САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

ЭВОЛЮЦИЯ ЯЗЫКОВ ПРИ МЕТАМОДЕЛИРОВАНИИ “НА ЛЕТУ” В DSM-ПЛАТФОРМЕ QREAL

Дипломная работа студента 545 группы
Птахиной Алины Ивановны

Научный руководитель	…… / подпись /	ст.преп. Литвинов Ю.В.
Рецензент	…… / подпись /	ст. преп. Кириленко Я. А.
“Допустить к защите” заведующий кафедрой,	…… / подпись /	д.ф.-м.н., проф. Терехов А.Н.

Санкт-Петербург
2014

SAINT PETERSBURG STATE UNIVERSITY

Mathematics & Mechanics Faculty

Software Engineering Department

LANGUAGES EVOLUTION IN METAMODELING “ON FLY” IN QREAL DSM-PLATFORM

by

Ptakhina Alina

Graduation Thesis

Scientific supervisor: …… Senior Lecturer Y.V. Litvinov

Reviewer …… Senior Lecturer I. A. Kirilenko

“Approved by” …… Professor A. N. Terekhov

Head of Department

Saint Petersburg

2014

Оглавление

Введение

В современном мире большой интерес вызывают средства, позволяющие ускорить процесс разработки программного обеспечения. Для этой цели применяются различные технологии, в том числе и визуальное моделирование[1]. При таком подходе программа представляется в виде набора диаграмм. Каждая диаграмма представляет собой модель, описывающую фрагмент функциональности ПО. Это позволяет сконцентрироваться на конкретной задаче, а не рассматривать все многообразие предметной области. Такой подход широко применяется при промышленной разработке.

Существуют программные средства, реализующие методы визуального моделирования. Эти средства поддерживают различные виды диаграмм, предоставляют возможность генерации кода в разные целевые платформы программирования и используются при анализе, проектировании и разработке программного обеспечения.

При проектировании часто возникает потребность в создании множества моделей предметной области. Соответственно возникает потребность в языке, который бы позволил упростить процесс описания этих моделей. Язык, используемый для создания других языков моделирования, называется метаязыком. Модели на метаязыке содержат описание всех абстракций визуального языка и правила построения из них визуальных моделей. Модель языка моделирования называется метамоделью.

Довольно часто при визуальном моделировании мы имеем дело с визуальными языками, созданными для использования в рамках конкретной предметной области. Они называются предметно-ориентированными визуальными языками моделирования (DSVL, Domain Specific Visual Language). Эти языки, в отличие от языков общего назначения, разрабатываются для решения определенного круга задач и, благодаря этому, могут привести к более быстрому и эффективному их решению. Такое

[1] Подробнее об этом можно прочитать, например, в [1].

сужение области применения визуального моделирования и получение за счет этого более эффективного автоматизированного решения является сутью подхода предметно-ориентированного моделирования (DSM, Domain-Specific Modeling).

Существуют специальные инструментальные средства, позволяющие быстро создавать визуальные языки и редакторы для них. Они называются DSM-платформами[2][3] или языковыми инструментариями. Такие инструменты позволяют быстро создавать инструментальные средства для визуальных языков.

Предметно-ориентированные визуальные языки часто подвергаются изменениям в зависимости от нужд предметной области. При этом бывают ситуации, когда модели, созданные в ранних версиях языка, в более поздних версиях не поддерживаются: они содержат элементы, которые были удалены, или у элементов был изменен набор свойств и т.д. Возникает задача автоматической адаптации моделей.

1 Обзор предметной области

В данной работе мы будем рассматривать DSM-платформу QReal [4] – проект научно-исследовательской группы кафедры системного программирования Санкт-Петербургского государственного университета.

1.1 DSM-платформа QReal

Платформа QReal позволяет автоматически генерировать код произвольных визуальных редакторов по описаниям их метамоделей. Таким образом, с помощью метамоделирования мы получаем быструю реализацию предметно-ориентированного визуального языка. Также в данной системе в рамках курсовой работы третьего курса автора [5] был реализован интерпретатор метамоделей, позволяющий эмулировать функциональность сгенерированного редактора. Благодаря ему появилась возможность производить изменения в метамодели без перегенерации кода и пересборки редактора. А в рамках курсовой работы четвертого курса автора [6] было реализовано так называемое метамоделирование “на лету”, которое позволило быстро и легко расширять систему.

1.2 Метамоделирование “на лету” в системе QReal

Метамоделирование “на лету” (см. Рис. 1) предоставило пользователю возможность вносить изменения в визуальный язык программирования прямо в процессе работы с языком: добавлять новые элементы языка, удалять ненужные с его точки зрения элементы и изменять существующие (под этим понимается изменение графического изображения элемента и его свойств). При этом все новые и измененные сущности сразу доступны для построения диаграммы, и в случае возможных конфликтов и некорректности системы пользователю предоставляется соответствующая информация о возможных последствиях. При таком подходе редактирование модели и метамодели объединено, что избавляет пользователя от необходимости мыслить в терминах двух абстракций. Уровень метамодели скрыт от пользователя, что позволяет ему полностью сконцентрироваться на задаче, а не на создании инструментария. При метамоделировании “на лету”

пользователь может использовать как существующую метамодель, загрузив ее в систему, так и может создать свой собственный язык “с нуля” и тем самым получить быструю реализацию предметно-ориентированного визуального языка.

