Санкт-Петербургский государственный университет

КИДЯНКИН Михаил Владимирович

Выпускная квалификационная работа

Технология создания визуальных языков на базе глубокого метамоделирования в REAL.NET

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5006.2017 «Математическое обеспечение и администрирование информационных систем»*

Профиль *Системное программирование*

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

Рецензент:
инженер-разработчик ООО «Крафтваерк»
Е. В. Кузьмина

Санкт-Петербург
2021

Saint Petersburg State University

Mikhail Kidiankin

Bachelor's Thesis

Visual languages development technology based on deep metamodeling in REAL.NET

Education level: bachelor

Speciality *02.03.03 «Software and Administration of Information Systems»*

Programme *CB.5006.2017 «Software and Administration of Information Systems»*

Profile: *System Programming*

Scientific supervisor:
C.Sc., System Programming chair docent
Y.V. Litvinov

Reviewer:
developer at LLC "Kraftvaerk"
E.V. Kuzmina

Saint Petersburg
2021

Оглавление

Введение

В современном мире существуют несколько классов задач, для которых традиционно применяются технологии визуального моделирования. Например, визуальные языки программирования могут применяться как аналог текстовых предметно-ориентированных языков, особенно если конечные пользователи не знакомы с программированием. В качестве примеров таких визуальных языков можно привести языки, применяемые в образовательных целях, например, языки программирования роботов сред TRIK Studio [9] и Lego Robolab [6]. Кроме этого, визуальные языки могут применяться и в сферах профессиональной разработки. Например, языки UML [4] для описания диаграмм достаточно часто [5] используются при разработке архитектуры программных систем.

Разработка среды визуального моделирования представляет собой трудоемкий процесс. Помимо непосредственно языка, среда обычно включает в себя такие компоненты, как редактор, генератор из визуальной модели в другой язык, а также может иметь дополнительные компоненты, специфические для предметной области применения. Для упрощения разработки такого рода систем для разных предметных областей могут применяться платформы предметно-ориентированного моделирования (DSM, domain-specific modeling), которые могут иметь универсальный редактор, генераторы, а также инструменты для описания визуальных языков, благодаря которым компоненты одной и той же DSM-платформы могут быть использованы в разных предметных областях.

Визуальные языки часто описывают с помощью других визуальных языков. Модель описания визуального языка называют *метамоделью*, а набор методов такого описания называют *метамоделированием*. Существуют несколько способов метамоделирования, например, классическое двухуровневое метамоделирование, применяемое в UML. При этом существуют исследования, указывающие на недостатки этого подхода [3]. Другая разновидность — глубокое метамоделирование — было

создано как вариант решения этих недостатков. Однако, в силу слабой распространенности глубокого метамоделирования остается открытым вопрос о применимости этого подхода к реальный проектам.

На кафедре системного программирования группой преподавателей и студентов разрабатывается REAL.NET [17] — платформа предметно-ориентированного моделирования, имеющая своей целью, помимо прочего, реализовать и исследовать различные подходы к метамоделированию, в том числе и к глубокому метамоделированию. На данный момент платформа имеет микросервисную архитектуру, несколько различных редакторов: графовый веб-редактор, редактор для управления умным домом, два редактора в виде настольных приложений — простой кроссплатформенный и более сложный для ОС Windows.

Система описания визуальных языков в REAL.NET выстроена таким образом, чтобы обеспечить возможность использования и тестирования разных подходов к метамоделированию. Именно поэтому REAL.NET может быть использован для исследования глубокого метамоделирования и его сравнения с другими подходами.

1. Постановка задачи

Целью данной работы является реализация в REAL.NET технологии глубокого метамоделирования. Для достижения данной цели были поставлены следующие задачи:

1. Провести исследование и реализовать поддержку логики глубокого метамоделирования с использованием имеющихся технологий метамоделирования в REAL.NET.

2. Разработать графический интерфейс веб-редактора визуального моделирования, обеспечивающего поддержку специфических требований работы с глубоким метамоделированием.

3. Провести апробацию поддержки глубокого метамоделирования с использованием предметно-ориентированных языков визуального моделирования.

2. Обзор

2.1. Метамоделирование

2.1.1. Двухуровневое метамоделирование

Для описания модели на визуальном языке обычно используется модель на другом визуальном языке, которая называется *метамоделью*. Данных подход можно продемострировать на примере диаграммы классов UML. На диаграмме классов языка UML отображаются классы, объекты, связи и другие элементы, специфические для данной предметной области и решаемой задачи. В то же время сами понятия языка UML — «класс», «связь» и другие — определены с помощью другого языка — MOF [11].

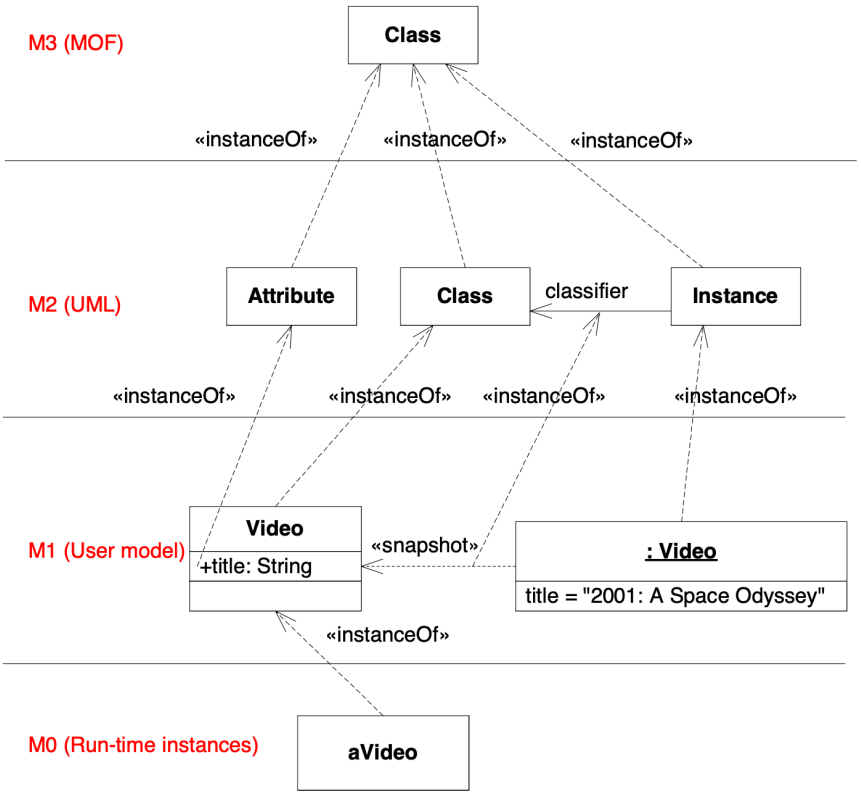

Рис. 1: Иерархия метамоделей из [10]

Таким образом, модель на языке UML можно разделить на четыре уровня (рисунок 1 из [10]). На уровне M0 представлены объекты предметной области, на M1 — модель этих объектов, на M2 — элементы

языка UML, а на M3 — элементы языка MOF.

На рисунке 1 между различными уровнями мы можем увидеть связь *инстанцирования* («instanceOf»). Она обозначает, что некоторый объект нижней модели является экземпляром некоторого объекта верхней модели.

В UML отношение инстанцирования возможно только между двумя элементами соседних уровней-моделей. Такой подход будем называть *двухуровневым метамоделированием*, поскольку инстацирование возможно только двумя уровнями, и каждая модель полностью может быть описана своей метамоделью.

2.1.2. Проблемы двухуровнего метамоделирования

Описанный выше подход представляет собой простую схему организации метамоделей, однако, существуют несколько проблем, которые возникают при данном подходе [3].

Неоднозначность классификации

Основная проблема неоднозначности классификации заключается в том, что инстанцирование можно понимать в двух значениях — как инстанцирование элемента модели от класса метамодели, либо как инстанцирование элемента-экземпляра от класса той же модели (например, в UML на диаграмме классов мы не имеем языкового инструмента, позволяющего контролировать, что некий объект нашей модели действительно может являться экземпляром некоторого класса-элемента этой же модели).

