Санкт-Петербургский государственный университет

Слугин Александр Николаевич

Выпускная квалификационная работа

Разработка системы для сбора данных контекста бизнес-процессов

Уровень образования: бакалавриат

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

Основная образовательная программа *СВ.5162.2021 «Технологии программирования»*

Научный руководитель:
старший преподаватель, к. т. н., Литвинов Ю. В.

Консультант:
начальник центра, АО «Нэксайн», Хусаинов А. М.

Рецензент:
ведущий инженер, АО «Нэксайн», Вилков М. Д.

Санкт-Петербург
2025

Saint Petersburg State University

Aleksandr Slugin

Bachelor's Thesis

Development of a system for collecting business process context data

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5162.2021 "Programming Technologies"*

Scientific supervisor:
C.Sc., senior lecturer, Y.V. Litvinov

Consultant:
Head of the center at JSC Nexign, A.M. Khusainov

Reviewer:
Senior Engineer at JSC Nexign, M.D. Vilkov

Saint Petersburg
2025

Оглавление

Введение

В любых компаниях существует множество бизнес-процессов, которые обеспечивают их внутреннюю деятельность, взаимодействие с клиентами и партнёрами. Для моделирования и автоматизации процессов используются **BPM-системы**[1] — системы управления бизнес-процессами [17]. При использовании таких систем в ходе выполнения каждого процесса формируется **контекст бизнес-процесса**, в который собирается вся информация, полученная от его участников. Сбор данных от участников процесса может быть реализован через разные каналы коммуникации, например, операторы связи могут использовать для взаимодействия с клиентами SMS-нотификации[2], USSD-команды[3], электронную почту, а в салонах обслуживания и контактных центрах — пользовательские интерфейсы для администраторов.

Несмотря на широкие возможности BPM-систем, сбор данных требует отдельной реализации для каждого канала коммуникации. Это приводит к созданию множества адаптеров — сервисов, направленных на сбор данных через определённый канал. При таком подходе необходимо поддерживать несколько адаптеров, а добавление нового канала требует не только разработки нового адаптера, но и внесение изменений в BPM-систему для возможности интеграции.

Для реализации взаимодействия между BPM-системой и адаптерами можно использовать различные системы интеграции приложений, что позволит BPM-системе работать только с интеграционной системой, исключив прямое взаимодействие с каждым адаптером. Применение такого подхода уменьшит количество необходимых изменений в BPM-системе при добавлении нового канала коммуникации. Но несмотря на большое количество разработанных систем интеграции, каждая из них специфична для своей области [7] и привязана к конкретным моделям исполнения, в связи с чем имеет значительные ограничения в своей функциональности [5].

[1] BPM — Business Process Management

[2] SMS — Short Message Service

[3] USSD — Unstructured Supplementary Service Data

Таким образом, существующие системы интеграции не предоставляют гибких настроек сбора данных под конкретные бизнес-процессы, следовательно их использование не избавляет от проблемы сопровождения множества адаптеров. В связи с этим компанией Nexign была поставлена задача разработки системы, которая обеспечит централизованное управление сбором данных для бизнес-процессов через различные каналы коммуникации. Предлагаемое решение позволит перенести логику сбора данных из адаптеров в разрабатываемую систему, тем самым упростив адаптеры и сократив затраты на их разработку и сопровождение. Также данный подход даст возможность изменять настройки сбора данных без модификаций в BPM-системе, что сделает возможным внедрять данную систему вместе с различными существующими BPM-системами.

1. Постановка задачи

Целью работы является разработка системы для сбора данных контекста бизнес-процессов для обеспечения масштабируемости по каналам коммуникации и простой настройки сбора данных. Для её выполнения были поставлены следующие задачи.

1. Провести обзор существующих решений для интеграции приложений.
2. Провести обзор существующих BPM-систем для выявления способов интеграции с ними.
3. Разработать архитектуру системы с учетом масштабируемости по каналам коммуникаций и возможности интеграций с различными BPM-системами.
4. Реализовать систему для сбора данных и адаптеры к каналам коммуникации.
5. Внедрить систему вместе с существующей BPM-системой.
6. Провести апробацию системы.

2. Обзор

2.1. Способы интеграции приложений

В данном разделе будет проведен обзор существующих подходов к интеграции нескольких приложений. Такие подходы могут быть полезны для интеграции BPM-системы и множества адаптеров к разным каналам коммуникации.

Интеграция корпоративных приложений — Enterprise application integration (EAI) — это область исследований, которая предлагает методологии и инструменты для разработки интеграционных решений, направленных на организацию работы множества приложений для повышения совместимости их данных или разработки новой функциональности в дополнение к существующей [16].

За всё время было создано большое количество интеграционных систем, которые можно разделить по их архитектурным подходам на четыре типа в зависимости от используемой топологии соединений (рис. 1) [10]:

1. Топология Point-to-Point или «точка-точка».
2. Топология Hub-and-Spoke или «хаб и спицы».
3. Enterprise Message Bus.
4. Enterprise Service Bus.

Дальше будут рассмотрены все типы топологий соединений, их преимущества и недостатки.

2.1.1. Топология Point-to-Point

Тип соединения приложений Point-to-Point или «точка-точка» является традиционным способом интеграции. При использовании данного подхода приложения напрямую взаимодействуют друг с другом, что позволяет обойтись без дополнительного программного обеспечения [19]. Но у такого способа интеграции есть недостатки, первый из

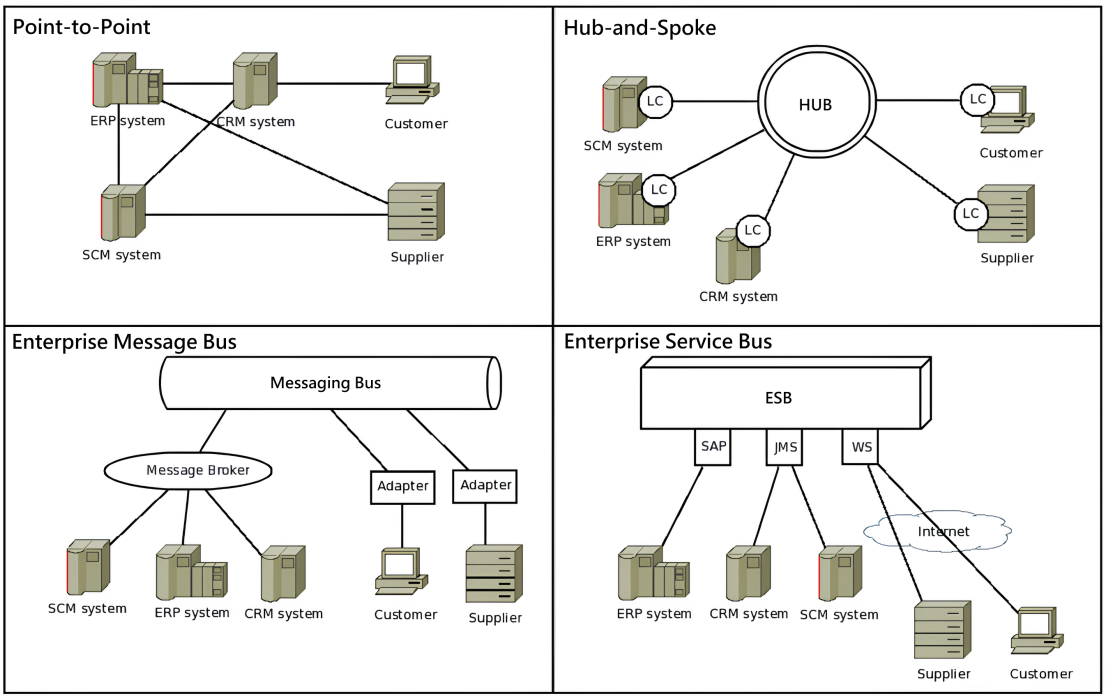

Рис. 1: Различные типы EAI [10]

которых заключается в создании большого количества связей между приложениями, что приводит к увеличению стоимости сопровождения системы. Также при таком типе соединения отсутствует централизованный способ управления интеграционной логикой. Таким образом, Point-to-Point соединение может быть использовано только при небольшом количестве интегрируемых приложений, когда создание и развертывание системы с индивидуальными настройками интеграции будет стоить дешевле, чем использование интеграционных решений других типов [14]. Пример такого соединения представлен на рисунке 1 в левом верхнем углу.