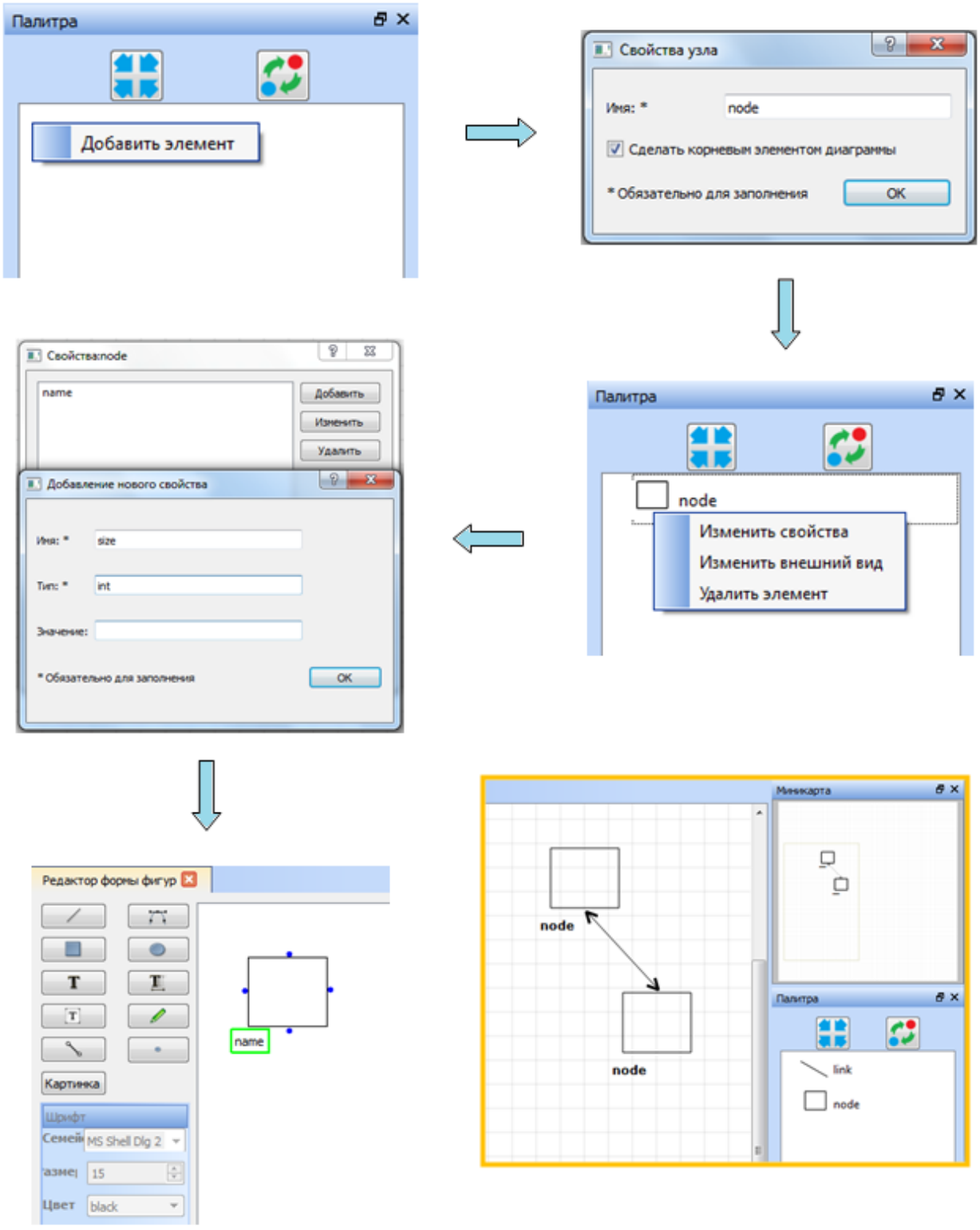

Рис 1. Метамоделирование “на лету” в QReal.

1.3 Эволюция визуальных языков моделирования

В течение жизненного цикла предметно-ориентированного решения используемые визуальные языки могут меняться. Представим ситуацию: пользователь создает визуальный язык для своей предметной области. Задав набор элементов языка, он на их основе создает модель. После этого он сохраняет модель и заканчивает на этом работу. Через некоторое время у него возникает потребность в создании модели, схожей по объектам с предыдущей моделью. Пользователь в режиме интерпретируемой метамодели открывает имеющеюся у него метамодель и производит необходимые изменения: добавляет новые элементы языка, удаляет ненужные, редактирует свойства элементов. После этого он создает нужную в данный момент модель. В такой ситуации произошло изменение визуального предметно-ориентированного языка. Старая модель пользователя стала несовместимой с текущей версией языка. О некоторых элементах, которые используются в ней, система в текущий момент может не иметь информации (они были удалены при изменении визуального языка), также возможны ситуации, когда элементам были добавлены новые свойства, либо были изменены или удалены существующие. Появляется задача адаптации созданных моделей под новые версии языка и поддержки эволюции языков.

2 Постановка задачи

Целью данной работы является обеспечение совместной эволюции модели и метамодели при метамоделировании "на лету" в DSM-платформе QReal. Для достижения этой цели были поставлены следующие задачи:

1. Разработать методику адаптации моделей визуального языка при изменении метамодели в процессе метамоделирования "на лету".
2. Реализовать средство для автоматической адаптации моделей под новые версии языка и поддержки эволюции языков при метамоделировании "на лету".
    a. Реализовать возможность открытия модели, созданной на основе предыдущей версии языка.
    b. Предоставить возможность восстановления удаленных и переименованных элементов языка и их свойств.
3. Произвести апробацию предложенной методики и выполнить тестирование реализованного средства на примере простого DSVL-языка.

3 Анализ существующих DSM-платформ

Рассмотрим, какие подходы и методики применяются для поддержки совместной эволюции модели и метамодели в современных DSM-платформах.

3.1 MetaEdit+

MetaEdit+ [7] – это среда для создания и использования предметно-ориентированных языков, разработанная в рамках научно-исследовательского проекта MetaPHOR в University of Jyväskylä. В состав MetaEdit+ входит средство создания языков моделирования и генераторов MetaEdit+ Workbench и инструмент MetaEdit+ Modeler, в котором полученные модельные языки можно использовать для создания и редактирования моделей (см. Рис. 2).

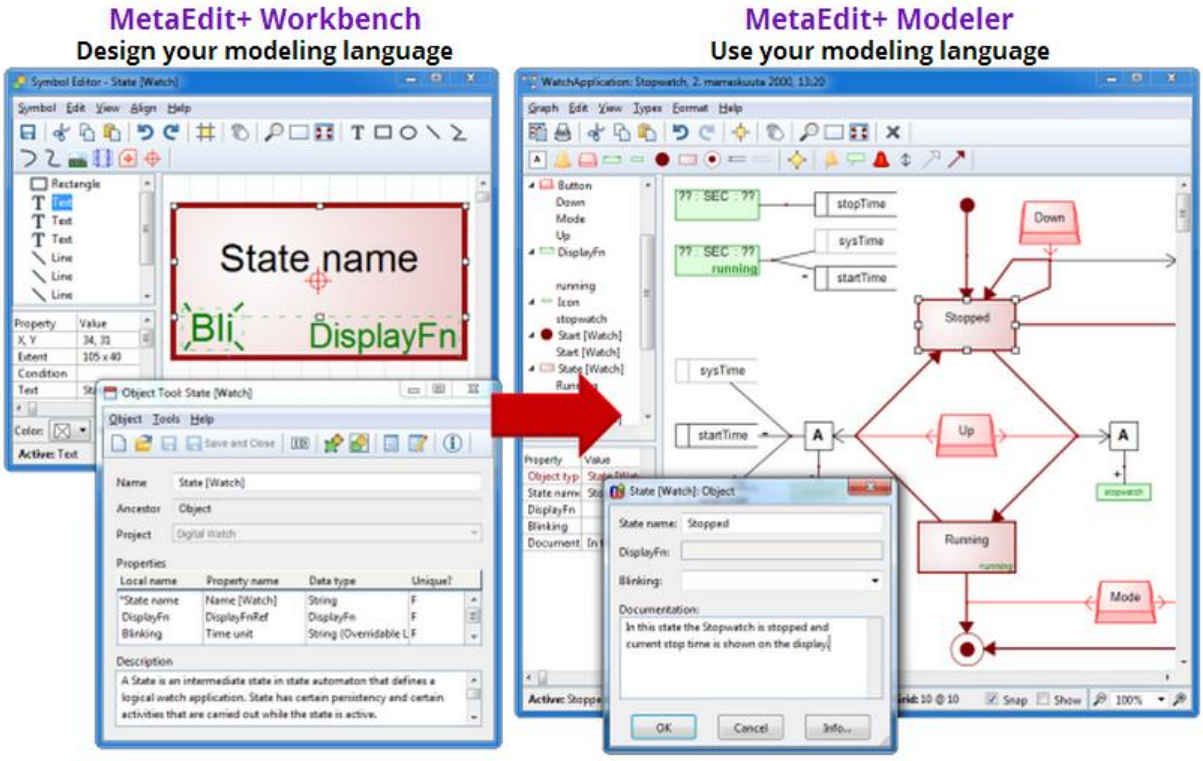

Рис 2. MetaEdit+ Domain-Specific Modeling (DSM) environment

При построении метамодели в MetaEdit+ Workbench используется язык метамоделирования GOPPRR (Graph-Object-Property-Port-Relationship-Role), получивший свое название от основных понятий, которыми он оперирует:

- граф (Graph) – создаваемый язык моделирования (совокупность объектов и связей);
- объект (Object) – сущность модельного языка;
- свойство (Property) – атрибут какого-либо объекта графа;
- порт (Port) - место на объекте, куда можно прицепить роль;
- связь (Relation) – задает отношение между объектами;
- роль(Role) – определяет, каким образом объект участвует в связи.