Умножение сущностей

Поскольку классических двухуровневый подход не предлагает возможностей инстанцирования через несколько уровней, то на каждом следующем уровне, если мы хотим в нашем языке поддерживать строгую типизацию, требуется, чтобы на уровне метамодели метамодели (метаметамодели) были определены сущности класса и экземпляра (рисунок 2 из [3]).

При этом, если уровней в модели много, то их определение должно

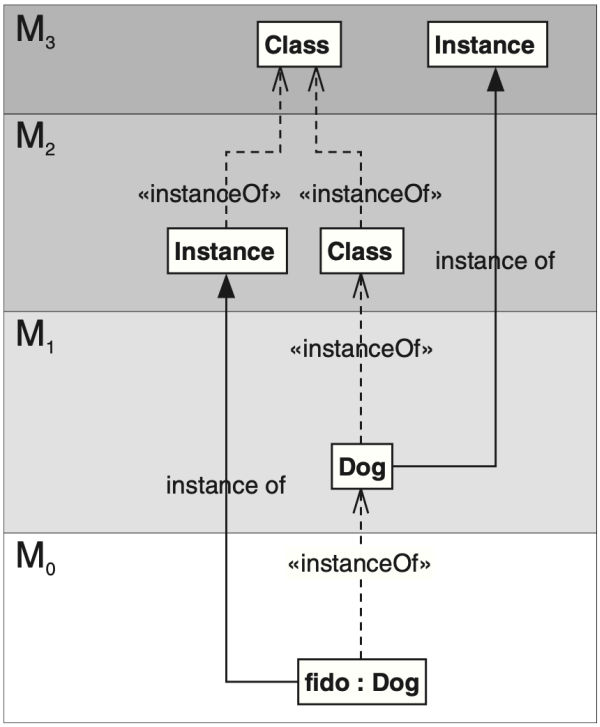

Рис. 2: Умножение сущностей из [3]

присутствовать на большом количестве уровней, повторяясь много раз. Данная проблема усложняет описание метамоделей, добавляет большое пространство для ошибок, а также ведет к лишнему увеличению числа элементов модели.

2.1.3. Глубокое метамоделирование

Понятие принципа глубокого метамоделирования можно сформулировать как набор практик, направленных на решение возникших в двухуровневом подходе проблем. Основной идеей подхода является введение понятия «класса-объекта» («clabject») [1], сущности, которая совмещает понятия класса и объекта. Другими словами, это объект, который может быть инстанцирован (как объект) и быть инстанцируемым (как класс) одновременно. Вводятся инструменты, позволяющие, в отличие от двухуровнего метамоделирования, производить инстанцирование на несколько уровней в глубину, а не только на один следующий уровень. Для формализации данного подхода вводятся следующие понятия:

Мощность и уровень элемента

Каждому «классу-объекту» сопоставляются два числа — *мощность* (potency) и *уровень* (level). Мощность элемента показывает, сколько еще раз данный элемент можно инстанцировать. Например, при инстанцировании элемента с мощностью 3 мы получаем элемент мощности 2. Элемент с мощностью 0 инстанцировать нельзя. Уровень элемента, с свою очередь показывает, на каком уровне модели данный элемент находится. Понятия мощности и уровня можно связать следующим образом: для инстанцирования элемента его мощность должна быть не меньше уровня. Такая система позволяет добавлять более гибкие ограничения при инстанцировании.

Одиночные и двойственные поля

Для более гибкой реализации инстанцирования в глубоком метамоделировании, понятия мощности и уровня можно обобщить и на атрибуты. Атрибут элемента — некоторое его свойство, обычно состоит из имени и значения, понятие заимствовано из UML [4]. При работе с атрибутами можно выделить два возможных сценария поведения. Первый — одиночные поля — могут иметь значения только если их мощность равна нулю, двойственные могут иметь значения при любом значении мощности, и даже могут переопределять значения при инстанцировании.

2.2. REAL.NET

REAL.NET [17] — универсальная среда для разработки визуальных языков, разрабатываемая на кафедре системного программирования.

2.2.1. Микросервисная архитектура

На данный момент REAL.NET реализует микросервисную архитектуру и состоит из следующих компонентов:

1. Репозиторий — основной сервис, в котором реализовано управление моделями: их создание и удаление, добавление и управление

элементами модели: инстанцирование элементов, работа с атрибутами. Именно в этом компоненте реализуются различные подходы к метамоделированию.

2. Дополнительные компоненты серверной части: сервис авторизации, сервис хранения сериализованных моделей, шаблон сервиса генерации, шлюз.

3. Редакторы: универсальный редактор графовых моделей (рисунок 3), в котором модель представляется в виде узлов и связей между ними, и табличный редактор правил системы «Умный дом».

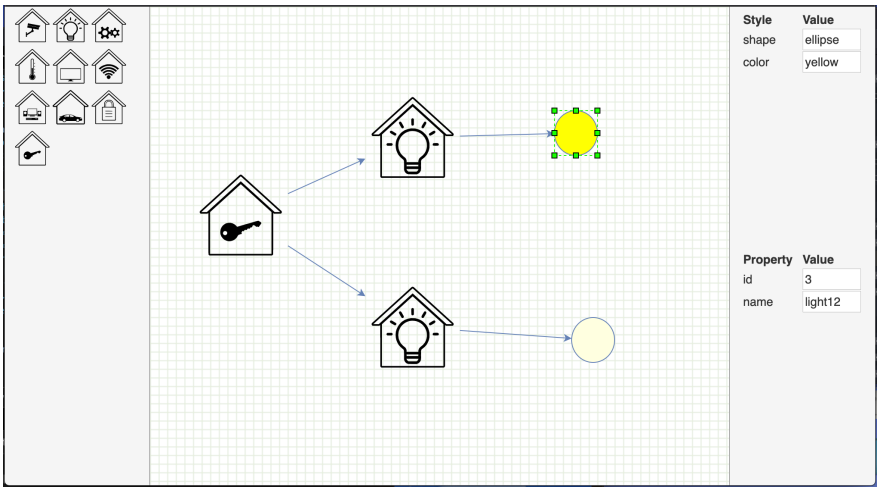

Рис. 3: Универсальный редактор

2.2.2. Слоистая архитектура репозитория

Репозиторий REAL.NET реализует слоистую архитектуру. Каждый слой представляет собой некий языковой уровень, реализованный на метамодели нижнего уровня. Нижний уровень реализует простую модель, каждый следующий уровень строит новый язык, который может добавлять дополнительную функциональность. Уровни выполнены достаточно независимо — использование метамодели одного уровня должно быть достаточно для описания модели. Рассмотрим каждый уровень подробнее:

Basic Metamodel

Данный слой предоставляет базовую метамодель, вводящую основные понятия любой модели: элемента, узла и связи, а также репозитория как хранилища элементов.

Core Metamodel

На данном уровне вводится понятие модели, репозитория, который теперь хранит модели, а не произвольные элементы. Понятие связи расширено: от нее наследуются три возможные модификации: генерализации (наследования), инстанцирования и ассоциации. На этом же уровне реализовано сохранение модели в файл.

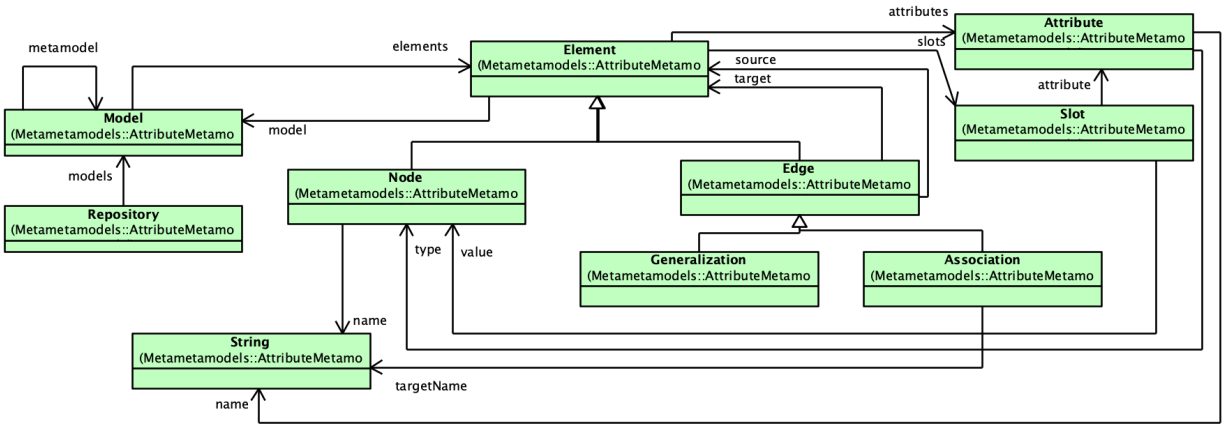

Рис. 4: Attribute Metamodel из [12]

Attribute Metamodel

Вводится понятие атрибута и *слота*. В данном случае атрибут хранит в себе сведения о названии атрибута и типе значения, но не хранит само значение. Для хранения значения используется слот, состоящий из атрибута и его значения. Данный подход позволяет, например, объекту иметь пустое значение для атрибута, а так же переиспользовать атрибут без его значения для разных объектов. Диаграмма уровня представлена на рисунке 4 [12].

