САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Гудошникова Анна Андреевна

Генерация метамодели языка по модели предметной области в проекте QReal

Бакалаврская выпускная работа

Допущена к защите

Зав.кафедрой:

д.ф.-м.н., профессор А.Н.Терехов

Научный руководитель:

ст. преподаватель Литвинов Ю.В.

Рецензент:

ст. преподаватель Луцив Д.В.

Санкт-Петербург

2014

SAINT-PETERSBURG STATE UNIVERSITY
Mathematics&Mechanics Faculty
Department of Software Engineering

Gudoshnikova Anna

Generation of language metamodel by domain model in Qreal project

Bachelor's Thesis

Admitted for defence.
Head of the chair:
professor Andrey Terehov

Scientific supervisor:
Senior Lecturer Yurii Litvinov

Reviewer:
Senior Lecturer Dmitry Luciv

Saint-Petersburg
2014

Оглавление

Введение

Процесс разработки программного обеспечения всегда требовал больших затрат, в частности, на глубинное изучение предметной области. На этот этап требуются большие человеческие ресурсы и время. Чтобы не рассматривать всю предметную область целиком, можно сконцентрироваться только на одной какой-то ее части, таким образом сокращая сроки выполнения того или иного проекта. Визуальное программирование, которому сейчас уделяется большое внимание, позволяет упростить создание программы путем представления отдельных ее аспектов в виде диаграмм.

Такой подход реализован в так называемых Computer-Aided Software Engineering (CASE) системах [1]. Они представляют средства генерации кода и используются в качестве инструментов для разработки, анализа и проектирования программного обеспечения. В CASE-системах используются различные визуальные языки, которые соответствуют рассматриваемой предметной области. На этих языках строятся различные модели для описания компонент программного обеспечения.

Процесс разработки CASE-системы вручную трудоемок, поэтому существуют metaCASE-системы, или DSM-платформы, которые позволяют автоматизировать такой процесс. В рамках автоматизации процесса разработки CASE-системы, как правило, создаются при помощи metaCASE-систем. Такие системы позволяют сгенерировать CASE-систему вместе с описанием предметно-ориентированного языка, который будет использован в этой CASE-системе.

Язык, с помощью которого описывается предметно-ориентированный язык, или просто визуальный язык, называется метаязыком. На метаязыке строятся различные модели, которые описывают данный визуальный язык. Такие модели называются метамоделями.

На кафедре системного программирования СПбГУ ведется разработка metaCASE-системы QReal. Она предоставляет средства, с помощью которых можно сгенерировать код CASE-систем по описанию метамодели визуального языка, который будет встроен в будущую CASE-систему.

Для создания и описания метамоделей визуального языка необходимо разбираться в предметной области, в рамках которой он создается. Всегда возникают проблемы с пониманием некоторых терминов и связей в рассматриваемой предметной области, поэтому этап анализа предметной области является неотъемлемой частью всего процесса разработки. В настоящее время этот этап, как правило, происходит в неформальном виде, в результате чего все равно остается недопонимание между аналитиком и программистом. Из-за этого возникают неточные метамодели визуального языка, из которых возникают противоречия или выявляется некоторое недопонимание того, что в итоге нужно реализовать. Таким образом, это негативно сказывается на сроках выполнения проекта.

В связи с этим были разработаны некоторые формальные методы анализа предметной области, в рамках которых строятся модель или модели предметной области. М. Мерник в работе [2] объясняет, что модель предметной области должна описывать так называемые сходства и изменчивости понятий.

Изменчивость понятий (variabilities) позволяет точно определить, какая информация должна быть специфицирована в реализации конкретной системы. Сходства (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 молучаем 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.

Очевидной трудностью является сам процесс извлечения информации, необходимой для построения метамодели визуального языка. Чтобы преодолеть данную трудность, необходимо валидно представить модель предметной области. В методе Feature-Oriented Domain Analysis (FODA) [3] предметная область рассматривается в виде модели функциональностей, реализуемых в рамках этой предметной области. В данной работе предлагается использовать данный метод. Он был выбран в виду его простоты и способности решить поставленную задачу.

В настоящее время в большинстве существующих DSM-платформ и, в частности, в проекте QReal нет средств для анализа предметной области, а более конкретно, для создания модели предметной области, а также средств построения по этой модели метамодели визуального языка.

Постановка задачи

Целью данной работы является создание средства генерации метамодели предметно-ориентированного языка по модели предметной области, которая предварительно описывается экспертом предметной области. Для достижения этой цели были сформулированы следующие задачи.

1. Рассмотреть различные формальные методы анализа предметной области.
2. Создать язык для описания предметной области. Язык должен быть понятен пользователю, а также быть подходящим для описания любой предметной области.
3. Реализовать генерацию метамодели предметно-ориентированного языка на основе модели предметной области и встроить ее в проект QReal.
4. Рассмотреть конкретную предметную область и сгенерировать метамодель по модели этой предметной области.

Глава 1. Обзор существующих подходов

В основе качества и надежности поставляемого ПО лежит, в первую очередь, понимание той сферы деятельности, где предстоит разработать продукт. Такая сфера деятельности называется предметной областью. На глубинное изучение предметной области всегда требуются большие затраты, но тем не менее, этот этап является обязательным в процессе разработки программного обеспечения. Но что каждый понимает под анализом предметной области, остается неясным. На данный момент нет четкого и ясного определения, что такое «анализ предметной области». Фере [4] объяснил этот термин следующими способами.

- Процесс идентификации, организации и представления релевантной информации о предметной области.
- Процесс, при котором знания клиента/пользователя идентифицируются, конкретизируются и систематизируются.
- Деятельность, которая предваряет системный анализ.

В настоящее время анализ предметной области проходит как правило в неформальном виде. Это привносит свои трудности в правильное понимание всех терминов, сущностей и связей при объяснении аналитиком требуемой задачи программистам. В связи с этим, было разработано множество формальных методов для анализа предметной области, которые будут рассмотрены ниже. В рамках каждого из методов должна быть создана или неким образом описана модель предметной области, о которой

говорилось во введении, а также такая модель должна удовлетворять определенным критериям, которые представлены в работе Мерника [2].