2.1.2. Топология Hub-and-Spoke

В топологии Hub-and-Spoke или «хаб и спицы» каждое приложение взаимодействует только с центральным узлом при помощи простых коннекторов. Центральный узел отвечает за управление соединениями и передачу данных между приложениями, благодаря чему можно легко управлять настройками интеграции. Главным преимуществом такого типа соединения является небольшое количество соединений для каж-

дого приложения, даже если приложению необходимо взаимодействовать со множеством других систем. Пример такого соединения представлен на рисунке 1 в правом верхнем углу.

Но в то же время подход с единственным центральным узлом имеет ограничения по масштабируемости при большом количестве передаваемых сообщений. В связи с этим была предложена архитектура Federated Hub and Spoke, которая заключается в наличии нескольких хабов, каждый из которых может иметь локальные и глобальные настройки [6]. Такой подход решает проблему масштабируемости, при этом сохраняя возможность централизованного управления интеграцией приложений.

Недостатком соединения Hub-and-Spoke также является отсутствие стандартных интерфейсов для интеграции: для каждого приложения необходимо разрабатывать коннектор для взаимодействия приложения и центрального узла. Это приводит к разработке дорогих индивидуальных решений, которые работают только в определенной среде [14].

2.1.3. Enterprise Message Bus

Для решения проблем топологии Hub-and-Spoke была предложена схема интеграции Enterprise Message Bus, основанная на передаче сообщений (MOM — Messaging-oriented Middleware) [10]. Она так же имеет центральный узел — шину сообщений, которая отвечает только за передачу и хранение сообщений и предоставляет единый интерфейс для подключения адаптеров к шине. Это позволяет распределить логику преобразования и маршрутизации сообщений по адаптерам, которые работают на той же платформе, что и приложения, благодаря чему улучшается масштабируемость системы [6]. Но такой подход усложняет управление интеграцией приложений из-за распределённости логики по адаптерам. Пример такого соединения представлен на рисунке 1 в левом нижнем углу.

2.1.4. Enterprise Service Bus

Развитием технологии Enterprise Message Bus стало создание топологии Enterprise Service Bus (ESB), широко используемой в интеграции корпоративных приложений. ESB представляет собой набор сервисов промежуточного программного обеспечения (middleware), которые предоставляют возможности интеграции [10]. При использовании такого подхода логика интеграции размещена внутри ESB-системы, а взаимодействие с приложениями происходит через заранее определенные коннекторы, которые входят в состав ESB-решения. Это позволяет сохранить преимущества как топологии Hub-and-Spoke, так и Enterprise Message Bus, а именно централизованное управление логикой интеграции и предоставление стандартизированных интерфейсов для взаимодействия с приложениями. Пример такого соединения представлен на рисунке 1 в правом нижнем углу.

2.1.5. IPaaS

В современном мире всё больше компаний используют облачные приложения, что приводит к задаче их интеграции. Для ее решения были созданы Integration Platform as a Service (IPaaS) — облачные платформы для интеграции приложений. IPaaS-системы включают в себя как функциональные возможности EAI-систем, так и преимущества SaaS-приложений[4], такие как высокая производительность и предсказуемые расходы [4].

При использовании облачных платформ для интеграции возникает риск нарушения безопасности и конфиденциальности данных из-за их передачи через интернет-соединение. Это усугубляется тем, что данные передаются не только поставщику платформы, но и сторонним компаниям, обеспечивающим облачную инфраструктуру. Кроме того, облачные платформы интеграции у каждого поставщика имеют свою реализацию и интерфейсы, что формирует высокую степень зависимости от конкретного поставщика IPaaS [14].

[4]SaaS — Software as a Service

2.1.6. Выводы

Далее будут проанализированы возможности применения существующих решений, основанных на рассмотренных способах интеграции, для реализации сбора данных контекста бизнес-процессов через различные каналы коммуникации.

Системы интеграции, использующие топологию Point-to-Point соединения приложений, не могут быть использованы, поскольку данная работа направлена на решение проблемы, связанной с применением подобного типа соединений между BPM-системами и адаптерами для каналов коммуникации.

Существующие системы интеграции, основанные на топологии Hub-and-Spoke, часто создаются под специфические требования, что ограничивает их возможности для расширения функциональности и приводит к зависимости от поставщика [10]. Поэтому использование таких систем приведёт к увеличенным затратам не только на сопровождение системы интеграции, но и при добавлении нового канала коммуникации.

Интеграционные решения, реализующие топологию соединений Enterprise Message Bus, используют стандартизированные протоколы для взаимодействия с приложениями. Например, брокер сообщений RabbitMQ может работать с такими протоколами как AMQP, MQTT, XMPP и STOMP, а брокер сообщений Kafka определяет свой собственный интерфейс для обмена данными. Фиксированный интерфейс позволит ускорить разработку адаптеров к каналам коммуникации и BPM-системам, исключая необходимость проектировать новый способ взаимодействия с интеграционной системой при добавлении нового канала или BPM-системы. Но необходимо отметить, что при использовании таких систем бизнес-логика по сбору данных будет распределена по адаптерам к BPM-системам и каналам коммуникации, что существенно затруднит управление сбором данных, разработку и сопровождение такого решения. Поэтому использование систем, основанных на топологии Enterprise Message Bus, нецелесообразно.

Системы интеграции, использующие топологию Enterprise Service

Bus, могут обеспечить простое добавление нового канала коммуникации благодаря стандартизированным интерфейсам и централизованное управление логикой интеграции. Но одним из требований к решению для сбора данных контекста бизнес-процессов является гибкая настройка параметров сбора данных, при этом логика сбора данных может быть нетривиальной и требующей хранения состояния взаимодействия с пользователем. Существующие ESB-системы не предоставляют таких возможностей, так как в первую очередь ориентированы на передачу данных между приложениями. Возможным решением может быть модификация ESB-системы с открытым исходным кодом, однако это потребует глубокого понимания внутренней архитектуры системы и значительных ресурсов на разработку и сопровождение. В связи с этим использование существующей ESB-системы неоправданно для решения задачи сбора данных контекста бизнес-процессов.

Для минимизации рисков, связанных с безопасностью и конфиденциальностью данных, применение IPaaS-решений исключается при реализации сбора данных контекста бизнес-процессов.

Таким образом, существующие системы интеграции приложений не предоставляют полноценных возможностей для реализации сбора данных контекста бизнес-процессов, поэтому необходимо разработать собственную систему, которая позволит гибко настраивать параметры сбора данных и минимизирует трудозатраты при добавлении нового канала коммуникации с пользователем.

2.2. Обзор способов интеграции с BPM-системами

Для интеграции разрабатываемой системы сбора данных контекста бизнес-процессов необходимо учитывать механизмы взаимодействия, предлагаемые BPM-системами. В данном разделе будут рассмотрены одни из самых популярных в России BPM-систем [20], такие как Camunda и Elma, а также CPM — Customer Problem Management — BPM-система компании Nexign. Основными задачами разрабатываемой системы при взаимодействии с BPM-системами являются получение ин-

формации о бизнес-процессе и его контексте, а также передача собранных данных от клиента обратно в систему. Поэтому будут проанализированы доступные в указанных BPM-системах методы интеграции, обеспечивающие выполнение этих задач, при этом не требующих модификаций в BPM-системе.

2.2.1. Camunda

Camunda — это платформа для моделирования и автоматизации бизнес-процессов с открытым исходным кодом [2]. Контекст бизнес-процесса в Camunda представлен набором переменных, доступных на протяжении всего жизненного цикла процесса. Для получения и обновления этих данных можно использовать следующие механизмы:

1. REST API

   Camunda предоставляет набор методов для управления процессами и их переменными. Система позволяет загружать данные в контекст процесса через HTTP-запросы, что делает возможным интеграцию без изменения бизнес-логики процесса.

2. Kafka Connector [3]

   Если при использовании Camunda настроена интеграция с другими системами через событийную архитектуру, например, с помощью Kafka Connector, возможно использовать её для интеграции системы сбора данных контекста.

3. External Tasks [9]

   Camunda поддерживает концепцию внешних задач, которые могут быть выполнены внешними системами. Такой подход может быть использован для сбора данных через разрабатываемую систему. Недостатком данного способа является необходимость доработки конфигурации процесса в Camunda.

2.2.2. Elma

Elma — это low-code BPM-платформа, ориентированная на автоматизацию бизнес-процессов и документооборота [22]. Для работы с контекстом бизнес-процессов могут быть использованы следующие механизмы:

1. REST и SOAP API

   Elma предоставляет REST и SOAP API для взаимодействия с внешними системами. Через данные интерфейсы можно получать и обновлять данные о контексте бизнес-процесса. Данный подход не требует модификации существующих процессов.

