Санкт-Петербургский государственный университет

Математико-механический факультет

Кафедра системного программирования

# “Поддержка эволюции визуальных языков в DSM-платформе QReal”

Курсовая работа студентки 344 группы

Агаповой Татьяны Юрьевны

Научный руководитель: ст. преп. Брыксин Т.А.

Санкт-Петербург

2014

# Оглавление

Введение....3
Постановка задачи....3
Обзор существующих подходов....5
Ручная спецификация....5
Операторный подход к миграции....5
Сопоставление моделей....6
Ограничения автоматических подходов....7
Требования к миграции в QReal....9
Описание выбранного подхода....11
Автоматическая миграция....12
Протоколирование....12
Модель разницы....12
Трансформации....13
Общая схема миграции....13
Апробация....13
Результат....16
Дальнейшие перспективы....16
Литература....18

# Введение

При модельно-ориентированном подходе к разработке программного обеспечения [1] программа представляется в виде набора моделей, описанных с помощью некоторых, чаще всего визуальных, языков моделирования. При этом использование языков, предназначенных для узкой предметной области, упрощает процесс моделирования и восприятие моделей человеком [4]. Таким образом, желательно наличие инструмента, который позволяет быстрое создание визуальных предметно-ориентированных языков и моделей на этих языках. Такие средства называются предметно-ориентированными платформами (DSM-платформами). Создаваемый в них язык, как правило, также описывается в виде модели на специализированном языке метамоделирования – так называемой метамодели этого языка.

Подобно прочим программным продуктам, языки моделирования развиваются со временем. Причиной необходимости эволюции языка может стать ошибка в моделировании предметной области или уточнение её понимания. В результате изменений в языке модели, созданные с помощью него, могут перестать ему соответствовать. В качестве примера такого несоответствия может послужить удаление из языка некоторого типа элементов. Тогда все экземпляры этого типа должны быть удалены из модели для восстановления её корректности или заменены экземплярами какого-либо другого типа, схожего по семантике.

Процесс переноса модели со старой версии языка на новую называется миграцией. Примитивный способ миграции – восстановление модели вручную пользователем языка. Данный подход крайне неэффективен ввиду своей трудоёмкости и в отношении крупных моделей может стать труднее, нежели разработка модели с нуля. Кроме того, в этом случае велика вероятность ошибки, что может нарушить семантическую целостность модели и всех артефактов, которые от неё зависят, например, генерируемого по данной модели исходного кода программы.

Ввиду рассмотренных трудностей необходим способ спецификации миграционной стратегии – описания изменений, которые необходимо совершить над моделью для восстановления её корректности. Эти изменения впоследствии могут применяться к модели автоматически, снижая риск возникновения ошибки и не обременяя пользователя необходимостью вручную исправлять несоответствия. В настоящее время существуют различные подходы к миграции моделей - от полностью ручных до автоматизированных.

# Постановка задачи

Целью данной курсовой работы является поддержка миграции моделей в DSM-платформе QReal, разрабатываемой на кафедре системного программирования Санкт-Петербургского государственного университета [11]. Эта система обладает некоторыми особенностями, которые должны быть учтены при разработке механизма миграции.

Задачами данной курсовой работы являются:

- исследование существующих подходов к миграции;
- анализ требований, которые накладывает на процесс миграции QReal;
- разработка и реализация удовлетворяющего данным требованиям подхода к миграции;
- апробация реализованного механизма на каком-либо визуальном языке.

## Обзор подходов

Существующие подходы к миграции моделей можно разделить на три категории: ручная спецификация миграции, операторный подход и сопоставление моделей [7].

### Ручная спецификация

В случае ручной спецификации разработчик метамодели задаёт трансформации моделей вручную. Трансформации могут быть описаны на языке программирования общего назначения либо с помощью текстовых или визуальных языков описания трансформаций. На Рис. 1 приведён пример преобразования типа Person в тип Contact и описание этого преобразования на ATLAS Transformation Language [3].

