Модел. и анализ информ. систем. Т. **21**, № **6** (2014) 107–119
©*Агапова Т. Ю., Брыксин Т. А.*, 2014

УДК 004.4’22

Поддержка эволюции визуальных языков в платформе QReal

Агапова Т. Ю., Брыксин Т. А.

Санкт-Петербургский государственный университет
198504 Россия, Санкт-Петербург, Старый Петергоф, Университетский пр., 28

e-mail: tatjana.agapova@gmail.com, t.bryksin@spbu.ru

получена 19 сентября 2014

Ключевые слова: метамоделирование, предметно-ориентированные языки моделирования, эволюция метамодели, миграция моделей

Как и другие программные продукты, языки моделирования развиваются со временем. В результате изменений в языке, модели на данном языке могут перестать соответствовать новой метамодели языка, что ведет к невозможности работы с ними с помощью инструментов моделирования. Таким образом, возникает проблема переноса моделей на новую версию языка. В настоящее время существуют различные подходы к решению данной проблемы – от полностью ручных до практически полностью автоматизированных. Данная статья описывает гибридный подход к миграции моделей, реализованный в DSM-платформе QReal, разрабатываемой на кафедре системного программирования Санкт-Петербургского государственного университета. Рассматриваемая система накладывает некоторые специфические требования, такие как поддержка режимов интерпретации метамодели и метамоделирования “на лету”. Представленный в статье подход реализует миграцию моделей при использовании данных возможностей.

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. Примерами подобных инструментов являются Eclipse Modeling Framework[1], MetaEdit+[2]. 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.

[1]Eclipse Graphical Modeling Project, URL: http://www.eclipse.org/modeling/gmp/
[2]MetaEdit+, URL: http://www.metacase.com/products.html

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. Модификация моделей, которая восстанавливает их соответствие метамодели, называется миграцией (см. рис. 1).

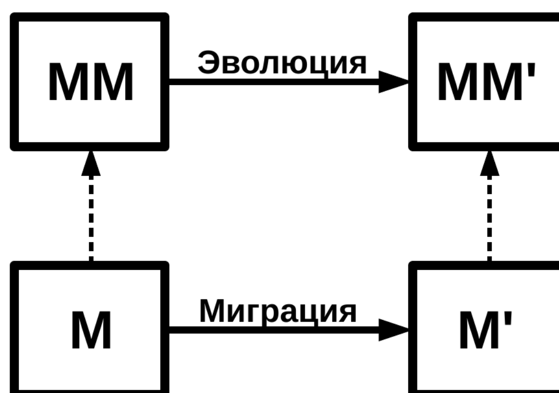

Рис. 1. Задача миграции моделей: при эволюции метамодели MM в метамодель MM' необходима миграция модели M, соответствующей MM, в модель M', соответствующую MM'

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 в новой версии. Кроме этого, 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. Далее в данной статье рассмотрены существующие подходы к определению миграционной стратегии, а также предлагается гибридный подход, разработанный для использования в DSM-платформе QReal и учитывающий специфические требования этой системы.

Схожая задача возникает в базах данных при эволюции схемы базы данных, когда данные необходимо перенести на новую схему [7]. С другой стороны, задача миграции моделей напоминает задачу обеспечения обратной совместимости приложений, когда требуется предоставить возможность работы со старыми файлами сохранений в новой версии приложения. При традиционной, не модельно-ориентированной разработке программного обеспечения, совместимость достигает-

ся с помощью программирования разработчиком приложения правил открытия старых сохранений.

1. Обзор существующих подходов

Вслед за [8] существующие подходы к миграции моделей можно разделить на три категории: ручная спецификация миграции, операторный подход и сопоставление моделей. Рассмотрим их подробнее.

1.1. 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 спецификации трансформаций. На рис. 2 приведён пример преобразования типа Person в тип Contact и описание этого преобразования на ATLAS Transformation Language [3].

