Санкт-Петербургский Государственный Университет

Математико-Механический Факультет

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

Курсовая работа

Храмышкиной Юлии Сергеевны, студентки 244 гр.

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

Санкт-Петербург

2015

Оглавление

Введение

Существуют различные подходы к разработке программ и приложений, одним из которых является визуальное программирование. В последнее время ему уделяется все больше внимания, потому что визуальное программирование позволяет моделировать ПО с разных точек зрения.

Модели описывают отдельные аспекты ПО, и благодаря этому можно сосредоточить всё внимание на разработке каких-либо отдельных свойств продукта, а не рассматривать всё многообразие предметной области целиком. Поэтому визуальное программирование позволяет упростить разработку и сделать ее более наглядной, а также этот подход помогает избежать различных ошибок во время написания кода. В частности, при работе в команде оно помогает быстро достичь взаимопонимания между разработчиками.

Для того, чтобы реализовать такой подход, требуются специализированные средства, которые называются CASE-системами. Их используют как инструменты для анализа, разработки, проектирования программного обеспечения, а также с их помощью происходит генерация необходимого кода.

В частности, во время разработки нередко требуется создание множества различных моделей для конкретной предметной области, поэтому возникает необходимость иметь специальный язык, который бы упрощал разработку других языков, позволяющих описывать создаваемые модели. Такой язык должен содержать описание всех необходимых при моделировании абстракций.

Для упрощения создания CASE-систем существуют metaCASE-системы, которые позволяют автоматически генерировать CASE-системы по их формальному описанию. Они нужны для создания предметно-ориентированных языков (DSL), благодаря которым можно решать специфичные задачи для конкретной предметной области. Например, в них можно быстро создавать редакторы своих собственных визуальных языков.

Язык, с помощью которого создаются другие языки моделирования, называется метаязыком. Он строится на основе модели, в которой содержится описание всех абстракций, необходимых при моделировании. Модель языка моделирования называется метамоделью.

В данной работе рассматривается metaCASE-система QReal, с помощью которой можно создавать специализированные среды визуального программирования. QReal позволяет автоматически генерировать код произвольных визуальных редакторов по описаниям их метамоделей.

В QReal существует три различных способа получения визуальных редакторов по метамодели(Рис.1): генеративные (с использованием промежуточного xml-представления и без него) и интерпретативный (позволяющий вносить изменения в получившийся редактор «на лету») [5].