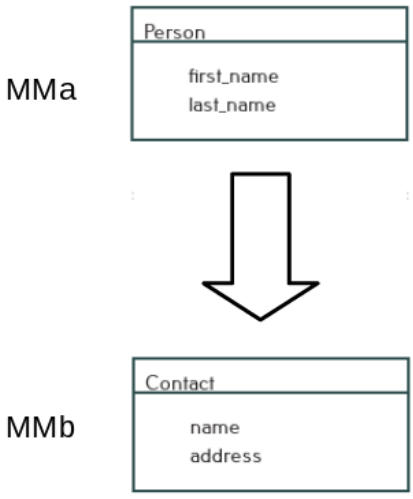



Рис. 1а: Миграция преобразования типа

```
module Person2Contact;
create OUT : MMb from IN : Mma;

rule Start {
    from
        p : MMa!Person
    to
        c : Mmb!Contact (
            name <- p.first_name +
                p.last_name
        )
}
```

Рис. 1б: Описание миграции на ATL

Преимущество ручной спецификации - в том, что разработчик языка имеет наиболее полный контроль над процессом миграции, а значит, семантика модели максимально сохраняется. Недостаток данного подхода - в трудоёмкости и необходимости учить язык спецификации миграционной стратегии разработчику метамодели.

### Операторный подход к миграции

При операторном подходе эволюция метамодели выражается в терминах композиции применённых к ней операторов, которые можно представить в виде графовых шаблонов, отражающих соответствие между старыми и новыми сущностями языка. Пример описания оператора переименования элемента языка с помощью языка QVT-Relations [5] изображён на Рис. 2.

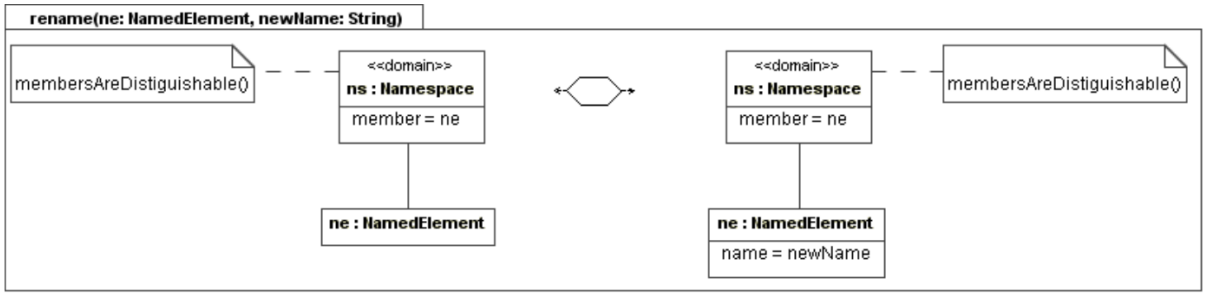



Рис.2: Описание переименования элемента на QVT-Relations

Каждому оператору соответствует трансформация, применяемая к модели на данном языке. Последовательность трансформаций, соответствующих операторам, с помощью которых была произведена эволюция метамодели, представляет собой миграционную стратегию.

При использовании данного подхода разработчик DSM-платформы должен предоставить библиотеку операторов. В зависимости от реализации операторы могут выбираться разработчиком языка при редактировании метамодели или выводиться из истории её изменений автоматически. Точность миграции зависит от полноты предоставляемой библиотеки операторов. Она может быть слишком бедной, в этом случае в ней просто отсутствуют необходимые операторы. Излишняя же избыточность библиотеки приводит к неоднозначности выбора того или иного оператора в каждом конкретном случае, что затрудняет и спецификацию миграции разработчиком метамодели, и автоматический вывод миграционной стратегии.

