Санкт-Петербургский государственный университет

Математическое обеспечение
и администрирование информационных систем
Системное программирование

Кидянкин Михаил Владимирович

Микросервисная архитектура DSM-платформы REAL.NET Web

Курсовая работа

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

Санкт-Петербург
2020

Оглавление

Введение

В современном мире часто может возникнуть задача, при которой процессы удобно описывать языком программирования, специально созданным для данной задачи. В качестве такого предметно-ориентированного языка может выступать визуальный язык программирования, применяемый для упрощения и наглядности. Достаточно часто визуальные языки находят свое применение в областях, где пользователи не являются профессиональными программистами, например, в робототехнике для школьников [9].

Поскольку создание новой среды программирования для каждого специфичного языка может быть затратно, имеет смысл рассматривать универсальные среды, в которых можно описывать новые языки и использовать их вместе с универсальными редакторами, генераторами и другими компонентами.

На кафедре системного программирования СПбГУ разрабатывается DSM-платформа (domain-specific modeling platform, платформа предметно-ориентированного моделирования) REAL.NET [11], основанная на принципе глубокого метамоделирования. При её разработке активно учитывается опыт создания среды QReal [20], которая, в свою очередь, использует стандартный двухуровневый подход. С помощью REAL.NET было реализовано несколько проектов: среда для программирования дронов симулятора AirSim [13], среда для управления «умной» теплицей [8]. Реализовано два редактора — простой кроссплатформенный и более сложный для ОС Windows.

На данный момент REAL.NET имеет монолитную архитектуру, которая, в отличие от микросервисной, представляет меньше возможностей для расширения и интеграции со специфичными для разных языков компонентами. Кроме этого, REAL.NET не имеет веб-редактора, преимущества которого перед нативнми приложениями заключаются в том, что он не требует установки, работает почти независимо от операционной системы и позволяет пользователю работать с нескольких устройств без дополнительной синхронизации. Для решения этих за-

дач было принято решение разработать веб-редактор с использованием микросервисной архитектуры для создания гибкой и масштабируемой системы работы с визуальными языками.

1. Постановка задачи

Целью данной курсовой работы является разработка серверной части DSM-платформы REAL.NET Web, а именно разработка архитектуры системы и нескольких сервисов. Для достижения цели были поставлены следующие задачи:

1. Проанализировать возможные подходы к организации микросервисных систем, произвести их сравнение.
2. Разработать некоторые микросервисы: авторизации, хранения данных, генерации, работы с метамоделями.
3. Проанализировать подходы к организации коммуникации микросервисов, реализовать выбранный.
4. Разработать схему простого развертывания решения.

2. Обзор

2.1. Существующие аналоги

В данном разделе рассматриваются системы работы с визуальными языками, имеющими веб-редактор или реализующими принцип глубокого метамоделирования. Данные системы были выбраны как одни из самых популярных в своей сфере (Node-RED для моделирования, Diagram Predicate Framework как инструмент глубокого метамоделирования), либо же как инструменты, значительно повлиявшие на развитие REAL.NET (QReal). Приведенные примеры призваны продемонстрировать актуальность работы.

QReal

QReal [20] — DSM-платформа, ранее разрабатываемая на кафедре Системного программирования СПбГУ. Имеет практическое применение в системе моделирования для роботов TRIK Studio [9]. Имеет веб-реализацию WMP [12]. Сейчас проект QReal практически не развивается, но результаты работы над ним во многом легли в основу при разработке REAL.NET. QReal, в отличие от REAL.NET, использует принцип простого двухуровнего метамоделирования. Таким образом, WMP не может послужить для апробации подхода глубокого метамоделирования, в то время как REAL.NET эту цель преследует.

Изначально WMP реализовывал монолитную архитектуру редактора, однако в процессе разработки было принято решение перехода на микросервисную архитектуру.

Node-RED

Одним из самых популярных инструментов для моделирования является среда Node-RED [17]. Редактор реализован на JavaScript и имеет функциональность генерации кода по заданному правилу. Кроме этого,

существует большое количество примеров и дополнительных библиотек, расширяющих функциональность среды. Однако пользователю не представляется простой возможности задания собственного языка (например, описание новых элементов требует создания как минимум трех конфигурационных файлов на JavaScript и HTML), при этом для метамоделирования используется двухуровневый подход.

