(Этап 0) Software Architecture Document

Черновик

Внимание: синие комментарии в разделах документа нужно удалять!

Примечание: структура SAD продиктована стандартом описания системной архитектуры "4+1" Ф. Крачтена (ссылка на обзор).

<Имя проекта>

Software Architecture Document

Версия <1.0>

История изменений

Дата	Версия	Описание	Автор
29.01.2021	1.0	черновик	Karinski, Andrey

Оглавление

Введение

Введение должно содержать обзор всего документа и включать цель, область применения, определения и сокращения, ссылки на другие архитектурно-важные документы.

Этот документ содержит полный архитектурный обзор системы и использует ряд архитектурных представлений в формате «4+1» (Филиппа Крачтена) для отображения различных аспектов системы. Сам документ служит для отражения важных архитектурных решений, принятых при проектировании системы.

Цель

Весь документ в целом содержит полный архитектурный обзор системы, используя ряд различных архитектурных представлений для отображения различных важных аспектов системы.

Данный же раздел описывает определяет место документа во всей проектной документации и кратко описывает структуру документа. Так же важно указать проектные роли лиц, которые этот документ используют в работе.

Документ SAD содержит обзор архитектуры информационной системы «Интеграционная Платформа», создаваемой как инструмент для обеспечения слабосвязанного взаимодействия многочисленных бизнес-приложений в рамках единых интеграционных сценариев.

Содержимое документа важно в первую очередь проектной команде (разработчикам и аналитикам), но он также может быть интересен стороне заказчика, чтобы получить подробную информацию о способах реализации функциональных и нефункциональных требований.

Документ разрабатывается системным архитектором, но может дорабатываться членами проектной команды, чьих навыков достаточно для проектирования и документирования архитектуры системы.

Первая версия документа создавалась после аналитической работы, но до перехода к программированию, следовательно, содержит лишь архитектурные концепции, которых следует придерживаться, но никак не подробную инструкцию по реализации. В дальнейшем документ будет дорабатываться и актуализироваться.

UML-диаграммы в документе были созданы в инструменте «Sparx Enterprise Architect».

Область действия

Краткое описание того, на что влияет данный документ: другие документы, роли, окружения.

Документ зависит от «Видения» (описания концепции системы), перечня «Вариантов использования», представляющих описание функциональных требований, и «Дополнительной спецификации», содержащей нефункциональные требования и атрибуты качества системы.

Документ оказывает влияние на «Спецификацию реализации Вариантов использования», «План развертывания» и «План измерений» (описывает метрики системы и критерии их определения).

Определения, сокращения и аббревиатуры

Здесь даны определения всех терминов и сокращений, необходимых для правильной интерпретации содержимого документа. Может содержать ссылку на раздел Глоссария, посвященный техническим терминам.

Список определений и сокращений, используемых в документе, вместе с их расшифровкой.

- *Каноническая модель* - шаблон проектирования, используемый для связи между различными форматами данных. Модель данных, которая является надмножеством всех других и являющаяся неким «общим языком», на котором все существующие системы обмениваются данными между собой. Применение такого паттерна способно значительно снизить затраты на реализацию сценариев интеграции бизнес-приложений.
- *Publisher-Subscriber* - шаблон проектирования передачи сообщений, в котором отправители сообщений, именуемые издателями (publishers), напрямую не привязаны к подписчикам (subscribers). Вместо этого сообщения делятся на классы и не содержат сведений о своих подписчиках, если таковые есть. Аналогичным образом подписчики имеют дело с одним или несколькими классами сообщений, абстрагируясь от конкретных издателей.
- *Компенсация* - аналог распределенной транзакции в системе с событийно-ориентированной архитектурой. Возврат к исходной точке цепочки успешно выполненных действий по причине того, что результат этих действий или побочные эффекты больше не представляют ценности и должны быть отменены.
- *ODATA* - открытый веб-протокол для запроса и обновления данных, позволяет выполнять операции с ресурсами, используя в качестве запросов HTTP-команды, и получать ответы в форматах XML или JSON.
- *GraphQL* - язык запросов и манипулирования данными с открытым исходным кодом, состоит из системы типов, языка запросов и семантики выполнения, статической проверки и самоанализа типов.

- *Идемпотентность* - свойство объекта или операции при повторном применении операции к объекту давать тот же результат, что и при первом.
- *Идемпотентный подписчик* - потребитель сообщений, который может правильно обрабатывать повторяющиеся сообщения: отслеживать и удалять уже обработанные сообщения-дубликаты.
- *Очередь "мертвых" сообщений* - очередь, куда перенаправляются сообщения, которые не смогли обработать (из-за ошибок бизнес-логики, изменившихся состояний и т.п.) получатели в обычных очередях.
- *Saga* - шаблон проектирования, способ управления согласованностью данных между сервисами в сценариях распределенных транзакций. Последовательность транзакций, которая обновляет каждую службу и публикует сообщение или событие для активации следующего шага транзакции. Если шаг завершается ошибкой - выполняет компенсирующие транзакции, которые отменяют предыдущие транзакции.
- *Retry* (*Повторитель*) - шаблон проектирования, позволяет приложению управлять временными сбоями при попытке подключиться к службе или сетевому ресурсу путем прозрачного повторного выполнения операции, завершившейся сбоем.
- *Circuit Breaker* (*Предохранитель*) - шаблон проектирования, используется для обнаружения сбоев и инкапсулирует логику предотвращения постоянного повторения сбоя во время обслуживания, временного сбоя внешней системы и т.п.

Ссылки

Здесь должен быть представлен полный список всех документов, на которые есть ссылки в других разделах данного документа. Не забыть указать источники.

- Нефункциональные требования перечислены в документе Supplementary Specification.
- Технологические ограничения описаны в документе Технологический стек.

Обзор

Краткое описание содержания документа и принципа его организации.

Документ описывает архитектуру как серию представлений: Варианты Использования, Логическое, Процессное, Реализации, Развертывания. Каждый архитектурно важный момент может снабжаться дополнительными диаграммами, поясняющими концепцию.

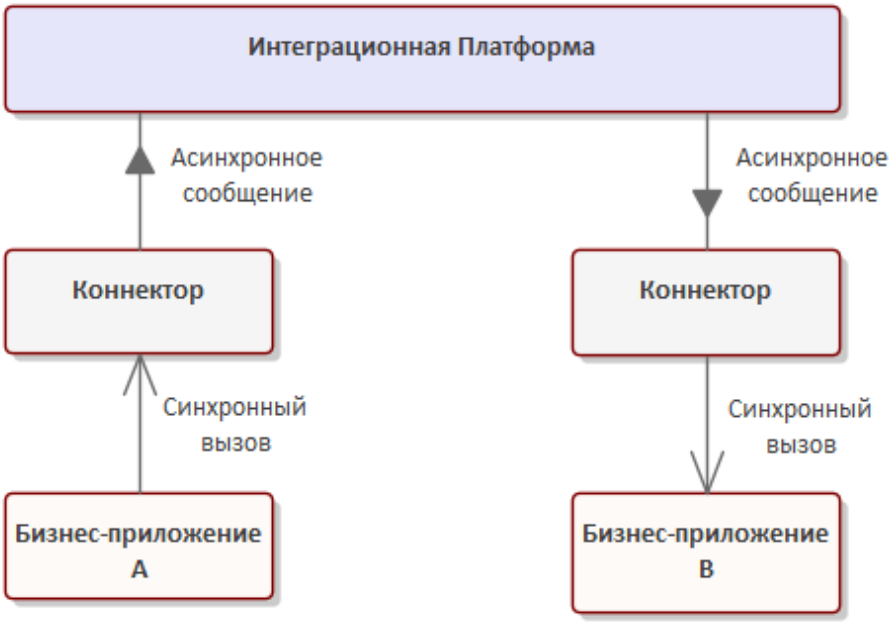
Архитектурное представление

Обобщенное и верхнеуровневое архитектурное представление системы, и то, как она представлена.
Описание метафоры архитектуры или основополагающего паттерна.

Кроме того, раздел должен содержать список необходимых архитектурных представлений (UseCase, Logical, Process, Deployment, Implementation) и для каждого представления объяснять, элементы модели они описывают.

Вся система строится по принципу интеграционной шины - то есть промежуточного компонента, выступающего в роли медиатора (посредника) для общения между собой бизнес-приложений. Каждое бизнес-приложение подсоединяется к шине (медиатору) через специальный коннектор, который адаптирует внутреннюю специфику конкретного бизнес-приложения к единообразному способу взаимодействия, характерному для самой шины. Кроме того, шина может содержать инфраструктурные сервисы - справочники, конвертеры и т.п.

