УДК 004

ОНТОЛОГИЧЕСКИЙ ПОДХОД К ПРОЕКТИРОВАНИЮ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ

Е. А. Малых[1] [0009-0008-0730-1603], А. А. Блощук[2] [0000-0001-6683-5973], О. М. Атаева[3] [0000-0003-0367-5575]

[1–3]*Московский университет имени С. Ю. Витте, г. Москва, Россия*

[3]*Федеральный исследовательский центр «Информатика и управление» РАН, г. Москва, Россия*

[1]warior227@yandex.ru, [2]abloshuk@muiv.ru, [3]oataeva@frccsc.ru

Аннотация

Несмотря на широкое использование микросервисной архитектуры в разработке программных систем, в настоящее время не существует формализованного подхода, обеспечивающего согласованное и гарантированное взаимодействие микросервисов на уровне передаваемых данных, что приводит к возникновению интеграционных ошибок и усложняет сопровождение распределенных систем. В работе предложен подход к организации взаимодействия микросервисов на основе онтологического моделирования, обеспечивающего формализацию структур данных и автоматизированную валидацию сообщений. Предложен метод преобразования в онтологических моделей формальных описаний схем данных основанный на спецификации схем GraphQL. Он позволяет автоматизировать процесс валидации данных и снизить количество интеграционных ошибок. Разработана также онтологическая модель, обеспечивающая анализ зависимостей между микросервисами и механизм валидации контрактов сообщений.

Практическая значимость работы заключается в достижении согласованного описания микросервисов, операций и форматов сообщений в результате использования онтологического подхода. Представление онтологии в виде графа позволяет анализировать зависимости между микросервисами и упрощает сопровождение крупных распределенных систем.

***Ключевые слова**: онтология, GraphQL Schema, интеграция данных, микросервисная архитектура, потоки сообщений, валидация данных, межсервисное взаимодействие, онтологическая модель, согласованность данных, управление схемами, шина данных.*

ВВЕДЕНИЕ

На сегодняшний день микросервисная архитектура является наиболее приоритетным выбором в проектировании распределенных программных систем. В них функционал разделен на отдельные автономные модули, называемые микросервисами. Они могут разрабатываться и использоваться независимо друг от друга. Актуальной проблемой является отсутствие согласованного обмена данными между микросервисами. При эволюции сервисов формат сообщений меняется, что приводит к интеграционным ошибкам. Нами рассмотрен подход к проектированию онтологической модели описания всей программной системы, который предусматривает взаимодействие на уровне абстракции, без привязки к конкретным языкам интеграции [1]. Такая онтология позволяет автоматически конвертировать модель в формальную схему и механизмы валидации. Практическое применение показано на примере онтологии в рамках информационной системы анализа научных текстов с целью их дальнейшей обработки.

БЛИЗКИЕ ПО ТЕМАТИКЕ ИССЛЕДОВАНИЯ

В работе [2] рассмотрена технология удаленного вызова процедур (Remote Procedure Call, RPC) как способ валидации сообщений. Авторы проанализировали существующие технологии с позиций повышения эффективности и производительности. Основное внимание уделено эффективности метода и формализации структуры данных преимущественно в контексте технологии Protocol Buffers. Преимуществами этого подхода являются строгая типизация и производительность, недостатком – привязка к конкретной технологии. Не учитываются также семантика и межсервисные зависимости, нет описания архитектуры системы.

В [3] исследована проблема согласованного управления данными в микросервисной системе, которая состоит из нескольких десятков сервисов. Автор

предложил использовать онтологию как инструмент описания метаданных. Предложенная модель позволяет абстрагироваться от каких-либо реализаций СУБД и накладывает ограничения на изменение данных микросервисами. На основе этой модели сформирован единый формат данных, согласованный между микросервисами. Но все же эта модель не является формализацией взаимодействия, описанием операций, потоков сообщений или же каких-либо контрактов сообщений. При этом, наглядно продемонстрировано применение онтологического подхода для решения задачи унификации и управления данными в микросервисной архитектуре.

Таким образом, существующие подходы, как правило, ориентированы на формализацию структур данных и не охватывают архитектурные аспекты взаимодействия микросервисов. Ограничения существующих решений указывают на необходимость разработки альтернативного подхода.

МОДЕЛЬ

В большинстве случаев взаимодействие между сервисами описывают фрагментно, причем API и форматы сообщений отдельно. Зависимости и полные цепочки взаимодействий между сервисами в каком-либо формализованном виде вовсе отсутствуют [4, 5]. В результате такого проектирования нет единой модели, которая позволила бы надежно управлять не только структурой самих данных, но и архитектурой сторонней системы, а именно нет ответа, какой сервис с каким может или должен взаимодействовать.

Нами была спроектирована онтология в редакторе Protégé, которая представляет собой единую концептуальную модель взаимодействия микросервисной архитектуры [6]. В ней описаны базовые сущности микросервиса (см. рис. 1), а также взаимодействие в виде операций и сообщений. На рисунке для облегчения понимания показан упрощенный вид спроектированной онтологии, представленной в формате PlantUML-схемы. Класс «Сервис» может предоставлять внешние методы взаимодействия в виде класса «API». Показана связь между классом API и классом операции, которая является самим методом API. Кроме того в формате сущностей описаны сообщения, схемы сообщений, поля схемы, а также тип поля. Указаны достаточно подробные связи между всеми сущностями.