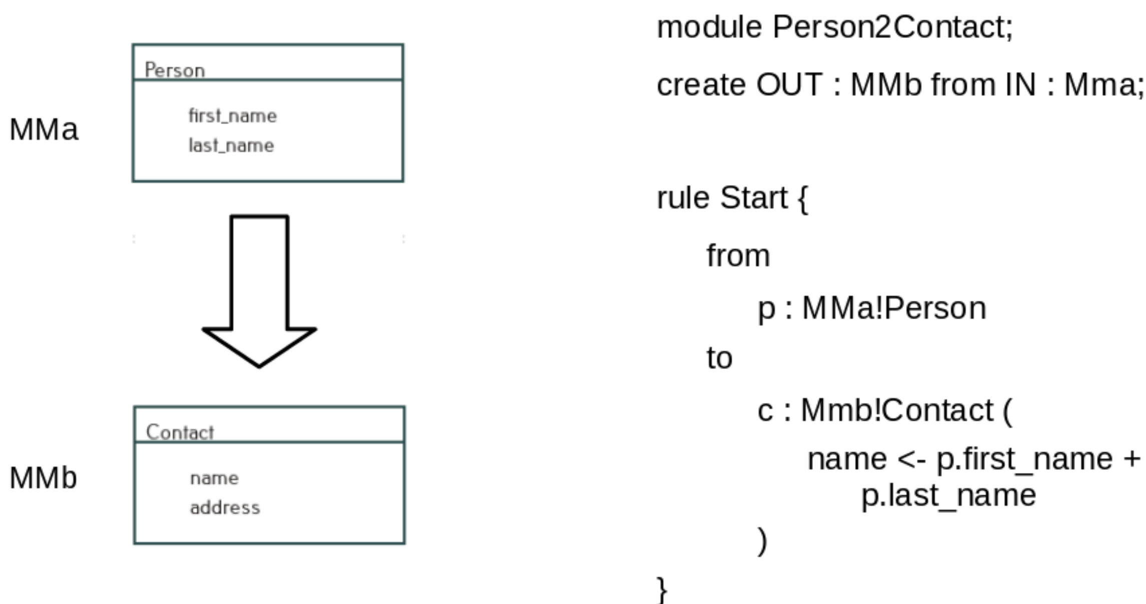

Рис. 2. Миграция преобразования типа (слева) и её описание на языке ATL (справа)

Ручная спецификация даёт полный контроль над миграцией, позволяет целиком определить миграционную стратегию и добиться максимальной точности преобразования, поскольку разработчик языка обладает наиболее полной информацией о семантике элементов языка и его эволюции. Недостаток данного подхода в его трудоёмкости. Разработчик языка вынужден дважды задать одни и те же изменения (при редактировании метамодели языка и при спецификации миграций), что повышает вероятность ошибок, несоответствий и неполноты миграционной стратегии,

которая сводит на нет попытку корректной миграции модели. Кроме того, разработчик языка вынужден выучить язык, на котором задаются миграции.

1.2. Операторный подход

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 визуального языка с помощью языка QVT-Relations [5] изображён на рис. 3.

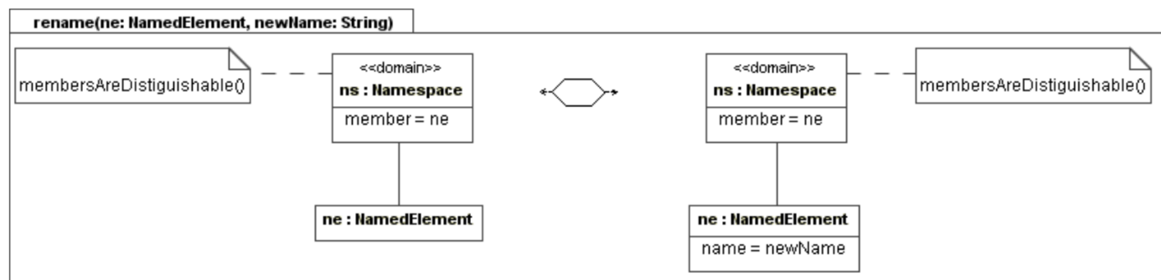

Рис. 3. Описание переименования элемента на языке QVT-Relations

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. Эффективность данного подхода зависит от полноты библиотеки операторов. Чем она полнее, тем точнее операторы, но в то же время излишняя 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.

1.3. Сопоставление моделей

Сопоставление моделей — автоматический подход, в котором для вывода миграционной стратегии используется один из двух механизмов: история изменений метамодели либо модель разницы старой и новой метамоделей.

В случае использования истории изменений метамодели все действия, совершённые над ней 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 в язык, исчезли или

заменились на другие, как изменились иерархии наследования, отношения вложенности элементов и т.д. Эта информация используется для вывода трансформаций мигрируемой модели.

Это наименее трудоёмкий для разработчика языка способ миграции, так как не требует от него никаких дополнительных действий.