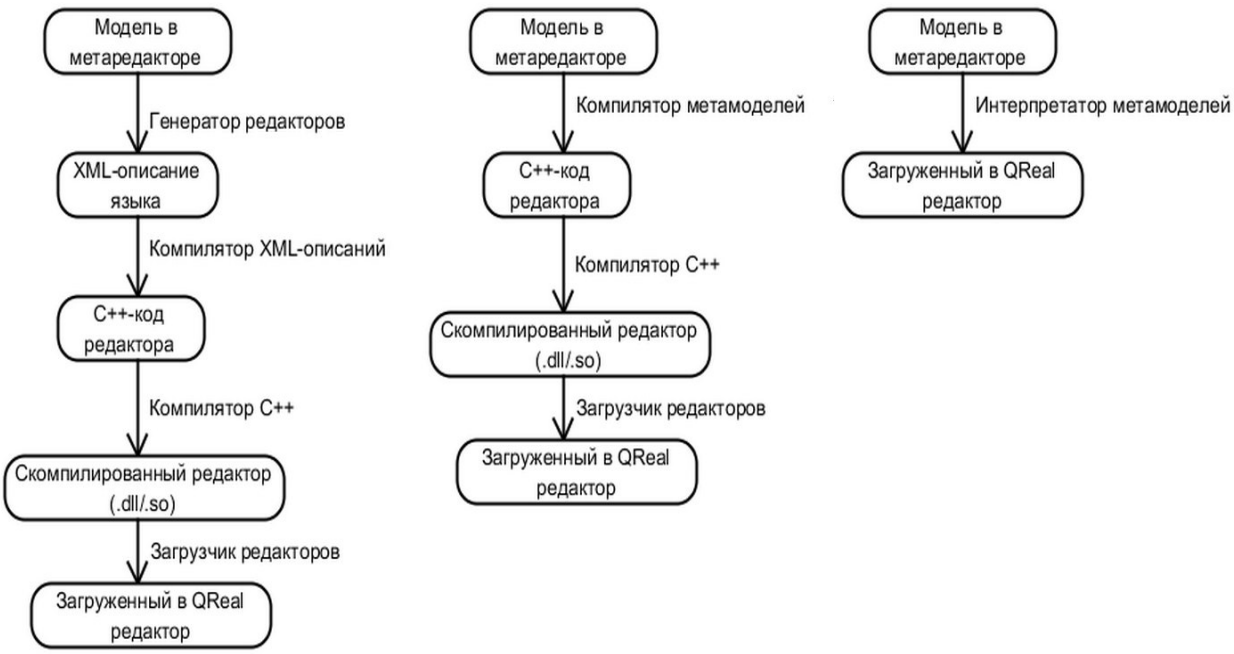

Рис 1. Способы получения визуальных редакторов

Таким образом, с помощью созданной в метаредакторе метамодели можно получить визуальный редактор путем генерации исходного кода редактора, также можно сначала сгенерировать XML-описание, а затем по нему

получить исходный код редактора, либо обойтись без генерации и открыть метамодель в интерпретаторе метамоделей.

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 высокая скорость работы, то генеративный будет предпочтительнее. Однако неизвестно, насколько один способ быстрее другого.

Стоит отметить, что ранее существовала утилита, способная сравнивать визуальные редакторы, полученные различными путями, но за время, прошедшее с момента, когда тестовая утилита была написана, интерфейс редакторов успел измениться, так что код стал несовместим с основной рабочей версией. Также потребовалась поддержка измерения времени работы методов в тестовой утилите. В частности, поддержка измерения времени была осуществлена Тарасовой П.М [6].

Таким образом, целью работы является 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 методов по отношению к остальным.

Поэтому были поставлены следующие задачи:

- Восстановить работоспособность утилиты для сравнения визуальных редакторов
- Сравнить функциональность редакторов, полученных различными способами
- Измерить время работы основных методов визуальных редакторов

1. Обзор существующих DSM-платформ

Существует множество различных DSM-платформ, которые позволяют использовать языки моделирования в процессе разработки информационной системы, специально созданные для той предметной области, в которой будет работать система.

1.1 MetaEdit+

MetaEdit+ – полнофункциональная среда разработки систем, с поддержкой возможности использования собственных языков моделирования, генераторов кода и документации, созданных с помощью MetaEdit+ Workbench. MetaEdit+ Workbench – средство создания языков моделирования и генераторов, предоставляющее простой, но мощный инструмент метамоделирования и инструмент для проектирования модельного языка [2].

Кроме возможностей создавать предметно-ориентированные языки разработчик получает CASE-средство, в котором можно их использовать.В MetaEdit+ пользователи могут работать одновременно сразу с несколькими проектами, причем в каждом из них может быть несколько моделей. Проекты могут содержать описание языков моделирования, а также модели, созданные с помощью них.

В этой среде разработчику предоставляется большой набор инструментальных средств, предназначенных для работы с языками моделирования (редакторы, репозиторий, генераторы и др.). Редакторы нужны для создания, изменения, удаления моделей и установления взаимосвязи между различными моделями.

Хранящаяся в репозитории информация используется для генерации кода и документации. С помощью генераторов MetaEdit+ можно проверить корректность созданных моделей и сгенерировать на их основе код или документацию. Также есть возможности для определения правил

трансформации моделей в код, MetaEdit+ позволяет импортировать и экспортировать метамодели в формат MXT (файл MetaEdit+ XML Types).

При построении метамодели в MetaEdit+ используется язык метамоделирования GOPRR (Graph, Object, Property, Relation, Role). MetaEdit+ позволяет без перезапуска приложения изменять метамодели и модели, созданные с их помощью. Так, пользователь может вносить изменения в описание метамодели, вновь импортировать ее в MetaEdit+ Workbench и продолжать работу над моделью, при этом DSM-платформа сама произведет все изменения в моделях, которые необходимы.

Таким образом, MetaEdit+ интерпретирует метамодели, а также поддерживает совместное изменение метамоделей и моделей.

1.2 Visual Studio Visualization and Modeling SDK

Visual Studio Visualization and Modeling SDK (бывш.DSL Tools) позволяет создавать собственные визуальные языки моделирования и строить для них графические редакторы [1].

