Санкт-Петербургский Государственный Университет

Математико-Механический факультет

Создание инструментов версионирования визуальных моделей в QReal

Курсовая работа студента 344 группы

Зимина Григория Александровича

Научный руководитель: асп. Д.А. Мордвинов

Санкт-Петербург

2015 год

Оглавление:

- Введение
- Обзор существующих решений
  - Visual Paradigm
  - ClioSoft’s Visual Design Diff (VDD)
  - Google Docs
  - Текущая реализация
- Архитектура
  - Архитектура интегрирования версионирования в проект
  - Архитектура для упрощенного режима работы
- Реализация
  - Реализация подключаемого модуля
  - Оптимизация сохранения для поддержки версионирования
  - Реализация взаимосвязи модуля версионирования и модуля для графического показа изменений
    - Решение конфликтов
    - Анализ сравнения визуальных моделей
    - Оптимизация сравнения визуальных моделей
- Итоговые варианты режимов работы
  - Упрощенный режим
  - Режим расширенного интерфейса
- Заключение
  - Итоги работы:
  - Пути развития:
- Список литературы и источников

Введение

Подавляющее большинство программных продуктов в мире написано на текстовых языках программирования. Версионирование кода здесь - общепринятая практика. Для этого существует множество средств - систем контроля версий (version control system - VCS), таких как Git, SVN (Subversion), Bazaar, и другие. Системы контроля версий помогают с легкостью отслеживать изменения в исходном коде, вернуться без потерь назад, наладить коммуникационный мост между группой программистов занимающихся одним проектом.

Но также существуют визуальное программирование, где программист оперирует моделями для создания программ, решая задачу на уровне алгоритмов, а не на уровне программного кода: Дракон-схемы, последовательные функциональные схемы (SFC— Sequential Function Chart), и другие. Здесь практика версионирования не так распространена.

После генерации программного кода по моделям, его так же можно версионировать (программисту придется разбираться в том, что сгенерировал генератор кода по модели, а для перехода обратно требуется генератор моделей по коду - также нетривиальная задача).

Но гораздо удобнее не переходить от уровня моделей к уровню кода и обратно, а версионировать такие программы на уровне моделей. Однако на данный момент не существует готовых решений пригодных для версионирования произвольных моделей. Поэтому каждая среда визуального программирования вынуждена решать эту задачу заново, либо пренебрегать подобной функциональностью ввиду её сложности. Но существуют инструменты, разработанные для определенных систем, они будут рассмотрены далее.

Постановка задачи

На кафедре системного программирования математико-механического факультета СПбГУ разрабатывается DSM-платформа QReal — инструмент, позволяющий быстро создавать новые визуальные языки и поддерживать их.

Система уже умеет сохранять версии с помощью централизованной системы контроля версий Subversion и визуально сравнивать модели (курсовая работа 3 курса Д.А. Мордвинова)[1].

Использование традиционных систем контроля версий обусловлено тем, что на диске модель хранится в виде дерева из каталогов и xml-файлов. Однако чтобы превратить версионирование кода в версионирование визуальных моделей, необходимо создавать интерфейсы взаимодействия с внешними клиентами контроля версий.

Цель курсовой работы состоит в следующем. Во-первых, требуется добавить поддержку нового подключаемого модуля для работы с внешним клиентом контроля версий Git. Необходимо реализовать поддержку коллективной работы над проектом, а именно основные возможности распределенной системы контроля версий, в том числе возможность работы с ветвями разработки. Во-вторых, так как проект QReal в том числе нацелен и на обучение программированию, и не все пользователи имеют опыт работы с системами контроля версий, то наряду с расширенным режимом работы, который обеспечит доступ к основным функциям системы контроля версий, необходимо реализовать упрощенный режим работы. Пользователь должен иметь возможность посмотреть хронологию изменений, узнать об отличиях и иметь возможность откатиться к выбранной версии. Для этого требуется расширить существующий интерфейс визуализации изменений. В-третьих, провести апробацию новой функциональности и оценить её влияние на работу системы.

Обзор существующих решений

На данный момент существует несколько систем, которые реализуют похожую функциональность.

Visual Paradigm

Система Visual Paradigm (VP-UML)[9] — кроссплатформенное средство визуального моделирования. Оно позволяет собирать требования к системе, создавать документацию, поддерживать совместную разработку. В том числе оно позволяет версионировать проект и предоставляет инструмент для визуального сравнения Visual Diff (рис. 1).