С построения GOPRR-модели и начинается создание метамодели. С помощью MetaEdit+ Workbench можно определять основные понятия языка, устанавливать на них ограничения и правила, задавать атрибуты, взаимосвязи между понятиями и различные правила интеграции нескольких языков в одной системе. Графическое представление понятий языка можно задать, воспользовавшись встроенным в систему редактором Symbol Editor, который позволяет как создать собственное изображение, так и загрузить уже существующее в формате SVG или BMP.

MetaEdit+ использует подход, основанный на интерпретации метамоделей. При этом при внесении изменений в описание метамодели все необходимые преобразования в соответствующих моделях DSM-платформа производит сама. Следует отметить, что созданный в системе объект невозможно удалить, но информацию о нем можно скрыть от пользователя: если явно не указывать, что данный объект используется в графе, то в палитре MetaEdit+ Modeler он отображен не будет. В случае же, если мы захотим добавить удаленный элемент из старой модели в палитру, нам придется явно добавить его в граф с помощью инструмента Graph Tool.

Такое поведение близко к тому, которое мы хотим реализовать, однако при таком подходе пользователь вынужден работать отдельно с моделью и метамоделью. Нашей же задачей является поддержка совместной эволюции модели и метамодели. Это означает, что все необходимые преобразования должны происходить автоматически, чтобы избавить пользователя от необходимости мыслить в терминах двух абстракций.

3.2 MetaLanguage

Система MetaLanguage – это языковой инструментарий, предназначенный для создания визуальных динамически настраиваемых предметно-ориентированных языков моделирования[8]. Впервые данная система была упомянута в 2008 году в межвузовском сборнике научных статей “Математика программных систем” в статье Л.Н.Лядовой и А.О.Сухова “Языковой инструментарий системы MetaLanguage”[9], в которой описывались базовые элементы метаязыка, и статье А.О.Сухова под названием “Среда разработки визуальных предметно-ориентированных языков моделирования”[10], в которой описывались преобразования моделей, производимые в случае изменения метамодели для поддержки согласованности системы. К сожалению, в открытом доступе средство MetaLanguage отсутствует, и оценить на практике его достоинства и недостатки не представилось возможным. Однако статьи заслуживают внимания, поскольку свидетельствуют о том, что исследования в данном направлении велись и были получены результаты.

Среда разработки MetaLanguage включает следующие компоненты:

- графический редактор – используется для создания, изменения, удаления моделей, а также установления взаимосвязей между различными моделями, описанными с помощью определенных графических нотаций;
- браузер объектов – инструментальное средство, предназначенное для просмотра и редактирования информации, хранящейся в репозитории;
- репозиторий – хранилище, содержащее информацию о метамоделях, моделях, сущностях, отношениях, атрибутах, ограничениях, пиктограммах, используемых для отображения сущностей и отношений; фактически представляет собой реляционную БД, может работать в многопользовательском режиме, для согласованности действий пользователей использует механизм транзакций;

- валидатор – проверяет соответствие модели ограничениям, заданным пользователем;
- генератор – на основе имеющихся моделей генерирует XML-файл, содержащий информацию о модели, и документацию, включающую название модели, информацию о разработчиках, которые создавали ее, графическое представление модели со ссылками на описание отдельных частей.

При открытии метамодели из репозитория загружаются все модели, созданные с помощью этой метамодели, все сущности, отношения метамодели и соответствующих моделей. При удалении метамодели удаляются все модели, созданные на ее основе. При внесении изменений в метамодель для поддержания системы в согласованном состоянии изменения вносятся и в зависимые от нее модели.

Рассмотрим, что происходит при удалении сущности из метамодели. Сначала выполняется поиск всех экземпляров этой сущности во всех моделях, созданных с помощью данной метамодели. Далее для каждого экземпляра удаляемой сущности происходит удаление всех отношений, связанных с данным экземпляром. После этого происходит удаление из репозитория информации обо всех отношениях, связанных с удаляемой сущностью метамодели, и удаление самой этой сущности: ее атрибутов, операций и ограничений.

Из этого обзора можно сделать вывод, что подход, применяемый в системе MetaLanguage, позволяет производить адаптацию моделей под новые версии языка и тем самым обеспечивает согласованность данной системы. Однако при таком подходе теряется возможность восстановления удаленных сущностей в виду того, что происходит полное удаление всей информации о них из репозитория. Кроме того, пользователь в данной системе по-прежнему вынужден мыслить в терминах двух абстракций.

4 Описание методики

Как упоминалось ранее, визуальные языки в современных DSM-платформах задаются с помощью метамоделей, то есть моделей синтаксиса языка. Они описывают, какие элементы могут находиться на диаграмме, какие свойства есть у этих элементов, и как элементы могут быть связаны друг с другом.

Для реализации поддержки эволюции визуальных языков воспользуемся следующей идеей: будем считать, что метамодель для измененных визуальных языков одна и всегда поддерживает актуальное для всех версий языка состояние. Это означает, что все объекты, созданные на разных этапах моделирования, всегда содержатся в метамодели. Информация об удаленных элементах и свойствах также хранится в метамодели, хотя на диаграмме и в палитре данные элементы не отображаются. Также мы будем хранить все предыдущие имена свойств элементов – это позволит избежать проблем в случае, если свойство было переименовано. И будем производить автоматическое преобразование свойств: автоматически добавлять новые свойства со значением по умолчанию, удалять несуществующие свойства, производить переименование свойств, если оно имело место быть, при смене типа свойства будем приводить, если это возможно, значения к данному типу, а в случае, если невозможно – обнулять.

Предложенный подход позволяет избежать конфликтных ситуаций при загрузке старой модели в версии новой метамодели, предоставляет возможность восстановления удаленных элементов и свойств, а также обеспечивает сохранение информации об иерархии эволюции метамоделей.

Остановимся на иерархии эволюции метамоделей и рассмотрим, что это такое, на примере, приведенном на рисунке 3. Предположим, мы создали метамодель Metamodel, далее мы ее каким-либо образом изменили, допустим, добавили новые элементы в нее – получили другую метамодель Metamodel’. Потом мы на основе метамодели Metamodel’ создали две другие метамодели для различных целей: ER Metamodel и UML Metamodel, каждую

из которых детализировали и получили метамодели следующего уровня: ER’ Metamodel, ER’’ Metamodel и UML2 Metamodel, SysML Metamodel соответственно.