Language Metamodel

Языковой уровень создан для введения языковых понятий, например, понятия перечислений (enum).

Infrastructure Metamodel

Инфраструктурный уровень предназначен для добавления элементов, специфических для использования конкретным редактором. Например, здесь может храниться информация о внешнем виде элемента, его положении на экране.

Facade

Уровень фасада предоставляет доступ к верхнему уровню стека метамоделей (например, к инфраструктурному). Данный подход позволяет, например, описать набор действий, которые могут быть использованы при использовании репозитория в качестве сервиса. Кроме этого, фасад позволяет взаимодействовать с системой, даже если верхний уровень меняется в процессе разработки.

2.3. Обзор аналогов

В данном разделе рассматриваются инструменты для работы с языками, основанными на принципе глубокого метамоделирования.

2.3.1. Melanee

Melanee [2] представляет собой среду для моделирования в терминах глубокого метамоделирования, авторами которой выступают непосредственно создатели концепции глубокого метамоделиования [3]. Среда представляет собой расширение над интегрированной средой разработки Eclipse. Внешний вид редактора, а также модель на языке приведена на рисунке 5.

Модель представляется в виде набора онтологических уровней. Элементы на нижнем уровне могут инстанцировать элементы верхнего уровня. Надстрочные числовые подписи обозначают количество возможных дальнейший инстанцирований (мощность). Стоит отметить, что несмотря на то, что данная среда разрабатывается создателями концепции глубокого метамоделирования, принципы работы в среде не полностью соответствуют изначальной модели [3] глубокого метамоделирования. Например, при работе с атрибутами мы видим более

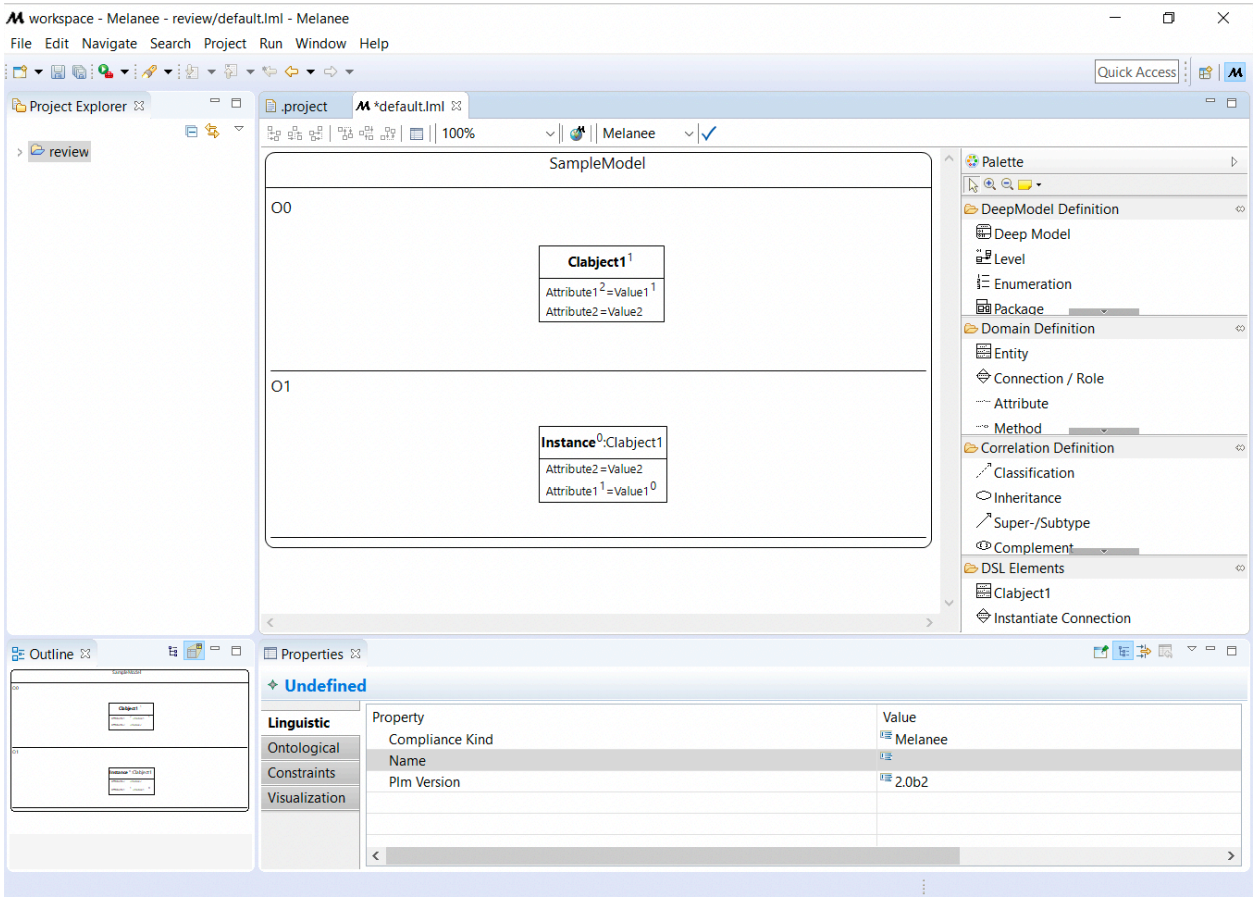

Рис. 5: Melanee

сложную схему инстанцирования — атрибуты могут инстанцироваться несколько раз, причем значение атрибута может иметь свое количество инстранцирований. Тем не менее, на данный момент данная среда представляет собой одну из самых развитых систем для работы с визуальными языками, реализующими принцип глубокого метамоделирования.

2.3.2. metaDepth

Другим интересным инструментом работы с глубоким метамоделированием является metaDepth [7]. Создатели делают упор на исследования возможностей применения глубокого метамоделирования при применении метода разработки, основанного на моделях (model-driven development). Данная среда не имеет графического интерфейса, работа с программой происходит через консоль (см. рисунок 6).