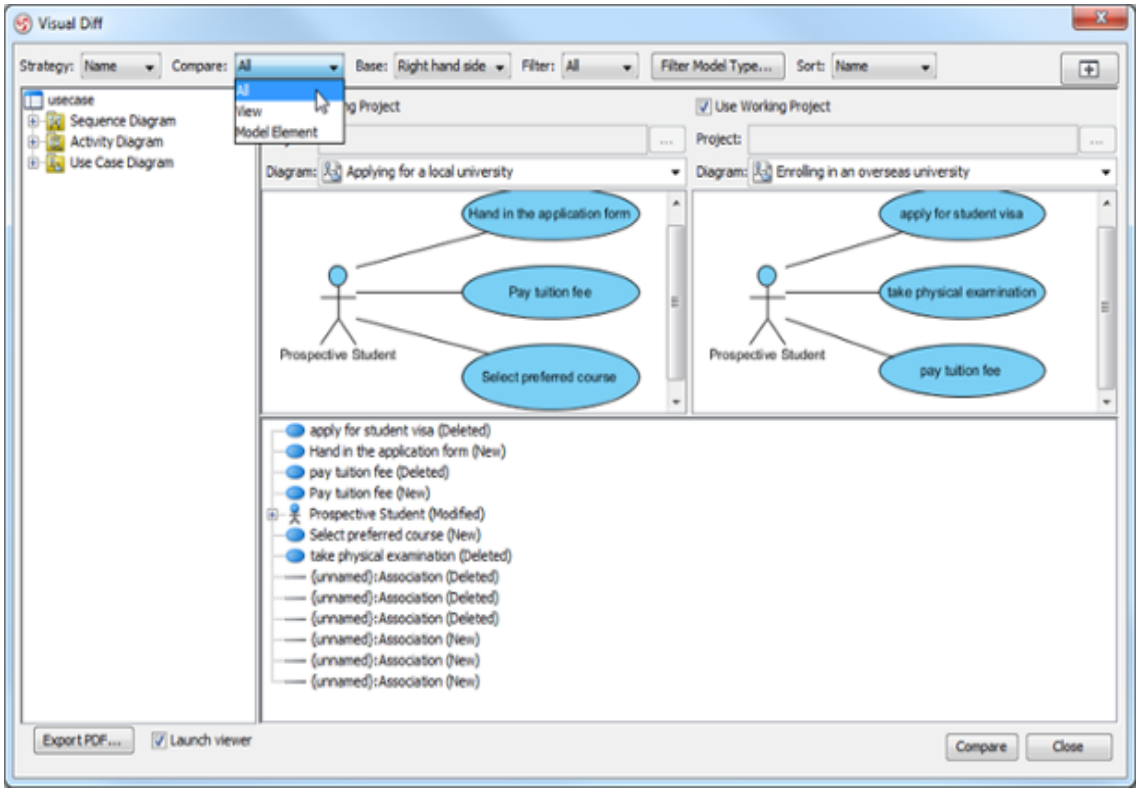

Рисунок 1. Окно работы инструмента версионирования VP-UML. Скриншот взят с сайта visual-paradigm.com[9].

К преимуществам этого инструмента можно отнести удобный графический интерфейс: подсветка измененных элементов и формирование подробного списка изменений в свойствах элементов; выбор типа разницы для отображения: графические изменения, изменения в модели или и то, и другое. Также имеется возможность сортировки результатов сравнения и упорядочивания их по категориям, возможность экспорта результатов сравнения в PDF.

ClioSoft's Visual Design Diff (VDD)

SOS Design Data Collaboration Platform [4] — кроссплатформенная платформа для управления аппаратной конфигурацией для аналоговых, смешанных и цифровых устройств. Платформа предоставляет среду разработки, которая обеспечивает возможности для совместной разработки и эффективного управления проектом от концепта устройства до его выпуска.

Для поддержки версионирования и совместной разработки платформа реализует следующий функционал: контроль версий файлов и директорий, которые отвечают за представление элементов на диске, в том числе отслеживание переименований или перемещений файлов; поддержка работы с ветвями разработки, в том числе создание, переключение и слияние ветвей; возможность отката к предыдущей версии проекта.

Но данная платформа имеет узкую направленность на язык Verilog.

Также компания предоставляет инструмент ClioSoft Visual Design Diff[5], который дает пользователям возможность сравнить две версии схемы, графически выделяя различия в редакторе (рис. 2).

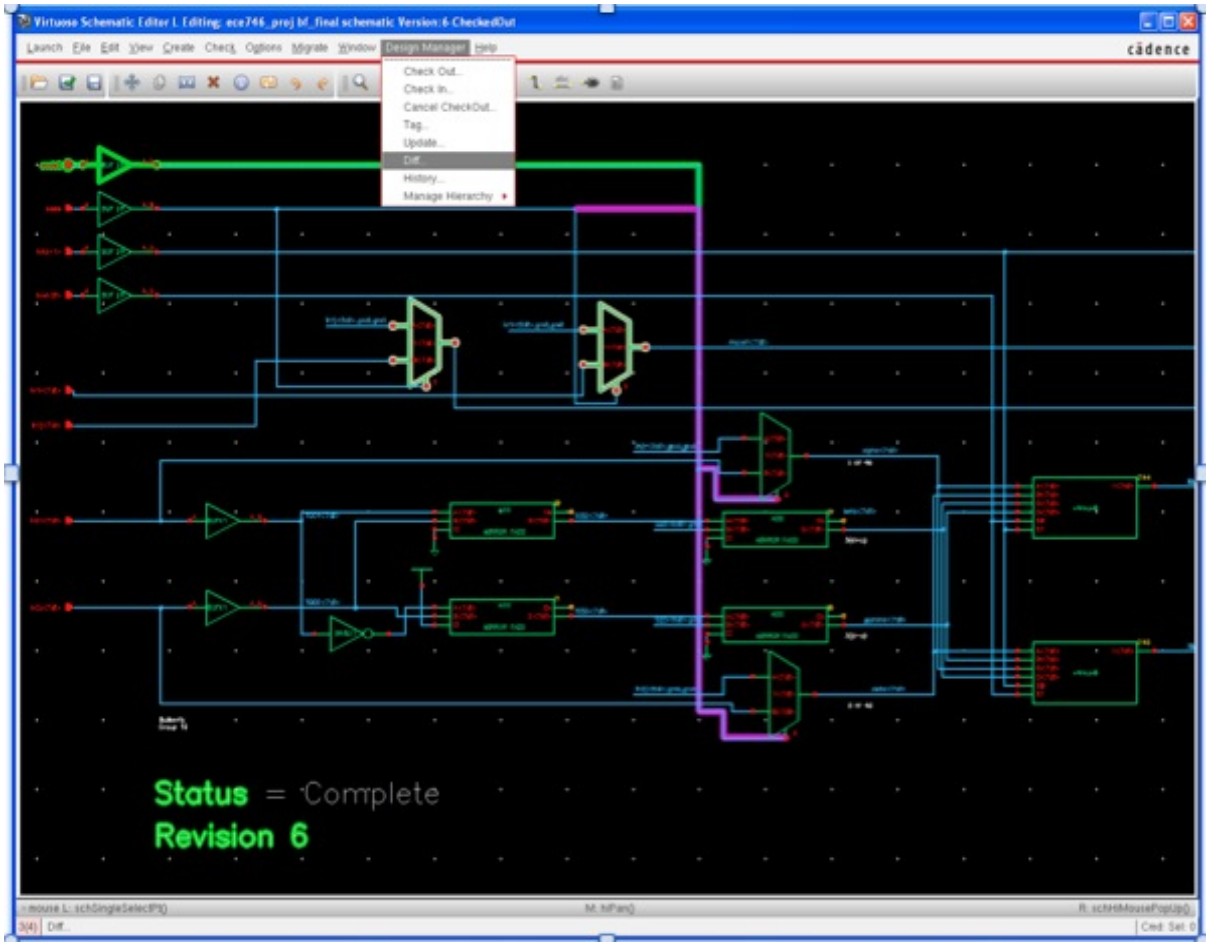

Рисунок 2. Окно работы визуального сравнения компании ClioSoft. Скриншот взят с сайта semiwiki.com[8]

Google Docs

Подсветка измененных частей натолкнула на еще один вариант визуализации изменений. Как удачный пример пользовательского интерфейса был рассмотрен Google Docs[6] (рис. 3).