1.4. Ограничения автоматической миграции

К сожалению, миграционная стратегия не может быть корректно выведена автоматически в общем случае, как было показано в [2]. Одной из причин этого являются сложные изменения метамодели, которые могут, во-первых, затрагивать несколько элементов (например, при изменениях в иерархии наследования), а во-вторых, состоять из последовательности более простых действий над метамоделью (к примеру, замена типа производится с помощью удаления старого типа и создания нового).

Ещё более сложной проблемой является зависимость миграционной стратегии от восприятия модели человеком и семантики, которую человек вкладывает в модель.

Данные сложности делают задачу автоматической миграции моделей неразрешимой в общем случае, 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.

2. Требования к механизму миграции моделей в QReal

QReal – это DSM-платформа, разрабатываемая на кафедре системного программирования Санкт-Петербургского государственного университета [12]. 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 [10], заключающееся в сопоставлении рефакторинга метамодели

и трансформации модели. Это позволит объединить редактирование метамодели и спецификацию миграционной стратегии, избавляя разработчика языка от необходимости производить одни и те же действия дважды и уменьшая вероятность ошибки.

Метамоделирование “на лету” [11] – 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”. Поскольку, как было указано выше, некоторые изменения в языке могут состоять из нескольких шагов пользователя, требуется возможность “на лету” распознавать последовательности действий, логически представляющие собой одно изменение. Кроме того, в этом режиме метамодель недоступна пользователю в явном виде, 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. Возможность выбирать между разными способами миграции (рефакторинги или автоматическая миграция). При этом эти механизмы не должны конфликтовать между собой и допускать одновременное использование.

2. Максимально возможная точность трансформации, сохранение семантики мигрируемой модели.

3. Поддержка всех трёх режимов работы с метамоделью.

3. 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 как история изменений метамодели, так 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, лог 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. Уже 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, при желании, 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).

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 изменений и разнице и собственно применение трансформаций к модели. Рассмотрим этот механизм в деталях.

3.1. Протоколирование

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

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;
- лог изменений между старой и новой версиями метамодели.

3.2. 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 история изменений позволяет распознать типичную последовательность и вывести соответствующую миграцию корректно.

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

3.3. Трансформации

С помощью истории изменений метамодели и модели разницы выводятся трансформации [9], 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.

3.4. 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, требуе-

мые 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. Лог 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.

Рассмотрим работу данного механизма на примере. На рис. 4 слева представлена часть метамодели языка описания бизнес-процессов BPMN [6]. При её создании разработчик метамодели допустил несколько ошибок: опечатку в свойстве text элемента ControlFlow и несоответствие имени ControlFlow спецификации языка. 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 BPMN, изображённая на рис. 4 справа.

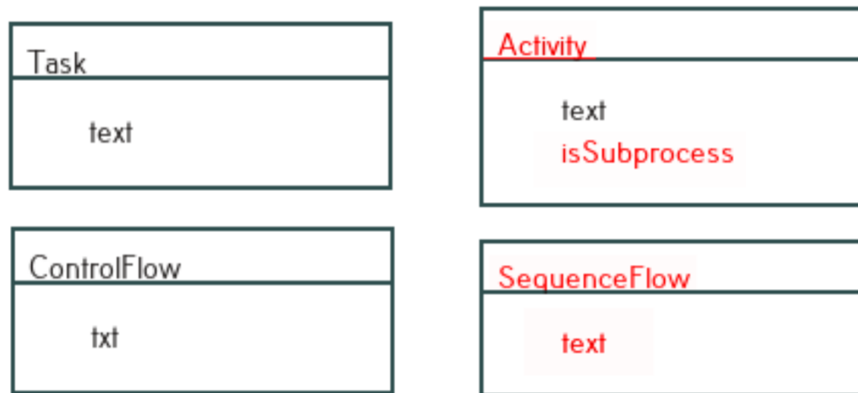

Рис. 4. Фрагменты устаревшей (слева) и исправленной (справа) метамодели языка BPMN

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, имея новую версию метамодели, можно 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 (см. рис. 5).

