# Оглавление

[Оглавление 3](#_Toc295182813)

[Введение 5](#_Toc295182814)

[Глава 1. Языково-ориентированное программирование и задачи сравнения и слияния 8](#_Toc295182815)

[1.1. Языково-ориентированное программирование 8](#_Toc295182816)

[1.2. Описание структуры модели 8](#_Toc295182817)

[1.3. Обзор существующих алгоритмов сравнения и слияния 10](#_Toc295182818)

[Глава 2. Алгоритмы сравнения и слияния моделей 13](#_Toc295182819)

[2.1. Атомарные изменения 13](#_Toc295182820)

[2.2. Алгоритм построения разницы 14](#_Toc295182821)

[2.3. Автоматизированное слияние 15](#_Toc295182822)

[2.3.1. Конфликты изменений 16](#_Toc295182823)

[2.3.2. Проблема дублирующихся идентификаторов 17](#_Toc295182824)

[2.3.3. Конфликты для непозиционных ролей 20](#_Toc295182825)

[2.4. Инкрементальное построение разницы 20](#_Toc295182826)

[2.5. Вычислительная сложность алгоритмов 21](#_Toc295182827)

[Глава 3. Внедрение в среду *JetBrains MPS* 22](#_Toc295182828)

[3.1. Структура программных компонент 22](#_Toc295182829)

[3.2. Интерфейс просмотра разницы 22](#_Toc295182830)

[3.3. Интерфейс автоматизированного слияния 24](#_Toc295182831)

[3.4. Отображение текущей разницы 25](#_Toc295182832)

[Заключение 28](#_Toc295182833)

[Источники 29](#_Toc295182834)

# Введение

**Актуальность проблемы.** В настоящее время получает широкое распространение парадигма языково-ориентированного программирования [4, 6]. В системах, разработанных для инструментальной поддержки данной парадигмы, таких как *JetBrains MPS* [9], код программ представляется в виде абстрактных синтаксических деревьев (моделей).

Для совместной работы над проектом разработчики пользуются системами контроля версий (*Subversion*, *Git*, *Mercurial* и др.), которые позволяют хранить и просматривать историю изменения программ, вносить изменения, создавать различные ветви разработки для независимой работы над разными частями проекта, производить их слияние, разрешая конфликты.

Популярные среды разработки, предназначенные для текстовых языков программирования (*IntelliJIDEA*, *Eclipse*, *NetBeans*, *PyCharm*, и т. д.), как правило, имеют достаточно тесную интеграцию с системами контроля версий, позволяя выполнять стандартные процедуры прямо из среды, например, просмотр истории и интерактивное слияние изменений. Также очень полезной на практике является подсветка текущих изменений относительно последней версии файла прямо в текстовом редакторе.

**Цель работы** — создание алгоритмов сравнения, слияния и отображения текущей разницы для моделей и внедрение в среду языково-ориентированного программирования.

**Основные задачи.** Для достижения указанной цели работы решены следующие задачи:

1. Разработка алгоритма построения разницы двух синтаксических деревьев (моделей) с возможностью отката отдельных изменений.
2. Разработка алгоритма автоматизированного слияния изменений моделей с возможностью ручного применения или отмены отдельных изменений.
3. Модификация алгоритма построения разницы с добавлением свойства быстрой реакции на изменение модели и перестроения актуального набора изменений в реальном времени.
4. Внедрение разработанных алгоритмов в среду разработки *JetBrainsMPS* и создание для них удобного графического пользовательского интерфейса.

**Научная новизна.** Научная новизна предлагаемых алгоритмов состоит в том, они, в отличие от известных алгоритмов слияния деревьев, рассчитаны на деревья с обязательными идентификаторами узлов, а инкрементальные алгоритмы сравнения не известны автору.

**Практическое значение** работы заключается в том, что описанные алгоритмы внедрены в среду *JetBrains MPS* и после следующего выпуска, запланированного на лето 2011 года, будут активно использоваться всеми пользователями среды.

**Апробация** данной работы производилась на VIII Всероссийской межвузовской конференции молодых ученых, (12–15 апреля 2011 года, в СПбГУ ИТМО) в виде устного доклада и опубликована в сборнике тезисов этой конференции[9].

