Санкт-Петербургский государственный университет

Фундаментальная информатика и информационные технологии
Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Гудошникова Анна Андреевна

Технология создания семейства приложений на основе анализа предметной области

Магистерская диссертация

Научный руководитель:
д. ф.-м. н., профессор Терехов А. Н.

Рецензент:
Оносовский В. В.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Fundamental Informatics and Information Technology
Mathematical and Software Support for Computers, Computer Systems and Networks

Gudoshnikova Anna

Technology for application family creation based on domain analysis

Master's Thesis

Scientific supervisor:
professor A.Terekhov

Reviewer:
V. Onossovski

Saint-Petersburg
2016

Оглавление

Введение

Тема переиспользования объекта в сфере разработки программного обеспечения в настоящее время рассматривается активно. Объектом переиспользования может быть что угодно, например: классы, методы, целые библиотеки, спецификации, требования, архитектуры, планы тестирования, тест-кейсы и многое другое.

Принято считать, что переиспользование как некая деятельность подразделяется на виды в зависимости от того, что будет переиспользовано: компоненты продукта, процесс получения программного продукта, технология для получения продукта или же знания, которые представляют некий опыт работы. Каждый следующий вид переиспользования представляет собой более высокий уровень абстракции.

В области программного обеспечения трудно понять, что именно надо переиспользовать и каким путем. Например возможна ситуация, когда переиспользование одного объекта тянет за собой переиспользование связанных с ним компонент. Таким образом возникает проблема контекста переиспользования.

Однако, если переиспользовать объекты в одной предметной области, то проблема контекста может быть сужена. Линейка продуктов подразумевает, что в линейке есть что-то общее, например: архитектура, компоненты, алгоритмы, методы и др., — и эти общие элементы находятся в одном и том же контексте. Общие части линейки должны быть переиспользованы. Очевидны преимущества такого переиспользования. Во-первых, это ускоряет процесс разработки, так как не надо реализовывать один и тот же функционал, а во-вторых, такое переиспользование обезопасит разработчков от ошибок, т.е. линейка продуктов будет соответствовать должному уровню качества.

Переиспользование в одной предметной области предполагает, что необходим анализ предметной области, чтобы понять, что и как переиспользовать. Для того, чтобы говорить о переиспользовании с помощью анализа предметной области, необходимо понять, что такое анализ предметной области, а сперва и что такое “предметная

область". Итак, предметная область — та область знаний, проблемы которой создаваемое программное обеспечение призвано решать. По Ругаберу [1], предметная область характеризуется своим словарем, принятыми соглашениями, архитектурным подходом и литературой.

Релевантная информация о предметной области должна быть предоставлена в каком-то объективном, легкодоступном и понятном виде. Такой вид называют моделью предметной области. По Мернику [2] модель предметной области должна включать в себя не только словарь терминов предметной области, но также должна описывать сходства и изменчивость этих понятий. Такая модель должна точно задавать границы области, т.е. четко и ясно описывать тот круг вопросов, который будет рассматриваться в рамках этой области.

Изменчивость понятий (от англ. "variabilities") позволяет точно определить, какая информация должна быть специфицирована в реализации конкретной системы. Сходства (от англ. "commonalities") используются для определения вычислительной модели (как множества общих операций) и примитивов языка. Реализуя сходства и дополняя полученную вычислительную модель той информацией, которая задается в экземпляре конкретной системы, мы получаем множество систем, основанных на одной вычислительной модели. Таким образом на базе одной модели предметной области может быть разработано целое множество различных систем в этой предметной области. Это заметно ускоряет процесс разработки ПО.

Можно сказать, что так или иначе, в настоящее время как бы ни происходил анализ предметной области и каким бы ни был итог этой деятельности, он в самом лучшем случае распечатывается в каком-либо виде или изложен в каких-либо электронных базах знаний, но никак явно не помогает в разработке программного обеспечения. Т.е. разработчики анализируют как-то эти знания и реализуют то, как они это поняли. Риск некорректного понимания специфики предметной области возрастает. Таким образом, некоторые особенности предметной области могут быть упущены, благодаря человеческому фактору. Далее возрастает риск реализации совсем не того продукта, который хотел

заказчик, т.е. требования были поняты по-другому, а следовательно, и реализованы не так, как хотелось бы заказчику. Результат — продукт не соответсвует требованиям.

Вследствие этого существует необходимость в инструменте, в котором деятельность по анализу предметной области играет ключевую роль при разработке программного обеспечения. Подразумевается, что, опираясь на деятельность по анализу предметной области, будет возможно сгенерировать некоторую проективную модель, так, чтобы разработчики и другие участники процесса могли бы положиться на такую модель и быть уверенными в ее валидности и актуальности. Таким образом, вероятно, уменьшится риск человеческой ошибки в понимании терминов и особенностей предметной области.

В качестве такого инструмента может быть рассмотрен metaCASE-инструмент. Такие инструменты призваны создавать CASE-средства (от англ. “Computer-Aided Software Engineering”) [3], в которых по описанию приложения в виде множества диаграмм можно сгенерировать код приложения. Такое описание создается на предметно-ориентированном визуальном языке. Описание сущностей и связей же самого такого визуального языка задается с помощью его метамодели. Такая метамодель описывается как раз в metaCASE-инструменте. Так, можно соединить деятельность по анализу предметной области и meta-технологию для почти автоматической генерации различных конфигураций приложений по модели предметной области. Такой подход заметно ускоряет процесс разработки. Также, реализуя инструменты для анализа предметной области в metaCASE-инструментах, получаем возможность получить именно тот продукт, который удовлетворяет выдвинутым требованиям.

Постановка задачи

Целью данной работы является создание технологии разработки семейства приложений, позволяющей по модели характеристик предметной области полуавтоматически генерировать предметно-ориентированный визуальный язык для конфигурирования продуктов семейства. Для достижения этой цели были сформулировны следующие задачи.

1. Рассмотреть различные генерационные подходы к созданию семейств приложений.

2. Описать метод переиспользования ПО, основанный на построении модели характеристик и генерации по ней метамодели визуального языка, который используется для конфигурирования и описания правил композиции переиспользованных компонентов целевого приложения.

3. Реализовать инструментальную поддержку предложенного метода на основе metaCASE-инструмента QReal.

4. Провести апробацию представленного подхода на примере создания семейства приложений для удаленного управления роботами разных моделей с мобильного телефона, а также на примере создания модельного семейства приложений текстовых редакторов.

1. Обзор существующих подходов

1.1. Методы анализа предметной области

В настоящее время сбор всей информации в некие базы знаний и понимается под анализом предметной области. Однако, Прието-Диаз [4] утверждает, что анализ предметной области — это некая активность, которая случается перед системным анализом. Итогом этой деятельности является модель предметной области, которая помогает в анализе системы в той же мере, как итог системного анализа, т.е. проанализированные требования и спецификации, помогает при задачах проектирования системы. В последовательной водопадной модели разработки программного обеспечения каждый следующий шаг идет на уточнение и специфицирование одной конкретной системы. В то время как с помощью анализа предметной области происходит обобщение всех систем в одной данной конкретной предметной области. Т.е. анализ предметной области — это более высокий уровень абстракции, чем системный анализ.

Рассмотрим другое определение анализа предметной области. Например, Фере [5] выделил такие определения этого термина.

1. Процесс идентификации, организации и представления релевантной информации о предметной области.

2. Процесс, при котором знания заказчика/пользователя идентифицируются, конкретизируются и систематизируются.

3. Деятельность по проекту, которая предваряет системный анализ.

Учитывая две вышеобозначенные точки зрения можно обобщить, что анализ предметной области — это деятельность, предваряющая системный анализ, целью которой является предоставление модели предметной области. Согласно Прието-Диазу [4] также заключается, что при такой деятельности, общие характеристики для схожих систем обобщаются, объекты и операции, общие для всех систем в заданной

предметной области, идентифицируются, определяется модель, которая описывает их взаимодействия.

Использование баз знаний, конечно, дает некий результат, но такие базы нужно сначала создать. На данный момент они создаются так называемым неформальным способом. Очевидны недостатки такого метода. Возможны случаи неполного словаря, отсутствие соглашений о понимании тех или иных терминов и др., что, в свою очередь, ведет к непониманию предметной области, а следовательно, к риску того, что будет реализован неполный или вообще другой функционал. Таким образом, были разработаны формальные методы для анализа предметной области, а также инструменты для такого анализа.

В работе [6] уже был приведен обзор формальных методов анализа предметной области. Обозначим здесь основные моменты.

Выбор метода анализа предметной области зависит от поставленных целей, которые необходимо достичь, например, целью может служить создание библиотеки переиспользуемых компонент. Но тем не менее, в рамках каждого из методов должна быть создана или неким образом описана модель предметной области.

Несмотря на разное понимание того, что такое «анализ предметной области» разными авторами, Аранго [7] показал, что все методы анализа предметной области придерживаются так называемого общего процесса получения модели предметной области, который включает в себя этапы: этап характеристики предметной области (от англ. “Domain characterisation”), этап сбора данных (от англ. “Data Collection”), этап анализа данных (от англ. “Data analysis”), этап классификации (от англ. “Classification”) и, наконец, этап оценки модели предметной области (от англ. “Evaluation of Domain Model”).