Фактически, взаимодействие между коннекторами (а значит, между бизнес-приложениями) осуществляется при помощи механизма асинхронного обмена сообщениями: каждый коннектор отправляет в шину сообщения (события) и может подписываться на сообщения других коннекторов. В теле сообщений коннекторы передают данные в формате Канонической Модели, то есть единой референсной информационной модели, являющейся единым общим языком для взаимодействия различных и специфичных бизнес-приложений.



Архитектурные цели и ограничения

Описание требований и целей системы, которые оказывают существенное влияние на архитектуру (например, объем данных, информационные потоки и нагрузка, безопасность, конфиденциальность, надежность, т.п.)
Так же может содержать описание особых ограничений, таких как:

- *Стратегии проектирования и реализации*
- *Инструменты разработки*
- *Структура команды*
- *Доступные ресурсы*
- *Унаследованный код*

Основный архитектурные цели системы исходят из перечня нефункциональных требований к ней, в подробностях описанных в документе Supplementary Specification. Краткий же перечень требований приведен ниже:

Обеспечение взаимодействия бизнес-приложений в рамках интеграционных сценариев

Общая информационная модель

Так как в интеграционных сценариях могут участвовать много различных бизнес-приложений, у каждого из которых своя собственная модель предметной области, то для упешного взаимодействия им требуется общая информационная модель (Каноническая Модель), являющаяся некоторым "общия языком". Такая модель должна быть с одной стороны достаточно выразительной для удовлетворения всех интеграционных задач, а с другой стороны, не содержать "лишней" специфики конкретных бизнес-приложений и их доменов.

Такая модель должна быть представлена в виде

- Аналитического представления - сущностей с атрибутами и отношениями, описанных в виде набора UML-диаграмм
- Транспортного представления - Json-схем для сериализации аналитических сущностей
- Интеграционного представления - набора интеграционных событий в виде классов UML и Json-схем

Слабосвязанная архитектура

К различным интеграционным сценариям, а следовательно, и к *Интеграционной Платформе*, обычно предъявляются противоречивые требования: с одной стороны, скорость взаимодействия, близкая к реальному времени, с другой - длительное время жизни некоторых сценариев, исчисляемое днями и неделями. Кроме того, многочисленные бизнес-приложения, которые необходимо объединить ради решения общих интеграционных задач, несут много собственной специфики, которая будет неявно (транзитивно) влиять на соседние приложения.

Следовательно, *Интеграционная Платформа* должна обеспечить максимально возможную логическую и физическую изоляцию между отдельными бизнес-приложениями. Наиболее разумным выбором для обеспечения "слабой связности" является реализация системы согласно паттерну "*Подписчик-Публикатор*" (*Publisher-Subscriber*).

Далее, реализация *Publisher-Subscriber* для обеспечения выполнения многочисленных интеграционных сценариев закономерно вызовет интенсивный обмен событиями между бизнес-приложениями. В процессе такого обмена потоки событий потребуется дополнительно обрабатывать: фильтровать, перераспределять, трансформировать и т.п. Будет разумным при реализации Интеграционной Платформы примененить к обработке потоков событий *Реактивный дизайн*.

Прямое чтения данных через Data API

В классических *событийно-ориентированных* (*event-driven*) системах данные между взаимодействующими компонентами обычно передаются небольшими фрагментами в составе сообщений. Однако при взаимодействии уже существующих бизнес-систем, построенных на более традиционных подходах, обмен данными зачастую требуется в *пакетном* режиме, то есть "редко, но много".

Для обеспечения нормального функционирования унаследованных бизнес-приложений Интеграционная Платформа должна предоставить средства обмена данными в режиме прямого чтения, через специальный Data API. Такой API должен обеспечить возможность максимально быстро получить большой объем данных по протоколам ODATA, GraphQL или объектному запросу (Object Criteria). Такие запросы не должны напрямую менять данные, все изменения обязаны выполняться отправкой специальных событий. В любом случае, данные модели могут меняться только в ООП-стиле, то есть целостность, бизнес-правила и инварианты должна контролировать сама *Каноническая Модель*.

Долгие бизнес-транзакции

Некоторые интеграционные сценарии имеют не только весьма длительный жизненный цикл (дни и недели), но и требуют транзакционной логики, то есть восстановления целостности данных участвующих в сценарии бизнес-приложений. Следовательно, *Интеграционная Платформа* должна обеспечить выполнение такой транзакционной логики с помощью *сценариев компенсации,* так как они не зависят от возможностей инфраструктуры, при необходимости допускают ручное разрешение конфликтов и хорошо укладываются в событийно-ориентированную архитектуру.

Нормализация и обогащение данных, передаваемых между бизнес-приложениями

Для обеспечения выполнения относительно сложных интеграционных сценариев от Интеграционной Платформы потребуется дополнительная обработка потоков интеграционных событий.

Отложенная валидация

Очевидно, что для корректного взаимодействия между несколькими бизнес-приложениями посредством единой *Канонической Модели* требуется обеспечение полной целостности и корректности этой модели в момент ее построения. Однако в случае с унаследованными приложениями добиться этого не всегда представляется возможным. Разумным решением в данной ситуации видится применение *отложенной валидации*, то есть

- передача данных бизнес-приложением в Интеграционную Платформу в не полностью консистентном состоянии
- валидация модели внутри Интеграционной Платформы
- нормализация модели средствами Интеграционной Платформы
- передача модели следующим приложениям уже в консистентном виде.

Обогащение модели

Частный случай нормализации данных модели - обогащение. Например, адреса, телефоны, имена организаций, единицы измерения и т.п., представленные в отдельных бизнес-приложениях в виде строковых или числовых примитивов, в Канонической Модели следует представлять как элементы справочников или специализированных сущностей. Следовательно, Интеграционная Платформа должна обеспечить возможность обогащения модели посредством общих платформенных сервисов (геокодирования, распознавания и нормализации текста, корпоративных справочников).

Сложные правила маршрутизации

Для реализации паттерна Publisher-Subscriber зачастую требуется более сложная логика, чем подписка на конкретное событие. Например, подписчик может ограничить такую подписку дополнительными условиями и ограничениями. Система должна предоставить разработчикам бизнес-приложений возможность гибко управлять подпиской на события по их темам, типам, времени жизни, метаданным и содержимому.

Контроль и мониторинг интеграционных сценариев

В распределенной системе, выполняющей одновременно множество асинхронных процессов, крайне важно обеспечить прозрачный и удобный мониторинг выполнения сценариев.

Система должна обеспечить визуализацию маршрута движения событий (сообщений) между приложениями, время обработки каждого события, метаданные событий, сбои и ошибки. Кроме того, система должна хранить необработанные по некоторым причинам (ошибки, истечение времени жизни) события в специальных очередях "мертвых сообщений", чтобы администратор системы мог вручную разрешить некоторые конфликтные и неоднозначные систуации.

Система должна обеспечить *структурное логирование* всех ошибок и важных ситуацию логики. Журналы, относящиеся к отдельным бизнес-приложениям, должны быть доступны их разработчикам. Журналы, содержащие события целого интеграционного сценария, должны быть доступны разработчикам всех бизнес-приложений, участвующим в процессе.

Представление Вариантов Использования (Use Case View)

Перечисление вариантов использования или пользовательских историй, если они представляют значительную важность для системы и обладают влиянием на архитектуру (то есть в процессе их реализации будет затронуто значительное число архитектурных артефактов), либо они подчеркивают или иллюстрируют важную часть архитектуры.