**Структура работы.** Работа изложена на 29 страницах, состоит из введения, трех глав и заключения. В главе 1 приведен обзор предметной области: описание парадигмы языково-ориентированного программирования, структура модели в среде среды *JetBrains MPS*, обзор существующих алгоритмов слияния деревьев и постановка задачи. В главе 2 описываются разработанные алгоритмы сравнения, автоматизированного слияния и инкрементального построения разницы моделей. В главе 3 описывается внедрение алгоритмов в среду языково-ориентированного программирования *JetBrains MPS*. В заключении подводятся итоги работы и выявляются потенциальные направления дальнейших исследований.

# Глава 1. Языково-ориентированное программирование и задачи сравнения и слияния

[TODOсодержание главы]

## 1.1. Языково-ориентированное программирование

Языково-ориентированное программирование — это концепция

## 1.2. Описание структуры модели

*Моделью* будем называть древовидную структуру программы, состояющую из узлов, свойств и ссылок на другие узлы. Далее будет использоваться нотация в терминах *JetBrains MPS*. Каждая модель имеет уникальный идентификатор и метаданные (такие, как список используемых моделей). Но основной частью модели является неупорядоченный список корневых узлов, то есть узлов, не имеющих родительских. Строго говоря, модель — это не дерево, а граф, представляющий из себя лес с дополнительными дугами особого рода — ссылками.

*Узел* модели — это структурная единица модели, содержащая обязательные элементы:

* концепт (тип узла), определяющий его структуру и семантику, ;
* идентификатор, уникальный в пределах модели ;

— и необязательные:

* несколько именованных ролей, в каждой из которых — упорядоченный список дочерних узлов одного назначения ;
* несколько именованных свойств (пар символьных строк имя — значение );
* несколько ссылок на другие узлы (пар роль — указатель на узел ).

Указатель на узел состоит из трех элементов:

* идентификатор модели, содержащей узел;
* идентификатор узла;
* информация для восстановления ссылки (строка, содержащая имя целевого узла).

Для лучшего понимания смысла элементов структуры, рассмотрим пример части модели. На Рис. 1приведена строка программы на языке *baseLanguage*, который является аналогом языка *Java* в среде *JetBrains MPS*. На Рис. 2 для этой строки показана структура части модели. На ней блокам в простых рамках соответствуют узлы (внутри приведены идентификаторы и концепты), в серых пунктирных — свойства, в точечных — ссылки на узлы, а в жирных пунктирных — роли дочерних узлов. [TODO вставить XML-представление]



Рис. 1. Пример части модели в редакторе.

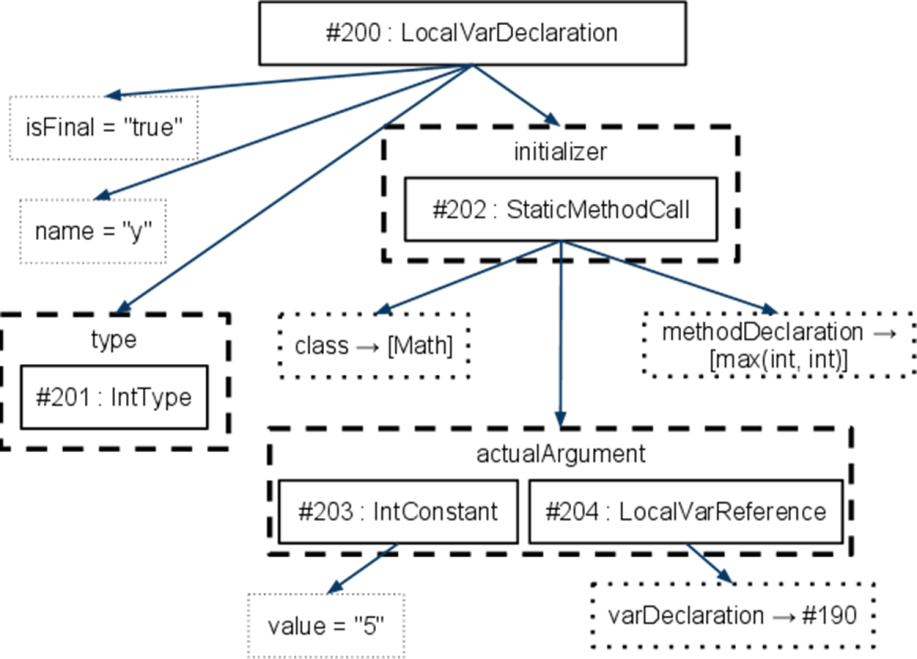


Рис. 2. Пример структуры модели

*Разница моделей (дельта)* — набор структурных изменений между двумя моделями, который можно отображать визуально в графическом пользовательском интерфейсе и полностью или частично применять к модели.