В этой среде генераторы кода на основе определенной пользователем модели строят по указанному шаблону код ее реализации. При создании нового DSL автоматически создаются проекты, в которых хранятся различные артефакты метамодели создаваемого DSL и настройки графического редактора. Спецификации метамодели и редактора подаются на вход генератору. После их компиляции запускается новый экземпляр MS Visual Studio, где в качестве языка разработки используется созданный DSL. Таким образом, создаваемый DSL используется исключительно как составная часть MS Visual Studio, вне этой среды программирования ни DSL, ни редактор использовать нельзя. Также в этой среде отсутствует возможность динамического изменения метамоделей [2].

1.3 Eclipse Graphical Modeling Framework

Eclipse Graphical Modeling Framework (GMF) -- технология разработки, созданная на базе среды Eclipse. GMF интегрирует две известные Eclipse-библиотеки – Eclipse Modeling Framework (EMF) и Graphical Editing Framework (GEF). [Сухов] Архитектура DSL, созданного с применением GMF,

построена на основе MVC-шаблона. Для создания уровней представления и контроллеров используется технология GEF, для создания моделей – технология EMF.

GEF состоит из двух частей: модуля OED и базовой библиотеки GEF. Графический редактор, созданный с помощью OED, визуализирует экземпляры Java-классов, описывающих объекты предметной области. Для синхронизации классов и визуальных диаграмм существует «прослойка» (Controller), который создается с помощью библиотеки GEF. Изменение модели происходит с помощью определенного набора команд, а не напрямую. Эти команды одновременно вносят изменения и в классы, и в визуальные диаграммы.

Можно создавать визуальные DSL с помощью GEF на базе любых моделей, но самая высокая эффективность достигается при интеграции технологии GEF с EMF. Технология EMF предназначена для проектирования приложений, которые используют модели бизнес-процессов, имеющих сложную структуру. Создание подобных приложений производится с помощью средств генерации EMF по модели. Кроме того, разработчик может расширять функциональность «вручную». Изменения будут сохранены и при последующих генерациях кода. Однако в EMF отсутствует возможность динамического изменения метамоделей.

Процесс создания DSL с помощью технологии Eclipse GMF состоит из следующих этапов:

1. Описание доменной модели (абстрактного синтаксиса)

2. Разработка графической модели (конкретного синтаксиса)

3. Разработка модели инструментов

4. Задание модели соответствия. Все предыдущие модели никак не связаны друг с другом. В них хранятся объекты, которые будут использованы при построении графического редактора, а то, как именно они будут использованы, определяется в модели соответствия.

5. Создание модели генератора, по которой производится генерация графического редактора для DSL. Данная модель является промежуточным представлением получаемого редактора. Она автоматически генерируется по модели соответствия и, возможно, дополняется «вручную».

1.4 MetaLanguage

MetaLanguage — среда для разработки визуальных предметно-ориентированных языков моделирования.

Среда разработки MetaLanguage состоит из: графического редактора, который нужен для различных операций с моделями, в частности, для установления взаимосвязей между ними; средства для просмотра и редактирования информации, хранящейся в репозитарии; репозитария – хранилища, содержащего информацию о метамоделях, моделях, сущностях и др.; средства для проверки соответствия модели заданным ограничениям; а также генератора, который на основе имеющихся моделей генерирует XML-файл, содержащий информацию о модели и документацию.

При открытии метамодели из репозитария загружаются все модели, созданные с помощью нее, все сущности и отношения. 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 [3].

Для работы с моделями и метамоделями используется одинаковый инструментарий. Процесс создания модели является итеративным: после создания некоторого языка его можно использовать как метаязык для создания другого, который также можно использовать в качестве метаязыка.

Таким образом, MetaLanguage является инструментом для создания визуальных динамически настраиваемых предметно-ориентированных языков моделирования.

MetaLanguage использует интерпретативный подход для описания моделей различных уровней абстракции. Процесс создания нового предметно-ориентированного языка начинается с построения метамодели, после этого происходит построение модели. Далее модель проходит валидацию. Полученную модель можно сохранить в виде XML-файла, содержащего все основные данные о модели. Помимо этого, можно сгенерировать на основе модели документацию. Важно отметить, что изменение метамодели может иметь место в любой момент во время создания

визуального языка, причем все изменения система автоматически применит к моделям, созданным с помощью этой метамодели.

1.5 QReal

MetaCASE-система QReal -- проект научно-исследовательской группы кафедры системного программирования СПбГУ. QReal позволяет автоматически генерировать код произвольных визуальных редакторов по описаниям их метамоделей [4].