2. Коннекторы к базам данных

   Elma имеет готовые коннекторы к базам данных MS SQL, Oracle, PostgreSQL, что может быть использовано в случае локально развёрнутой BPM-системы (On-Premise). Однако при использовании такого способа возникает риск нарушения консистентности данных, а также требуется модификация бизнес-процессов в Elma для возможности интеграции.

2.2.3. CPM

CPM — Customer Problem Management — продукт компании Nexign, предназначенный для управления бизнес-процессами организации. Стоить отметить, что CPM ориентирован на обработку обращений клиентов, что отражено в его названии. Для работы с контекстом бизнес-процессов могут быть использованы следующие механизмы:

1. REST API

   Продукт может интегрироваться с другими системами и приложениями с помощью API по стандарту OpenAPI. С помощью данного интерфейса имеется возможность получать и изменять данные контекста бизнес-процессов.

2. AMQP

   Продукт поддерживает отправку AMQP-уведомлений внешним системам о событиях, связанных с обработкой бизнес-процесса. Интерфейс доступен для неопределенного круга внешних систем – подписчиков сообщений, что позволяет получать информацию о процессе, не требуя модификаций в BPM-системе.

3. Решение

3.1. Требования

Перед разработкой системы для сбора данных контекста бизнес-процессов были выделены функциональные требования, которым должна удовлетворять система.

1. Возможность конфигурирования параметров сбора данных для каждого бизнес-процесса.
2. Возможность сбора данных через разные каналы коммуникаций и добавления нового канала коммуникации с минимальными трудозатратами.
3. Возможность интеграции с различными BPM-системами.
4. При взаимодействии с клиентом через чат-боты или мессенджеры необходимо обеспечить диалог, направленный на сбор данных.

3.2. Архитектура

В данном разделе будут описаны архитектурные решения, которые были приняты для разработки системы сбора данных контекста бизнес-процессов.

В связи с тем, что необходимо обеспечить гибкую настройку параметров сбора данных для каждого бизнес-процесса, становится ясно, что необходим центральный узел, который будет отвечать за управление процессом сбора данных, а также за интеграцию BPM-систем с адаптерами к каналам коммуникации.

Чтобы минимизировать затраты при добавлении новых каналов, адаптеры к ним должны оставаться максимально простыми в реализации. Этому способствует унифицированный интерфейс взаимодействия с адаптерами: с одной стороны, система для сбора данных предоставляет интерфейс, с помощью которого адаптеры будут передавать полученные от пользователя данные в BPM-систему, а с другой — использует

стандартизированный интерфейс для получения данных от пользователя, который реализуют все адаптеры.

При интеграции с различными BPM-системами необходимо обеспечить простоту реализации коннекторов к ним. Однако из-за разнообразия способов интеграции с BPM-системами и различия в форматах данных требуется преобразование данных, полученных из BPM-системы, в доменные классы разрабатываемой системы. Для этого коннектору необходимо иметь доступ к доменной модели системы сбора данных. В связи с этим было принято решение разрабатывать для каждой BPM-системы собственный коннектор, который будет частью интеграционной системы.

Для организации взаимодействия с пользователем планируется использовать различные мессенджеры, следовательно при построении диалога необходимо поддерживать асинхронную коммуникацию, так как пользователь может отвечать с существенной задержкой. Поэтому система должна хранить текущее состояние сбора данных для каждого бизнес-процесса и пользователя. Это позволит избежать повторных запросов информации у пользователя. Для реализации такой функциональности было решено использовать базу данных, в которой будут храниться сущности-задачи, каждая из которых представляет собой задачу для сбора одного атрибута. Такие задачи создаются при получении данных из BPM-системы, а затем обрабатываются последовательно в рамках каждого бизнес-процесса.

3.2.1. Модель компонентов системы

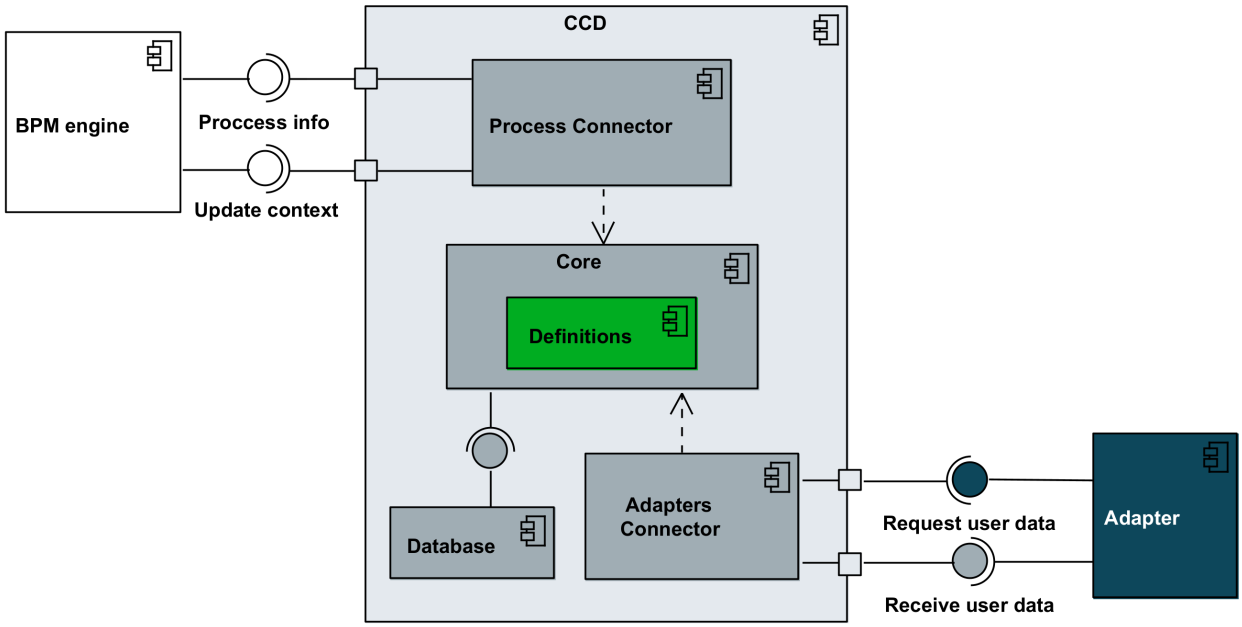

Рис. 2: Диаграмма компонентов системы для сбора данных контекста бизнес-процессов

На рисунке 2 представлена диаграмма компонентов системы для сбора данных контекста бизнес-процессов. Разрабатываемая система получила название CCD — Collecting of Context Data.

Внутренний компонент «Process Connector» разрабатывается для каждой BPM-системы и отвечает за взаимодействие с ней: получение информации о бизнес-процессе и его контексте (интерфейс «Process info»), преобразование данных в объекты классов доменной модели CCD и наоборот, а также обновление контекста бизнес-процесса для передачи полученных от пользователя данных (интерфейс «Update context»).

Внутренний компонент «Core» содержит основную бизнес-логику системы, которая легко модифицируется для каждого обрабатываемого бизнес-процесса при помощи конфигурационных файлов в формате JSON. За работу с конфигурацией отвечает компонент «Definitions». Также компонент «Core» обеспечивает работу с базой данных для хранения состояний сбора данных для каждого бизнес-процесса и предоставляет интерфейсы, которые реализуют компоненты «Process

Connector» и «Adapters Connector». Это позволяет без изменений компонента «Core» подменять реализации других компонентов.

Компонент «Adapters Connector» отвечает за взаимодействие с адаптерами к каналам коммуникации. Для запроса данных от пользователя используется стандартизированный интерфейс («Request user info»), который должны реализовывать все адаптеры. Для передачи полученных данных предоставляется единый интерфейс для адаптеров. Также данный компонент содержит логику по формированию текста сообщения и проверки корректности введённых пользователем данных.

3.3. Выбор технологий

После разработки концептуального решения были выбраны технологии, с помощью которых будет реализована система для сбора данных контекста бизнес-процессов.

С учетом реализации системы в компании Nexign, в которой большинство систем реализуются на языках программирования Java и Kotlin, а также имеется набор внутренних библиотек, написанных на Java, были рассмотрены JVM-совместимые языки. Это позволило бы переиспользовать существующие библиотеки. Кроме того, в компании имеется большое количество разработчиков, обладающих экспертизой в Java, что упрощает сопровождение и дальнейшее развитие системы на этом языке. В связи с этим в качестве основного языка программирования была выбрана Java.

В качестве дополнительных инструментов для разработки были использованы инструменты с открытым исходным кодом: фреймворк Spring [15], обеспечивающий построение модульной и расширяемой архитектуры, и система сборки Apache Maven [1], позволяющая управлять зависимостями и собирать исполняемые файлы приложения.