Отсутствие графического интерфейса значительно усложняет при-

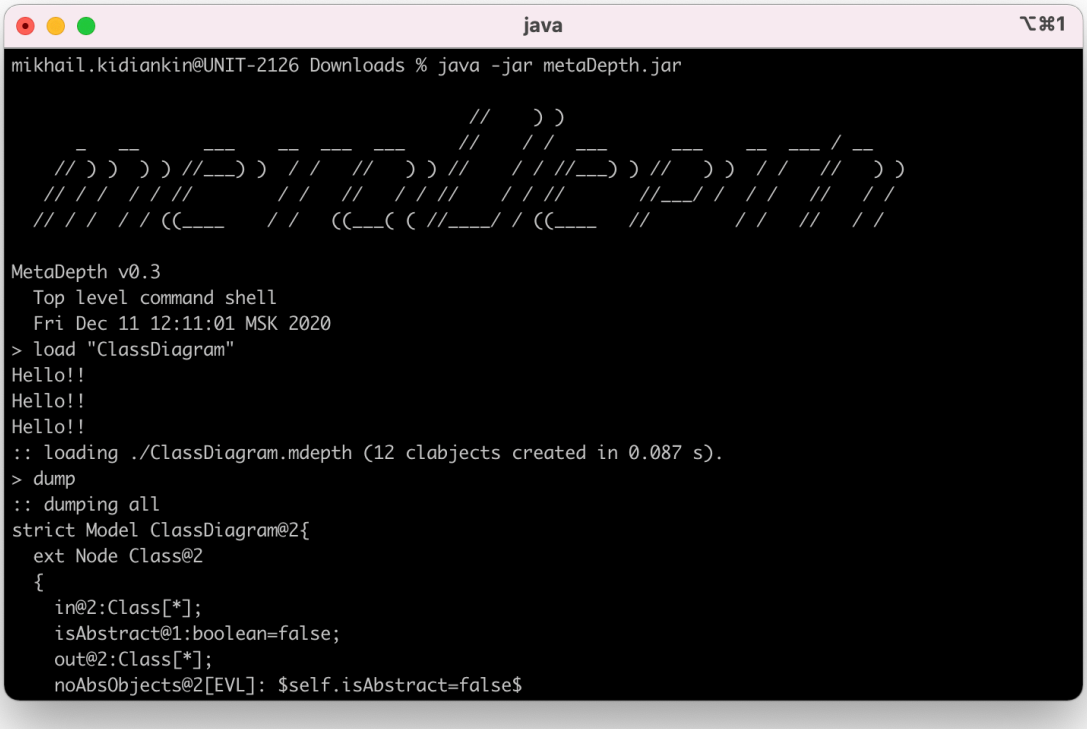

Рис. 6: metaDepth

менение пользователям, не знакомым с командной строкой. Тем не менее, аспекты использования и реализации глубокого метамоделирования могут быть полезны при работе с такого рода языками.

2.3.3. Diagram Predicate Framework

Diagram Predicate Framework (DPF) [18] — проект, одной из основных целей которого является формализация концептов model-driven engineering (MDE). Одна из ключевых особенностей — предлагаемый авторами визуальный язык, который используется для описания ограничений, а также который, помимо прочего, используется для реализации автодополнения. В качестве одного из способов описания моделей DPF реализует принцип глубокого метамоделирования. Особенностью данной среды является наличие нескольких вариантов редактора: один редактор основан на платформе Eclipse, второй — веб-редактор на Java Script. К сожалению, сейчас проект практически не развивается

— последние статьи на сайте проекта [23] датированы 2016 годом, а веб-редактор на сайте проекта не запускается. Тем не менее, данный инструмент интересен наличием веб-редактора, а также применением глубокого метамоделирования.

3. Глубокое метамоделирование в репозитории REAL.NET

Поскольку REAL.NET уже предоставляет большое количество возможностей для работы с визуальными языками и моделированием, было принято решение реализовать глубокое метамоделирование с переиспользованием уровней метамоделирования репозитория REAL.NET.

Однако, поскольку работа с языками, основанными на принципе глубокого метамоделирования, требует специфического редактора (например, для поддержки ортогонального метамоделирования), мы не можем использовать инфраструктурный уровень и уровень фасада. Реализация уровня глубокого метамоделирования производится над языковым уровнем. Таким образом, мы можем переиспользовать, например, работу с атрибутами.

Авторы концепции глубокого метамоделирования предлагают следующую модель (рисунок 7).

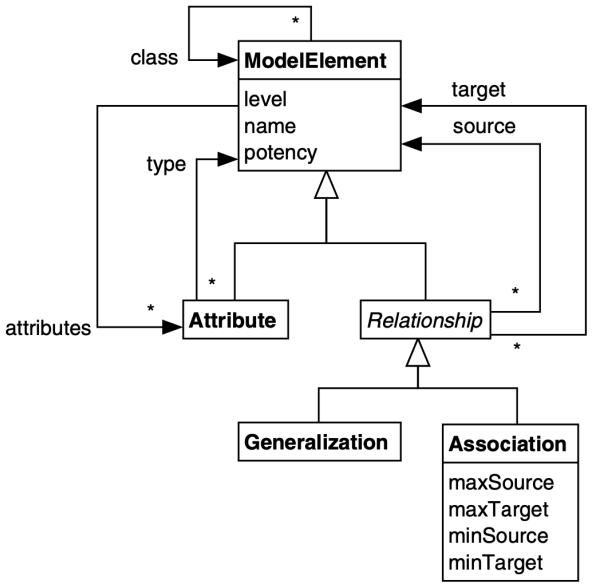

Рис. 7: Модель глубокого метамоделирования из [3]

При попытке наложить ее на уровни REAL.NET (рисунок 4) мы сталкиваемся с некоторыми ограничениями. Необходимо было разрешить конфликты при попытке использования оригинальной модели в

контексте имеющихся уровней репозитория REAL.NET. Для этого были приняты следующие архитектурные решения (рисунок 8):

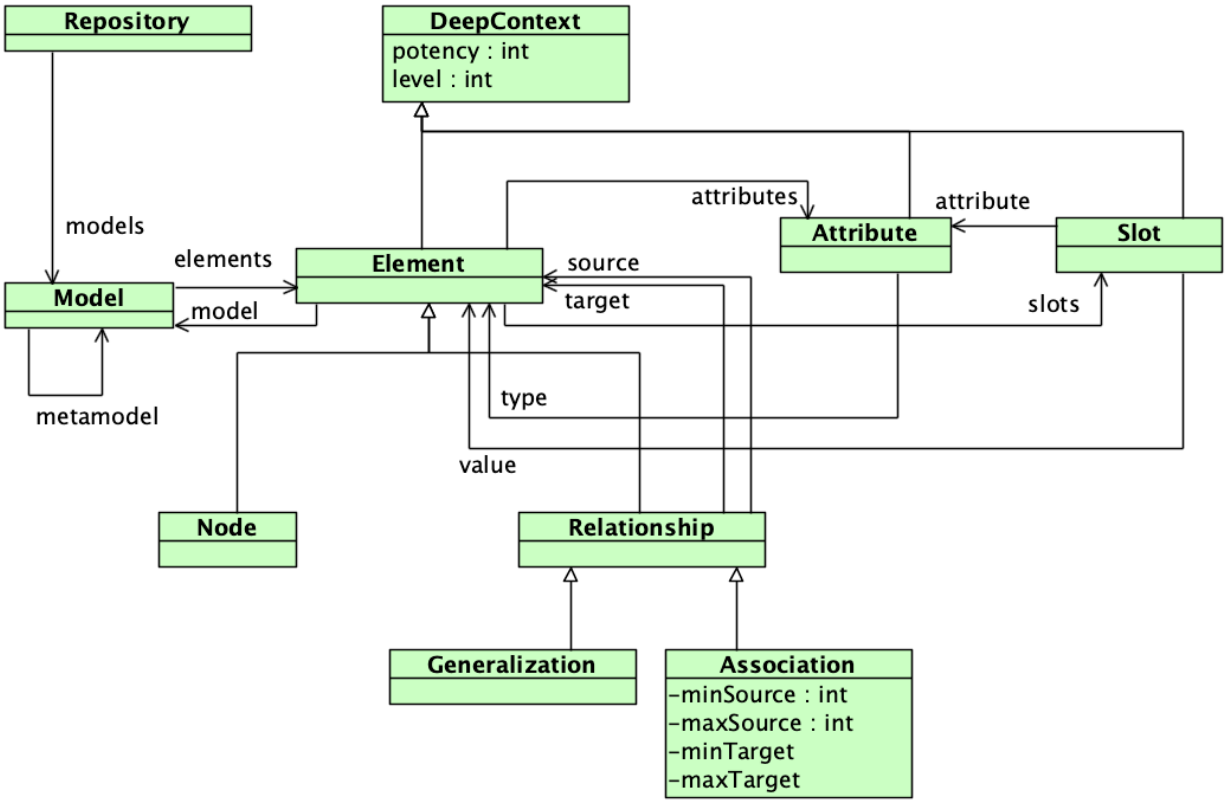

Рис. 8: Модель глубокого метамоделирования в REAL.NET

- Был введен дополнительный элемент контекста глубокого метамоделирования *(DeepContext)*. Это решение было принято, поскольку, в отличие от оригинальной модели глубокого метамоделирования, в REAL.NET атрибуты не наследуются от элементов. Это вызвано тем, что общий контекст описания элементов и атрибутов вне концепции глубокого метамоделирования отсутствует. В глубоком метамоделировании он появляется — и элементы, и атрибуты имеют общие параметры, такие как мощность и уровень, которые и были вынесены в отдельный объект.