Логическое представление (Logical View)

Описываются архитектурно значимые части модели системы, такие как ее разбиение на подсистемы, пакеты и модули. Для каждого важного пакета потребуется описать

- *значимые классы (разумеется, без подробностей, так как они будут проясняться только в процессе реализации)*
- *их обязанности*
- *несколько важных отношений, атрибутов и операций*

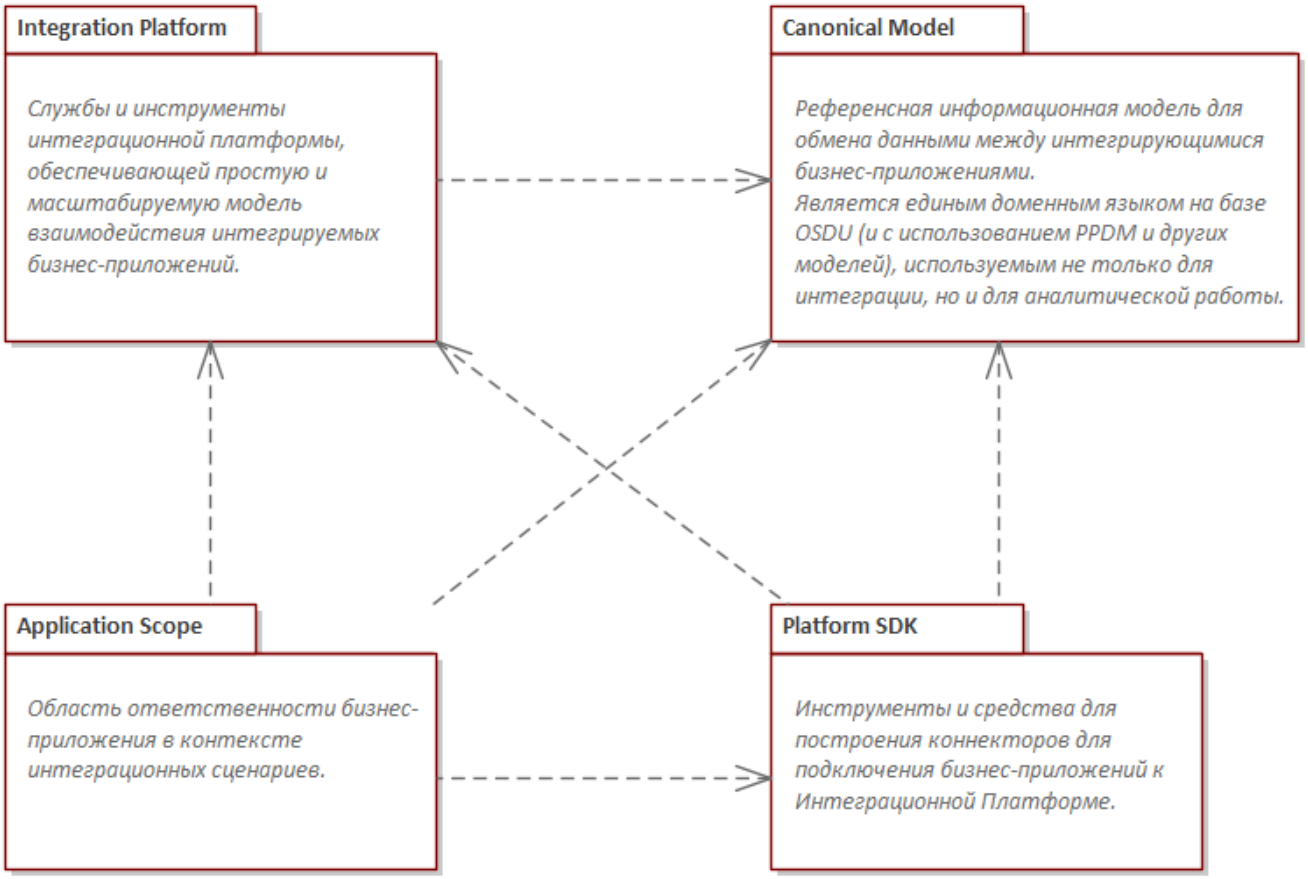
Так же стоит отметить, что описанные в разделе классы являются концептуальными (т.е. артефактами моделирования, а не реализации) и могут не соответствовать программным классам из модели реализации.

Обзор

Описание общих принципов декомпозиции модели с точки зрения слоев и иерархии пакетов.

Система условно делится на четыре основных пакета:

- Integration Platform - все, что напрямую касается обеспечения работы слабосвязанного событийно-ориентированного взаимодействия приложений.
- Canonical Model - единая референсная информационная модель, ее проекции (логическая, транспортная, контракты) и методические материалы.
- Application Scope - область ответственности бизнес-приложений (прежде всего коннекторы к приложениям, сами бизнес-приложения).
- Platform SDK - инструменты и инструкции для разработки новых коннекторов.



Архитектурно значимые пакеты

Для каждого значимого пакета включить подраздел с его описанием и диаграммой со всеми значимыми содержащимися в нем классами и пакетами. Для каждого значимого класса указать его имя, краткое описание, и при необходимости некоторые из его основных обязанностей, операций и атрибутов.

Система является не бизнес-приложением, а прикладным инструментом (платформой) для обеспечения легкой и удобной интеграции множества бизнес-приложений (уже существующих или только создающихся). Следовательно, функциональных требований в привычном виде у системы нет, а следовательно, содержимое архитектурно-значимых пакетов будет представлено в разделе "Представление реализации (Implementation View)". Так же следует заметить, что все перечисленные в п. 3 "Архитектурные цели и ограничения" требования являются архитектурно-значимыми, то есть оказывающими явное и измеримое влияние на архитектуру системы.

Реализация Вариантов использования

Раздел иллюстрируют, как система работает внутри, посредством описания реализаций нескольких выбранных (и важных) сценариев; объясняет, как для этого используются различные элементы модели.

По перечисленным в п. 5.2 причинам Варианты Использования не приводятся, а подробности работы бизнес-логики системы будут описаны в п. "Представление реализации".

Процессное представление (Process View)

Описание декомпозиции системы на бизнес-процессы. Удобно организовать раздел по группам взаимодействующих процессов. Хорошо так же проследить зависимость от соответствующих use cases.

TODO

Представление развертывания (Deployment View)

Описание физических (сетевых, аппаратных) конфигураций, в которых будет развернута и запущена создаваемая система. Для каждой конфигурации должны быть указаны физические узлы (сервера, процессоры), выполняющие код, и их взаимосвязи (сетевые соединения, очереди, т.п.), отображение процессов на физических узлах.

Основные компоненты системы развернуты в OpenShift в отдельных узлах:

- **Integration Middleware** - узел, где развернуты сервисы интеграционной платформы
- **Integration Adapter** - узел, где развернут коннектор к бизнес-приложению
- **PAAS** - место развертывания сервисов PAAS

На схеме ниже внутри узла **Integration Adapter** показано и само бизнес-приложение. Такой вариант развертывания вполне возможен (так как коннектор это организационно часть приложения), но не обязателен, приложение может развертываться отдельно.

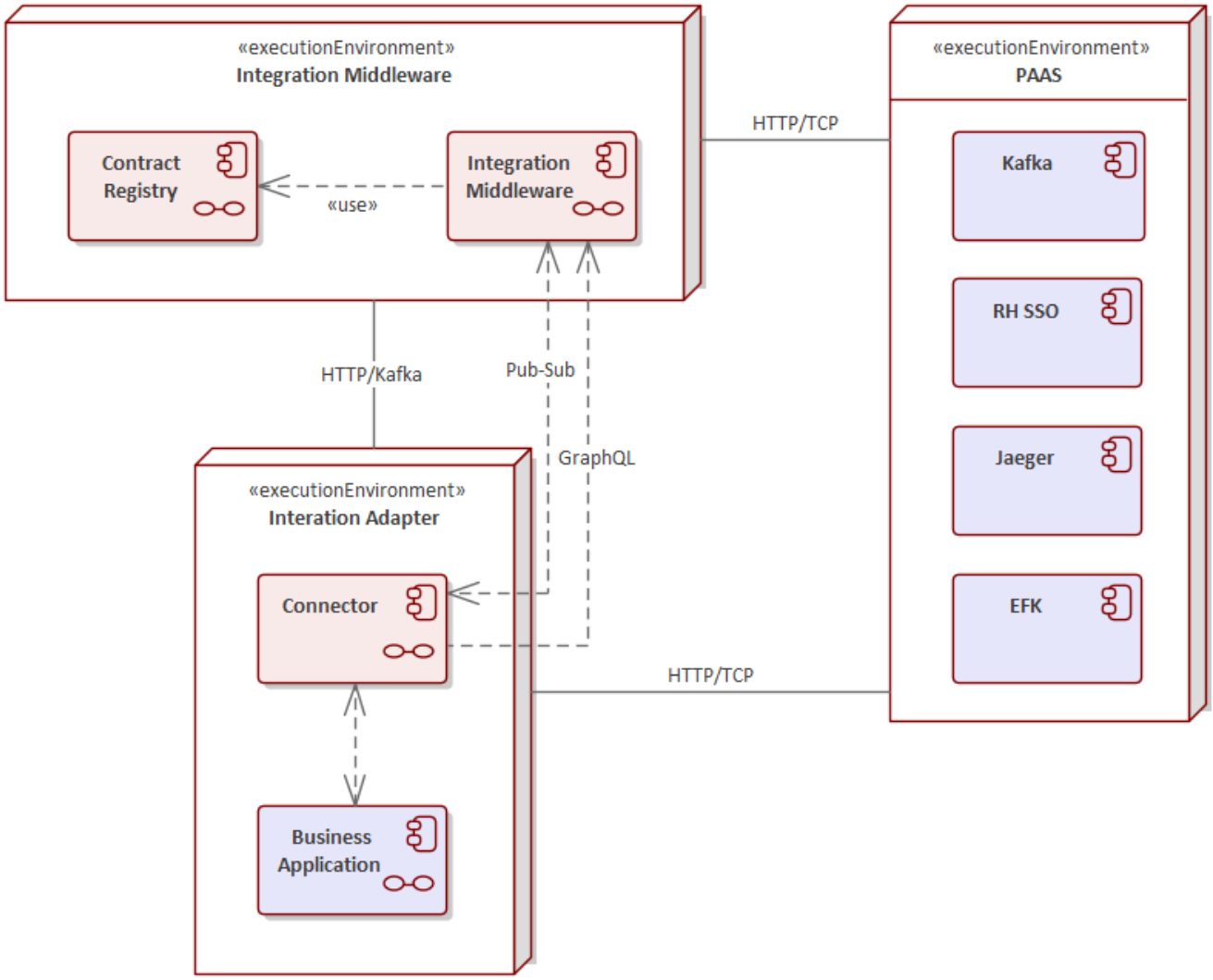
На схеме представлены основные единицы развертывания:

- Компонент Integration Middleware - обеспечивает обмен событиями между коннекторами бизнес-приложений.
- Компонент Contract Registry - хранит метаданные публикуемые бизнес-приложениями событий (контракты).
- Компонент Connector - посредник (медиатор), обеспечивающий логическую изоляцию и надежную связь по модели Pub-Sub с интеграционной платформой.

Кроме того, на схеме (сиреневым цветом) отмечены компоненты, не входящие напрямую в Интеграционную Платформу, но используемые ей:

- Kafka - брокер обмена сообщениями, используемый как ядро компонента Integration Middleware
- RH SSO - сервис авторизации, нужный коннекторам бизнес-приложений
- Jaeger - служба трассировки маршрутов сообщений
- EFK - средства структурного логирования

Перечень инфраструктурных компонентов неполный и может быть доработан в следующих версиях документа.



Представление реализации (Implementation View)

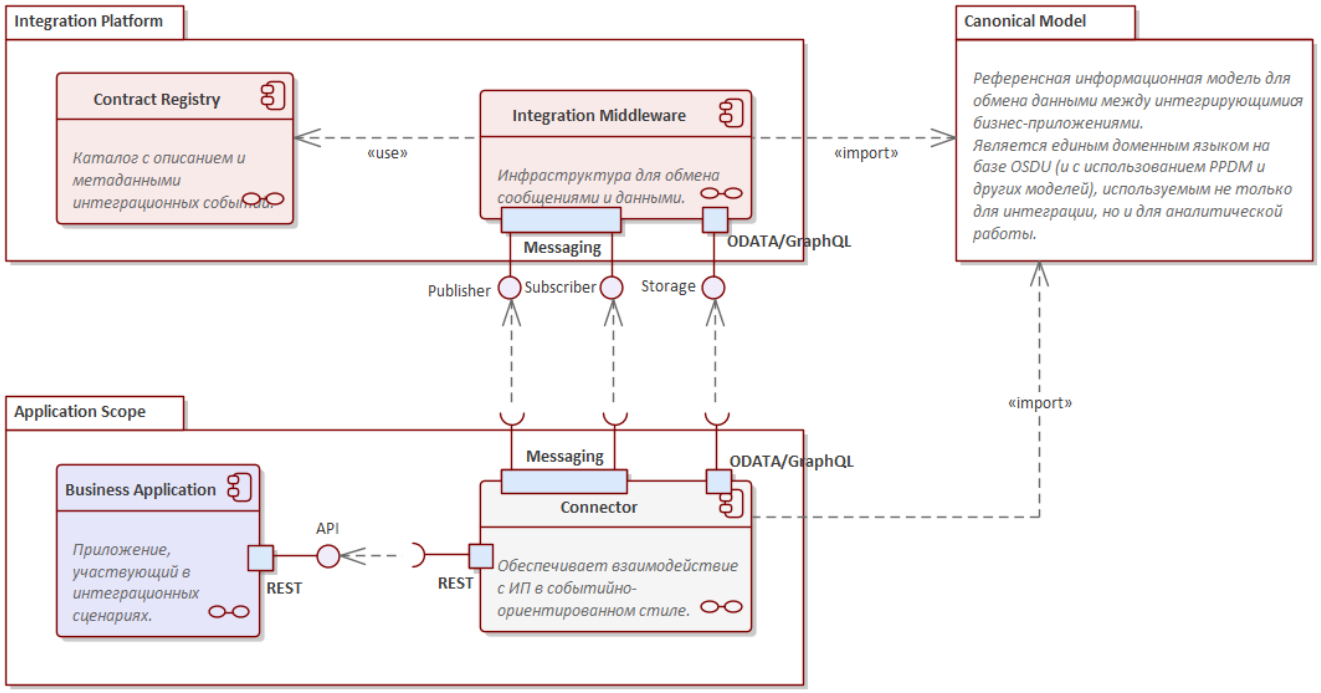
Описание общей структуры модели реализации, декомпозиция на уровни и подсистемы, любые архитектурно значимые элементы реализации.

Обзор

Описание основных уровней и слоев системы с точки зрения реализации, правила разделения на слои, границы между слоями. Имеет смысл описывать в виде диаграмм компонентов.

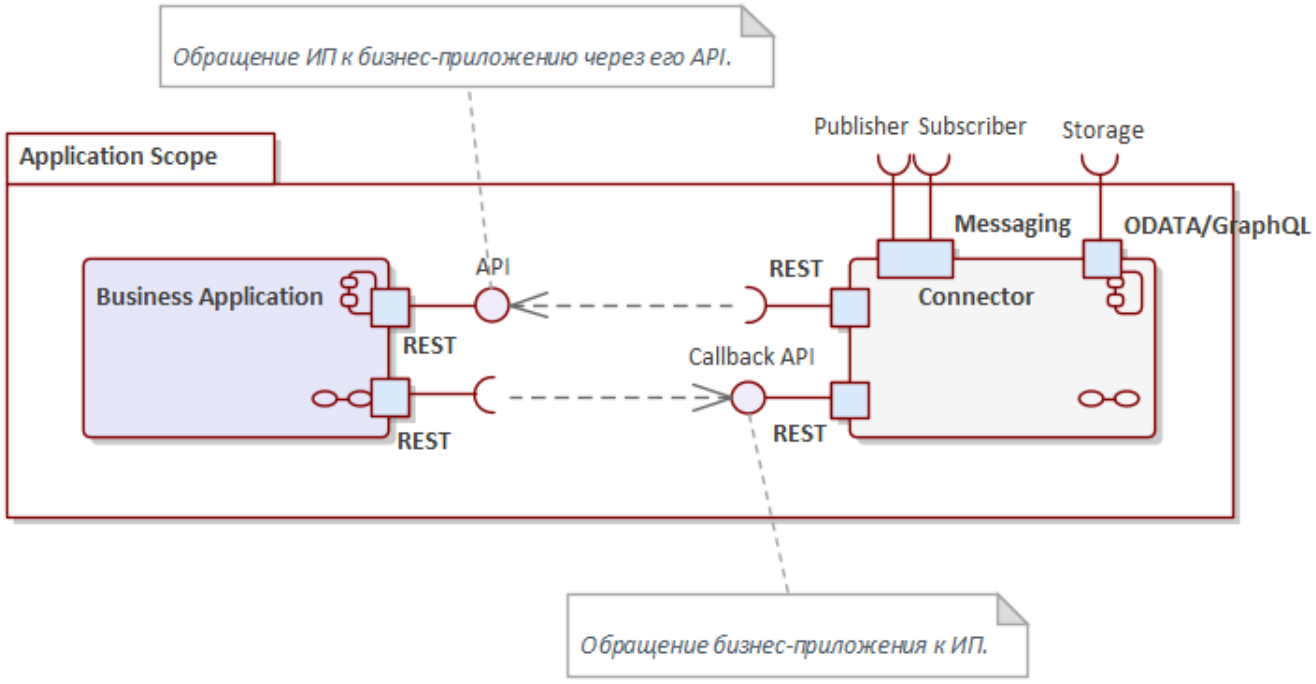
На верхнем уровне взаимодействие бизнес-приложения с интеграционной платформой представлено на диаграмме ниже.
Очевидно, что основное взаимодействие осуществляется между компонентами Integration Middleware (части Интеграционной Платформы) и Connector (в области ответственности бизнес-приложения) посредством паттерна Publisher-Subscriber, реализованного на базе механизма обмена сообщениями. Данные, которыми обмениваются коннекторы, передаются в теле сообщений. Дополнительно, для случаев, когда понадобится прочитать достаточно большие данные, которые не сможет эффективно обработать брокер обмена сообщениями, коннектор обратится за данными напрямую через интерфейс Storage, а в теле сообщения будет передан критерий выбора.