Для хранения данных о состояниях сбора атрибутов была выбрана реляционная база данных PostgreSQL [13], поскольку она является корпоративным стандартом в компании Nexign и используется в большинстве продуктов, включая BPM-систему CPM, а также лежит в основе

собственной СУБД, разработанной внутри компании. Для управления миграциями схемы базы данных используется инструмент с открытым исходным кодом Liquibase [11], позволяющий версионировать изменения и автоматически применять их при развёртывании.

В качестве возможных решений для взаимодействия с базой данных были рассмотрены ORM-системы Hibernate [8] и MyBatis [12]. Hibernate предоставляет высокоуровневую абстракцию над базой данных, что упрощает работу с объектной моделью, но при этом скрывает реализацию SQL-запросов. Это может усложнять контроль над производительностью выполнения запросов и отладку сложных запросов. Поэтому для взаимодействия с базой данных был выбран фреймворк MyBatis, который дает возможность явно управлять SQL-запросами и контролировать преобразование данных в объекты Java.

4. Особенности реализации

4.1. Концепция задач для сбора данных

Для снижения количества запросов в BPM-систему и для обеспечения асинхронной коммуникации с пользователем необходимо хранить состояние сбора данных для каждого бизнес-процесса и пользователя. В связи с этим была введена специальная сущность — задача, которая содержит информацию о сборе конкретного атрибута. На рисунке 3 представлена диаграмма классов, описывающая модель задачи.

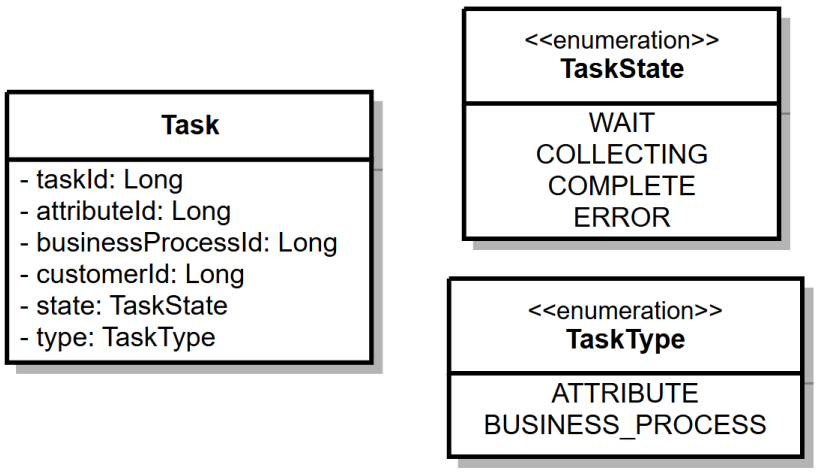

Рис. 3: Диаграмма классов, описывающая модель задачи для сбора данных

Задача может иметь один из двух типов, определяемых с помощью перечисления «TaskType». Тип задачи указывает, какую роль она выполняет в процессе сбора данных:

- ATTRIBUTE — задача для сбора одного конкретного атрибута от пользователя.
- BUSINESS_PROCESS — задача, предназначенная для агрегирования нескольких задач типа «ATTRIBUTE» в рамках одного бизнес-процесса и пользователя. Служит финальной задачей по сбору данных конкретного бизнес-процесса и выполняется только после всех задач в данном бизнес-процессе с типом «ATTRIBUTE».

В процессе выполнения задачи меняется её состояние, определяемое перечислением «TaskState»:

- WAIT — начальное состояние задачи. Данное состояние используется при создании задачи и означает, что задача еще ни разу не была взята в обработку.
- COLLECTING — состояние задачи, при котором данные запрошены у пользователя, но ответ ещё не получен.
- COMPLETE — задача успешно выполнена, необходимые данные собраны и сохранены.
- ERROR — при выполнении задачи произошла ошибка. Например, причиной ошибки может быть недоступность адаптера к каналу коммуникации.

Стоит отметить, что одновременно для каждого пользователя в статусе «COLLECTING» может быть не более одной задачи. А при выборе задач для выполнения в первую очередь отбираются задачи тех бизнес-процессов, в которых уже начат сбор данных — то есть присутствуют задачи для сбора атрибутов как в статусе «COMPLETE», так и в статусе «WAIT». Такие ограничения позволяют обеспечить пошаговое заполнение данных: сначала пользователь последовательно вводит данные для одного бизнес-процесса, и только после завершения сбора всех необходимых атрибутов для него переходит к заполнению данных для других бизнес-процессов.

Создание задач происходит при обработке информации, полученной из BPM-системы. Для каждого атрибута, который нужно собрать, создается отдельная задача, содержащая идентификаторы пользователя, бизнес-процесса и атрибута. После создания задач для всех атрибутов внутри одного бизнес-процесса, создается задача с типом «BUSINESS_PROCESS». Она будет выполнена только после успешного завершения всех задач для сбора атрибутов в этом бизнес-процессе. При ее выполнении отправляется запрос в BPM-систему для обновления контекста бизнес-процесса с учетом собранных данных.

4.2. Последовательность сбора данных

4.2.1. Получение данных из BPM-системы

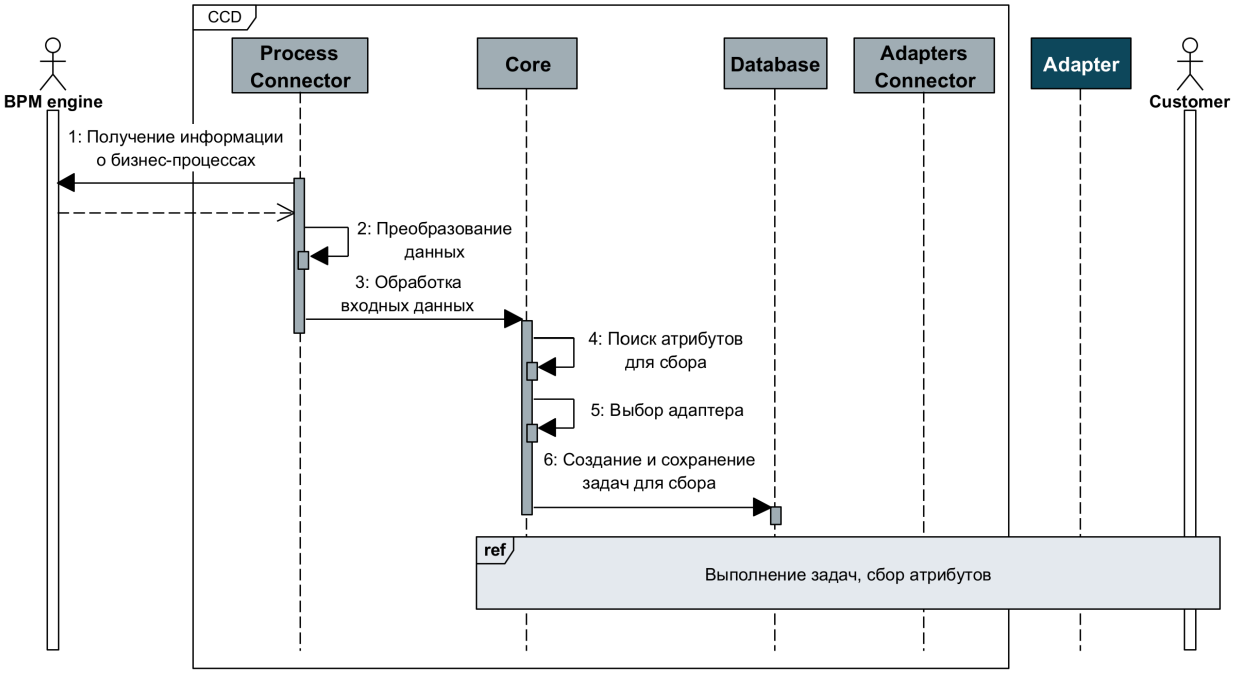

Рис. 4: Диаграмма последовательностей, описывающая получение данных из BPM-системы

На рисунке 4 представлена диаграмма последовательностей, описывающая процесс обработки входных данных из BPM-системы.

Процесс сбора начинается с получения данных контекста бизнес-процесса из BPM-системы. Для этого могут быть использованы различные подходы в зависимости от выбранного способа интеграции с BPM-системой (раздел 2.2), например, это может быть периодический опрос BPM-системы через REST API или получение нотификаций по протоколу AMQP.

Для основного варианта интеграции был выбран продукт CPM, а для взаимодействия с ним используется REST API. В связи с эти получение данных о необходимых бизнес-процессах реализовано с помощью периодического опроса, интервал которого настраивается через конфигурационные параметры приложения.