Формальные методы анализа различаются некоторыми техниками, которые применяются при анализе данных, а также параметрами, которые используются при классификации.

На этапе анализа данных могут использоваться следующие техники:

- объектно-ориентированная;

- декомпозиция функциональности или данных;
- анализ качества;
- количественный анализ;
- технология, основанная на случаях использования (от англ. “case-based technology”).

При классификации или на этапе основного моделирования выделяются параметр или параметры, по которым и происходит разделение. Такими параметрами могут служить:

- фасеты (от англ. “facets”) или аспекты;
- характеристики (от англ. “features”);
- возможности (от англ. “capabilities”);
- требования (от англ. “requirements”).

Также в работе были рассмотрены следующие методы для формального анализа предметной области.

1. DARE (от англ. “Domain Analysis and Reuse Environment”) [8]. Ключевым моментом в этом методе является создание так называемой книги предметной области.
2. DSSA (от англ. “Domain-Specific Software Architectures”) [9]. Этот метод позволяет создать словарь предметной области при помощи анализа пользовательских сценариев.
3. FODA (от англ. “Feature-Oriented Domain Analysis”) [10], об этом методе будет рассказано далее.
4. FORM (от англ. “Feature-Oriented Reuse Method”) [11], является расширением метода FODA. В рамках метода рассматривается 4х-уровневая модель характеристик (от англ. “feature model”).

5. ODE (Ontology-based Domain Engineering) [12]. Метод соединяет онтологии и объектно-ориентированную технологию.

6. ODM (Organization-Domain Modeling) [13]. В данном методе создаются раздельные, но взаимосвязанные концептуальные модели, в результате получается так называемая сеть моделей (от англ. “model web”).

1.2. Метод анализа предметной области FODA

Большое распространение получил получил метод FODA (от англ. “Feature-Oriented Domain Analysis”) [10]. В этом методе модель предметной области строится в три этапа: контекстный анализ, чтобы установить масштаб задачи, моделирование предметной области для определения пространства задачи, и наконец, моделирование архитектуры, т.е. спуск на уровень решений и реализации.

Модель предметной области включает в себя словарь терминов, а также три модели: модель сущность-связь (от англ. “entity-relationship model”), которая показывает, как соотнесены между собой сущности, какие между ними отношения, модель характеристик (от англ. “feature model”), которая описывает пространство требований, и функциональная модель (от англ. “functional model”), которая определяет, какие характеристики являются общими для продуктов в предметной области.

Модель характеристик строится на основе выделения характеристик, которыми должен обладать рассматриваемый программный продукт. Модель характеристик рассматривается именно на уровне предметной области, т.е. характеристики включают в себя описание тех или иных аспектов даной предметной области, а не на уровне реализации продукта. В этом методе характеристики могут быть:

- обязательными;

- обязательными, но между которыми есть выбор, т.е. одна из них должна быть реализована;

- опциональные или дополнительные характеристики.

Таким образом, можно описать целое семейство связанных друг с другом программных продуктов в одной предметной области. Также в статье был дан конкретный синтаксис для описания таких моделей. Пример можно увидеть на Рис. 1

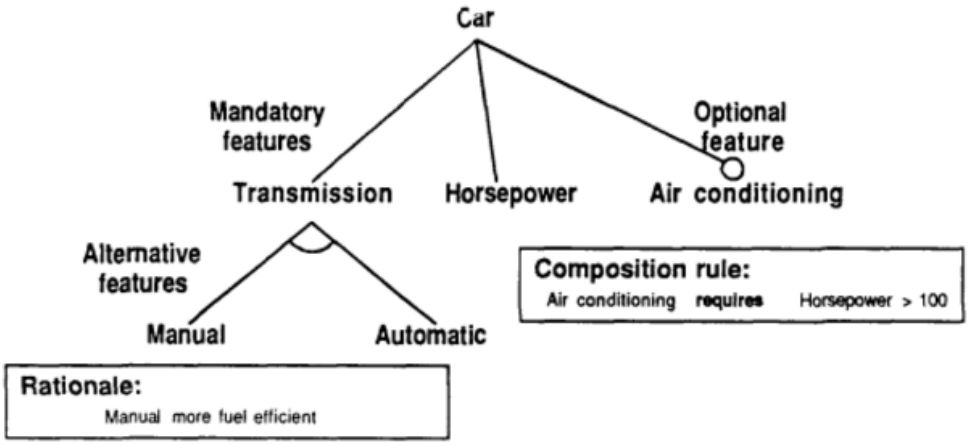

Рис. 1: Пример диаграммы характеристик автомобиля [10].

На Рис. 1 можно видеть, какие характеристики являются обязательными, между каким есть альтернатива, а какие являются опциональными, также могут задаваться ограничения на модель и отношения между характеристиками. Статья про метод FODA датируется 1990 годом. С тех пор появились инструментальные средства, которые позволяют не только строить модели характеристик, но также и подвергать их определенным операциям, например, объединять модели с сохранением валидных конфигураций, выделять подмодели и др. Например, таким инструментальным средством является FAMILIAR [14]. Есть также средства, в которых осуществляется полное управление изменениями в линейке продуктов на всех стадиях жизненного цикла разработки программного обеспечения, начиная от управления требованиями до тестирования. Таким средством, например, является pure::variants [15]. Строить модель характеристик довольно просто и наглядно, вероятно, поэтому этот метод приобрел такую популярность.

1.3. Подходы к разработке линейки продуктов, включающие переиспользование

1.3.1. Предметно-ориентированные языки

Для программирования в конкретной предметной области существуют так называемые предметно-ориентированные языки или DSL (от англ. “domain-specific languages”) [2]. Такие языки призваны быть простыми и естественными для использования экспертами в предметной области, но покрывают ограниченное множество приложений. Разрабатывая программное обеспечение в определенной предметной области, предметно-ориентированные языки легче использовать, чем языки общего назначения, а также уменьшается время разработки и стоимость поддержки приложений, созданных на таких языках. Предметно-ориентированные языки могут быть как текстовыми (примерами могут служить, TeX/LaTeX, SQL, Prolog, VHDL и др.), так и визуальными. Так, на визуальном предметно-ориентированном языке можно описать компоненты системы в виде диаграмм, и по таким диграммам можно сгенерировать код системы. Такой подход реализован в Computer-Aided Software Engineering (CASE) системах [3]. Так как процесс разработки конкретной CASE-системы со встроенным предметно-ориентированным языком трудоемок, существуют metaCASE-системы, которые генерируют CASE-систему вместе с предметно-ориентированным языком, который будет использоваться в CASE-системе. Язык, с помощью которого описывается предметно-ориентированный язык, называется метаязыком. Модели, построенные на метаязыке, называются метамоделями. Таким образом, метамодель предметно-ориентированного языка хранит в себе знания о предметной области, т.е. способствует переиспользованию знаний.

1.3.2. Порождающее программирование

Порождающее программирование (от англ. “generative programming”) [16] — это технология проектирования и реализации

модулей программного обеспечения, которые могут быть собраны для генерации специализированных систем, к которым предъявляются особые требования. Порождающее программирование основано на инженерии предметной области (от англ. “Domain Engineering”), которая может состоять из трех частей: анализ предметной области (об этом говорилось выше), проектирование предметной области (т.е. разработка архитектуры, общей для всех систем в данной предметной области) и реализация предметной области (т.е. реализация средств многократного применения). На начальной стадии определяется предметная область, моделируются характеристики и понятия. Затем проектируется архитектура и выявляются компоненты реализации. Потом устанавливаются знания о конфигурациях и реализуются компоненты. Основные принципы порождающего программирования.

1. Разделение понятий (от англ. “separation of concerns”). Это понимается в том же самом смысле, что и в объектно-ориентированном программировании. Т.е. избегать программного блока кода, который одновременно реализует несколько вещей. Целью порождающего программирования является отделение реализации каждого компонента системы в отдельный блок кода.

2. Параметризация различий. Параметризация позволяет реализовать не одно приложение, а целое семейство приложений в определенной области.

3. Анализ и моделирование зависимостей и взаимодействий.

4. Отделение пространства задачи от пространства решения, т.е. от пространства конкретной реализации.

5. Исполнение предметно-ориентированных оптимизаций.

Главной целью технологии порождающего программирования является автоматическая генерация продуктов в одной линейке. Определенная компонента может быть автоматически получена из спецификаций, которые описывают ее, они могут быть представлены на текстовом

DSL или визуальном. Архитектура приложения должна быть спроектирована, учитывая различные комбинации характеристик, но сама архитектура располагается в пространстве конкретной реализации.

1.3.3. Методология разработки, основанная на моделях

Основным артефактом в методологии разработки, основанной на моделях (от англ. “Model-driven engineering” или “MDE”) является модель. Модель определяется как абстрактное описание программного обеспечения, которое скрывает некоторые аспекты с целью упрощения системы. Модели создаются на основе формальных и явно заданных метамоделей. В таком подходе рассматриваются несколько метамоделей, которые позволяют охарактеризовать систему с разных сторон в определенной предметной области. В отличие от DSLs, MDE подразумевает наличие последовательных уточняющих трансформаций моделей, которые описывают целевую систему. Такие трансформации приводят в последующем к генерации кода. Первыми инструментами, которые поддерживали MDE подход, были CASE-системы, о которых говорилось выше.