## Сопоставление моделей

Сопоставление моделей - автоматический подход, в идеале не требующий никакого вмешательства разработчика либо пользователя метамодели в процесс миграции. Подходы на основе сопоставления моделей используют один из двух механизмов: историю изменений метамодели либо модель разницы старой и новой метамоделей.

В случае использования истории изменений все действия, совершённые над метамоделью в процессе её редактирования, записываются и сохраняются вместе с метамоделью. Впоследствии, при миграции эти записи анализируются для вывода миграционной стратегии.

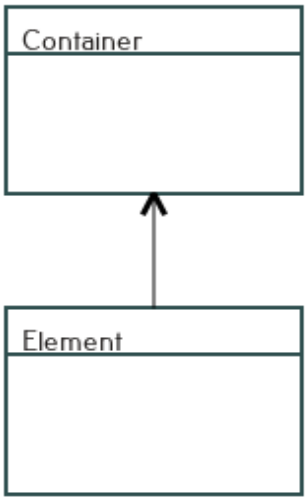
Модель разницы является представлением различий старой и новой метамоделей и отвечает на следующие вопросы: какие типы добавились, исчезли или заменились на другие, как изменились иерархии наследования, отношения вложенности элементов и т.д. Эта информация используется для вывода трансформаций мигрируемой модели.

## Ограничения автоматических подходов

К сожалению, во многих случаях миграция не может быть выведена автоматически [2]. Одной из причин этому являются сложные изменения метамодели. Во-первых, одно изменение может затрагивать несколько элементов метамодели (например, при изменениях в иерархии наследования). Во-вторых, изменение может состоять из последовательности действий над метамоделью (к примеру, замена типа производится с помощью удаления старого типа и создания нового). Необходимость различать сложные изменения и последовательности независимых простых изменений требует наличия продвинутых механизмов анализа истории изменений метамодели и модели разницы.

Более сложной проблемой является зависимость миграционной стратегии от восприятия моделей человеком (пользователем или разработчиком языка). Рассмотрим эту проблему на примере.

Допустим, имеется язык, который позволяет создавать контейнеры элементов:



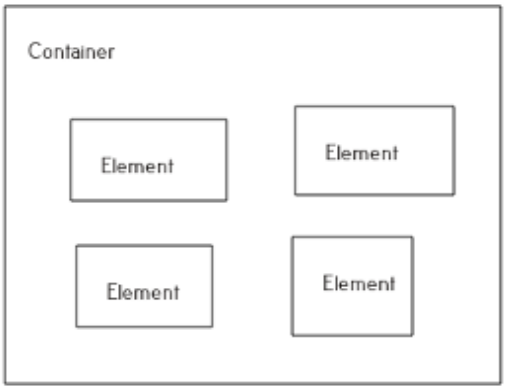
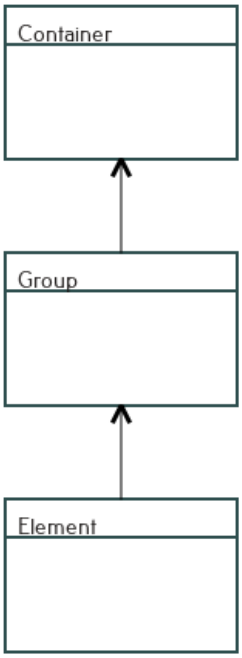




Рис. 3а: Метамодель языка контейнеров Рис. 3б: Пример модели на этом языке

В случае, если разработчик этого языка примет решение добавить ещё один уровень вложенности, требуя группировать элементы внутри контейнеров, необходимо каким-то образом осуществить такую группировку в уже существующих моделях (новая метамодель языка и одна из возможных группировок изображены на Рис. 4).

Если допускать, что группировка может осуществляться по произвольному признаку, то определить автоматически наиболее подходящий в каждом случае способ группировки не представляется возможным. Более того, это не под силам даже разработчику языка, так как в общем случае для этого требуется знание смысла, который вкладывает пользователь в такую группировку.

