МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра Прикладная математика и информатика
(наименование)

09.04.03 Прикладная информатика
(код и наименование направления подготовки)

Информационные системы и технологии корпоративного управления
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Методы интеграции программного обеспечения класса middleware на примере ПАО «Русский продукт»

Студент В.В. Горбунов
(И.О. Фамилия) (личная подпись)

Научный руководитель д.т.н., доцент, С.В. Мкртычев
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

ОГЛАВЛЕНИЕ

Введение

Если век прошлый для корпоративных информационных систем можно считать началом создания локальных систем для сбора, ввода и обработки информации, то век нынешний дает старт глобальной интеграции и автоматизации информационного обмена между такими системами.

Существование современного субъекта экономической деятельности не мыслимо без средств интеграции. Взаимодействие с внутренними системами, с информационными сервисами контрагентов, с государственными службами становится полноправной частью внутренних и внешних бизнес-процессов.

Для решения вопросов, касающихся взаимодействия таких информационных систем, предназначено middleware - связующее программное обеспечение. В корпоративной среде крупных предприятий наиболее часто используется связующее программное обеспечение класса сервисных шин предприятия (Enterprise Service Bus, ESB). Этот программный комплекс позволяет не только решить внутренние вопросы с традиционной для больших предприятий сервис-ориентированной архитектурой (Service-Oriented Architecture, SOA), но и создать уникальный интеграционный механизм, позволяющий в сжатые сроки создавать интеграционные решения любых масштабов.

На предприятии ПАО “РУССКИЙ ПРОДУКТ” в 2018 году было принято решение об интеграции корпоративной информационной системы средствами единого продукта - сервисной шины предприятия. Для решения этого вопроса появилась необходимость провести несколько исследований, такие как выбор подходящего по всем параметрам решения, внедрение, интеграция в него имеющихся продуктов и разработка новых продуктов на его основе.

Актуальность темы данного исследования заключается в том, что предприятию требуется интегрировать в корпоративную информационную систему продукт класса сервисных шин предприятия для приведения

внутреннего информационного обмена к единому виду и обеспечить возможность создания быстрых интеграционных решений.

Объектом исследования является связующее программное обеспечение.

Предметом исследования являются методы интеграции связующего программного обеспечения на примере ПАО «РУССКИЙ ПРОДУКТ».

Целью работы является интеграция связующего программного обеспечения в корпоративную информационную систему ПАО «РУССКИЙ ПРОДУКТ».

Для достижения цели исследования необходимо выполнить следующие **задачи**:

1. Провести исследование темы связующего программного обеспечения.
2. Провести анализ существующей корпоративной информационной системы предприятия и сформулировать требования к внедряемому решению.
3. Изучить существующие решения класса сервисных шин предприятия и выбрать наилучший вариант для интеграции.
4. Интегрировать сервисную шину предприятия в корпоративную информационную систему.
5. Провести приемочные испытаний в формате автоматизированного тестирования.

Гипотезой этого исследования является предположение о том, что интеграция связующего программного обеспечения позволит использовать его в качестве полноценного транспорта, что повысит архитектурные качества и интеграционные возможности информационной системы предприятия.

Методы исследования. В работе использованы эмпирические и математические методы исследования: наблюдение, сравнение, программирование, тестирование и эксперимент.

Новизна исследования заключается в разработке метода интеграции связующего программного обеспечения.

Практическая значимость исследования заключается в возможности практического применения разработанного метода для интеграции связующего программного обеспечения на средних и крупных предприятиях.

Теоретической основой исследования являются книжные издания в основном зарубежных издателей, отечественные и зарубежные научные статьи.

Основные этапы исследования проводились в период с 2018 по 2020 год в несколько этапов.

На первом этапе устанавливалась разработанность и перспективность темы исследования. Были проанализированы результаты ранее проведенных исследований по выбранной теме. Выявлены нерешенные проблемы, обосновывалась актуальность темы, определялся объект и предмет исследования, составлен план исследования и система методов исследования. Были сформулированы цели исследования и гипотезы, на основе которых была поставлена задача.

На втором этапе было выполнено исследование корпоративной информационной системы предприятия, анализ существующих решений на соответствие предъявляемым требованиям и принято решение о разработке собственного программного продукта.

На третьем этапе была выполнена разработка связующего программного обеспечения, интеграция его в корпоративную информационную систему предприятия, создание на его базе нового интеграционного решения и произведена оценка с точки зрения практического использования результатов.

На четвертом этапе была проведена апробация разработанного программного продукта путем проведения автоматизированного тестирования.

На защиту выносятся:

1. Связующее программное обеспечение, интегрированное в КИС ПАО “РУССКИЙ ПРОДУКТ”

2. Результаты оценки эффективности интеграции связующего программного обеспечения в КИС ПАО “РУССКИЙ ПРОДУКТ”

Эта работа состоит из введения, четырех глав, заключения и списка используемой литературы и приложений.

В первой главе рассматриваются теоретические аспекты вопроса. В ней рассмотрено связующее программное обеспечение, сервисные шины предприятия, сервис-ориентированная архитектура. Рассмотрены проблемы интеграции сервисных шин предприятия на базе корпоративной информационной системы. Выполнен анализ рынка существующих решений и принято решении о разработке собственного продукта.

Во второй главе исследована корпоративная информационная система предприятия, составлены требования к внедряемому программному обеспечению, разработаны модели программного продукта, выбраны инструменты для разработки.

В третьей главе выполнена разработка сервисной шины предприятия и ее интеграция в корпоративную информационную систему предприятия.

В четвертой главе выполнена апробация результатов интеграции с использованием механизмов автоматизированного тестирования. По результатам тестирования сделаны выводы о качестве интеграции и ее эффективности.

По теме диссертации опубликованы две статьи:

Горбунов В.В. Подход к интеграции сервисной шины предприятия // Студенческий: электрон. научн. журн. 2020. № 40(126). URL: https://sibac.info/journal/student/126/194926 (дата обращения: 21.12.2020).

Горбунов В.В. Расширенные способы взаимодействия с сервисной шиной предприятия // Журнал “Инновации. Наука. Образование”. 2020. № 23 (Декабрь 2020 года). URL: https://innovjourn.ru/nomer/23-nomer (дата обращения: 21.12.2020).

Глава 1 Связующее программное обеспечение и особенности его интеграции

1.1 Современное состояние проблемы связующего программного обеспечения

Современный уровень проработанности вопроса чрезвычайно низок. В литературе и открытых источниках очень редко появляются публикации, связанные с сервисной шиной предприятия и целиком ей посвященные. Как правило, это издания, рассматривающие сервис-ориентированную архитектуру, сопроводительная литература, издания, вскользь касающиеся вопроса, которые рассматривают сервисную шину предприятия как платформу для разработки, либо небольшие научные статьи. Причем в отечественной литературе теоретический вопрос не проработан совсем. Литературные источники, как-либо затрагивающие данный вопрос, доступны только на иностранном языке либо в переводах.

Это связано с тем, что такое программное обеспечение относится к разряду внутрикорпоративного программного обеспечения, исследования на эту тему крайне редко публикуются и зачастую ограничены грифами, ограничивающими область использования документов. Такие документы, помимо своей обычной ценности, как высокотехнологический наработанный материал, имеют особенности, которые раскрывают подробности внутренней архитектуры предприятия, выполняющего исследование.

Из книжных изданий при создании исследования наиболее активно использовались несколько книг, в основном это были зарубежные книги.

“Service-Oriented Architecture. Analysis and Design for Services and Microservices” большая работа Thomas Erl по теме сервис-ориентированных архитектур. Книга содержит большое количество полезной информации от теоретических основ до практических применений, разработки веб-сервисов, микро-сервисов. Рассматриваются все вопросы вплоть до версионности.

В “SOA in Practice” N. M. Josuttis дается большой теоретический обзор сервис-ориентированной технологии и небольшой экскурс в историю вопроса. В ней так же рассматриваются вопросы, касающиеся сервисной шины предприятия.

В издании “Open Source ESBs in Action” T. Rademakers и J. Dirksen рассказывается об интеграции таких продуктов как Mule ESB и ServiceMix. Книга практически не содержит теоретических основ, больше обращая внимание на реальные примеры использования.

“JBoss ESB Beginner's Guide” – руководство по использованию продукта JBoss ESB, помимо своего основного назначения, содержит теоретический раздел по темам сервисных шин предприятия и сервис-ориентированной архитектуры.

Книга “Designing Web APIs”, созданная коллективом Brenda Jin, Saurabh Sahni, and Amir Shevat, рассказывает об основах проектирования Web-API. Эта книга содержит много полезной практической информации об основах сетевых сервисов и правильных подходах к их проектированию.

В процессе исследования было изучено некоторое количество научных публикаций по теме исследования.

В статье “Модификация адаптеров подключения к единой сервисной шине (ESB) для повышения эффективности и надёжности корпоративной информационной системы вуза” С. П. Семенова и Я. Б. Татаринцева рассматриваются вопросы повышения корректности передаваемых данных с помощью разработки дополнительных адаптеров.

Под авторством Ю. Н. Зацаринной, Р. Р. Рахматуллиной и Г. И. Ризвановой была опубликована статья “Информационная транспортная шина предприятий (ESB) в распределенных энергетических компаниях”, в которой рассматривались теоретические вопросы, касающиеся сервис-ориентированной технологии, сервисной шины предприятия и практической пользы от ее внедрения.

В сборнике трудов “Безопасные информационные технологии”, седьмой всероссийской научно-технической конференции, выпущенной под редакцией В.А. Матвеева, МГТУ им. Н.Э.Баумана, была опубликована статья “NTLM-авторизация при интеграции данных через ESB”, которая помимо теоретических аспектов сервисной шины предприятия описывает один важный момент при ее использовании – доменного механизма аутентификации.

1.2 Принципы построения и функциональность связующего программного обеспечения

1.2.1 Связующее программное обеспечение

Связующее или промежуточное программное обеспечение, англ. middleware — это широкий и многогранный термин, обозначающий специальное, технологическое программное обеспечение, используемое для обеспечения взаимодействия между различными программными продуктами, комплексами, платформами, сервисами и информационными системами.

Сам термин появился довольно давно, в 1968 году на конференции научного комитета НАТО, в докладе Александра Агапеева мл., была описана архитектурная модель развертывания программного обеспечения (рисунок 1).

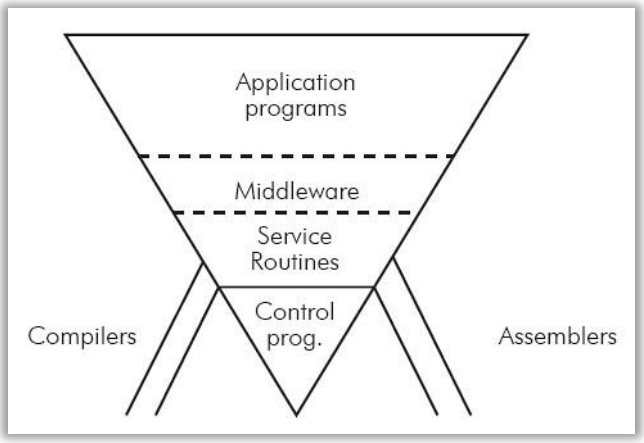

Рисунок 1 – Инвертированная пирамида Агапеева

На тот момент позиция Агапеева была критической к такому подходу, он отмечал уязвимость таких систем, поскольку они трудны в поддержке, тяжело масштабируемы и имеют высокую стоимость сопровождения [46, с. 14].

В настоящее время эта область активно развивается, за прошедший, более чем полувековой период, создано большое количество различного программного обеспечения, которое можно классифицировать как связующее, и оно занимает довольно значительную часть рынка. В частности, существует программное обеспечение для взаимодействия на различных уровнях, в том числе с операционной системой, системного и сервисного, сетевого, веб-сервисов и баз данных. Принципиальной особенностью такого программного обеспечения всегда является организация взаимодействия между различными информационными системами в виде приложения посредника.

По оценке издания Gartner [68], проведенного им в 2018 году, объем специализировано программного обеспечения, которого можно отнести к связующему, можно оценить в 28,4 миллиардов долларов в 2017 году. Эта цифра обозначает прирост в 12 процентов по сравнению с 2016 годом.

Такое активное развитие связующего программного обеспечения можно объяснить несколькими причинами.

С одной стороны, появилась техническая основа для разработки таких решений – необходимый набор как программных инструментов, позволяющих реализовать такой подход, так и набор стандартов, которые дали возможность унифицировать информационный обмен.

С другой стороны, рынок связующего программного обеспечения поддерживается спросом потребителей таких систем, поскольку все больше предприятий активно развивают собственные информационные сервисы и для их разработки используют такое программное обеспечение или использует его для приведения взаимодействия программных продуктов к единому виду, в едином архитектурном стиле.

