**Санкт-Петербургский Государственный Университет**

**Математико-Механический Факультет**

# «Сравнение способов получения редакторов по метамодели и скорости их работы»

Курсовая работа

Тарасовой Полины Максимовны, студентки 244 гр.

Научный руководитель: ст.преп. Литвинов Ю.В.

Санкт-Петербург

2015

# Введение

С помощью визуальных языков программирования можно создавать приложение путем манипулирования некоторыми графическими объектами, то есть без непосредственного написания кода. Такой подход упрощает разработку и она становится более доступной для масс. То есть приложение могут создавать и модифицировать люди, не имеющие опыта и знаний о синтаксисе и об особенностях языка программирования, на котором строится используемый визуальный язык.

При таком подходе программа изображается как множество диаграмм. Тогда определить полную модель приложения можно путем создания комбинаций различных типов графических объектов.

Такой способ визуализации имеет ряд преимуществ:

- интуитивно понятное графическое изображение облегчает процесс анализа и проектирования, а также помогает сфокусироваться на деталях
- повышается эффективность взаимодействия между участниками разработки
- облегчается восприятие сложных систем и ориентирование в них
- при узкой специализации модели такая визуализация упрощает ее проектирование

Диаграммы, в отличие от программного кода, ближе к предметной области и доступны более широкому кругу заинтересованных лиц, так, к построению системы могут быть привлечены не только профессиональные разработчики, но и эксперты в данной предметной отрасли.

Для автоматизации процесса разработки информационной системы используются специальные средства -- CASE-системы, с помощью которых можно моделировать систему на каком-либо визуальном языке программирования. CASE-системы способствуют сокращению времени разработки и проектирования информационных систем, упрощают этот процесс и повышают качество программного обеспечения.

Было бы удобно, при работе в какой-то конкретной предметной области оперировать непосредственно понятиями, определенными в этой области, а не их реализацией в коде. То есть нужен язык, ориентированный на решение определенного круга задач. Для таких целей используют специальные предметно-ориентированные языки (DSL), которые повышают уровень абстракции вне написания кода.

Но для каждой новой предметной области необходим новый визуальный язык. Таким образом возникает потребность в инструменте, который бы позволил автоматизировать процесс разработки новых визуальных языков. Для этого используют metaCASE системы. Такой подход называется предметно-ориентированным моделированием(DSM).

Главное достоинство DSM-платформы -- возможность интеграции в одной системе сразу нескольких DSL. Это часто требуется, так как для отдельных задач удобно использовать совершенно разные языки.

Язык описания предметно-ориентированного языка называется метаязыком. Соответственно, модели, которые описывают создаваемый визуальный язык, построенные с помощью метаязыка, называются метамоделями. В отличие от CASE-систем, metaCASE позволяют пользователю изменять метамодель. Также, с их помощью можно быстро создавать редакторы своих собственных визуальных языков.

На данный момент существует множество средств для разработки графических редакторов DSL с опцией определения собственных графических нотаций: MetaEdit+, MetaLanguage, QReal и др.

В QReal существует три различных способа получения редакторов по метамодели: генеративные (с использованием промежуточного xml-представления и без него) и интерпретативный (позволяющий вносить изменения в получившийся редактор «на лету»). Сложно сказать, какой из способов является наиболее предпочтительным. Можно лишь отметить, что если требуется возможность изменять метамодель прямо в процессе создания визуального языка, то наиболее предпочтительным окажется интерпретативный способ, а если требуется наиболее высокая скорость работы, то удобнее будет генеративный способ. Соответственно, хочется найти и определить оптимальный способ получения визуальных редакторов.

Поэтому целью нашей работы являлось сравнение визуальных редакторов, полученных различными способами, но по одной метамодели, и выявление различий в скорости их работы, а именно, насколько велики различия при выборе какого-либо из способов по отношению к остальным.

Соответственно, необходимо было выполнить следующие задачи:

- Сравнить функциональность редакторов, полученных различными способами
- Поддержать измерение времени в тестовой утилите

Данная работа будет интересна и полезна тем, кто занимается визуальным метамоделированием и кого интересуют различные способы и методы разработки редакторов визуальных языков.

