Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Системное программирование

Храмышкина Юлия Сергеевна

Реализация системы проверки заданий по визуальному моделированию в QReal

Выпускная квалификационная работа

Научный руководитель:

к.т.н., доцент кафедры системного программирования Литвинов Ю.В.

Рецензент:

инженер-консультант Коновалов М.В

Санкт-Петербург

2017

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems

Software Engineering

Iuliia Khramyshkina

Implementation of visual modeling exercises checking system in QReal

Graduation Project

Scientific supervisor:
C.Sc., Docent Yurii Litvinov

Reviewer:
Consultant Engineer Mikhail Konovalov

Saint-Petersburg

2017

Введение

Часто при разработке программного обеспечения требуются значительные усилия на проектирование и анализ предлагаемых решений для создаваемого продукта. После этого выполняется реализация системы, и для этого существует множество различных языков программирования и иных вспомогательных инструментов. Для успешного выполнения работы на стадиях анализа и проектирования в большинстве случаев требуется умение выделять наиболее важную, существенную информацию и не акцентировать внимание на деталях. В связи с этим появляется понятие модели системы, которое означает как раз то, что в модели будет присутствовать самая важная информация о системе без лишних деталей.

Для человеческого восприятия наиболее удобными и наглядными являются визуальные модели, которые, как правило, представляются в виде диаграмм. Такое представление помогает разработчикам понимать и объяснять какие-либо идеи или решения быстрее, чем с использованием менее наглядных представлений. Однако для того, чтобы этот подход был эффективным, нужно, чтобы все разработчики понимали тот язык, с помощью которого изображаются диаграммы, и чтобы он был одинаковым для всех участников в команде. В противном случае, понимание наоборот снизится, и этот подход будет неэффективным и даже вредным.

Поэтому для того, чтобы таких проблем не возникало, был создан унифицированный язык моделирования (UML). На данный момент в нем существует 14 различных видов диаграмм, наиболее популярной из которых является диаграмма классов.

Однако часто овладение навыками визуального моделирования вызывает затруднения. Поэтому обучение визуальному моделированию вызывает интерес у исследователей. В частности, в одной из исследовательских работ в этой области был поставлен эксперимент, в

котором участвовало около 150 человек, часть из которых обучали визуальному моделированию. Этот эксперимент показал, что студенты, изучавшие и освоившие визуальное моделирование, справляются с поставленными заданиями успешнее, чем те, кто не изучал или изучал, но не освоил [1].

В СПбГУ на кафедре системного программирования существует научно-исследовательский проект QReal, представляющий собой систему, с помощью которой можно создавать предметно-ориентированные визуальные языки. В в одной из предыдущих работ автора данного текста [2] в системе QReal был поддержан редактор диаграммы классов UML для дальнейшего использования с целью обучения студентов визуальному моделированию.

1. Постановка задачи

Целью данной работы является разработка системы проверки учебных заданий по визуальному моделированию в среде QReal. Система предназначена для практических занятий и дистанционного обучения.

Пользователь сможет нарисовать решение своей задачи, а затем проверить его на правильность. Проверка будет производиться путем сопоставления реального решения с одним из заранее подготовленных идеальных, правильных решений данной задачи.

В рамках данной дипломной работы поставлены следующие задачи:

- разработать подход, в рамках которого будет осуществляться создание системы проверки заданий;
- разработать алгоритм сопоставления шаблонов на диаграммах;
- разработать архитектуру системы проверки заданий;
- реализовать систему проверки заданий по визуальному моделированию;
- провести апробацию.

2. Обзор

Для того, чтобы решить задачу поиска шаблонов на диаграммах в контексте данной работы, были проанализированы существующие решения в этой области.

2.1 Поиск шаблонов на диаграммах

2.1.1 Извлечение всех отношений графа

Одной из задач, в которых требуется осуществлять поиск шаблонов на диаграммах, является обнаружение паттернов проектирования.

Использование шаблонов проектирования приносит много полезного разработчикам, позволяя использовать опыт и знания своих коллег. Знание паттернов проектирования и умение их находить очень важно для понимания программы и её дальнейшего сопровождения. Следовательно, требуется найти способ, позволяющий эти паттерны находить.

В частности, в статье [3] приводится один из таких способов, а именно — используется представление диаграммы в виде обычного графа для установления изоморфизма между конкретным решением и тем шаблоном, с которым оно должно совпадать. Преимущество такого подхода заключается в том, что варианты каждого шаблона проектирования, а также любое его вхождение может быть обнаружено. Здесь берутся два графа, одним из которых является сама исследуемая система, а другим — соответствующий шаблон проектирования.

Перед тем, как проводить сопоставление, необходимо извлечь все отношения в графе. После извлечения всех отношений, каждое отношение графа представлено в группе с отношениями, имеющими ту же семантику.

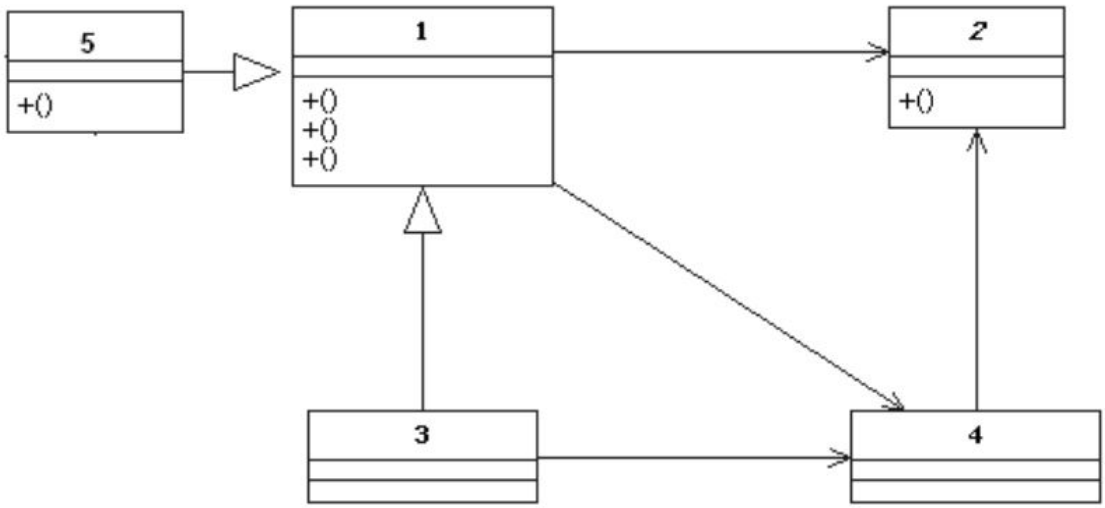

Рис 1. UML Diagram of System Design (рисунок взят из [3])