1.3.4. Mélusine подход

В работе Эстублиера [17] представлены требования, которым должен удовлетворять метод разработки линейки продуктов с помощью переиспользования, а также обозначен и некоторый исследовательский опыт, т.е. те уроки, которые надо учитывать при разработки больших систем в рамках одной предметной области. Например, при разработке линейки продуктов надо рассматривать следующие аспекты:

1. полагаться на абстрактное и стабильное описание задачи, которую надо решить;

2. явно обозначить все изменения в терминах характеристик продукта, а не в терминах конкретной реализации;

3. переиспользовать крупномодульные компоненты.

Также авторы выделили четыре требования, которым должен удовлетворять метод создания линейки приложений с помощью переиспользования:

1. позволять развитие абстрактной архитектуры;
2. механизмы для реализации изменчивости системы должны быть в области проблемы, а не конкретной реализации, т.е. должны быть возможности, которые помогут описать большое семейство приложений без деталей реализации;
3. соответствие между архитектурой, которая описана в терминах характеристик, присущих системе, и реализацией компонент должно быть на высоком уровне абстракции;
4. переиспользовать те компоненты, которые уже были реализованы в других проектах, таким образом, чтобы начать разрабатывать линейку продуктов не с нуля, а имея на руках уже какие-то реализованные компоненты, возможно даже из другой предметной области.

По последнему пункту необходимо заметить, чтобы эти реализованные компоненты должны быть в том же контексте, что и те компоненты, которые нам необходимы в текущем проекте. Если они будут реализованы в другом контексте, то переиспользование их в текущем проекте может быть неприменимо. На 2005 год, когда была выпущена статья Эстублиера и Вега, авторами были рассмотрены четыре способа создания линейки продуктов при помощи переиспользования, далее они рассматриваются в разрезе от выше обозначенных требований и аспектов. Авторы данной статьи выделили три основных способа разработки линейки продуктов с помощью переиспользования: DSLs, порождающее программирование, подход, реализующий методологию MDE, а также DSM (от англ. “Domain Specific Modeling”). DSM основано на трех вещах: конкретный DSL, генератор и базовое средство разработки (от англ. “framework”). Результатом анализа предметной области

также является модель предметной области, однако, в отличие от порождающего программирования, в DSM такая модель представляет лишь основные понятия и поведение, такая модель статична и не подразумевает изменений. Все изменения должны быть представлены с помощью конкретного DSL, который получается из модели предметной области. В разрезе выдвинутых авторами требований и опытных наблюдений были рассмотрены выше обозначенные подходы. В DSL подходе в качестве абстрактной архитектуры рассматривается сам язык, который стабилен и описывает предметную область. Также при таком подходе можно переиспользовать крупномодульные компоненты только для специально разработанных систем. DSL не фокусируется на изменчивости проблемы, которую надо решить, можно только изменить реализацию. Компилятор может соотнести архитектуру и компоненты, однако это скрыто в коде компилятора. Нельзя переиспользовать сторонние компоненты, так как DSL является подходом, так называемым top-down, то есть реализует спуск с верхнего уровня абстракции. Подход DSL представлен на Рис. 2

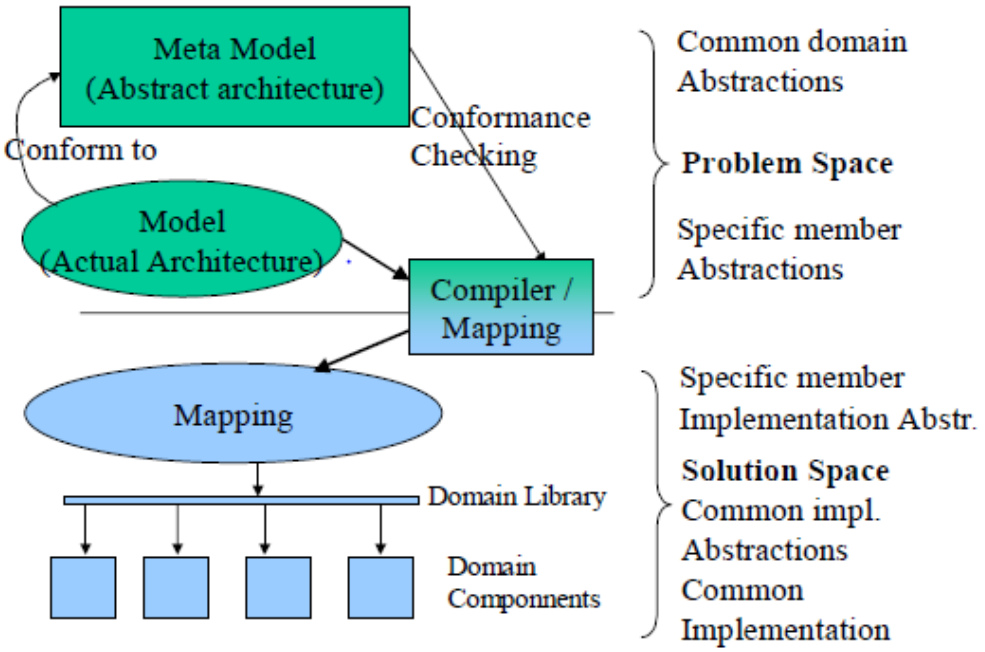

Рис. 2: DSL подход [17].

Подход порождающего программирования кажется более выиг-

рышным по сравнению с DSL подходом. Сама архитектура располагается в пространстве реализации, однако DSL может выполнить роль стабильного описания проблемы, которую надо решить. Многие характеристики в этом подходе принадлежат пространству решений, а не пространству проблемы. Однако модель характеристик статична, никакой эволюции абстрактной архитектуры не предполагается. В методологии MDE модели и метамодели обеспечивают абстрактное и стабильное описание проблемы. Изменчивость может быть представлена на уровне метамодели. При таком подходе нелегко переиспользовать высоко-абстрактные компоненты. Однако метамодели статичны. MDE является подходом, который реализует спуск с абстрактного уровня до уровня реализации, т.е. top-down подход. В MDE подходе есть механизм транформации, который как раз нацелен на соответствие архитектуры и компонент на высоком уровне абстракции. В DSM присутствуют три уровня изменчивости, которыми можно независимо управлять. Крупномодульное переиспользование доступно, но только конкретно в данной предметной области. В современных DSM-инструментах реализована классическая кодовая генерация. Переиспользование сторонних компонент ограничено.

Авторами данной статьи был предложен собственный подход Mélusine, который удовлетворяет всем трем урокам и трем требованиям, которые были обозначены выше. Он основан на MDE подходе. Модель предметной области рассматривается как метамодель, представленная на одном из формальных языков, таких как MOF или UML. Существует интерпретатор, который транслирует каждое понятие в метамодели в Java-класс, а конкретные модели в экземпляры этих классов. Модель предметной области дополняется моделью характеристик, которая содержит в себе дополнительное поведение системы. При таком подходе, модель характеристик может быть определена после получения модели предметной области, что, в свою очередь, позволяет поддерживать ее независимо. Характеристики рассматриваются как комплексные понятия, относящиеся к интерпретатору. Авторы используют аспектно-ориентированные техники для

реализации характеристик и для последующего их соответствия с понятиями в предметной области. На Рис. 3 показано, как можно соединить подход порождающего программирования и DSM подход.

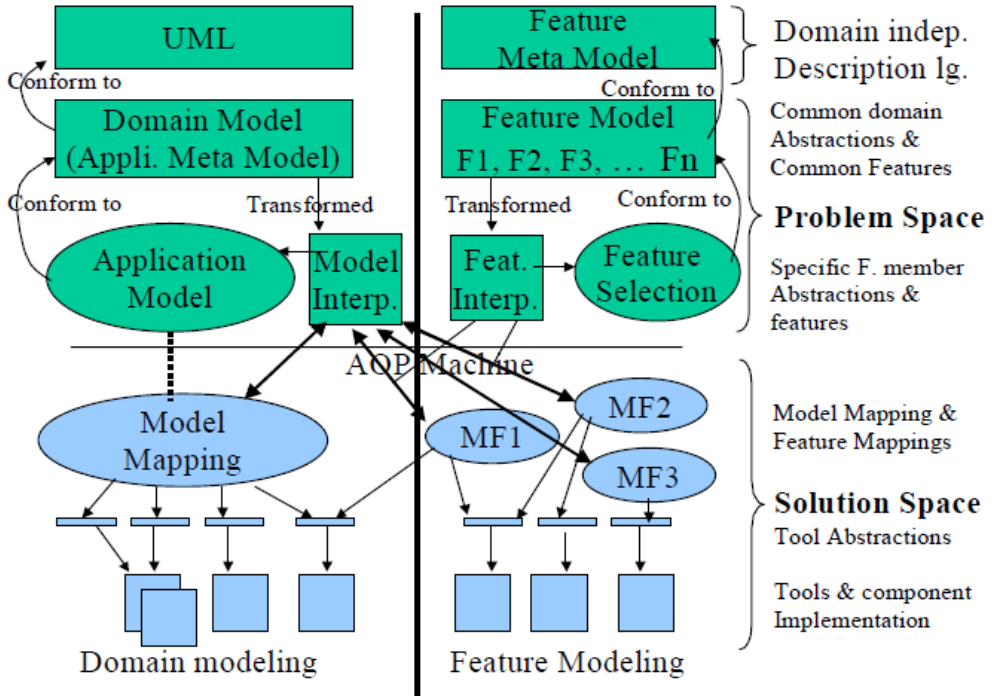

Рис. 3: Предметное моделирование и моделирование характеристик [17].