# Обзор существующих популярных DSM-платформ

**MetaEdit+**

MetaEdit+ – полнофункциональная среда разработки систем, поддерживающая создание и использование предметно-ориентированных языков. MetaEdit+ позволяет применять несколько встроенных языков моделирования, обеспечивает одновременный доступ к среде для нескольких пользователей, а также предоставляет возможность создавать диаграммы, матрицы и таблицы. С помощью мультиязыковых средств возможно переиспользовать конструкции существующих в инструментарии языков.

Одной из подсистем MetaEdit+ является MetaEdit+ Workbench. Это мощный ресурс для создания генераторов и языков моделирования.

Соответственно, кроме возможности создания предметно-ориентированного языка, разработчику доступно и CASE-средство, в котором этот язык можно использовать. Также, MetaEdit+ является многоплатформенной средой.

Инструментальные средства MetaEdit+

- редактор -- позволяет создавать, изменять, удалять модели и устанавливать взаимосвязи между различными моделями;
- репозиторий -- хранит информацию о доступных языках моделирования; о моделях, и всех их внутренних компонентах;
- генераторы - осуществляют проверку правильности созданных моделей и позволяют сгенерировать документацию или программный код на их основе.

Разработчик может вносить изменения в имеющиеся генераторы или создавать новые. Кроме использования встроенных генераторов кода, MetaEdit+ также позволяет создавать генераторы для собственных языков программирования и моделирования.

MetaEdit+ дает возможность импортировать и экспортировать метамодели в формат MXT, базирующийся на распространенном стандарте XML. Использование своего собственного формата файлов сказывается на открытости данной технологии.

Метамоделирование в MetaEdit+

MetaEdit+ делает возможным для разработчика самому определять языки моделирования и генераторы для них. Разработчик создает метамодель конкретной предметной области, используя язык метамоделирования, определяя основные компоненты, и их характеристические составляющие и различные правила интеграции нескольких языков в одной системе. Для

построении метамодели в данном инструментарии используется язык метамоделирования GOPRR(Graph, Object,Property,Relation,Role).

Создание метамодели начинается с построения GOPRR-модели, затем для построения модели предметной области она может быть открыта в MetaEdit+ Workbench. Если при этом загрузить метамодель в систему, то она может использоваться в качестве метаязыка. Посредством интерпретативного подхода MetaEdit+ возможно производить динамическое изменение метамоделей.

Однако, стоит отметить, что в MetaEdit+ отсутствует полнофункциональная среда программирования, которая бы предоставляла возможность вносить изменения в сгенерированный системой код «вручную»; и нет возможности трансформации визуальных моделей.

### MetaLanguage

MetaLanguage – это языковой инструментарий, используемый для создания визуальных динамически настраиваемых предметно-ориентированных языков моделирования.

Инструментальные средства MetaLanguage:

- графический редактор – используется для создания, изменения, удаления моделей и позволяет устанавливать взаимосвязей между различными моделями, описание которых выполнено посредством графических нотаций;
- браузер объектов – инструментальное средство, используемое для редактирования и просмотра информации, содержащейся в репозитории;
- репозиторий – хранилище, которое содержит информацию о компонентах системы, может работать в многопользовательском режиме;
- валидатор – совершает проверку соответствия модели ограничениям, заданным пользователем;
- генератор – дает возможность на основе имеющихся моделей сгенерировать XML-представление или документацию модели. В XML-файле будет содержаться вся информацию о модели, ее внутреннем представлении и о зависимостях, в которых она принимает участие. Документация модели содержит в себе: информацию о разработчиках, название модели и о ее графическом представлении с ссылками на описание отдельных ее частей.

При открытии метамодели все модели, созданные с помощью этой метамодели, сущности, отношения метамодели и соответствующих моделей загружаются из репозитория. При удалении метамодели, также будут удалены

и все модели, базирующиеся на данной. При изменении метамодели модифицируются и зависимые от нее модели, что позволяет поддерживать согласованность системы.
При удалении сущности из метамодели выполняется поиск всех экземпляров данной сущности во всех моделях, которые были созданы с помощью удаляемой. Затем для каждого экземпляра удаляемой сущности происходит удаление всех отношений, связанных с данной единицей. Далее вся эта информация удаляется и из репозитория.
Подход, применяемый MetaLanguage, предоставляет возможность производить адаптацию моделей под новые версии языка и таким образом обеспечивает согласованность системы. Тем не менее теряется возможность восстановления удаленных сущностей, так как вся информация о них полностью удаляется из репозитория.