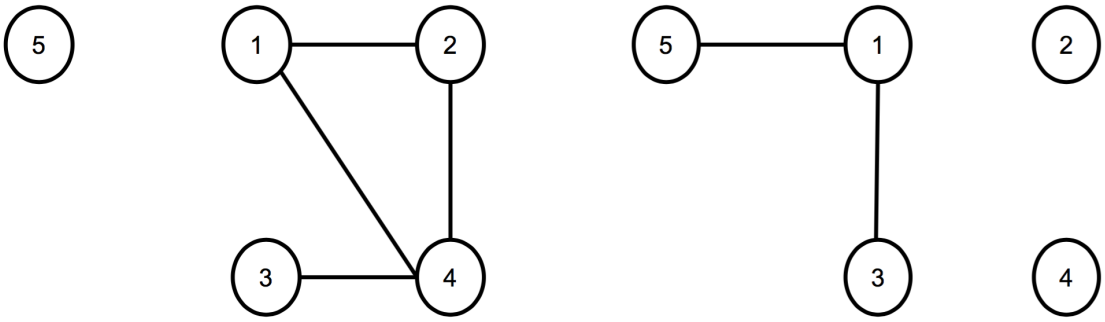

Рис. 2 Direct Association Relationship Graph и Generalization Relationship Graph

2.1.2 Подход на основе использования метрик

Авторы работы [4] рассуждают о том, что модели в разработке программного обеспечения позволяют инженерам уменьшить сложность программных систем, а также улучшить понимание требований и облегчить разработку, тестирование, поддержку и др. Часто при создании новой системы с нуля получается так, что многое из придуманного уже существует или когда-то существовало. Поэтому чтобы избежать дублирования и избыточных расходов времени и ресурсов, задача переиспользования существующих решений приобретает первостепенное значение.

Рассматривается два способа решения этой задачи, один из которых состоит в объединении обеих моделей (причем в качестве основы берутся их общие части), и использовании полученной модели как эталонной для выявления разницы исходных моделей, другой состоит в том, что нужно иметь эффективный репозиторий наряду с эффективным механизмом поиска информации. Так или иначе в каждом из случаев, проверка моделей на соответствие друг другу является фундаментальной операцией. Однако решения часто моделируются независимо и нередко существует некоторая несогласованность, различия в дизайне, конфликты между моделями, и это ожидаемо. Поэтому сходства и различия моделей следует измерять количественно для того, чтобы иметь точное соответствие.

Для того, чтобы выяснить, насколько похожи две модели, вводятся типы информации о подобии:

- лексическая информация (именование);
- внутренняя информация;
- информация окрестности.

Степень схожести между элементами сравниваемых моделей имеет значение от 0.0 до 1.0 и может быть представлена в двумерной матрице подобия. Лексическая информация используется для сравнения имен классов. Внутренняя информация используется для измерения подобия свойств, атрибутов и поведения сравниваемых элементов. А третий тип информации используется для измерения структурного подобия между соседями сравниваемых элементов.

Использование любого из способов измерения подобия по отдельности, как правило, не приводит к точному сопоставлению. Например, два класса могут иметь схожие названия, но совершенно различные свойства, и наоборот. Следовательно, опираясь только на сходство имен, нельзя сказать, похожи два класса или нет. Аналогичные ошибки могут произойти и с двумя другими способами.

Составные метрики, собранные из индивидуальных метрик с различными весами, позволяют рассматривать различные аспекты сходства одновременно, благодаря чему являются более эффективными.

Составные метрики подобия представляются в виде комбинации других индивидуальных показателей, каждому из которых назначен вес, что позволяет наиболее точно определить сходства/различия между сравниваемыми элементами. Веса, присваиваемые каждому показателю, имеют решающее значение для точности выбранной метрики. В связи с этим, были проведены эксперименты, выявляющие подходящие весовые коэффициенты для различных составляющих, с которыми подробнее можно ознакомиться в [4].

2.1.3 Метод сопоставления на основе правил

В работе [5] был предложен еще один подход, с помощью которого можно находить шаблоны проектирования на диаграммах. Авторы выделяют несколько видов, категорий поиска шаблонов, а именно:

- точное обнаружение шаблона;
- точное обнаружение антипаттернов;
- примерное обнаружение шаблона;
- подход с использованием формальных языков для описания шаблонов.

Впоследствии для решения задачи поиска шаблонов авторы предлагают общий язык, с помощью которого шаблоны могут быть выражены, и алгоритм сопоставления на основе заданных правил. Для элементов каждого типа: классы, методы, различные виды отношений (ассоциации, агрегации и др), созданы отдельные множества, отвечающие за каждый из видов. А диаграмма, в свою очередь, представляется как пара, состоящая из конечного набора классов и набора отношений. Далее это трансформируется в более формальный вид, на основе которого запускается алгоритм сопоставления.

Этот алгоритм является недетерминированным, и поэтому в случае, если для элемента, взятого в качестве исходного, алгоритм завершается успешно (то есть находится один из известных шаблонов), тогда та часть диаграммы, в которой шаблон был найден, удаляется. Это делается для того, чтобы уменьшить недетерминированность.

2.3 Обзор существующих решений, поддерживающих сопоставление шаблонов

На данный момент сопоставление шаблонов используется в системах для трансформации моделей их их верификации. Ниже рассмотрены примеры таких систем.

2.3.1 VIATRA

VIATRA (VIsual Automated model TRAnsformations) — платформа (фреймворк), позволяющая трансформировать и валидировать окружение систем, разработанных с использованием UML, посредством автоматической проверки требований согласованности, полноты, надежности [6].

VIATRA2, в первую очередь, направлена на разработку трансформации моделей для поддержки model-based систем. Основной концепцией в определении трансформаций модели в VIATRA2 является шаблон. Шаблон представляет собой набор элементов модели, расположенных в определенной структуре, удовлетворяющей некоторым дополнительным ограничениям. Шаблоны могут быть сопоставлены с определенными экземплярами модели, причем, так как система активно развивается и улучшается, проводятся эксперименты с алгоритмами их сопоставления [7].

В случае удачного сопоставления шаблонов, последующие манипуляции моделью могут быть выполнены посредством использования правил трансформации графа.

2.3.2 AToM3

В перечне платформ, позволяющих создавать предметно-ориентированные визуальные языки, существует кроссплатформенный проект AToM3 (A Tool for Multi-Formalism and Meta-Modelling, [8]).

Помимо обычных возможностей таких систем, в AToM3 есть поддержка других возможностей таких, как:

- трансляция моделей, причем они могут быть нарисованы и на одном визуальном языке, и на разных;
- оптимизация (замена одних конструкций на другие, более эффективные).

Трансляция моделей в AToM3 осуществляется с помощью преобразования графов. Так как в общем случае задача поиска подграфа в графе NP-полная, на больших графах возникают существенные проблемы с производительностью, однако при рассмотрении ее на малых графах и при наличии большого числа ограничений глубина поиска значительно уменьшается. В качестве преимуществ этого подхода отмечаются визуальность, высокоуровневость, серьезная теоретическая основа и формальность.

Также важным аспектом является то, что в AToM3 все правила преобразования графов задаются отдельно. Кроме того, в AToM3 можно одновременно вести работу сразу с несколькими языками сразу. Однако из-за этого при создании правил преобразования моделей из одного языка в другой требуется иметь обобщенные связи, способные соединять элементы различных языков.