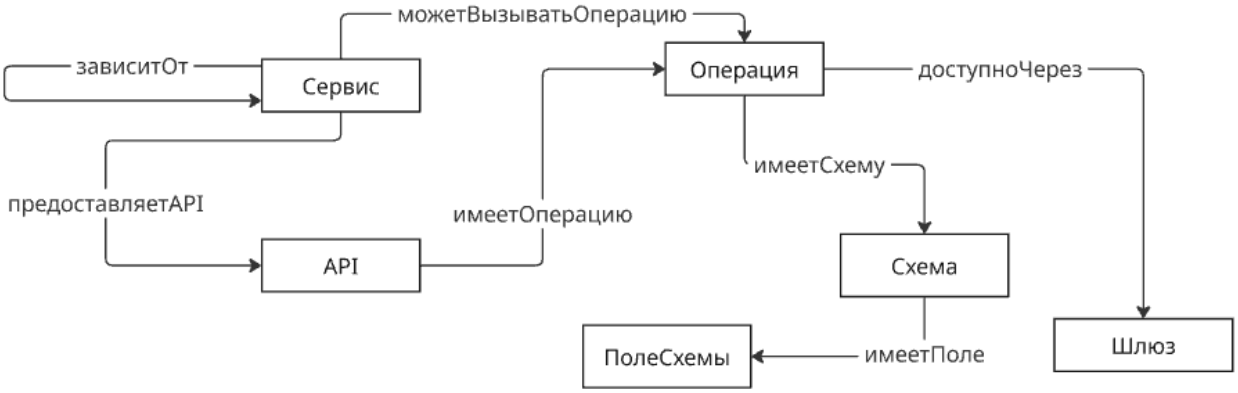

Рис. 1. Упрощенный вид спроектированной онтологии взаимодействия микросервиса.

Выбор онтологического подхода обусловлен его способностью объединять описания структуры данных и семантики взаимодействий в рамках единой модели [7]. По сравнению с существующими решениями предлагаемый онтологический метод является более сложным и затратным на начальном этапе проектирования, что связано с необходимостью построения и сопровождения онтологической модели. Но эти затраты компенсируются повышением согласованности архитектурных решений, а также обеспечивается централизованное управление схемами сообщений. Дополнительным преимуществом является формальный анализ межсервисных зависимостей. Такой анализ упрощает эволюцию схем без нарушения целостности архитектуры.

С точки зрения производительности дополнительные накладные расходы отсутствуют. Проверка сообщений выполняется на основе ограничений, заранее сгенерированных в формате GraphQL Schema. Обращение к онтологической модели при обработке сообщений не требуется. Основными задачами этого процесса являются обеспечение контроля жизненного цикла [8] всех микросервисов и проверка сообщений по согласованной схеме [9].

ПРАКТИЧЕСКАЯ ЗНАЧИМОСТЬ

Для подтверждения практической применимости подхода была отдельно сформирована часть микросервисной системы, описанной в рамках разработанной онтологии. Рассматриваемый фрагмент включает три микросервиса: сервис

формирования эмбеддингов документов (EmbeddingService), сервис построения графов знаний (GraphService) и сервис загрузки (UploadService).

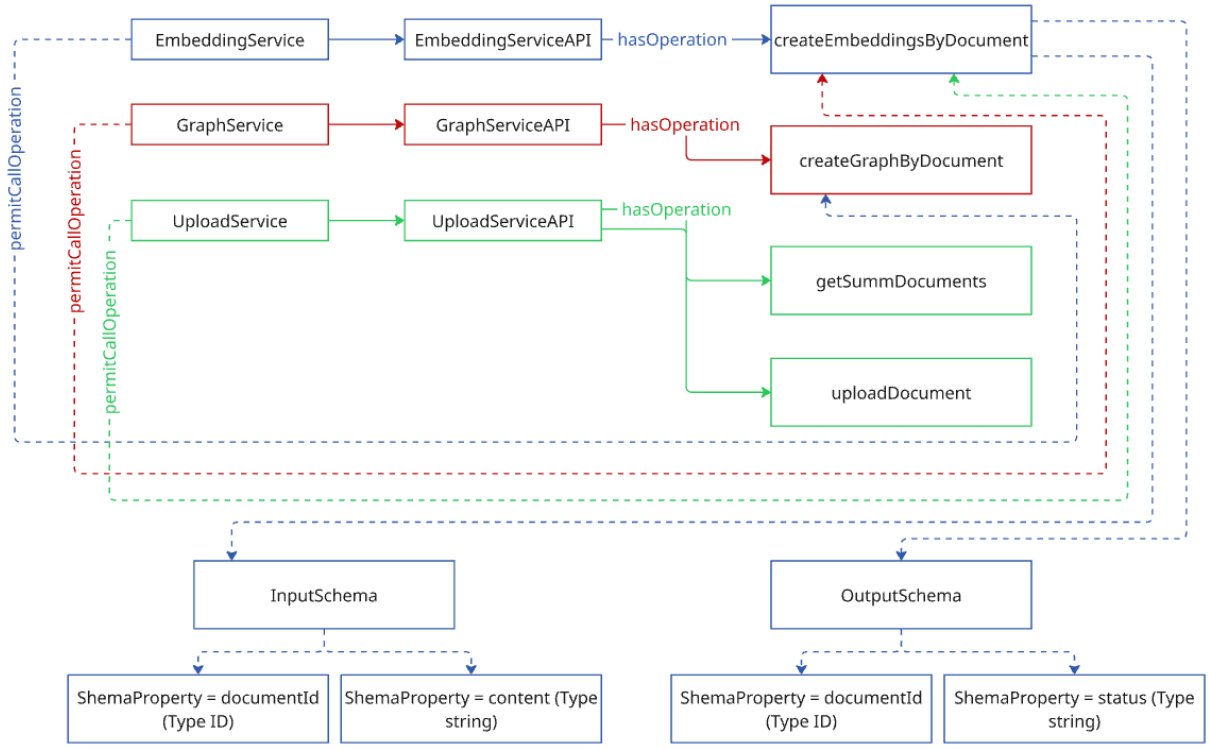

Рис. 2. Онтологическое представление сервисов, API-операций и правил их взаимодействия.

Диаграмма на рис. 2 отражает структуру сервисов, их программные интерфейсы, операции, а также схемы входных и выходных данных. Кроме того, на ней показаны правила допустимых вызовов операций между различными сервисами. Цвет элементов используется для обозначения принадлежности к сервису.