После получения данных о бизнес-процессе происходит их преобразование в формат классов доменной модели CCD. После чего данные

передаются на обработку в компонент «Core», в котором данные о каждом полученном бизнес-процессе обрабатываются в отдельном потоке, что позволяет сократить время на обработку.

Для каждого бизнес-процесса происходит фильтрация атрибутов для определения тех, которые необходимо запросить у пользователя. Для этого используется информация из конфигурационных файлов, в которых для каждого бизнес-процесса указан список атрибутов для сбора и условия, при которых их нужно собрать.

После получения списка атрибутов для сбора выполняется выбор канала коммуникации для связи с пользователем. Выбор канала также основан на конфигурации сбора данных.

После этого выполняется сохранение в базе данных всей необходимой информации и создание задач для сбора атрибутов, после чего выполнение потока, отвечающего за обработку входных данных, заканчивается.

4.2.2. Выполнение задач для сбора атрибутов

На рисунке 5 представлена диаграмма последовательностей, описывающая выполнение задач сбора атрибутов и обработку сообщений пользователя.

Для выполнения задач, хранящихся в базе данных, используется периодический вызов функции обработки задач. Интервал между вызовами настраивается через конфигурационные параметры CCD.

На первом этапе осуществляется поиск задач в базе данных, находящихся в состоянии «WAIT» и готовых к выполнению. Каждая найденная задача запускается на обработку в отдельном потоке, что позволяет распараллелить выполнение и уменьшить для пользователя время ожидания следующего атрибута для сбора.

Если задача относится к типу «ATTRIBUTE», она вместе с необходимыми данными передаётся в компонент «Adapters Connector». Этот компонент формирует сообщение для пользователя и вызывает соответствующий адаптер для отправки сообщения.

В случае успешной отправки статус задачи обновляется на

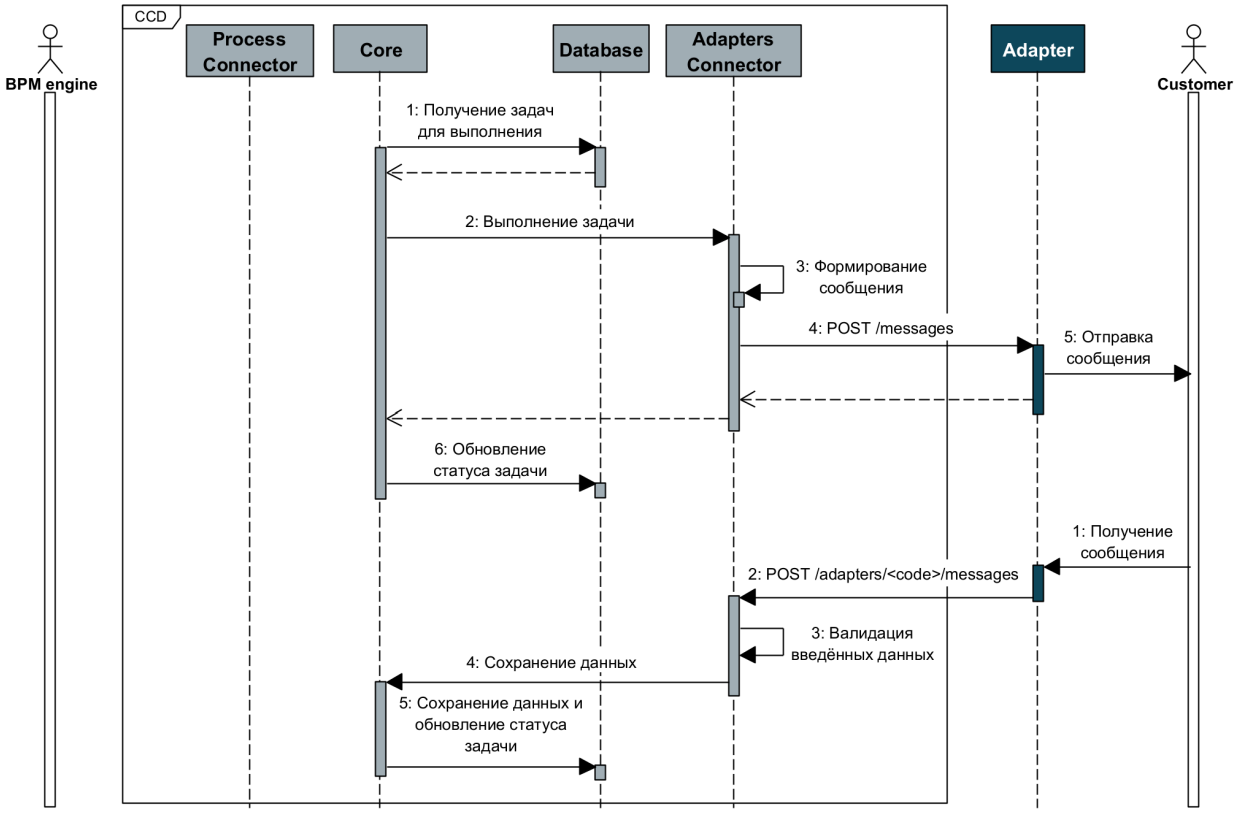

Рис. 5: Диаграмма последовательностей, описывающая выполнение задач с типом «ATTRIBUTE» и обработку сообщений пользователя

«COLLECTING». Если при вызове адаптера произошла ошибка, задача переводится в статус «ERROR». На этом выполнение потока, отвечающего за обработку задачу, завершается.

4.2.3. Обработка сообщений пользователя

При получении сообщения от пользователя адаптер передаёт его, вместе с данными о пользователе, в систему CCD через единый интерфейс.

После получения данных, введённых пользователем, выполняется их валидация на соответствие типу запрашиваемого атрибута. Если проверка не выполняется, то этот же атрибут повторно запрашивается у пользователя.

При успешной валидации данные передаются в компонент «Core», в котором сохраняются полученные данные и обновляется состояние соответствующей задачи на «COMPLETE».

4.2.4. Обновление контекста бизнес-процессов в BPM-системе

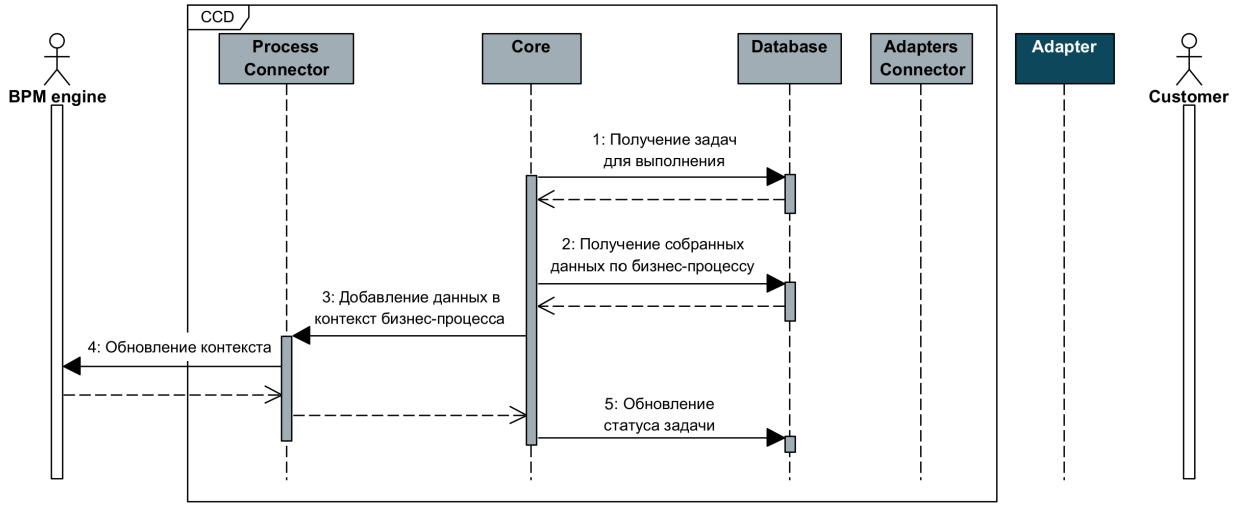

Рис. 6: Диаграмма последовательностей, описывающая выполнение задач с типом «BUSINESS_PROCESS»

На рисунке 6 представлена диаграмма последовательностей, описывающая выполнение финальной задачи сбора бизнес-процесса, которая предназначена для отправки собранной информации в BPM-систему. Следует отметить, что вызов под пунктом 1 на рисунке 6 совпадает с вызовом под пунктом 1 «Получение задач для выполнения» на рисунке 5.