2.3.3 GenGED

GenGED — проект, на базе которого реализуется анимированная симуляция диаграмм [9]. Главной его особенностью является то, что при интерпретации диаграммы пользователь видит не изменение диаграммы на визуальном языке, за которым, возможно, сложно уследить, а некоторую

анимацию. Это нужно потому, что иногда создание диаграммы на визуальном языке, её интерпретация и, в целом, представление предметной области задачи, бывает неочевидным для пользователей. А за счет перехода к анимации от обычных диаграмм непонимание уменьшается.

GenGED предназначается для некоммерческого и свободного использования, однако имеет существенный недостаток — отсутствие кроссплатформенности (доступны только Linux и Solaris).

Для перехода от диаграмм к анимации используется преобразование графов. Как и везде, где используется преобразование графов, из-за того, что задача поиска подграфа в графе NP-полная, возможно существенное снижение производительности на больших графах. Также стоит отметить, что в этом решении нет возможности задавать дополнительные ограничения на правила преобразования и их применение, поэтому оно является в определенной степени упрощением решения, используемого в AToM3. Подробнее описание данного подхода описано в [10, 11], где рассмотрены набор ресторанов, обслуживающих покупателей на машинах [10], и анимация работы лифта с помощью сетей Петри [11].

2.3.4 GROOVE

GROOVE — открытое программное средство, которое служит для преобразования графов [12]. Также в нем есть возможность автоматической верификации полученных после преобразования моделей. GROOVE является свободным проектом, реализованным на Java.

Создатели GROOVE считают, что графы являются хорошей основой для разработки программных систем. Их главные достоинства в том, что возможно визуальное представление графов, хотя это является практичным только для мелкомасштабных примеров, и что самое главное — богатая формальная основа. Кроме того, графы достаточно гибкие, чтобы взаимодействовать со всеми видами моделей, заранее не ограничивая их. Также в теории преобразования графов создатели находят математический

инструмент для формализации преобразований моделей, таких как рефакторинг, верификация, эволюция моделей и др. [13].

Они видят применение своих технологий в контексте работы с UML-диаграммами, XML-моделями, а также в моделировании состояний систем во время выполнения программ.

2.3.5 MetaLanguage

В MetaLanguage существует возможность трансформации моделей посредством применения правил преобразования графов. Это происходит согласно алгоритму, описанному в статье [14], который состоит из предположения, что в рассматриваемых моделях ребер меньше, чем вершин, благодаря чему каждое ребро можно сопоставить с определенной вершиной.

Алгоритм начинает свою работу с поиска всех вхождений произвольной связи из шаблона в исходной модели. Далее для каждой найденной связи происходит рекурсивный поиск остальных связей из шаблона. В конце вершины, инцидентные связям, проверяются и добавляются в соответствующий шаблону граф.

2.3.6 Eclipse Modeling Framework

Eclipse Modeling Framework (EMF) содержит инструментарий для разработки метамоделей и работы с моделями, созданными на их основе [15]. Также существует компонент, предоставляющий возможность рефакторинга моделей — EMF Refactor. Он использует модуль, отвечающий за преобразование моделей.

EMF-преобразование модели основывается на преобразовании исходной модели в целевую, на основе определенных правил, представимых в виде визуального языка [16]. Все правила состоят из двух основных частей — левой и правой части правила. Левая часть описывает предусловие преобразования, а правая — постусловие. Элементы, присутствующие в левой части и отображающиеся в правой, должны присутствовать в исходной

модели и не изменяться во время преобразования. Элементы, присутствующие в левой части, но отсутствующие в правой, удаляются в процессе преобразования. А те элементы, которые есть в правой части, но которых нет в левой, наоборот, создаются в целевой модели. Также применимость правил может ограничиваться дополнительными условиями, если это требуется. Например, NAC (Negative application condition) — условие, при котором правило рефакторинга не может быть применено.

В EMF Refactor доступно графическое представление диаграммы трансформации модели, а также для вновь созданных рефакторингов доступны предварительный просмотр целевой модели и отмена внесенных изменений.

2.3.7 Generic Modeling Environment

Generic Modeling Environment — это среда метамоделирования, базирующаяся на языке UML. В ней есть возможность создания предметно-ориентированных языков, а также есть возможность преобразования моделей с помощью плагина C-SAW (The Constraint-Specification Aspect Weaver) [17].

В C-SAW исходная модель переводится в целевую с помощью спецификаций преобразования. В них определяется, что происходит с каждым конкретным элементом в данном преобразовании. Каждая спецификация состоит из аспектов и стратегий. Аспекты требуются для того, чтобы начать преобразование, а само преобразование определяется стратегией. Они описываются с помощью Embedded Constraint Language (ECL) — встроенного языка ограничений. Этот механизм применяется также и при рефакторинге моделей. Для этого существует еще один плагин (Model refactoring browser), встроенный в GME, который вместе с C-SAW дает возможность рефакторинга моделей. В отличие от EMF, механизмы рефакторинга в C-SAW описываются на текстовом языке, а не на визуальном.

3. Подход к решению

3.1 Особенности задач

Система проверки учебных заданий, разрабатываемая в контексте данной работы нацелена, в первую очередь, на закрепление основных навыков и умений у студентов, полученных на лекциях по визуальному моделированию.

Задачи, которые предоставляются студентам для решений, должны иметь лаконичные решения и не являться громоздкими. Однако допускается существование задач, в которых имеется несколько вариантов решений.

3.2 Общая концепция

В конечном итоге в реализуемой системе всё будет выглядеть следующим образом.

Студент получает задание, результатом выполнения которого должна быть диаграмма классов. Он создает диаграмму классов с помощью редактора UML в среде QReal, после чего нажимает кнопку “проверить решение”, и в ответ получает сообщение о том, справился он с задачей или нет.

Учитель для каждого конкретного задания создает базу решений, которые считаются корректными в контексте данной задачи. Так как решений у одной и той же задачи может быть много, а рисовать их все может быть слишком трудоемко, предлагается следующее решение.

Учитель рисует какую-либо диаграмму, удовлетворяющую данной задаче, которая впоследствии становится одним из идеальных решений. Затем, если диаграмму можно разбить на логические блоки, учитель выделяет их.

Для каждого логического блока из списка существующих шаблонов выбираются те, которым этот логический блок может соответствовать.

Подходящие варианты шаблонов выбираются в порядке популярности их использования для решения данной задачи (с точки зрения учителя).

В случае, если для данного логического блока не существует всех необходимых шаблонов, тогда недостающие создаются и добавляются к уже существующим.

3.3 База мини-шаблонов

Для задания различных вариантов корректных решений и впоследствии для сопоставления обычного решения с идеальным будет существовать база мини-шаблонов, которую можно будет пополнять в случае, если требуемого в ней еще нет. Для этого будет нужно нарисовать небольшую диаграмму и сохранить её как шаблон. Гибкость решений будет достигаться за счет наличия базы мини-шаблонов и разбиения диаграммы на логические блоки, соответствующие этим шаблонам.