Тем не менее, несмотря на разное понимание того, что такое «анализ предметной области», Аранго [5] показал, что все методы анализа предметной области придерживаются так называемого общего процесса получения модели предметной области. Схема этого процесса приведена на Рис.1.

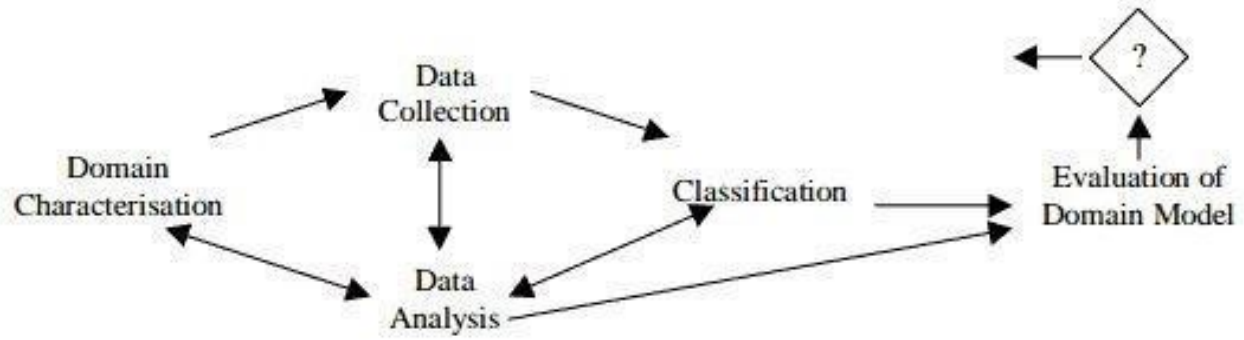

Рисунок 1. Схема общего процесса получения модели предметной области [5]

Этап характеристики предметной области (Domain characterisation) подразумевает анализ реализуемости проекта и фазу планирования. На этапе сбора данных (Data collection) выбираются источники релевантной информации в данной области, которые могут варьироваться от экспертов предметной области до различных документов и т.д. Этап анализа данных (Data analysis) называется вспомогательным моделированием. Целью этого этапа является выделение сходств и различий между используемыми элементами. На следующем этапе (Classification), или на этапе основного

моделирования, смоделированная на предыдущем шаге информация уточняется, выделяются целые блоки с похожими описаниями, добавляются новые описания к существующим или новым блокам, и затем эти блоки выстраиваются в обобщенную иерархию. На заключительном этапе (Evaluation of Domain Model) происходит оценка созданной модели предметной области, любые недочеты исправляются.

Формальные методы анализа различаются некоторыми техниками, которые применяются при вспомогательном моделировании, 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 (case-based technology)

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

- Фасеты (facets) или аспекты
- Функциональность (features)
- Возможности (capabilities)
- Требования (requirements)

Анализ предметной области также используется в задаче переиспользования [4]. В этом случае выбор метода для анализа предметной области будет зависеть и от типа объекта, который будет впоследствии переиспользован, и от задачи переиспользования. В качестве задачи переиспользования может рассматриваться, например, построение библиотеки переиспользуемых компонент.

Объекты, которые будут переиспользоваться, идентифицируются во время процесса анализа предметной области. Они могут быть:

- отдельными компонентами продукта или продуктом (software product reuse) [6, 7, 8, 9]
- процессом (software process reuse) [10]
- технологией (software technology reuse) [11]
- знаниями или опытом, которые были приобретены вследствие предыдущих проектов (software experience reuse) [12, 13]

В частности, метамодель предметно-ориентированного языка, о которой говорилось во введении, хранит в себе знания о предметной области, т.е. способствует переиспользованию знаний. При проектировании предметно-ориентированного языка методы анализа предметной области важны, так как способствуют лучшему переиспользованию знаний.

1.1 Подходы к анализу предметной области

Как говорилось выше, в силу трудностей, которые возникают при использовании неформальных методов анализа предметной области, были разработаны формальные методы анализа. Ниже дается описание некоторых таких формальных методов.

1) DARE (Domain Analysis and Reuse Environment) [14]. В рамках этого метода вся собранная информация об области собирается в так называемую книгу предметной области (domain book), которая содержит также и универсальную архитектуру для приложения в этой области, и библиотеку переиспользуемых компонент.

2) DSSA (Domain-Specific Software Architectures) [15]. Этот метод основан на сценариях. Пользователи описывают некоторые сценарии, которые будут в дальнейшем использоваться для разработки словаря данной области.

3) FODA (Feature-Oriented Domain Analysis) [3] Метод основан на выделении той функциональности, которую нужно будет реализовать в рамках рассматриваемой области.

4) FAST (Family-Oriented Abstractions, Specification and Translation) [16]. Этот метод анализирует сходства и различия между продуктами в рамках одной линейки продуктов.

5) FORM (Feature-Oriented Reuse Method) - расширение метода FODA[17]. В рамках метода рассматривается 4х-уровневая модель характеристик (feature model).

6) ODE (Ontology-based Domain Engineering) [18]. Метод соединяет онтологии и объектно-ориентированную технологию. Онтологии содержат понятия и связи, их определение, свойства и ограничения, выраженные в виде аксиом. Этот подход направлен на получение библиотеки объектов из описанной онтологии. Библиотека строится на создании взаимо-однозначного соответствия между понятиями, отношениями, свойствами и ролями в онтологии и сущностями, использующимися в объектно-оринетированной технологии: классами, ассоциациями, атрибутами и ролями.

7) ODM (Organization-Domain Modeling) [19]. В данном методе создаются раздельные, но взаимосвязанные концептуальные модели, в результате получается так называемая сеть моделей (model web). В такой сети моделей внутримодельные отношения имеют свою семантику. Фокусировка происходит на определенные ключевые понятия предметной области внутри такой сети моделей. Такие понятия определяют, какие модели рассматриваются как центральные концептульаные модели (central concepts models) для данной предметной области. Другие же модели призваны адекватно оценивать и объяснять изменчивость (variability) этих концепутальных моделей. Такие вспомогательные модели играют роль моделей функциональностей (feature models) внутри сети моделей для предметной области.