*Слияние изменений модели* — объединение и применение дельт между исходной моделью и двумя ее модификациями, с ручным разрешением конфликтов.

## 1.3. Обзор существующих алгоритмов сравнения и слияния

При работе с моделями необходимы специальные алгоритмы для просмотра изменений, слияния и отображения текущей разницы. Существуют алгоритмы для текстового кода программ, например, например, в утилитах *diff*, *diff3* и подобных [7]. Но они плохо подходят для моделей, даже если их применять к сериализованным версиям, то есть преобразованным в текстовый вид в виде *XML*-файлов. Алгоритмы не применимы по следующим причинам.

1. Они не учитывают древовидной структуры файла, и значит, в результате слияния может получиться некорректный *XML*-код).
2. Они заставляют пользователя вникать в *XML*-представление файла, которое для него непривычно и сложно для восприятия.
3. Отображение отличий от базовой версии вообще бесполезно, потому что редактирование моделей происходит на более высоком уровне, с помощью тексто-подобных (проекционных) редакторов.

Существуют также различные алгоритмы для построения разницы и слияния графов и деревьев, имеющие ряд недостатков. Алгоритмы построения разницы сами по себе, как правило, не рассматриваются, обычно они являются составной частью алгоритмов слияния изменений.

Большинство известных алгоритмов слияния деревьев [2, 5, 8] ориентированы на работу с деревьями без идентификаторов у узлов, уникальность которых следует сохранять, и ссылок, корректность которых следует поддерживать. В процессе слиянии могут возникать ситуации, когда в модели возникает несколько узлов с одинаковыми идентификаторами — это проблема, которую следует уметь обходить.

Некоторые алгоритмы слияния, такие как описанный в [8], работают с изменениями типа перенос поддерева. В случае, когда конфликтов нет, обнаружение и корректное слияние таких изменений очень удобно. Но при возникновении конфликтов становится сложно визуализировать изменения в графическом интерфейсе так, чтобы пользователю они были понятны. В том числе это сложно, когда просмотр разницы (с частичными откатами) или слияние изменений происходит в окне с двумя или тремя проекционными редакторами, где соответствующие изменения выделены цветом и соединительными элементами между различающимися местами в редакторах, соответствующих разным версиям. Концепция, в которой пользователь работает с изменениями типа вставки, удаления и замены, намного проще для восприятия и манипуляции. Поэтому в данной работе от обработки изменений типа перемещение поддерева было решено отказаться.

Автором была предпринята попытка поиска работ, касающихся инкрементальных алгоритмов построения разницы деревьев, но в результате такие работы не были найдены.

Итак, существующие алгоритмы сравнения и слияния не подходят в явном виде для решения поставленных задач, потому как не удовлетворяют вышеописанным требованиям.

# Глава 2. Алгоритмы сравнения и слияния моделей

[TODO содержание главы]

## 2.1. Атомарные изменения

Разница моделей — это неупорядоченный набор *атомарных изменений*. В наших алгоритмах атомарные изменения обладают следующими свойствами:

* возможность применять к исходной модели или откатывать в результирующей модели в любом порядке, соответствующим образом модифицируя модель;
* возможность отображения отдельно в проекционном редакторе (то есть, отсутствие пересечений затрагиваемых подграфов);
* понятность пользователю.

*Группой узлов* называется множество последовательных узлов одной роли одного родительского узла с индексами от включительно до исключительно. Очевидно, размер группы равен .