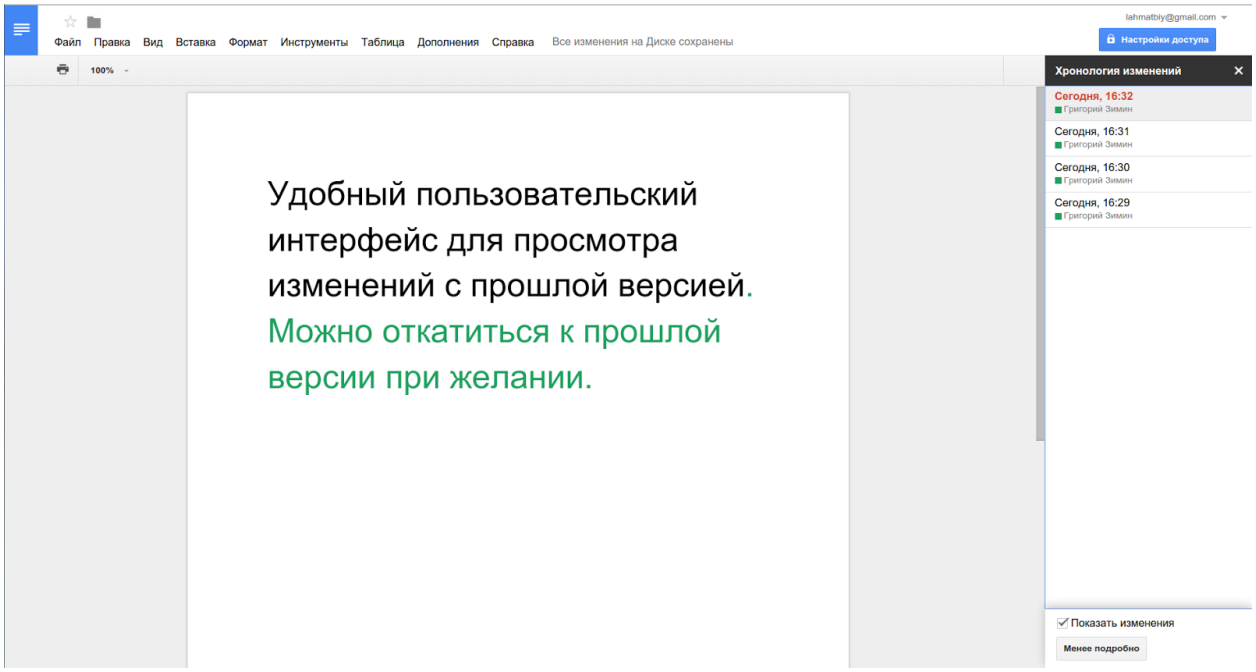

Рисунок 3. Google Docs.

При работе с документом Google Docs в фоновом режиме версионирует все изменения. Благодаря этому можно просматривать хронологию изменений и при желании загрузить версию, нажав на кнопку “восстановить эту версию”, выбранную в списке.

Было отмечено, что любое действие сохраняется, и можно просмотреть подробный список изменений, а можно краткий, содержащий весомые изменения.

Текущая реализация

В существующем решении была реализована инфраструктура для внедрения модулей внешних клиентов систем контроля версий. Отдельно модули версионирования позволяют сохранять версии и работать с ними, но не имеют возможности отобразить разницу между версиями графически.

Ввиду несовершенства ядра системы на момент написания работы Д.А. Мордвинова не представлялось возможности вынести сравнение визуальных моделей

и отображение графической разницы в отдельный модуль. Это осложняло гибкую настройку связей модулей версионирования и интсрументов визуального сравнения.

В существующем варианте реализации визуализации сравнения двух версий пользователю предлагалось диалоговое окно, на котором были показаны две версии и можно было посмотреть детали конкретных различий [1] (рис. 4).

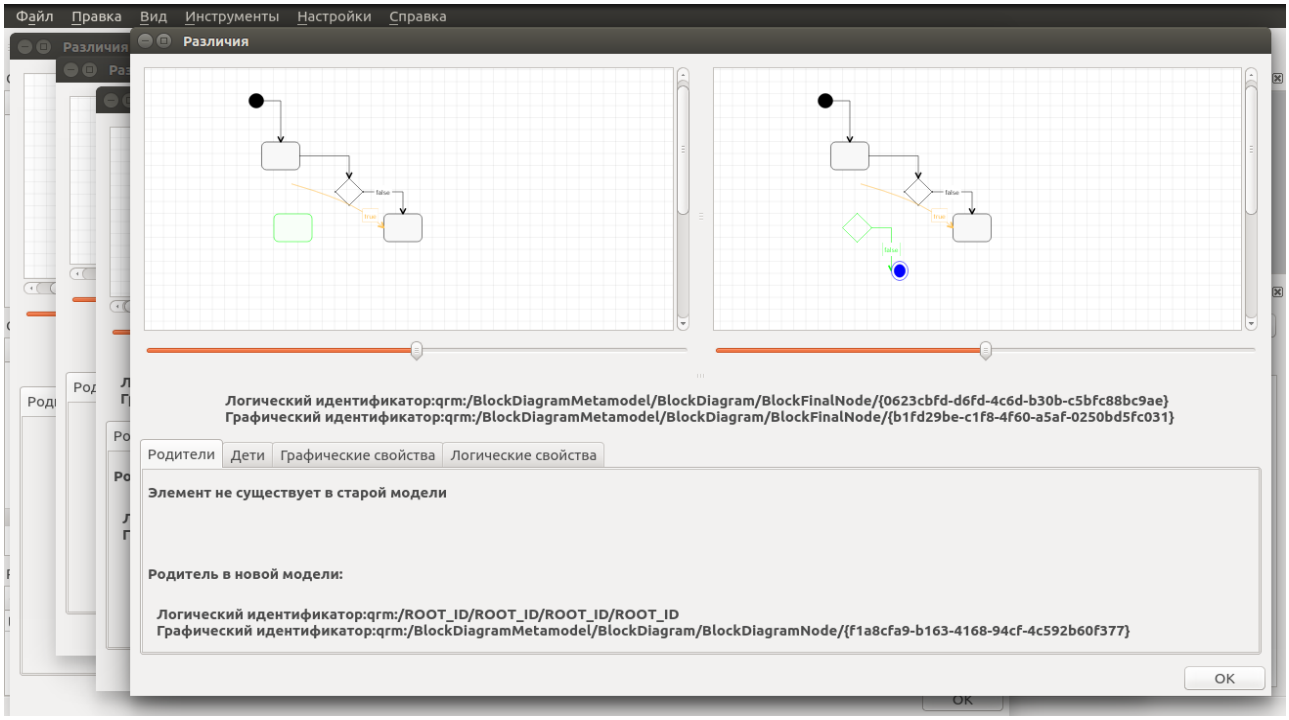

Рисунок 4. Существующее решение.