Данная системы позволяет строить модели, которые довольно подробно и точно описывают конкретную предметную область. И для различных уровней детализации описаний используются аналогичные конструкции.
В силу использования одного и того же инструментария при работе с моделями и метамоделями, возможен и итеративный процесс создания модели: после создания некоторого языка его можно использовать в качестве метаязыка для создания другого языка, который также может быть использован как метаязык и т.п. Следовательно, разработчик получает достаточно мощный языковой инструментарий для создания визуальных динамически настраиваемых DSL.

## QReal

QReal — metaCASE-система для создания визуальных редакторов, предоставляющая средства для автоматической генерации кода произвольных визуальных редакторов посредством описания их метамоделей.

Генеративный подход к созданию редакторов

Редактор инкапсулирует в себе информацию о множестве объектов, которые допустимы на диаграммах данного типа, также он должен быть способен к корректной интерпретации хранящихся в репозитории значений атрибутов элементов. Более того, редактор должен понимать
логические правила расположения элементов на соответствующих типах диаграмм.

Создание набора таких редакторов кодированием их «вручную» не самый оптимальный способ из-за множества причин(слабая расширяемость, опасная масштабируемость, усложнение архитектуры CASE-пакета и т.д.).

Поэтому был реализован следующий подход к автоматическому созданию графических редакторов:

- с помощью анализа типовых редакторов выделяется базовая функциональность абстрактного редактора, кодируемая «вручную» на C++;
- специфика метамоделей нужных диаграмм представляется с помощью с XML-описания;
- по ним генерируется код на C++, который вместе с базовой функциональностью предоставляет полную функциональность описанных диаграмм.

### Генерация редакторов по метамодели

Процесс генерации редакторов в QReal организован следующим образом: описания метамоделей подаются на вход утилите-генератору, осуществляющей анализ предоставленных XML-файлов, далее для каждой сущности определяется список свойств, в котором также указывается их тип, список элементов-родителей (свойства и логика родительских элементов могут быть наследуемы), отображаемое имя сущности и т.д. Дополнительно сущности имеют секцию описания их графического представления. Так обеспечивается форматирование значений, подставляемых из репозитория. В ходе разбора соответствующих секций XML-описаний генератор заменяет теги параметризации соответствующим C++ кодом для получения нужных значений атрибутов элемента, при этом теги форматирования текста сохраняются.

После разбора XML-описаний метамоделей производятся проверки семантической корректности описаний редакторов.
В заключительной фазе создаются:
• классы на языке C++;
• файлы, содержащие описания графического представления элементов;
• внутренние средства, предоставляющие доступ к значениям атрибутов элементов в репозитории из различных модулей проекта.
• дополнительный код, необходимый для присоединения генерируемых к проекту генерируемых классов.

### Метаредактор

Метаредактор (редактор графических свойств) предоставляет пользователям визуальный язык, который является подмножеством стандартного языка, описывающий все метамодели. Практика показывает, что достаточно реализовать лишь небольшой набор элементов MOF:
• классы
• ассоциации между ними

• атрибуты классов

Этих сущностей достаточно для описания метамодели всех возможных редакторов, поддерживаемых CASE-средством.

# Реализация

В QReal существует три различных способа получения редакторов по метамодели: генеративные (с использованием промежуточного xml-представления и без него) и интерпретативный (позволяющий вносить изменения в получившийся редактор «на лету»). Для сравнения редакторов была реализована тестовая утилита. Однако, в силу продолжительной неиспользуемости, она утратила работоспособность. Одна из подзадач данной курсовой работы -- исправление нерабочих участков кода данной утилиты, чем и занималась Храмышкина Ю.С.

Также стояла задача внедрить в утилиту измерение времени работы методов  для разных подходов получения визуальных редакторов. Для этого был написан блок кода, который исполняется в момент вызова методов из тестовой утилиты. Он содержит в себе вычисление дисперсии, среднеквадратичного отклонения и матожидания.

В этом блоке интересующий метод вызывается двадцать раз, каждое значение записывается в массив, и происходит суммирование этих величин. Далее идет цикл, в котором подсчитывается дисперсия для каждого элемента массива. Затем находится ее среднее значение, а по нему и общее среднеквадратичное отклонение времени работы метода. После, данные передаются для дальнейшей обработки, подробности которой содержатся в курсовой работа Храмышкиной Ю. С.