Таким образом, в данном случае невозможно произвести автоматическую миграцию с сохранением семантики модели. Всё, что можно сделать, - это выбрать некоторый вырожденный вариант группировки (допустим, считать, что все элементы принадлежат одной группе) и предупредить пользователя о возможной потере смысла.



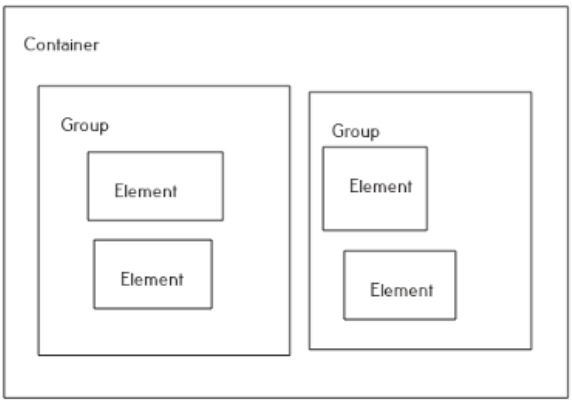




Рис. 4а: Преобразованный язык контейнеров

Рис. 4Б: Возможное преобразование модели

Описанные сложности делают задачу автоматической миграции моделей в общем случае неразрешимой, в связи с чем желательной представляется возможность оптимизировать процесс автоматической миграции с помощью ручной спецификации сложных изменений пользователем или разработчиком языка. Выбор конкретной гибридной реализации миграции моделей зависит от особенностей DSM-платформы, в которой требуется реализовать поддержку этой функциональности, поскольку они могут накладывать на миграцию различные специфические ограничения.

# Требования к миграции в QReal

Метамоделирование в QReal может осуществляться в трёх режимах: генерации, интерпретации и метамоделирования “на лету”. Рассмотрим, какие требования накладывает каждый из них на процесс миграции.

В режиме генерации по описанию метамодели на метаязыке автором языка генерируется подключаемый модуль, содержащий информацию о языке, представляемом метамоделью. В этом случае вполне допустима ручная спецификация миграции, так как она даёт возможность более тонко контролировать процесс миграции. В то же время некоторые простые изменения (например, замена имени типа) могут быть выведены автоматически, упрощая работу разработчику языка.

В режиме интерпретации метамодель загружается в интерпретатор, который поддерживает такой же программный интерфейс, как и генерируемый подключаемый модуль. Интерпретация предполагает ускорение цикла “метамоделирование - загрузка метамодели - моделирование”, что ведёт к необходимости ускорения процесса миграции. Один из способов – использование операторного подхода с предоставляемой библиотекой операторов и автоматическим выводом миграционной стратегии. Другой способ – преобразование существующего в QReal механизма рефакторингов [9], заключающееся в сопоставлении рефакторингу метамодели трансформацию модели. Это позволит объединить редактирование метамодели и спецификацию миграционной стратегии, избавляя разработчика языка производить одни и те же действия дважды и уменьшая вероятность ошибки.

Метамоделирование “на лету” [10] - возможность изменения языка прямо в процессе его использования. Этот режим требует максимально возможной автоматизации процесса миграции, поскольку метамодель и модель должны в каждый момент времени оставаться в консистентном состоянии, а ручная спецификация трансформаций сводит на нет преимущества метамоделирования “на лету”. В то же время, некоторые изменения в языке могут состоять из нескольких шагов пользователя (к примеру, при замена типа включает в себя удаление старого типа и добавление нового), поэтому нужна возможность “на лету” распознавать последовательности действий, логически представляющие собой одно изменение. Кроме этого, метамодель в явном виде недоступна пользователю, поэтому применение рефакторингов не представляется возможным.