Основным типом связующего программного обеспечения является сервисная шина предприятия. Это программный продукт, на данный момент, является эволюционной вершиной всего того, что было произведено в области связующего программного обеспечения.

1.2.2 Сервисная шина предприятия

Сервисные шины предприятия, как полноценный программный продукт, появились в середине 2000-ых годов как развитие брокеров сообщений [60]. Термин был введен компанией Sonic Software, разработчиком брокера сообщений SonicMQ.

Явление сервисной шины предприятия не нашло своего отображения в отечественных, государственных стандартах. В соответствии с ГОСТ Р ИСО/МЭК ТО 12182-2002 [1] его можно классифицировать как программное средство.

Используя эту методику, программные продукты, относящиеся к типу сервисных шин предприятия, можно классифицировать как программное обеспечение с функцией контролируемого и централизованного обмена формализованными сообщениями, с режимом эксплуатации в виде автоматизированной обработки данных, в режиме реального времени и прикладной областью корпоративных информационных систем.

Авторы JBoss ESB Beginner's Guide [42] описывают сервисную шину предприятия как супервизора в архитектурном слое всего предприятия, обеспечивающего асинхронную связь между сервисами.

Рик Робинсон, доктор философии в области физики и архитектор информационных систем компании IBM, определяет минимальный набор функций сервисной шины предприятия. Робинсон считает, что сервисная шина предприятия должна предоставлять функции маршрутизации и адресации и их администрирование, одну из форм обмена сообщениями

(запрос/ответ, публикация/подписка и т.д.) и поддержку как минимум одного, общедоступного транспортного протокола. [64].

Таким образом сервисную шину предприятия можно определить как программное обеспечение, имеющее в своем составе три основных программных элемента:

1. Транспортное средство для доступа к сервисной шине предприятия.
2. Исполнительный модуль управления маршрутами сообщений.
3. Брокер очереди сообщений.

Такая конфигурация обеспечит выполнение предъявляемых к ней требований по функциональности. Транспортное средство получает данные от клиентов сервисной шины предприятия и передает их на исполнительный модуль. В свою очередь исполнительный модуль ставит сообщение в очереди брокера сообщений - сообщения передаются получателям для обработки.

Средства администрирования и управления в этот список не входят, поскольку могут быть сторонним программным обеспечением в том случае, если сервисная шина предприятия управляется через веб-браузер, статическую конфигурацию в виде конфигурационных файлов или через управляющие таблицы базы данных.

Поскольку основные принципы построения сервис-ориентированных архитектур соответствуют характеристикам систем, развернутых на базе сервисных шин предприятия, то программные архитектуры, разработанные на базе сервисных шин предприятия, являются сервис-ориентированными архитектурами.

1.2.3 Сервис-ориентированная архитектура

Сервис-ориентированная архитектура является особым, модульным подходом к разработке комплексов программного обеспечения. Элементы такого комплекса при взаимодействии между собой вместо классических

способов взаимодействия используют открытые стандарты, такие, как сетевые веб-сервисы, как правило развернутые на основе протокола SOAP. Однако каких-либо ограничений на транспорт в этой архитектуре нет [64], в рамках нее можно использовать любой транспорт, который поддерживают оба сервиса, к примеру стандартный файловый обмен.

Сам термин SOA ввел Александр Пасик, бывший аналитик Gartner, для обозначения класса программного обеспечения среднего уровня, которое он преподавал в 1994 году. Пасик работал до того, как были изобретены XML или веб-службы, но основные принципы SOA с тех пор не изменились [41, с. 10].

Идея такого программного взаимодействия возникла как ответ на сложности в разработке корпоративного программного обеспечения. Необходимость взаимодействия с другими членами информационного пространства диктовала правило – необходимость иметь общий набор интерфейсов для взаимодействия, причем чаще всего он ограничивался одной из платформ для разработки программного обеспечения. В сервис-ориентированных системах для того, чтобы появилась возможность взаимодействия между гетерогенными системами, используются стандартизированные протоколы обмена.

Суть такого архитектурного взаимодействия заключается в том, что используемые программные продукты и сервисы становятся полностью независимыми и автономными субъектами информационного пространства, к которому они принадлежат. С помощью собственных схем объектов они объявляют правила информационного обмена с другими членами единого информационного пространства.

Такая независимость от транспорта и стандартизированный способ взаимодействия позволяет значительно облегчить разработку новых членов информационного пространства, поскольку их можно реализовывать на любых платформах и любых языках. Общим языком взаимодействия являются формализованные средства обмена, такие как язык XML и формат JSON.

По своей сути сервис-ориентированные системы являются системами децентрализованными, с одной стороны, это обеспечивает устойчивость системы – без единой точки доступа система значительно устойчивее, но в этом заключается и их значительный недостаток. Информационный обмен происходит без участия единого центра, отвечающего за формат информационного обмена. Создание информационной системы на основе сервисной шины предприятия исправляет этот недостаток, используя единую точку входа и единый набор схем объектов для использования в информационном обмене, позволяя работать на разных программно-аппаратных платформах и между границами организаций [11]. Формально это обмен стабильности на качество системы, но если проблема устойчивости может быть относительно легко решена, то проблема качества данных, проходящих по каналу обмена на больших системах, может оказаться нерешаемой проблемой.

Сервис-ориентированные системы могут быть развернуты без сервисной шины предприятия, используя различные алгоритмы для транспорта и для организации очередей сообщений, однако, любая система, построенная на основе сервисной шины предприятия, является сервис-ориентированной системой. При этом сервисная шина предприятия расширит возможности такой информационной системы функциями управляемого и маршрутизируемого транспорта и очередью обработки сообщений.

1.2.4 Брокеры сообщений для сервисных шин предприятия

Брокеры сообщений — это специальные, системные приложения по обработке сообщений, неотделимая и одна из основных частей сервисной шины предприятия. Они относятся к подклассу связующего программного обеспечения, ориентированного на обработку сообщений. В сервисной шине предприятия, после того как сообщение первично получено и обработано, оно отправляется именно в брокер сообщений.

Связь серверных шин предприятия с брокерами сообщений это эволюционный процесс, имевший в своей основе повышенные требования к связующему программному обеспечению в начале 2000-ых годов. Брокеры сообщений не могли обеспечить необходимую функциональность по вопросам, касающимся сетевых коммуникаций и полноценного администрирования потоков сообщений. Именно этим функционалом расширила сервисная шина предприятия возможности брокера сообщений и такой ход развития был вполне закономерен.

Первоочередной задачей брокера сообщений является посредничество при получении и доставке сообщений от системы отправителя к системе получателя. При этом брокер должен формировать из сообщений упорядоченную очередь для того, чтобы получатель мог обрабатывать их последовательно.

При работе брокеры сообщений должны проверять сообщения на наличие ошибок, и, в случае их наличия, отказывать отправителю в постановке сообщения в очередь.

Функционально брокеры сообщений могут поддерживать возможности по маршрутизации сообщений, но в отличие от возможностей сервисной шины предприятия, маршрутизация брокера сообщений ограничена только ее подписчиками.

Подписчики брокера сообщений – внешние приложения, которые подключены к одной или нескольким очередям сообщений. Они получают сообщения от брокера, когда он ставит его в очередь.

Одним из положительных свойств брокера сообщений является возможность перманентного хранения сообщений в локальных и сетевых базах данных – это позволяет восстановить очередь при возникновении исключительной ситуации в самом брокере, серьёзной ошибке или перезагрузке сервера.

Для информационного обмена с брокером сообщений, стандартом де-факто стал стандарт AMPQ Advanced Message Queuing Protocol - расширенный протокол очереди сообщений [34, с. 47].

1.3 Основные проблемы интеграции сервисных шин

К основным проблемам, касающимся интеграции шин предприятия, можно отнести следующие вопросы – высокая нагрузка при работе с формализованными данными, интеграция программного обеспечения на новый транспорт шины, единая точка входа в шину.

Высокие накладные расходы на работу с формализованными данными являются серьезной проблемой, влияющей на производительность. Для того, чтобы перевести строковой массив формата XML и JSON в его объектный вид, т.н. процедура десериализации, или объект в строковой формат, т.н. процедура сериализации, требуется значительная производительность.

Если процедура сериализации может быть не особо требовательна к ресурсам, то процесс десериализации всегда был уязвимым местом. Проблема заключается в обработке строковых значений. Современные процессоры не обладают функциями, позволяющими быстро обрабатывать формализованные форматы, как это, к примеру, можно сделать с работой по вычислениям с помощью команд сопроцессора или с ускорением криптографических функций с помощью аппаратного расширения AES. Вся обработка формализованных строковых объектов сводится к многократному чтению и поиску шаблонов по буферу в памяти с использованием команд процессора, наследованных из 90-ых годов, если используется 32-х битная архитектура, и начала нулевых годов, если используется 64-х битная [16]. Эта проблема особо ярко проявляется в высоконагруженных информационных системах, где преобразование объектов, которыми обмениваются субъекты информационной системы, может происходить тысячи раз в секунду. Недостаточно оптимизированный алгоритм обработки не сможет обеспечить

нужную производительность входящего потока обрабатываемых данных. Для решения той проблемы необходимо предварительно провести исследования, позволяющие определить текущий объем передаваемых данных, которые планируется завести в сервисную шину предприятия. Перед внедрением сервисной шины предприятия необходимо провести синтетические тесты, которые позволят подтвердить возможность обмена информацией в нужном объеме. Если объем обрабатываемых данных будет меньше, чем объем обрабатываемых сервисной шиной предприятия, то это позволит гарантировать обработку нужного объема данных.

Интеграция программного обеспечения является не менее актуальной проблемой. В идеальной ситуации все программное обеспечение в информационной системе, в которой установлена сервисная шина предприятия, должно разворачиваться на чистой системе – новом архитектурном решении, в котором проработаны все типы объектов. Такой подход позволит избежать двойственности в отношении к данным, поскольку любой объект имеет только одно представление. Но в реальности такие ситуации предельно редки, к тому моменту, когда предприятия задумываются об интеграции, в их корпоративной информационной системе находятся десятки и сотни различных программных средств в разной стадии жизненного цикла.

Этот вопрос можно решить двумя различными путями - через собственные средства программного обеспечения, при наличии в нем встроенной возможности интеграции с шиной, либо возможностью внести модификации в это обеспечение. Поскольку первый вариант на практике встречается крайне редко, то реальным фактором, определяющим возможность полноценного внедрения, остается только наличие исходных текстов программного средства.

Единая точка входа в шину, являясь несомненным плюсом сервисной шины предприятия в плане централизации, является также и ее уязвимым местом. В случае отказа сервиса или перегрузки точки работа шины

предприятия станет невозможна. Для решения этой проблемы возможны два варианты работы единой точки входа – спаренный режим из нескольких экземпляров или режим работы в виде кластера.

1.4 Программное обеспечение класса сервисных шин предприятия

Для обзора продуктов были отобраны несколько основных продуктов, которые присутствуют на рынке и занимают значительную его часть. Отбор осуществлялся с помощью поисковых служб, в ответах, которые они выдали по запросу “ESB” и “Сервисная шина предприятия”. Наиболее популярными зарубежными продуктами оказались следующие решения: IBM Integration Bus, Microsoft Biztalk ESB, Oracle ESB/SOA и Mule ESB.

На рынке отечественных продуктов наиболее популярными оказались следующие решения: Datareon ESB и Mediator ESB. В дополнение к ним было добавлено решение ESB Галактика, поскольку вся учётная система предприятия состоит из продуктов корпорации “Галактика”, а также вариант собственной разработки на основе брокера сообщений RabbitMQ.

Анализ будет состоять из двух частей: общего обзора программного обеспечения для получения общих сведений о каждом продукте и детального сравнения исследуемых продуктов в итоговых таблицах.

Для детального анализа, были отобраны стандартные свойства сервисных шин, которые будут использованы как предварительный набор для их сравнения. В список таких свойств вошли: наличие адаптера для работы с SQL, поддержка внутренних моделей для использования адаптерами, поддержка маршрутизации, инструмент для разработки моделей, настройки маршрутизации, возможность разработки своего адаптера, API описание, библиотеки, поддержка .NET, возможность получить демоверсию, кластеризация, наличие курсов обучения, отладка маршрута, производительность сообщений в секунду, русскоязычная документация и интерфейс и ориентировочная стоимость внедрения.

1.4.1 IBM Integration Bus

Корпоративный продукт от компании IBM, который содержит в себе интеграционный узел (брокер сообщений и сервер), инструменты для разработки объектов и маршрутов, набор приложения для управления сервером. Ранее этот продукт был известен под именем IBM WebSphere Message Broker [70].