3.4 Представление диаграмм

После того, как были созданы идеальные решения или получено решение, требующее проверки, возникает вопрос о том, как именно представить диаграмму, чтобы затем эти сопоставления проводить.

Предлагается следующее: в случае, когда рассматривается идеальное решение, для каждого логического блока диаграммы хранить набор узлов, соответствующий классам и интерфейсам, а также хранить наборы отношений и соответствующих узлов для каждого типа отношений (ассоциация, агрегация, наследование). В случае, когда рассматривается решение, которое должно пройти проверку, предлагается хранить его точно таким же образом, но без первоначального деления на логические блоки.

4. Алгоритм сопоставления

В результате анализа существующих решений поиска шаблонов на диаграмме для реализации поставленной задачи предлагается следующий алгоритм их сопоставления.

После того, как был создан набор идеальных решений, а также ученик создал решение, которое должно пройти проверку, начинается сопоставление.

Сначала из набора идеальных решений берется логический блок и один из его вариантов и происходит его сопоставление с решением ученика, причем на данном этапе рассматриваются только сущности (классы и интерфейсы), входящие в каждое из решений. Это происходит до тех пор, пока все логические блоки из идеального решения не будут сопоставлены с решением, предложенным учеником.

Далее, происходит сопоставление связей в сравниваемых решениях. В решении, которое предлагается учеником, те связи, которые находятся внутри логических блоков, помечаются как принадлежащие данному логическому блоку. Все оставшиеся отношения связывают различные логические блоки. В случае, если внешние отношения в решении студента и в решении учителя связывают одни и те же логические блоки, то решение считается верным.

Если на одном из этапов сопоставления реальное решение и идеальное не совпали, то в случае, если алгоритм только начал свою работу, решение считается неправильным. В противном случае, алгоритм останавливается и вновь начинает свою работу, рассматривая очередную из еще не рассмотренных перестановок логических блоков.

5. Архитектура системы

Ранее в системе QReal был поддержан редактор диаграммы классов UML, а также в QReal есть возможность поддержки плагинов-инструментов, которые могут обладать какой-либо логикой. Поэтому система проверки заданий в итоге была реализована в QReal качестве плагина-инструмента.

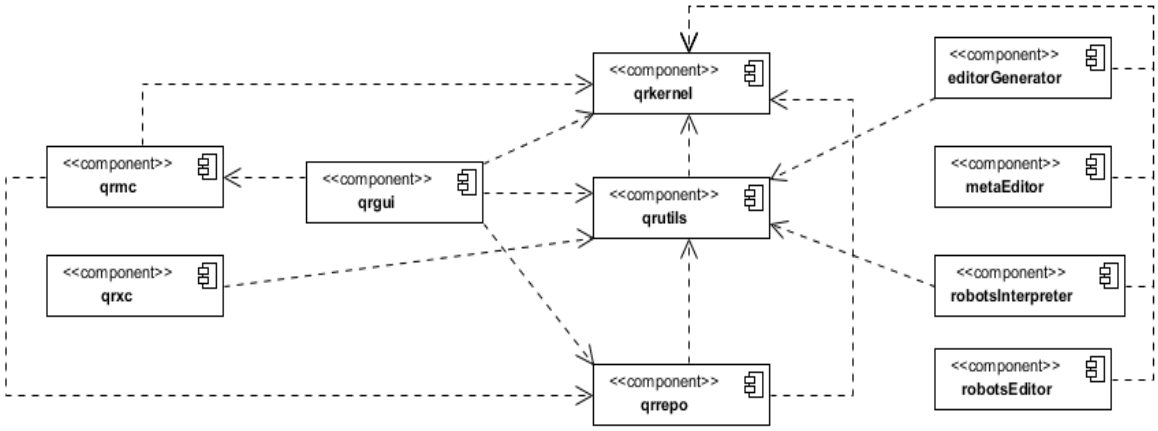

Рис. 3. Диаграмма компонентов QReal рисунок взят из [18])

На рис. 3 представлена диаграмма компонентов среды QReal. Стоит отметить, что архитектура системы QReal имеет довольно сложное устройство [19]. Вследствие этого плагины и плагины-инструменты вынуждены использовать код из общей части QReal. Например, классы, такие как Id и Exception, содержащиеся в компоненте qrkernel, используются как в плагинах, так и практически во всех частях системы. Также часто плагинам требуется доступ к репозиторию с моделями, который находится в компоненте qrrepo. Помимо этого, главной программой для взаимодействия с пользовательским интерфейсом и ответственной за управление плагинами является qrgui. В частности, предоставляемая плагинами-инструментами функциональность может быть зарегистрирована в qrgui, а затем вызвана пользователем.

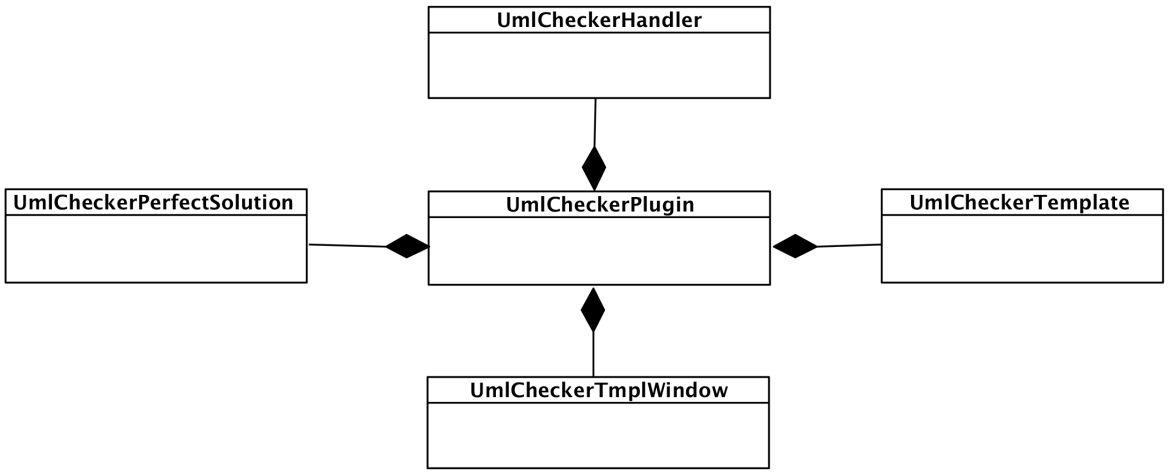

Рис 4. Диаграмма классов системы проверки заданий

В свою очередь, в реализуемой системе проверки заданий, устройство которой изображено на рис 4, существует несколько основных компонентов.

UmlCheckerPlugin отвечает за управление всеми компонентами, относящимися к системе проверки заданий, и за создание системы проверки заданий в QReal.

UmlCheckerTemplate отвечает за создание и сохранение шаблонов, которые впоследствии могут быть использованы в качестве возможных вариантов для логических блоков. Как правило, все шаблоны являются небольшими, состоящими из 3-7 элементов.