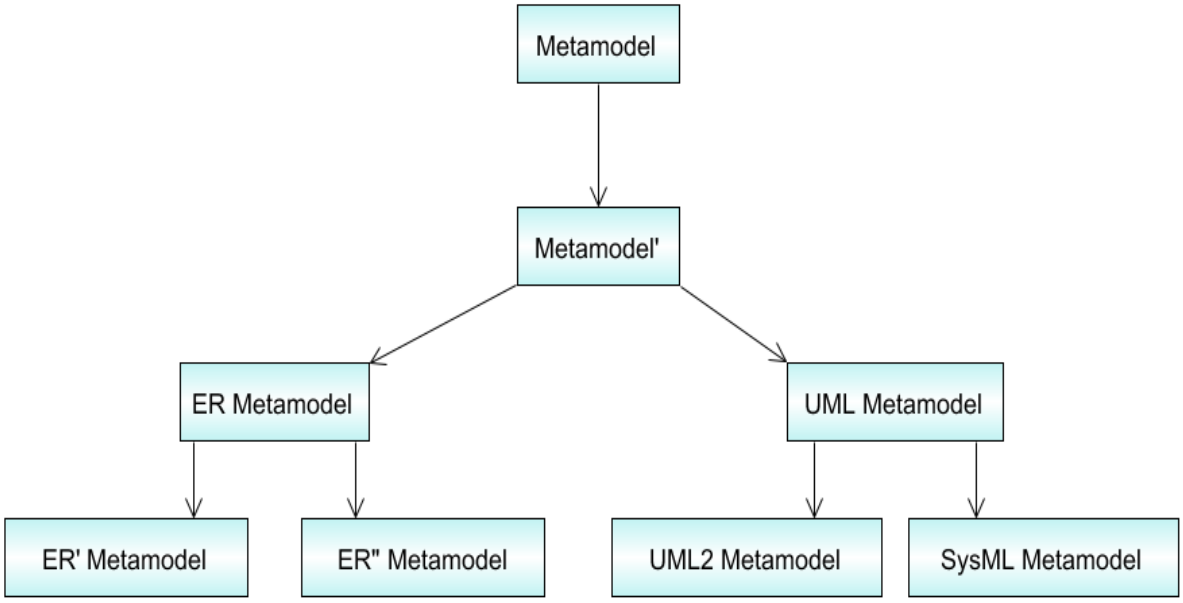

Рис 3. Пример иерархии эволюции метамоделей.

В этом примере метамодель SysML Metamodel является модифицированной версией метамодели UML Metamodel, и, значит, пользователь должен иметь возможность открывать модели, построенные на основе UML Metamodel, а также на основе метамоделей MetaModel’ и Metamodel. Что касается моделей, построенных на основе других метамоделей, приведенных в примере, то никакой информации об их объектах не должно содержаться в метамодели SysML Metamodel, поскольку данные метамодели не являются ее ранними версиями и, по сути, не имеют никакого отношения к ней.

Рассмотрим, как предложенный подход можно применить в DSM-платформе QReal, с помощью диаграммы, представленной на рис. 4. Пусть в режиме метамоделирования “на лету” мы создали предметно-ориентированный визуальный язык и сохранили полученную метамодель Metamodel 1 и модель model 1. Далее мы открыли созданную метамодель Metamodel 1 и постепенно создали сначала одну измененную версию языка –

получили Metamodel 2 и model 2, а потом другую, никак не связанную с предыдущей, версию языка, получив Metamodel 3 и model 3.

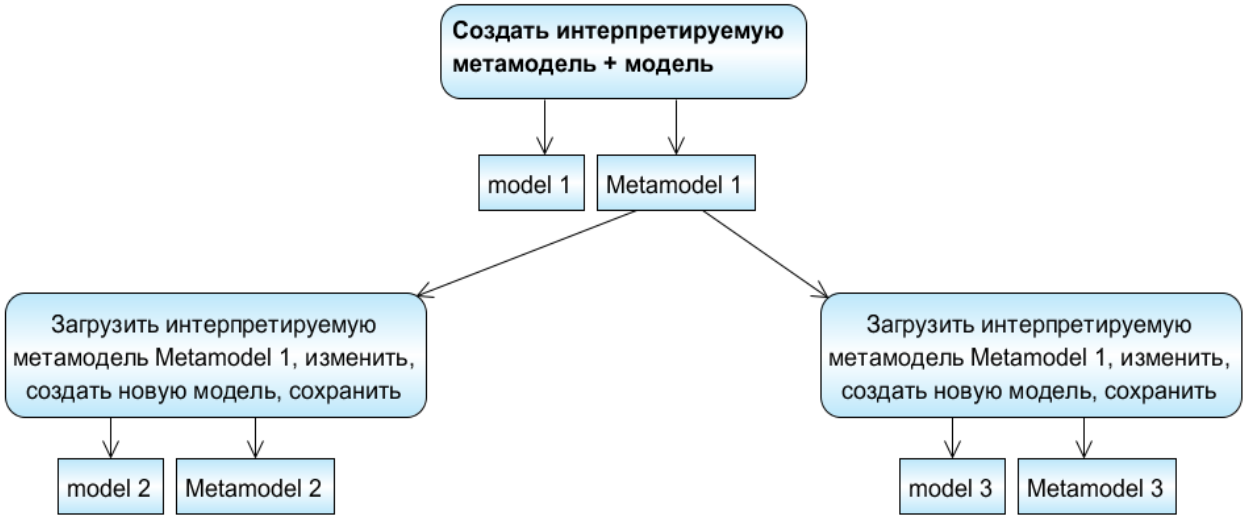

Рис 4. Иллюстрация к применению методики в QReal.

В данном примере метамодели Metamodel 2 и Metamodel 3 содержат всю информацию об объектах и свойствах, которые присутствуют в метамодели Metamodel 1. Таким образом, если мы откроем в режиме метамоделирования “на лету” любую из данных метамоделей и попытаемся открыть модель model 1, то после некоторых преобразований, которые будут рассмотрены далее, мы сможем увидеть интересующую нас модель. Однако сведениями об объектах, специфичных для данных метамоделей, и модификациях, которые произошли после того, как мы изменили Metamodel 1, ни одна из данных метамоделей по отношению к другой не владеет. Это означает, что на базе одного визуального языка мы получили две совершенно разные, никак не связанные между собой версии этого языка. А значит модели, созданные для одного визуального языка, не должны поддерживаться в другом.

5 Особенности реализации

Рассмотрим, как предложенная методика может быть реализована в виде инструментального средства. Для этого подробнее остановимся на некоторых технических моментах и автоматических преобразованиях, которые должны происходить в системе.

5.1 Адаптация модели под новую версию языка

При открытии модели, созданной на основе предыдущей версии языка, конфликтные ситуации могут возникнуть в следующих случаях:

- Элемент, присутствующий в модели, был удален в процессе моделирования и отсутствует в текущей версии языка.
- У элемента, присутствующего в модели, был изменен набор свойств.

Рассмотрим подробнее эти случаи и опишем необходимые преобразования и действия, которые должны происходить в системе, в каждом конкретном случае.