Значительным преимуществом решения является кроссплатформенность, которая обеспечивает возможность установки сервера как на Windows платформу, так и на Linux-Unix совместимые. Причем версия для Windows разработана на платформе .NET, а версии под Linux и Unix разработаны на платформе Java.

Содержит в себе богатый функционал по отладке бизнес-логики обработки сообщений и возможности по их созданию. Помимо стандартного набора адаптеров, для работы с сервисами поддерживает работу через TCP/IP - подключение к сервису напрямую через сокет, что может упростить задачу разработки адаптеров. Имеет возможность использования сервера приложений.

Поддерживает внутреннюю разработку на языках ESQL, Java, .NET, XSLT. Разработка адаптеров возможна только на языке Java, однако присутствует возможность написания библиотек аналогичной функциональности на .NET, для подключения к адаптерам NETInputNode, NETComputeNode. Полная стоимость решения 4 863 270 рублей.

1.4.2 Microsoft BizTalk Server

Этот продукт является довольно большим решением со всеми функциями, которые должен предоставлять полноценный ESB продукт. Разработан для OS семейства Microsoft. Содержит в себе брокер сообщений, механизмы для автоматизации бизнес-процессов, функции для создания

внутренних моделей, маппинг и преобразование данных, поддерживает разработку адаптеров в среде .NET - содержит в поставке готовые компоненты для их разработки.

Содержит в поставке поддержку транспортных адаптеров для работы с файлами, почтой, FTP, HTTP. Поддерживает протокол SOAP.

В комплекте присутствует диспетчер, в котором есть среда разработки моделей и настройки бизнес-процессов.

Производительность сервера на обычной конфигурации – около 100 сообщений в секунду. Может быть увеличено до более чем 1000 сообщений установкой на мощный сервер, либо установкой серверов в кластер. В данный момент поддержка этого классического решения практически прекращена и Microsoft сфокусировал свое внимание на разработке облачной версии [73]. Полная стоимость решения 1 698 092 рубля (2 процессора на 2х ядрах).

1.4.3 Oracle ESB, Oracle SOA

Эти два продукта рассматриваются вместе, поскольку по сути являются одним продуктом, выпускаются одной компанией и имеют отличия только в наборе дополнительных компонентов. Продукт ESB позиционируется именно как шина предприятия и предоставляет соответствующие возможности, сравнимые с IBM Integration Bus.

Продукт Oracle SOA является более функциональной версией продукта, которая включает продукты для управления бизнес-процессами и адаптерами для подключения к нестандартным источникам.

Оба продукта имеют возможность проектирования моделей и обладают большим количеством адаптеров, перекрывающим все требования предприятия. Средства отладки позволяют исследовать маршрут сообщений [71].

Для разработки адаптеров можно использовать как родной для продукта язык Java, так и платформу .NET. Оба продукта поддерживают возможности

кластеризации. Полная стоимость обоих решений одинаковая - 3 712 500 рублей.

1.4.4 MULE ESB

Эта кроссплатформенная шина предприятия с открытым программным кодом, официально предоставляется бесплатно, но для полноценной работы требуется купить полную, платную версию. Представляет значительное число коннекторов, которое позволяет соединяться с сервисами FILE, HTTP, FTP, REST, SOAP, поддерживает работу с форматами XML и JSON. Обладает возможностью для отладки сообщений. Поддерживает работу в кластере.

Дает возможность вызывать функции не только асинхронно, но и синхронно что позволяет адаптеру инициатору ожидать ответ от адаптера исполнителя.

Сервис разработан для платформы Java, интеграция с другими платформами возможна через WEB-сервисы [67]. Обычная версия поставляется бесплатно, версия для коммерческого использования стоит 6 307 200 рублей.

1.4.5 ESB Галактика

Это решение является специально разработанным для продуктов семейства “Галактика” и позволяет интегрировать между собой различные продукты корпорации “Галактика”.

Основная функция решения - периодически проверять наличие изменений в базе данных продуктов “Галактика” и информировать об этом подписчиков таких событий. Фактически это система для репликации баз данных собственных приложений корпорации “Галактика” и негалактических приложений. Основным отличием от обычной репликации является обязательное использование промежуточных, “канонических моделей”,

которые обязывают подключаемые адаптеры использовать для обмена единый формат моделей сущностей, которыми идет обмен. Для работы использует брокер сообщений RabbitMQ [50].

Решение предназначено, в основном, для интеграции продуктов корпорации “Галактика” между собой, адаптеры имеются для базовых продуктов Галактики, таких как ERP, AMM, EAM и других. Для интеграции с другими решениями нужно разрабатывать свой адаптер, но описания протокола для обмена между адаптерами на данный момент не разработчик не имеет.

Стоимость продукта 576 000 рублей на ядро и 780 000 стоит лицензия на разработчика. Полная расчетная стоимость 3 084 000 рублей.

1.4.6 DATAREON ESB

Отечественный продукт от компании Axelot. Является кроссплатформенным продуктом, который может работать под управлением операционных систем как Windows, так и Linux. Ядро системы разработано на C#, вспомогательные приложения на Java. Имеет более скромные возможности по реализации бизнес-логики и поддерживает стандартный набор коннекторов, в котором можно отметить компоненты для связи с базами данных 1С [66].

Обладает хорошими возможностями по отладке, примерно сравнимыми с уровнем продуктов от Microsoft и IBM. Содержит большое количество адаптеров для подключения к различным системам. Для разработки адаптеров используется платформа .NET.

Программный комплекс входит в единый реестр российских программ для электронных вычислительных машин и баз данных. Не обладает возможностями кластеризации. Полная стоимость необходимой лицензии 1 744 962 рубля.

1.4.7 Mediator ESB

Данный продукт является отечественной разработкой. Обладает довольно скромным, но достаточным количеством адаптеров. В качестве диспетчера управления используется отдельный продукт - платформа «Lorica Secunda», которая предоставляет веб-интерфейс для администрирования шиной. По заверениям разработчиков поддерживаются все операционные системы.

Шина разработана в основном на Java. Имеет встроенный язык программирования. Возможность разрабатывать собственный адаптеры [72].

Присутствует специальный интерфейс для реализации формализованных запросов к базам данных. Решение не поддерживает кластеризацию. Полная стоимость решения составляет 900 000 рублей за пакет, в который входит 300 000 за лицензию и годовой поддержка за 600 000.

1.4.8 Собственное решение на основе RabbitMQ

RabbitMQ не является продуктом который можно классифицировать как сервисную шину предприятия, это специальное связующее программное обеспечение, предназначенное для обмена сообщениями, составная часть шины предприятия, которое позволяет реализовать механизмы, позволяющие приложениям обмениваться между собой в рамках упорядоченной очереди сообщений.

RabbitMQ является свободно распространяемым, бесплатным продуктом, в своей лицензии практически не имеющий ограничений для использования в корпоративных системах [69]. Предоставляя функции обмена сообщениями, он является по сути шиной предприятия, без управляющих модулей, готовых адаптеров и внутренних моделей. Работает через протокол AMQP, который представляет собой посылку в формате XML, с служебными данными и полем Payload, в котором содержится передаваемое сообщение. В

качестве хранилища данных использует базу данных Mnesia. В поставке имеет библиотеки для работы с Java, .NET, Perl, Python, Ruby, PHP. Предельно простой и надежный сервис, поддерживает кластеризацию.

Для работы в среде предприятия может использоваться как самостоятельная единица, сервис, к которому подключаются приложения-адаптеры. Для этого необходима разработка таких адаптеров и общей для них архитектуры объектов. Примерная стоимость разработки такого решения, с учетом особенностей предприятия, около 300 000 тысяч рублей.

Выводы по первой главе

Таким образом, на основании проделанного исследования, можно сделать следующие выводы: сервисная шина предприятия — это программный продукт, относящийся к классу связующего программного обеспечения и соответствующий сервис-ориентированной архитектуре. Продукты такого класса используются для унификации информационного обмена компонентов информационной системы как между собой, так и с внешними информационными системами через единую точку доступа.

С учетом того, что в современном бизнесе взаимодействие играет значительную роль, интеграция продукта такого класса может значительно повысить скорость и эффективность создания интеграционных решений.

Вопрос внедрения продуктов такого класса практически не затронут современными исследованиями. Большая часть работ, в которых упоминается серверная шина предприятия, редко затрагивает теоретические основы архитектуры и вопросы внедрения. Вся литература посвящена вопросам, как правило, косвенно связанным с сервисной шиной предприятия.

На основании проведенного исследования созданы таблицы 1.1 и 1.2 с указанием соответствия функциональности различных вариантов интеграции с требованиями, предъявляемыми к интегрируемому продукту.

Таблица 1.1 – Соответствие функциональности - импортные решения

Функция/Система	**IBM Integration Bus**	**Microsoft Biztalk ESB**	**Oracle ESB/SOA**	**Mule ESB**
SQL Адаптер	Да	Да	Да	Да
Поддержка внутренних моделей	Да	Да	Да	Да
Поддержка маршрутизации	Да	Да	Да	Да
Инструмент для разработки моделей, настройки маршрутизации	Да	Да	Да	Да
Возможность разработки своего адаптера	Да	Да	Да	Да
API описание, библиотеки, поддержка .NET	Да	Да	Да	Частично
Возможность получить демоверсию	Да	Да	Да	Да
Кластеризация	Да	Да	Да	Да
Наличие курсов обучения	Да	Да	Да	Да, онлайн
Отладка прохода сообщения по маршруту	Да	Да	Да	Да
Производительность сообщений в секунду	Нет данных	Около 100	Нет данных	Нет данных
Русскоязычная документация	Нет	Да	Нет	Да
Русифицированный интерфейс	Да	Да	Админ. консоль	Да
Ориентировочная стоимость внедрения (руб.)	4 863 270	1 698 092	3 712 500	6 307 200

Таблица 1.2 – Соответствие функциональности - отечественные решения

Функция/Система	**Datareon ESB**	**Mediator ESB**	**Галактика ESB**	**Свое решение**
SQL Адаптер	Да	Да	Нет	Да
Поддержка внутренних моделей	Да	Да	Да	Да
Поддержка маршрутизации	Да	Да	Нет	Да
Инструмент для разработки моделей, настройки маршрутизации	Да	Да	Нет	Нет
Возможность разработки своего адаптера	Да	Да	Нет	Да
API описание, библиотеки, поддержка .NET	Нет	Нет	Нет	Да
Возможность получить демоверсию	Да	Да	Да	Нет
Кластеризация	Нет	Нет	Нет	Нет
Наличие курсов обучения	Да	Нет	Да	Нет
Отладка прохода сообщения по маршруту	Да	Нет	Нет	Да
Производительность сообщений в секунду	Нет данных	Нет данных	Нет данных	> 100
Русскоязычная документация	Да	Да	Да	Да
Русифицированный интерфейс	Да	Да	Да	Нет
Ориентировочная стоимость внедрения (руб.)	1 744 962	900 000	3 084 000	300 000

Поскольку предприятие обладает собственной командой разработчиков с учетом проведенного исследования, самым выгодным, эффективным и быстрым решением будет разработка собственного программного продукта.

Такой метод позволит разработать сервисную шину предприятия с учетом особенностей самого предприятия. Полученный программный продукт будет адаптирован под нужды предприятия и будет содержать только те компоненты, которые необходимы в работе.

Для того, чтобы разработать и интегрировать в корпоративную информационную систему продукт такого класса, на следующей стадии исследования необходимо провести предварительные этапы разработки, такие как исследование информационной структуры предприятия для составления требований к будущему продукту. Создание моделей будущего программного продукта и выбор инструментов для его разработки.

Глава 2 Анализ методов и подходов к интеграции, разработка моделей и алгоритмов связующего программного обеспечения

Для того чтобы начать работы по разработке сервисной шины предприятия, необходимо предварительно провести ряд исследований, которые станут основой технических требований, по которым будет разработан новый программный продукт.

Должны быть проанализированы существующие методы и подходы к интеграции связующего программного обеспечения с критической точки зрения, сделаны предложения по совершенствованию и их обоснование.

После исследования корпоративной информационной системы, на которой происходит внедрение, будет создан набор функциональных и параметрических требований к объекту.

После этого этапа исследования будет произведено проектировании структуры и компонентов создаваемого программного продукта.

2.1 Анализ методов и подходов в интеграции связующего программного обеспечения

Сервисная шина предприятия является нестандартным программным продуктом, вопрос интеграции которого связан не столько с особенностями самой сервисной шины, сколько с особенностями корпоративной информационной системы, в которую он интегрируется.

В связи с этим можно сделать вывод, что нет унифицированных методов и подходов к вопросам, связанным с интеграцией такого продукта. Конкретная стратегия и отдельные решения выбираются на основе результатов исследования, которое должно состоять из результатов анализа корпоративной информационной системы, содержащих набор необходимых программных свойств и характеристик для связующего программного обеспечения. Эти сведения найдут свое отражение при формализации задачи

для разработки, в создаваемых моделях, функциональных требованиях и ожидаемых характеристиках.