Основой системы QReal является абстрактное ядро, реализующее общую для всех редакторов и элементов диаграмм функциональность, и подключаемые модули – наследуемые от ядра классы, реализующие специфику конкретных редакторов. Они автоматически генерируются по XML-описаниям метамоделей создаваемых языков. QReal позволяет описывать метамодели в текстовом виде: на языке, являющимся расширением XML, а также есть возможность графического задания метамоделей создаваемых языков и графического представления их элементов на диаграммах (Рис.2).

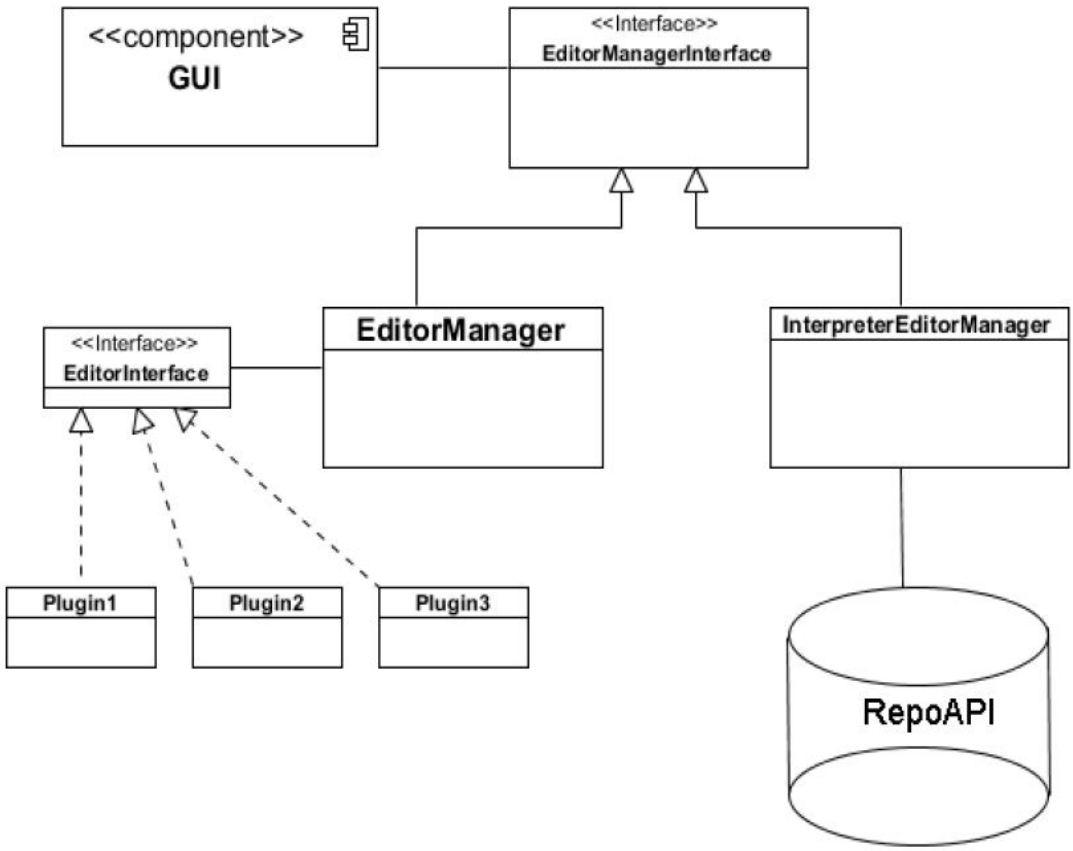

Рис 2. Устройство metaCASE-системы QReal

MetaCASE-система QReal устроена следующим образом: существует интерфейс, который отвечает за всю информацию о языке: о сущностях, связях и взаимодействиях между ними в метамодели. Его реализует класс интерпретатора, который для получения информации имеет непосредственный доступ к репозиторию с метамоделью, причем интерпретация моделей происходит непосредственно во время работы приложения. Этот класс отвечает за интерпретативный режим.

Также этот интерфейс реализует класс EditorManager, в котором содержится набор редакторов. Каждый из них реализует интерфейс EditorInterface, отвечающий за отдельную вкладку в палитре для каждого визуального редактора. В свою очередь, этот класс отвечает за генеративный режим. Визуальные редакторы генерируются из файла метамодели в код на C++ во время компиляции приложения, для этого как раз используется EditorInterface, причем генерация обязана завершиться до начала работы приложения.

Ранее, была реализована тестовая утилита для сравнения визуальных редакторов, полученных с помощью различных подходов. Она проверяла идентичность результатов генерации плагинов с помощью qrxc (qreal xml compiler) и qrmc (qreal metamodel compiler), и результата генерации плагина с помощью qrxc с результатом интерпретации метамодели. Методы, которые система должна протестировать, задаются в текстовых файлах (изначально система тестирует все методы).