Также на диаграмме видно, что Интеграционная Платформа и бизнес-приложение обмениваются данными в формате Канонической Модели, использующейся как единый общий язык.



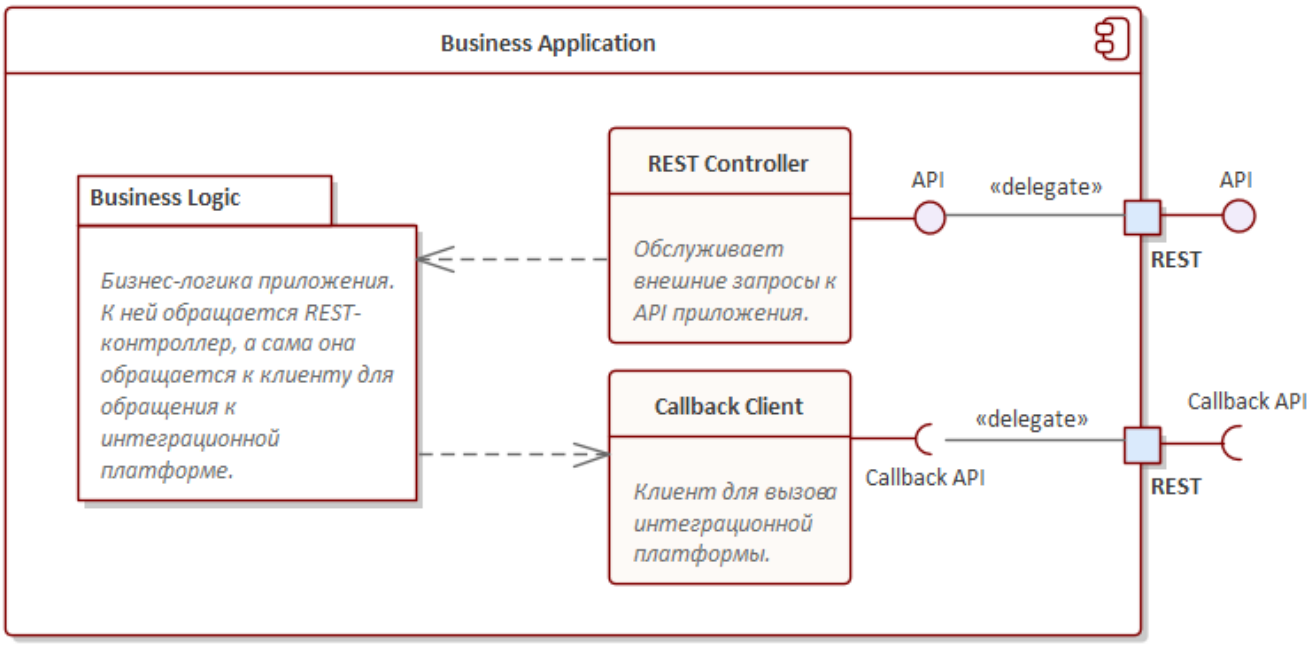
Далее будет рассмотрено внутреннее устройство наиболее важных с архитектурной точки зрения компонентов.

Взаимодействие между коннекторм и бизнес-приложением:



Само бизнес-приложение внутри может взаимодействовать со своим коннектором следующим образом:

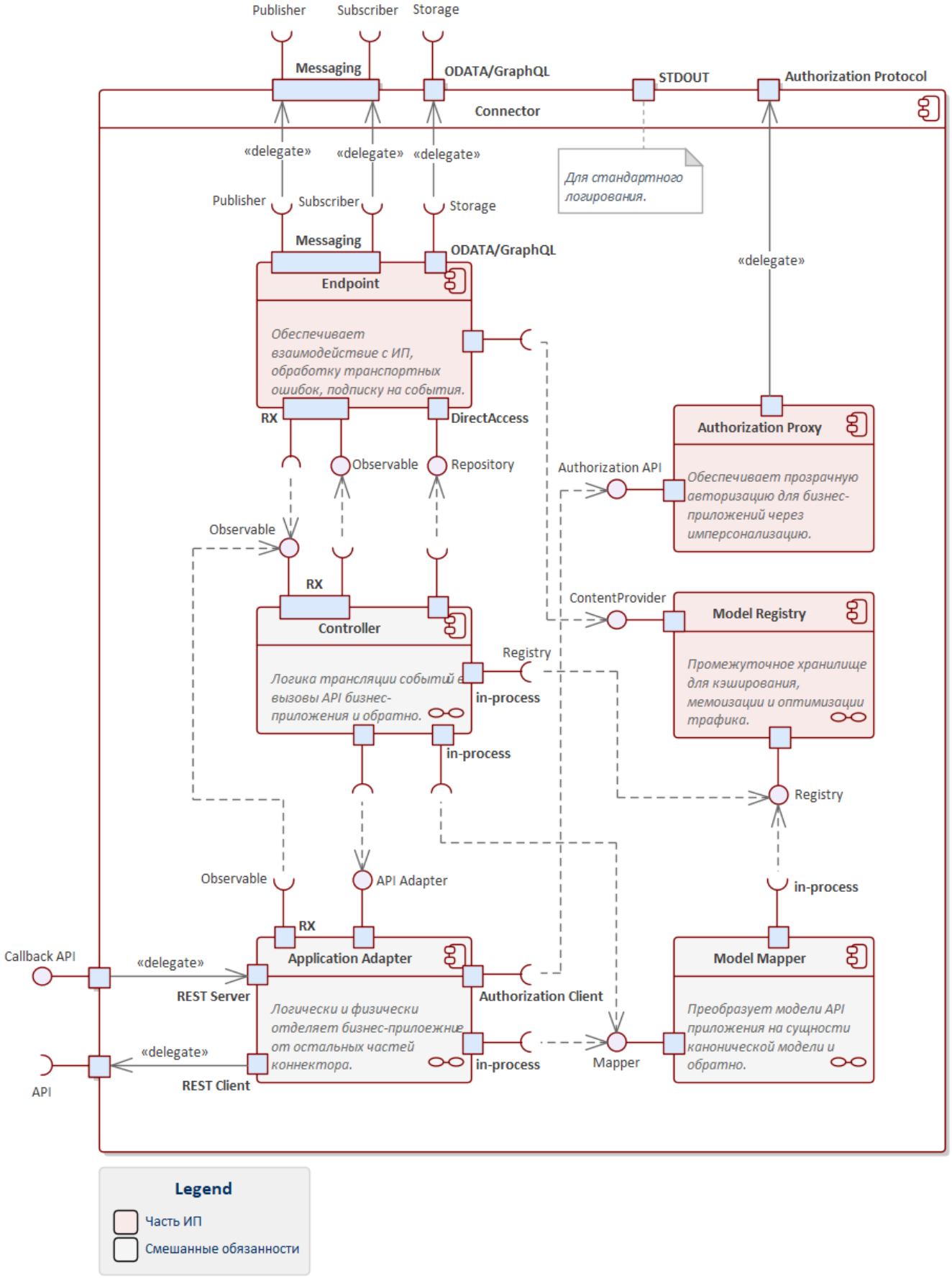
- Через REST Controller, чтобы обслуживать внешние запросы, поступающие от коннектора (а значит, и от других приложений посредством Интеграционной Платформы)
- Через Callback Client, чтобы самому инициировать обращение к другим приложениям (через коннектор и Интеграционную Платформу)



Далее рассмотрено внутреннее устройство самого коннектора.
Коннектор состоит из нескольких основных компонентов, обеспечивающих его основную функцию: служить адаптером между бизнес-приложением (имеющим свою специфику, свою модель предметной области, свои контракты, жизненный цикл, особенности развертывания и т. п.) и Интеграционной Платформой.