При исследовании корпоративной информационной системы, нужно выделить три основных направления - уточнение набора программного обеспечения предназначенного для работы через сервисную шину предприятия, изучение области их взаимодействия и уточнение объема информационного обмена.

Уточнение набора интегрируемого программного обеспечения должно сузить круг интегрируемых в сервисную шину приложений, поскольку не каждое приложение может быть интегрированно настолько, что позволит полноценно использовать этот способ взаимодействия.

Изучение области взаимодействия программных средств позволит выделить основные точки, для которых должны быть созданы адаптеры в сервисной шине.

Объем информационного обмена, программ, отобранных ранее, позволит выяснить тот объем информации, который планируется перевести на сервисную шину предприятия и способы интеграции. Эта информация будет использована как требование к продукту в виде минимального количества сообщений, которое ему необходимо обрабатывать в определённый временной промежуток.

2.2 Исследование корпоративной информационной системы

Корпоративная информационная система предприятия состоит из множества разрозненных компонентов. Это многочисленные интеграционные сервисы с контрагентами и партнерами компании, свое и чужое прикладное программное обеспечение и учетные системы, которые представляют из себя продукты класса ERP. Все эти системы имеют одну общую черту - единую точку взаимодействия, базу данных MS SQL Server (рисунок 2).

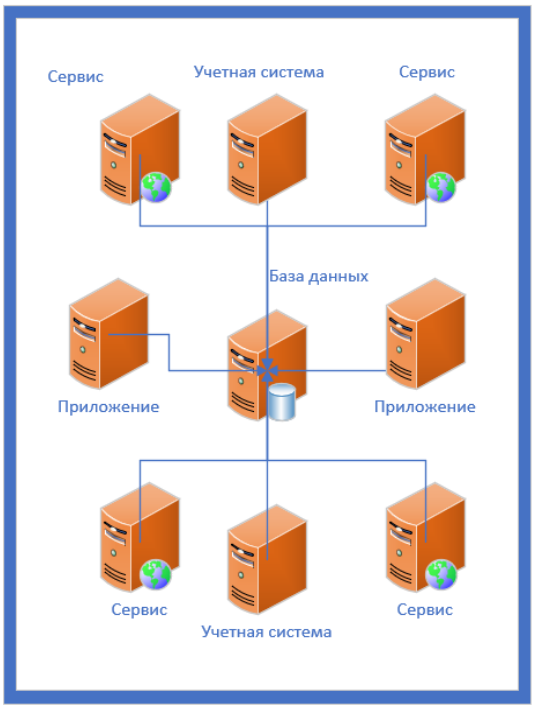

Рисунок 2 – Существующая схема взаимодействия

Как видно из схемы, все информационные потоки процессов пересекаются с базой данных. Поскольку планируется направить все потоки через сервисную шину предприятия, то расчётная нагрузка на сервисную шину должна соответствовать тем нагрузкам, которые ложатся на используемую базу данных.

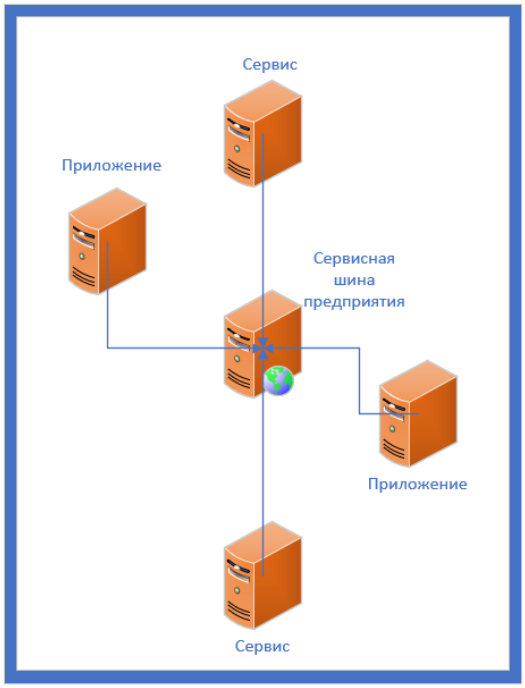

Рисунок 3 – Работа через серверную шину предприятия

При такой схеме на нее ложится основная нагрузка и это дало возможность рассчитать объем траффика, который корректно должна обрабатывать сервисная шина (рисунок 3).

С помощью специального программного средства для базы данных SQL Server Profiler [56]. Это специальный интерфейс для создания трассировок базы данных Microsoft SQL Server и управления ими, а также для анализа и воспроизведения полученных результатов. Этот продукт идет в поставке с комплектом программного продукта Microsoft SQL Server.

Это приложение использует специальные отладочные механизмы, которые предоставляет ему Microsoft SQL Server, к которому производится подключение.

События сохраняются в файле трассировки или в другой базе данных, которые затем могут быть проанализированы или использованы для воспроизведения определенных последовательностей шагов для выявления возникших проблем. В данном случае с его помощью будет выполнен анализ объема, обрабатываемого базой данных.

Для проведения этого исследования было выделено 60 минут. После остановки SQL Server Profiler, путем запроса к базе данных, где хранились все данные, прошедшие через базу данных, был выполнен анализ, его результаты отображены на таблице 2.1.

Таблица 2.1 – Анализ объема передаваемых запросов корпоративной информационной среде предприятия

	Время мин.	Количество запросов всего шт.	Объем запросов всего в байтах	Максимум сообщений в мин. шт.	Максимум байт в мин.
ERP Галактика	60	1446199	264795122	67532	12607088
Программы для интеграции на шину	60	2002	513183	59	31641

На основе этих данных были сделаны выводы о невозможности работы через сервисную шину предприятия основой учетной системы, программного комплекса ERP Галактика. Это произошло по причине значительного объема трафика, создаваемого экземплярами этих приложений, вместе они создавали более тысячи запросов в секунду, в то время как все остальные приложения и сервисы создавали не более 10 запросов в секунду в моменты пиковой нагрузки. Это связанно с особенностями работы ERP Галактика и формируемых ей запросов. Такой объем данных не может быть заведен в сервисную шину предприятия, поскольку для осуществления такой процедуры нужны мощности, значительно превышающие те, которыми обладает предприятие.

После формирования списка приложений на интеграцию из данных, были отобраны те элементы корпоративной информационной системы, которые предполагается интегрировать в сервисную шину предприятия.

К таким относятся приложения, которые имели либо встроенную возможность работы с сервисной шиной предприятия, либо имели доступный исходный код, с помощью модификации которого их можно было интегрировать в сервисную шину. Из списка были исключены приложения, подготовленные к выводу из эксплуатации. По результатам была составлена таблица, содержащая основные характеристики информационного обмена учетных систем и приложений, предназначенных для интеграции на сервисную шину предприятия.

На основе полученного списка были сформированы требования к проведению исследований - для полноценной работы необходимая такая пропускная способность шины, которая будет позволять обрабатывать в минуту не менее 59 сообщений, со объемом передаваемой информации не менее 31641 байт.

Вторым вопросом, который получил свой ответ после исследования корпоративной информационной системы предприятия, стал вопрос о

создаваемых адаптерах. На данный момент требуются адаптеры для доступа к SQL Server в вариантах исполнения в синхронном режиме работы и асинхронном, адаптер, который будет использоваться для получения ответов для клиентов, работающих в асинхронном режиме работы и веб-адаптер, который будет выполнять роль единой точки входа в сервисную шину предприятия и адаптер для получения ответов асинхронных запросов.

Таким образом этап предварительного сбора информации завершен, сформулированы требования к разрабатываемому программному продукту и можно приступить к этапу проектирования.

2.3 Разработка моделей и алгоритмов связующего программного обеспечения

Проектирование связующего программного обеспечения не отличается от проектирования обычного программного обеспечения, в первую очередь производится составление моделей.

Составление моделей, или моделирование – это замещение одного объекта-оригинала объектом-моделью, внешние свойства модели, которые важны при взаимодействии с другими системами, называются характеристиками системы [8, с. 5].

Этот этап исследования будет состоять из создания диаграммы последовательности, диаграммы классов, диаграммы вариантов использования, структурно-функциональной модели проектируемого продукта, логического и физического моделирования объектов базы данных, выбора алгоритмов работы.

Логические модели отобразят создаваемые программные сущности без привязки к конкретному инструменту или базе данных. Такой метод моделирования позволяет абстрагироваться от вопросов реализации, поставив в основу не инструмента, а принципы, закладываемые в построение программного продукта.

Физические модели отобразят сущности, реализованные с использованием выбранного инструмента.

Общая концепция моделирования выглядит как переход от логических моделей к физическим, с получением в качестве финального результата готового программного объекта [35, 154].

2.3.1 Диаграммы последовательности обработки сообщений

Для моделирования взаимодействия элементов моделей можно использовать диаграмму последовательности - особый вид диаграмм, предназначенный для представления взаимодействия между элементами модели программной системы в виде условной линий процесса и сообщений, которыми они обмениваются [15, 202].

Такими диаграммами будет описано прохождение пакетов по сервисной шине. Предусмотрено несколько вариантов последовательности обработки запросов, такие как синхронный запрос одного адаптера к другому, синхронный запрос с ответом от адаптера и асинхронный запрос с ответом.

При использовании синхронного запроса клиентское приложение из своего исходящего адаптера отправляет через сервисную шину предприятия данные на входящий контур другого адаптера (рисунок 4). В данном случае клиентское приложение не интересует ни факт доставки, ни ответ от системы получателя. Данные отправляются однонаправленно потоком сообщений в сторону получателя. Такой способ можно назвать классическим взаимодействием с сервисной шиной предприятия.

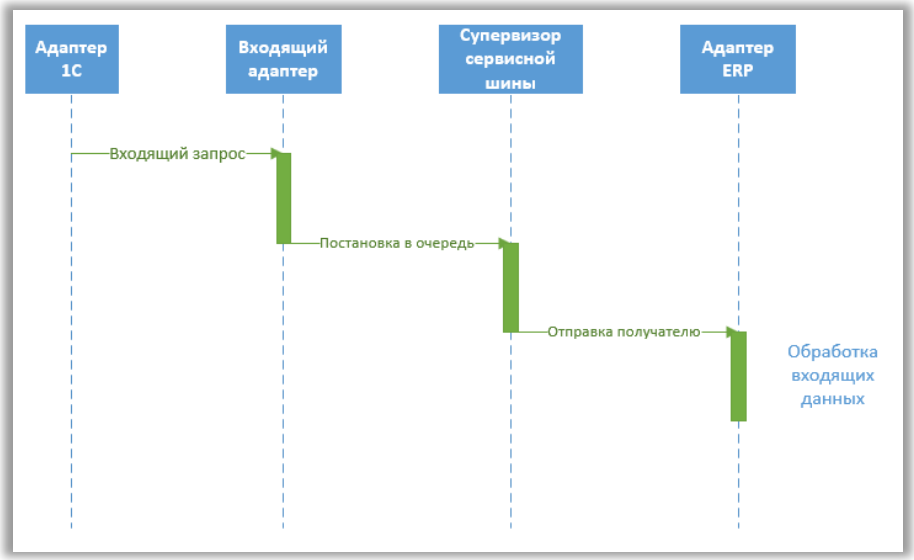

Рисунок 4 – Синхронный запрос

Способ работы, основанный на синхронном способе обмена с ответом, подразумевает более активное взаимодействие систем и характеризует более оперативный информационный обмен. Он используется тогда, когда необходимо на запрос получить ответ опрашиваемой системы. В этом случае используются адаптеры, имеющие возможность оперативного сетевого взаимодействия типа ответ-запрос такие, как веб-сервер. При получении входящего метода типа POST веб-сервер отправляет данные по сервисной шине предприятия и ожидает ответ от нее с помощью средств взаимодействия, которые предоставляет сервисная шина (рисунок 5). Наиболее удачным тут может быть использование механизма подписчик/публикатор, которые предоставляет брокер сообщений типа RabbitMQ. После получения от сервисной шины адаптер передает ответ клиенту в виде формализованного сообщения и после этого отключает соединение. Такой метод взаимодействия может быть полезен при получении информации из базы данных - в запросе к сервисной шине отправляется формализованный SQL запрос, в ответе – формализованный ответ в виде документа в формате XML или JSON.