Внешний вид веб-редактора представлен на рисунке 1.

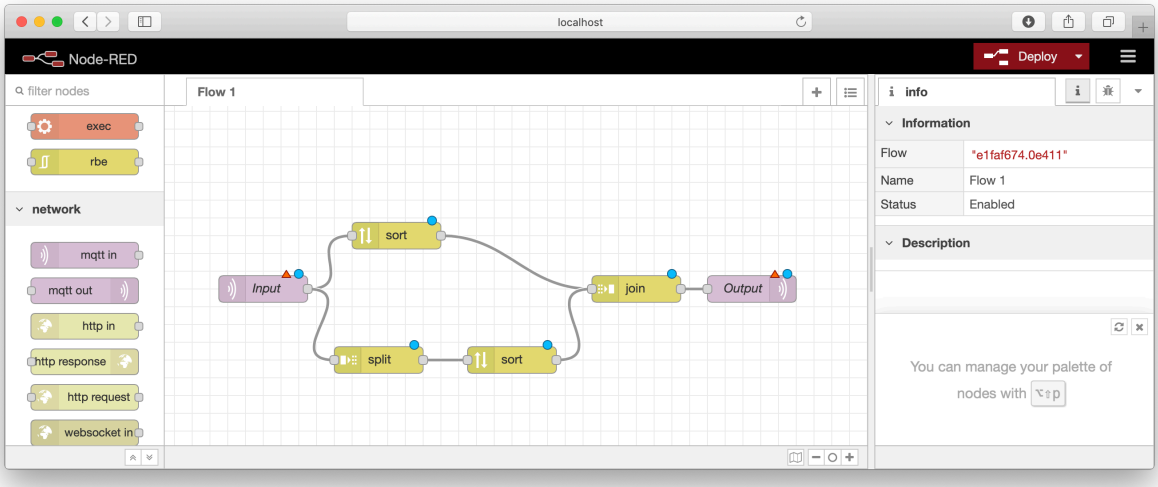

Рис. 1: Веб-редактор Node-RED

Diagram Predicate Framework

Diagram Predicate Framework (DPF) [15] — инструмент, позволяющий описывать модели на визуальном языке, заданном с помощью принципа глубокого метамоделирования. DPF распространяется в качестве плагина для среды программирования Eclipse, имеет веб-редактор. DPF позиционирует себя как инструмент для исследований методик разработки, управляемых моделями (model-driven development). Таким образом, DPF представляет особый интерес как один из самых продвинутых редакторов, реализующих принципы глубокого метамоделирования.

2.2. Используемые технологии

2.2.1. REAL.NET

REAL.NET — среда для создания и использования визуальных языков, разрабатываемая на кафедре системного программирования СПбГУ, главной особенностью которой является реализуемый принцип глубокого метамоделирования [11].

Существует несколько способов метамоделирования. Достаточно распространенным является подход двухуровнего метамоделирования, в котором правила языка задаются лишь одной метамоделью. В качестве примера такого языка можно привести UML [4]. Несмотря на свою простоту, этот подход имеет существенные недостатки, связанные с недостаточной выразительностью двухуровнего метамоделирования. Эти недостатки хорошо описаны [1] и активно обсуждаются в сообществе, занимающемся вопросами метамоделирования. В качестве альтернативы предлагаются подходы, допускающие описание визуальных языков с помощью нескольких метамоделей, некоторые из которых могут являться метамоделями друг друга. Такой подход называется многоуровневым (глубоким) метамоделированием [2].

Разрабатывается на платформе Microsoft .NET.

Среди основных компонентов среды можно выделить:

1. Репозиторий — основной компонент REAL.NET, отвечающий за задание моделей и метамоделей визуального языка. Написан на языке F#. С помощью репозитория поддерживаются все операции работы с моделями: создание и удаление моделей, добавление и удаление элементов модели, задание атрибутов элементов. Именно этот компонент для работы с моделями реализует принцип глубокого метамоделирования.