UmlCheckerPerfectSolution отвечает за создание и сохранение правильных решений, предлагаемых учителем. Учитель предлагает правильное решение, выделяет все логические блоки, содержащиеся в нем, и выбирает для них всевозможные альтернативы. Также он выбирает все возможные внешние отношения, которые связывают логические блоки.

UmlCheckerTmplWindow отвечает за отображение и выбор шаблонов для логического блока или связи.

UmlCheckerHandler отвечает за сопоставление реального решения с идеальным.

6. Особенности реализации

В результате анализа существующих решений, изучения архитектуры QReal, разработки алгоритма сопоставления шаблонов была реализована система проверки заданий по визуальном моделированию в среде QReal.

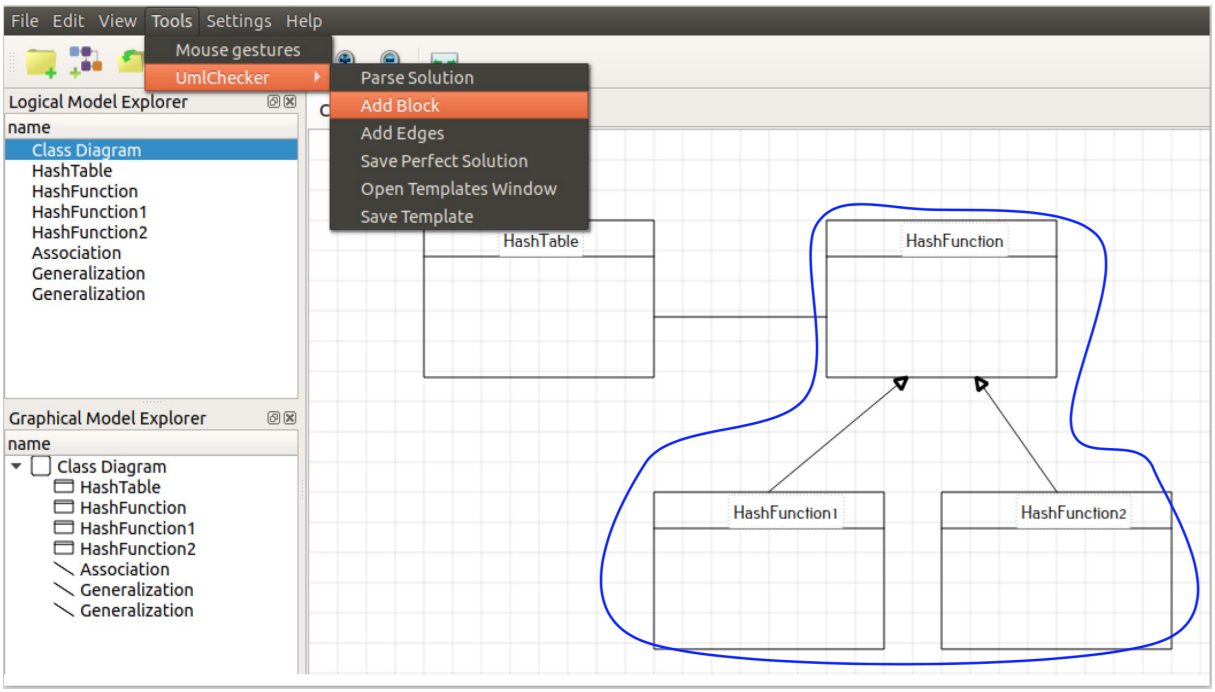

Рис 5. Система проверки заданий

На рис 5. изображено то, как выглядит доступное пользователю меню. На диаграмме изображено решение, создаваемое учителем для одной из задач. Цветом на рисунке выделен логический блок элементов, определенный учителем, для которого можно будет выбрать все варианты, которые с точки зрения учителя будут иметь такую же или похожую семантику. После нажатия на кнопку “Add Block” появляется окно с выбором подходящих для логического блока шаблонов рис 6.

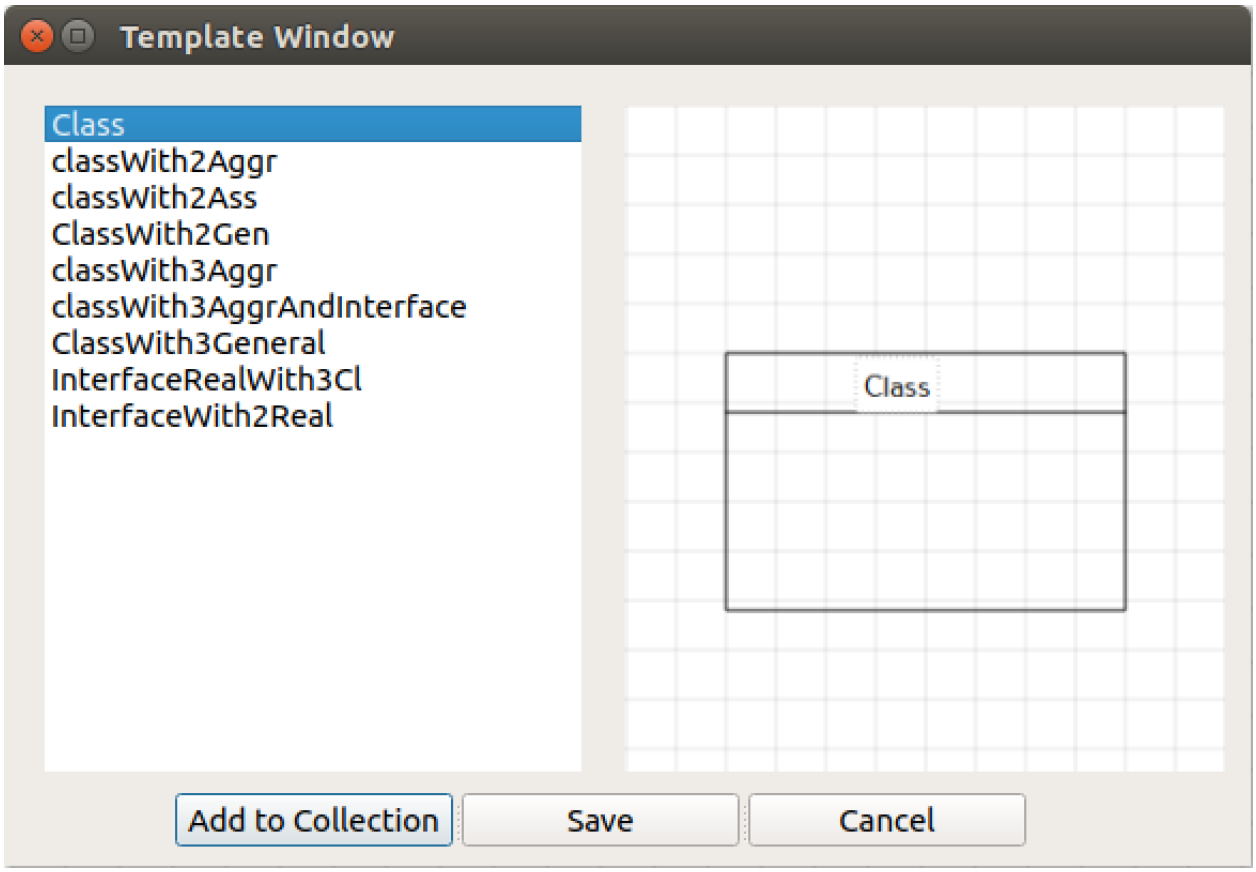

Рис 6. Окно выбора шаблонов для логического блока

Для того, чтобы для рассматриваемого логического блока добавить все необходимые варианты, нужно выбрать подходящие и нажать на кнопку “Add to Collection”, после этого следует сохранить свой выбор. Также есть возможность выбрать альтернативы для связей между блоками. Для этого нужно выделить связь, нажать на кнопку “Add Edge” и так же выбрать все подходящие варианты и сохранить их.