Этот подход хотя и удовлетворяет всем требованиям, которые были выдвинуты самими авторами, но представляется довольно сложным для понимания и реализации. Так же в этой статье представлена ссылка на другую статью этих же авторов [18], где они рассматривают данный подход на примере одной из систем. Стоит отметить, что никакого промышленного применения такого подхода не было, а также такой подход не был оценен сторонними экспертами, можно полагаться только на выводы авторов.

Если касаться применимости существующих подходов к управлению изменениями, то довольно хороший обзор таких подходов приведен в работе Чена [19]. В этой работе были применены фильтры для поиска статей, которые так или иначе представляют методы для менеджмента изменений. Однако были оценены только те методы, в которых предоставляется анализ применимости такого метода. Оказалось, что

на 2010 год только один метод имеет оценку стороннего эксперта, а не самих разработчиков метода [20]. Также почти отсутствуют методы, которые имеют строгий анализ применимости предложенного метода, а 96% из всех рассмотренных методов оцениваются только для одной области, что является не лучшим показателем применимости предложенных методов по управлению изменениями в программных продуктах.

1.4. Выводы

Как можно заметить из обзора, существуют различные методы для разработки семейства приложений с помощью переиспользования, многие основываются на применении визуального или текстового DSL. Многие статьи описывают применения метода FODA, который используется в основном для представления изменчивости в семействе приложений в дополнение к модели предметной области. Если метод FODA использовался для представления модели предметной области, то такие методы ограничивались только этой моделью, дальнейшего использования модели для генерации приложений не было, только для получения систематизированных знаний о предметной области. Есть также подходы, которые генерируют код, но при этом используется напрямую модель предметной области, что не очень понятно и удобно для непрограммистов, т.е. для эспертов в предметной области. Таким образом из обзора можно сделать вывод, что простого способа использовать модель предметной области для построения визуального языка, а затем и для генерации семейства приложений не наблюдается. Также можно заключить, что промышленное использование всех обозначенных выше методов для создания семейства приложений, включающее переиспользование, не рассматривалось. Задача использовать модель предметной области для создания семейства приложений актуальна, и стоит проводить дальнейшие исследования в этой теме.

2. Описание подхода

Наш подход использует идеи метода FODA для проведения деятельности по анализу предметной области. Предлагается строить модель характеристик в качестве модели предметной области. Для этого мы используем визуальный редактор, в котором и происходит построение модели характеристик. Такое построение довольно удобно для экспертов предметной области. Для построения модели характеристик используется предварительно разработанный предметно-ориентированный язык, который понятен эксперту предметной области [6].

После того, как модели характеристик готовы, каждая характеристика реализуется как переиспользуемая и настраиваемая компонента на любом языке программирования, будь то C#, Java или C++. Библиотека переиспользуемых компонент формируется за счет этих реализованных характеристик. Такой процесс требует высококвалифицированных программистов и требует больше усилий, чем просто разработать одно приложение, но такой процесс позволяет переиспользовать реализованные характеристики, просто обращаясь к библиотеке реализованных компонент для создания стольких приложений, сколько требуется. Более того, такой процесс масштабируем. Так, мы можем добавлять новые характеристики в библиотеку реализованных компонент и таким образом получить возможность создавать более сложные приложения с большим количеством возможных конфигураций. На этой стадии разработки эксперты предметной области должны работать с программистами, так как модели или диаграммы характеристик используются как входные данные для создания библиотеки реализованных компонент.

Следующим шагом является создание предметно-ориентированного языка, который позволяет комбинировать и конфигурировать характеристики из библиотеки реализованных компонент для реализации приложений в данной конкретной предметной области. Именно на этом шаге наш подход отличается от всех общих стратегий пе-

реиспользования. Обычно стараются сгенерировать код приложения напрямую из модели характеристик, каким-то образом выделяя те характеристики, которые должны быть реализованы в приложении. Такой подход работает, если количество изменяемых характеристик мало. Однако есть примеры, когда вариативная модель состоит из более 1000 характеристик, например [21].

Также в общем случае характеристики могут иметь собственные свойства, которые позволяют их конфигурировать, а эти свойства могут быть различных типов. Более того, характеристики могут быть связаны друг с другом или некоторые их свойства могут терять свое значение при отсутствии или наличии других характеристик. Такие правила должны быть реализованы неявно в генераторе приложения и требуют, чтобы программисты всегда соблюдали их. Мы предлагаем такой подход, при котором такие правила отображаются явно благодаря предметно-ориентированному языку. Такой язык позволяет объявлять модели синтаксически некорректными, в случае если они не соблюдают эти правила. Таким образом, уменьшается возможность человеческой ошибки и количество необходимой информации для эффективного использования системы.

Используя DSM-платформы можно относительно быстро создать предметно-ориентированный язык, который включает в себя необходимую информацию о предметной области, однако мы уже имеем модель характеристик, поэтому можно сгенерировать визуальный предметно-ориентированный язык, используя эту модель. Таким образом, знания эксперта предметной области напрямую используются для создания приложений. Генератор берет в качестве входных данных модель характеристик и строит метамодель предметно-ориентированного языка. Метамодель можно просмотреть и отредактировать в метаредакторе DSM-платформы. Характеристики из модели характеристик становятся сущностями в метамодели, такая метамодель затем редактируется, чтобы задать визуальную форму для элементов предметно-ориентированного языка, а также чтобы задать необходимые свойства для каждого элемента. Любое векторное изображение может играть роль визуального

отображения элемента в предметно-ориентированном языке, однако хорошая практика выбирать такое изображение, которое отображает сущность характеристики. Например, если приложение имеет кнопки, то “кнопка” становится сущностью в предметно-ориентированном языке и выглядит как кнопка на диаграмме. Для каждой характеристики свойства добавляются в метаредакторе, такое свойство должно соответствовать библиотеке реализованных компонент. Свойства имеют название, тип и значение по умолчанию. На этом шаге также возможно задать некие ограничения на метамодель, которые будут проверяться на этапе редактирования модели. Если какие-либо ограничения будут нарушены, пользователь не сможет построить модель. На следующем шаге мы используем генератор редакторов DSM-платформы для создания визуального редактора для нашего созданного языка. Этот шаг полностью автоматизирован. Когда редактор сгенерирован и загружен в DSM-платформу, мы можем использовать его для создания диаграмм, которые бы описывали наши целевые приложения.

Следующий шаг — генерация кода на целевом текстовом языке, который будет вызывать библиотеку реализованных характеристик и склеивать реализованные характеристики вместе. Для этого мы должны спуститься на уровень метамодели и определить правила генерации для метамодели. Этот шаг проводится только однажды для одной конкретно определенной области после того, как библиотека реализованных компонент и метамодель готовы, и затем тот же генератор используется для каждого создаваемого по этой технологии приложения. Рекомендации для разработки предметно-ориентированного генератора представлены в DSM-литературе, например [22]. Хорошей практикой является создавать первое приложение вручную, потом создать модель, которую предполагается получить после генерации, затем найти места в коде, которые предполагаются быть параметризованными информацией из модели и позволить генератору заменять такие места в коде данными из модели. Такой процесс продолжается, пока рукописное приложение не становится шаблоном, который генератор дополняет информацией, взятой из

модели. Написанное вручную приложение и, следовательно, генератор должны активно использовать библиотеку реализованных компонент, в идеале генератор должен только склеивать код из тех характеристик, которые имеются в библиотеке.

После всех пройденных шагов у нас имеется библиотека реализованных характеристик, визуальный редактор для простого предметно-ориентированного языка, который позволяет описывать, как характеристики комбинируются и конфигурируются в конкретном приложении, и генератор, который автоматически производит завершенное приложение по модели на предметно-ориентированном языке, используя библиотеку реализованных характеристик как предметно-ориентированную среду исполнения. Сейчас мы уже можем создать столько приложений, сколько требуется, просто создавая модели и автоматически генерируя завершенный исполняемый код. Безусловно, это лишь теоретическое заключение. Практически можно обозначить следующее. Всегда будет наблюдаться необходимость в модификации диаграммы характеристик, расширять библиотеку реализованных компонент, и, следовательно, метамодель предметно-ориентированного языка, изменять генератор и даже делать некоторые изменения в сгенерированном коде приложения. Такой подход не является “серебряной пулей”. Можно только сказать, что такой подход может обеспечить лучшее разделение понятий, лучше использовать знания и опыт экспертов в предметной области в команде разработки. Общая схема всего процесса, описывающая отношения между инструментами и ролями разработчиков, приведена на Рис. 4.