1.2 Подходы для описания модели предметной области

Для описания модели предметной области существует множество подходов, в частности, в каждом из вышеперечисленных формальных методов используется свой подход. Далее рассматриваются только те виды моделей, которые применительны к данной работе.

1.2.1 Модель функциональностей

Метод FODA [3] используется нами для генерации метамодели предметно-ориентированного языка в терминах предметной области по модели функциональностей (feature model) этой предметной области.

Модель функциональностей включает в себя описание функциональности, которой должен обладать будущий программный продукт. Согласно Мернику [2], такая модель должна включать в себя:

- диаграмму функциональностей, представленную в виде декомпозиции всей функциональности и ее характер, т.е. является ли данная характеристика обязательной к реализации, альтернативой, т.е. существует выбор между реализуемыми характеристиками, но одна должна быть выбрана обязательно в качестве реализуемой, или же дополнительной характеристикой, которая не обязательно реализуется;
- определение семантики функциональностей;

- правила для задания композиции функциональностей, т.е. является ли данная комбинация характеристик валидной или они не согласуются друг с другом;
- возможно, некоторые причины, почему выбирается та или иная характеристика.

Пример модели функциональностей [2] представлен на Рис.2.

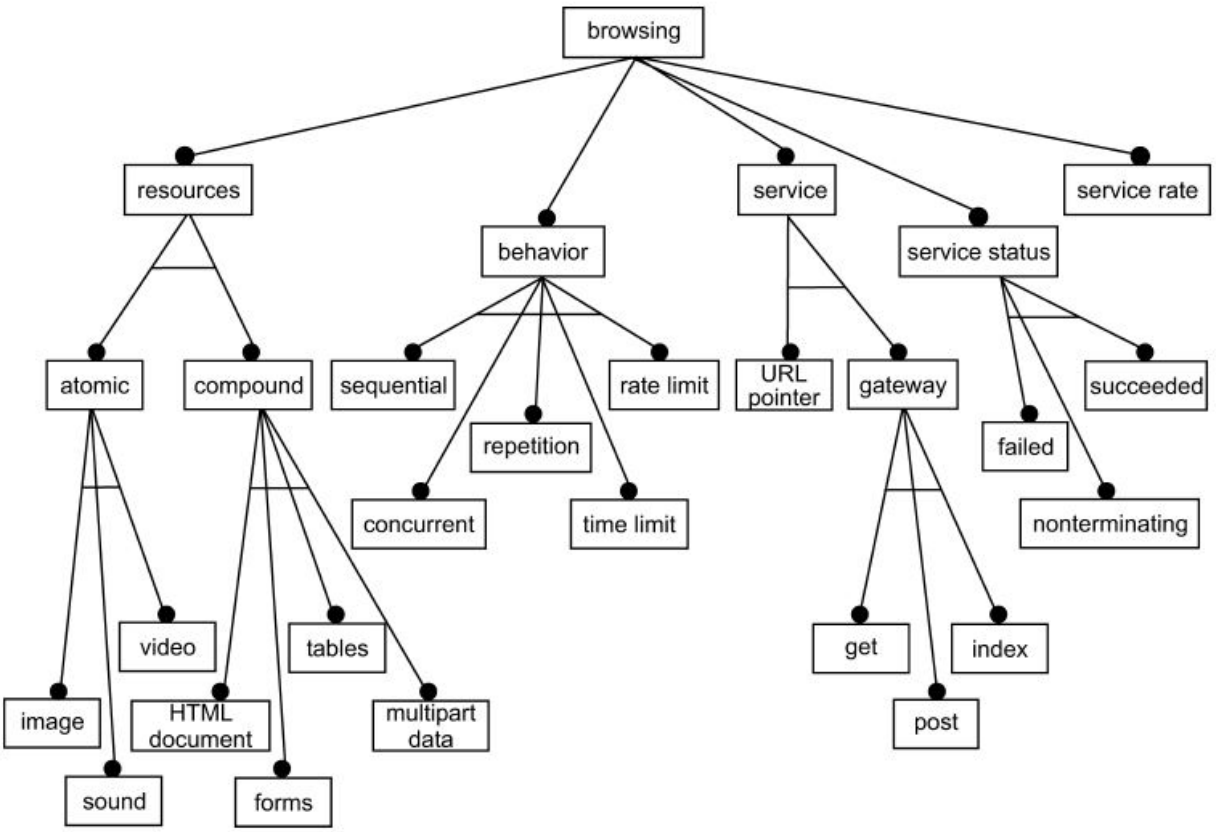

Рисунок 2. Диаграмма функциональностей для веб-браузера [2]

В примере модели функциональностей, представленном выше, рассмотрена предметная область веб-браузера. Предметная область в

данном случае состоит из ресурсов (resources), поведения браузера (browsing behavior), и сервисов (services), а именно их тип (type), статус (status) и режим работы (rate). Связь с поперечной полосой означает альтернативность, а без нее – обязательно к реализации. В представленной модели нет связи, обозначающей дополнительную характеристику.

1.2.2 Модель сущность-связь

Модель предметной области можно также описать и с помощью обычной диаграммы сущность-связь (entity-relationship). Такая модель показывает, какие сущности есть в предметной области, и как они связаны между собой. Пример такой модели представлен на Рис.3.