- Был добавлен элемент слота. В оригинальной модели глубокого метамоделирования понятие слота отсутствует, однако впоследствии появляется в работах [3]. Как и все прочие элементы модели

глубокого метамоделирования, слоты также наделены контекстом (уровнем и мощностью).

- Было добавлено понятие *модели* как сущности, хранящей в себе элементы. Каждая модель имеет информацию о своей метамодели, которая в свою очередь также является моделью. Кроме этого, для каждого элемента также было добавлено свойство принадлежности некой модели.

- Было добавлено понятие *репозитория* как хранилища моделей. Репозиторий позволяет создавать модели, инстанцировать модели на основе других моделей — метамоделей, а также удалять модели.

Данная модель была реализована уровнем над языковой моделью в сервисе репозитория REAL.NET.

Кроме этого, была реализована специфическая логика работы с моделями глубокого метамоделирования. Была добавлена проверка мощности элемента и автоматический расчет уровня при инстанцировании. Аналогичные ограничения на мощность были добавлены и при реализации инстанцирования атрибутов и слотов элемента. При добавлении слотов учитывается, является ли соответствующий атрибут одиночным или двойственным, так как для одиночных атрибутов их значения могут быть добавлены только при нулевой мощности. Мощность и уровень слота при этом показывают, на сколько уровней значение атрибута сохраняется при инстанцировании элемента. Такой подход позволяет создавать элементы с фиксированными значениями атрибутов, которые не будут добавлены в инстанцируемые элементы, что позволяет уменьшить количество атрибутов, избежать их многократного повторения и упростить элементы пользовательской модели (рисунок 9 из [3]).

Особое внимание было уделено поддержке строгой типизации для значений слота. При добавлении атрибута необходимо указать элемент-тип для будущего значения. Затем, при добавлении слота с соответствующим атрибутом, в качестве значений могут выступать лишь элементы, инстанцированные от элемента-типа атрибута. Для упрощения

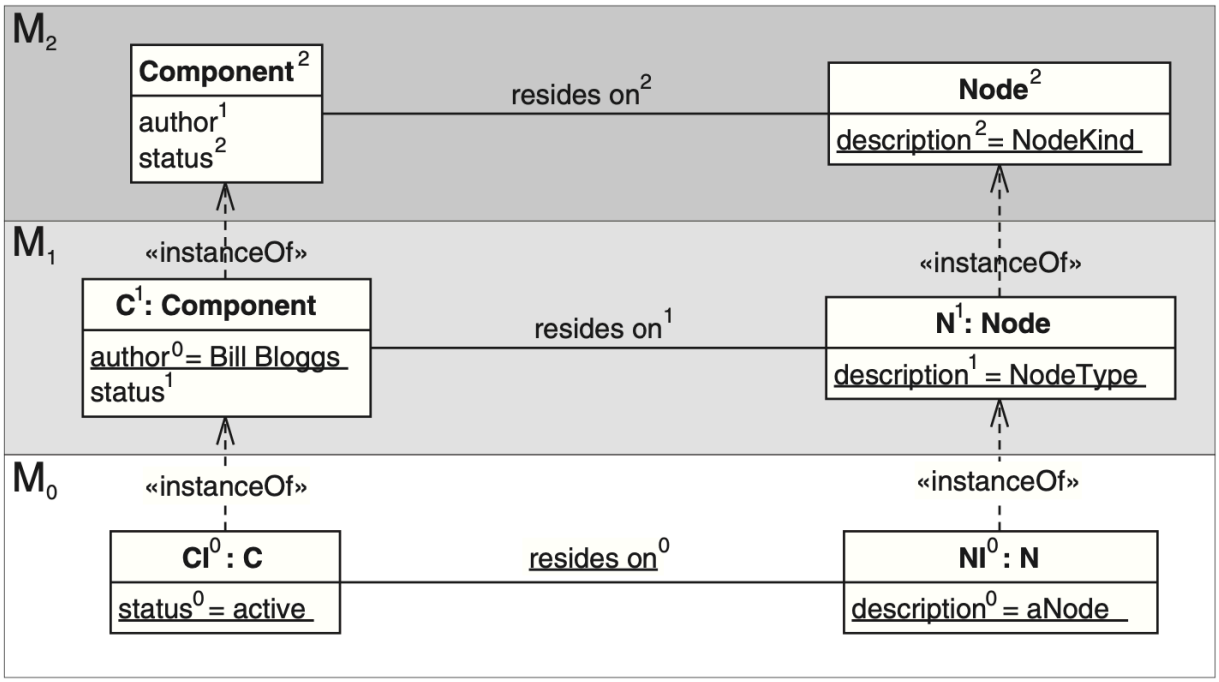

Рис. 9: Инстанцирование атрибутов из [3]

этого ограничения редакторами был реализован метод для поиска всех возможных элементов моделей различных уровней для значения данного атрибута.

Были написаны unit-тесты для проверки основной функциональности, а также была построена демонстрационная модель, содержащая все основные элементы. Разработка уровня глубокого метамоделирования ведется в репозитории [15].

4. Редактор глубокого метамоделирования

Редактор глубокого метамоделирования разделен на два основных компонента — серверный и клиентский (рисунок 10).

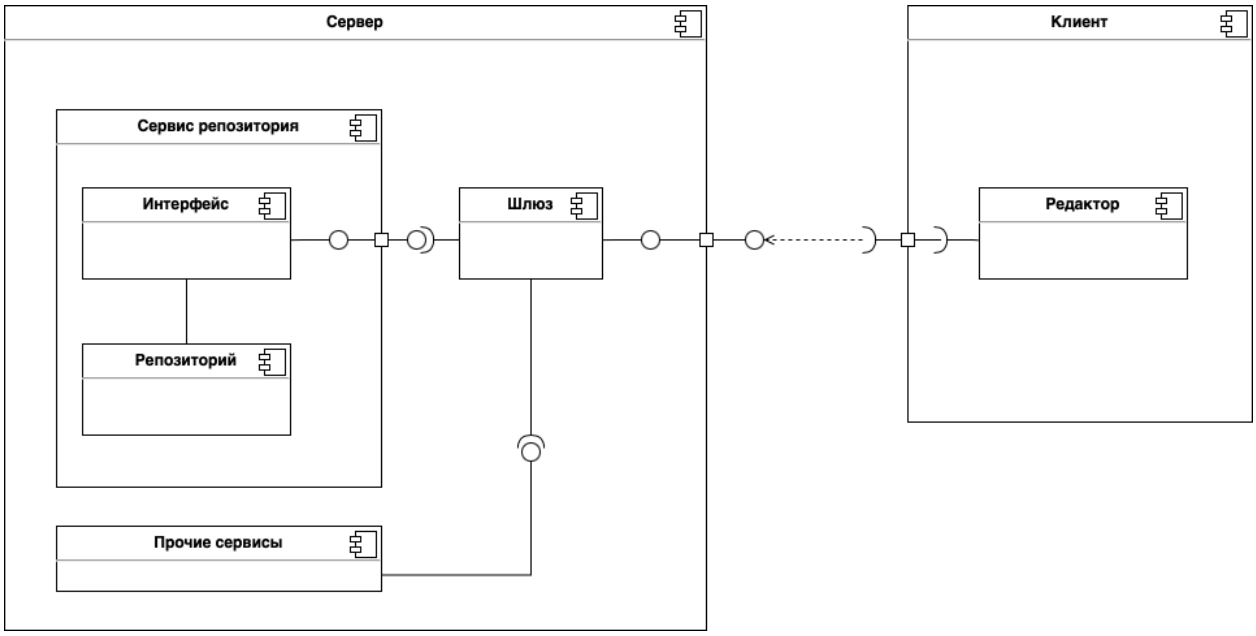

Рис. 10: Клиент-серверная архитектура веб-редактора

В рамках работы для реализации поддержки глубокого метамоделирования необходимо было добавить новую функциональность к уже имеющимся наработкам. Основные работы были посвящены реализации нового программного интерфейса (API) для поддержки глубокого метамоделирования со стороны серверной части, а также усовершенствованиям веб-редактора и его пользовательского интерфейса со стороны клиентской части.