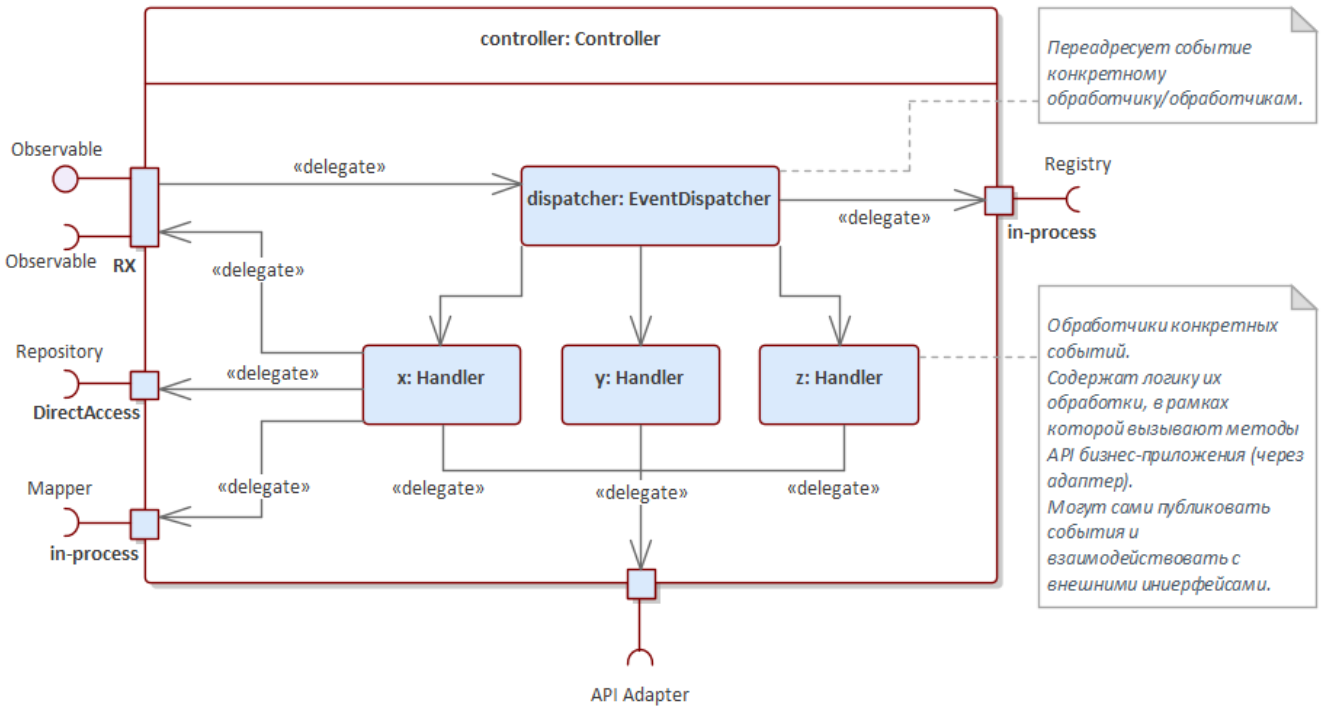
Обязанности основных компонентов:

- Application Adapter - логическая и физическая изоляция особенностей и специфики бизнес-приложения, от остальной логики коннектора
- Controller - содержит логику обработки входящих и исходящих событий
- Model Mapper - преобразования между канонической моделью и моделью API бизнес-приложения
- Authorization Proxy - авторизация и аутентификация бизнес-приложений
- Endpoint - транспорт для обмена сообщениями с брокером по сети


Publisher
Subscriber
Storage
Messaging
ODATA/GraphQL
STDOUT
Authorization Protocol
Connector
«delegate»
«delegate»
«delegate»
Для стандартного логирования.
Publisher
Subscriber
Storage
Messaging
ODATA/GraphQL
«delegate»
Endpoint
Обеспечивает взаимодействие с ИП, обработку транспортных ошибок, подписку на события.
RX
DirectAccess
Observable
Repository
Authorization Proxy
Authorization API
Обеспечивает прозрачную авторизацию для бизнес-приложений через имперсонализацию.
Observable
RX
ContentProvider
Model Registry
Controller
Registry
Логика трансляции событий в вызовы API бизнес-приложения и обратно.
in-process
Промежуточное хранилище для кэширования, мемоизации и оптимизации трафика.
in-process
Registry
Observable
API Adapter
in-process
RX
Callback API
«delegate»
Application Adapter
Model Mapper
REST Server
Authorization Client
Логически и физически отделяет бизнес-прилоежние от остальных частей коннектора.
«delegate»
in-process
Mapper
Преобразует модели API приложения на сущности канонической модели и обратно.
API
REST Client
Legend
Часть ИП
Смешанные обязанности

Работа коннектора в общем случае начинается с поступления в компонент Endpoint нового сообщения через интерфейс Subscriber. Сообщение будет десериализовано, обеспечена идемпотентность на уровне получателя (если включен такой режим) и преобразовано в событие, которое затем будет отправлено через реактивный API в компонент Controller.

Компонент Controller, получив новое событие, внутри выберет для этого события нужный обработчик (Handler) и делегирует ему выполнение нужного сценария. Долгоживущие бизнес-сценарии могут быть реализованы посредством паттерна Saga, хотя в общем случае этого стоит избегать по соображениями масштабирования.

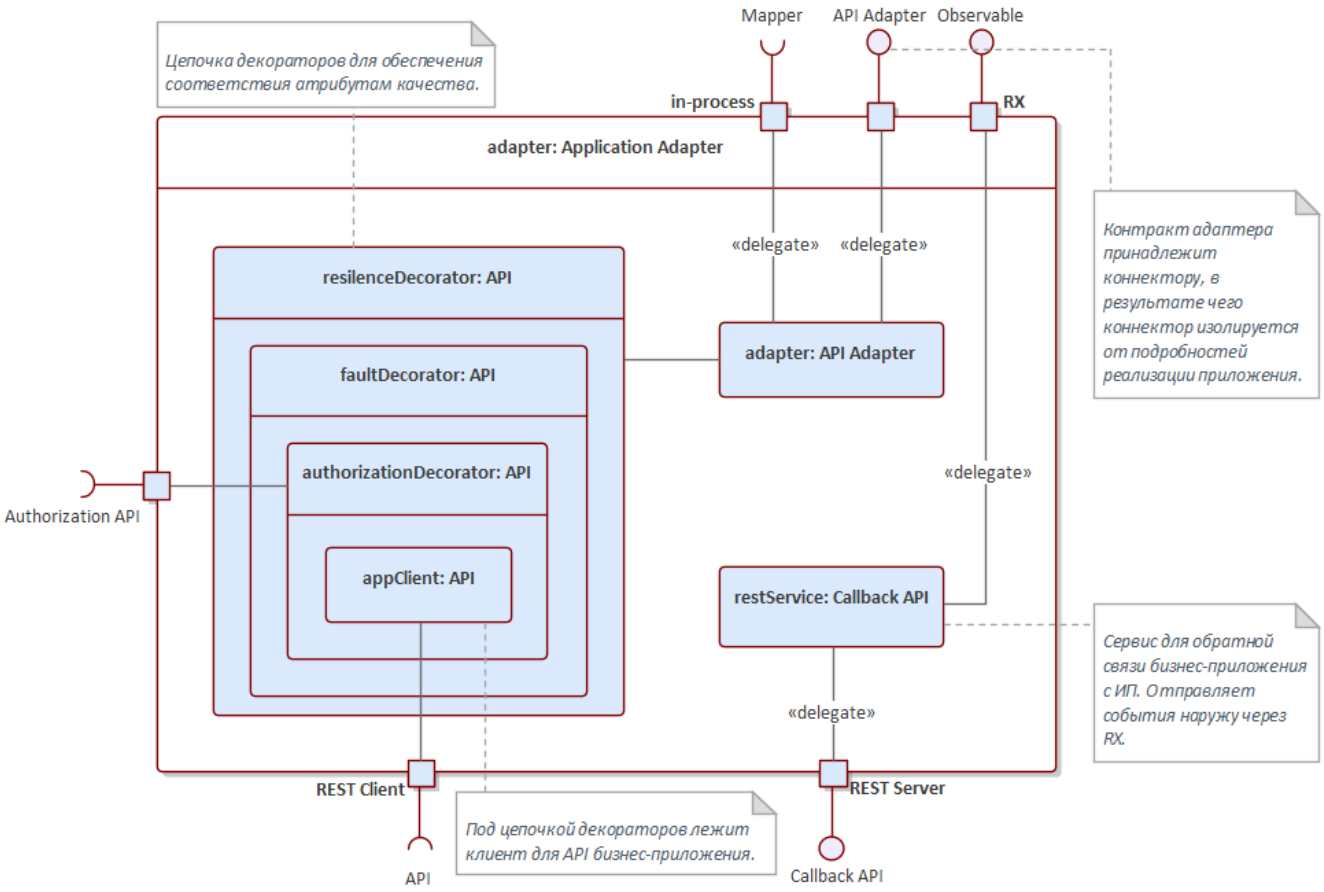


Обработчики событий компонента Controller обычным образом обращаются к бизнес-приложению через его адаптер, компонент Application Adapter.