2. Два редактора моделей — один простой кроссплатформенный, второй — более продвинутый, работающий только на ОС Windows. Написаны на C#. Позволяют производить операции с моделями

с помощью их графического представления. Продвинутый редактор имеет функциональность расширения с помощью плагинов.

В рамках предыдущей работы [10] для репозитория был описан API и реализован сервис, позволяющий управлять репозиторием с помощью HTTP-запросов, с расчетом на то, что его можно будет использовать в качестве сервиса для будущего веб-редактора.

2.2.2. ASP.NET

ASP.NET — разрабатываемый Microsoft набор инструментов, позволяющих создавать веб-приложения на платформе Microsoft .NET. Каждое приложение ASP.NET представляется в схеме MVC (Model-View-Controller), в которой разделяются модели, описывающие основные объекты, с которыми работает приложение на уровне фасада; представление, описывающее отображение данных; контроллер, отвечающий за обработку запросов. При разработке сервиса используется такая же парадигма, за исключением того, что в ней отсутствует представление.

Данный фреймворк был выбран как основной для разработки сервисов, поскольку он позволяет разрабатывать их на платформе .NET, на которой уже разработаны компоненты REAL.NET. Кроме этого, ASP.NET представляет большое количество встроенных возможностей, которые могут быть полезны при разработке, имеет большое количество документации и активное сообщество.

2.2.3. Docker

Поскольку развертывание системы, состоящей из нескольких практически независимых сервисов, которые могут иметь сложные зависимости и требования, является сложной задачей, были принято решение использовать контейнеры. Для обеспечения работы с контейнерами был использован один из самых популярных инструментов развертывания и управления приложениями — Docker [16]. Этот инструмент имеет большое количество документации, поддерживает работу в популярных

операционных системах (Windows, macOS, Linux). Кроме этого, Docker позволяет собственными средствами описывать процессы развертывания многоконтейнерных приложений. Работа с Docker-контейнерами поддержана во многих популярных системах непрерывной интеграции, что позволяет, например, автоматизировать процессы конвейеризации приложений.

3. Описание решения

3.1. Микросервисная архитектура

В рамках разработки серверной части веб-редактора было принято решение использовать микросервисную архитектуру. Данный подход подразумевает разделение проекта на несколько небольших модулей (микросервисов) и обладает следующими свойствами:

1. Каждый микросервис — отдельное самостоятельное приложение, которое может разрабатываться на своем языке, с использованием специфических технологий.

2. Каждый микросервис выполняет свою специализированную задачу. Таким образом, микросервисная архитектура — архитектура с чётким разделением обязанностей.

3. Сообщение между микросервисами происходит с помощью заданного протокола. При этом каждый микросервис имеет явный публичный интерфейс, ограничивающий набор действий, которые он может совершать.

4. Каждый микросервис стремится использовать свою базу данных. Использование общих ресурсов избегается.

Для универсальной платформы моделирования как REAL.NET применение микросервисов может упростить интеграцию дополнительных, специфичных для конкретной реализации языка, компонентов. Для разных задач, реализуемых с помощью REAL.NET, могут потребоваться дополнительные компоненты, такие как генераторы кода, компоненты взаимодействия с устройствами и т.д. Использование микросервисной архитектуры позволит использовать компоненты, технологии которых могут сильно отличаться от используемого в среде моделирования. Такой подход позволит упростить интеграцию с компонентами, специфичными для языков. Кроме этого, при публикации приложения

в сети Интернет, использование микросервисов может облегчить масштабирование, например, путем размещения микросервисов на разных серверах.

Безусловно, микросервисная архитектура имеет свои недостатки [3], например, сложность начального этапа разработки — требуется разделить функциональность и выбрать протоколы. Однако с учетом представленных достоинств, а также ссылаясь на опыт при разработке WMP, микросервисный подход был выбран как основной для проекта.

Диаграмма планируемой архитектуры приведена на рисунке 2.