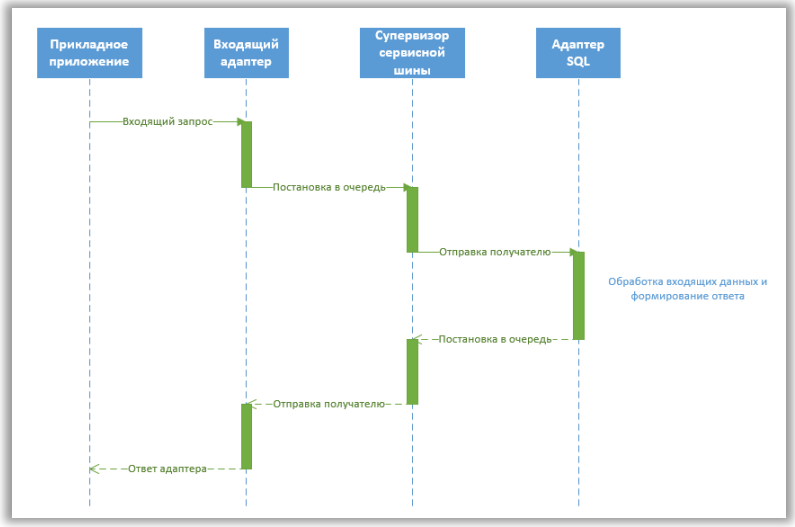

Рисунок 5 – Синхронный запрос с ответом

Третий вариант работы с сервисной шиной подразумевает гибридное взаимодействие с адаптером, которое имеет признаки синхронного и асинхронного обмена. Этот вариант может быть наиболее применим при работе с системами, которые выполняют многоступенчатую обработку поступающих данных в соответствии с какой-либо программной логикой или реализуя сложный бизнес-процесс. Такой вариант работы во многом похож на синхронный, но с одним важным отличием. В этом случае на входящий запрос адаптер отправляет полученный объект в брокер сообщений и отвечает на запрос сообщением об успешной обработке входящего запроса и разрывает соединение. После получения данных адаптером получателем и произведения необходимых действий над данными создается новое сообщение, которое содержит идентификатор ReplyTo, который идентифицирует изначальный запрос, т.к. идентификатор у этого сообщения будет свой, а найти ответ клиент сможет только с использованием изначального идентификатора. Клиентская система, отправитель запроса, может запрашивать результат выполнения по номеру отправленного сообщения и получить полноценный ответ после его выполнения (рисунок 6). Этот вариант наиболее удобен по балансировке нагрузки, так как не ставит сервисную шину предприятия или ее адаптер в

какие-либо временные рамки и позволяет обрабатывать сообщения в порядке обычной очереди.

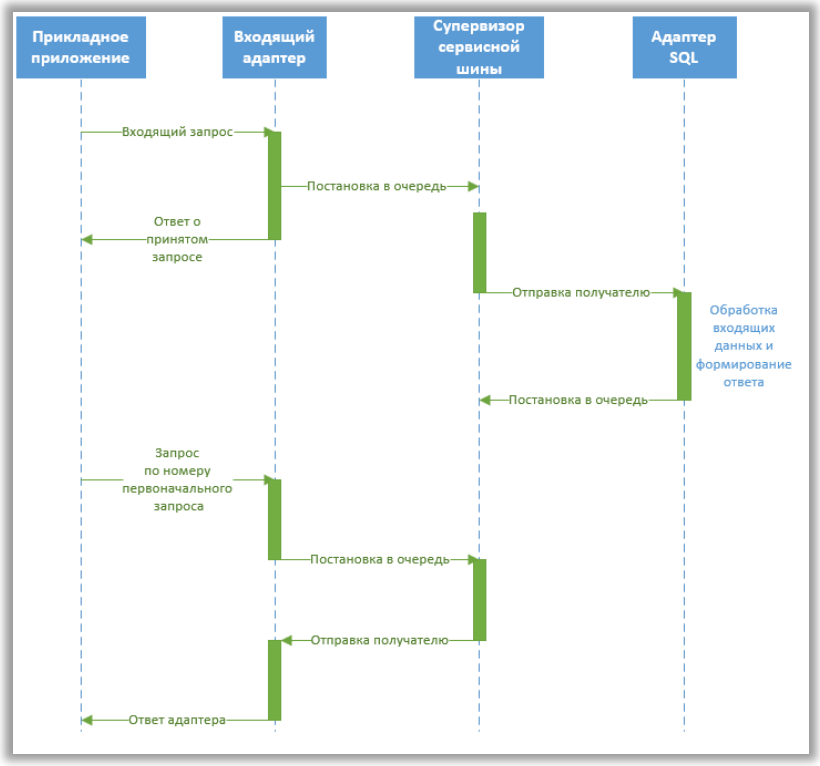

Рисунок 6 – Синхронно-асинхронный запрос с ответом

2.3.2 Диаграмма классов сервиса сервисной шины

Одна из основных целей объектно-ориентированного программирования — это моделирование реальных систем способами, аналогичными процессам мышления. Создание классов — это объектно-ориентированный способ создания таких моделей. Вместо использования структурированного подхода, когда данные и поведение являются логически отдельными сущностями, объектно-ориентированный подход инкапсулирует данные и поведение в объекты, которые взаимодействуют друг с другом [48, 75].

Диаграмма классов отображает такие объекты в формате, демонстрирующем общую структуру иерархии классов программной единицы, их кооперацией, атрибуты и поля, методы, интерфейсы и

взаимосвязей между ими. Этот метод моделирования широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования [51].

Общая схема для основного модуля и внутренних адаптеров отображена на рисунке 7.

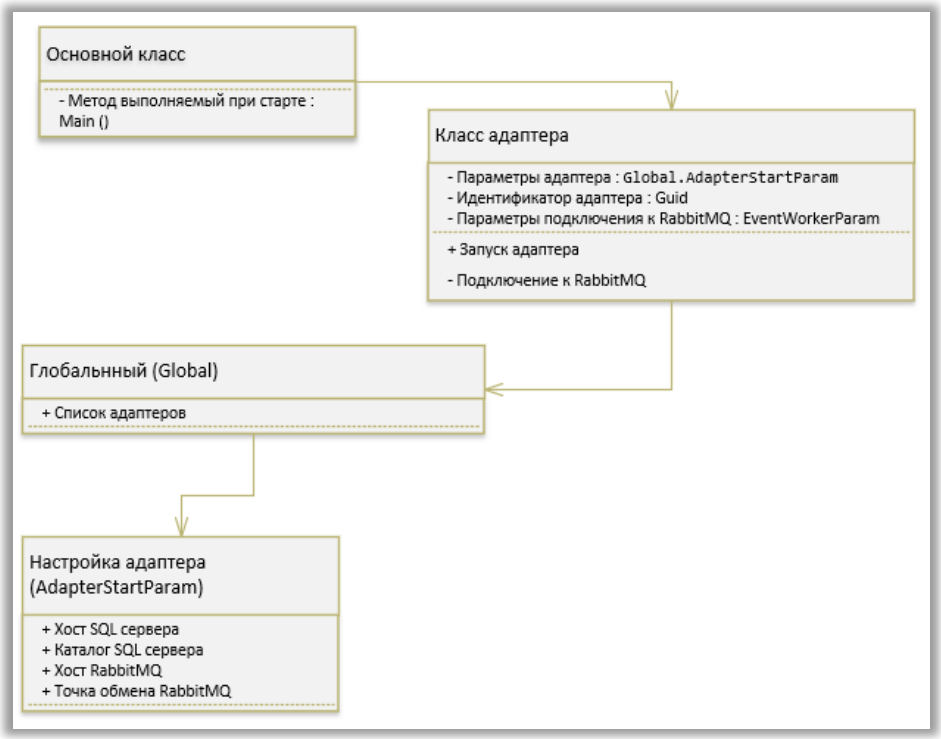

Рисунок 7 – Диаграмма классов

2.3.4 Диаграмма процесса обработки сообщения

Для отображения процесса обработки сообщения будет использоваться структурно-функциональная модель. Эта модель совмещает в себе два похода к моделированию. Структурный – когда отображается структура моделируемого объекта, существенные для целей исследования свойства и взаимосвязи компонентов объекта. Функциональный - когда отражается внешне воспринимаемое поведение и функционирование самого объекта. Такая модель даст общее понимание о принципах работы разрабатываемого продукта [5, с. 4].

Модель отображена на рисунке 8, в ней отображены структурные элементы сервисной шины и их функциональное взаимодействие.

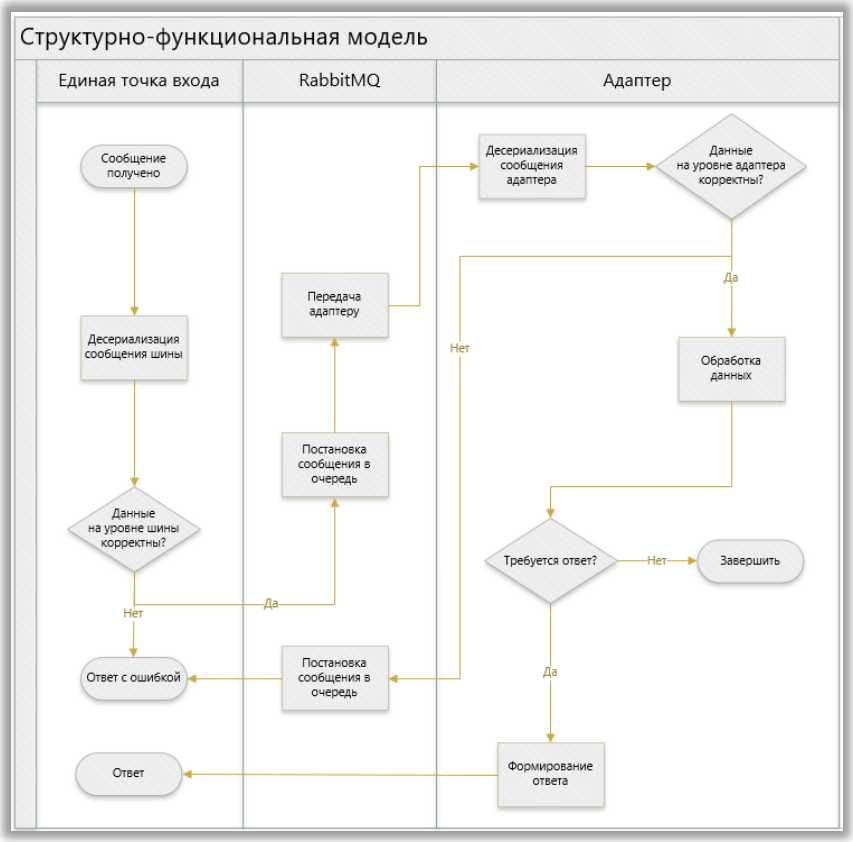

Рисунок 8 - Диаграмма процесса обработки сообщения

2.3.5 Логическая модель БД

Логические модели создаются на основе фактов и утверждений для того, чтобы создать отражение какой-либо сущности без привязки к конкретной технологии, через которую оно будет реализовываться. Такой подход позволяет на ранних этапах проектирования создавать структуры будущего программного продукта, не задумываясь о том, каким образом это будет реализовываться. Логическая модель — это основа для создания физической модели [39, с. 41]. Логическая модель создаваемой базы данных отображена на рисунке 9.

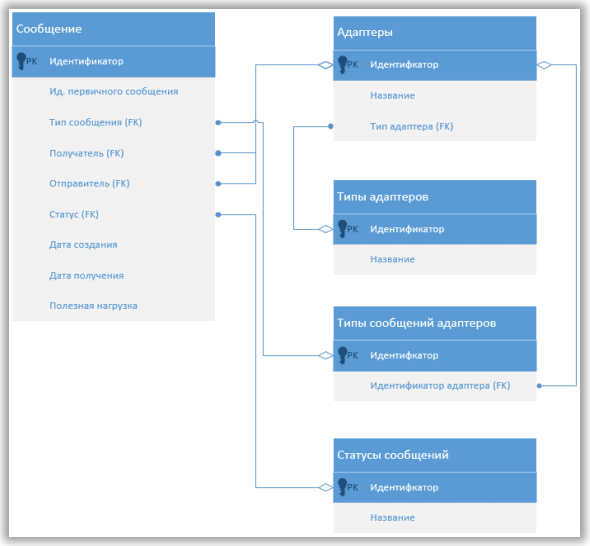

Рисунок 9 – Логическая схема базы данных

2.3.6 Физическая модель БД

Физические модели отображают сущности объектов в той нотации, в которой они будут использоваться в виде программных компонентов [65]. На рисунке 10 отображена физическая схема объектов базы данных. В ней расположены четыре таблицы – Messages, Adapters, AdaptersTypes, Statuses и MesagesTypes. В каждой таблице присутствует первичный ключ, с помощью которого будет осуществляться индексирование [37, с 492] и связь с другими таблицами по внешнему ключу [19, с. 88].