Как упоминалось ранее, при удалении элемента информация о нем продолжает храниться в системе, хотя в палитре данный элемент отсутствует и скрыт от пользователя. Для того, чтобы иметь возможность отображать удаленный элемент при открытии старой модели и предоставить пользователю возможность его дальнейшего добавления в палитру было принято следующее решение: установить всем элементам метамодели новое булевское свойство “isHidden”, показывающее был ли удален данный элемент и следует ли скрывать информацию о нем от пользователя. Это свойство не отображается в редакторе свойств элемента и задается системой автоматически. В случае, если пользователь решит восстановить удаленный ранее элемент, то он просто может найти любой его экземпляр на диаграмме, вызвать контекстное меню элемента правой кнопкой мышки и выбрать пункт меню “Добавить элемент в палитру”. После этого элемент автоматически добавится в палитру и будет доступен для построения диаграммы.

Рассмотрим, что происходит в случае, если в измененной версии языка

меняется набор свойств элементов. В такой ситуации, чтобы избежать возможных конфликтов при открытии модели, созданной на основе ранней версии языка, необходимо производить преобразование свойств. В данной работе был предложен следующий интуитивно понятный и логичный способ преобразования: при открытии старой модели в измененной версии языка необходимо элементам этой модели автоматически добавлять новые свойства со значением по умолчанию, удалять несуществующие свойства и производить переименование свойств, если оно имело место быть. В случае, если был изменен тип свойства, то согласно описанной ранее методике, будет произведена попытка преобразования значения к данному типу.

5.2 Обработка одноименных элементов и свойств

Для поддержки эволюции языков было реализовано восстановление удаленных и переименованных свойств элементов. Таким образом, в случае, если пользователь попытается добавить свойство с именем, использовавшимся ранее, то ему будет предоставлен список всех удаленных и переименованных свойств с этим именем. С технической стороны это выглядит следующим образом: для каждого свойства элемента хранится список всех его предыдущих имен. В случае, если пользователь производит переименование свойства, новое имя добавляется в этот список. При добавлении нового свойства элементу сначала происходит проверка, в ходе которой у данного элемента ищутся все одноименные свойства, когда-либо использовавшиеся в системе. Если список таких свойств не пуст, то сведения о них предоставляется пользователю в виде диалогового окна. В диалоговом окне восстановления свойств для каждого элемента списка одноименных свойств указывается вся необходимая информация о нем: состояние (удалено/переименовано в [текущее имя свойства]/используется), тип свойства и значение по умолчанию. Пользователь по желанию может либо восстановить прежнее свойство (при этом в случае переименованного свойства произойдет смена его текущего имени), либо создать одноименное

новое. При этом никаких конфликтных ситуаций не возникнет, поскольку, хотя данные свойства и имеют одинаковое имя, внутри системы им в соответствие будут поставлены различные идентификаторы.

В рамках данной работы была реализована также возможность восстановления удаленных элементов языка, в случае если пользователь пытается создать одноименную сущность или связь. Для этого для всех элементов языка при создании новой сущности или связи аналогичным образом происходит поиск всех элементов данного типа с указанным именем. Если список таких элементов не пуст, то сведения о них предоставляются пользователю в виде диалогового окна, содержащего для каждого элемента информацию о всех его свойствах с типом и значением по умолчанию. Аналогично диалоговому окну восстановления свойств, пользователю предоставляется возможность либо восстановить один из элементов приведенного списка со всеми его свойствами, либо создать свой новый элемент с таким именем.

6 Апробация

Проведем апробацию предложенной методики и выполним тестирование реализованного средства на примере простого DSVL-языка. Для апробации был специально выбран простой визуальный язык, чтобы не углубляться в предметную область, для которой этот язык был разработан, а сосредоточиться на самом процессе адаптации и поддержки эволюции языков.

В качестве предметно-ориентированного визуального языка рассмотрим DSVL язык простой графики с возможностью задания формы фигуры с помощью графических примитивов и отношений над примитивами (см. Рис 4).

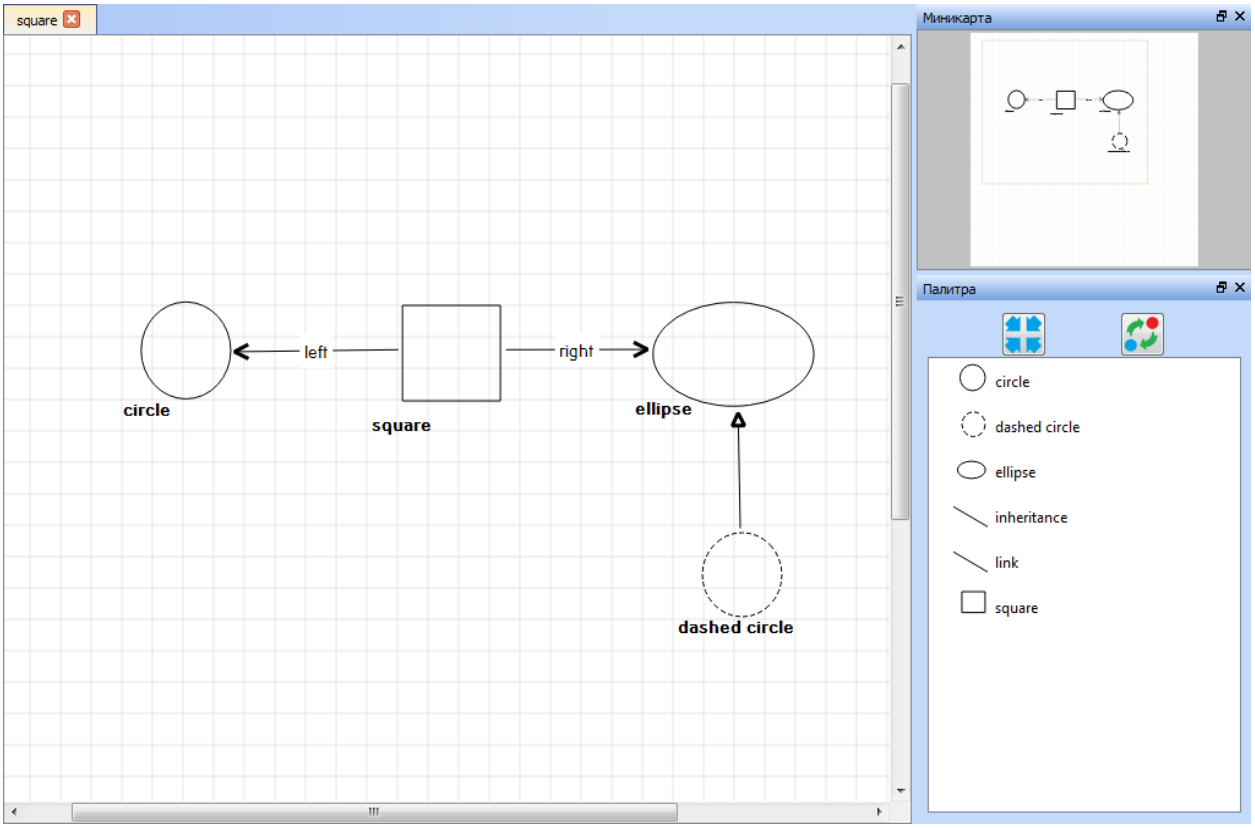

Рис 5. Визуальный язык простой графики.

