted.

3. Таненбаум Э., Ван Стеен М. Распределенные системы. Принципы и парадигмы. Питер, 2003. 877 с.

4. Ellis C. a., Gibbs S.J. Concurrency control in groupware systems // ACM SIGMOD Rec. 1989. Т. 18, № 2. С. 399–407.

5. Операциональное преобразование [Электронный ресурс]. URL: http://en.wikipedia.org/wiki/Operational_transformation (дата обращения: 18.05.2015).

6. Powers S. Unix Power Tools. O’Reilly Media, 2002.

7. Chacon S. Pro Git. Apress, 2009.

8. Lindholm T. A three-way merge for XML documents // Proc. 2004 ACM Symp. Doc. Eng. - DocEng ’04. 2004. № October. С. 1.

9. Документация утилиты GNU Patch [Электронный ресурс]. URL: http://savannah.gnu.org/projects/patch/ (дата обращения: 18.05.2015).

10. Fredriksson K., Grabowski S. Average-optimal string matching // J. Discret. Algorithms. Elsevier, 2009. Т. 7, № 4. С. 579–594.

11. Макконнелл С. Совершенный код. Мастер-класс. Москва: Издательство «Русская редакция», 2010. 896 с.

12. Пузыревский И. Асинхронное программирование [Электронный ресурс]. URL: https://events.yandex.ru/lib/talks/1760/ (дата обращения: 18.05.2015).