Она принимает на вход метамодель, запускаются qrxc и qrmc соответственно, генерируя код, далее код собирается и загружаются плагины, после чего в MainClass возвращаются два объекта типа EditorInterface для qrxc и qrmc. Далее EditorInterface для qrxc приводится к EditorManager, а также создается InterpreterEditorManager. После MainClass создает экземпляры тестирующих классов, передавая им EditorInterface для тестирования qrxc и qrmc и EditorManagerInterface для тестирования qrxc и интерпретатора. У них вызывается метод “тестировать”. И результаты тестирования заносятся в

списки, которые впоследствии попадают в HtmlGenerator, возвращающий таблицы с результатами сравнения.

Реализация

Ранее описанная тестовая утилита в своё время не была встроена в основной проект. С тех пор, произошло много изменений в QReal, так что стало необходимо восстановить работоспособность тестовой утилиты.

Для этого были исправлены некоторые фрагменты тестовой утилиты, отвечающие за генерацию кода, его сборку и создание визуальных редакторов. Также пришлось изменить и добавить пути до файлов, необходимых для сборки тестовой утилиты, относящихся к qrxc, qrmc и др.

После чего встал вопрос о работе непосредственно самой тестовой утилиты. Некоторые тестирующие методы работали некорректно либо не существовали вовсе, вследствие чего была настроена корректная работа некоторых методов, работающих неверно, и добавлена часть недостающих методов.Так, например, были исправлены методы такие, как enumValues(), isParentOf() и др. и было добавлено несколько методов таких, как diagramPaletteGroup(), paletteGroupDescription().

Во время работы с тестовой утилитой были выявлены несоответствия между визуальными редакторами, полученными с помощью qrxc, qrmc и интерпретатора метамоделей. Эти различия пришлось устранить путем корректирования методов соответствующих компиляторов и интерпретатора. Так, например, некорректные результаты выдавали методы редактора, полученного с помощью интерпретации метамодели, а именно: Elements(), EnumValues(). Первый метод был немного подкорректирован, а последний был переписан с нуля.

Далее была реализована идея, иллюстрирующая то, как именно будет измеряться время работы каждого из методов. Измерением времени в рамках курсовой работы занималась Тарасова П.М. В частности, в данной работе была подготовлена база для того, чтобы можно было правильно измерять время, получать результаты измерений и их анализировать. Для всех методов каждого из визуальных редакторов производятся замеры времени, которые сохраняются в отдельный список соответствующего тестирующего класса в формате: “Имя метода”, “Время работы в первом редакторе”, “Время работы во втором редакторе”. После отрабатывания всех методов вся информация вместе с результатами сравнения передаётся в HtmlGenerator, в котором помимо таблиц сравнения результатов всех методов теперь генерируются таблицы с измерениями времени для всех методов каждого из редакторов.

Впоследствии было создано несколько различных метамоделей и были проведены эксперименты с ними. Сравнивались скорости работы четырнадцати методов (пример измерений для трёх методов приведён в таблице). В среднем в двенадцати из них генеративный режим по сравнению с интерпретативным был быстрее приблизительно в 500 раз.

Имя метода	**Генеративный режим**		**Интерпретативный режим**	
	Мат.ожидание (мкс)	Среднеквад-е откл-е (мкс)	Мат.ожидание (мкс)	Среднеквад-е откл-е (мкс)
Element description	64	2	18000	1700
Get property default values	59	<1	39000	3000
Is parent of	250	26	33000	7900

Таблица 1.

Это связано с тем, что в интерпретативном режиме при вызове методов часто требуется обход всей метамодели, а в генеративном используются предвычисленные значения, полученные при компиляции, поэтому при таком подходе большая часть методов выполняется за константное время.

Заключение

В рамках курсовой работы были получены следующие результаты:

- Восстановлена работоспособность тестовой утилиты
- 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] 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.

[2] 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.

[3] 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)

[4] 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)

[5] Ю.В.Литвинов, Разработка визуальных предметно-ориентированных языков (диссертация), URL: https://github.com/qreal/articles/blob/master/litvinovPhDThesis/dissertation.pdf (дата обращения: 10.05.2015)

[6] П.М.Тарасова, Сравнение способов получения редакторов по метамодели и скорости их работы, курсовая работа (не опубликовано)