Механизм миграции должен быть реализован таким образом, чтобы достичь максимальной точности, и в то же время быть достаточно гибким, чтобы поддержать метамоделирование “на лету”, а в случае генерации избавить разработчика языка от необходимости вручную задавать миграции. Таким образом, можно сформулировать следующие требования к реализуемому подходу:

1. Поддержка всех трёх режимов работы с метамоделью.
2. Возможность выбирать между разными способами миграции (рефакторинги или автоматическая миграция). При этом эти механизмы не должны конфликтовать между собой и допускать одновременное использование.
3. Максимально возможная точность трансформации, сохранение семантики мигрируемой модели.

## Описание выбранного подхода

Рассматриваемый гибридный подход совмещает в себе точность ручной спецификации и прозрачность сопоставления моделей. В его основе лежит идея максимальной автоматизации миграции. Для вывода миграционной стратегии используются история изменений метамодели и модель разницы между старой и новой метамоделями.

В истории изменений хранятся элементарные преобразования метамодели, которые разработчик производит над ней при редактировании: переименование, удаление, добавление или перемещение элемента. Однако некоторые изменения в языке состоят из последовательности таких действий, и механическое применение всех действий в отдельности может привести к потере семантики мигрируемой модели (например, элемент удалён, а потом создан со значениями свойств по умолчанию, а нужно было заменить). Таким образом, историю необходимо анализировать для выявления таких последовательностей.

Как уже было отмечено ранее, часть эволюционных изменений нельзя вывести автоматически, что ведёт к необходимости предоставления разработчику языка возможности спецификации миграционной стратегии вручную. Упомянутый выше механизм рефакторингов представляет собой удобную возможность задавать сложные изменения метамодели. Таким образом, с рефакторингами можно обращаться так же, как с элементарными преобразованиями, автоматически выводя соответствующие миграции. Кроме того, для более тонкого контроля над процессом миграции моделей можно предоставить разработчику языка возможность задавать миграции, соответствующие данному рефакторингу метамодели.

Наконец, требуется способ определения миграции в случае, если она не задана разработчиком языка, а автоматически выведена не может быть ввиду неоднозначности вывода. В этом случае наиболее эффективным с точки зрения сохранения семантики модели является сообщение пользователю о неоднозначностях и принуждение его самостоятельно произвести миграцию.

В итоге, разработчик языка может выбирать способ спецификации миграционной стратегии исходя из требуемой точности и режима метамоделирования. При генерации и интерпретации есть возможность задавать рефакторинги и, дополнительно, соответствующие миграции. Кроме этого, можно полагаться на автоматический вывод миграционной стратегии. При метамоделировании “на лету” метамодель эволюционирует инкрементально, изменения небольшие, что может упростить анализ и увеличить эффективность автоматической миграции. Ошибки разработчика языка при задании миграции не приводят к невозможности загрузить модель, а всего лишь заставляют пользователя вручную мигрировать её (при этом не полностью, а только лишь касательно элементов, миграция которых неоднозначна).

## Автоматическая миграция

Основным и наиболее содержательным элементом рассмотренного подхода является механизм автоматической миграции, включающий в себя ведение истории изменений, восстановление по этой истории метамодели, соответствующей мигрируемой модели, вывод разницы двух метамоделей, вывод трансформаций по истории и разнице и собственно применение трансформаций к модели. Рассмотрим этот механизм в деталях.

### Протоколирование

Для вывода миграционной стратегии необходим доступ к старой и новой метамоделям, а также история изменений метамодели между этими версиями. Чтобы получить его, все изменения протоколируются и сохраняются вместе с метамоделью. Метамодель языка доступна в любом режиме, а значит, доступена и история изменений, и на данном этапе поддержка миграции не зависит от режима работы.

Записи в истории создаются при каждом элементарном действии или рефакторинге. Каждая запись хранит информацию о старом и новом состоянии метамодели, что позволяет легко откатывать изменения и получать старые версии метамодели. Для обозначения версий, к которым можно откатываться, при сохранении метамодели создаются контрольные точки.