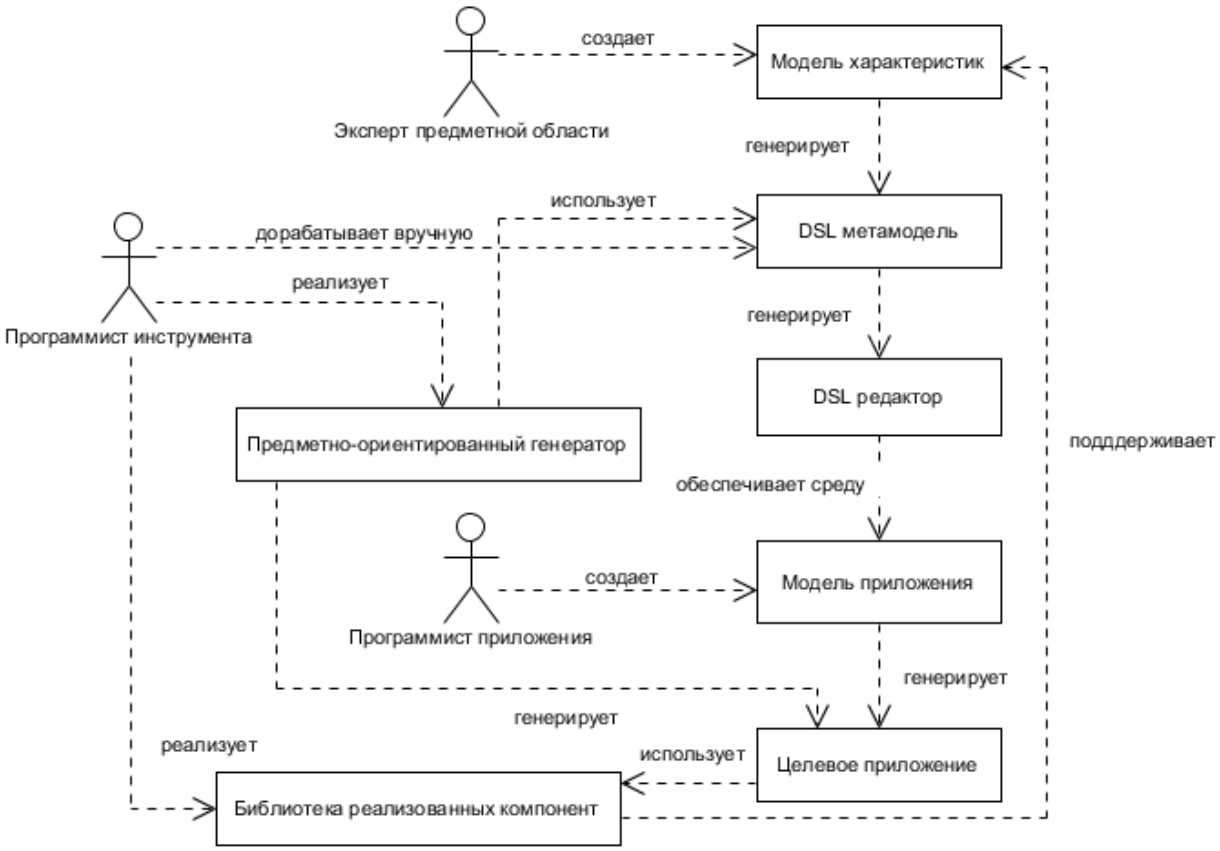

Рис. 4: Общее описание процесса.

3. Реализация

Для апробации данного подхода был выбрана DSM-платформа QReal, разрабатываемая на кафедре системного программирования СПбГУ. Этот инструмент был выбран потому, что обеспечивает простой и эффективный способ создавать визуальные редакторы для предметно-ориентированных языков.

QReal — система с многоуровневой архитектурой. Каждый визуальный редактор, который создается в системе, является отдельным подключаемым модулем. Система QReal включает в себя абстрактное ядро, которое предоставляет базовый функционал для всех редакторов, и модули, реализующие всю специфику языков. Модули одинаково трактуются абстрактным ядром.

Архитектура данного инструмента представлена на Рис. 5. Ядро инструмента состоит из компонентов qrmc, qrgui, qrxc, qrkernel, qrutils, qrrepo. Компоненты ”domainAnalysis” и ”appFamily” являются подключаемыми модулями.

Элементы метамодели (модели) хрянятся в репозитории. Эти элементы имеют логическое и графическое представления, которые хранятся в отдельных файлах для того, чтобы их можно было редактировать и версионировать независимо. Графическая и логическая модели ссылаются на один и тот же репозиторий, но обращаются к нему по разным интерфейсам. Хранятся такие логические и графические элементы связаннами.

Подключаемый модуль (или плагин-инструмент) изменяет пользовательский интерфейс в соответствии с функциональностью плагина-инструмента. Взаимосвзяь происходит по следующим интерфейсам.

- CustomizationInterface — интерфейс для настройки внешнего вида пользовательского интерфейса под конкретный плагин-инструмент.

- PluginConfigurator — класс-контейнер, который содержит те объекты пользовательского интерфейса, которые нужны для

плагина-инструмента.

- ToolPluginInterface — интерфейс плагина-инструмента. Позволяет пользовательскому интерфейсу получить список действий, выполняемых плагин-инструментом, и информацию, как эти действия отражаются в пользовательском интерфейсе.

QReal имеет визуальный метаредактор, визуальный инструмент для определения ограничений, визуальный редактор форм и C++ библиотеку, которая позволяет быстро определить правила генерации.

Редактор для диаграмм характеристик и генератор, который создает метамодель предметно-ориентированного языка реализованы как плагин-инструменты в ядре QReal (на Рис. 5 обозначен как "domainAnalysis"). Язык для описания диаграмм характеристик сам по себе является предметно-оринетированным языком для области анализа предметной области, поэтому он был реализован также с помощью метаредактора QReal. Таким образом, после создания модели предметной области пользователю остается все лишь нажать кнопку "Генерировать метамодель" в интерфейсе QReal, и получить готовый редактор метамодели визуального языка. Затем генератор метаредактора используется для генерации еще одного плагина QReal, который нужен для получения визуального редактора для созданного языка. Генератор реализован вручную на языке C++ с использованием библиотеки Qt. В компоненте "appFamily" реализуется генератор кода для каждой предметной области.

Чтобы реализовать генератор кода приложений, необходимо сначала проанализировать код и выделить те части, которые будут изменяться. Такие части помечаются, и создается шаблон кода. Пример шаблонного кода представлен на Рис. 6. В примере строчки, которые изменяются, отмечаются символом "@", т.е. строчки которые создают пункт меню "Edit" и его выпадающие пункты. Неизмененные части помещаются в предметно-ориентированную библиотеку. Затем генератор принимает на вход созданные шаблоны кода и изменяет их в зависимости от данных, полученных из модели. Например,

если в модели указано, что есть пункт меню ”Edit”, то генератор изменяет данный шаблон путем замены нужных строчек. Такой код дальше связывает и конфигурирует большой каркас приложений. Подразумевается, что кода, который будет переделан в шаблоны, немного. Основной и содержательный код будет помещен в библиотеку и будет только вызываться в случае необходимости.

Таким образом, весь предметно-ориентированный код, который накоплен компаниями, не нужно переписывать в шаблоны.

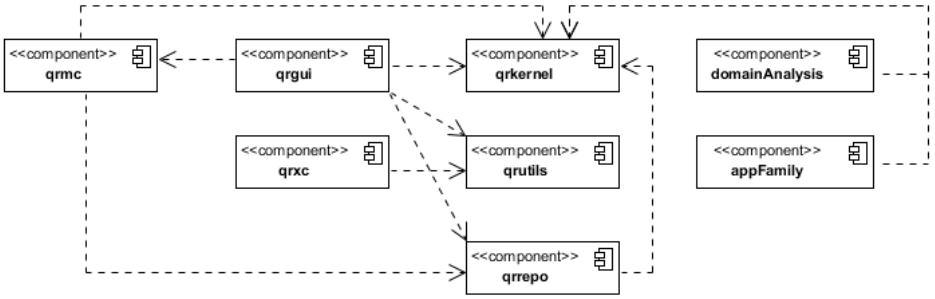

Рис. 5: Архитектура инстурмента QReal.

```
public class Editor extends JFrame {
    JTextArea area = new JTextArea(0, 0);
    JScrollPane scrolPane = new JScrollPane(area);
    public JPopupMenu popmenu = new JPopupMenu();

    @JMenu menu = new JMenu("Edit");@
    @JMenuItem menuItem = new JMenuItem("copy");@
    @JMenuItem menucut = new JMenuItem("cut");@
    @JMenuItem menupaste = new JMenuItem("paste");@
```

Рис. 6: Пример шаблонного кода.

Для апробации подхода было реализовано модельное приложение для удаленного управления с мобильного телефона разными моделями роботов — “Joystick”. Обоснованием для реализации такого рода приложения было то, что для управления разными моделями роботов требуются разные элементы управления. Например, одна модель может управляться только двумя клавишными панелями, но другая — одной клавишной панелью и двумя кнопками. Такое модельное приложение было реализовно на языке C# для платформы Windows Phone. Также было реализовано на языке Java модельное приложение —

“TextEditor”. Различие в конфигурациях различных текстовых редакторов обуславливается теми характеристиками, которыми обладает данный конкретный текстовый редактор.

4. Апробация подхода

4.1. Семейство приложений для удаленного управления роботами

На предметно-ориентированном языке для описания диаграмм характеристик была описана диаграмма предметной области для удаленного управления роботами с мобильного телефона. Соответствующая модель предоставлена на Рис. 7.