Приведем описание возможных типов атомарных изменений. В скобках указаны их параметры. Параметры содержат минимум информации об изменении, они позволяют только сравнивать изменения друг с другом. Для применения изменения к модели необходимо также иметь доступ к исходной и результирующей моделям, для которых была посчитана разница.

1. *Добавлен корневой узел* (идентификатор).
2. *Удален корневой узел* (идентификатор).
3. *Изменено свойство* (идентификатор, имя).
4. *Изменена ссылка* (идентификатор, роль).
5. *Группа узлов заменена на другую* (идентификатор, роль, индексы исходной группы и результирующей ). Частные случаи:
   1. группа узлов удалена ();
   2. группа узлов вставлена ().

## 2.2. Алгоритм построения разницы

Построение разницы происходит при рекурсивном обходе узлов двух моделей: старой и новой. На каждом этапе порождаются новые атомарные изменения, которые собираются в список.

1. Сравнение множеств идентификаторов корневых узлов исходной и результирующей модели с выявлением добавленных и удаленных. Порождаются соответствующие изменения типов «добавлен/удален корневой узел».
2. Для всех идентификаторов корневых узлов, которые присутствуют в обеих версиях модели, запускается рекурсивный просмотр и сравнение соответствующих узлов старой и новой моделей.
3. Сравниваются множества свойств, порождаются изменения типа «изменено свойство». Удаление свойства — частный случай изменения.
4. Аналогично сравниваются ссылки, порождаются изменения типа «изменена ссылка», в том числе для удалений.
5. Для каждой роли дочерних узлов рассматриваются два списка узлов: из старой и новой моделей. Для этих списков находится наибольшая общая подпоследовательность [3] (сравнение происходит только по идентификаторам). Те узлы, которые не вошли в нее, считаются добавленными, удаленными или замененными, порождаются соответствующие изменения типа «группа узлов заменена». Это проиллюстрировано на Рис. 3: здесь сверху показаны узлы из старой модели, снизу — из новой; порождены изменения «удален узел », «узел заменен на последовательность » и «вставлен узел в позицию после ». Узлы, вошедшие в общую подпоследовательность, попарно сравниваются рекурсивно, начиная с пункта 2.

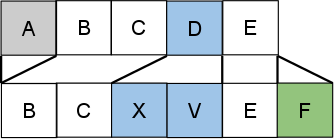


Рис. 3. Наибольшая общая подпоследовательность узлов роли

## 2.3. Автоматизированное слияние

Задача автоматизированного слияния состоит в следующем. Даны три версии модели: исходная и две ее модификации и . Требуется создать модель , интегрирующую эти модификации. [TODO принципиальная схема]

B

Общий способ решить задачу прямолинеен: необходимо построить две разницы: между и , между и , — а затем последовательно применить к модели B все атомарные изменения из обеих разниц. Но на самом деле все оказывается не так-то просто, существует ряд трудностей.

1. Изменения могут *конфликтовать* друг с другом. Это значит, что два изменения из противоположных разниц нельзя применить одновременно, потому как это действие некорректно. В этом случае требуется вмешательство пользователя для ручного разрешения конфликта. Например, если в значение некоторого свойства было изменено одним образом, а в — другим, то только пользователь может решить, какое из этих двух изменений применить, а какое — исключить. Для этого необходимо предусмотреть соответствующие элементы графического интерфейса.
2. Некоторые изменения могут быть *симметричными*. Это значит, что применение каждого из двух изменения из противоположных разниц в отдельности приведет к одному и тому же эффекту. Значит, применять оба изменения не требуется, достаточно применить одно из них. Например, если в *M*1был удален некоторый узел и в *M*2 был удален тот же узел (с тем же идентификатором), то это изменение следует применить только один раз.
3. При применении изменений, связанных с добавлением узлов, могут возникнуть конфликты идентификаторов, то есть ситуация, когда в модели оказывается несколько узлов с одинаковыми идентификаторами. Это делает модель некорректной, поэтому необходимо придумать способ обхода этой проблемы.

### 2.3.1. Конфликты изменений

Группы дочерних узлов и называются *смежными*, если они находятся в одной роли одного узла, и пересечение интервалов .