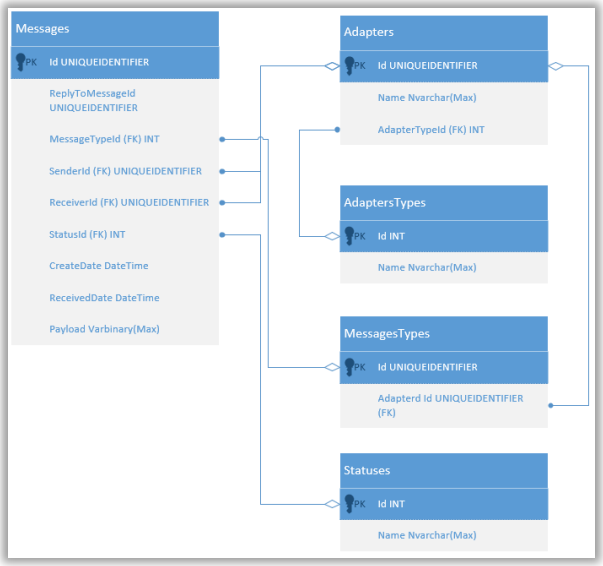

Рисунок 10 – Физическая схема базы данных

2.3.7 Разработка алгоритмов

Для создания алгоритма работы сервисной шины необходимо разработать блок схемы, по которым будут созданы соответствующие методы – программные элементы, содержащие код.

В псевдопараллельной многозадачной среде, такой как Windows или Linux, для клиентских приложений существует два варианта выполнения кода – в основном потоке или из дочернего программного потока, объекта типа Thread, запущенного из основного потока.

Такой механизм поточной многозадачности позволяет использовать параллельные операции и добиться большей производительности [30, с. 2], что особенно необходимо при работе с несколькими адаптерами.

После запуска и проведения подготовительных стадий, таких как чтение конфигурации и запуск встроенных адаптеров, код самого сервиса будет выполняться на зацикленном участке в основном потоке, каждый адаптер будет выполняться параллельно с основным в отдельных потоках. Все дочерние потоки будут существовать вплоть до того момента, пока не закончена работа основного потока.

Такой подход связан с тем, что работа адаптеров подразумевает наличие программных блокировок при выполнении различных функций, таких как выполнение запроса к БД или RabbitMQ, ожидание входящего соединения от веб-клиента и на время этого ожидания сервис будет недоступен для других действий, что значительно снизит его производительность и приведет к отказам в обслуживании клиентов [31, с. 181]. Для любого сервиса, в том числе и сервисов класса сервисных шин предприятия, такое поведение недопустимо.

Общую схему работы основного потока и адаптеров отображена в блок-схеме на рисунке 11.

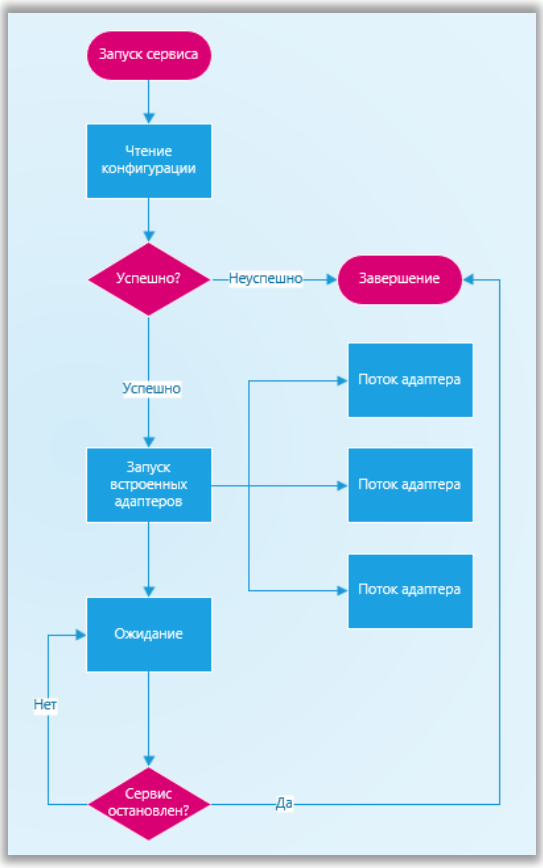

Рисунок 11 – Алгоритм работы сервиса

2.4 Выбор инструментов для разработки сервисной шины

Для предстоящей разработки будут выбраны инструменты для создания самого сервиса и его встроенных адаптеров, базы данных и ее управления.

Для разработки сервиса необходимо выбрать специальный инструментарий - среду разработки, платформу, которая будет отвечать целям его использования при обеспечении достаточной функциональности. При этом, этот инструментарий должен быть максимально приближен к той политике разработки программного обеспечения, которая сложилась на предприятии. Такой подход позволит обеспечить разработку собственными силами без привлечения подрядчиков.

Исходя из имеющихся кадровых возможностей предприятия, для разработки сервисной шины было принято решение остановится на варианте языка C# и платформе .NET.

В качестве базы данных будет использоваться Microsoft SQL Server 2019, поскольку этот именно этот тип баз данных используется на предприятии.

Выводы ко второй главе

Этот этап был вторым из четырех этапов процесса интеграции связующего программного обеспечения на базе предприятия. В нем была произведена предварительная стадия разработки сервисной шины предприятия, которая включала в себя исследование корпоративной информационной системы предприятия, моделирование и создание алгоритмов и выбор инструментов для разработки.

Запланированные задачи были решены в полном объеме, подготовленный материал готов к использованию.

Исходя из результатов исследования, надо использовать полученные рекомендации и данные для разработки программного продукта.

Внедрение разработанного продукта позволят повысить качественность архитектуры корпоративной информационной системы и улучшить интеграционные возможности предприятия в целом.

Научно-технический уровень НИР сложно оценивать в силу отсутствия альтернативных работ в литературе.

В целом исследование выполнено на основе стандартных методических рекомендаций, отсутствующие в рекомендациях исследованиях проводились на основе субъективной оценки.

Следующим, третьим этапом исследования, станет процесс разработки сервисной шины предприятия и ее интеграция в корпоративную информационную систему предприятия.

По результатам интеграции должны быть проведены испытания, подтверждающие качество и характеристики разработанного программного продукта.

Глава 3 Реализация сервисной шины предприятия

Разработка программного обеспечения будет производится по технологии нисходящего программирования, методом последовательной детализации.

Этот метод используется в тех случаях, когда структура объекта в общем известна и программный код создается на основе известных или разработанных моделей. В данном случае имеются разработанные на предыдущем этапе исследования модели, которые составят программную архитектуру программного продукта.

Разработка будет ввестись по нисходящей, сверху вниз, уровень за уровнем. После создания проекта приложения будут созданы классы и используемые объекты, и значения. Методы будут созданы без кода, заполнение их логикой будет производится последовательно, после каждого этапа будет проводиться отладочное тестирование, которое является важной частью процесса разработки [21, с. 7].

Сначала будет разработано ядро сервисной шины, основной ее каркас – инициализация, запуск потоков, цикл ожидания, который содержится в основном модуле создаваемого проекта.

После этого будут разработаны вспомогательные модули, которые обеспечат работу ядра шины с конфигурационным файлом, обеспечат поддержку работы с формализованными документами, базой данных и брокером сообщений. Также будет создан глобальный модуль, который будет содержать данные, используемые несколькими модулями. Такой способ модульного разделения позволяет создать более простое решение, в которой автономность модулей будет уменьшать их связи зависимость от основного модуля [27, 309].

Следующим этапом станет разработка встроенных адаптеров, каждый из которых будет создан на основании разработанных моделей и требуемых функциональных возможностей.

3.1 Разработка программного ядра сервисной шины

На этом этапе будет использоваться выбранный ранее инструмент Visual Studio 2019. Первоначально сервис будет разработан в виде консольного приложения. Такой подход позволит провести более качественное тестирование и отладку создаваемого приложения.

Поскольку разрабатываемый сервис не предусматривает взаимодействие с пользователем, требует максимальной устойчивости и работы без входа пользователя в систему [43, с. 11], то по окончании разработки и тестирования проект будет сконвертирован в сервис Windows, поскольку возможности Windows сервиса позволяют успешно бороться с отказами, используя механизмы автоматического перезапуска его службы и работа в качестве сервиса позволит более гибко управлять режимами его работы [59].

Для создания продукта создается новый проект, который берется из шаблона консольного приложения .NET Framework. В качестве платформы выбрана .NET Framework 4.7.2 как последняя на время разработки версия классического фреймворка.

Следующим шагом в проект добавляются компоненты для работы с брокером сообщений RabbitMQ, это необходимо, потому что что платформа .NET не умеет работать сама с сервисом такого. В Visual Studio, через службу управления пакетами NuGet, в проект сервисной шины устанавливается пакет RabbitMQ.Client версии 6.2.1, который распространяется по лицензии. как свободное программное обеспечение.

После создания приложения создаются элементы основного модуля Program. Это статический метод Main и статический метод Init.

Метод Main зафиксирован в настройках проекта как вызываемый по умолчанию, после запуска приложения ему передается управление. В нем происходит запуск метода инициализации, а далее запускаются адаптеры сервисной шины предприятия (рисунок 12). Практически весь метод заключен

в блок обработки ошибок try/catch — это сделано для того, чтобы ошибки, произошедшие внутри блока были корректно отработаны, и в консоль выведено сообщение об ошибке.

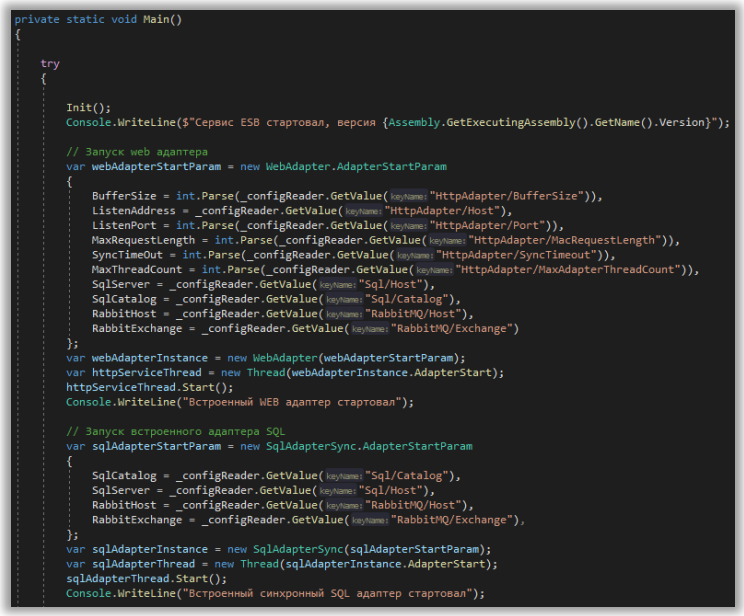

Рисунок 12 – Код основного модуля

Метод Init используется для выполнения первоначальной инициализации, при которой будут прочитаны конфигурационные данные для работы сервиса. В конфигурационном файле описаны все необходимые данные для работы – настройки адаптеров, подключение к RabbitMQ и MS SQL Server.

3.2 Разработка вспомогательных модулей

В проекте используется три вспомогательных модуля – ConfigReader, Global, Sql и Xml.

Модуль ConfigReader используется для чтения параметров конфигурации, в нем присутствуют два публичных метода GetValue и LoadConfig. Метод GetValue возвращает указанное в параметре значение конфигурационного файла в виде строки. Дальнейшее приведение к нужному типу выполняется из метода, который вызвал получение данных. Метод

LoadConfig загружает конфигурацию из указанного файла, а при его отсутствии из значения по умолчанию, которое представляет из себя имя сборки и расширение “.cfg”.

В модуле Global используется для совместного использования классов и объектов другими модулями. В нем созданы два класса AdapterStartParam и InternalAdapters. Первый содержит набор параметров для старта адаптера, второй содержит структуру, которая описывает набор внутренних адаптеров. В модуле созданы два статических объекта InternalAdaptersList и AdaptersLock. Первый используется для хранения внутренних адаптеров, загруженных из базы данных методом Init() основного модуля, второй используется для блокировки одновременного доступа, поскольку используется из разных адаптеров. [29, с. 678]

В модуле Sql создан конструктор - метод, который выполняется при инициализации класса, который его содержит [28, с. 198]. Для выполнения запросов к базе данных создан открытый метод ExecuteScalar. Он является оболочкой для метода System.Data.SqlClient.SqlCommand.ExecuteScalar(), который используется для получения выполнения запроса который возвращает скалярный ответ [45, с. 25]. С его помощью можно получить ответ от базы данных, выполняемый в формате запроса, который возвращает скалярное значение.