Сервис, обозначенный синим цветом, отвечает за обработку эмбеддингов документов. Он представлен компонентом EmbeddingService и интерфейсом EmbeddingServiceAPI. В интерфейсе определена операция createEmbeddingsByDocument. Она формирует векторное представление документа. Красным цветом обозначено все, что связано с сервисом построения графовой структуры знаний. Он представлен компонентом GraphService и интерфейсом GraphServiceAPI. Через этот интерфейс выполняется операция createGraphByDocument. Она формирует граф знаний на основе содержимого

документа. Зеленым цветом отмечен сервис работы с документами, содержащий компонент UploadService и интерфейс UploadServiceAPI. В интерфейсе определены операции uploadDocument и getSummDocuments. Первая выполняет загрузку документа, вторая возвращает сводную информацию о документах.

Связь между интерфейсами и операциями обозначена отношением hasOperation. Оно показывает, что интерфейс содержит определенные операции. Таким образом фиксируется структура сервиса, его интерфейса и доступных функций.

Пунктирные линии на диаграмме обозначают правила вызова операций между сервисами. Они связаны отношением permitCallOperation определяющим какие операции могут вызывать другие операции системы. Каждая пунктирная линия имеет цвет соответствующего сервиса, что показывает, какой сервис инициирует вызовы. В итоге устанавливаются зафиксированы допустимые взаимодействия между компонентами системы.

В нижней части диаграммы представлены схемы данных. Они описывают структуру входных и выходных параметров операций. Входная схема обозначена как InputSchema, она содержит свойства documentId и content. Идентификатор документа имеет тип ID. Содержимое документа имеет строковый тип. Выходная схема обозначена как OutputSchema, она содержит свойства documentId и status. Идентификатор использован для связи результата с исходным документом. Поле status отражает состояние выполнения операции.

Онтология описывает структуру сервисов, операции, типы данных и правила взаимодействия. Однако сама по себе модель не обеспечивает контроль выполнения этих правил во время работы системы. Для этого требуется механизм, который реализует заданные ограничения при обработке сообщений. В рассматриваемом подходе таким интерфейсом выступает GraphQL-сервер. Он принимает запросы и перенаправляет их к соответствующим сервисам. На этом этапе важно проверить корректность сообщения и допустимость выполнения операции. Проверка должна учитывать несколько условий. Сообщение должно соответствовать структуре GraphQL-схемы, а также необходимо убедиться, что вызываемая операция разрешена и соответствует правилам взаимодействия сервисов.

Механизм формирования ограничений

Возникает необходимость разработки механизма валидации сообщений и проверки разрешений операций. Такой механизм должен использовать информацию из онтологии и применять ее при обработке GraphQL-запросов. Это позволит обеспечить контроль корректности взаимодействия сервисов и соблюдение архитектурных ограничений системы.

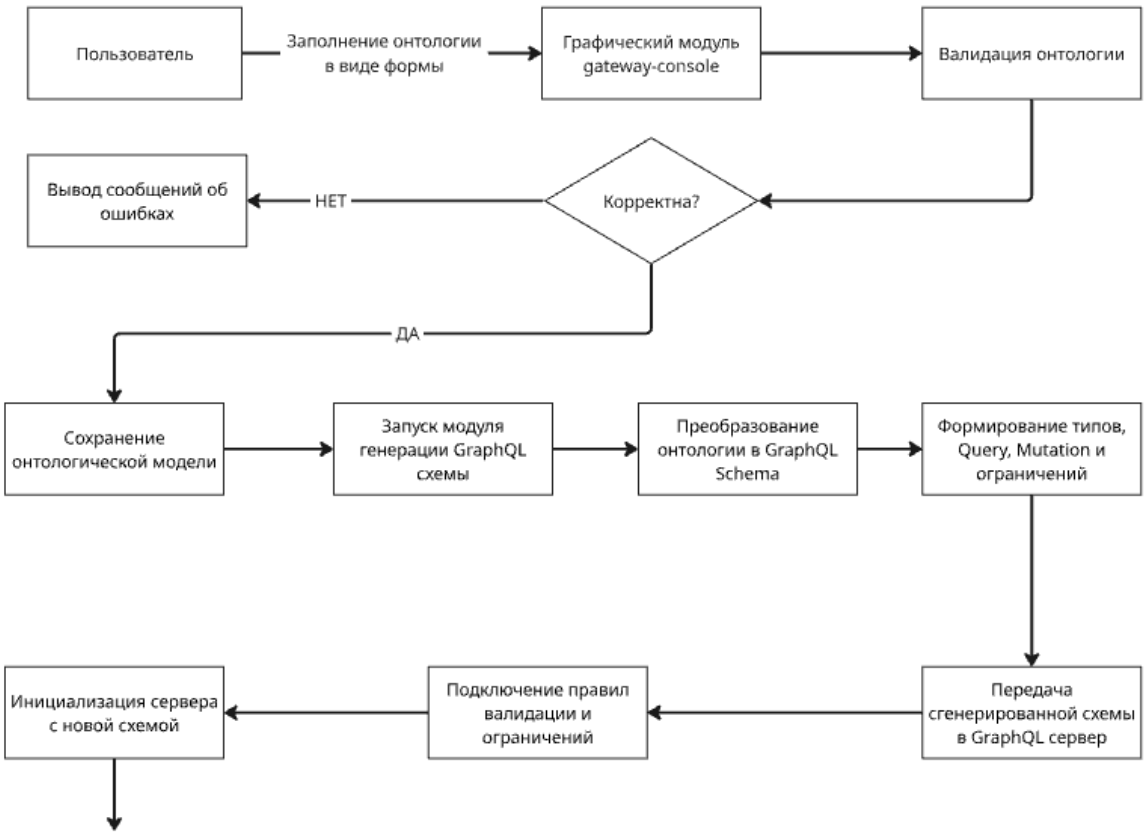

Рис. 3. Алгоритм формирования и применения ограничений GraphQL-сервера на основе онтологической модели.