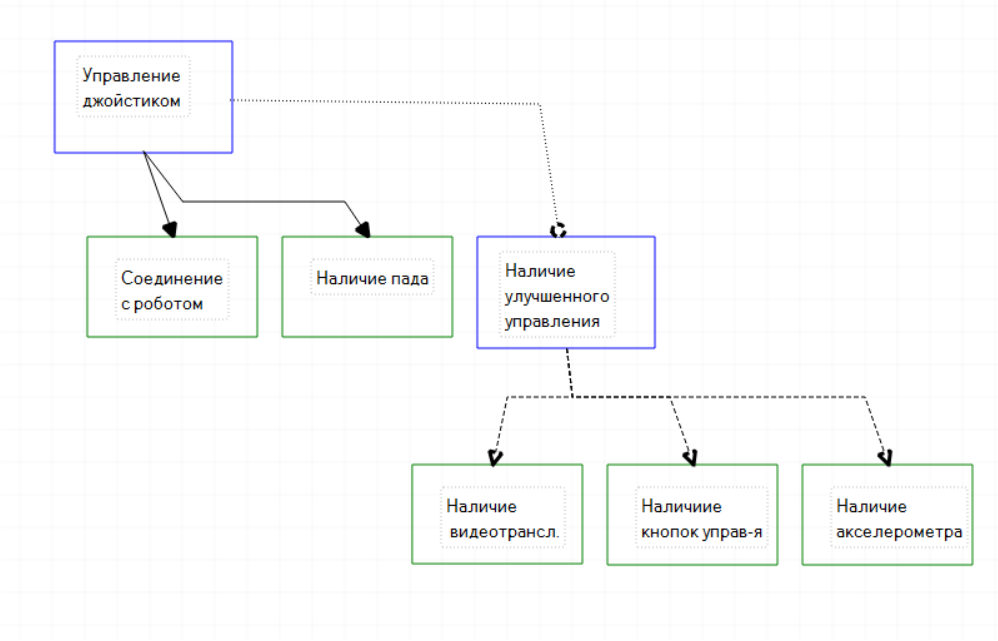

Рис. 7: Модель характеристик для удаленного управления роботами с мобильного телефона.

На модели характеристик, представленной выше, характеристики “Соединение с роботом” и “Наличие пада” являются обязательными для реализации в любом приложении в этом семействе. Группа характеристик “Наличие улучшенного управления” является опциональной, но в то же время в этой группе характеристиками улучшенного управления являются “Наличие видеотрасляции”, “Наличие кнопок управления”, “Наличие акселерометра”, между которыми существует альтернатива в реализации, но одна из них должна быть реализована, если есть улучшенное управление.

По этой модели характеристик была сгенерирована метамодель предметно-ориентированного языка, который нужен для описания различных конфигураций приложений в этом семействе. Сгенерированная метамодель предоставлена на Рис. 8.

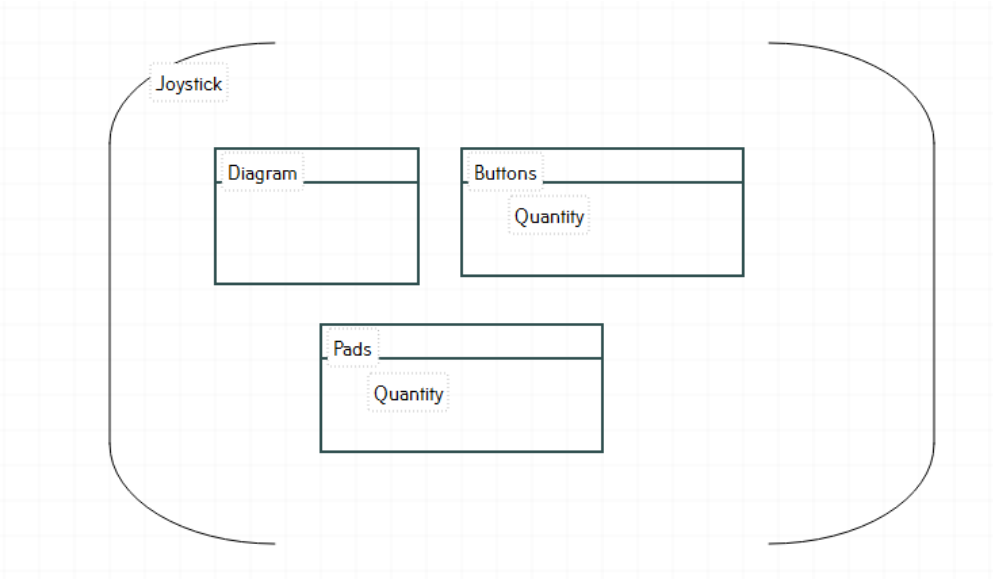

Рис. 8: Метамодель визуального языка для семейства приложений по управлению роботами.

Как можно увидеть, метамодель получилась очень простая. На этом шаге мы можем доработать нашу метамодель и обозначить, что сущности “Buttons” и “Pads” имеют свойство “Quantity”, т.е. кнопки и клавишные панели управления имеют свойство “количество”. На этом же этапе в редакторе форм мы задаем визуальные представления для этих сущностей. Следующим этапом является генерация визуального предметно-ориентированного языка средствами инструмента QReal. Пример сгенерированного визуального языка предоставлен на Рис. 9. В примере можно видеть, что в визуальном редакторе мы можем специфицировать свойство “Quantity” для определения количества управляющих элементов для конкретной реализации приложения под конкретную модель робота, явно указав значение этого свойства.

Как можно видеть на Рис. 9, визуальные формы для элементов предметно-ориентированного языка соответствуют смыслу, которые представляют эти характеристики, т.е. для сущности “Кнопка” бы-

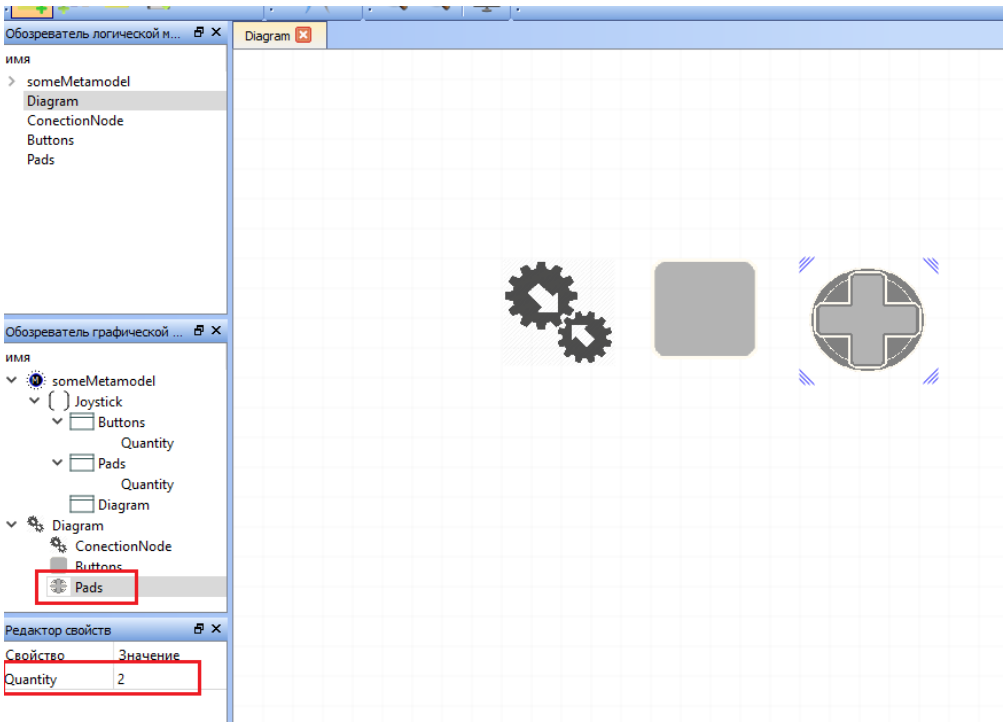

Рис. 9: Визуальный редактор предметно-ориентированного языка для области удаленного управления роботами.

Рис. 10: Снимки экрана сгенерированного приложения в семействе пультов для удаленного управления роботами.

ло загружено изображение, отображающее кнопку. Также, мы задали количество клавишных панелей управления — 2. Предметно-ориентированный генератор считывает эту информацию из модели и подставляет в шаблон кода для приложения. Поэтому сгенерируется код на C# приложения, реализующий именно две клавишные панели управления. Также для этого примера мы указали, что кнопок в целевом приложении тоже будет две. Снимки экрана сгенерированного приложения представлены на Рис. 10.

Можно сказать также, что этот пример довольно простой, чтобы продемонстрировать все возможности представленного подхода.

4.2. Семейство приложений текстовых редакторов

В качестве следующей предметной области для апробации данной технологии было рассмотрено семейство текстовых редакторов. Текстовые редакторы обладают вариативностью, которая связана с их назначением и применением. Например, в приложении ”Записки” в Windows 10 отсутствует характеристика для сохранения записки в файл. Как и в предыдущем примере, модель предметной области была описана на предметно-ориентированном языке. Такая модель представлена на Рис. 11.