К общим плюсам для упрощенного и для расширенного режимов можно отнести “детали”, где можно подробно посмотреть об изменениях свойств конкретных элементов (нижняя часть рис.4) и подсветку измененных элементов модели.

Существующий вариант удобен для просмотра разницы, но не подходит для быстрого просмотра версий, например как это делает Google Docs. Так же отсутствовала возможность редактировать модели. Не было унифицированной визуализация изменений и отсутствовала асинхронная загрузка моделей.

К основным минусам работы упрощенного режима было решено отнести то, что ручной ввод версий для сравнения полезен лишь для опытного пользователя, т.к. если

для SVN версии — это номера ревизий, то у Git-а версии — это хеши коммитов. Начинающего пользователя это, скорее всего, оттолкнет. Логично ручной ввод версий использовать в расширенном режиме работы, чтобы дать больше свободы, но в упрощенном режиме работы лучше оградить пользователя от излишних действий.

Архитектура

Архитектура интегрирования версионирования в проект

Существующая архитектура была частично реструктурирована, опишем всю архитектуру более подробно (рис. 5).

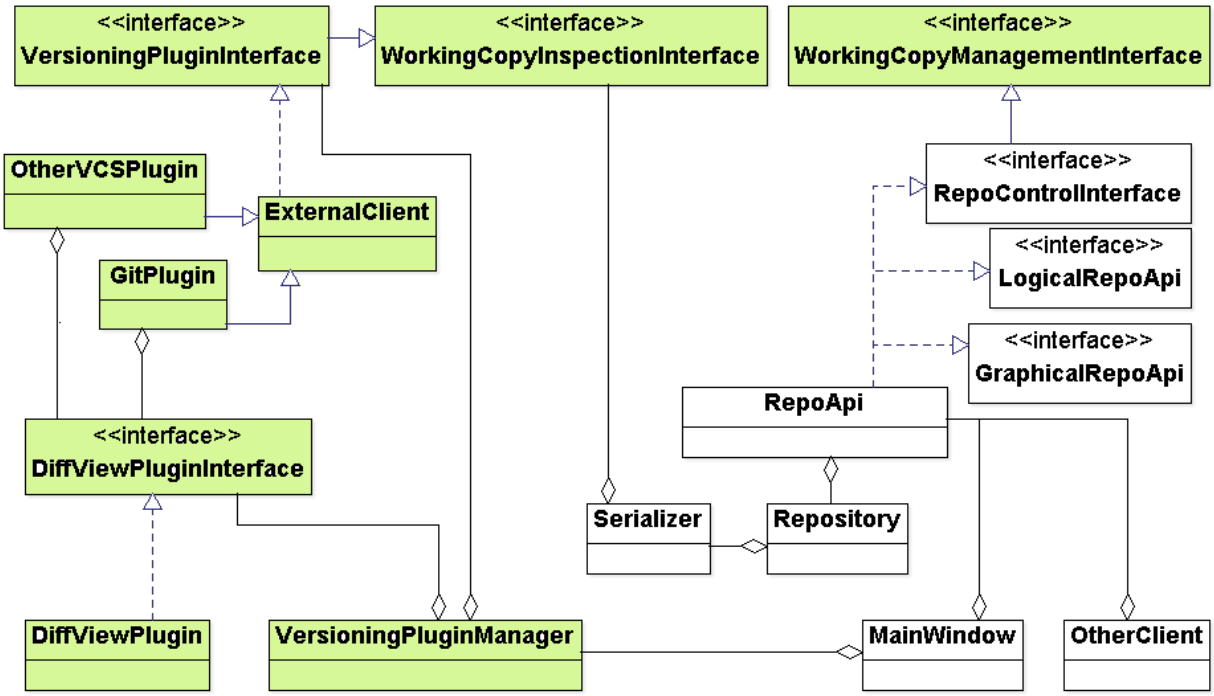

Рисунок 5. Архитектура. (Белый — исходная архитектура до версионирования, зеленый — изменения)

Как видно из архитектуры, для поддержки нового клиента системы контроля, необходимо реализовать всего один модуль.

Для внедрения систем контроля версий был расширен интерфейс сериализатора — класса сохраняющего проект на диск, чтобы получить возможность распаковывать и запаковывать проект для работы внешних клиентов версионирования. Так как во время работы с моделью не происходит обращений к диску, то системы контроля версии работают с информацией, полученной при последнем сохранении.

Чтобы при сохранении сообщать системе контроля версий об изменениях в файлах соответствующих элементам модели, сериализатор агрегирует в себе интерфейс, который должны реализовать все модули версионирования, позволяющий сообщать модулям о добавлении/удалении/модификации файлов. Таким образом, в процессе сохранения системы контроля версий версионируют измененные файлы.

Вся общая функциональность внешнего абстрактного клиента системы контроля версий инкапсулирована в классе ExternalClient. Он обрабатывает изменения файлов, перенаправляя запрос активному клиенту системы контроля версий. И создает процессы для выполнения команд управления внешних клиентов систем контроля версий, синхронно или асинхронно их выполняя. Модули конкретных систем контроля версий наследуются от него и переопределяют методы управления командами специфичными для конкретных клиентов.

Изменилась концептуальная идея связи подключаемых модулей, отвечающих за клиенты версионирования и подключаемых модулей, отвечающих за сравнение моделей и визуализацию изменений. Теперь каждый подключаемый модуль внешнего клиента контроля версий агрегирует в себе модуль визуализального сравнения моделей, таким образом добавилась гибкость выбора модуля, отвечающего за визуализацию изменений и за сравнение моделей. Это дало возможность модулям версионирования напрямую вызывать методы сравнения моделей. За настройку и работу версионирования и его связь с визуальным сравнением моделей отвечает менеджер версионирования. Это единственная точка связи системы с возможностями версионированием.

Кроме того, в целях повышения удобства работы системы с инфраструктурой версионирования был расширен интерфейс взаимодействия с внешним клиентом. Был добавлен метод, позволяющий определить существует ли в системе пользователя клиент подключаемого модуля, чтобы иметь возможность отключать подключаемые модули внешних клиентов контроля версий. Также был добавлен метод, возвращающий имя внешнего клиента, чтобы различать активные клиенты. Для работы с любым режимом (автоматизированным или расширенным) был расширен интерфейс модуля, отвечающий за построение разницы моделей. Кроме того был переписан

класс, реализующий функциональные объекты для асинхронного выполнения процессов клиентов систем контроля версий.