Посредством этих значений и устанавливаются величины, определяющие разницу времени при использовании различных подходов к созданию редакторов.

Одним из результатов работы утилиты являются две таблицы, отображающие насколько корректно работают методы и время их работы. Одна из таблиц является результатом сравнения QReal XML-Compiler(QRXC) и Interpreter, другая -- QRXC и QReal Metamodel Compiler(QRMC).

Эта табличная информация способствовала обнаружению несоответствий в результатах работы подходов. Так были определены значительные расхождения при генерации редактора посредством QRMC. И появилась новая задача, а именно: исправить методы, работающие некорректно.

//QRMC использует частично шаблонную реализацию. Получаемые //данные из метамодели помещаются в заранее отмеченные места в //программном коде.

Процесс генерации кода на C++ посредством QRMC устроен следующим образом: в отличие от компилятора метамоделей, который использует еще и промежуточное XML-представление, при данном подходе данные из метамодели помещаются в заранее отмеченные места в файлах, реализациия которых помимо этих участков прописана по умолчанию.

В силу продолжительной невостребованности данной библиотеки множество методов и классов отрабатывали в нынешней версии QReal некорректно или их реализация отсутствовала вовсе. Также, данные получаемые из метамодели были неполные и потребовалось внести некоторые дополнительные правки для разрешения данной проблемы.

Помимо небольших корректировок, были внесены и существенные изменения в программный код. Была восстановлена работоспособность многих методов, например: getEnumValues(), getPropetyNames(), initDescriptionMap().

Также, был изменен и репозиторий для сохранения файлов, полученных при генерации посредством данного компилятора метамоделей. Необходимость этой модификации возникла, потому что в QReal происходили значительные изменения, которые в течение длительного времени никак не поддерживались в QRMC.

Кроме того, были внесены изменения и в имена сконфигурированных модулей: был добавлен суффикс, отвечающий за отладочную конфигурацию.

После всех этих модификаций был протестирован ряд метамоделей и в результате получилось, что генеративный режим быстрее, чем интерпретативный в среднем в 500 раз. Однако это не умаляет значимости интерпретативного подхода. В случае, когда необходимо вносить изменения в редактор при непосредственной работе с ним (то есть без перегенерации кода) следует использовать этот подход. А генеративный подход стоит использовать, если изменения такого рода не предполагаются и требуется высокая скорость работы.

## Результаты

- Выявлены и исправлены несоответствия между редакторами, полученными различными способами
- Измерено время работы основных методов визуальных редакторов, а также проведен анализ и сделаны выводы о том, что генеративный режим быстрее интерпретативного приблизительно в 500 раз

## Список литературы

[1] Domain-Specific Modeling with MetaEdit+. URL: http://www.metacase.com/ (дата обращения: 10.05.2015)

[2] Домашняя страница проекта QReal на GitHub, URL: https://github.com/qreal/qreal (дата обращения: 20.02.2015)

[3] Терехов А.Н., Брыксин Т.А., Литвинов Ю. В., Смирнов К.К., Никандров Г.А., Иванов В.Ю., Такун Е.И. Архитектура среды визуального моделирования QReal. Системное программирование. Т. 4. СПб.: Изд-во СПбГУ. 2000. С. 165–169.

[4] А.О.Сухов. Сравнение систем разработки визуальных предметно-ориентированных языков / Математика программных систем: межвуз. сб. науч. ст. – Пермь: Изд-во Перм. гос. нац. исслед. ун-та, 2012. – Вып. 9. – С. 84–101.

[5] А.И.Птахина, Интерпретация метамоделей в metaCASE-системе QReal, курсовая работа, СПбГУ, кафедра системного программирования, 2012. URL: http://se.math.spbu.ru/SE/YearlyProjects/2012/list (дата обращения: 08.02.2015)

[6] А.И.Птахина, Разработка метамоделирования “на лету” в metaCASE-системе QReal , курсовая работа, СПбГУ, кафедра системного программирования, 2013. URL: http://se.math.spbu.ru/SE/YearlyProjects/2013/list (дата обращения: 08.02.2015)

[7] Ю.В.Литвинов, дис, Разработка визуальных предметно-ориентированных языков (не опубликовано).

[8] Ю.С. Храмышкина, название?, курсовая работа (не опубликовано)