При выполнении задачи с типом «BUSINESS_PROCESS» сперва осуществляется получение всех ранее собранных атрибутов. Затем эти данные передаются в компонент «Process Connector», в котором формируется запрос в BPM-систему для добавления собранных данных в контекст бизнес-процесса.

После успешного добавления данных в BPM-систему статус задачи обновляется на «COMPLETE», если при обновлении данных произошла ошибка, статус задачи устанавливается в значение «ERROR».

4.3. Доменная модель

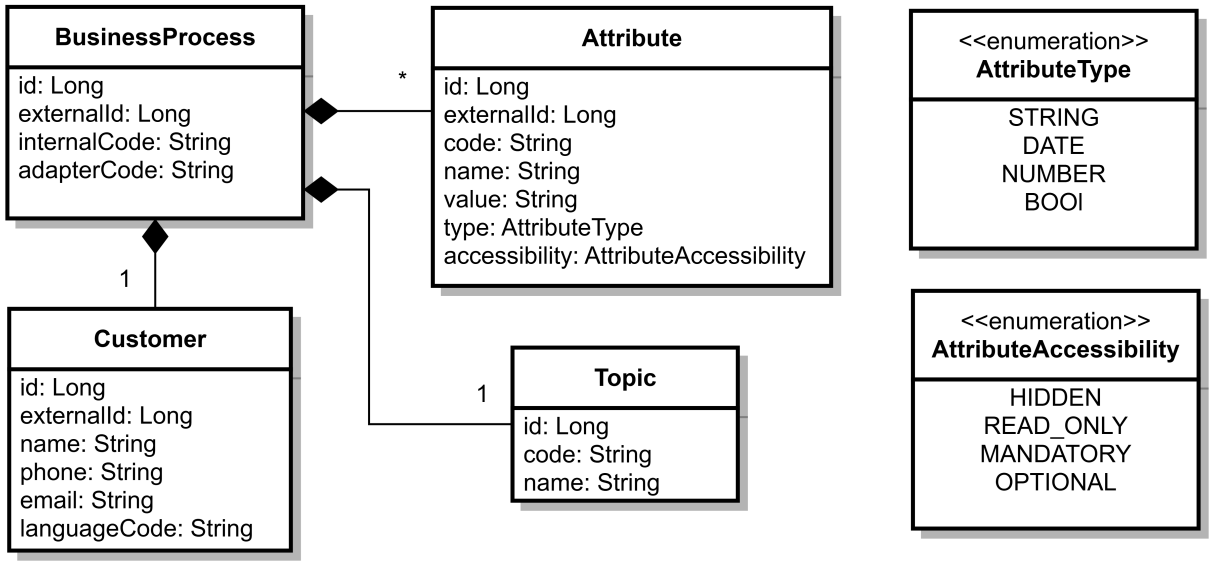

Рис. 7: Диаграмма классов доменной модели

На рисунке 7 представлена диаграмма классов доменной модели разрабатываемой системы. Приведенные классы используются для передачи и обработки данных о бизнес-процессе и его контексте внутри системы. Компонент «Process Connector» после получения данных о бизнес-процессе от BPM-системы преобразует их в объекты этих классов. Стоит отметить, что при проектировании доменной модели была рассмотрена модель, используемая в продукте CPM, в связи с чем названия свойств классов специфичны для CPM, но в других BPM-системах имеются аналогичные по функциональности свойства.

Класс «BusinessProcess» является основной сущностью и описывает все данные о бизнес-процессе. Свойство «internalCode» служит для идентификации процесса внутри CCD, а значение этого свойства задаётся через конфигурационные файлы. Свойство «adapterCode» указывает на адаптер к каналу коммуникации, который будет использован для запроса данных от пользователя. Значение этого свойства определяется на этапе выбора адаптера для бизнес-процесса (пункт 5 на рисунке 4). Значения остальных свойств класса формируются на основе данных, полученных от BPM-системы.

Класс «Topic» предназначен для определения типа экземпляра бизнес-процесса. В продукте CPM данная сущность служит для классификации обращений и определяет процесс обработки. При интеграции с другими BPM-системами этот класс может быть использован для идентификации бизнес-процесса, экземпляром которого являются поступившие из BPM-системы данные.

Для идентификации пользователя и способов связи с ним используется класс «Customer».

Класс «Attribute» определяет атрибуты контекста бизнес-процесса. Каждый атрибут содержит значение, код и имя атрибута, а также тип (класс «AttributeType») и видимость атрибута (класс «AttributeAccessibility»).

Необходимо отметить, что каждая сущность включает свойство «id», значение которого генерируется с помощью последовательности при сохранении в базу данных, а также свойство «externalId», представляющее идентификатор во внешней BPM-системе. Такой подход позволяет не зависеть от внешних идентификаторов и гарантировать уникальность идентификаторов, используемых внутри CCD.

4.4. Структура базы данных

На рисунке 8 представлена схема базы данных. Схема содержит таблицы, соответствующие основным сущностям доменной модели системы, а также таблицу «Task» для хранения задач по сбору данных.

Необходимо отметить, что таблица «Task» содержит поля «businessProcessId», «customerId» и «attributeId», несмотря на то, что значения «businessProcessId» и «customerId» могут быть получены транзитивным соединением таблиц, зная только «attributeId». Было решено использовать такую денормализованную форму, так как одним из самых частых запросов к базе данных является выборка задач, доступных для выполнения. Для выполнения такого запроса необходима группировка и фильтрация по полям «businessProcessId» и «customerId». Хранение этих значений в таблице «Task» позволяет создать индексы

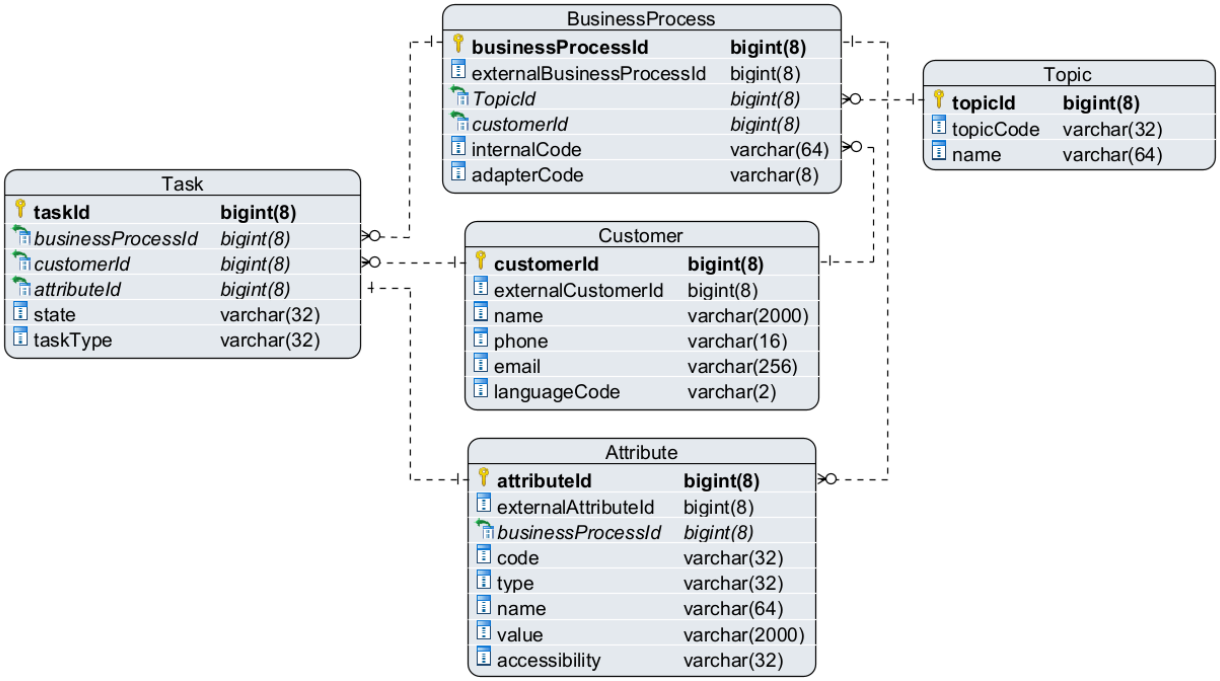

Рис. 8: Диаграмма схемы базы данных

по ним, благодаря чему ускоряется выполнение запроса.