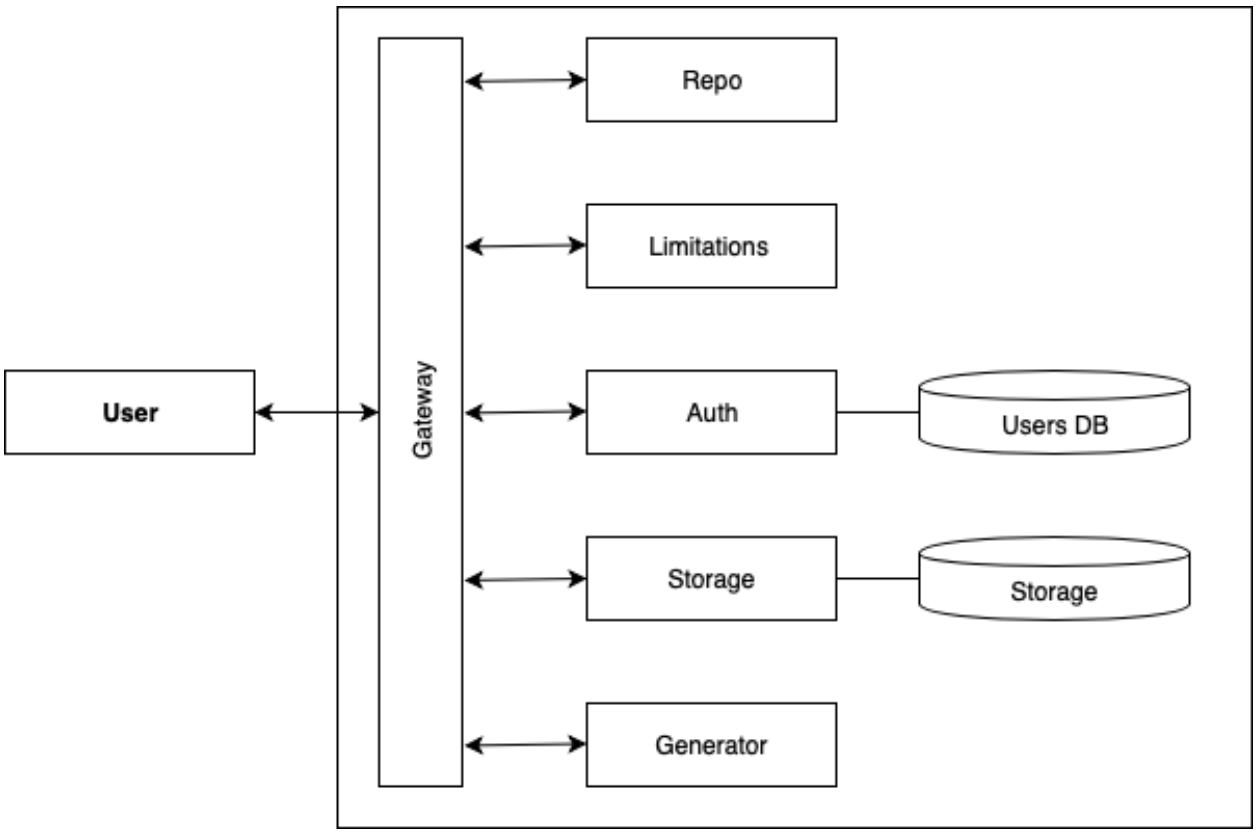

Рис. 2: Диаграмма архитектуры серверной части веб-редактора

Сообщения между сервисами передаются с помощью REST-запросов.

3.2. Сервисы

3.2.1. Репозиторий REAL.NET

В рамках предыдущих работ [10] репозиторий REAL.NET уже был оформлен в виде RESTful сервиса. Этот сервис находился в составе монолитного проекта REAL.NET. Был произведен процесс выделения репозитория (как компонента REAL.NET, не сервиса) в отдельный проект. Микросервис репозитория был организован из уже разработанного сервиса, в котором при помощи git-подмодулей используется уже отделенная версия репозитория. Таким образом, репозиторий и его сервис теперь представляют отдельные приложения.

3.2.2. Авторизация

Для обеспечения пользовательского контроля была выбрана аутентификация с использованием JWT токена [7]. Данный подход представляет универсальный способ аутентификации для различных методов авторизации — стандартная авторизация через микросервис авторизации, авторизация через сторонние ресурсы, которая может быть реализована в будущем. Сервис авторизации хранит в себе данные о пользователях, зарегистрированных в приложении. По запросу на аутентификацию, включающему в себя логин и пароль, сервис возвращает JWT токен, используемый в дальнейшем шлюзом для авторизации пользователя.