Узлы называются *эквивалентыми*, если равны их идентификаторы, концепты, списки свойств и ссылок, а также в каждой роли списки дочерних узлов эквивалентны, то есть все дочерние узлы эквивалентны соответствующим, первый эквивалентен первому, второй — второму, и т. д.

Рассмотрим типы конфликтов и симметричных изменений, возникающие при слиянии моделей.

1. В обеих модификациях модели *добавлен корневой узел с одинаковым идентификатором*. Случай, когда обе версии добавленного узла эквивалентны, — не конфликт, а пара симметричных изменений.
2. *Свойство узла изменено в обеих версиях*. В случае, если модифицированные значения свойства совпадают, это не конфликт, а симметрия.
3. *Ссылка изменена (в том числе удалена или добавлена) в обеих версиях*. Если новые указатели ссылок совпадают, это не конфликт, а симметрия.
4. *Смежные группы дочерних узлов заменены в обеих версиях*. Если исходные группы совпадают (), а результирующие имеют одинаковую длину и результирующие узлы попарно эквивалентны, то эти изменения не конфликтующие, а симметричные.
5. *В одной модификации группа заменена, а в другой — есть изменения в поддеревьях узлов из этой группы*.

### 2.3.2. Проблема дублирующихся идентификаторов

Как уже было сказано выше, при последовательном применении отдельных изменений может возникнуть ситуация, когда в модели возникает несколько узлов с одинаковыми идентификаторами, а это некорректная ситуация. Очевидна идея при применении изменений, касающихся добавления узлов, присваивать им новые идентификаторы. Но тогда все ссылки, указывавшие на эти затронутые узлы, станут неразрешенными, т. е. указывающими на несуществующие узлы.

То есть по возможности необходимо сохранять старые идентификаторы. А если конфликт неминуем — то новым узлам следует присваивать новые идентификаторы.

Рассмотрим не самую редкую ситуацию, возникающую, например, когда в измененной модели был перенесен узел из одного места в другое. В этом случае есть два изменения: узел удален и узел добавлен. Назовем узел буквой .

1. Применяется изменение «узел добавлен». Возник конфликт идентификаторов. Обозначим новую копию узла как .
2. Применяется изменение «узел удален». В результирующей модели теперь нет узла , но есть его копия , с другим идентификатором. Все существовавшие ссылки на узел стали неразрешимыми «сломались».

Для противодействия данной проблеме предлагается сохранять все замены идентификаторов, к которым приходилось прибегать во избежание конфликта (). А когда идентификаторы освобождаются (например, ), то эти замены «возвращаются». В примере выше после второго шага следует узлу вернуть идентификатор .

Рассмотрим работу данного метода в более сложном случае. Пусть есть роль некоторого узла, в котором было четыре дочерних узла: , , и . В первой модификации модели узел был перемещен в последнюю позицию (после ), а во второй — между и (Рис. 4, а). Как видно, обе разницы состоят из пары изменений: «узел удален» и «узел вставлен». Изменения «узел удален» являются симметричными. Изменения «узел вставлен» являются не конфликтующими. Первым применяется изменение «узел вставлен» из второй модификации. Очевидно, оно приводит к конфликту идентификаторов, и вставленная копия узла получает идентификатор (Рис. 4, б). Затем применяется изменение «узел вставлен» из первой модификации. Снова возникает конфликт идентификаторов: новая копия получает идентификатор (Рис. 4, в). Наконец, применяется одно из симметричных изменений «узел удален». После этого идентификатор освобождается, и поэтому одной из копий: или — следует вернуть исходный идентификатор , чтобы сохранить ссылки, указывающие на . Какой именно — разницы нет, потому для определенности можно решить, что возвращать идентификатор следует первой копии, в данном случае —  (Рис. 4, г).

B

C

D

A

A

B

A’

C

D

B

C

A

D

B

C

D

A

A

B

C

D

B

C

A

D

B

C

D

A

A

B

A’

C

A’’

D

B

C

A

D

B

C

D

A

B

A

C

A’’

D

B

C

A

D

а)

в)

б)

г)

Рис. 4. Применение изменений, приводящих к дублирующимся идентификаторам

## 2.3.3. Конфликты для непозиционных ролей

[TODO рассказать про непозиционные роли]

## Инкрементальное построение разницы

Для отображения разницы относительно базовой версией модели (из репозитория) в процессе редактирования модели необходим инкрементальный алгоритм, который реагирует на события изменений модели и частично перестраивает разницу, поддерживая ее актуальность. При инициализации разница строится при помощи простого алгоритма, описанного выше.