После того, как учитель создал набор правильных решений, ученик может нарисовать решение задачи и проверить его на правильность. Он должен сохранить решение, затем нажать на кнопку “Parse Solution” и выбрать ту задачу, решение к которой он нарисовал. На рис 7. представлен пример работы программы в случае, когда ученик справился с поставленной задачей.

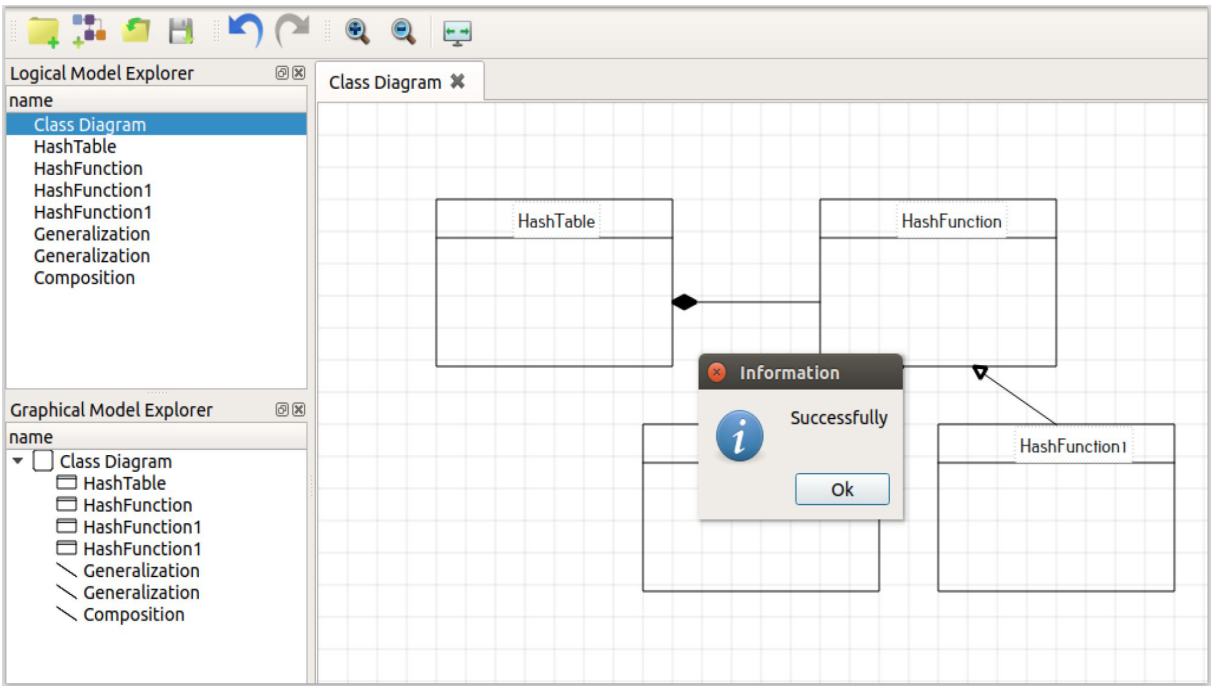

Рис 7. Пример работы системы проверки заданий

7. Апробация

После окончания разработки системы была проведена ее апробация на группе из 7 человек. Апробация проводилась на двух задачах, одна из которых являлась продолжением и усложнением другой. Всего было прислано около 20 вариантов решений задач, среди которых были как правильные, так и неправильные. Как правило, система выполняла классификацию точно, то есть те решения, которые соответствовали замыслу учителя, считались правильными, а остальные считались неправильными.

Также был проведен эксперимент на диаграмме с большим количеством сущностей для того, чтобы узнать, при каком количестве сущностей время проверки станет слишком большим. Эксперимент показал, что время работы алгоритма, который сопоставляет реальное решение с идеальным, становится неприемлемым, когда на диаграмме становится больше, чем 25-30 сущностей (то есть около 40-50 элементов, включая связи). Это обусловлено тем, что задача поиска подграфа в графе является NP-полной, и в связи с этим, в общем случае получается экспоненциальный рост времени выполнения.

На рисунках 8-11 представлены примеры решений, полученные во время апробации.