Для работы с формализованными данными создан модуль Xml, он содержит три публичных метода, которые используются другими программными модулями – FormatXml, Serialize и Deserialize, которые являются обертками методов из пространства имен System.Xml, System.Xml.Serialization и System.Xml.Linq. Метод FormatXml используется для форматирования текста XML документа в удобный и читаемый вид. Метод Serialize используется для преобразования объекта в формализованный документ формата XML. Метод Deserialize является обобщённым методом, такие методы объявляют параметры типа уже внутри метода [4, с. 141]. Он используется для обратной операции – создания объекта из XML документа.

3.3 Разработка адаптеров

Для интеграции с корпоративной информационной системой будут созданы четыре встроенных адаптера, каждый из которых будет выполнять одну определенную на этапе моделирования функциональность.

Веб-адаптер, который является единой точкой входа в сервисную шину, будет разработан для взаимодействия клиентов и адаптеров.

Адаптер синхронных SQL запросов будет предоставлять клиентам сервисной шины возможность быстрого выполнения произвольного SQL кода в базе данных.

Адаптер асинхронных SQL запросов будет выполнять похожую функциональность, но для сложных запросов, которые долго длятся и выходят за рамки, которые можно вложить в принципы сетевого взаимодействия.

Для получения обработанных сообщений в сервисной шине будет использоваться специальный синхронный адаптер, который будет возвращать клиентам из базы данных результат обработки их запросов в асинхронном формате.

Разработка адаптеров будет производиться последовательно в силу того, что каждый адаптер обладает уникальным набором функциональности. Каждый из них будет разрабатываться индивидуально без применения шаблонов.

Архитектурно все адаптеры будут иметь общие для всех элементы, такие как конструктор, специальный программный элемент – особый метод, который выполняется при создании экземпляра класса. В качестве набора параметров получает он стандартный конфигурационный набор данных в экземпляре класса AdapterStartParam. Это позволит сохранить передаваемый набор данных в отдельном закрытом поле _threadStartParam.

Все адаптеры будут иметь общий метод AdapterStart(), этот метод будет выполняться из основного модуля при старте адаптера.

В каждом адаптере будет открытое свойство AdapterId типа Guid, которое будет идентифицировать адаптер по уникальному коду. Такой тип идентификатора выбран ввиду особенностей работы сервисной шины – он может гарантировать уникальность идентификаторов, приходящих из разных источников [63].

Все остальные свойства будут уникальными для каждого адаптера и будут описаны при их разработке.

3.3.1 Разработка веб-адаптера

Этот адаптер будет разработан по особой схеме, поскольку он является единой точкой входа в сервисную шину, поэтому на него накладывается дополнительные требования по созданию механизма взаимодействия со всеми остальными адаптерами.

Адаптер содержит основной класс WebAdapter, в котором описаны все используемые элементы. Помимо общих элементов с другими адаптерами он содержит собственный набор элементов, которые отвечают за взаимодействие с клиентами.

После инициализации адаптер соединяется с сервером RabbitMQ для получения входящих сообщений из своей очереди и открывает для входящих клиентских соединений порт, указанный в переданных ему настройках, с помощью экземпляра класса HttpListener, и в блокирующем режиме ожидает входящие соединения [52]. На каждую попытку соединения он открывает дочерний поток, который в качестве параметра получает набор конфигурации и контекст входящего соединения. Таким образом адаптер не тратит время на обработку запроса, сразу возвращаясь к приему входящих соединений.

Созданный таким образом поток обрабатывает полученные данные, извлекает поле Payload и отправляет их в RabbitMQ, в очередь адаптеру - получателю, указанному в сообщении.

3.3.2 Разработка синхронного SQL адаптера

Этот адаптер предназначен для быстрого выполнения запросов, он ограничен таймаутами сетевого взаимодействия веб-адаптера, поэтому должен быть максимально оптимизирован для оперативного выполнения наложенных на него функциональных обязанностей.

В созданном модуле SqlAdapterSync создан одноименный основной класс, содержащий все используемые элементы.

Помимо стандартных элементов адаптера он содержит метод EventConsumer, который вызывается при получении нового сообщения через брокера сообщений и запускает дочерний поток для его обработки. Метод EventWorker, вызываемый из метода EventConsumer, является непосредственным обработчиком сообщения. В нем происходит проверка сообщения на соответствие формату, его обработка, запрос к базе данных и возвращение ответа в адаптер источник запроса.

В классе присутствуют вложенные вспомогательные классы, ErrorResponse экземпляр которого создается в ответ на некорректное сообщение, AdapterSqlRequest в который десериализируется полученное сообщение и EventWorkerParam, экземпляр которого отправляется обработчику сообщения.

Также в модуле присутствуют вспомогательные элементы, для формирования сообщений об ошибках, десериализации входящих сообщений и для создания экземпляра с набором параметров для дочернего программного потока.

3.3.3 Разработка асинхронного SQL адаптера

Асинхронный адаптер имеет возможность обрабатывать запросы последовательно, он спроектирован как дочерний поток основного потока и может обрабатывать входящие запросы последовательно из одного потока. Однако с учетом возможной нагрузки, на каждый полученный запрос он также будет формировать дочерние потоки, которые будут самостоятельно обрабатывать запросы. Это сделано для того, чтобы повысить скорость обработки запросов [55] и соответствовать сформулированным требованиям по пропускной возможности.

В модуль названный SqlAdapterAsync созданы требуемые для реализации функциональности программные элементы.

Метод SqlAdapterAsync, конструктор, выполняющий загрузку конфигурации в экземпляр класса, метод EventCosumer – для получения сообщений от брокера, метод EventWorker для работы с полученным сообщением из отдельного программного потока.

В качестве вспомогательных элементов в классе присутствуют вложенные классы ErrorResponse – для формирования ответа при ошибке, SqlAdapterRequest в экземпляр которого десериализируется запрос из сообщения, класс EventWorkerParam экземпляр которого передается созданному потоку EventWorker.

3.3.4 Разработка адаптера для получения сообщений

Этот адаптер разработан как синхронный адаптер, в задачу которого входит получение ответов из базы данных, и передача их клиенту в рамках того взаимодействия, которое предоставляет веб-сервис.

Он является промежуточным звеном между веб-сервисом и базой данных. По принципам своей работы он похож на синхронный модуль SQL запросов. Получая запрос от сервисной шины, он осуществляется его

обработку и формализацию путем десериализации полученного сообщения в объект. После этого он отправляет запрос в базу данных, получает ответ, создает новое сообщение, прикладывает к нему полученные данные в поле Payload и отправляет в сервисную шину.

3.4 Разработка базы данных

Разработка базы данных — это многоступенчатый процесс, который требует системного подхода к организации процесса. На первом этапе будет создана сама база данных. На втором ее архитектурная основа - табличные элементы. Третьим этапом будет создание логических связей между таблицами, создание первичных и внешних ключей. Далее будут созданы вспомогательные программные элементы базы данных – процедуры и функции, разработанные в формате языка Transact-SQL (T-SQL) - диалекте языка SQL, который предоставляет фирменные расширения от Microsoft [7, с. 20]. На предпоследнем этапе разработки базы данных будет произведено создание индексов по полям, которые требуются для быстрого поиска. Финальной стадией разработки станет загрузка начальных сведений в справочники базу данных.

Создание самой базы данных выполняется с помощью скрипта, содержащего код в формате T-SQL. В нем указаны основные параметры создаваемой базы данных, название, расположение фалов, содержащих данные и транзакционные журналы, их объемы и прочие свойства

Создание табличных элементов имеет смысл начинать в соответствии с разработанной моделью сверху-вниз с первостепенных членов – наиболее используемых таблиц, вниз до элементов, содержащих справочную информацию.

Это будет соответствовать принципам нисходящей разработки - последовательной детализации, избранной ранее для разработки программных

элементов, поскольку именно такой тип разработки наиболее характерен для баз данных [58].

Таким образом порядок создания таблиц выглядит так – Messages, MessagesTypes, Adapters, AdaptersTypes, Statuses.

Для создания таблицы с сообщениями были использованы выбранные ранее при создании физической модели типы. Для поля ReceivedDate установлено значение по умолчанию – текущая дата. Создан первичный ключ на столбце Id.

При создании таблицы MessagesTypes использовались только два поля – идентификатор, который выступает в качестве суррогатного первичного ключа и идентификатор типа сообщения.

Таблица Adapters была создана SQL скриптом, который отображен на рисунке 12. В нем указан идентификатор адаптера, название и идентификатор типа адаптера.

Следующим этапом стало создание таблицы типов адаптеров AdapterTypes. В нем была создана таблица, в которой созданы поля для идентификатора типа адаптера и его название.

Финальной стадией этого этапа стало создание таблицы статусов сообщений MessagesStatuses. В ней использованы только идентификатор и название статуса.

Для создания условий, при которых будет поддерживаться логическая целостность связи таблиц, будут созданы внешние ключи на зависимых таблицах.

На таблице Adapters будет создан внешний ключ к таблице AdaptersTypes для того, чтобы тип адаптера всегда был указан из этого списка.

Целостность таблицы Messages защищают три внешних ключа, на полях SenderId и ReceiverId имеется связь с таблицей Adapters, статус же сообщения ограничивается набором идентификаторов из таблицы MesaagesStatuses.

Таблица MessagesTypes имеет один внешний ключ, на поле AdapterId связанный идентификатором адаптера из таблицы Adapters.

Для того, чтобы иметь наиболее гибкую систему взаимодействия с базой данных, запросы на получение и модификацию данных будут выведены в отдельные программные элементы базы данных – функции и процедуры.

Эта процедура будет производиться с помощью подтипа языка SQL – языка описания данных, Data Definition Language, который используется для создания объектов базы данных [26].

Суть функции базы данных заключается в выполнении действия, не затрагивающего существующие данные. В них должна указываться исключительно работа по получению данных, такие типы элементов языка SQL как SELECT. Внутри функций можно указать и выполнение процедур при выполнении условия, что они тоже не меняют данные, но основное их назначение – получение данных. В качестве параметров можно указывать имеющиеся типы базы данных, все они будут переданы в код функции и могут быть там использованы.

Для получения списка адаптеров будет использоваться функция GetAdapters, которая будет отдавать данные сразу в формате XML. Это сделано для удобства работы с полученным списком и обеспечения гибкости работы с данными [25], через метод сериализации ответ сразу будет загружаться в объект.

Для получения сообщений будет использоваться функция GetResponseMessage, которая на входе будет получать идентификатор сообщения и получателя, а на выходе возвращает табличную переменную. Работа функции оптимизирована использованием индекса [47, с. 5].

Для сохранения ответов асинхронных запросов создается процедура InsertReplyMessage.

3.5 Настройка RabbitMQ

Настройка RabbitMQ происходит в три этапа - создание обменника, создание очередей и их привязка к обменнику.

Объект типа Exchange – это обменник (рисунок 13), через который, проходят все сообщения сервисной шины, он выполняет функции маршрутизации, но не выполняет функции хранения сообщений, для этого используются очереди [3].

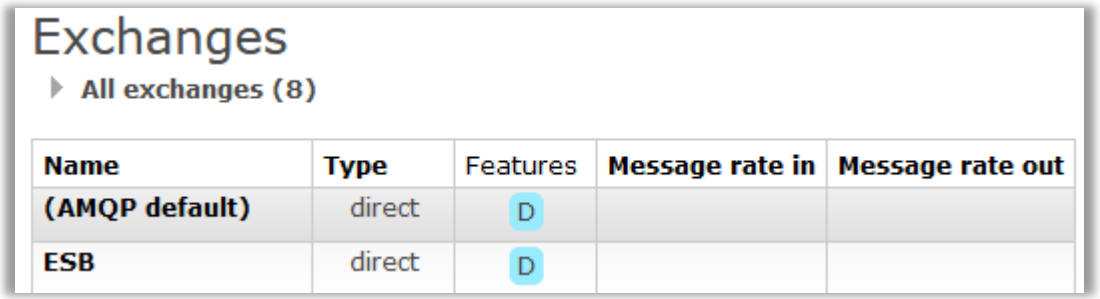

Рисунок 13 – Созданный обменник ESB

Создание очередей (рисунок 14) необходимо для получения и хранения сообщений для конкретных потребителей. После создания все очереди будут привязаны к обменнику через настройку маршрутизации, поскольку в выбранном режиме обменник выполняет отправку сообщений в очереди только в соответствии с настройками маршрута [40, с. 20]. Этот этап необходим, поскольку передача данных из обменника в очередь возможно только при настроенной маршрутизации [17].