Приведем возможные события изменения модели и реакцию на них.

1. *Корень добавлен*. Проверить, есть ли изменение «корень удален» для этого идентификатора. Если да, то его необходимо удалить, и построить разницу между этим узлом в базовой и текущей версиях, добавив все выявленные изменения. Если нет, то добавить изменение типа «корень добавлен».
2. *Корень удален*. Проверить, есть ли изменение «корень добавлен» для этого идентификатора. Если да, то его необходимо удалить. Если нет — удалить все изменения внутри этого узла и добавить изменение типа «корень удален».
3. *Свойство изменено*. Проверить, находится ли оно в измененном поддереве (то есть, в поддереве некоторого узла, входящего в группу изменения типа «группа заменена»). Если да, то событие следует проигнорировать. Если нет, то: если новое значение не совпадает со старым, то добавить изменение типа «свойство изменено», иначе — проигнорировать.
4. *Ссылка изменена или удалена*. Проверить, находится ли ссылка в измененном поддереве. Если да — проигнорировать. Если нет: если новый указатель не совпадает со старым, то добавить изменение типа «ссылка изменена», иначе — проигнорировать.
5. *Дочерний узел добавлен или удален*. Проверить, находится ли родитель этого узла в измененном поддереве. Если да, то проигнорировать. Если нет, то удалить все изменения для данной роли дочерних узлов (включая изменения ссылок и свойств), затем перестроить разницу для этой роли и родительского узла, построив наибольшую общую подпоследовательность, и т. д.

## 2.5. Вычислительная сложность алгоритмов

[TODO]

# Глава 3. Внедрение в среду *JetBrains MPS*

[TODO содержание главы]

## 3.1. Структура программных компонент

При внедрении в среду разработки *JetBrains MPS* вышеописанных алгоритмов были созданы модули, написанные на языке *baseLanguage* (аналоге *Java*) с использованием его расширений для работы с коллекциями, кортежами и замыканиями. Атомарные изменения представляются абстрактным классом *ModelChange* и его наследниками. Основными методами этого класса являются *apply*, применяющий изменение к модели, переданной в качестве аргумента, и *getOppositeChange*, возвращающий обратное изменение, применение которого аналогично откату. Класс *ChangeSetBuilder* для двух моделей строит объект *ChangeSet*, содержащий набор изменений. Класс *MergeContext* содержит состояние текущей сессии слияния моделей: исходные модели, два набора изменений (объектов *ChangeSet*), множества пар конфликтующих и симметричных изменений, множества примененных и отмененных изменений, текущую модель, и информацию об идентификаторах, замененных во избежание дублирования. Объекты класса *ModelChangesManager* хранят текущую разницу между моделью и ее базовой версией, реагируют на события изменения модели и уведомляют компоненты графического интерфейса о создании и удалении изменений.

[TODO более подробно, рассказать про каждый класс]

## 3.2. Интерфейс просмотра разницы

Интерфейс для просмотра разницы двух версий модели состоит из двух диалоговых окон, представленных на Рис. 5. В первом окне отображается дерево модели, в которой корневые узлы отмечены разными цветами: зеленый означает, что узел добавлен, серый — узел удален, синий — что узел имеет иные изменения.

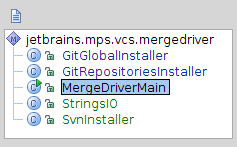


Рис. 5. Интерфейс сравнения моделей. Корневые узлы

При двойном клике на корневом узле открывается окно с разницей корневых узлов, представленное на Рис. 6. В этом окне есть два редактора для каждой из версий. Изменения помечены в редакторах теми же цветами: зеленый означает добавленные узлы, серый — удаленные, а синий — прочие изменения. Для удобства пользователя есть: панель статуса, отображающая значения цветов и количество изменений и панель инструментов, позволяющая быстро переключаться на соседние узлы или переходить к предыдущему или следующему изменениям в данном корневом узле. При отображении разницы между текущей моделью и предыдущей ее версией можно редактировать текущую модель непосредственно при просмотре разницы, а также откатывать изменения.