На рис. 3 представлен алгоритм формирования ограничений для GraphQL-сервера на основе онтологической модели. Диаграмма показывает последовательность действий от ввода данных пользователем до инициализации сервера с новой схемой [10]. Процесс начинается с пользователя, который работает со специально разработанным графическим интерфейсом системы. Через форму он вводит или меняет элементы онтологии. Эти данные передаются в графический модуль gateway-console, использующийся для редактирования структуры модели и управления ее параметрами.

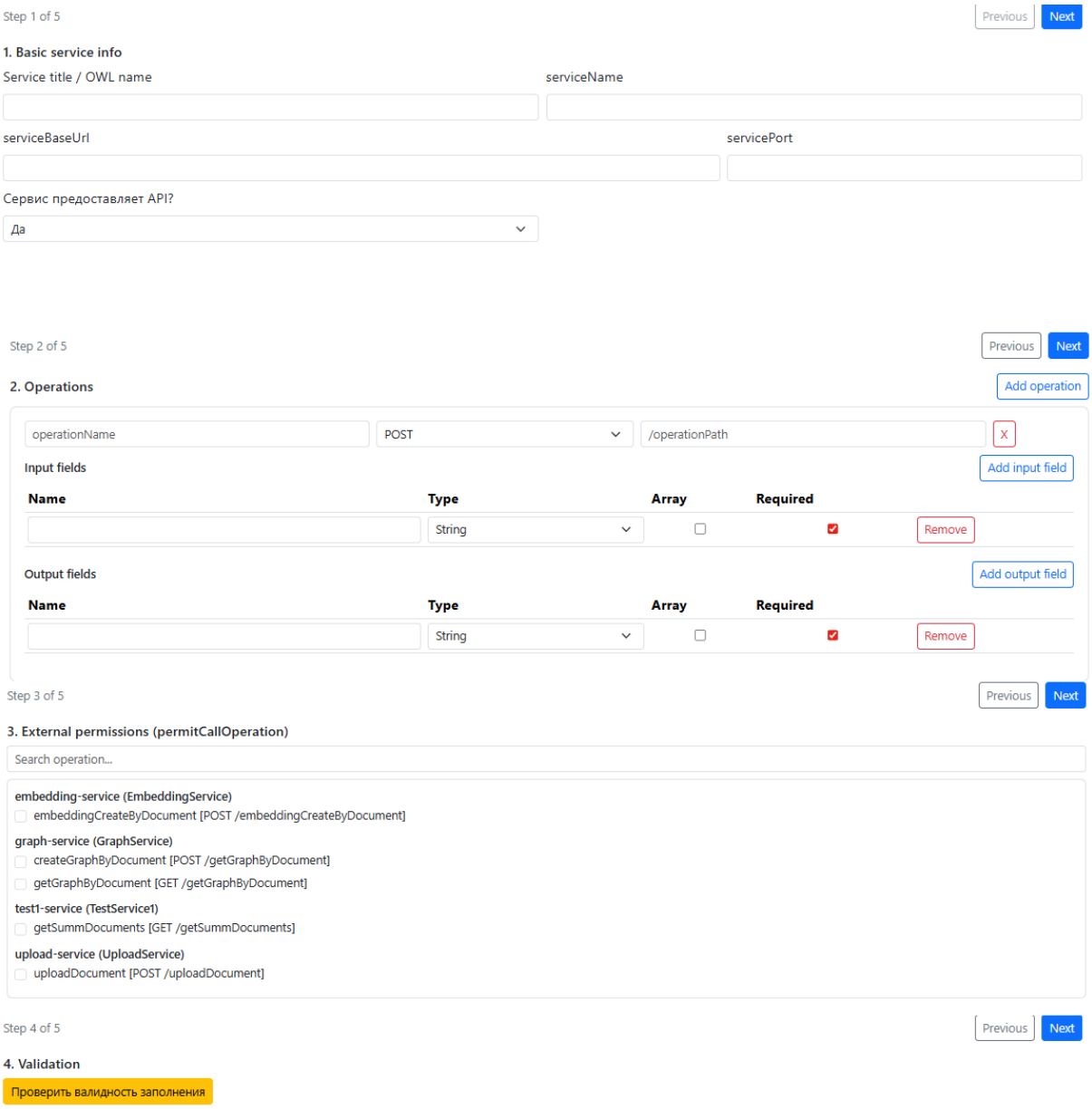

Рис. 4. Интерфейс пошагового заполнения параметров микросервиса и его операций в модуле формирования онтологии.

Интерфейс реализован в виде последовательности шагов и продемонстрирован на рис. 4. На первом этапе задаются основные параметры сервиса, включая его имя, адрес и порт. Далее пользователь описывает операции сервиса, указывает путь вызова, метод запроса и структуру входных и выходных данных. Для каждого поля определяются тип данных, обязательность и возможность использования массива. Следующий этап предназначен для задания разрешенных вызовов внешних операций. Пользователь может выбрать операции

других микросервисов, к которым разрешен доступ. Эти правила формируют ограничения взаимодействия между сервисами. На завершающем этапе выполняется проверка корректности введенных данных.

Система анализирует корректность структуры и связей между элементами. Результат проверки определяется условием корректности модели. Если обнаружены ошибки, система формирует сообщения об ошибках. Эти сообщения передаются пользователю. После получения сообщений об ошибках пользователь может исправить введенные данные. Если онтологическая модель проходит проверку, она сохраняется. Сохраненная структура используется в дальнейшем процессе генерации схемы. После этого запускается модуль генерации GraphQL-схемы. На этом этапе онтология преобразуется в формальное описание GraphQL Schema.

В результате преобразования формируются основные элементы схемы. Создаются типы данных, а также операции `Query` и `Mutation`. Одновременно формируются ограничения, которые будут применяться при обработке запросов.

На рис. 5 показан интерфейс формы этапа подтверждения создания сущности микросервиса. После заполнения формы пользователем выполняется автоматическая проверка корректности введенных данных. Если структура модели соответствует заданным ограничениям, система отображает окно подтверждения создания онтологических сущностей.