Сервис позволяет добавлять новых пользователей, т.е. поддерживает регистрацию.

JWT токен имеет ограниченный срок действия, поэтому его утечка не является настолько же критичной, как утечка пароля. При этом пароль (а точнее его хэш с модификакотором входа хэш-функции) используется и хранится лишь в одном сервисе и используется только для аутентификации. Таким образом, данный подход повышает общую безопасность системы.

3.2.3. Шлюз

Шлюз — основной координирующий сервис системы. Считая, что каждый прочий микросервис разрабатывается отдельно и запускается на своем сервере, возникает необходимость организовать их совместную работу путем направления всех клиентских запросов по правильному маршруту. Благодаря такому подходу, ни клиенту, ни разработчику, не нужно знать адрес и порт микросервиса.

Для упрощения маршрутизации было принято решение описать шлюз с помощью библиотеки Ocelot [19], рекомендуемой официальной документацией ASP.NET для реализации шлюзов. Библиотека позволяет описывать маршруты шаблонами, а также управлять тем, для каких сервисов необходима аутентификация, а для каких нет. Все настройки могут быть в одном конфигурационном JSON-файле.

Для того, чтобы воспользоваться возможностью указания требований аутентификации, на шлюзе был реализован процесс проверки JWT токена. Для запросов ко всем сервисам, кроме сервиса авторизации, было потребовано иметь корректный JWT токен. Всем остальным сервисам нет необходимости проверять токены, если мы потребуем исполнять запросы, пришедшие только из шлюза. Таким образом, мы можем гарантировать аутентификацию на этапе маршрутизации запросов.

3.2.4. Хранилище

Для реализации возможности работы с моделями разными пользователями с различных устройств был реализован сервис хранилища, имеющий следующие функции:

1. Сохранение текущего состояния репозитория с указанием пользователя и времени.
2. Загрузка сохраненной версии репозитория для данного пользователя.
3. Получение данных о всех сохраненных версиях.

4. Возможность сохранения вместе с репозиторием дополнительных сведений.

Стоит отметить, что хранилище лишь управляет версиями репозитория — непосредственное хранение файлов сериализованных репозиториев осуществляется сервисом репозитория. Такой подход позволяет разделить обязанности по работе с репозиториями и конкретными моделями, при этом не создает нагрузку на сеть и повышает безопасность системы.

3.2.5. Генератор

Несмотря на то, что для каждой конкретной реализации визуального языка, требующего генерацию кода, генераторы будут отличаться, многие общие части у генераторов присутствуют — части сервиса, отвечающие за получение данных из репозитория о модели. В связи с этим был разработан шаблон сервиса генерации, который получает данные модели для более удобного использования уже специфическими генераторами.

Другими участниками проекта был реализовал генератор на основе Razor [14], позволяющий производить генерацию по шаблону и объекта-модели.

3.3. Коммуникация микросервисов

Использование микросервисной архитектуры предполагает несколько возможностей для организации коммуникации между микросервисами.

Например, используется сервис *Шлюз*, задача которого заключается в перенаправлении запросов от клиентов и самих сервисов к нужным микросервисам. Такой подход позволяет скрыть от клиентской части и от самих миросервисов структуру системы, что дает дополнительную гибкость при изменении архитектуры (например, можно разделить один микросервис на два, при этом для клиентской части изменения не будут видны).

При этом было принято решение не использовать дополнительных сервисов коммуникации между микросервисами, таких как *Сервисная шина предприятия* [5], свойственных для сервис-ориентированных архитектур. Мотивация отказа от них заключается в том, что при добавлении сервисов, специфичных для конкретных реализаций визуальных языков, нужно будет требовать от них интеграцию с шиной, что усложняет внедрение новых микросервисов.

Таким образом, в нашей архитектуре доступ ко всем сервисам осуществляется через шлюз, при этом каждый сервис сам должен осуществлять взаимодействие с системой без помощи дополнительных сервисов.