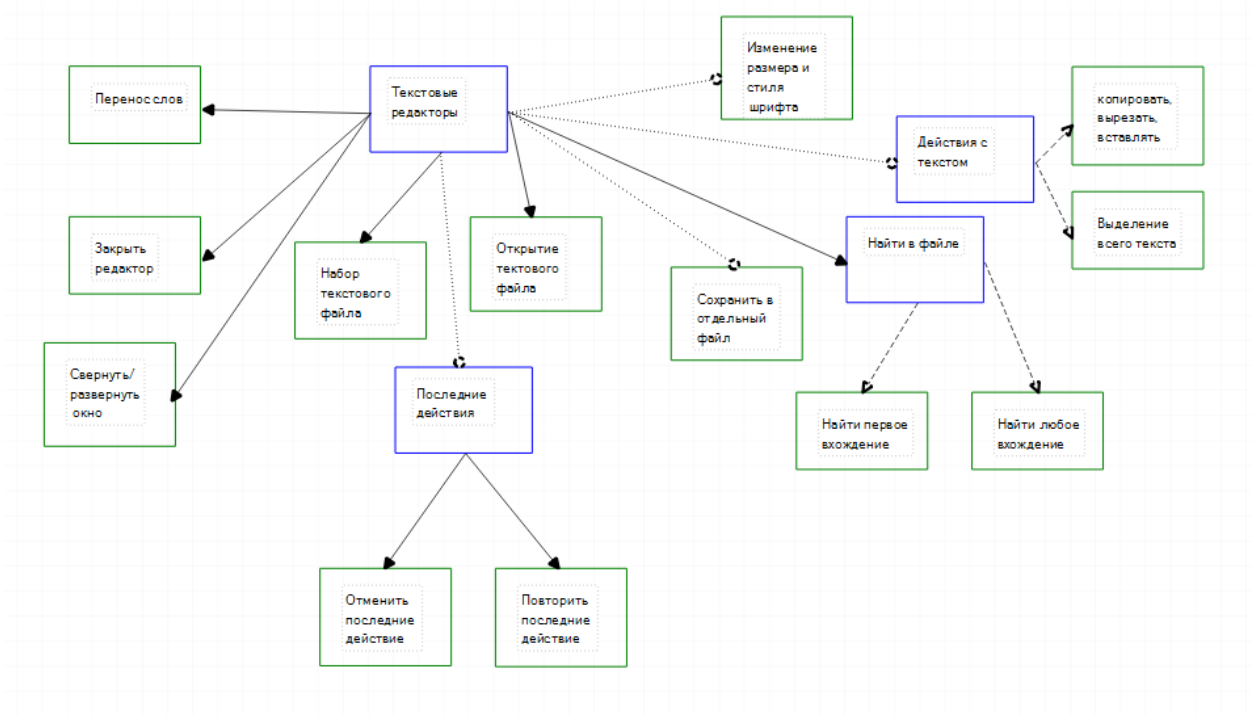

Рис. 11: Модель предметной области для текстовых редакторов.

Как можно увидеть, для текстовых редакторов опциональными характеристиками могут являться “Перенос слов”, “Изменение размера и стиля шрифта”, “Действия с текстом” и др. Если мы захотим реализовывать функционал, связанный с последними действиями пользователей (на модели “Последние действия”), то мы обязательно должны реализовать отмену и повтор последнего действия пользователя. По такой модели характеристик была сгенерирована метамодель предметно-ориентированного визуального языка. На Рис. 12 представ-

лена доработанная вручную эта метамодель.

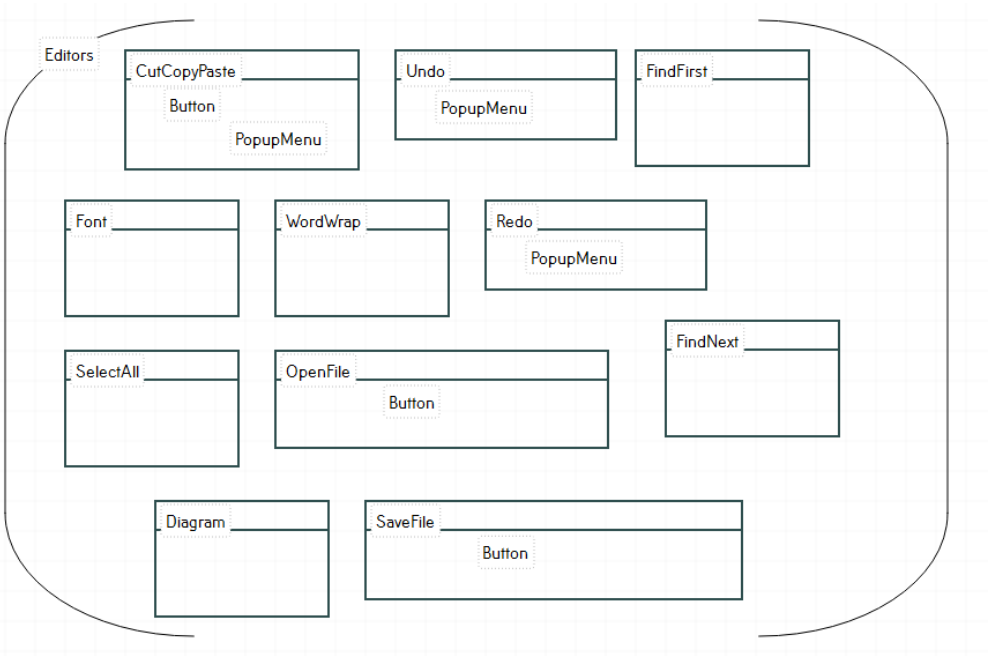

Рис. 12: Метамодель визуального предметно-ориентированного языка для семейства текстовых редакторов.

Дополненными вручную свойствами сущностей языка являются “Button”, обозначающее то свойство, что данная функциональность будет доступна через кнопки на панели инструментов, и “PopupMenu”, обозначающее, что эта функциональность будет доступна через выпадающее меню по нажатию правой клавиши мыши на рабочей области. Оба этих дополнительных свойства являются булевыми значениями: либо такая возможность реализуется, либо нет.

По такой метамодели был сгенерирован визуальный редактор для задания требуемых конфигураций текстовых редакторов. Снимок экрана такого редактора представлен на Рис. 13.

Как видно из снимка экрана на Рис. 13, мы выставляем свойства у элемента “Cut,copy,paste” “Button” — “Истина”, “PopupMenu” — ”Ложь”. А также требуем, например, чтобы была реализована функциональность изменять шрифт (на Рис. 13 обозначена как ”Font”). По этой модели был сгенерирован код на Java, который реализует сооветствующие функциональности. Скришоты приложения представлены на Рис. 14.

Можно видеть, что в выпадающем меню нет функциональностей

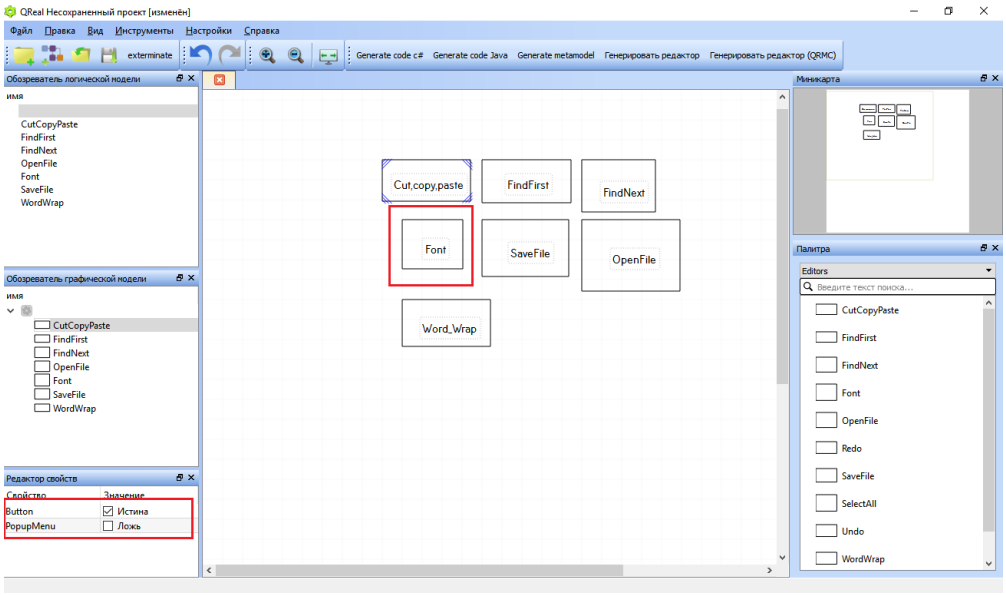

Рис. 13: Визуальный редактор для задания конфигураций текстовых редакторов.

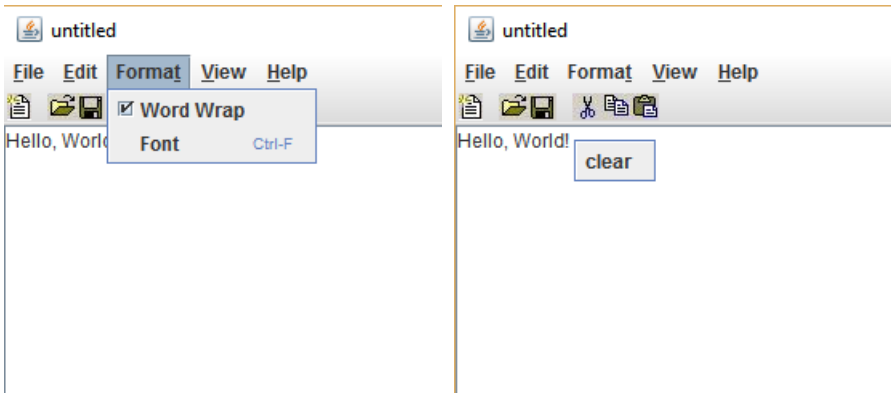

Рис. 14: Пример сгенерированного приложения по модели.