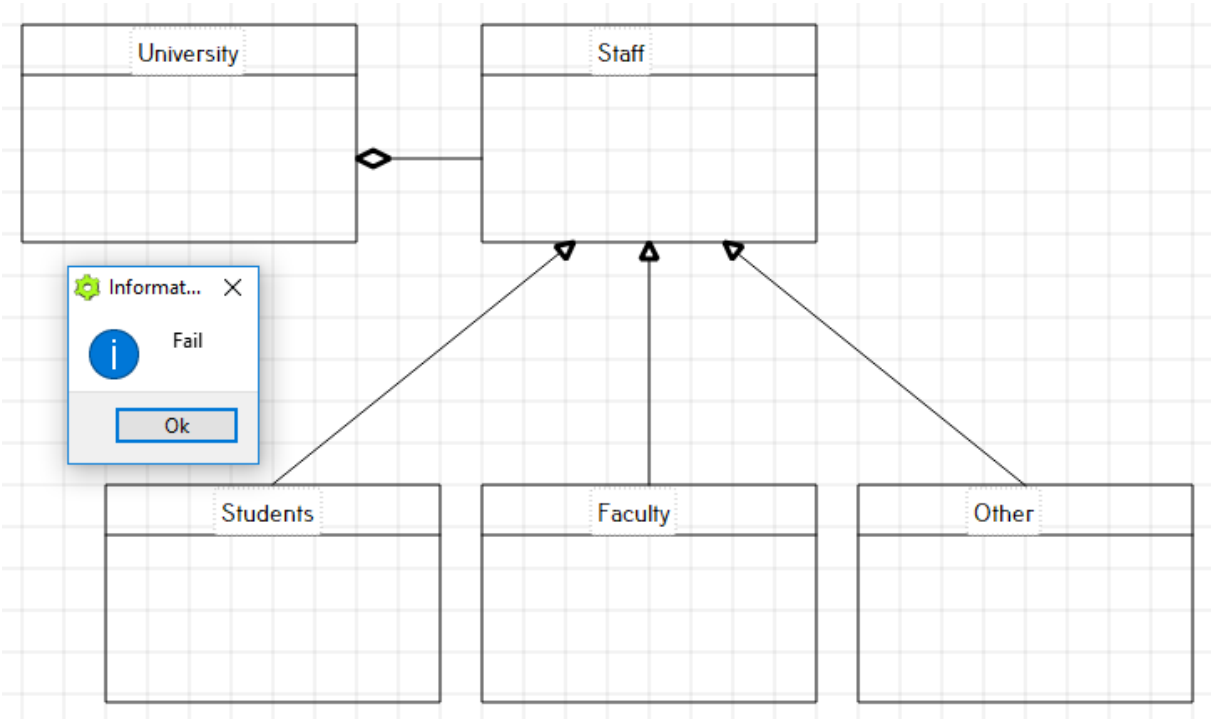

Рис 8. Задача 1. Неправильное решение

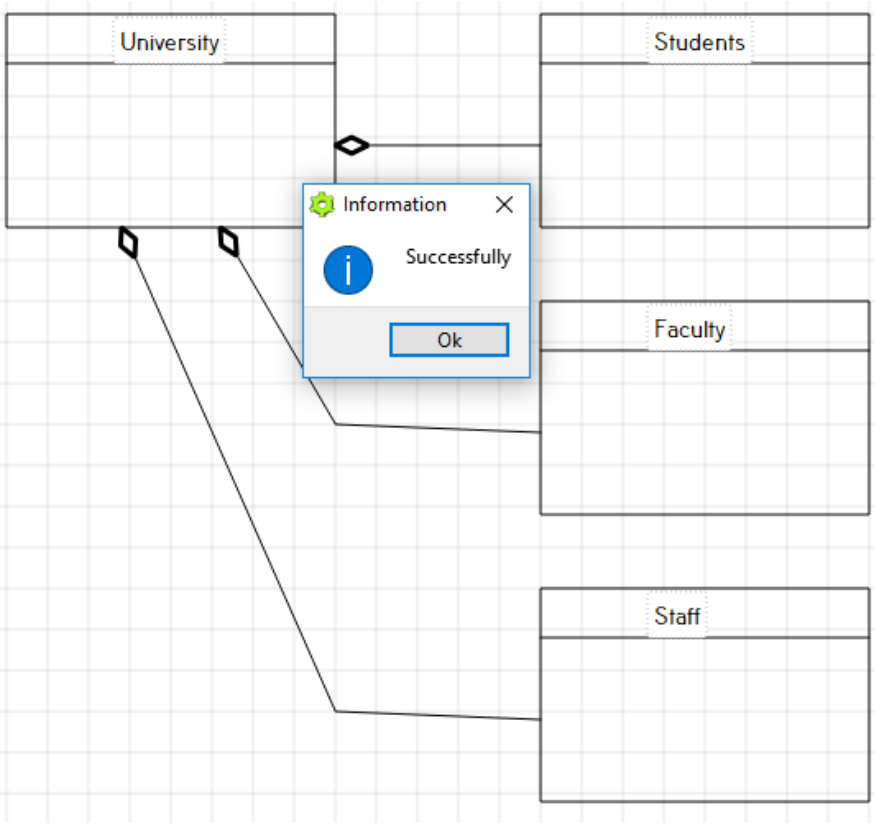

Рис 9. Задача 1. Правильное решение

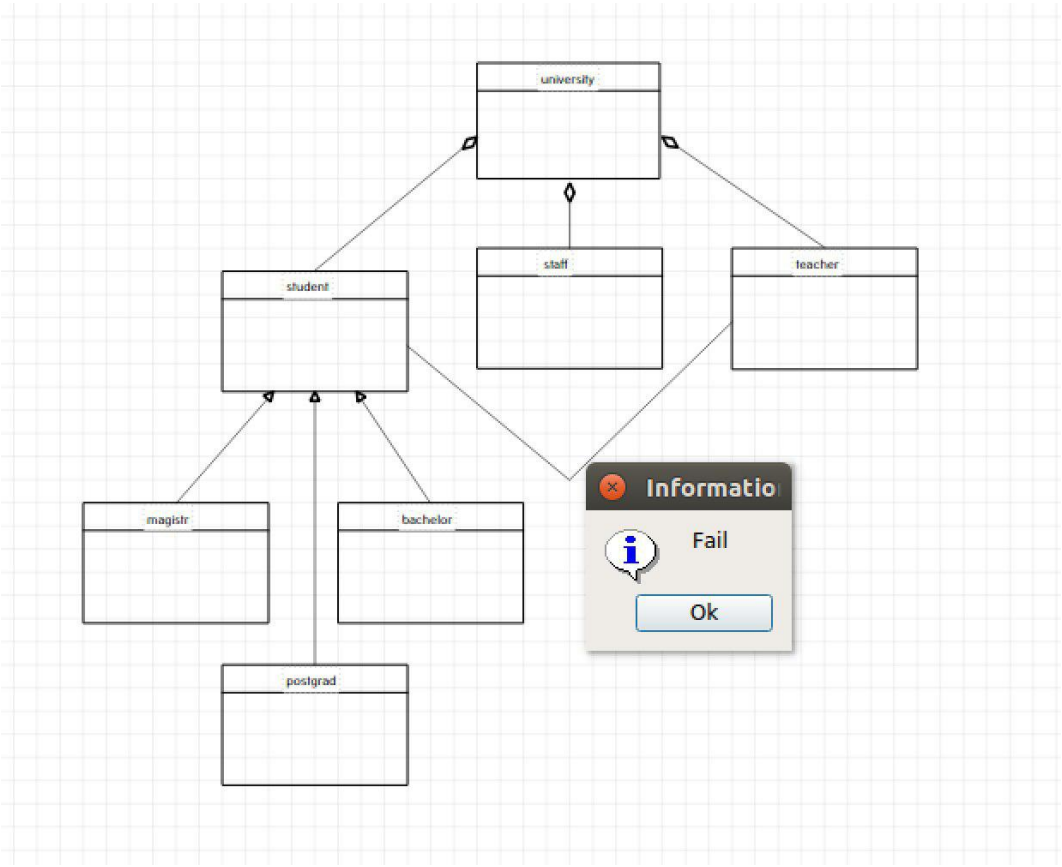

Рис 10. Задача 2. Неправильное решение

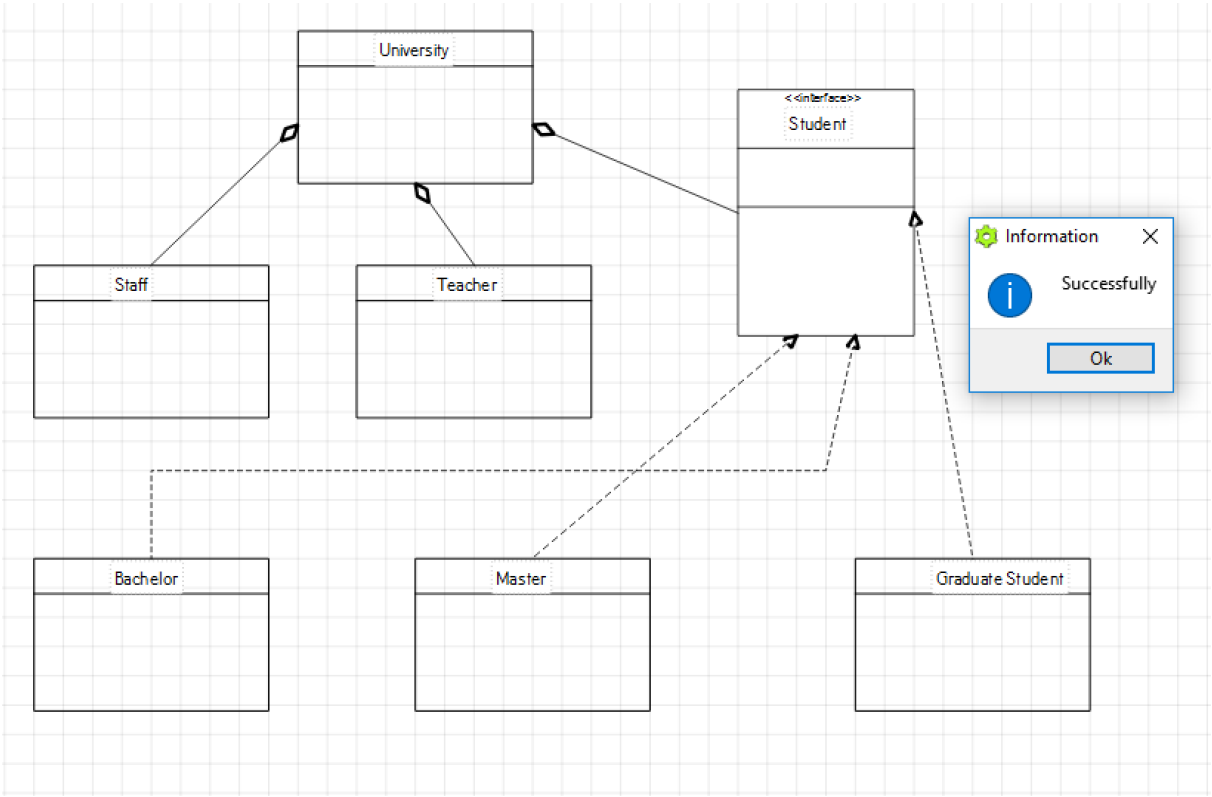

Рис 11. Задача 2. Правильное решение

Заключение

В рамках данной выпускной квалификационной работы были получены следующие результаты:

- разработан подход, отражающий ключевые особенности предлагаемых для решения задач и основные элементы, на которых строится алгоритм;
- разработана архитектура системы проверки заданий по визуальному моделированию;
- разработан алгоритм сопоставления шаблонов на диаграммах, проверяющий схожесть реального решения с одним из ожидаемых идеальных;
- реализована система проверки заданий в среде QReal (C++/Qt) [20];
- проведена апробация системы проверки заданий на группе студентов.

Список литературы

[1] Lano K., Yassipour-Tehrani S., Alfraihi H. Experiences of Teaching Model-based Development, 2015.

[2] Храмышкина Ю.С., Литвинов Ю.В. Поддержка диаграммы классов языка UML 2.5 в среде QReal // Материалы научно-практической конференции студентов, аспирантов и молодых учёных "Современные технологии в теории и практике программирования". СПб.: Изд-во Политехн. ун-та, 2016. С. 86-87.

[3] Gupta M., Pande A. Design patterns mining using subgraph isomorphism: Relational view //International Journal of Software Engineering and Its Applications (IJSEIA). – 2011. – Т. 270.

[4] Al-Khiaty M. A. R., Ahmed M. UML Class Diagrams: Similarity Aspects and Matching //Lecture Notes on Software Engineering. – 2016. – Т. 4. – №. 1. – С. 41.

[5] Ballis D., Baruzzo A., Comini M. A rule-based method to match Software Patterns against UML Models //Electronic Notes in Theoretical Computer Science. – 2008. – Т. 219. – С. 51-66.

[6] VIATRA. URL: http://www.eclipse.org/viatra/ (дата обращения: 18.05.2017)

[7] Bergmann G. et al. Incremental pattern matching in the viatra model transformation system //Proceedings of the third international workshop on Graph and model transformations. – ACM, 2008. – С. 25-32.