3.4. Развертывание решения

Одной из сложностей, возникающих при использовании микросервисной архитектуры, является процесс развертывания решения. Для упрощения этой процедуры было принято решение использовать Docker совместно с сервисами непрерывной интеграции. Процесс можно описать следующими шагами:

1. Разработчик вносит изменения в проект на GitHub.
2. Сервис непрерывной интеграции собирает и упаковывает каждый сервис в образ Docker-контейнера.
3. Образы выкладываются в хранилище GitHub Package Registry.

Теперь для запуска приложения необходимо лишь запустить небольшой скрипт, который при необходимости загрузит образы микросервисов и запустит их каждый в своем контейнере. Кроме этого, Docker имеет специальный инструмент Compose для более гибкой настройки развертывания для более сложных проектов, который может быть использован в дальнейшем. Таким образом, для тестового локального запуска проекта нет необходимости даже собирать его. Если возникает необходимость сборки проекта, то она тоже решается использованием

простого скрипта. Благодаря такому подходу мы полностью избавляемся от необходимости установки зависимостей, ручной настройки портов. Для запуска необходимо лишь иметь установленный Docker.

Поиск удобного процесса развертывания решения осуществлялся до начала разработки сервисов. Был разработан прототип, который продемонстрировал применимость описанного решения, после чего оно применялось с первых этапов разработки.

4. Апробация

В рамках проекта по созданию веб-версии REAL.NET другими участниками разрабатывались клиентские части приложения:

- Универсальный редактор, использующий стандартный подход к организации модели через блоки и связи (рис. 3).
- Редактор системы «Умный дом», в основе которого лежит табличный подход к организации пользовательской модели (рис. 4).

Данные редакторы используют сервисы серверной части для своей работы. Возможность реализовать полную функциональность редакторов на основе сервисов серверной части служит её апробацией.

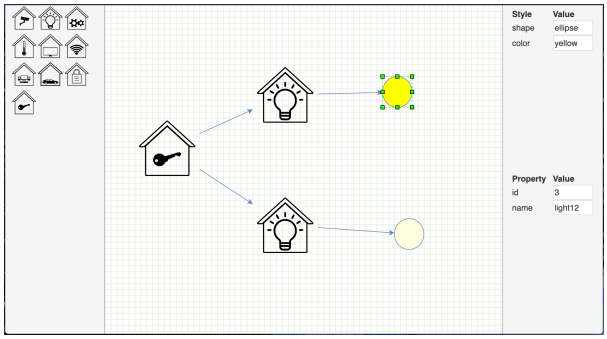

Рис. 3: Универсальный редактор

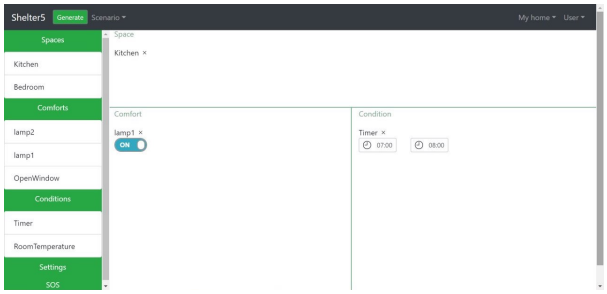

Рис. 4: Редактор системы «Умный дом»

Кроме этого, разработанные сервисы оснащены средством документирования Swagger [18], который позволяет производить запросы к серверной части в целях тестирования.

Заключение

В рамках курсовой работы были достигнуты следующие результаты:

1. Проанализированы подходы к организации микросервсиной архитектуры.
2. Разработаны некоторые микросервисы: авторизации, репозитория, хранилища, генератора, шлюз.
3. Проанализированы подходы к организации коммуникации микросервисов, реализован выбранный.
4. Разработана схема простого развертывания решения.

Данный проект имеет несколько возможных вариантов дальнейшего развития. Например, это могут быть задачи, направленные на упрощения взаимодействия с пользователем (например, реализация авторизации через сторонние сервисы), задачи, направленные на производительность системы (добавление кэширования, асинхронности), либо же на развитие возможностей интеграции среды с другими перспективными проектами.

Результаты данной работы являются частью готовящейся к публикации в сборнике трудов конференции SEIM-2020 статьи «REAL.NET Web — web-based multilevel domain-specific modeling platform», Mikhail Kidiankin, Yurii Litvinov, Valeria Ivasheva, Elizaveta Kuzmina, Yuniya Kim, Angelina Chizhova.