Каждая модель хранит информацию о версиях всех языков, необходимых для её загрузки.

Таким образом, в любой момент времени доступны:

- текущая версия метамодели;
- номер версии метамодели, необходимой для корректной загрузки мигрируемой модели;
- сама эта версия метамодели;
- история изменений между старой и новой версиями метамодели.

### Модель разницы

История изменений позволяет проследить последовательности изменений в языке и выявить, какие из них логически составляют одно целое, но для вывода миграции также полезна модель разницы, дающая наглядное представление эволюции языка между версиями.

В идеале она должна представлять разницу между метамоделями так, как её видит человек: какой тип заменился каким, как изменились иерархии наследования, экземпляры каких типов требуется поменять ввиду их зависимости от изменившихся и т.д.

Следует заметить, что, хотя модель разницы выводится из истории изменений, из чего можно предположить её избыточность, в некоторых ситуациях с ней работать

удобнее. К примеру, она даёт компактное представление изменений в иерархиях наследования, что важно для миграции, но недоступно из одной только истории изменений.

В то же время в модели разницы отсутствует информация о последовательности изменений. При замене типа посредством удаления одного элемента и последующего создания другого в модели разницы старый и новый элемент оказываются никак не связаны, в то время как в истории можно распознать типичную последовательность и вывести соответствующую миграцию корректно.

Таким образом, история изменений и модель разницы - взаимодополняющие механизмы, увеличивающие эффективность автоматической миграции при совместной работе.

## Трансформации

С помощью истории изменений и модели разницы выводятся трансформации [8], описывающие изменения, которые нужно совершить над мигрируемой моделью для восстановления соответствия метамодели.

Трансформация состоит из левой и правой частей. Левая часть - образец, который нужно найти в модели, правая часть – образец, на который нужно заменить левую часть. Кроме этого, трансформация содержит также соответствие элементов левой и правой части и соответствие их свойств.

## Общая схема миграции

При открытии файла с моделью проверяется возможность её загрузки с помощью доступных в данный момент редакторов (представляющих собой подключаемые модули или загружаемых с помощью интерпретатора). В случае, если версии, требуемые для загрузки модели, меньше, чем доступные версии нужных редакторов, запускается миграция модели, в течение которой производятся следующие действия:

1. С помощью истории изменений воссоздаются старые версии необходимых метамоделей.
2. Для каждой из них выводится модель разницы.
3. История изменений между версиями и модель разницы передаются анализатору, который с помощью них выводит последовательность трансформаций, представляющих собой миграционную стратегию.
4. Трансформации (включая заданные разработчиком метамодели операторы) последовательно применяются к модели.

## Апробация

Рассмотрим работу данного механизма на примере. На Рис. 5а представлена часть метамодели языка BPMN [6]. При её создании разработчик метамодели допустил

несколько ошибок: опечатку в свойстве text элемента ControlFlow и несоответствие имени ControlFlow спецификации BPMN. Кроме того, было обнаружено, что тип Task (простое действие) следует изменить на более общий тип Activity, который может представлять как простые действия, так и сложные.

После исправления этих ошибок была получена новая метамодель языка, изображённая на Рис. 5б.