В изображенном на рисунке визуальном языке имеются четыре сущности, представляющие простые геометрические фигуры: квадрат, эллипс и 2 круга, отличающихся типом границы. Также в визуальном языке присутствуют 2 типа связи: связь, отвечающая за позиционирование объектов относительно друг друга, и связь, показывающая то, что один

элемент находится внутри другого элемента. Модель на рисунке 5 задает фигуру, в центре которой расположен квадрат, слева от него круг, а справа эллипс, внутри которого находится пунктирный круг.

Предположим, мы создали простую модель, описанную выше, сохранили ее и решили удалить один из кругов, а именно сущность “пунктирный круг” из нашего визуального языка. В результате этого действия наша метамодель изменилась – в ней данный элемент отмечен, как удаленный. Мы получили другую версию визуального языка. Однако, несмотря на все произошедшие изменения, мы можем открыть модель, созданную нами ранее (см. Рис. 6). Как мы видим, удаленная сущность “пунктирный круг” отсутствует в палитре, однако она присутствует на диаграмме, и мы можем добавить ее в палитру и использовать далее в процессе моделирования через контекстное меню, выбрав пункт “Добавить элемент в палитру”.

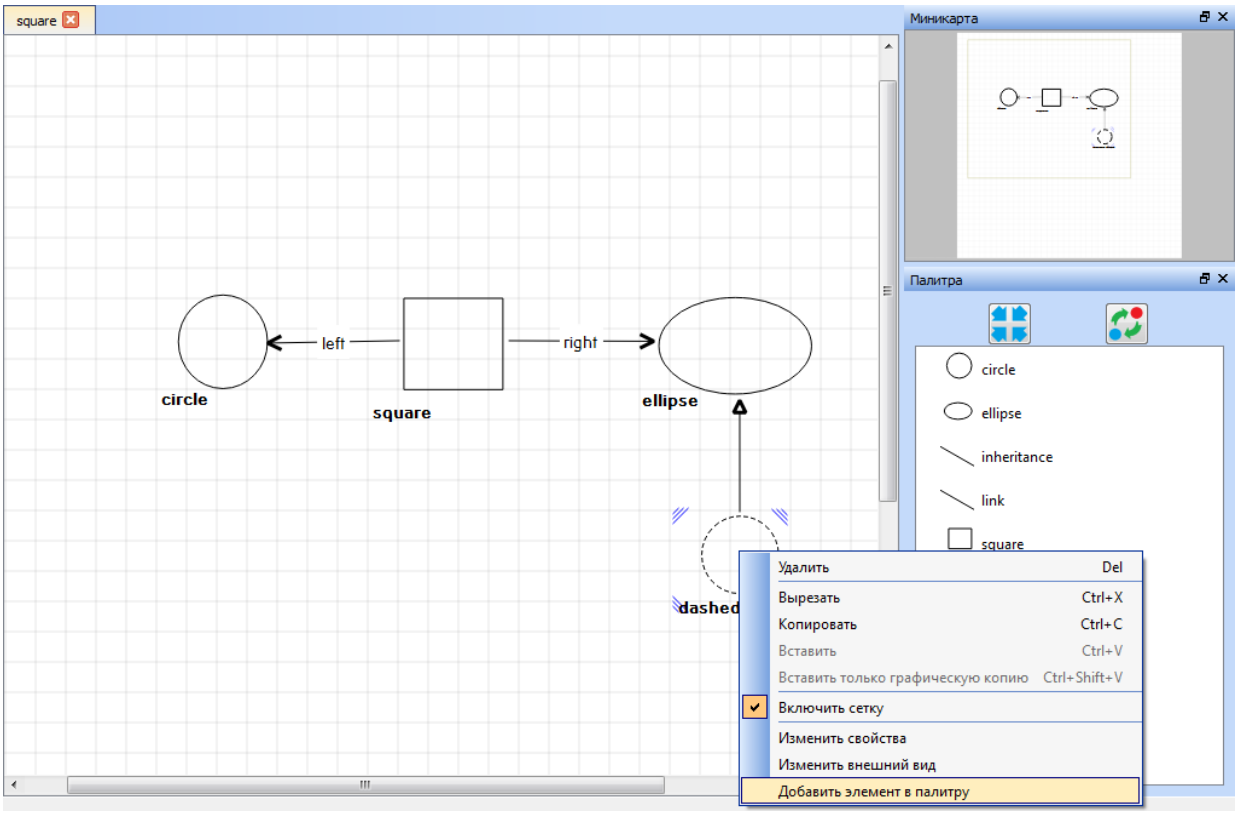

Рис 6. Открытие модели в измененном визуальном языке простой графики.

Рассмотрим теперь, каким образом происходит восстановление удаленного или переименованного свойства. Добавим сущности “square”

свойства “side”, задающее длину стороны квадрата, свойство “color” – цвет и свойство, отвечающее за то, содержится ли элемент внутри другого элемента “isContainer”. Потом удалим последнее свойство, создадим новое с таким же именем с помощью диалогового окна восстановления свойств и переименуем его в “container”. Если в дальнейшем мы снова попытаемся создать свойство “isContainer”, то сможем увидеть информацию о предыдущих двух наших свойствах в диалоговом окне восстановления свойств (см. Рис. 7).

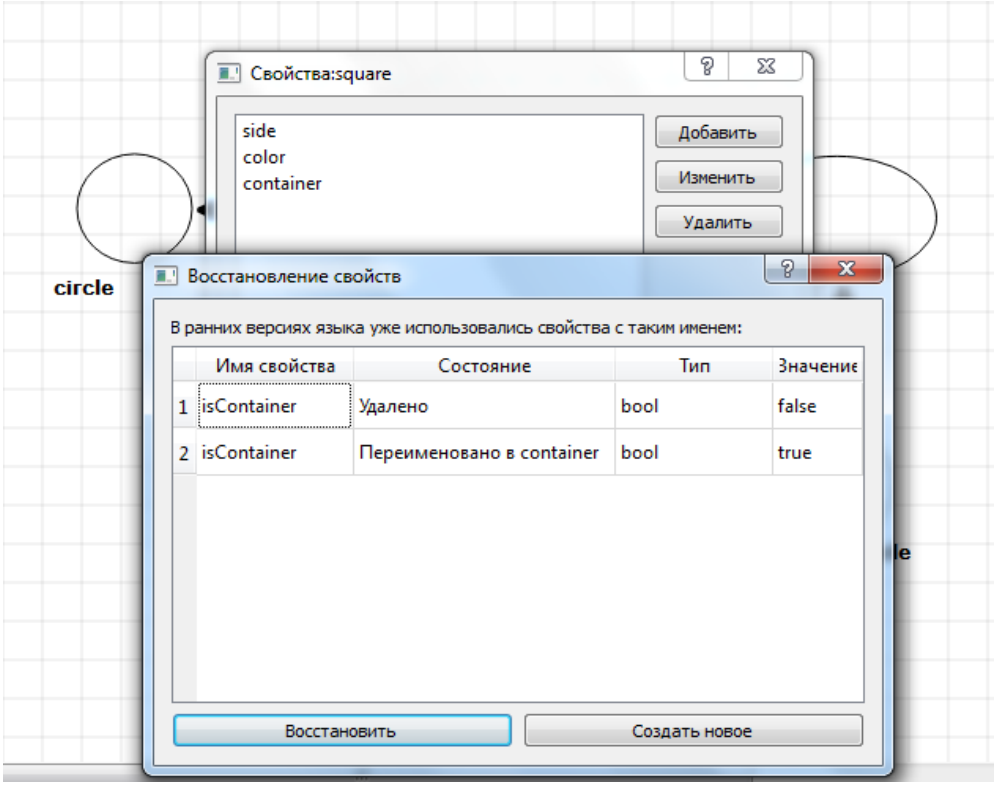

Рис 7. Диалоговое окно восстановления свойств.