Данный проект является проектом с открытым исходным кодом, разработка ведется в репозитории [6].

Список литературы

[1] 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.

[2] Atkinson Colin, Kühne Thomas. In Defence of Deep Modelling // Inf. Softw. Technol. — Vol. 64. — Butterworth-Heinemann, 2015. — P. 36–51. — URL: http://dx.doi.org/10.1016/j.infsof.2015.03.010 (online; accessed: 15.12.2019).

[3] Baskarada Sasa, Nguyen Vivian, Koronios Andy. Architecting Microservices: Practical Opportunities and Challenges // Journal of Computer Information Systems. — 2018. — 09. — P. 1–9.

[4] Booch Grady, Rumbaugh James, Jacobson Ivar. The unified modeling language user guide. — Addison-Wesley, 2005. — P. 475.

[5] The Enterprise Service Bus: Making service-oriented architecture real / Marc-Thomas Schmidt, B. Hutchison, P. Lambros, Rob Phippen // IBM Systems Journal. — 2005. — 02. — Vol. 44. — P. 781 – 797.

[6] GitHub репозиторий проекта [Электронный ресурс]. — 2019. — URL: https://github.com/REAL-NET/web-editor-backend (дата обращения: 15.12.2019).

[7] JSON Web Token (JWT) : RFC : 7519 / RFC Editor ; Executor: Bradley J. Jones, M., N. Sakimura : 2015. — May. — URL: https://www.rfc-editor.org/refs/ref7519.txt (online; accessed: 15.12.2019).

[8] Kuzmina Elizaveta, Litvinov Yurii. Implementation of "smart greenhouse" visual programming tool using deep metamodeling // CEUR Workshop Proceedings. — Vol. 2372. — RWTH Aahen University, 2019. — 1. — P. 67–73.

[9] Mordvinov Dmitry, Litvinov Yurii, Bryksin Timofey. TRIK Studio: Technical Introduction // PROCEEDINGS OF THE 20TH CONFERENCE 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] Кидянкин М. В. Инструмент клиент-серверного взаимодействия с REAL.NET. — Санкт-Петербург : Санкт-Петербургский государственный университет, 2019.

[11] Литвинов Ю.В., Кузьмина Е.В. и др. Среда предметно-ориентированного визуального моделирования REAL.NET // СПИСОК-2017. Материалы 7-й всероссийской научной конференции по проблемам информатики. — 2017. — С. 80–89.

[12] Литвинов Ю.В., Черниговская Л.А. Разработка онлайн-метаредактора QReal-Web // Материалы научно-практической конференции студентов, аспирантов и молодых учёных "Современные технологии в теории и практике программирования". — Российская Федерация : Издательство Санкт-Петербургского Государственного Политехнического Университета, 2016. — С. 29–31.

[13] 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.

[14] Официальная документация проекта Razor [Электронный ресурс]. — 2020. — URL: `https://docs.microsoft.com/en-us/aspnet/core/razor-pages` (online; accessed: 20.05.2020).

[15] Официальная страница проекта Diagram Predicate Framework

[Электронный ресурс]. — 2019. — URL: http://dpf.hib.no (дата обращения: 15.12.2019).

[16] Официальная страница проекта Docker [Электронный ресурс]. — 2019. — URL: http://docker.com (дата обращения: 13.12.2019).

[17] Официальная страница проекта Node-RED [Электронный ресурс]. — 2019. — URL: https://nodered.org (дата обращения: 15.12.2019).

[18] Официальная страница проекта Swagger [Электронный ресурс]. — 2020. — URL: https://swagger.io (online; accessed: 15.05.2020).

[19] Реализация шлюзов API с помощью Ocelot [Электронный ресурс]. — 2019. — URL: https://docs.microsoft.com/ru-ru/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gateways-with-ocelot (дата обращения: 13.12.2019).

[20] Терехов А.Н., Брыксин Т.А., Литвинов Ю.В. QReal: платформа визуального предметно-ориентированного моделирования // ПРОГРАММНАЯ ИНЖЕНЕРИЯ. — № 6. — Новые технологии, 2013. — С. 11–19.