4.1. Сервис API глубокого метамоделирования

Для поддержки взаимодействия веб-редактора с репозиторием REAL.NET используется микросервис, предоставляющий программный интерфейс на основе протокола HTTP. Сервис построен на основе фреймворка ASP.NET [8], как одного из основных для построения веб-сервисов в среде .NET.

Сервис имеет архитектуру Model-Controller. Компонент Model служит для описания основных объектов предметной области работы сервиса, того, какие объекты сервис принимает и какие передает. В случае сервиса интерфейса глубокого метамоделирования объекты представляют собой описания элементов модели глубокого метамоделирования репозитория REAL.NET, такие как узел, ассоциация, атрибут, слот и т.д. Для упрощения конвертации реальных объектов репозитория и их представления в модели была использована библиотека автоматической конвертации Automapper [22].

Компонент Controller служит для описания методов взаимодействия с репозиторием REAL.NET посредством HTTP-запросов. Они оперируют объектами компонента Model. Были описаны два контроллера — один для работы с репозиторием на уровне моделей (получение списка моделей, создание и инстанцирование моделей, удаление моделей), второй для работы с элементами конкретной модели (добавление, инстанцирование, удаление узлов и связей, работа с атрибутами и слотами).

Стоит отметить, что получившиеся контроллеры для работы с моделями глубокого метамоделирования имеют различия с контроллерами классического двухуровневого подхода. Они обусловлены разными моделями описания данных — элементы модели глубокого метамоделирования имеют другую иерархию, имеют дополнительные специфические параметры (уровень, мощность). Кроме этого, отличаются и сами методы, что обусловлено, например, другим подходом к инстанцированию элементов.

Модель и методы были задокументированы с помощью фреймворка Swagger [24]. Разработка ведется в репозитории [16].

4.2. Веб-редактор глубокого метамоделирования

В рамках деятельности проекта REAL.NET разрабатывается новый веб-редактор. Было принято решение разрабатывать редактор глубокого метамоделирования на его основе с целью ускорить разработку нового основного веб-редактора, а также для его апробации.

Редактор представляет собой веб-приложение, написанное на языке TypeScript, который был выбран как язык программирования со строгой типизацией, позволяющий транспилировать код в JavaScript, который поддерживается для работы во всех современных веб-браузерах. Для реализации пользовательского интерфейса используется React.js [21] как одна из самых популярных библиотек, поддерживающие большое количество сторонних библиотек для реализации моделирования диаграмм. Поддержка построения диаграмм реализована на основе библиотеки React Flow [20], так как она, в сравнении с аналогами, имеет развитую систему ограничений, широкие возможности модификации узлов и хорошую документацию.

Для реализации взаимодействия с серверной частью, были описаны объекты модели сервиса репозитория (элементы, модели, узлы и т.д.), а также методы, осуществляющие запросы к контроллерам сервиса репозитория для управления моделями репозитория и элементами.

Кроме этого, в рамках работы над поддержкой глубокого метамоделирования была реализована следующая функциональность пользовательского интерфейса:

- Было реализовано динамическое получение палитры — элементов, которые могут быть помещены на диаграмму. В классическом подходе к метамоделирования элементы палитры — элементы метамодели, в случае же глубокого метамоделирования они представляют собой узлы всех моделей нижних уровня, что и позволяет помещать в модель (инстанцировать) элементы более нижних уровней. Аналогично добавлена возможность выбора типа ассоциации.

- Была добавлена возможность изменения имени элемента. Изменение имени элемента также сопровождается проверкой на уровне репозитория.

- Был добавлен переключатель моделей. Он позволяет работать с несколькими моделями разных уровней, например, задать мета-

модель, а затем сразу же приступить к работе с моделью, получив в палитру новые элементы.

- Были добавлены поля для редактирования специфических для глубокого метамоделирования свойств элементов: уровня, мощности.

- Была добавлена поддержка работы с атрибутами и слотами: отображение имеющихся атрибутов и слотов, диалоговые окна для добавления новых с подсказками для значений требуемых типов.

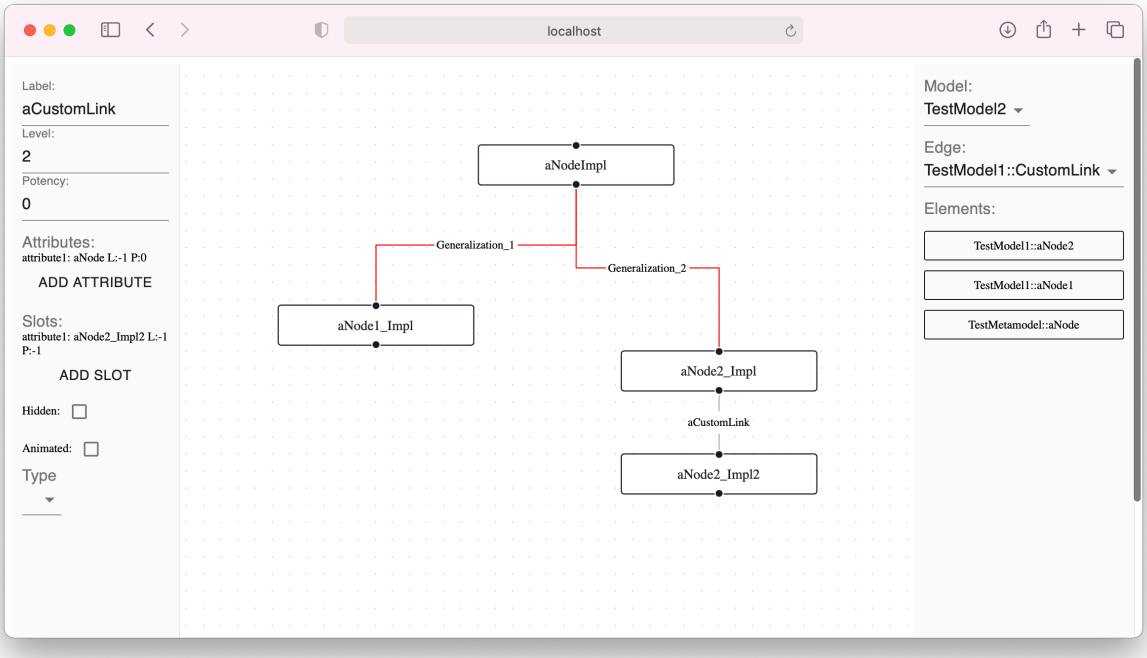

Рис. 11: Редактор глубокого метамоделирования в REAL.NET

Внешний вид редактора представлен на рисунке 11. Разработка ведется в репозитории [14].

5. Апробация

Поскольку глубокое метамоделирование предоставляет новый подход к работе с моделями, и для того, чтобы иметь возможность применять его на постоянной основе, было принято решение провести апробацию полученного решения двумя способами. Во-первых, нужно показать, что глубокое метамоделирование позволяет работать с классическими двухуровневыми визуальными языками. Во-вторых, нужно продемонстрировать новую функциональность на других, более специфичных визуальных языках глубокого метамоделирования.

5.1. Язык двухуровневого метамоделирования

В качестве языка для демонстрации возможностей работы с классическим двухуровневым языком был выбран язык программирования дронов AirSim [19]. Он имеет большое количество элементов метамодели, а также различные виды соединений. Была построена метамодель языка в терминах элементов глубокого метамоделирования, а затем была повторена модель на языке, использующаяся в REAL.NET для демонстрации языка.

Внешний вид редактора с моделью представлен на рисунке 12.

5.2. Языки глубокого метамоделирования

Апробация технологий глубокого метамоделирования была проведена на примере двух языков визуального моделирования. В первом случае был использован язык для построения модели, описанной авторами подхода глубокого метамоделирования для демонстрации глубоко инстанцирования. Во втором случае был описан язык, использующий возможности глубокого метамоделирования для решения проблемы описания подпрограмм в двухуровневом подходе, обнаруженной в предыдущих работах с визуальными языками при реализации среды программирования роботов TRIK Studio [9].