Удалим теперь сущность “square” из нашего языка, кликнув по нему правой кнопкой в палитре и выбрав соответствующий пункт в появившемся контекстном меню. Если после этого мы попробуем добавить сущность с именем “square”, то увидим наш старый элемент и все его свойства в открывшемся диалоговом окне восстановления элемента (см. Рис. 8).

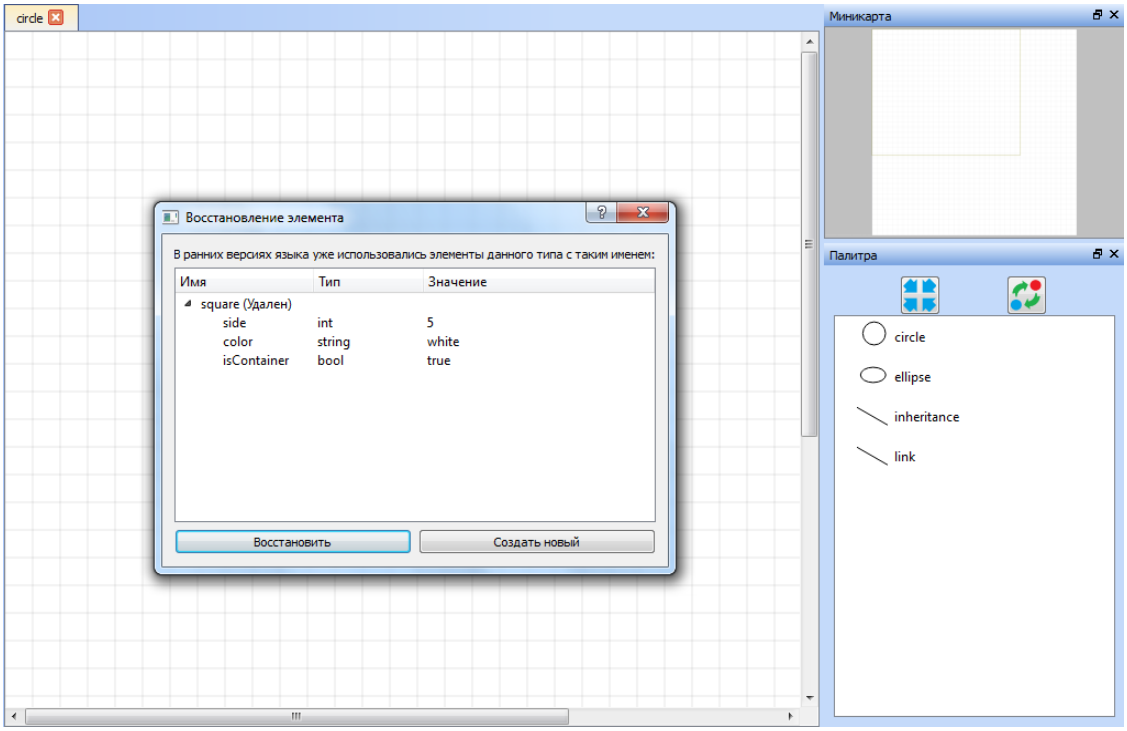

Рис 8. Диалоговое окно восстановления элемента.

7 Обсуждение

Обсудим другой подход, который рассматривался для решения поставленной задачи, и сравним его с выбранной методикой, описанной выше. Этот подход использует идею, похожую на реализованную в формате XMI (XML Metadata Interchange). XMI – это стандарт OMG (The Object Management Group) для обмена метаданными с помощью языка XML[11][12].

XMI может использоваться для любых метаданных, если их метамодель может быть выражена с помощью MOF (Meta-Object Facility, метаязык, на котором определена метамодель языка UML[2]). Чаще всего XMI применяется в качестве формата обмена UML-моделями, но может использоваться и для других языков. Формат XMI содержит описание не только метамодели языка, но и конкретной модели, т.е. он может использоваться в качестве формата сохранения в средствах визуального моделирования. Также он позволяет описывать разницу между двумя версиями объектной модели. Для этого используется элемент difference, содержащий операции Add, Delete и Replace, которые описывают множество различий MOF объектов. Каждой операции соответствует элемент с атрибутами: xmi.id, xmi.label, xmi.href и др. С их помощью указывается, к каким именно объектам модели необходимо применить данные операции.

7.1 Альтернативный вариант решения

В альтернативном подходе, аналогично формату XMI, предлагается хранить совместно с моделью метамодель, на основе которой она была создана. А при открытии модели сравнивать ее метамодель с текущей метамоделью языка. Существуют специализированные инструменты, позволяющие сравнивать XML-файлы, содержащие описания метамоделей, например: Microsoft's XML Diff and Patch Tool [13], xmldiff [14], DiffDog [15] и многие другие. Таким образом, путем сравнения метамоделей мы всегда сможем преобразовать новую метамодель и добиться поддержки старой

[2] Unified Modeling Language. URL: http://uml.org/ [дата просмотра: 17.05.2014]

модели. Однако минусом такого подхода является то, что мы не можем определить, какая из метамоделей была создана раньше, а какая позже. Мы всегда можем преобразовать одну в другую, даже если метамодели были созданы для разных визуальных языков моделирования. В результате происходит потеря информации об иерархии эволюции метамоделей.