Архитектура для упрощенного режима работы

Для создания упрощенного режима работы версионирования была создана обертка подключаемого модуля внешнего клиента с ограниченным набором методов, нужных только для этого интерфейса. (рис. 6).

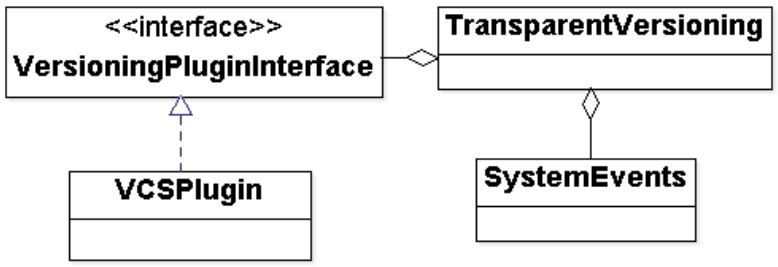

Рисунок 6. Архитектура прозрачного режима.

Существуют несколько стратегий сохранения версии в фоновом режиме: сохранять версию при любом изменении, сохранять при фиксированном количестве произведенных изменений, и другие. Сейчас мы ограничиваемся сохранением версии при явном нажатии пользователя на кнопку “сохранить” и при автосохранении проекта. Чтобы понять, что пользователь сохранил проект, класс прозрачного версионирования подписывается на сигналы о системных событиях, а именно на сигнал, что сохранение выполнено, чтобы добавить версию в систему контроля.

Это решение не идеально, но задача создания стратегий сохранения версий в фоновом режиме — отдельная тема для исследований, и в рамках этой курсовой работы она не решалась.

Реализация

Реализация подключаемого модуля

Реализация подключаемого модуля сводилась к реализации интерфейса, который должны уметь поддерживать все системы контроля версий, добавленные как отдельные модули.

К тривиальным шагам относятся реализация запуска инициализации локального репозитория, добавление коммита, реализация других простых операций[1]. Рассмотрим нетривиальные вопросы. Возникает проблема, когда хочется сделать не совсем привычные для системы контроля версий операции. Задача реализовать упрощенный режим работы подразумевает, что версионирование моделей должно работать и в отсутствии удаленного репозитория, и не требовать подключения к сети. Также хотелось бы научиться хранить все изменения в проекте в последовательном хронологическом порядке.

Рассмотрим ситуацию: локальный и удаленный репозитории синхронизированы, мы хотим откатиться до прежней версии, выполняя в локальном репозитории команду git reset --hard <v> (команда откатывает файлы репозитория к состоянию версии — <v>, удаляя несуществующие на тот момент файлы). В этот момент в локальном репозитории теряются все сведения о коммитах поздних версий проекта (формально это не совсем так, их можно еще получить, но после прохода сборщика мусора информация будет утеряна). При наличии удаленного репозитория это не является проблемой, так как мы можем получить их, синхронизируясь. Проблема стала понятной: нужно найти способ сохранять информацию о ранних версиях, при откате к старой, локально. Так же необходимо решить задачу составления хронологического списка .

Были рассмотрены 3 решения:

1. Можно хранить хэш самого старого коммита, таким образом при следующем

[1] Вся необходимая документация по коммандам git бралась в книге “Pro Git” Scott Chacon[3].

выборе версий мы сможем показать весь список сделанных коммитов в надежде, что наш следующий запрос к списку версий будет раньше, чем пройдет сборщик мусора операционной системы. Решение работает в частных случаях, но если сборщик мусора пройдет по локальному репозиторию раньше, пользователь может потерять часть своей работы безвозвратно. Или же можно требовать для версионирования доступ к удаленному репозиторию и все изменения фиксировать и в удаленном репозитории, а при необходимости синхронизировать их.

2. Второе решение подразумевает использовать ветви разработки. Каждый раз при запросе отката к более поздней версии будем создавать новую ветвь и там выполнять откат. Очевидно, что данные мы сохраним. Но с другой стороны, только после одного отката, мы получим 2 ветки, а линейный по количеству запросов на откат рост ответвлений приведет к дереву с большим количеством веток. В расширенном режиме работы такое решение сродни обычному использованию ветвей в системе контроля версий Git, и для опытного пользователя не составит труда. Но если мы, к примеру, хотим назначить каждому пользователю свою ветвь разработки — предположим, в разных режимах работы, то нагромождение ветвей, созданных в автоматизированном режиме, будет излишним.

3. Третий вариант состоит в том, чтобы создавать ветку, в ней выполнять откат, а потом переносить первый коммит к голове текущей рабочей ветки, и после удалить временную ветку. На рисунке 7 операция представлена разбитой на 4 состояния: главная ветка (рис. 7.а); главная ветка и временная ветка с отмеченной для отката второй версией (рис. 7.б); главная ветка и временная ветка с откатом до версии (рис. 7.в); версия перенесена в главную ветку, временная ветка удалена (рис. 7.г).

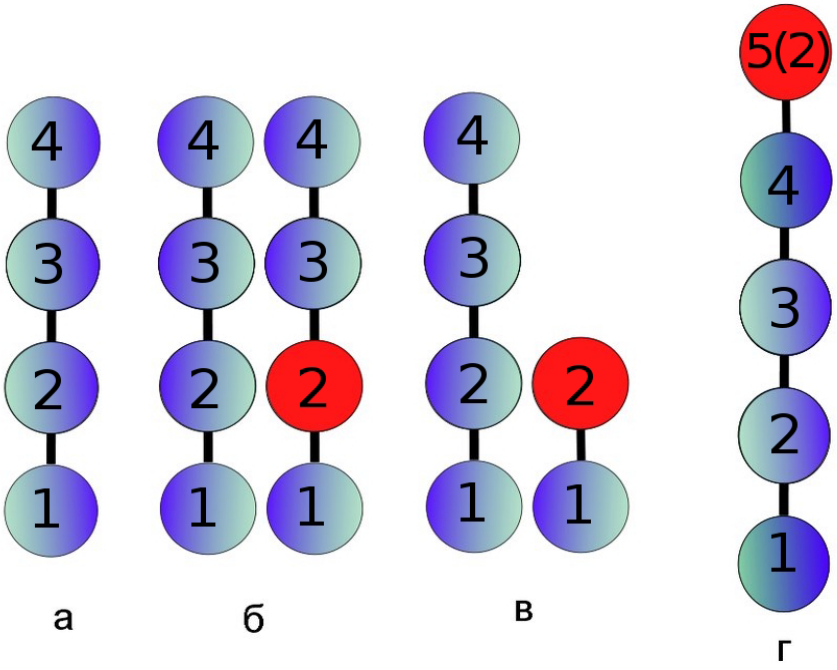

Рисунок 7. Процесс отката к поздней версии.