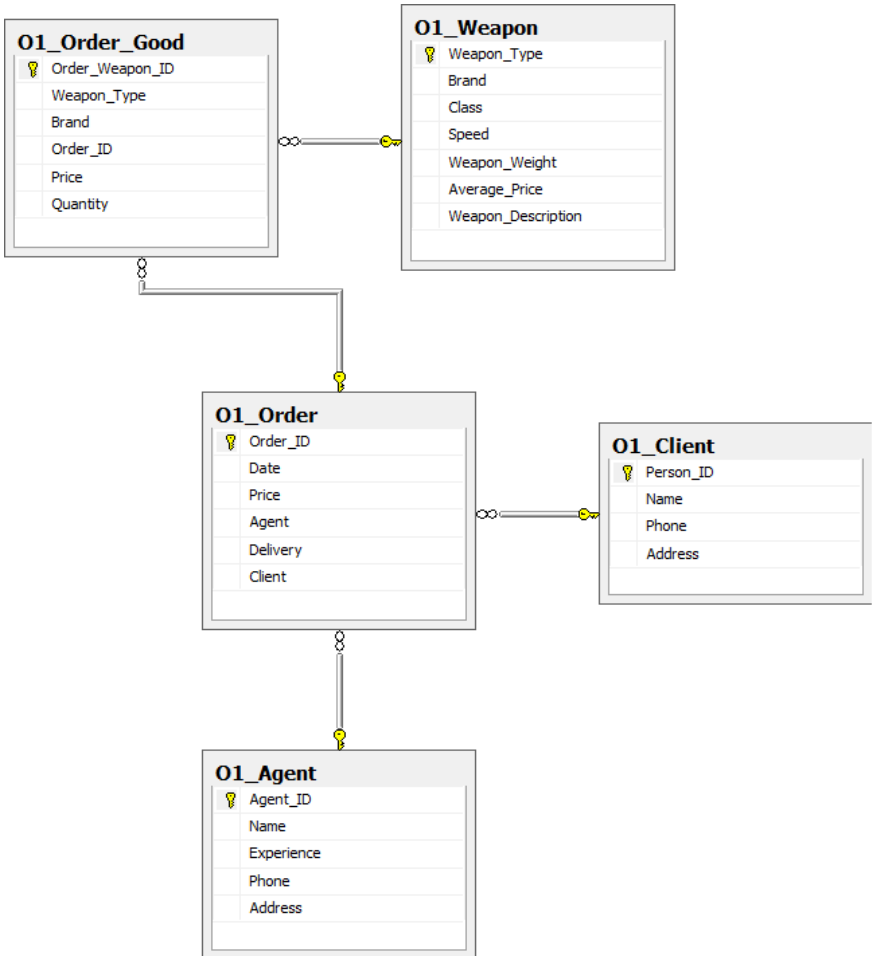

Рисунок 3. Диаграмма сущность-связь применительно к области продажи оружия

В этом примере мы можем понять, какие сущности существуют в предметной области, связанной с оружейным магазином, а также увидеть как эти сущности связаны между собой.

Чтобы наиболее полно и подробно описать предметную область, возможно совмещение данных моделей.

1.3. REAL-IT

Идея использовать модели предметной области для генерации кода не нова. Так, на базе объектно-ориентированного CASE-пакета REAL и одноименной методологии [20] на кафедре системного программирования СПбГУ было создано модельно-ориентированное технологическое решение REAL-IT [21], предназначенное для разработки и сопровождения промышленных информационных систем. Основным принципом данного решения является построение визуальных моделей предметной области и генерация кода по этим моделям.

В REAL-IT основной моделью разрабатываемой системы является модель данных. Частью процесса моделирования данных является построение схемы данных. В REAL-IT для моделирования схемы баз данных используется модель классов CASE-пакета REAL, являющаяся расширением модели классов UML [22]. При описании схемы данных не используются методы класса. Схема данных хоть и использует объектно-ориентированную нотацию, по факту является диаграммой “сущность-связь”, расширенной отношением тип-подтип. Про модель “сущность-связь” говорилось выше. Так как семантика предметной области обычно содержит ограничения на возможные связи, то в REAL-IT для описания подобных ограничений используется специальная визуальная нотация, разработанная на основе диаграмм кооперации UML.

Одной из задач в системе REAL-IT было моделирование пользовательского интерфейса [23]. Известно, что одной модели данных

могут соответствовать различные интерфейсы в зависимости от взгляда на систему. Моделью интерфейса называется представление данных. При построении модели интерфейса происходит выборка объектов и их атрибутов из модели данных. Таким образом, одной модели данных может соответствовать множество различных моделей интерфейса.

В REAL-IT реализован генератор пользовательского интерфейса, основанный на выделении необходимой информации из модели данных. Эти генераторы успешно использовались при создании ряда промышленных информационных систем.

Глава 2. Описание языка

Для описания модели предметной области требуется специализированный язык. К такому языку выдвигаются определенные требования.

- Любая предметная область должна быть описываемой на этом языке.
- Язык должен быть понятен пользователю, а именно эксперту предметной области, который и будет использовать этот язык.

Функциональность может детализироваться, например, обобщенная характристика “Создавать игру” может включать в себя задание времени и места проведения игры. В созданном языке описание любой характеристики задается внутри прямоугольника (Рис. 4), если характеристика является общей, то прямоугольник - синий, если детализированная - зеленый.

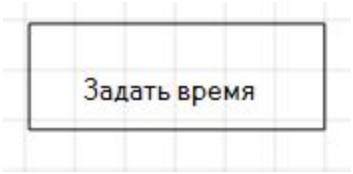

Рисунок 4. Пример определения характеристики

Как говорилось выше, язык должен отражать то, какая функциональность является обязательной к реализации, между какими характеристиками существует выбор, и какая функциональность является дополнительной, т.е необязательной к реализации.

В созданном языке выше описанное требование определяется связями между прямоугольниками, в которых формулируется та или иная характеристика. Так, обязательными к реализации характеристиками являются те, к которым идет непрерывная направленная стрелка (Рис. 5(а)). Если между характеристиками существует выбор, то стрелки должны идти от одного и того же прямоугольника с описанной обобщенной функциональностью к детализированным характеристикам, и такие стрелки являются пунктирными (Рис. 5(б)). С помощью этих связей можно создать любое логическое отношение между характеристиками. К примеру, если требуется реализовать две функциональности, и для каждой существует выбор, то можно создать две обобщенные функциональности, соединенные непрерывной связью (Рис. 5(а)), и от каждой из них провести две пунктирных стрелки (Рис.5(б)) к соответствующим вариантам. К дополнительным характеристикам идет пунктирная линия с кружком на конце (Рис. 5(в)).