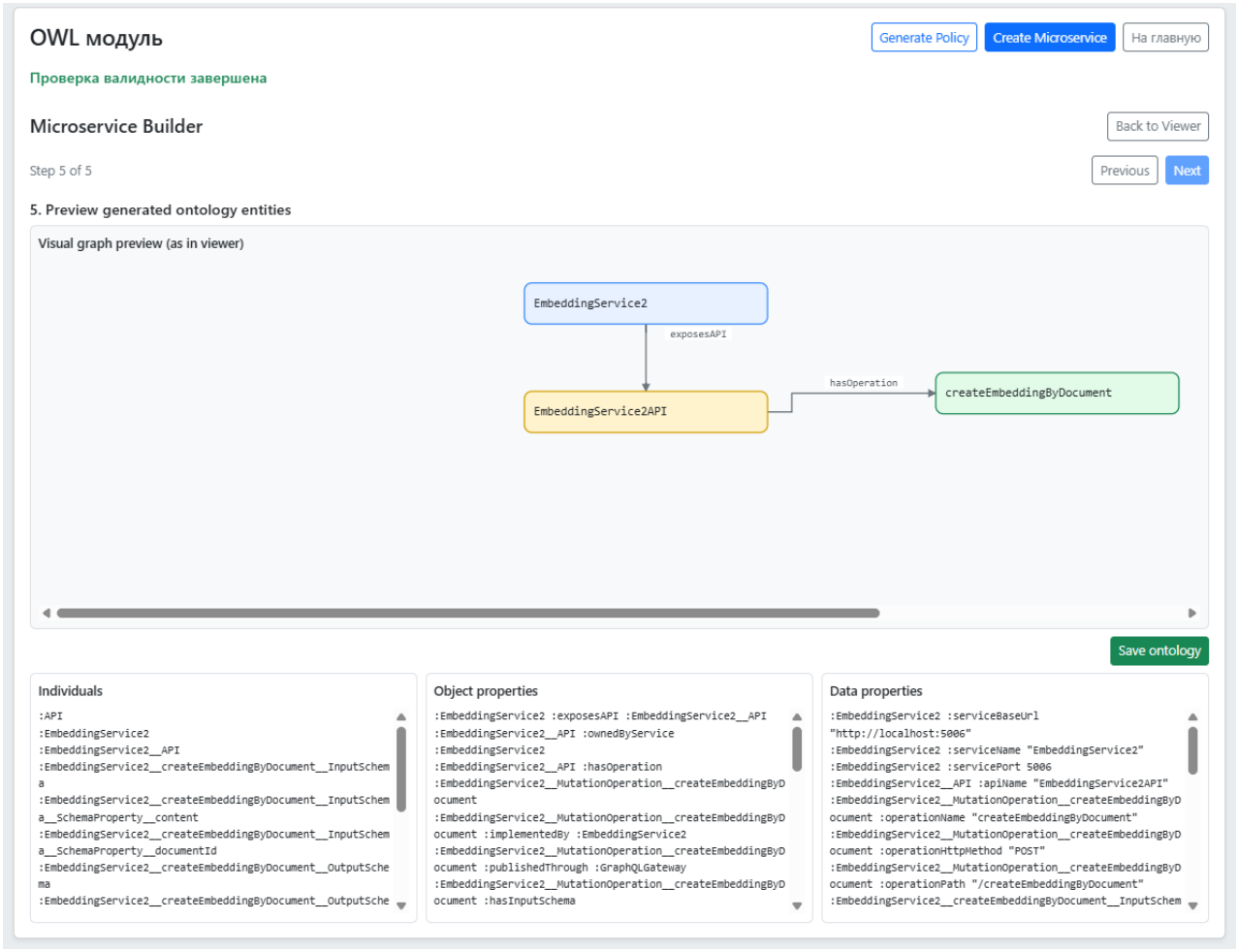

Рис. 5. Интерфейс предварительного просмотра и подтверждения созданных сущностей онтологической модели микросервиса.

В верхней части интерфейса отображается сообщение о завершении проверки корректности модели. Это означает, что структура онтологии успешно прошла валидацию. Пользователю доступен режим предварительного просмотра сформированных сущностей. Центральная часть окна содержит графическое представление онтологии. В ней визуализируются созданные элементы модели и связи между ними. В данном примере это сервис `EmbeddingService2`, его программный интерфейс `EmbeddingService2API` и операция `createEmbeddingByDocument`. Между элементами показываются отношения, которые описывают структуру сервиса и его API. Нижняя часть окна содержит текстовое представление созданных элементов онтологии, таких как индивидуумы, объектные свойства и свойства данных. Эти элементы отражают структуру

сервисов, их интерфейсов и операций, а также параметры конфигурации и описания методов.

Сформированная схема передается в GraphQL-сервер. Затем подключаются правила валидации и ограничения, полученные из онтологической модели. На завершающем этапе выполняется инициализация сервера с новой схемой. Это позволяет серверу использовать заданные правила при обработке входящих сообщений.

Механизм соблюдения ограничений

Механизм проверки входящих сообщений в GraphQL-сервере обеспечивает контроль и соблюдение заданных ограничений во время работы системы. Он анализирует структуру запроса и сопоставляет ее с описанием, полученным из онтологической модели. Процесс работы данного механизма представлен на рис. 6.