Было принято решение реализовать третий метод, так как несмотря на довольно-таки изощренный метод получения от ветки лишь одного коммита, дерево коммитов не загромождается большим количеством новых веток и мы не требуем подключения к сети Интернет и наличия удаленного репозитория.

Еще одним важным с точки зрения пользователя шагом было решение прятать возможность версионирования в инструментах меню, если путь к клиенту был не указан. Мотивация проста: не показывать неработающие кнопки. Но это решение программа принимает в настоящем времени, то есть если во время работы пользователь укажет верный путь до клиента, то возможность версионировать модели тут же станет доступной.

Оптимизация сохранения для поддержки версионирования

После профилирования процесса сохранения модели при сохранении 10-100-500

элементов были выявлены существенные недостатки при добавлении, модифицировании или удалении элементов в текущей версии в системе контроля версий. Добавление поэлементных изменений в новой версии модели существенно замедляло ход сохранения, так как для каждого элемента создавался новый процесс внешнего клиента, что добавляло накладные расходы. Теперь элементы для добавления, модифицирования или удаления собираются в контейнеры и передаются параметрами в методы клиента контроля версий, который в свою очередь должен оптимально обработать изменения. Суть в том, что обычно системы контроля версий могут обновлять/удалять/добавлять файлы списками, тем самым мы уходим от проблемы создания множества процессов. Это сократило время сохранения в 10 и более раз (таб. 1).

Кол-во эл-в	Без версионирования		С версионированием			
			До оптимизаций		После оптимизаций	
	Среднее знач., мс	σ, мс	Среднее знач., мс	σ, мс	Среднее знач., мс	σ, мс
10	29	2	828	26	167	8
100	132	9	6981	203	452	23
500	705	68	51к	6к	1903	232

Таблица 1. Сравнение времени сохранения с версионированием и без. Значения округлены до мс. Количество испытаний — 10. Испытание: в новый проект добавлялось k элементов и выполнялось сохранение.

В целом, интегрирование версионирования в ход сохранения проекта на диск замедляет работу сохранения в 3-6 раз (таб. 1). Это происходит из-за необходимости распаковывать прошлое сохранение, чтобы найти удаленные, модифицированные и добавленные элементы, и вызывать методы внешнего клиента версионирования для модификации их состояний под контролем системы контроля версий. В целом это приемлемо для небольших проектов. Но хорошей оптимизацией было бы изменение

концепции сохранения: сделать его асинхронным, так как если речь пойдет о сохранении тысячи и более элементов, то сохранение будет оцениваться секундами. Блокировка основного потока графического пользовательского интерфейса на долгое время будет неприемлемой.

Реализация взаимосвязи модуля версионирования и модуля для графического показа изменений

Как уже говорилось ранее, было решено, что модуль версионирования, обеспечивающий управление внешним клиентом, должен быть неотделим от модуля, отвечающего за показ изменений. С одной стороны, это разные подключаемые модули, и лучше бы уменьшить их связанность, но с другой стороны из-за их неотделимости нет необходимости усложнения вызова методов через менеджер подключаемых модулей. Но и сливать воедино оба модуля не следует, так как это уменьшит возможность их переиспользования, поэтому каждый модуль внешнего клиента агрегирует интерфейс визуального сравнения моделей.

Модуль демонстрации изменений нужен модулю внешнего клиента версионирования в случаях, когда пользователь хочет сравнить два разных коммита из истории изменений последовательных или непоследовательных или сравнить текущую диаграмму с последней или произвольной версией, сохраненной системой контроля версий.

Решение конфликтов

Также модуль сравнения моделей нужен, когда пользователь хочет слить две ветви разработки (при явном слиянии ветвей или же при процессе запроса обновлений из удаленного репозитория). Если при слиянии возникнет конфликт в xml-файле, отвечающем за представление элемента диаграммы, то мы не сможем открыть диаграмму — необходимо будет решить конфликты.

Можно решать конфликты в текстовой форме, но тогда нужно дорабатывать xml-представление элементов, и часть нерешенных системой конфликтов просить пользователя решить вручную. Это решение противоречит идее визуального программирования: пользователю придется разбираться в тексте. Такая же проблема возникает, если мы будем генерировать код по модели, и пытаться решить конфликты в коде, опять же предоставив пользователю решать часть конфликтов. Более того, не все диаграммы могут генерироваться из кода.

Решено было использовать поиск различий в моделях с целью найти изменения, чтобы показать конфликты. Если различий в двух моделях нет, то проблема не носит семантический характер (например, проблема в порядке сериализации), и мы можем слить ветви, решая конфликты в любую из сторон.

Процесс выполняется в три шага. Сначала происходит попытка слить две ветви автоматически, если конфликтов не возникло, то слияние завершается. Затем, после построения модели разницы, происходит проверка существования семантических конфликтов. Если их нет, то происходит повторная попытка слить ветви с решением конфликтов в одну из сторон. Иначе система просит пользователя разрешить конфликты самостоятельно (рис. 9).