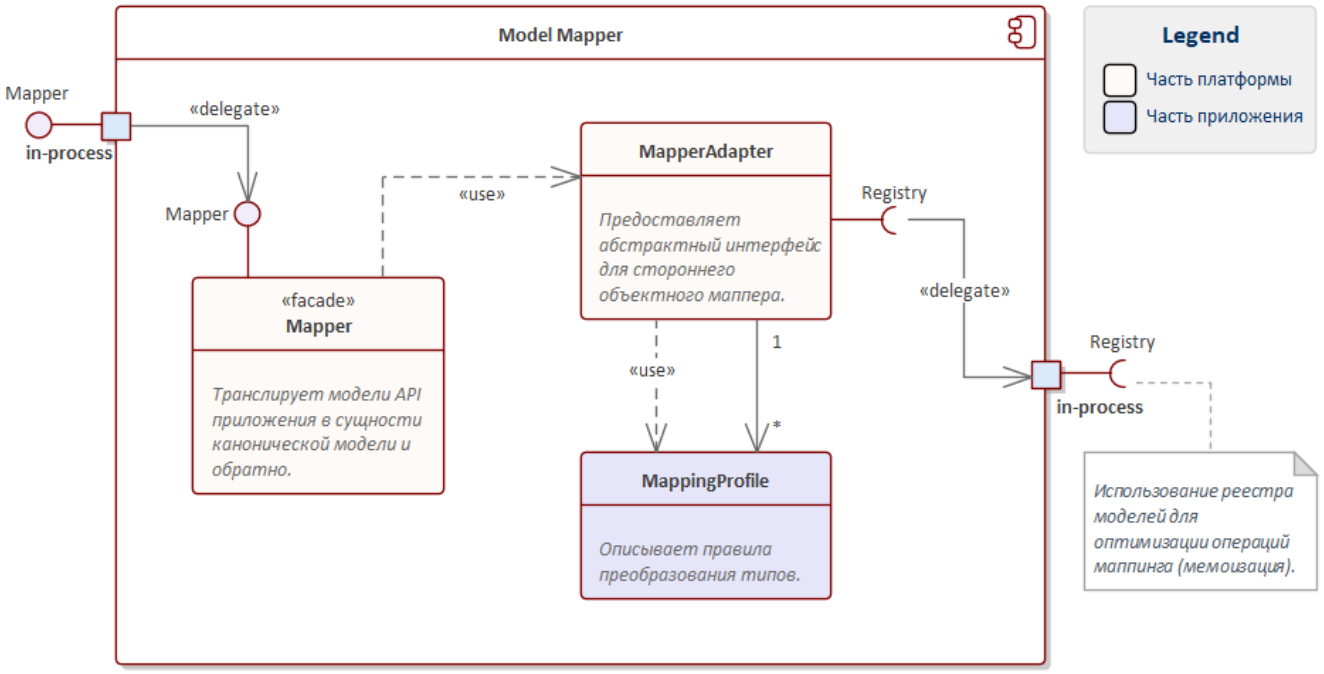
Ниже представлено внутреннее устройство адаптера приложения.

Основной паттерн, примененный при реализации этого адаптера - цепочка декораторов, построенных вокруг клиента бизнес-приложения. Каждый такой адаптер обеспечивает удовлетворение одного из атрибутов качества:

- Авторизацию (через делегирование Authorization Proxy посредством интерфейса Authorization API).
- Обработка ошибок и сбоев (журналирование ошибок, восстановление после исключений, т.п).
- Восстановление связи с бизнес-приложением (в случае с временной недоступностью).

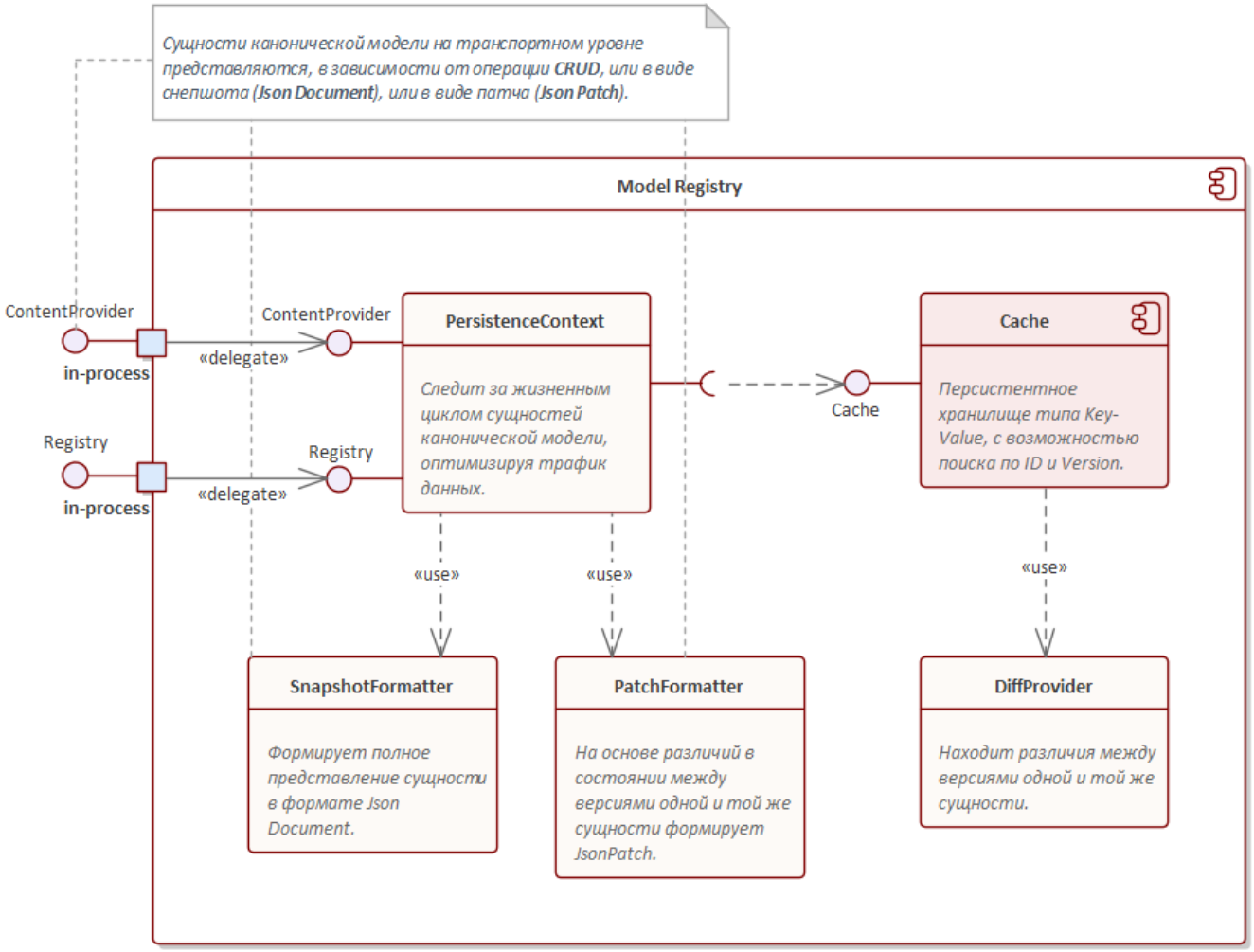


Когда Controller обращается к Application Adapter, то он передает данные в формате Канонической Модели, но бизнес-приложение, разумеется, ничего о канонической модели не знает. Для того, чтобы общение между ними было естественным, передаваемые данные должны быть сконвертированы из формата Канонической Модели в модель API бизнес-приложения и в обратную сторону. Эти функции выполняет компонент Model Mapper.