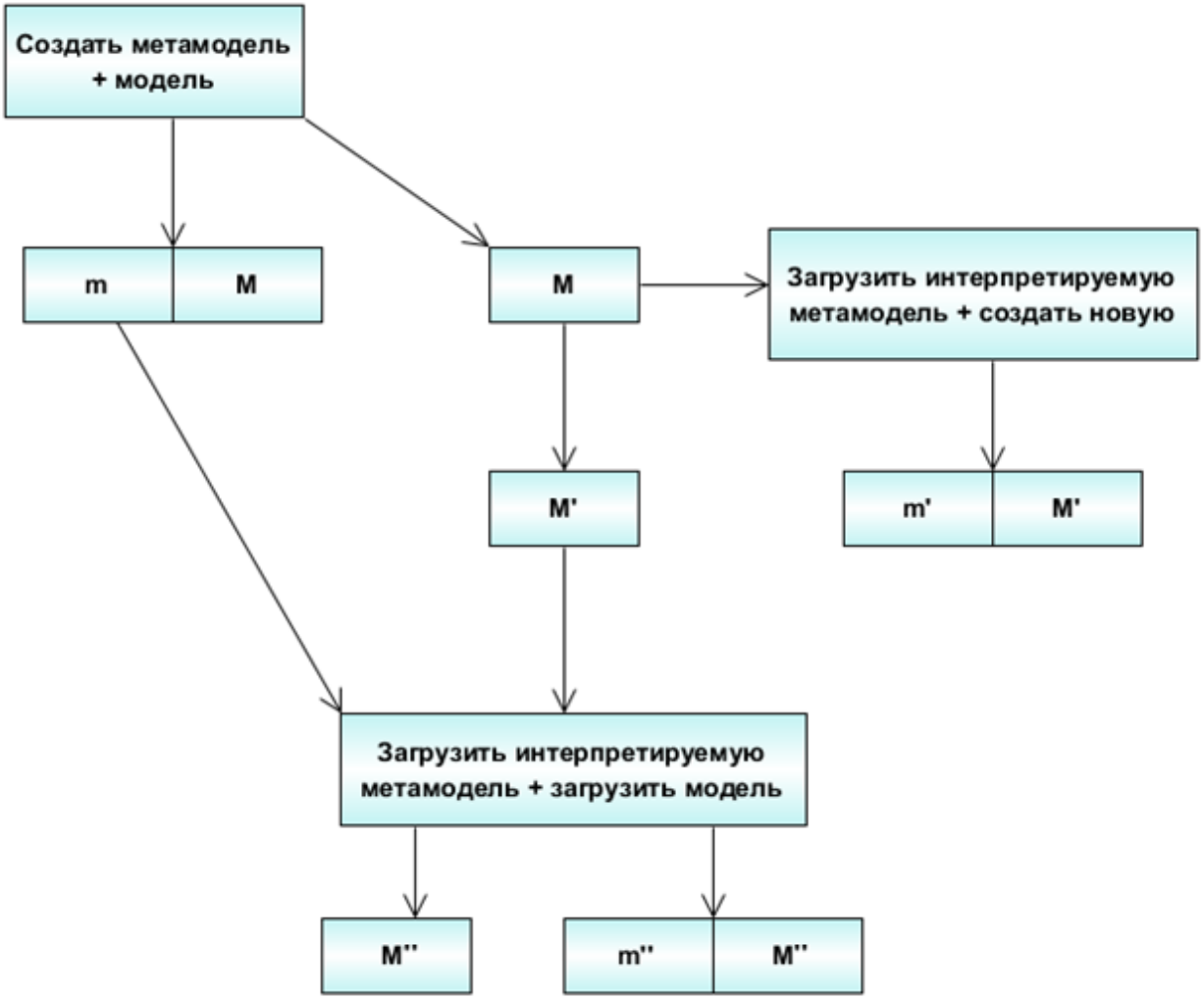

Рис 9. Альтернативный вариант архитектуры.

Рассмотрим подробнее данную идею на примере (см. Рис. 9). Пусть в режиме метамоделирования “на лету” мы создали предметно-ориентированный визуальный язык и сохранили полученную метамодель M и модель m. Далее мы загрузили интерпретируемую метамодель M и видоизменили наш язык: добавили новые элементы, удалили ненужные, изменили свойства и т.д. и сохранили новую модель m’ и полученную метамодель M’. Текущей версией метамодели стала M’. Если теперь мы в

режиме интерпретируемой метамодели загрузим метамодель M’ и попытаемся открыть модель m, созданную для ранней версии языка, то путем сравнения элементов и свойств метамоделей M и M’ мы сможем преобразовать текущую версию метамодели и добиться поддержки открытия старой модели при измененной метамодели.

В качестве решения данный подход не был выбран по той причине, что реализованное в итоге решение позволяет сохранить информацию об истории изменений в метамодели. Это помогает при разработке визуального языка, поскольку позволяет восстановить удаленные элементы и свойства непосредственно при работе с языком.

Заключение

В рамках данной дипломной работы были получены следующие результаты:

1. Разработана методика адаптации моделей визуального языка при изменении метамодели в процессе метамоделирования "на лету".
2. Реализовано средство для автоматической адаптации моделей под новые версии языка и поддержки эволюции языков при метамоделировании "на лету".
    a. Реализована возможность открытия модели, созданной на основе предыдущей версии языка.
    b. Реализована возможность восстановления удаленных / переименованных элементов языка и их свойств.
3. Произведена апробация предложенной методики и выполнено тестирование реализованного средства на примере простого DSVL-языка

Полученные результаты были представлены автором данной работы на конференции “СПИСОК-2014” [16].

Список литературы

1. Д.Кознов. Основы визуального моделирования. // Бином. Лаборатория Знаний, Интернет-Университет Информационных Технологий – 2008.
2. А.Павлинов, Д.Кознов, А.Перегудов, Д.Бугайченко, А.Казакова, Р.Чернятчик, Т.Фесенко, А.Иванов. О средствах разработки проблемно-ориентированных визуальных языков // Системное программирование. / Вып. 2, под ред. А.Н.Терехова и Д.Ю.Булычева. Спб.: Изд. СПбГУ, 2006. С. 121-147.
3. Steven Kelly, Juha-Pekka Tolvanen. Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Pr. – 2008.
4. А.Н.Терехов, Т.А.Брыксин, Ю.В.Литвинов и др. Архитектура среды визуального моделирования QReal. // Системное программирование. Вып. 4. СПб.: Изд-во СПбГУ. 2009, С. 171-196
5. А.И.Птахина. Интерпретация метамоделей в metaCASE-системе QReal, курсовая работа, СПбГУ, кафедра системного программирования, 2012. http://se.math.spbu.ru/SE/YearlyProjects/2012/YearlyProjects/2012/345/345_Ptakhina_report.pdf [дата просмотра: 17.05.2014]
6. А.И.Птахина. Разработка метамоделирования “на лету” в metaCASE-системе QReal, курсовая работа, СПбГУ, кафедра системного программирования,2013. http://se.math.spbu.ru/SE/YearlyProjects/2013/YearlyProjects/2013/445/445-Ptakhina-report.pdf [дата просмотра: 17.05.2014]
7. Domain-Specific Modeling with MetaEdit+. URL: http://www.metacase.com/ [дата просмотра: 17.05.2014]
8. Е.Б.Замятина, Л.Н.Лядова, А.О. Сухов. Мультиязыковое моделирование с использованием DSM платформы MetaLanguage / Информатизация и связь. – 2013. – № 5. – С. 11-14.
9. Л.Н.Лядова, А.О. Сухов. Языковой инструментарий системы MetaLanguage, Математика программных систем: межвуз. сб. науч. ст. / М34 Перм. гос. ун-т. – Пермь, 2008. С. 40-51.

10. А.О.Сухов. Среда разработки визуальных предметно-ориентированных языков моделирования, Математика программных систем: межвуз. сб. науч. ст. / М34 Перм. гос. ун-т. – Пермь, 2008. С. 84-94

11. XML Metadata Interchange (XMI). URL: http://www.omg.org/spec/XMI/ [дата просмотра: 17.05.2014]

12. Hongji Yang. Advances in UML and XML-based Software Evolution. // Idea Group Inc (IGI) – 2005.

13. Using the XML Diff and Patch Tool. URL: http://msdn.microsoft.com/en-us/library/aa302294.aspx [дата просмотра: 17.05.2014]

14. Xmldiff. URL: http://www.logilab.org/project/xmldiff [дата просмотра: 17.05.2014]

15. DiffDog – Altova. URL: http://www.altova.com/diffdog.html [дата просмотра: 17.05.2014]

16. А.И.Птахина. “Эволюция языков при метамоделировании “на лету” в DSM-платформе QReal” // СПИСОК-2014: Материалы всероссийской научной конференции по проблемам информатики. 23-25 апреля 2014г., Санкт-Петербург. (Принята к публикации).