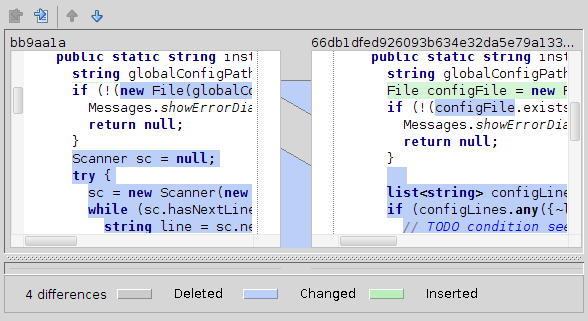


Рис. 6. Интерфейс сравнения корневых узлов

## 3.3. Интерфейс автоматизированного слияния

Интерфейс для автоматизированного слияния тоже состоит из двух диалоговых окон, аналогичных окнам просмотра разницы. В главном окне, представленном на Рис. 7, так же как в интерфейсе просмотра разницы, отображено дерево модели с корневыми узлами, но здесь они, помимо зеленого, серого и синего, могут быть отмечены красным цветом, означающим, что узел содержит неразрешенные конфликты изменений. В скобках указано, где изменен узел: в локальной версии (*local*), в репозитории (*remote*), в обеих с конфликтами (*with conflicts*) или без (*both modified*).На панели инструментов есть кнопки, позволяющие: начать процесс слияния сначала; применить автоматически все изменения в корневых узлах без конфликтов; для выделенных корневых узлов применить только изменения из одной из версий.

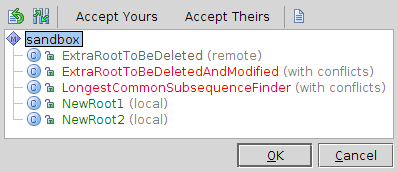


Рис. 7. Интерфейс слияния моделей. Корневые узлы

При двойном клике на корневом узле в дереве открывается диалог с тремя редакторами, представленный на Рис. 8, позволяющий откатить или применить отдельные изменения при помощи активных элементов на полях редакторов. Изменения подсвечиваются в редакторе тремя цветами, красный означает конфликтующие изменения. На панели инструментов, помимо кнопок для перехода к соседним корневым узлам и к соседним изменениям, есть кнопка автоматического применения всех не конфликующих изменений.

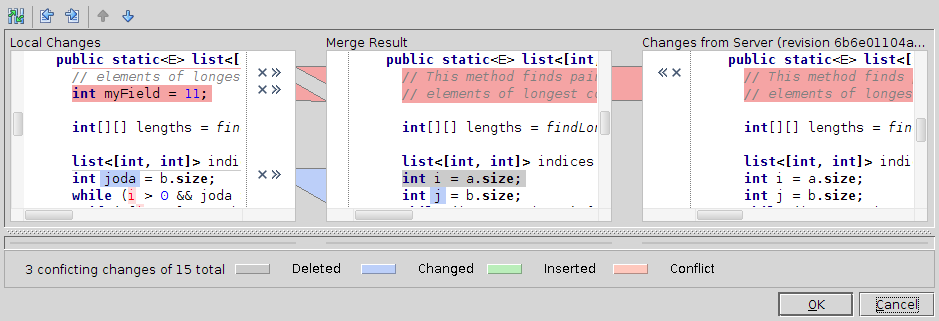


Рис. 8. Интерфейс слияния корневых узлов

## 3.4. Отображение текущей разницы

Отображение текущей разницы происходит не только в основном окне редактора, но также и в структурном дереве проекта.

В редакторе изменения относительно базовой версии модели отмечаются в левом поле редактора полосками зеленого, синего и серого цветов, как показано на Рис. 9. При клике на полоску открывается панель инструментов, позволяющая откатить соответствующие ей изменения, а также перейти к следующей или предыдущей группе изменений.