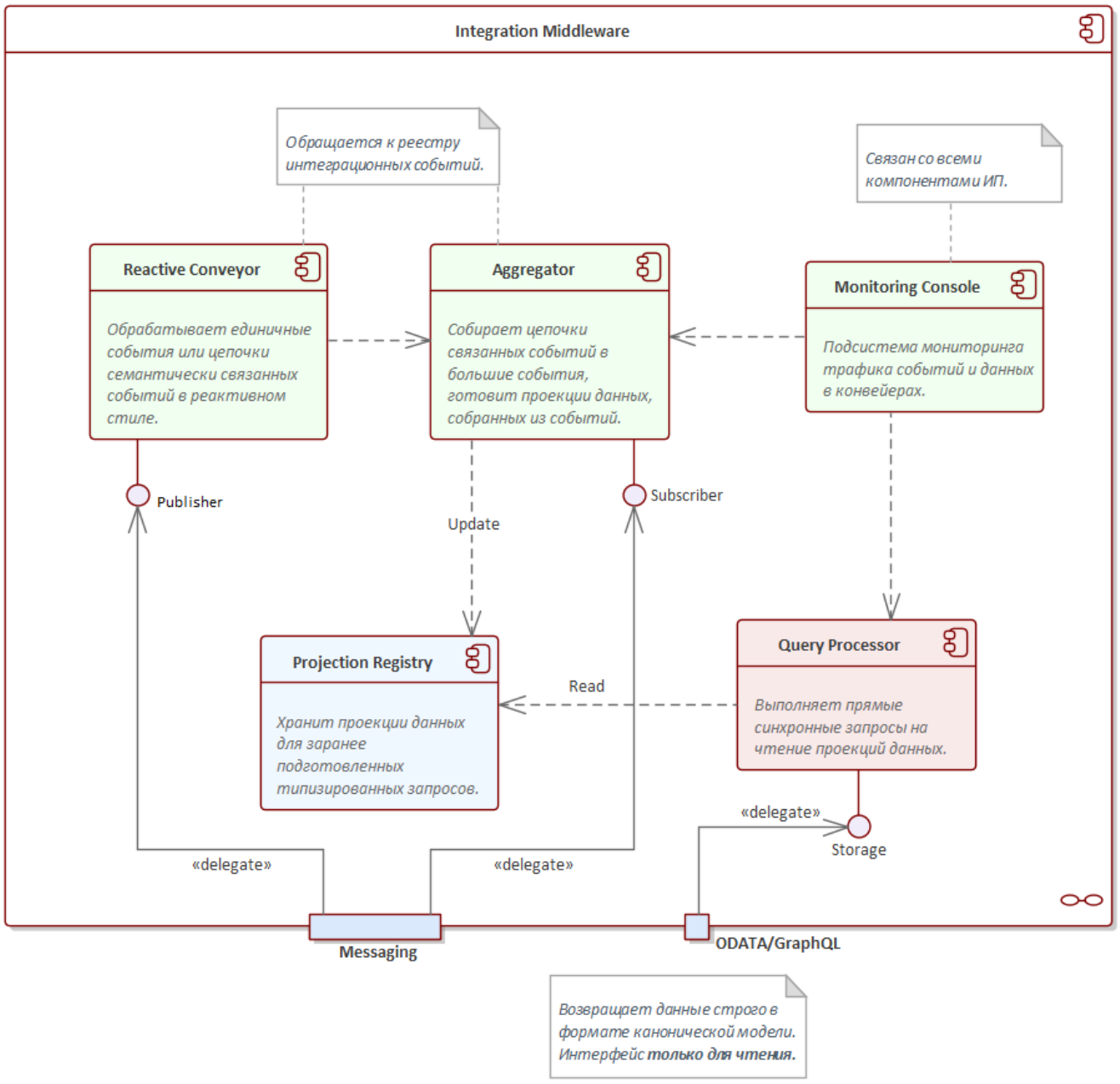
Как было указано ранее, Endpoint в основном передает данные в теле сообщений. Для операций, которые создают новые экземпляры сущностей Канонической Модели, к телу сообщения будут добавлены представления этих сущностей в формате Json Document. Однако, если операция меняла уже существующие сущности, то имеет смысл передавать эти изменения как "дельту" в виде Json Patch.

Для автоматического формирования этих представлений данных (Json Doc и Json Patch) предназначен компонент Model Registry.
Компонент ведет историю изменений сущностей, и на основе анализа различий между версиями одной и той же сущности (жизненный цикл ограничен интеграционным сценарием) определяет, какое представление сгенерировать.

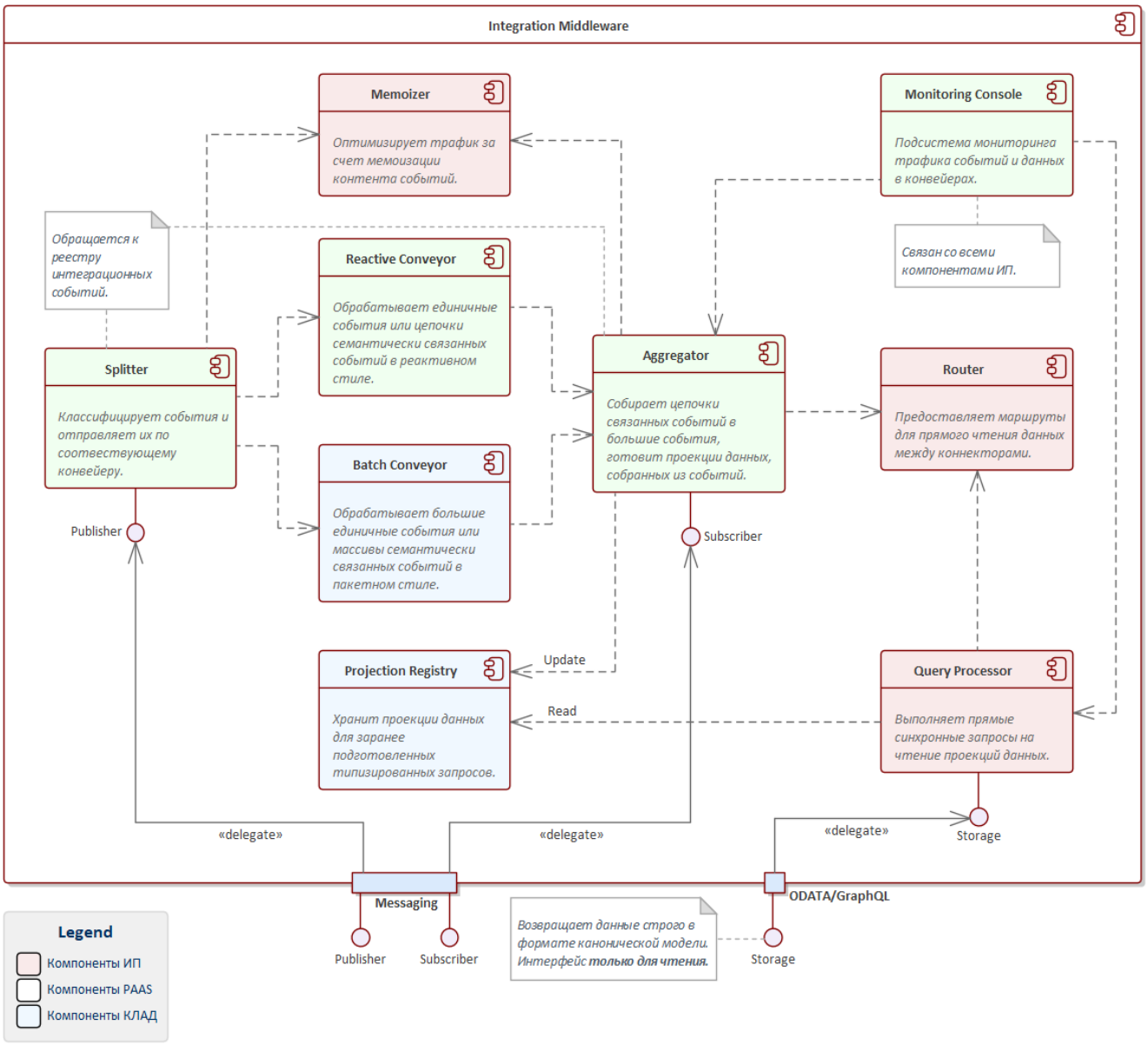


После отправки сообщения компонентом Endpoint оно попадает по сетевому протоколу в Integration Middleware.
Его внутреннее стройство представлено на диаграмме ниже.

Через интерфейс Publisher внешнее сообщение попадает в конвейер, где в зависимости от содержимого (и служебного заголовка) дополнительно обрабатывается, а затем перенаправляется подписчикам через интерфейс Subscriber. Некоторые сообщения могут послужить составной частью проекций данных, асинхронно формируемых службой Query Processor из внешних источников (БДПИ и Data Market).



Вероятно, в дальнейшем компонент Integration Middleware будет дополнен дополнительными компонентами для оптимизации работы в разных аспектах.



Слои

Для каждого уровня включить подраздел с его описанием, перечнем подсистем, диаграмму компонентов.

Правила декомпозиции и явного разделения на слои отдельных компонентов будет определено в процессе проектирования.

Представление данных (опционально)

Описание схемы хранения данных. Раздел не является обязательным и наполняется в случае, если хранение данных нетривиально и значимо с точки зрения общей архитектуры.

В данной фазе не требуется.

Размер и Производительность

Описание основных характеристик системы в плане объема данных и требований к производительности, особенно влияющих на архитектуру.

Все наиболее важные и архитектурно-значимые требования перечислены в документе Supplementary Specification.

Качество

Описание того, как архитектура системы отражает влияние нефункциональных требований и атрибутов качества, таких, как безопасность, масштабируемость, надежность, повторное использование и т.п. TODO