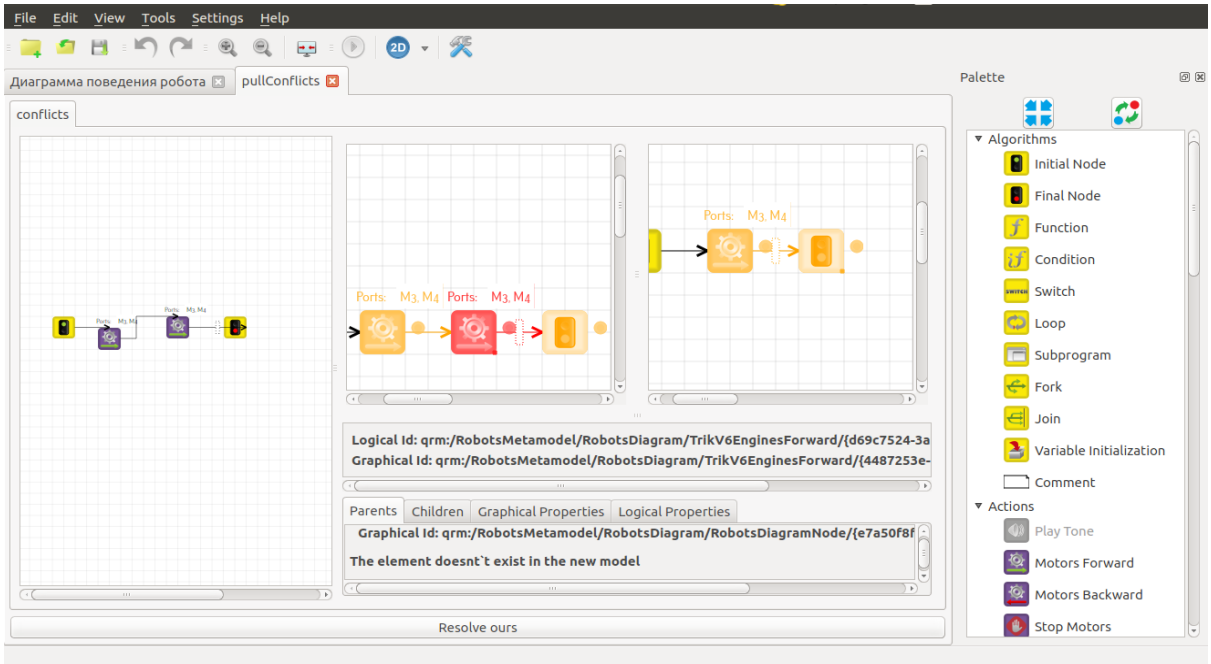

Рисунок 9. Окно решения конфликтов.

С помощью “деталей” пользователь сможет просмотреть отличия в графическом и логическом представлениях элемента, решить конфликт в диаграмме и завершить процесс слияния двух ветвей разработки.

После решения пользователем конфликтов, снова выполняется слияние ветвей разработки, но уже с решением конфликтов в сторону пользователя.

Анализ сравнения визуальных моделей

Имея две загруженные в память модели. Можно запросить у них информацию о всех элементах, информацию об их отношении родства (предок-потомок) и об их свойствах. Это дает достаточно информации для построения модели разницы. Стадия сопоставления элементов тривиальна, так как QReal присваивает всем элементам при создании уникальный идентификатор.

Стадия сопоставления элементов по уникальному идентификатору, присваиваемому каждому элементу в системе QReal при создании, дает хорошие результаты. Но это решение так же имеет некоторые недостатки. Рассмотрим две ситуации. Пользователь совершает процесс рефакторинга (выносит часть элементов в подпрограмму). Или пользователь переносит часть элементов диаграммы в новую диаграмму. Система изменит идентификаторы всем переносимым элементам, но элементы не поменяются, хотелось бы уметь это обнаруживать. В первом случаем можно реализовать процесс, так чтобы идентификатор элемента не менялся, и тогда проблема будет решена автоматически. Во втором случае это сделать сложнее. Чтобы находить и сопоставлять такие элементы, можно решить задачу поиска максимального изоморфного подграфа двух графов [7], например, представив модели как графы, где вершинами будут элементы с их свойствами, а ребрами — связи потомок-предок. неотделимости нет необходимости усложнения вызова методов через менеджер подключаемых С другой стороны мы покажем пользователю, что эти элементы появились в диаграмме как новые, что является приемлемым результатом сравнения. Поэтому в этом решении нет острой необходимости, но в целом, такой инструмент был

бы полезен, и в планах дальнейшего развития на него можно обратить внимание.

Оптимизация сравнения визуальных моделей

После профилирования процесса построения модели разницы и создания окна изменений, было обнаружено, что до 90% времени тратится на инициализацию жестов мышью, это происходит, так как представление системы сравнения диаграмм — это надстройка над представлением редактора диаграмм. После добавления возможности выключать инициализацию жестов мышью это “бутылочное горлышко” было устранено.

Итоговые варианты режимов работы

Упрощенный режим

В итоге, объединив плюсы всех рассмотренных аналогов, решено было дать пользователю в автоматизированном режиме работы возможность последовательно просматривать изменения от версии к версии, и при желании загрузить любую из них, а также оставить возможность просмотреть детали (по умолчанию они свернуты в целях экономии места) (рис. 9).

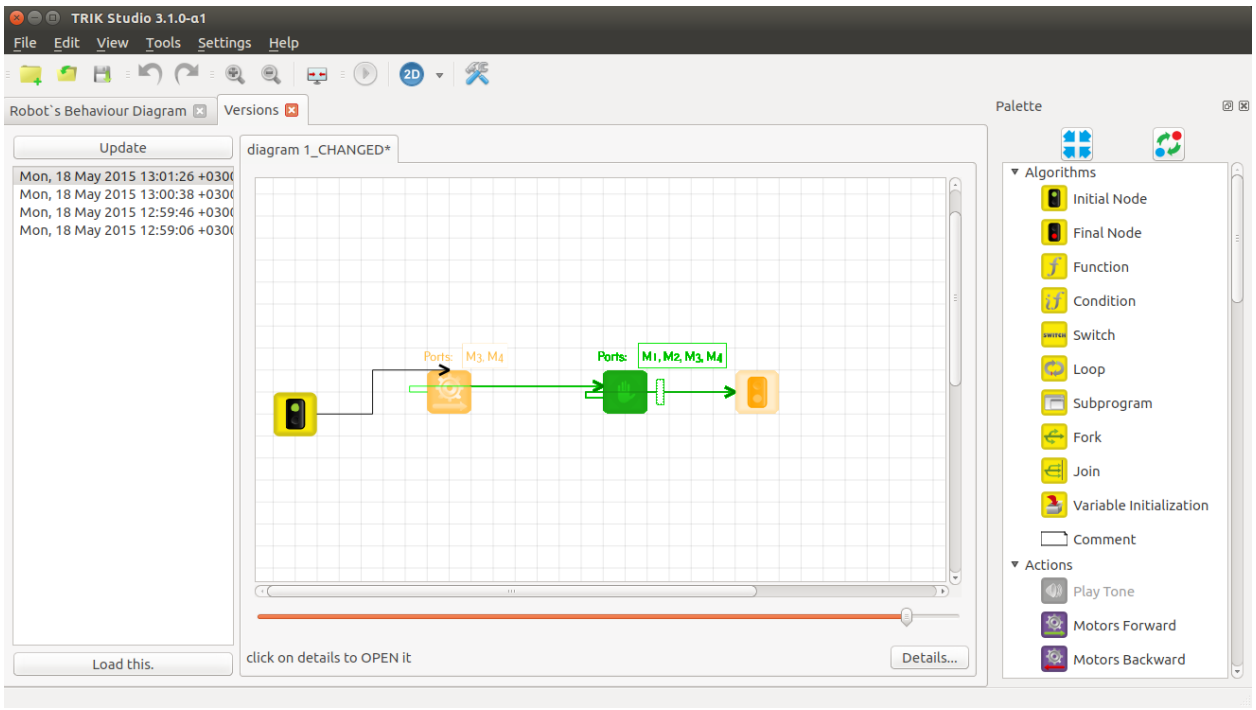

Рисунок 9. Окно просмотра изменений в автоматизированном режиме работы.