“cut,copy,paste”, т.е. нет возможности для вырезания, копирования и вставки. Но эти возможности доступны по кнопкам на панели инструментов.

Можем изменить модель и получить другое приложение. На Рис. 15 представлена измененная модель, в которой функциональность по вырезанию, копированию и вставке текста доступна как через кнопки на панели инструментов, так и из выпадающего меню. Еще можно заметить, что мы вообще убрали функциональность по изменению шрифта. Снимки экрана нового сгенирированного приложения представлены на Рис. 16.

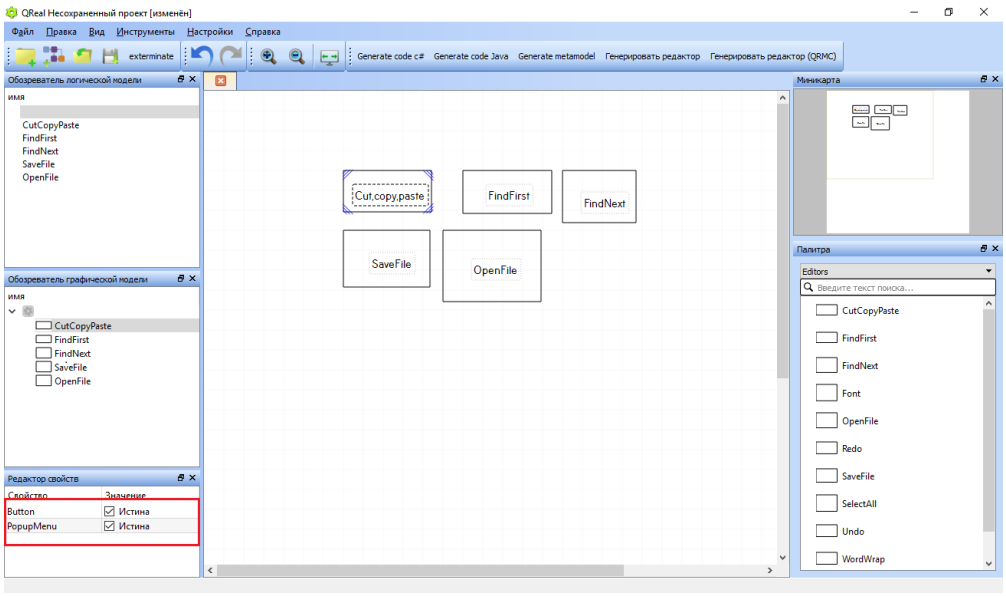

Рис. 15: Пример измененной модели в визуальном редакторе для задания конфигураций текстовых редакторов.

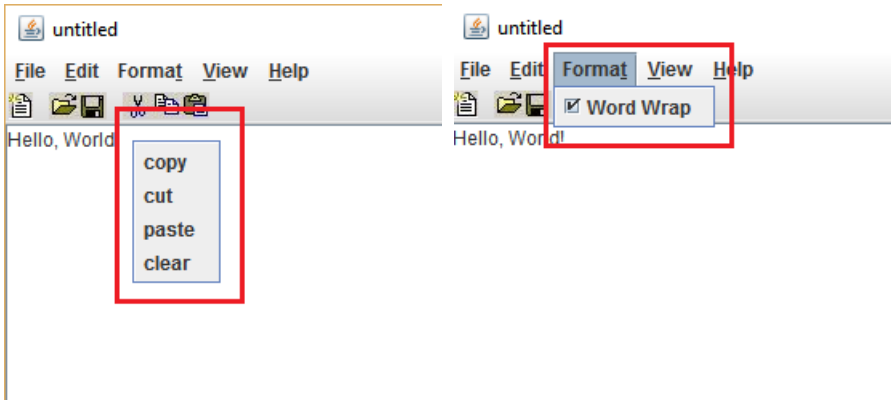

Рис. 16: Пример сгенерированного приложения по измененной модели.

Данный пример тоже довольно простой, но позволяет представить, что данная технология доступна для любого рода приложений на различных языках программирования, будь то C# или Java.

Заключение

В рамках данной квалификационной работы были достигнуты следующие результаты.

1. Выполнен обзор и проведен анализ различных генерационных подходов к созданию семейств приложений.

2. Предложен метод переиспользования ПО, основанный на построении модели характеристик и генерации по ней метамодели визуального языка, который используется для конфигурирования и описания правил композиции переиспользованных компонентов целевого приложения.

3. Инструментальная поддержка предложенного метода реализована на основе metaCASE-инструмента QReal и состоит из следующих компонентов: редактор модели характеристик и генератор метамодели визуального языка.

4. Проведена апробация представленного подхода на примере создания семейства приложений для удаленного управления роботами разных моделей с мобильного телефона, а также на примере создания модельного семейства приложений текстовых редакторов. Для каждой из этих предметных областей были реализованы: библиотека переиспользуемых компонентов, модель предметной области, редактор визуального языка, примеры приложений в области.

5. Результаты данной работы представлены на научно-практической конференции студентов, аспирантов и молодых ученых Северо-Запада “Современные технологии в теории и практике программирования” от 26 апреля 2016 года.

6. Результаты данной работы приняты программным комитетом для представления на семинаре SYRCoSE 2016 года.

Список литературы

[1] Rugaber S. Domain analysis and reverse engineering // White Paper, January. 1994.

[2] Mernik M., Heering J., Sloane A. M. When and how to develop domain-specific languages // ACM computing surveys (CSUR). 2005. Vol. 37, no. 4. P. 316–344.

[3] Кознов Дмитрий Владимирович. Основы визуального моделирования // М.: Изд-во Интернетуниверситета информационных технологий, ИНТУИТ. ру, БИНОМ, Лаборатория знаний. 2008.

[4] Prieto-Diaz R. Domain analysis for reusability // Software reuse: emerging technology / IEEE Computer Society Press. 1988. P. 347–353.

[5] Ferré X., Vegas S. An evaluation of domain analysis methods // 4th CASE/IFIP8 International Workshop in Evaluation of Modeling in System Analysis and Design / Citeseer. 1999. P. 2–6.

[6] Гудошникова А. А. Генерация метамодели языка по модели предметной области в проекте QReal, – ВКР бакалавра. 2014.

[7] Arango Guillermo. Domain analysis methods // Software Reusability. 1994. С. 17–49.

[8] DARE: Domain analysis and reuse environment / W. Frakes, R. Prieto, C. Fox et al. // Annals of Software Engineering. 1998. Vol. 5, no. 1. P. 125–141.

[9] Taylor R. N., Tracz W., Coglianese L. Software development using domain-specific software architectures // ACM SIGSOFT Software Engineering Notes. 1995. Vol. 20, no. 5. P. 27–38.

[10] Feature-oriented domain analysis (FODA): Tech. Rep.: / K. C. Kang, S. G. Cohen, J. A. Hess et al.: DTIC Document, 1990.

[11] FORM: A feature-; oriented reuse method with domain-; specific reference architectures / Kyo C Kang, Sajoong Kim, Jaejoon Lee [и др.] // Annals of Software Engineering. 1998. Т. 5, № 1. С. 143–168.

[12] Falbo Ricardo de Almeida, Guizzardi Giancarlo, Duarte Katia Cristina. An ontological approach to domain engineering // Proceedings of the 14th international conference on Software engineering and knowledge engineering / ACM. 2002. С. 351–358.

[13] Simos Mark, Anthony Jon. Weaving the model web: A multi-modeling approach to concepts and features in domain engineering // Software Reuse, 1998. Proceedings. Fifth International Conference on / IEEE. 1998. С. 94–102.

[14] FAMILIAR-project. 2012. URL: https://github.com/FAMILIAR-project.

[15] pure::variants. URL: http://www.pure-systems.com.

[16] Czarnecki Krzysztof. Generative programming: Principles and techniques of software engineering based on automated configuration and fragment-based component models. 1998.

[17] Estublier J., Vega G. Reuse and variability in large software applications // ACM SIGSOFT Software Engineering Notes. 2005. Vol. 30, no. 5. P. 316–325.

[18] An approach and framework for extensible process support system / Jacky Estublier, Jorge Villalobos, LE Anh-Tuyet [и др.] // Software Process Technology. Springer, 2003. С. 46–61.

[19] Chen Lianping, Babar Muhammad Ali. A systematic review of evaluation of variability management approaches in software product lines // Information and Software Technology. 2011. Т. 53, № 4. С. 344–362.

[20] Denger Christian, Kolb Ronny. Testing and inspecting reusable product line components: first empirical results // Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering / ACM. 2006. С. 184–193.

[21] The Variability Model of The Linux Kernel. / Steven She, Rafael Lotufo, Thorsten Berger [и др.] // VaMoS. 2010. Т. 10. С. 45–51.

[22] Kelly Steven, Tolvanen Juha-Pekka. Domain-specific modeling: enabling full code generation. John Wiley & Sons, 2008.