4.5. Конфигурация системы

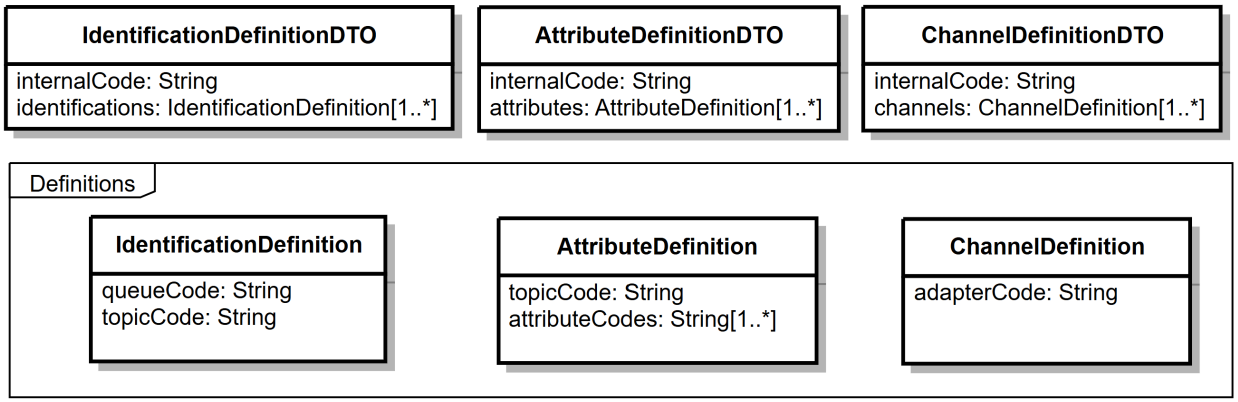

Рис. 9: Диаграмма классов конфигурации

Для конфигурирования логики обработки и сбора данных в разрабатываемой системе используются JSON-файлы, с помощью которых можно задать необходимые параметры сбора данных для каждого бизнес-процесса. На рисунке 9 представлена диаграмма классов, описывающих структуру конфигурационных данных. Эта структура соответ-

ствует формату заполнения JSON-файлов. В связи с тем, что конфигурация используется для разных целей, было решено разделить ее на три типа, каждый из которых задаётся в разных файлах.

Конфигурация в формате класса «IdentificationDefinitionDTO» используется для идентификации бизнес-процессов в компоненте «Process Connector», а в случае интеграции с BPM-системой по REST API — для составления поискового запроса. В рамках данной конфигурации задаётся код процесса, который будет использован внутри системы CCD для дальнейшей обработки процесса. Этот код присваивается процессу после его получения и идентификации (свойство «internalCode» в классе «BusinessProcess», раздел 4.3). Список возможных способов идентификации процесса задаётся с помощью класса «IdentificationDefinition», который включает в себя код очереди процесса («queueCode») и код типа процесса («topicCode»). Для успешной идентификации или поиска процесса должны совпасть все значения в рамках одного объекта «IdentificationDefinition», но если таких объектов задано несколько, то необходимо выполнение условий хотя бы для одного из них.

Так как условия, заданные с помощью «IdentificationDefinition», могут выполняться для нескольких экземпляров бизнес-процессов, то свойство «internalCode» фактически используется для группировки процессов с одинаковыми конфигурациями атрибутов и каналов коммуникации.

Конфигурация в формате класса «AttributeDefinitionDTO» используется в компоненте «Core» для определения атрибутов, которые необходимо запросить у пользователя. Благодаря коду процесса, заданному после идентификации, можно получить список конфигурации атрибутов только для текущего бизнес-процесса. С помощью объектов класса «AttributeDefinition» можно для каждого кода типа процесса (свойство «topicCode») задавать список кодов атрибутов для сбора (свойство «attributeCodes»).

Конфигурация в формате класса «ChannelDefinitionDTO» предназначена для выбора адаптера к каналу коммуникации для связи с пользователем. С помощью класса «ChannelDefinition» для каждого бизнес-

процесса определяется список кодов возможных адаптеров. При этом сами параметры адаптеров — их коды, тип используемого контакта и URL-адрес для отправки запросов — указываются в конфигурационном файле Spring-приложения.

4.5.1. Добавление конфигурационных файлов и валидация

Стоить отметить, что конфигурационные файлы могут быть подключены после сборки приложения, для этого достаточно указать путь к файлам с помощью параметра classpath при запуске приложения. Файлы каждого типа конфигурации определяются с помощью префикса, что позволяет подключать любое количество файлов конфигурации.

При запуске приложения все данные из конфигурационных файлов загружаются в память и проходят валидацию, которая включает в себя:

1. контроль уникальности задаваемого в конфигурации внутреннего кода бизнес-процесса (свойство «internalCode»);
2. проверка наличия для каждого процесса всех типов конфигураций;
3. проверка существования кодов адаптеров, которые задаются конфигурацией в формате «ChannelDefinitionDTO».

Данные проверки позволяет предотвратить работу приложения в некорректном состоянии конфигурационных настроек.

4.6. Интерфейс для адаптеров

Для минимизации затрат на разработку новых адаптеров к каналам коммуникации были зафиксированы интерфейсы, по которым все адаптеры будут взаимодействовать с CCD.

4.6.1. Отправка сообщения пользователю

Каждый адаптер должен обрабатывать HTTP-запросы, содержащие контактные данные пользователя и сообщение пользователю (ли-

Листинг 1: Пример запроса к адаптеру для отправки сообщения пользователю.

```
POST /messages
```

```
{
  "contact": "<контакт клиента>",
  "message": "<сообщение пользователю>"
}
```

стинг 1). В контактных данных каждому адаптеру передаются данные, необходимые для работы в соответствующем канале коммуникации, например, для чат-ботов это может быть номер телефона, а для email-бота — электронная почта клиента.

4.6.2. Получение сообщения от пользователя

Система CCD обрабатывает входящие запросы от адаптеров (листинг 2), при этом в параметре пути используется идентификатор адаптера «adapterCode», а в теле запроса передаются контактные данные клиента и введённое сообщение. Параметр «adapterCode» нужен для определения адаптера, из которого пришел запрос. Это позволяет сопоставить введенное пользователем сообщение с соответствующей задачей по сбору данных.

Листинг 2: Пример запроса из адаптера в CCD для обработки сообщения от пользователя.

```
POST /adapters/<adapterCode>/messages
```

```
{
  "contact": "<контакт клиента>",
  "message": "<сообщение от пользователя>"
}
```

4.7. Реализация адаптеров

Адаптеры к каналам коммуникации реализуются как отдельные сервисы, взаимодействующие с CCD по HTTP-интерфейсу. Каждый адаптер отвечает только за отправку сообщений пользователю и приём его

ответов, которые передаются в CCD для дальнейшей обработки. Благодаря тому, что бизнес-логика централизована в CCD, адаптеры остаются простыми в реализации.

В качестве примеров были разработаны два адаптера: чат-бот для мессенджера Telegram и чат-бот для социальной сети ВКонтакте. Они были реализованы на языке Python с использованием фреймворка FastApi [23] и библиотек aiogram [18] и vk-api [21].

5. Внедрение

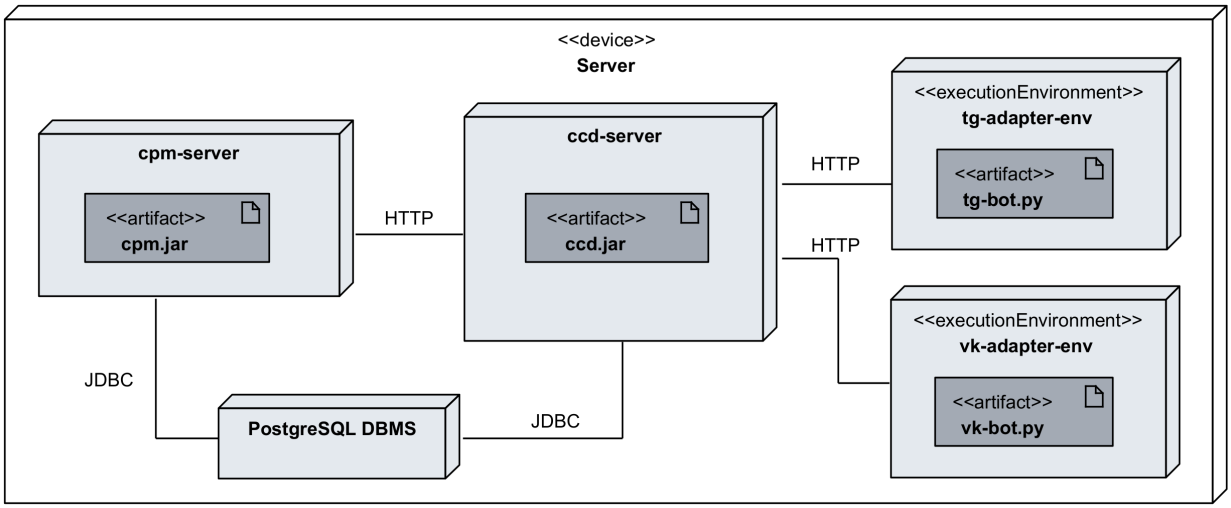

Рис. 10: Диаграмма развёртывания системы

После разработки системы для сбора данных контекста бизнес-процессов было выполнено её тестовое развёртывание совместно с продуктом CPM на существующей инфраструктуре. Диаграмма развёртывания системы представлена на рисунке 10.