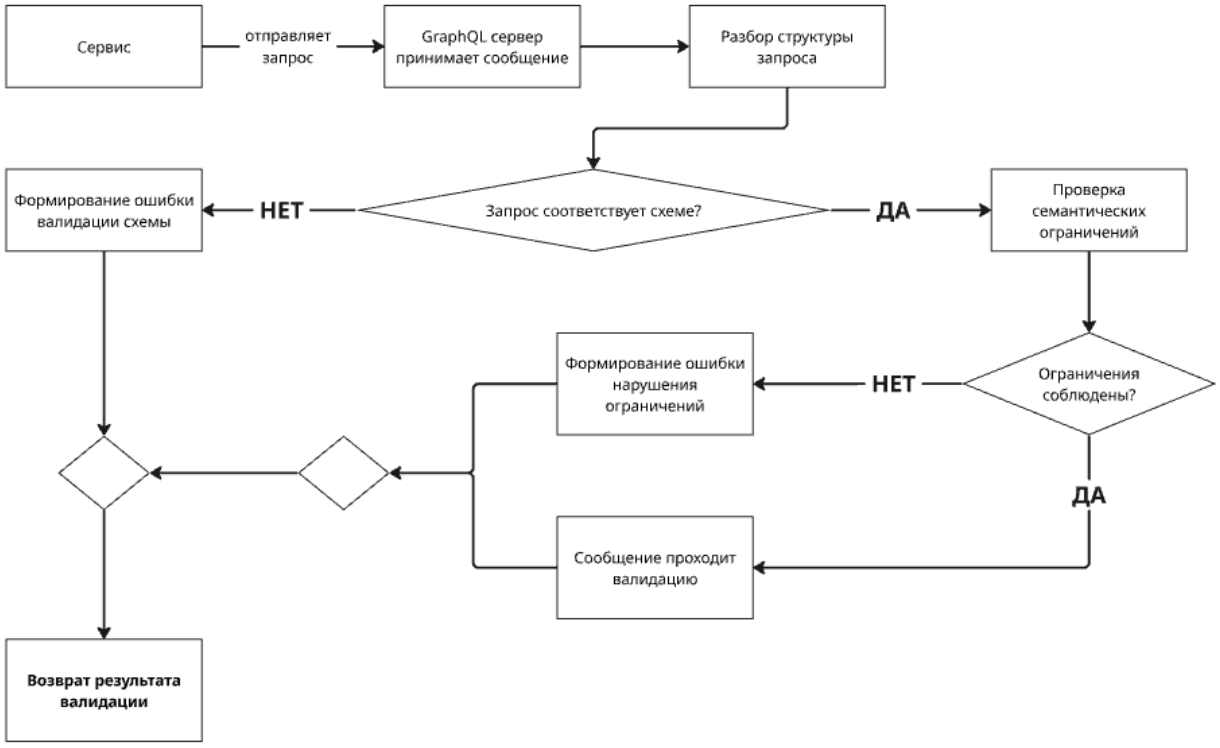

Рис. 6. Алгоритм валидации сообщений и проверки ограничений в GraphQL-сервере.

На рис. 6 представлена схема обработки запроса, поступающего в GraphQL-сервер. Процесс начинается с отправки запроса сервисом или клиентским приложением. Сообщение поступает на сервер, далее выполняется анализ структуры запроса. На этом этапе определяется соответствие запроса GraphQL-схеме, которая была ранее сгенерирована на основе онтологии. Если структура запроса не соответствует схеме, формируется сообщение об ошибке валидации. В этом случае дальнейшая обработка запроса не выполняется: сервер возвращает результат проверки с описанием ошибки. Если структура запроса корректна, выполняется следующий этап проверки. На этом этапе анализируются семантические ограничения: проверяются допустимость вызова операции и соответствие запроса правилам взаимодействия сервисов. Эти правила были определены ранее в онтологической модели.

В случае, когда все проверки выполнены успешно, сообщение считается корректным. Запрос проходит валидацию и может быть передан для дальнейшего выполнения. Проверка выполняется непосредственно в GraphQL-сервере. Это позволяет предотвращать выполнение недопустимых операций и поддерживать согласованность взаимодействия сервисов системы.

На рис. 7 приведен фрагмент кода, который демонстрирует результат преобразования экземпляров онтологии в GraphQL-схему. Этот код не отражает всю реализацию сервера. Он показывает только часть, связанную с ограничениями вызовов операций и проверкой структуры сообщений.

Названные элементы формируются автоматически на основе информации, содержащейся в онтологии. Фрагмент демонстрирует пример формирования правил разрешенных операций. В онтологической модели такие ограничения задаются через отношение разрешенных вызовов между сервисами. После конвертации эти данные преобразуются в структуру конфигурации GraphQL-сервера. В коде формируется объект `POLICY_RULES`. Он содержит описание допустимых запросов для каждого микросервиса. Для каждого сервиса указываются разрешенные операции `Query` и `Mutation`. В результате ограничения, заданные в онтологии, становятся исполняемыми правилами и учитываются при обработке запросов.

```
// From OWL ontology.
// Source: Microservice.serviceName + Microservice.permitCallOperation + operations
const POLICY_RULES = {
  "embedding-service": {
    Query: [
      "getGraphByDocument",
    ],
    Mutation: [
      "createGraphByDocument",
      "embeddingCreateByDocument",
    ],
  },
  "graph-service": {
    Query: [
      "getGraphByDocument",
      "getSummDocuments",
    ],
    Mutation: [
      "createGraphByDocument",
      "embeddingCreateByDocument",
    ],
  },
  "upload-service": {
    Query: [
      "getGraphByDocument",
    ],
    Mutation: [
      "createGraphByDocument",
      "embeddingCreateByDocument",
      "uploadDocument",
    ],
  },
};

module.exports = {
  POLICY_RULES,
};
```

Рис. 7. Фрагмент конфигурации ограничений вызова операций микросервисов, сформированный на основе онтологической модели.

На рис. 8 показан пример формирования части GraphQL-схемы. Создается тип ответа операции `createEmbeddingByDocument`, описывается структура возвращаемого сообщения. Указываются поля `documentId` и `message`, а также их типы. Далее определяется описание операции `Mutation`. В нем задаются параметры запроса и их типизация. Такой фрагмент представляет преобразование схем сообщений из онтологической модели в формальное описание GraphQL.

```
const embeddingServiceEmbeddingcreatebydocumentType = new GraphQLObjectType({
  name: "EmbeddingServiceEmbeddingcreatebydocument",
  fields: () => ({
    documentId: { type: new GraphQLNonNull(GraphQLID) },
    message: { type: new GraphQLNonNull(GraphQLString) },
  }),
});

const embeddingServiceQueryFields = {
};

const embeddingServiceMutationFields = {
  embeddingCreateByDocument: {
    type: embeddingServiceEmbeddingcreatebydocumentType,
    args: {
      content: { type: new GraphQLNonNull(GraphQLString) },
      documentId: { type: new GraphQLNonNull(GraphQLID) },
    },
```

Рис. 8. Фрагмент GraphQL схемы операции, сформированной на основе схемы сообщения из онтологии.