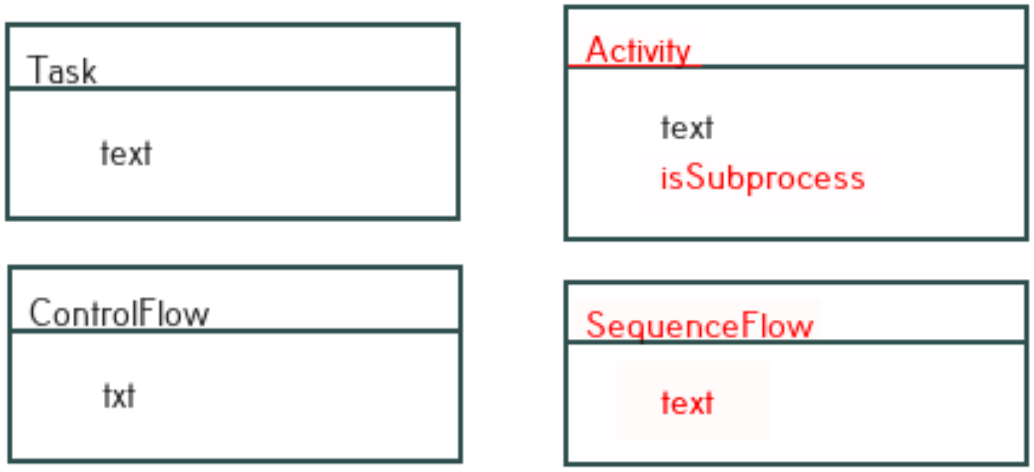



Рис. 5а: Устаревшая метамодель BPMN  Рис. 5б: Исправленная метамодель BPMN

Изменениям, выполненным в процессе редактирования этой метамодели, может соответствовать следующая история изменений:

1. Rename Property_2: txt -> text.
2. Add Property_3: isSubprocess, bool.
3. Rename Node_1: Task -> Activity.
4. Rename Edge_1: ControlFlow -> SequenceFlow.

По этим записям можно, имея новую версию метамодели, восстановить старую, а по этим двум метамоделям построить модель их разницы, которая в удобной форме описывает замену типов (включающую в себя переименование типа и/или переименование его свойств). Заметим, что в модели разницы нет никаких упоминаний о свойствах Activity.text или Activity.isSubprocess, несмотря на то, что последнее появлялось в истории изменений. Дело в том, что данные свойства в данном случае не влияют на процесс миграции: со свойством Activity.text не происходило никаких изменений, а Activity.isSubprocess только добавилось, что не может привести к потере данных из старых моделей. Таким образом, модель разницы отражает только те изменения, которые имеют важность для вывода трансформаций.

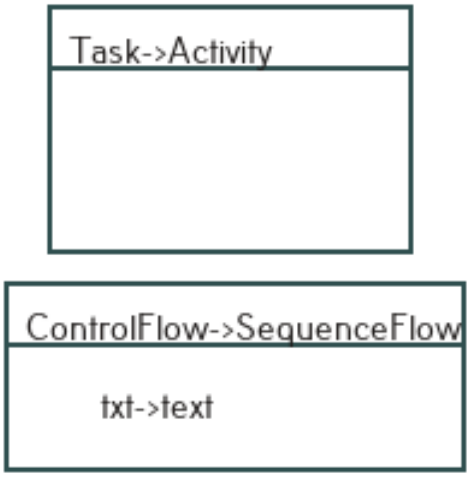



Рис. 6: Метамодель разницы для эволюции BPMN

Имея представление в виде модели разницы, легко построить соответствующие трансформации моделей:

1. ReplaceType: Task -> Activity.
2. ReplaceType: ControlFlow -> SequenceFlow @ txt -> text.

# Результат

В рамках проведённой работы были исследованы существующие подходы к миграции моделей и разработан гибридный подход, удовлетворяющим требованиям DSM-платформы QReal. Данный подход основан на сопоставлении моделей с использованием истории изменений метамодели и модели разницы. Кроме того, он предполагает возможность ручной спецификации миграционной стратегии с помощью механизма рефакторингов QReal.

К основным достоинствам данного подхода можно отнести гибкость, позволяющую разработчику языка выбирать между автоматической миграцией и ручным описанием трансформаций и/или рефакторингов. Кроме того, изменения метамодели отслеживаются, позволяя выводить некоторые простые миграции прямо в процессе редактирования метамодели. Рассмотренный способ миграции пригоден для реализации в системе, поддерживающей как подключение языков в виде заранее сгенерированных подключаемых модулей, так и одновременную разработку языка и модели на этом языке посредством метамоделирования “на лету”.