Для запуска системы использовалась виртуальная машина с предварительно установленными и настроенными базой данных PostgreSQL и системой CPM. Для хранения данных системы CCD была создана отдельная схема в существующей базе данных, к которой были применены миграции схемы базы данных с помощью Liquibase.

В качестве адаптеров к каналам коммуникации использовались чат-боты для Telegram и ВКонтакте, для которых были подготовлены изолированные виртуальные окружения Python со всеми необходимыми зависимостями. Запуск адаптеров осуществлялся внутри этих окружений.

После заполнения конфигурационного файла CCD с параметрами взаимодействия с адаптерами, CPM и базой данных, был запущен исполняемый JAR-файл, собранный с помощью Maven.

6. Апробация

После развёртывания системы для проверки её работоспособности было проведено ручное тестирование сценариев сбора данных контекста бизнес-процессов. Для этого в продукте CPM были сконфигурированы несколько простых бизнес-процессов, для которых необходимо было получить значения атрибутов от клиентов. Соответствующая конфигурация была внесена в систему CCD. В ходе тестирования все данные, введённые пользователями через каналы коммуникации Telegram и ВКонтакте, корректно сохранялись в контексте бизнес-процессов в продукте CPM, что подтверждает корректную работу системы для сбора данных.

Для оценки сложности реализации адаптера к новому каналу коммуникации для системы сбора данных был получен отзыв от разработчика, самостоятельно реализовавшего адаптер для электронной почты. В отзыве было отмечено: «Благодаря использованию единого интерфейса и удобной архитектуры, разработка адаптера становится максимально простой и эффективной». При этом был выявлен недостаток разработанной системы, связанный с неэффективным взаимодействием с BPM-системой CPM. Однако данное замечание может быть оперативно исправлено, поскольку коннектор к CPM реализован в отдельном модуле и может быть доработан независимо от основной части системы.

Для оценки удобства конфигурирования параметров сбора данных разработанной системы был получен отзыв от конфигуратора бизнес-процессов, который попробовал настроить сбор данных бизнес-процессов через CCD. В качестве недостатка был выделен тот факт, что при наличии нескольких экземпляров бизнес-процесса на одного клиента, клиент не может отказаться от заполнения каких-либо процессов, если они были сконфигурированы для сбора данных. Несмотря на данное ограничение, было отмечено повышение скорости обработки бизнес-процессов и снижение нагрузки на операторов поддержки, достигнутое за счёт автоматизации взаимодействия с клиентами.

Заключение

В ходе данной работы были достигнуты следующие результаты.

1. Проведен обзор существующих подходов для интеграции приложений и проанализированы возможности их применения для реализации системы сбора данных контекста бизнес-процессов.

2. Проведен обзор двух наиболее популярных BPM-систем, используемых в России, а также BPM-системы CPM компании Nexign. В результате обзора были выявлены способы интеграции с BPM-системами.

3. Разработана архитектура системы с учетом масштабируемости по каналам коммуникаций и возможности интеграций с различными BPM-системами.

4. Реализована система для сбора данных, обеспечивающая интеграцию с продуктом CPM, возможность конфигурирования и диалог с пользователем для запроса атрибутов. Также реализованы адаптеры к каналам коммуникации в виде чат-ботов для Telegram и ВКонтакте.

5. Разработанная системы была внедрена совместно с существующей BPM-системой CPM компании Nexign для тестового использования.

6. Проведена апробация системы.

Исходный код проекта закрыт и находится под соглашением о неразглашении.

Список литературы

[1] Apache Maven. — URL: https://maven.apache.org/ (дата обращения: 1 апреля 2024 г.).

[2] Camunda. — URL: https://camunda.com/ (дата обращения: 9 марта 2024 г.).

[3] Camunda, Kafka Connector. — URL: https://docs.camunda.io/docs/8.7/components/connectors/out-of-the-box-connectors/kafka/ (дата обращения: 9 марта 2024 г.).

[4] Ebert Nico, Weber Kristin, Koruna Stefan. Integration Platform as a Service // Journal of Business Information Systems Engineering. — 2017. — Vol. 59. — P. 375–379. — URL: https://link.springer.com/article/10.1007/s12599-017-0486-0 (дата обращения: 9 марта 2025 г.).

[5] Frantz Rafael Z., Corchuelo Rafael, Molina-Jiménez Carlos. A proposal to detect errors in Enterprise Application Integration solutions // Journal of Systems and Software. — 2012. — Vol. 85, no. 3. — P. 480–497. — URL: https://doi.org/10.1016/j.jss.2011.10.048 (дата обращения: 3 марта 2025 г.).

[6] Goel Anurag. Enterprise Integration EAI vs. SOA vs. ESB. — 2006. — URL: http://ggatz.com/images/Enterprise_20Integration_20-_20SOA_20vs_20EAI_20vs_20ESB.pdf (дата обращения: 8 марта 2025 г.).

[7] Gorkhali Anjee, Xu Li Da. Enterprise Application Integration in Industrial Integration: A Literature Review // Journal of Industrial Integration and Management. — 2016. — Vol. 1, no. 4. — URL: https://doi.org/10.1142/S2424862216500147 (дата обращения: 2 марта 2025 г.).

[8] Hibernate. — URL: https://hibernate.org/ (дата обращения: 6 мая 2024 г.).

[9] Kazachenko Alexandr. Camunda external tasks. — URL: https://habr.com/ru/companies/tbank/articles/490656/ (дата обращения: 9 марта 2024 г.).

[10] Kusák Bc. David. Comparison of Enterprise Application Integration Platforms // Charles University in Prague, Faculty of Mathematics and Physics. — 2010. — URL: https://dspace.cuni.cz/handle/20.500.11956/33964 (дата обращения: 3 марта 2025 г.).

[11] Liquibase. — URL: https://www.liquibase.com/ (дата обращения: 6 мая 2024 г.).

[12] MyBatis. — URL: https://mybatis.org/mybatis-3/ (дата обращения: 6 мая 2024 г.).

[13] PostgreSQL. — URL: https://www.postgresql.org/ (дата обращения: 6 мая 2024 г.).

[14] Soomro Tariq Rahim, Awan Abrar Hasnain. Challenges and Future of Enterprise Application Integration // International Journal of Computer Applications. — 2012. — Vol. 42, no. 7. — P. 42–45. — URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d7ec0f68be6d35e374a18b400939b379b9b363c9 (дата обращения: 8 марта 2025 г.).

[15] Spring Framework. — URL: https://spring.io/ (дата обращения: 1 апреля 2024 г.).

[16] Survey on the run-time systems of enterprise application integration platforms focusing on performance / Daniela L. Freire, Rafael Z. Frantz, Fabricia Roos-Frantz, Sandro Sawicki // Journal of Software: Practice and Experience. — 2019. — Vol. 49, no. 3. — P. 341–360. — URL: https://doi.org/10.1002/spe.2670 (дата обращения: 2 марта 2025 г.).

[17] Wikipedia. Bussines Process Management. — URL: https://en.wikipedia.org/wiki/Business_process_management (дата обращения: 3 марта 2025 г.).

[18] aiogram: асинхронный фреймворк для Telegram API. — URL: https://aiogram.dev/ (дата обращения: 11 мая 2024 г.).

[19] tadviser. Enterprise application integration. — URL: https://www.tadviser.ru/index.php/Статья:Enterprise_application_integration (дата обращения: 8 марта 2025 г.).

[20] tadviser. Управление бизнес-процессами, рынок России. — URL: https://www.tadviser.ru/index.php/Статья:Business_Process_Management_System_-_Управление_бизнес-процессами_(рынок_России) (дата обращения: 9 марта 2024 г.).

[21] vk-api – Python модуль для создания скриптов для социальной сети Вконтакте. — URL: https://vk-api.readthedocs.io/en/latest/ (дата обращения: 11 мая 2024 г.).

[22] Системы управления бизнес-процессами ELMA. — URL: https://www.elma-bpm.ru/ (дата обращения: 9 марта 2024 г.).

[23] Фреймворк Fast API. — URL: https://fastapi.tiangolo.com/ (дата обращения: 11 мая 2024 г.).