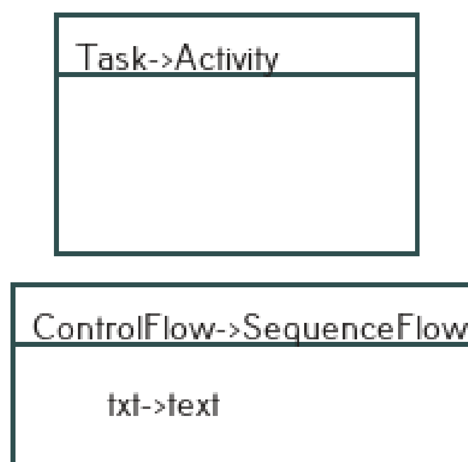

Рис. 5. Метамодель разницы для эволюции BPMN

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

1. ReplaceType: Task -> Activity.

2. ReplaceType: ControlFlow -> SequenceFlow @ txt -> text.

4. Результат, дальнейшие перспективы

В данной статье рассмотрен гибридный подход к миграции моделей, сочетающий сильные стороны ручной спецификации, операторного подхода и сопоставления моделей.

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, 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.

Наконец, если разработчик метамодели забудет указать необходимую трансформацию, а автоматическая миграция окажется неспособна вывести её, пользователь будет оповещён о необходимости восстановить корректность модели. Это возвращает нас в ситуацию, при которой средства миграции отсутствуют, но в данном случае эта ситуация является исключительной, так как можно рассчитывать, что подавляющая часть трансформаций будет указана разработчиком языка или выведена автоматически, что значительно снижает трудоёмкость правки модели вручную.

Можно выделить два направления, в отношении которых можно усовершенствовать данный подход и его реализацию. Первое из них – предоставление богатой библиотеки рефакторингов и операторов с целью упрощения процесса спецификации трансформаций разработчиком метамодели. Второй способ улучшения – исследование 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.

Список литературы

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. P. 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. *Rahm E., Bernstein P.A.* A Survey of Approaches to Automatic Schema Matching // The VLDB Journal. Springer-Verlag, 2001.

8. *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. P. 6–15.

9. *Rozenberg G.* Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1: Foundations. World Scientific, 1997.

10. *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 [*Kuzenkova A.S., Litvinov Yu.V.* Podderzhka mekhanizma refaktoringov v DSM-platforme QReal // Materialy mezhvuzovskogo konkursa-konferentsii studentov, aspirantov i molodykh uchenykh Severo-Zapada "Tekhnologii Microsoft v teorii i praktike programmirovaniya". St. Petersburg: SPbSTU Press, 2013. P. 71–72 (in Russian)].

11. *Птахина А.И.* Разработка метамоделирования “на лету” в системе QReal // Список-2013: Материалы всероссийской научной конференции по проблемам информатики. СПб.: Изд-во ВВМ, 2013. С. 28–36. [*Ptakhina A.I.* Razrabotka metamodelirovaniya “na letu” v sisteme QReal // Spisok-2013: Materialy vserossiyskoy nauchnoy konferentsii po problemam informatiki. St. Petersburg: VVM, 2013. P. 28–36 (in Russian)].

12. *Терехов А.Н., Брыксин Т.А., Литвинов Ю.В.* QReal: платформа визуального предметно-ориентированного моделирования. // Программная инженерия. 2013. № 6. С. 11–19. [*Terekhov A.N., Bryksin T.A., Litvinov Yu.V.* QReal: platforma vizual’nogo predmetno-orientirovannogo modelirovaniya // Programmnaya inzheneriya. 2013. No 6. P. 11–19 (in Russian)].

Support for Visual Languages Evolution in DSM-platform QReal

Agapova T. Y., Bryksin T. A.

St. Petersburg State University,
Universitetsky prospekt, 28, Peterhof, St. Petersburg, 198504, Russia

Keywords: metamodelling, domain specific modelling languages, metamodel evolution, model migration

Like other software artefacts, DSMLs evolve in time. When a DSML changes, instance models might no longer conform to the new DSML metamodel and hence cannot be manipulated with a modelling tool. Therefore, a need for models migration to a new version of metamodel arises. Today, various approaches to this problem exist - from entirely manual to mostly automated. This paper describes a hybrid approach to model migration implemented in DSM platform QReal, which is being developed by the research group of Software Engineering Chair of St. Petersburg State University. That DSM platform implies some specific requirements, such as the support of metamodel interpretation and metamodeling “on the fly” modes. The presented approach realizes model migration when using one of those specific features.

Сведения об авторах:
Агапова Татьяна Юрьевна,
Санкт-Петербургский государственный университет,
математико-механический факультет, студентка;
Брыксин Тимофей Александрович,
Санкт-Петербургский государственный университет,
математико-механический факультет, старший преподаватель