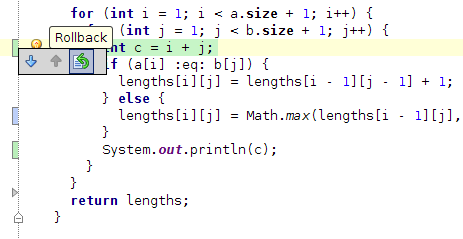


Рис. 9. Отображение текущей разницы в редакторе

В структурном дереве проекта отображаются модули, модели, их полная структура вплоть до ссылок и свойств. Текущая разница в этом дереве отображается путем подсветки соответствующих элементов зеленым и синим цветами (серым подсвечивать нечего, так как удаленные элементы отсутствуют в текущей версии модели). Внешний вид дерева проекта с подсветкой текущей разницы приведен на Рис. 10.

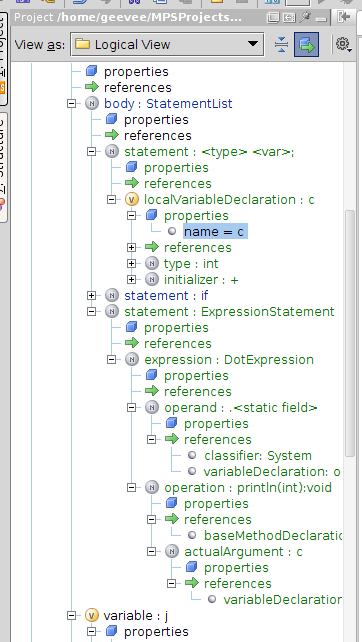


Рис. 10. Отображение текущей разницы в дереве проекта

# Заключение

В работе решены следующие задачи:

1. Разработан алгоритм построения разницы двух синтаксических деревьев (моделей) с возможностью отката отдельных изменений.
2. Разработан алгоритм автоматизированного слияния изменений моделей с возможностью ручного применения или отмены отдельных изменений.
3. Модифицирован алгоритм построения разницы, добавлено свойство инкрементальности, то есть быстрой реакции на изменение модели и перестроения актуального набора изменений в реальном времени.
4. Разработанные алгоритмы внедрены в среду разработки *JetBrains MPS*, для них создан удобный графический пользовательский интерфейс.

Можно выделить следующие направления дальнейших исследований:

* поддержка изменений типа перенос в некоторых случаях без потери прозрачности для пользователя;
* оптимизация инкрементального алгоритма сравнения за счет отказа от полного перестроения разницы для измененного поддерева при событии типа «добавлен/удален дочерний узел»;
* особая обработка конфликтов в пределах одной роли дочерних узлов в случаях, когда соблюдение порядка в роли необязательно.

# Источники

1. *Геращенко, Е. В.* Алгоритмы сравнения и слияния абстрактных синтаксических деревьев в среде языково-ориентированного программирования // Сборник тезисов докладов конференции молодых ученых, Выпуск 1. Труды молодых ученых. – СПб: СПбГУ ИТМО, 2011. С. 228–229.
2. *Asklund U.* Identifying conflicts during structural merge // NordicWorkshoponProgramming Environment Research '94. — Lund, Sweden, 1994. — pp. 231–242.
3. *Bergroth L.* A survey of longest common subsequence algorithms // String Processing and Information Retrieval. - A Curuna, Spain, 2000. — pp. 39–48. — ISBN 0-7695-0746-8.
4. *Dmitriev S.* Language Oriented Programming: The Next Programming Paradigm. — JetBrains, November 2004.  
   <http://www.onboard.jetbrains.com/is1/articles/04/10/lop/>.
5. *La Fontaine R.* Merging XML files: a new approach providing intelligent merge of XML data sets // XML Europe. — Barcelona, Spain, 2002.
6. *Fowler M.* Language Workbenches: The Killer-App for Domain Specific Languages?. — June 12, 2005.  
   <http://www.martinfowler.com/articles/languageWorkbench.html>.
7. *Johnson M. K.* Diff, patch and friends // Linux Journal. — August 1996. — 1996(28es).
8. *Lindholm T. A* Three-way Merge for XML Documents // ACM symposium on Document engineering. - New York, NY, USA : ACM, 2004. — Vol. DocEng '04. — pp. 1–10.
9. *JetBrains.* Сайтсреды разработки JetBrains MPS. <http://www.jetbrains.com/mps/>