Режим расширенного интерфейса

В расширенном интерфейсе пользователю предоставлена возможность

работать с удаленным и локальным репозиториями: сохранение и управление версиями, синхронизация с внешним репозиторием (тестовый пример удаленного репозитория[3]), работа с ветками разработки (рис. 10).

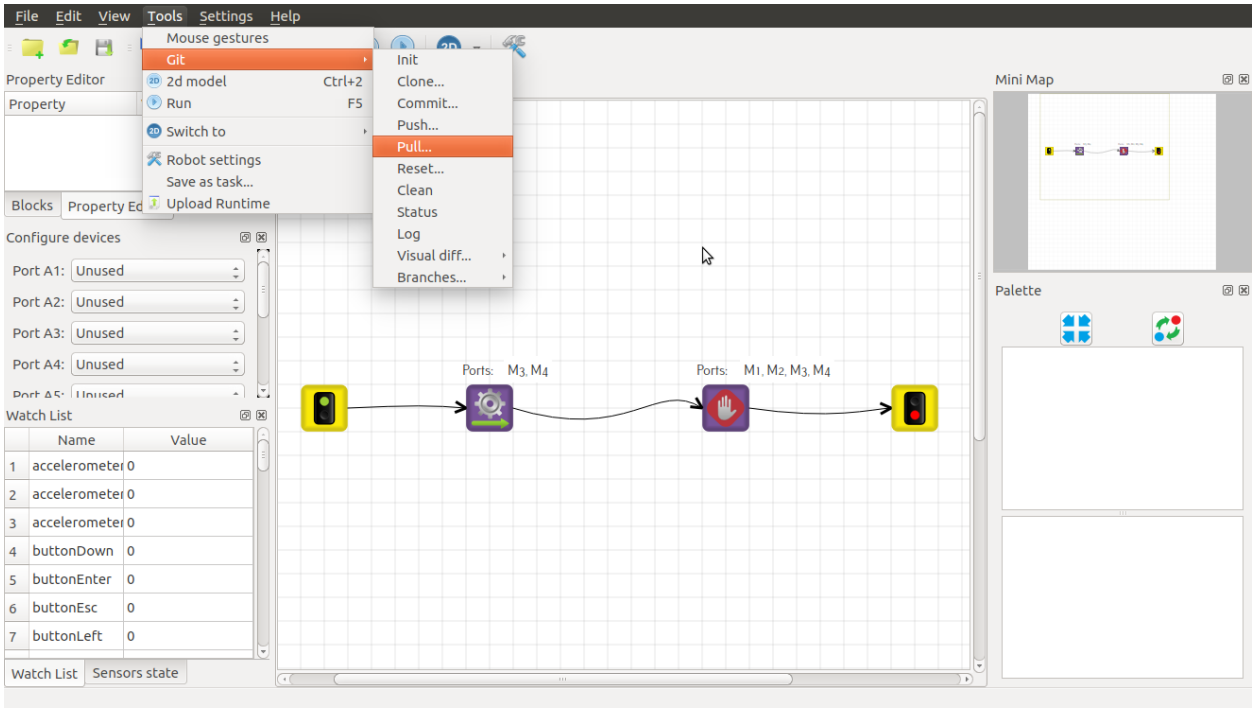

Рисунок 10. Меню расширенного интерфейса.

Заключение

Итоги работы:

Была добавлена поддержка нового подключаемого модуля для работы с внешним клиентом контроля версий Git. Реализована поддержка коллективной работы над проектом, а именно основные возможности распределенной системы контроля версий, в том числе возможность работы с ветвями разработки. Реализованы два режима работы. Упрощенный, где пользователь имеет возможность посмотреть хронологию изменений, узнать об отличиях и иметь возможность откатиться к выбранной версии. И расширенный, где пользователю предоставлен интерфейс, обеспечивающий доступ к основным функциям системы контроля версий. Расширен интерфейс работы с внешним клиентом системы контроля версий. Расширен интерфейс визуализации изменений. Проведены оптимизации критических по времени участков кода, ускорившие работу версионирования в несколько раз. Проанализировано влияние версионирования на процесс сохранения проекта. Проведена апробация новой функциональности на диаграммах подключаемого модуля TRIK Studio.

Пути развития:

В дальнейшем можно решить следующие задачи. Реализовать стратегии фонового сохранения версий. Добавить возможность выделять наборы значимых изменений, пример — выбор подробного/сжатого показа изменений. Так же можно реализовать алгоритм поиска максимального изоморфного подграфа двух графов, желательно в онлайн-режиме, для поиска семантически эквивалентных элементов диаграмм. Так же можно изменить концепцию сохранения модели и работу с ней: например, ассинхронно поддерживать рабочую директорию соответствующую модели, таким образом уйдя от необходимости распаковки и запаковки проекта для систем контроля версий.

Список литературы и источников

[1] Мордвинов Д.А. Создание средств визуального сравнения моделей в QReal, 2011.

[2] Удаленный репозиторий / https://github.com/ZiminGrigory/test2

[3] Chacon S. Pro git. — Apress, 2009.

[4] ClioSoft's SOS Design Data Collaboration Platform/ http://www.cliosoft.com/products.php#hardware_configuration_platforms

[5] ClioSoft's Visual Design Diff / http://www.cliosoft.com/products.php#visual_design_diff

[6] Google document / https://www.google.ru/intl/en/docs/about/

[7] Lee J. et al. An in-depth comparison of subgraph isomorphism algorithms in graph databases //Proceedings of the VLDB Endowment. – VLDB Endowment, 2012. – Т. 6. – №. 2. – С. 133-144.

[8] semiwiki.com (скриншот инструмента) / https://www.semiwiki.com/forum/content/1444-managing-differences-schematic-based-ic-design.html

[9] Visual Paradigm Diff / http://www.visual-paradigm.com/support/documents/vpumluserguide/26/39/6689_whatisvisual.html