[8] De Lara J., Vangheluwe H. AToM3: A Tool for Multi-formalism and Meta-modelling //International Conference on Fundamental Approaches to Software Engineering. – Springer Berlin Heidelberg, 2002. – С. 174-188.

[9] Bardohl R., Ermel C., Weinhold I. GenGED–a visual definition tool for visual modeling environments //International Workshop on Applications of Graph

Transformations with Industrial Relevance. – Springer Berlin Heidelberg, 2003. – С. 413-419.

[10] Ermel C. et al. Animated simulation of integrated UML behavioral models based on graph transformation //Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on. – IEEE, 2005. – С. 125-133.

[11] Ehrig P. N. H., Ermel C., Taentzer G. Simulation and animation of visual models of embedded systems //Embedded Systems–Modeling, Technology, and Applications. – Springer Netherlands, 2006. – С. 11-20.

[12] Rensink A. The GROOVE Simulator: A Tool for State Space Generation // AGTIVE 2003 – Revised Selected and Invited Papers, volume 3062 of Lecture Notes in Computer Science, Berlin, Springer Verlag, 2004, P. 479–485.

[13] GRaphs for Object-Oriented VErification (GROOVE). URL: http://groove.sourceforge.net/groove-index.html (дата обращения: 18.05.2017)

[14] Sukhov A.O., Lyadova L.N. Horizontal Transformations of Visual Models in MetaLanguage System // Proceedings of the 7th Spring/Summer Young Researchers' Colloquium on Software Engineering. SYRCoSE, 2013. 9 p.

[15] Сорокин А.В., Кознов Д.В., Обзор Eclipse Modeling Project. // Системное программирование. Вып. 5. СПб.: Изд-во СПбГУ. 2010, с. 6-31

[16] Enrico Biermann, Karsten Ehrig, etc., EMF Model Refactoring based on Graph Transformation Concepts. // Electronic Communications of the EASST. Volume 3. 2006, pp. 3-19

[17] Jing Zhang , Yuehua Lin , Jeff Gray, Generic and Domain- Specific Model Refactoring using a Model Transformation Engine. // Volume II of Research and Practice in Software Engineering. 2005, pp. 199-218

[18] QReal Wiki. URL: https://github.com/qreal/qreal/wiki (дата обращения: 18.05.2017)

[19] 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.

[20] QReal. URL: https://github.com/qreal/qreal/ (дата обращения: 18.05.2017)