(а) (б) (в)

Рисунок 5. Типы связей

Как было замечено в начале этой главы, язык должен удовлетворять двум требованиям. Первое требование было, что на языке можно описать любую предметную область. Действительно, если предметную область рассматривать как набор функциональностей, которые реализуемы в ней, то на созданном языке можно описать любую предметную область, так как в каждой области есть функциональности, которые должны быть обязательно в ней реализованы, также в ней всегда присутствует элемент альтернативы, т.е. предоставляется некий выбор между характеристиками, и даже рассматриваются некие дополнительные характеристики, которые являются необязательными, но также релевантно отражают функциональность в данной предметной области. Например, такие функциональности могут рассматриваться в предметной области, но им может не уделятся особого внимания.

Второе требование зыключалось в том, что язык должен быть понятен пользователю. Это действительно так, потому как созданный язык довольно прост и понятен для неспециалиста в моделировании, а значит будет понятен любому эксперту, который представляет конкретную

предметную область.

Метамодель созданого языка представлена на Рис. 6. По этой метамодели средствами QReal был создан редактор, в котором можно создавать модели предметной области на описанном выше языке.

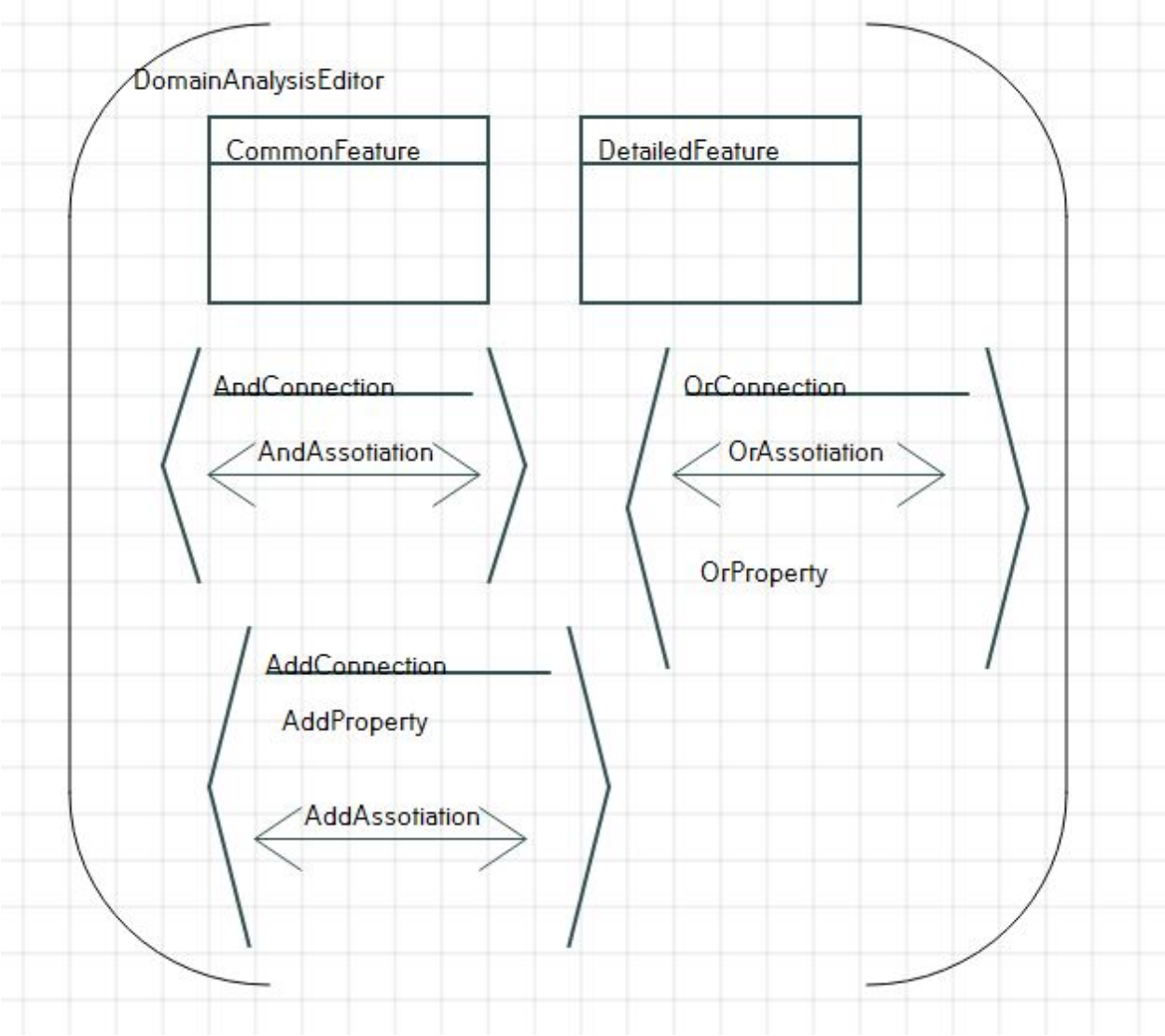

Рисунок 6. Метамодель созданного языка

Глава 3. Описание генератора

3.1 QReal

QReal - платформа предметно-ориентированного моделирования (DSM-платформа), позволяющая быстро и удобно разрабатывать новые визуальные редакторы, ориентированные на специализированные визуальные языки, описывая их метамодели на метаязыке.

3.1.1 Архитектура QReal

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

3.1.2 Метамодель языка и связь ее элементов с репозиторием QReal

Метамодель описывает синтаксические правила языка. Метамодели языков обрабатываются генератором редакторов QReal, генерируя код редактора на C++. Такой код компилируется вместе с кодом QReal. Для каждого описанного редактора на выходе получается подключаемый

модуль, являющийся файлом динамической библиотеки. Такая библиотека используется для визуализации редактора.

Элементы модели (или метамодели) хранятся в репозитории. Такие 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 должны ссылаться на один и тот же репозиторий, но обращаться к нему по разным интерфейсам.

Доступ к репозиторию осуществляется по трем интерфейсам.

- ControlApi – интерфейс управления репозиторием. Никаких данных об элементах не предоставляет.
- GraphicalRepoApi – часть интерфейса для графической модели. Содержит методы для работы с чисто графическими свойствами объектов.
- LogicalRepoApi – интерфейс для логической модели. Здесь нет никаких графических свойств для элементов модели.

Логические и графические элементы хранятся в репозитории связанными.