ЗАКЛЮЧЕНИЕ

Разработанная онтология микросервисов является единой формальной моделью архитектуры всей системы. Под онтологией микросервисов понимается формальная семантическая модель, описывающая сущности микросервисной архитектуры и их взаимосвязи. Особенностью онтологического метода проектирования является автоматическая согласованность всей цепочки взаимодействия. Поскольку онтология содержит не только сущности, но и связи между ними, после загрузки в графовую базу можно выполнять сложные запросы, например поиск всех сервисов, зависящих от какого-либо события [11]. Микросервисы, операции, сообщения и схемы данных становятся вершинами графа, а их взаимодействие и зависимости – ребрами. Графовое представление [12] позволяет организовать извлечение информации и генерацию модели взаимодействия, ориентированные на архитектуру системы. В дальнейщем возможна разработка интеллектуального ассистента [13]. Он будет способен отвечать на вопросы о структуре системы, доступных операциях, форматах сообщений. Это существенно снизит порог вхождения для новых разработчиков системы. Предлагаемый подход снижает время изучения документации и устраняет необходимость ручного анализа API и схем сообщений. Это особенно важно для крупных микросервисных систем, где количество сервисов и интеграций может доходить до нескольких десятков, а то и сотен.

СПИСОК ЛИТЕРАТУРЫ

1. *Oumoussa I., Saidi R.* The ontology-based mapping of microservice identification approaches: a systematic study of migration strategies from monolithic to microservice architectures // Computers. 2025. Vol. 14, No. 4. P. 133.

2. Шитько А.М., Пацей Н.В. Интеграция микросервисов на основе RPC // Информационные технологии: тезисы докладов 82-й научно-технической конференции профессорско-преподавательского состава, научных сотрудников и аспирантов (с международным участием). Минск: Белорусский государственный технологический университет, 2018. С. 31–32.

3. *Балес А.И.* Унифицированная модель данных и ее применение в микросервисной архитектуре // Современные информационные технологии и ИТ-образование. 2020. Т. 16, № 2. С. 416–425.
https://doi.org/10.25559/SITITO.16.202002.416-425

4. *Anderson C. et al*. An ontology-based reasoning framework for context-aware applications // Proceedings of the International and Interdisciplinary Conference on Modeling and Using Context. Cham: Springer International Publishing, 2015. P. 471–476.

5. *Гусенков А.М., Бухараев Н.Р., Биряльцев Е.В.* Построение онтологии предметной области на основе логической модели данных // Электронные библиотеки. 2020. Т. 23, № 3. С. 390–417.
https://doi.org/10.26907/1562-5419-2020-23-3-390-417

6. *Карев А.Н., Федосин С.А.* Онтологический подход к интеграции информационных систем // Перспективы науки. 2023. Т. 168, № 9. С. 26–29.

7. *Чернов П.К., Рабчевский Е.А.* Создание интегрированной модели данных из разнородных источников, содержащих цифровые следы // Вестник ПГУ. Математика. Механика. Информатика. 2022. С. 81–87.
https://doi.org/10.17072/1993-0550-2022-2-81-87

8. *Ketul Kishorbhai Dusane* Cloud Messaging Systems Architecture and Implementation // Journal of Computer Science and Technology Studies. 2025. Vol. 7, No. 8. P. 739–746. https://doi.org/10.32996/jcsts.2025.7.8.86

9. Huanyu Li, Olaf Hartig, Rickard Armiento, Patrick Lambrix. Ontology-Based GraphQL Server Generation for Data Access and Data Integration // Semantic Web. 2024. Vol. 15, No. 5. P. 1639–1675.

10. *Ломов П.А., Малоземова М.Л.* Обучение и применение нейросетевой языковой модели для пополнения онтологии // Труды Кольского научного центра РАН. 2020. № 8–11. С. 38–45.

11. *Зимнуров М.Ф., Астраханцева И.А.* Методология создания многосвязных структур данных с применением LLM в рабочих проектах // Современные наукоемкие технологии. Региональное приложение. 2025. № 1. С. 76–83.

12. *Папуша С.И.* Онтология и графовые базы данных // Проблемы экономики и юридической практики. 2020. Т. 16, № 3. С. 268–272.

13. *Ломов П.А., Шишаев М.Г., Диковицкий В.В.* Преобразование OWL-онтологий для визуализации и использования в качестве основы пользовательского интерфейса // Онтология проектирования. 2012. № 3. С. 49–61.

AN ONTOLOGICAL APPROACH TO DESIGNING A MICROSERVICE ARCHITECTURE

E. A. Malykh[1] [0009-0008-0730-1603], A. A. Bloshchuk[2] [0000-0001-6683-5973], O. M. Ataeva[3] [0000-0003-0367-5575]

[1–3]*Moscow Witte University, Moscow, Russia*

[3]*Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Moscow, Russia*

[1]warior227@yandex.ru, [2]abloshuk@muiv.ru, [3]oataeva@frccsc.ru

Abstract

Despite the widespread use of microservice architecture in the development of software systems, there is no formalized approach that ensures consistent and guaranteed interaction of microservices at the level of transmitted data, which leads to integration errors and complicates the maintenance of distributed systems. The purpose of the study is to develop an approach to the organization of microservices interaction based on ontological modeling, providing formalization of data structures and automated validation of messages. The paper presents a method for converting formal descriptions of data schemas into ontological models based on the GraphQL schema specification. This method allows you to automate the data validation process

and reduce the number of integration errors. An ontological model has been developed that provides an analysis of dependencies between microservices and a mechanism for validating message contracts.