Помимо этого, такой механизм миграции легко расширяем. Пользователь DSM-платформы может расширить его возможности с помощью добавления собственных рефакторингов и операторов миграции. Кроме того, разработчик DSM-платформы обладает возможностью развивать подсистему автоматической миграции, не влияя при этом никаким образом на заданные пользователем трансформации.

Также в рамках данной курсовой работы реализовано ядро подсистемы автоматической миграции, которое включает в себя:

- ведение истории изменений метамодели;
- восстановление старой метамодели с помощью истории изменений;
- вывод модели разницы двух метамоделей;
- вывод и применение трансформаций.

С помощью данного механизма была реализована автоматическая миграция моделей при переименованиях элементов, связей и свойств.

## Дальнейшие перспективы

В дальнейшем планируется добавить поддержку рефакторингов и возможность ручной спецификации миграции. Также желательной является организация оповещений пользователя о результатах миграции: в случае, если разработчик метамодели забудет указать необходимую трансформацию, а автоматическая миграция окажется неспособна вывести её, пользователь должен отреагировать на эту ситуацию и произвести миграцию самостоятельно.

Ещё одно направление дальнейшей работы - исследования сложных или неоднозначных изменений с целью усовершенствования автоматической миграции. К возможным улучшениям можно отнести анализ графического представления метамодели и расположение её элементов относительно друг друга, а также анализ имён сущностей метамодели для поиска похожих имён, синонимов или связанных по смыслу понятий. Результаты этих исследований могут быть использованы в дальнейшем для разработки способов миграции моделей между принципиально разными метамоделями, а не только между различными версиями одной и той же метамодели.

# Литература


1. Brambilla, M., Cabot, J., Wimmer, M. Model-Driven Software Engineering in Practice // Morgan & Claypool. 2012. 182 p.
2. Gruschko, B., Kolovos, D., Paige, R. Towards synchronizing models with evolving metamodels // International Workshop on Model-Driven Software Evolution, MoDSE. 2007.
3. Jouault, F., Kurtev, I. Transforming Models with ATL // MoDELS'05 Proceedings of the 2005 international conference on Satellite Events at the MoDELS. Springer-Verlag Berlin, Heidelberg, 2006, pp. 128-138.
4. Kelly, S., Tolvanen, J. Domain-Specific Modeling: Enabling Full Code Generation // Wiley-IEEE Computer Society Press. 2008. 448 p.
5. OMG, MOF 2.0 Query/View/Transformations RPF, URL: http://www.omg.org/spec/QVT/1.1/
6. Owen, M. and Raj, J. BPMN and Bussines Process Management. Popkin Software, 2003
7. Rose, L.M., Paige, R.F., Kolovos, D.S., Polack, F.A.C. An Analysis of Approaches to Model Migration // Joint MoDSE-MCCM 2009 Workshop - Models and Evolution, 2009, pp. 6-15.
8. Rozenberg G. Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1: Foundations. World Scientific, 1997.
9. Кузенкова А.С., Литвинов Ю.В. Поддержка механизма рефакторингов в DSM-платформе QReal // Материалы межвузовского конкурса-конференции студентов, аспирантов и молодых ученых Северо-Запада "Технологии Microsoft в теории и практике программирования". СПб.: Изд-во СПбГПУ, 2013. С. 71-72.
10. Птахина А.И. Разработка метамоделирования “на лету” в системе QReal // Список-2013: Материалы всероссийской научной конференции по проблемам информатики. 2013г ., Санкт-Петербург. — СПб.: Изд-во ВВМ, 2012. С. 28-36.
11. Терехов А.Н., Брыксин Т.А., Литвинов Ю.В. QReal: платформа визуального предметно-ориентированного моделирования. // Программная инженерия, 2013, № 6, С. 11-19.