3.1.3 Интерфейсы между подключаемыми модулями и пользовательским интерфейсом

Подключаемый модуль (или плагин-инструмент) изменяет пользовательский интерфейс в соответствии с функциональностью плагина-инструмента.

Взаимосвязь 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 работы плагина-инструмента.
- ToolPluginInterface – интерфейс плагина-инструмента. Позволяет пользовательскому интерфейсу получить список действий, выполняемых плагином, и информацию, как эти действия отражаются в пользовательском интерфейсе.

3.2 Генератор метамодели

Созданный генератор метамодели реализует интерфейс плагинов-инструментов в QReal.

Сущности, определяемые в метамодели визуального языка, должны быть заданы так же, как и в модели предметной области. Таким образом, генератор построен на алгоритме обхода модели по детализированным функциональностям, создавая соответствующие узлы языка в репозитории для метамодели.

Любой редактор, генерируемый по метамодели визуального языка, может быть задан посредством xml-описания этой метамодели. Поэтому необходимо знать, какие сущности должны принадлежать такой

метамодели, чтобы впоследствии описать их в xml-файле.

Такой список элементов-сущностей, принадлежащих метамодели, приводится ниже.

Diagram - корневой элемент метамодели описания языка. В его свойствах должна быть перечислена системная информация. Описывается тегом <diagram>

Editor - сущность-редактор. В свойствах этого элемента указывается его рабочее имя и имя, которое будет отображаться при загрузке нового редактора метамодели в среду. Содержит поле NodeName, которое должно содержать имя элемента, который будет являться корневым для метамоделей в создаваемом редакторе. Описывается тегом <editor> и вклдаывается в элемент <diagram>.

Node - сущность, которая описывает узел языка (сущность) в метамодели. Содержит свойства name (идентификатор узла) и displayedName (отображаемое на палитре элементов имя). Описание формы узла метамодели и возможность его масштабирования определяется свойством shape. В xml-файле описывается тегом <node> и вкладывается в элемент <editor>.

Edge - сущность, которая описывает связи или ассоциации в создаваемой метамодели. Содержит свойства name (идентификатор ассоциации) и displayedName (отображаемое имя ассоциации). Также есть свойства, задающие тип линии. Описывается тегом <edge> и вкладывается в <editor>.

Созданный генератор метамодели, создавая в репозитории сущности метамодели, заполняет ее свойства, чтобы по xml-файлу этой метамодели можно было правильно сгенерировать редактор.

Созданная генератором метамодель, возможно, будет требовать ручной доработки. Таким образом, будущий визуальный язык будет более точно описан.

Глава 4. Пример генерации метамодели

4.1 Построение модели функциональностей

Пример модели функциональностей на созданном языке можно увидеть на Рис. 7. Эта модель представляет спортивную предметную область, а именно, представлены функциональности, которые можно реализовать в конкретном футбольном приложении. В общем, футбольным экспертом описывается вся модель предметной области, связанной с футболом. Затем берется выборка из всех функциональностей для реализации определенного класса футбольных приложений. В примере рассматривается приложение, которое помогает людям собраться для игры в футбол. Как видно, функциональности вида “создавать игру”, “создавать команду” и другие аналогичные являются обязательными для реализации в этом приложении. Характеристика “регистрировать команду” является дополнительной. А при выборе позиции игрока мы можем выбрать из двух вариантов: “Выбор позиции вратаря или полевого игрока” (т.е. пользователю будет предоставлен выбор между тем, быть ему вратарем или полевым игроком) и “Выбор между нападением и защитой” (здесь пользователю представляется выбор между тем, быть ему в защите (т.е. быть защитником или вратарем), или же играть в нападении).

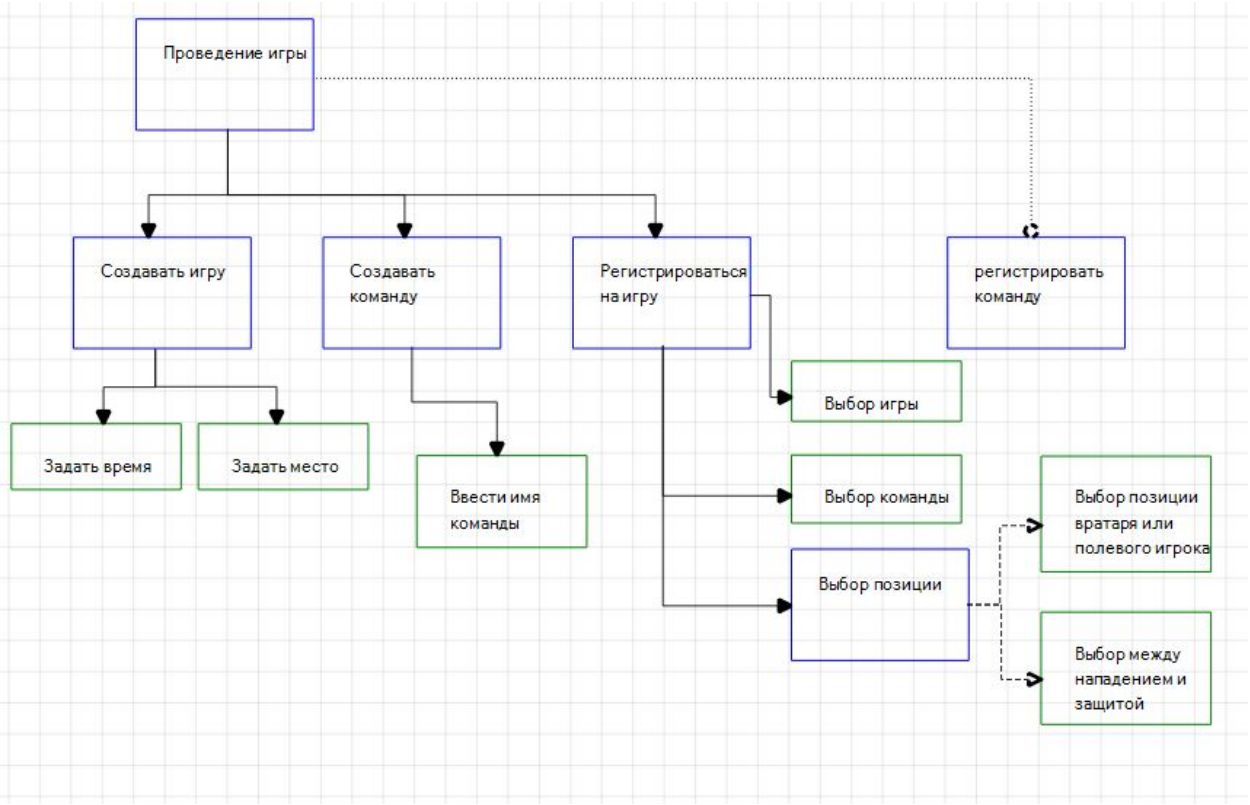

Рисунок 7. Модель функциональностей на созданном языке, описывающая сбор людей для игры в футбол