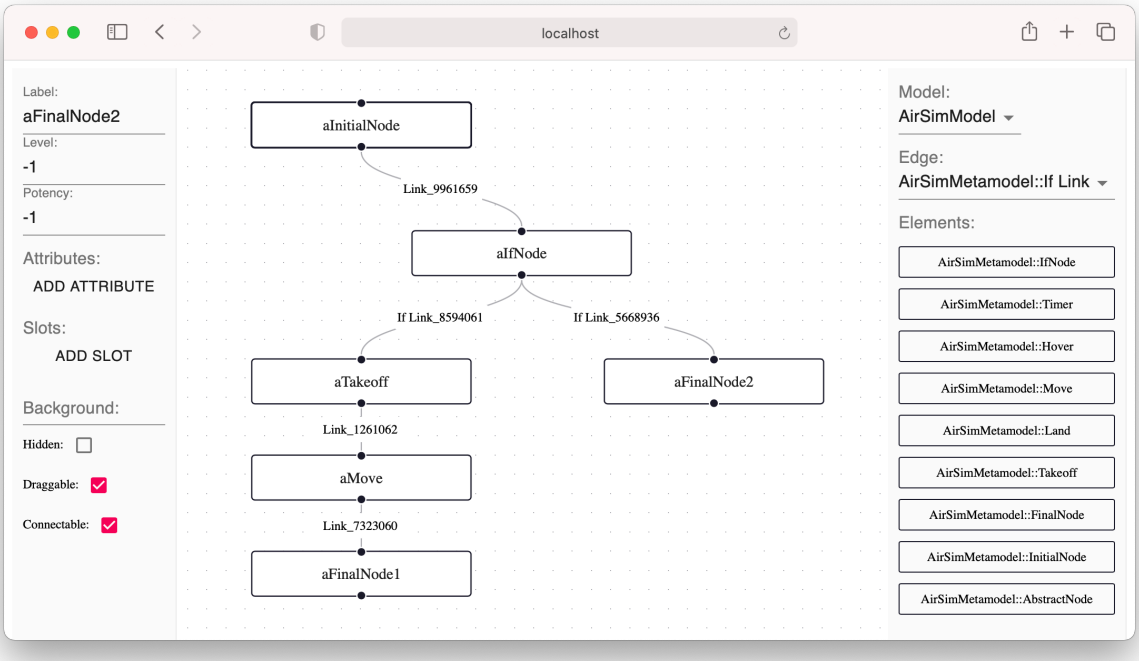

Рис. 12: Модель на языке программирования дронов AirSim

5.2.1. Язык для демонстрации глубокого инстанцирования

Рассмотрим многоуровневую модель (рисунок 13 из [3]), используемую авторами глубокого метамоделирования для описания глубокого инстанцирования. На данной модели изображены концепции глубокого метамоделирования: мощности элементов (узлов и связей), атрибуты с различными мощностями, одиночные и двойственные (с подчеркиванием).

Для демонстрации поддержки глубокого инстанцирования в работе был построен визуальный язык, на котором была описана данная модель. Были описаны различные типы атрибутов, их значения, элементы различных мощностей и уровней. Реализация среднего уровня модели (M1) в редакторе REAL.NET представлена на рисунке 14.

5.2.2. Язык описания подпрограмм управления роботами

При реализации языка для управления роботами в TRIK Studio [9] необходимо было поддержать подпрограммы, представляющие собой от-

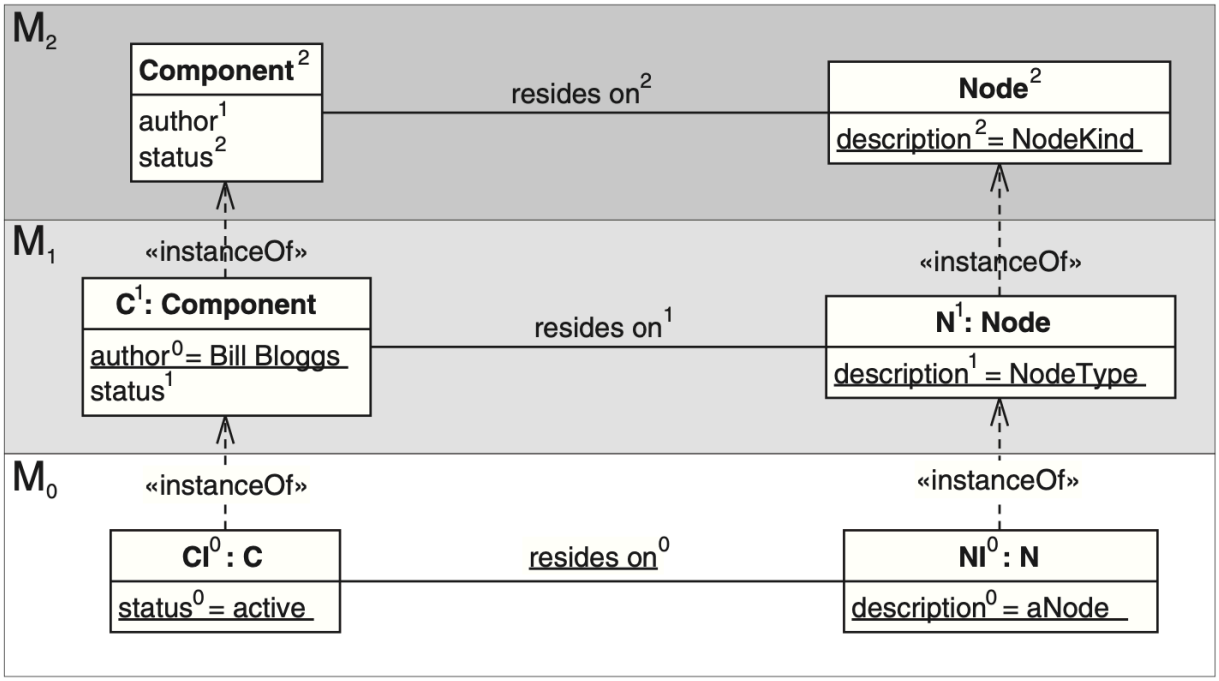

Рис. 13: Глубокое инстанцирование из [3]

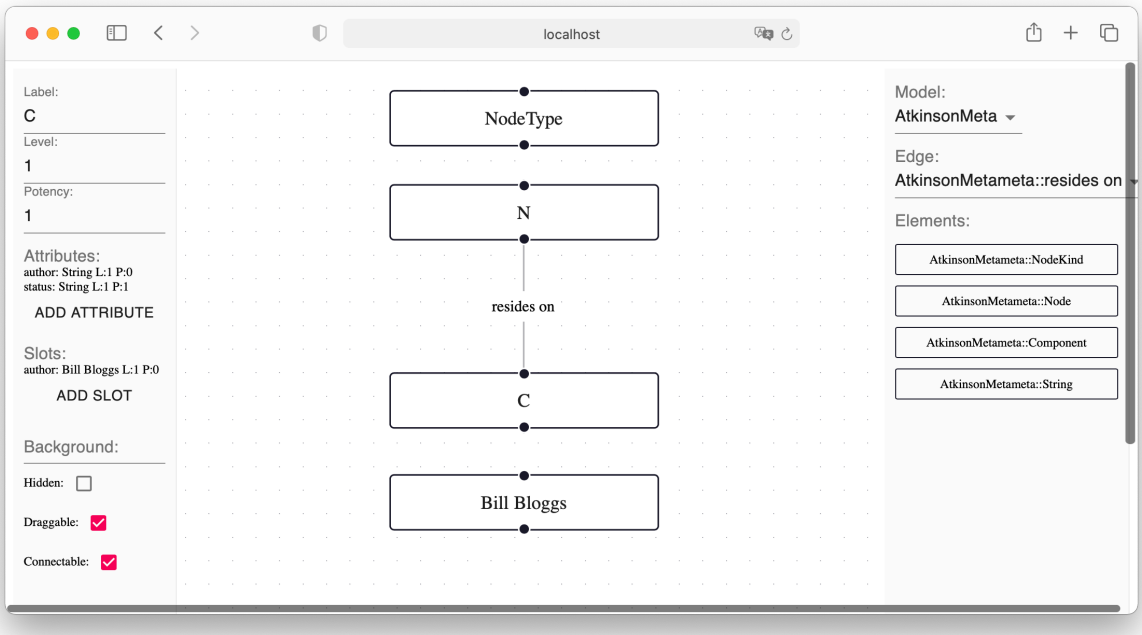

Рис. 14: Глубокое инстанцирование в REAL.NET

дельные модели, которые можно было бы переиспользовать в основной модели управления роботом. Однако при использовании двухуровнего подхода описание подпрограмм сопряжено с некоторыми сложностями. С одной стороны, пользователь использует элементы метамодели при описании подпрограммы, то есть подпрограмма описывается на уровне модели. В то же время пользователь хочет переиспользовать подпрограммы на уровне модели, например, просто выбирая подпрограмму из палитры доступных элементов, что требует описание подпрограммы на уровне метамодели. Таким образом, мы получаем противоречие, показывающее невозможность описать подпрограмму с использованием чистых языковых инструментов моделирования.