The practical significance of the work lies in achieving a consistent description of microservices, operations, and message formats as a result of using an ontological approach. The representation of the ontology in the form of a graph makes it possible to analyze the dependencies between microservices and simplifies the maintenance of large distributed systems.

Keywords*: ontology, GraphQL schema, data integration, microservice architecture, message flows, data validation, service interoperability, ontological model, data consistency, schema management, data bus.*

REFERENCES

1. *Oumoussa I., Saidi R.* The ontology-based mapping of microservice identification approaches: a systematic study of migration strategies from monolithic to microservice architectures // Computers. 2025. Vol. 14, No. 4. P. 133.

2. *Shitko A.M., Patsei N.V.* Integration of microservices based on RPC // Information Technologies: Proceedings of the 82nd Scientific and Technical Conference of Academic Staff, Researchers, and Postgraduate Students (with international participation). Minsk: Belarusian State Technological University, 2018. P. 31–32.

3. *Bales A.I.* A unified data model and its application in microservice architecture // Modern Information Technologies and IT Education. 2020. Vol. 16, No. 2. P. 416–425. https://doi.org/10.25559/SITITO.16.202002.416-425

4. *Anderson C. et al.* An ontology-based reasoning framework for context-aware applications // Proceedings of the International and Interdisciplinary Conference on Modeling and Using Context. Cham: Springer International Publishing, 2015. P. 471–476.

5. *Guseenkov A.M., Bukharaev N.R., Biryaltsev E.V.* Construction of a domain ontology based on a logical data model // Russian Digital Library. 2020. Vol. 23, No. 3. P. 390–417. https://doi.org/10.26907/1562-5419-2020-23-3-390-417

6. *Karev A.N., Fedosin S.A.* An ontological approach to the integration of information systems // Science Prospects. 2023. Vol. 168, No. 9. P. 26–29.

7. *Chernov P.K., Rabchevsky E.A.* Creation of an integrated data model from

heterogeneous sources containing digital traces // Bulletin of PSU. Mathematics. Mechanics. Informatics. 2022. P. 81–87.
https://doi.org/10.17072/1993-0550-2022-2-81-87

8. *Dusane K.K.* Cloud messaging systems architecture and implementation // Journal of Computer Science and Technology Studies. 2025. Vol. 7, No. 8. P. 739–746. https://doi.org/10.32996/jcsts.2025.7.8.86

9. *Li H., Hartig O., Armiento R., Lambrix P.* Ontology-based GraphQL server generation for data access and data integration // Semantic Web. 2024. Vol. 15, No. 5. P. 1639–1675.

10. *Lomov P.A., Malozemova M.L.* Training and application of neural network language models for ontology population // Proceedings of the Kola Science Center of the Russian Academy of Sciences. 2020. No. 8–11. P. 38–45.

11. *Zimnurov M.F., Astrakhantseva I.A.* Methodology for creating multi-connected data structures using LLMs in practical projects // Modern High Technologies. Regional Application. 2025. No. 1. P. 76–83.

12. *Papusha S.I.* Ontologies and graph databases // Problems of Economics and Legal Practice. 2020. Vol. 16, No. 3. P. 268–272.

13. *Lomov P A., Shishaev M.G., Dikovitsky V.V.* Transformation of OWL ontologies for visualization and use as a basis for user interfaces // Ontology of Designing. 2012. No. 3. P. 49–61.

СВЕДЕНИЯ ОБ АВТОРАХ

МАЛЫХ Евгений Александрович – аспирант Московского университета имени С. Ю. Витте по направлению «Системный анализ, управление и обработка информации, статистика». Научные интересы: автоматизация бизнес-процессов, интеграция источников данных в разнородных информационных системах.

Evgeniy Aleksandrovich MALYKH – PhD student at Moscow Witte University, specializing in System Analysis, Management, and Information Processing, Statistics. Research interests: business process automation, integration of data sources in heterogeneous information systems.

email: warior227@yandex.ru;

ORCID: 0009-0008-0730-1603

БЛОЩУК Андрей Алексеевич – кандидат технических наук, доцент кафедры информационных систем, Московский университет имени С. Ю. Витте. В 2004 году присвоена степень кандидата технических наук в 4-м ВНИИ (г. Юбилейный, МО). Более десяти лет преподавания дисциплин ИТ-направленности. Издан ряд научных статей по инновационным методам автоматизации различных ас пектов деятельности образовательной организации. Сфера научных интересов: автоматизация процессов образовательной деятельности. Применение информационных технологий в образовании.

Andrey Alekseevich BLOSHCHUK – Candidate of Engineering Sciences (PhD equivalent), associate professor at the Department of Information Systems, Moscow Witte University. He was awarded his Candidate of Engineering Sciences degree in 2004 from the 4th Central Research Institute (4th VNII) in Yubileyny, Moscow Oblast. With over a decade of experience teaching IT-related disciplines, he has authored numerous scientific articles on innovative methods for automating various aspects of educational organization activities. His research interests include the automation of educational processes and the application of information technologies in education.

email: abloshuk@muiv.ru

ORCID: 0000-0001-6683-5973

АТАЕВА Ольга Муратовна – старший научный сотрудник Вычислительного центра им. А.А. Дородницына ФИЦ ИУ РАН, к. т. н. Область научных интересов: системное программирование, базы данных, и нженерия знаний и онтологии.

Olga Muratovna ATAEVA – senior researcher at the Dorodnicyn Computing Center, Federal Research Center “Computer Science and Control” of the Russian Academy of Sciences; Candidate of Technical Sciences (PhD). Research interests: systems programming, databases, knowledge engineering, ontologies. Number of scientific publications – 80.

email: oataeva@frccsc.ru

ORCID: 0000-0003-0367-5575

Материал поступил в редакцию 12 апреля 2026 года