4.2 Генерация метамодели по построенной модели

По представленной модели была сгенерирована метамодель, которую можно увидеть на Рис.8.

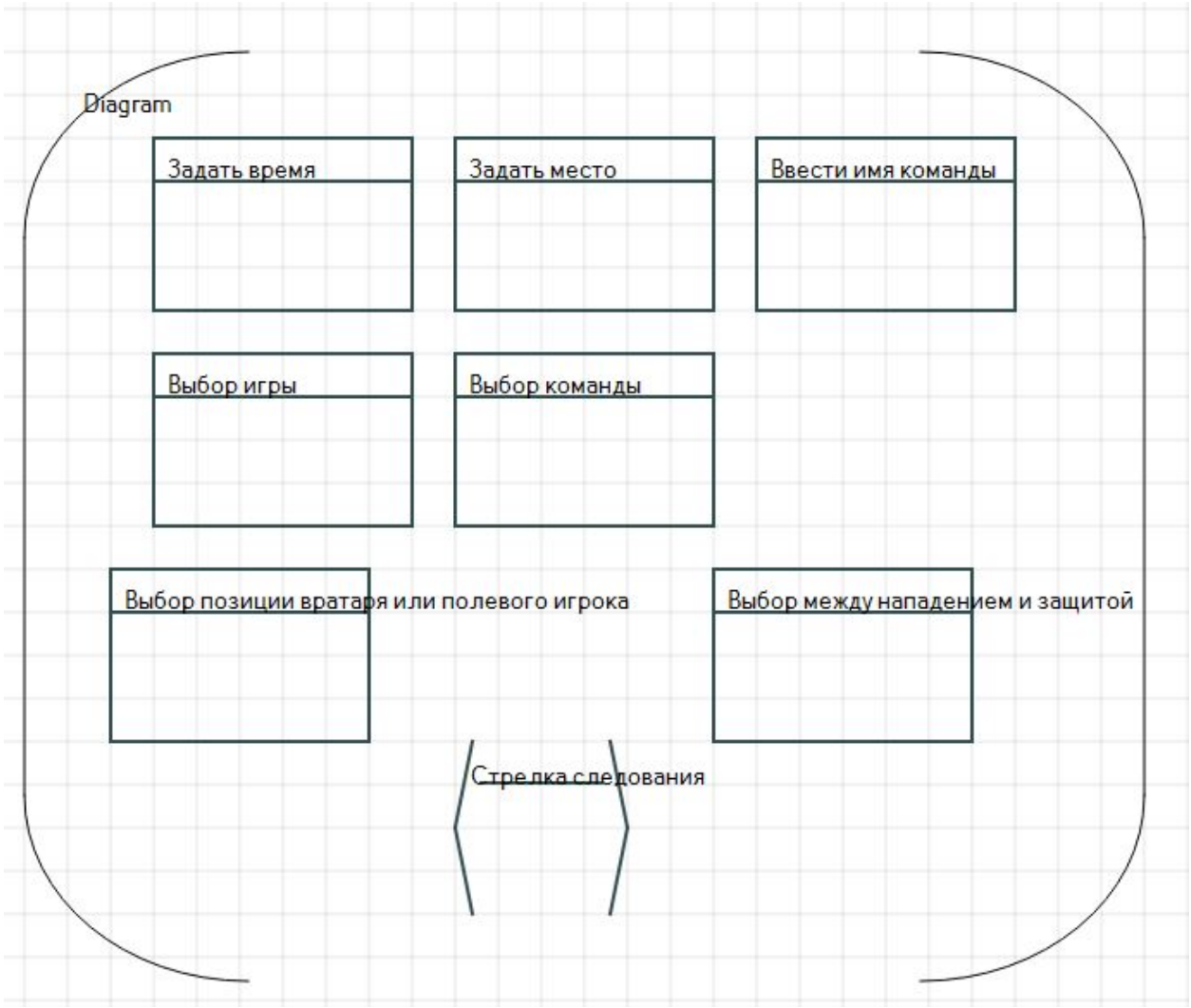

Рисунок 8. Метамодель языка разработки футбольных приложений, полученная из модели предметной области

Такая метамодель задает предметно-ориентированный язык для класса футбольных приложений. Этот язык оперирует терминами, которые приняты в данной предметной области. В дальнейшем, на таком языке описываются различные модели создаваемого приложения, и по ним генерируется код самого приложения. Пример моделей, описывающей

создаваемое приложение, представлен на Рис.9. А именно, представлены две диаграммы состояний, описывающих поведение приложения.