Данную проблему, можно решить, например, используя многоуровневую модель, где на одном уровне находятся базовые элементы метамоделирования, на втором — описанные подпрограммы, а на третьем — пользовательская модель. При этом без использования глубокого метамоделирования элементы первого уровня должны быть повторно описаны на втором уровне для использования на третьем пользовательском уровне, что порождает проблему умножения сущностей.

Глубокое метамоделирование позволят избежать данных проблем: для демонстрации был описан следующий язык. На первом уровне были описаны базовые узлы-действия робота (переместиться, включить звук) мощности 1 и элемент подпрограммы мощности 2. Второй уровень используется для описания подпрограмм с помощью элементов первого уровня, при этом на нем элементы подпрограмм имеют мощность 1, а остальные элементы — 0. Тогда на третьем уровне описания основной программы пользователю доступны для инстанцирования элементы обоих уровней, мощности которых больше 0, т.е. все элементы первого уровня (базовые действия) и элементы-подпрограммы второго уровня.

Кроме этого, была продемонстрирована возможность добавления различных атрибутов для создаваемых подпрограмм, что позволяет, например, создавать подпрограммы со специфическими наборами аргументов.

Таким образом, с помощью двух описанных примеров было показано соответствие предлагаемого решения теоретическим концепциям глубокого метамоделирования, а также его практическую применимость.

Заключение

В рамках подготовки выпускной квалификационной работы были достигнуты следующие результаты.

1. Изучены подходы к метамоделированию (двухуровневое, глубокое), проведен обзор сред глубокого метамоделирования (Melanee, metaDepth, DPF), разработана модель глубокого метамоделирования на базе REAL.NET.

2. Реализован веб-редактор глубокого метамоделирования:

   - реализован API для клиент-серверного взаимодействия репозитория REAL.NET и редактора;
   - реализован графический интерфейс редактора, поддерживающий глубокое инстанцирование, редактирование метамоделей нижних уровней; уровня, мощности, атрибутов и слотов элемента.

3. Проведена апробация поддержки глубокого метамоделирования:

   - с помощью двухуровневого визуального языка программирования дронов AirSim;
   - с помощью языков глубокого метамоделирования: языка для демонстрации глубокого инстанцирования и языка описания подпрограмм управления роботами.

С исходным кодом проекта можно ознакомиться на странице организации REAL.NET [13] и репозиториях [14], [15], [16].

Список литературы

[1] Atkinson C. Meta-modelling for distributed object environments // Proceedings First International Enterprise Distributed Object Computing Workshop. — 1997. — P. 90–101.

[2] Atkinson Colin, Gerbig Ralph. Flexible Deep Modeling with Melanee // Modellierung 2016, 2.-4. März 2016, Karlsruhe - Workshopband / Ed. by Stefanie Betz, Ulrich Reimer. — Vol. P-255 of LNI. — GI, 2016. — P. 117–122. — Access mode: https://dl.gi.de/20.500.12116/843.

[3] Atkinson Colin, Kühne Thomas. The Essence of Multilevel Metamodeling // Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools. — Berlin, Heidelberg : Springer-Verlag, 2001. — P. 19–33.

[4] Booch Grady, Rumbaugh James, Jacobson Ivar. Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series). — Vol. 10. — 1999. — 01.

[5] Dobing Brian, Parsons Jeffrey. How UML is used // Commun. ACM. — 2006. — 05. — Vol. 49. — P. 109–113.

[6] Erwin Ben, Cyr Martha, Rogers Chris. LEGO Engineer and RoboLab: Teaching Engineering with LabVIEW from Kindergarten to Graduate School. — Vol. 16. — 2000. — 01.

[7] Lara Juan, Guerra Esther. Deep Meta-modelling with MetaDepth. — Vol. 6141. — 2010. — 06. — P. 1–20.

[8] Microsoft. Официальная страница ASP.NET [Электронный ресурс]. — 2021. — Режим доступа: https://dotnet.microsoft.com/apps/aspnet (дата обращения: 25.04.2021).

[9] Mordvinov Dmitry, Litvinov Yurii, Bryksin Timofey. TRIK Studio: Technical Introduction // PROCEEDINGS OF THE 20TH CONFER-

ENCE OF OPEN INNOVATIONS ASSOCIATION (FRUCT 2017) / Ed. by S Balandin. — Proceedings Conference of Open Innovations Association FRUCT. — Canada : IEEE Canada, 2017. — P. 296–308.

[10] OMG. OMG Unified Modeling Language (OMG UML) Infrastructure. — 2011. — 08.

[11] OMG. Meta-Object Facility Specification. — 2019. — 10.

[12] REAL.NET. GitHub репозиторий документации REAL.NET [Электронный ресурс]. — 2020. — Режим доступа: https://github.com/REAL-NET/docs (дата обращения: 13.12.2020).

[13] REAL.NET. GitHub организация проекта [Электронный ресурс]. — 2021. — Режим доступа: https://github.com/REAL-NET (дата обращения: 25.04.2021).

[14] REAL.NET. GitHub репозиторий проекта клиентской части веб-редактора REAL.NET [Электронный ресурс]. — 2021. — Режим доступа: https://github.com/REAL-NET/web-editor-frontend (дата обращения: 25.04.2021).

[15] REAL.NET. GitHub репозиторий проекта репозитория REAL.NET [Электронный ресурс]. — 2021. — Режим доступа: https://github.com/REAL-NET/Repo (дата обращения: 25.04.2021).

[16] REAL.NET. GitHub репозиторий проекта серверной части веб-редактора REAL.NET [Электронный ресурс]. — 2021. — Режим доступа: https://github.com/REAL-NET/web-editor-backend (дата обращения: 25.04.2021).

[17] REAL.NET Web — Web-based multilevel domain-specific modeling platform / Mikhail Kidiankin, Yurii Litvinov, Valeria Ivasheva et al. // PROCEEDINGS OF FIFTH CONFERENCE ON SOFTWARE ENGINEERING AND INFORMATION MANAGEMENT (SEIM-2020). — 2020. — P. 45–51.

[18] Rossini Alessandro. Diagram Predicate Framework meets Model Versioning and Deep Metamodelling : Ph. D. thesis / Alessandro Rossini. — 2011. — 10.

[19] Небогатиков И.Ю., Литвинов Ю.В. Создание визуального предметно-ориентированного языка программирования дронов для симулятора AirSim // Современные технологии в теории и практике программирования. — Российская Федерация : Издательство Санкт-Петербургского Государственного Политехнического Университета, 2018. — 4. — С. 66–68.

[20] Официальная страница библиотеки React Flow [Электронный ресурс]. — 2021. — Access mode: https://reactflow.dev (online; accessed: 25.04.2021).

[21] Официальная страница библиотеки React.js [Электронный ресурс]. — 2021. — Режим доступа: https://ru.reactjs.org (дата обращения: 25.04.2021).

[22] Официальная страница проекта Automapper [Электронный ресурс]. — 2021. — Access mode: https://automapper.org (online; accessed: 25.04.2021).

[23] Официальная страница проекта Diagram Predicate Framework [Электронный ресурс]. — 2021. — Режим доступа: https://ict.hvl.no/dpf/ (дата обращения: 25.04.2021).

[24] Официальная страница проекта Swagger [Электронный ресурс]. — 2021. — Access mode: https://swagger.io (online; accessed: 25.04.2021).