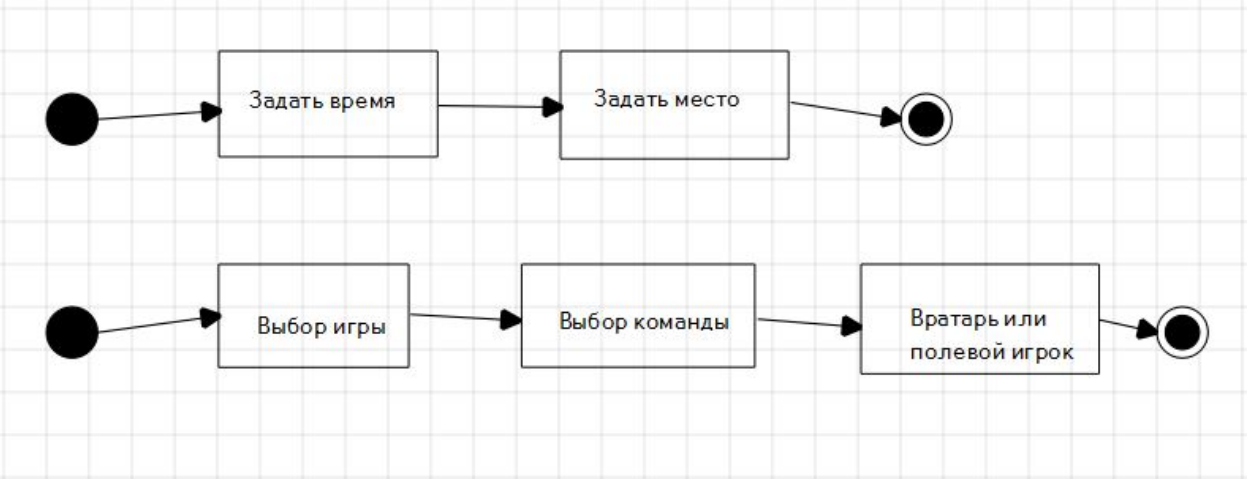

Рисунок 9. Пример диаграмм состояний, описывающих поведение приложения

Заключение

В рамках данной квалификационной работы были достигнуты следующие результаты.

1. Рассмотрены и изучены некоторые формальные методы анализа предметной области.
2. Создан язык, с помощью которого эксперт может описать конкретную предметную область.
3. В рамках технологии QReal реализован графический редактор, в котором можно строить модели предметной области на созданном метаязыке.
4. Реализован генератор метамодели, основанный на модели предметной области.
5. Рассмотрена конкретная предметная область и генерация метамодели предметно-ориентированного языка в этой предметной области по модели функциональностей этой области.

Список литературы

[1] Кознов Д.В. Основы визуального моделирования. Спб: Изд. Бином. Лаборатория знаний. 2008. 245 с.

[2] Mernik M. When and How to Develop Domain-Specific Languages, ACM Computing Surveys, Vol. 37, No. 4, December 2005.P. 316–344.

[3] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. 1990. Feature-oriented domain analysis (FODA) feasibility study. Tech. rep. CMU/SEI-90-TR-21. Software Engineering Institute, Carnegie Mellon University.

[4] X.Ferre, S.Vegas. An Evaluation of Domain Analysis Methods.

[5] G.Arango. Software Reusability, chapter 2. Domain analysis methods. P.17-49.Workshops M.E.Horwood, London 1994.

[6] J. Neighbors. The draco approach to constructing software from reusable components. IEEE Transactions on Software Engineering, Vol. SE-10, no. 5. P. 564-574, September 1984.

[7] R. Prieto-Díaz. Domain analysis for reusability. In Proceddings COMPSAC'87. P. 23-29, Tokio, Japan, October 1987.

[8] P.C. Cornwell. HP domain analysis: Producing useful models for reusable software. Hewlett-Packard Journal, article 5. August 1996.

[9] P.M. Maccario. The domain analysis integrated in an object oriented development methodology. In Proceedings of the Eighth Workshop on Institutionalizing Software Reuse, Columbus, Ohio, March 1997. Ohio State.

[10] C. Hollenbach and W. Frakes. Software process reuse in an industrial setting. Technical Report TR-96-04, Department of Computer Science, Virginia Tech., 2990 Telestar CT., Falls Church, VA 22042, January 1996.

[11] A. Birk. A knowledge acquisition method for domain analysis of software engineering technologies. IESE-Report 019.97, Fraunhofer IESE, 1997.

[12] V.R. Basili, L.C. Briand and W.M. Thomas. Domain analysis for the reuse of software development experiences. In Proceedings of the 19th Annual Software Engineering Workshop, NASA/GSFC. P. 11-24. NASA/GSFC. December 1994.

[13] S. Henninger. Accelerating the successful reuse of problem solving knowledge through the domain lifecycle. In Proceedings of the Fourth International Conference on Software Reuse. P. 124-133, Orlando, Florida, April 1996.

[14] Frakes, W., Prieto-Diaz, R., and Fox, C. 1998. DARE: Domain analysis and reuse environment. Annals of Software Engineering 5. P. 125–141.

[15] Taylor, R. N., Tracz, W., and Coglianese, L. 1995.Software development using domain-specific software architectures. ACM SIGSOFT Software Engineering Notes 20, 5. P. 27–37.

[16] Weiss, D. and Lay, C. T. R. 1999. Software Product Line Engineering. Addison-Wesley.

[17] Kyo C. Kang, Sajoong Kim, Jaejoon Lee1, Kijoo Kim, Gerard Jounghyun Kim, Euiseob Shin. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures.

[18] Falbo, R. A., Guizzardi, G., and Duarte, K. C. 2002. An ontological approach to domain engineering. In Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02). ACM. P. 351–358.

[19] Simos, M. and Anthony, J. 1998. Weaving the model web: A multi-modeling approach to concepts and features in domain engineering. In

Proceedings of the 5th International Conference on Software Reuse. IEEE Computer Society. P. 94–102.

[20] Терехов А.Н. и др. Real: методология и CASE-средство разработки информационных систем и программного обеспечения систем реального времени. Программирование. – 1999 – №5. – С. 44-51.

[21] Иванов А.Н. Технологическое решение REAL-IT: создание информационных систем на основе визуального моделирования // Сб. “Системное программирование”. Под ред. проф. А.Н. Терехова и Д.Ю.Булычева - Спб: Изд. СПбГУ, 2004. С. 89-100.

[22] Романовский К.Ю., Кузнецов С.В., Кознов Д.В. Объектно-ориентированных подход и диаграммы классов UML. Объектно-ориентированное визуальное моделирование. – Спб.: 1999. – С. 21-56.

[23] А. Иванов, С. Стригун. Технологическое решение REAL-IT: автоматизированная разработка пользовательского интерфейса. Сб. “Системное программирование”, под ред. проф. А.Н.Терехова и Д.Ю. Булычева. Спб: Изд. СПбГУ, 2004. С. 124-147.