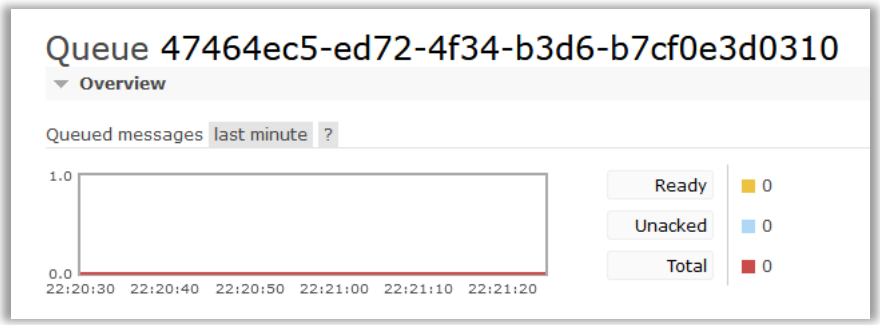

Рисунок 14 – Созданная очередь

3.6 Отладка и пробный запуск

Для проведения отладки будет произведен запуск разработанного приложения в среде Visual Studio. Эта среда имеет свой собственный

отладчик, который позволяет отлаживать код, написанный на одном из поддерживаемых языков, в том числе на C#.

Проект был запущен с помощью клавиши F5 и остановился на выбранной точке входа в начало кода. С помощью клавиши F10 был выполнен пошаговый проход всех строк программного кода. Такой тип отладки используется для поиска конкретной линии кода, содержащей ошибку [38, с. 184].

Далее отладка была проведена до точки, в которой происходит цикл ожидания вплоть до окончания работы сервиса (рисунок 15). Такое поведение программы под отладчиком показало, что код не содержит ошибок и может быть выполнен самостоятельно.

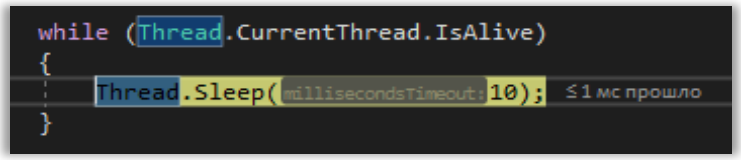

Рисунок 15 – Указатель текущего расположения выполняемого кода

После этого приложение было выполнено без контроля отладчика и на экране появилось окно сервиса с отображением его версии, информации о старте адаптеров и сообщением о готовности принимать сообщения (рисунок 16).

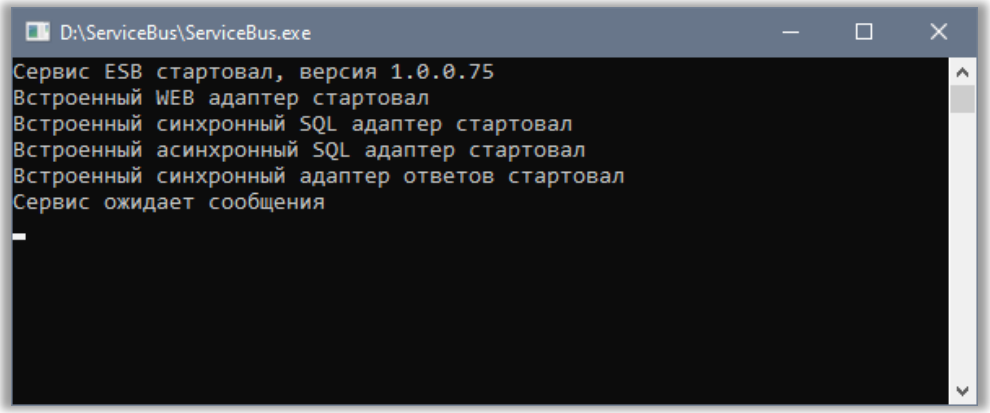

Рисунок 16 – Окно выполняющегося приложения

Таким образом, на основании полученных результатов можно сделать выводы о полноценном запуске всех элементов разработанного продукта и готовности разработанного программного продукта к проведению приемочных испытаний.

Выводы по третьей главе

Таким образом, в результате проведенного исследования, было разработано связующее программное обеспечение класса сервисных шин предприятия.

На первом этапе исследования было разработано программное обеспечение. Оно состоит из нескольких модулей, которые разделены по принципу функциональности, которая в них заложена, и базы данных. Программные объекты создавались путем кодирования с помощью инструмента Visual Studio 2019 на языке C#. С помощью продукта SQL Server Management Studio была создана база данных, табличные элементы их логические ограничения и индексы, были внесены первичные данные, в таблицы содержащие справочные элементы.

На втором этапе была произведена настройка брокера сообщений RabbitMQ.

На третьем этапе была проведена предварительная отладка программного продукта с помощью встроенного отладчика Visual Studio и пробный запуск созданного решения.

На основании проделанного исследования можно сделать выводы о готовности разработанного программного продукта к приёмочным испытаниям, которые будут состоять из функционального и нагрузочного тестирования.

Следующей стадией исследования станет проведение приёмочного испытания созданного продукта путем избранных ранее вариантов тестирования.

Глава 4 Апробация результатов

4.1 Выбранный подход к проведению апробации

Для того чтобы подтвердить рабочую гипотезу, которая заключаются в том, что интегрируемое решение будет полноценно работать в корпоративной информационной системе, нужно провести экспериментальное исследование, которое будет заключаться в проведении приёмочного испытания, которое будет проводится в форме автоматизированного тестирования.

Такое тестирование является стадией разработки программного продукта, частью его жизненного цикла, которая предшествует стадии промышленной эксплуатации [2].

В задачи тестирования обычно не входит выявление конкретных дефектных участков программного кода и никогда не входит исправление дефектов – это задача отладки, которая выполняется по результатам тестирования системы [9, с. 7].

Суть этого исследования будет заключаться в том, что разработанное решение будет проверено с использованием различных вариантов тестирования, подобранных с учетом специфики программного продукта.

Тестирование проводится на конечной стадии разработки программного продукта, поскольку именно в такой форме он может быть протестирован полноценно и сделаны итоговые выводы [10, с. 53].

Для упрощения процедуры тестирования необходимо создать специальный формуляр - чек-лист, который представляет собой набор идей и тест-кейсов [20, с. 45]. Чек-лист представляет собой обычный список, в него возможно добавление ожидаемых результатов, но это необязательное правило. Для того чтобы чек-лист был действительно полезным инструментом, он должен обладать рядом важных свойств:

1. Логичность - чек-лист пишется на основе целей и для того, чтобы помочь в достижении этих целей.

2. Последовательность и структурированность - достигается за счёт оформления чек-листа в виде многоуровневого списка.

3. Полнота и не избыточность - заключается в том, что чек-лист должен представлять собой аккуратную “сухую выжимку” идей, в которых нет дублирования и в то же время ничто важное не упущено [14, с. 109].

Для проведения тестов должен быть выбран метод подхода к самому тестированию. Существует три таких подхода: метод белого ящика, метод серого ящика и метод черного ящика. Эти подходы во многом схожи, но есть ключевые различия, которые позволяют выбрать и использовать именно тот тип, который более походит для этого [33, с. 21].

Одной из особенностей сервисной шины предприятия является обмен формализованными документами [62], т.е. имеется абсолютно четкое понимание того, какие данные должны быть на ее входе и на выходе. С учетом этой особенности и в целях получения более достоверных результатов [36, с. 23], для тестирования этого программного продукта выбран подход к тестированию по типу черного ящика.

В качестве тестов будут использоваться следующие варианты тестирования: функциональное, функциональное с негативным сценарием и нагрузочное.

Функциональное тестирование подтвердит работоспособность программного продукта [53], негативный сценарий тестирования позволит проверить поведение сервисной шины при его некорректном использовании, а нагрузочное тестирование покажет соответствие требуемым характеристикам.

Такой порядок тестирования выбран по причине того, что проведение тестов требует их четкого и последовательного выполнения. Это связанно с тем, что выполнение части тестов имеют смысл только в случае корректного выполнения предыдущих этапов. Тест на нагрузочное тестирование не предоставит корректной информации в случае, если функциональность

программного продукта некорректна или некорректна реакция тестируемого продукта в случае исключительных ситуаций.

В целях контроля и мониторинга передаваемых данных будет проводиться мониторинг информационного сетевого обмена. Для этого необходим специальный программный инструмент - анализатор сетевого траффика [22].

Программы такого класса используются для визуального отображения содержимого сетевого траффика, которое проходит через узел, на котором оно запущено. Такая информация поможет изучить сетевое взаимодействие между системами и подтвердить корректность проводимых тестов. С учетом специфики информационного обмена, для целей этого тестирования наиболее подходит подтип анализаторов для HTTP обмена. Для этого тестирования выбран HTTP Analyzer V7, поскольку он имеет бесплатную версию и обладает всеми необходимыми возможностями для исследования.

4.1.1 Функциональное тестирование

Функциональное тестирование — это особый вид испытания программного обеспечения, который используется для проверки корректности работы исследуемого программного продукта в рамках заявленной функциональности [23, с. 21].

Для функционального тестирования используется особый набор – функциональные требования. Эти требования описывают поведение системы и ее взаимодействие с внешним окружением [12, с. 38]. Элементами такого тестирования становятся ее обычные действия - вычисления, преобразования, проверки и обработка данных. В контексте проектирования функциональные требования в основном влияют на дизайн системы.

Для сервисной шины предприятия такими требованиями являются – прием сообщения в согласованном формате от клиентского приложения и

передача корректного ответа в соответствии с согласованными форматами документов.

Тестирование будет разбито на две части. В первой его части будет проверен синхронный адаптер SQL, во второй его асинхронный аналог. На вход сервисной шины будет подано сообщение, которое будет содержать корректно составленный запрос, на выходе будут проверены полученные от сервисной шины данные, на соответствие согласованным схемам. В результате теста будет получена структура, информация о которой будет выведена после теста.

Функциональное тестирование с негативным сценарием используется для того, чтобы перед запуском программного продукта в эксплуатацию проверить насколько корректно он обрабатывает исключительные ситуации, в данном случае – вызванные некорректным поведением программных клиентов.

Такое тестирование позволит ответить на вопрос – не повлечет ли использование некорректных данных в реальной работе отказ в обслуживании тестируемого программного продукта.

Для этого будут использованы специально созданные сообщения для сервисной шины предприятия, в которых будет содержаться специально заложенная ошибка и каждое из которых так или иначе не будет подходить под нормальные условия работы, поскольку не будет соответствовать согласованным схемам.

Ошибки будут заложены во входящем сообщении для веб-адаптера, в полезной нагрузке, предназначенной для синхронного адаптера SQL и полезной нагрузке, предназначенной для асинхронного адаптера SQL.

Такой сценарий тестирования позволит проверить качественность разработанного решения в его процедурах обработки ошибок.

В результате правильного прохождения теста будет должно быть получено сообщение, с информацией о корректном прохождении теста и об ошибке, которая будет выведена в текстовом поле после теста.

4.1.2 Нагрузочное тестирование

Нагрузочное тестирование необходимо для проверки работоспособности сервиса под нагрузкой и его потенциальной пропускной способности [44, с. 63].

Это тестирование сможет ответить на вопрос насколько разработанное решение отвечает требованиям и параметрам корпоративной информационной системы, которые были сформированы ранее (таблица 1).

Нагрузочное тестирование будет проводиться с использованием специальных искусственных нагрузочных механизмов, которые будут эмулировать реальную нагрузку на сервисную шину предприятия.

Особое внимание к пропускной способности связано с высокими накладными расходами на работу с формализованными данными и являются серьезной проблемой, влияющей на производительность [13]. Такое тестирование необходимо, поскольку некорректное сообщение может вызвать отказ сервисной шины предприятия и в случае настойчивых запросов со стороны клиентского приложения может полностью остановить работу сервиса.

Имеющиеся универсальные средства тестирования не подходят для этих целей, поскольку сервисная шина предприятия принимает только данные, которые соответствуют ее схемам.

Для проведения этого исследования необходимо создать специальный программный продукт, который будет нагружать сервисную шину как это происходит при активной деятельности клиентов. В данном случае будет использовано отдельное приложение, которое будет передавать данные в необходимом, указанном объеме на вход сервисной шины предприятия.

Для измерения будут использоваться несколько значений. В первую очередь это скорость отклика сервиса – это необходимо для того, чтобы определить, насколько приложенная загрузка влияет на скорость работы

сервисной шины и определить количество возможных запросов в рамках требуемых значений.

Вторым значением для измерений будут ошибки сервиса, в ситуации, когда он не сможет обработать запрос из-за приложенной к нему нагрузки.

Третьим значением для измерения выбран объем сетевого трафика, которым обмениваются системы. Этот показатель также влияет на производительность и скорость обработки.

Таким образом, общая методика исследования представляет из себя формулирование требований для разработки приложения для тестирования, его разработку, тестирование сервисной шины предприятия и анализ полученных данных.

4.2 Проведенные исследования

Добавление в чек листа тестов основывается на анализе, который был сделан в предыдущей главе. В программе Excel была создана таблица, в которой содержатся все выбранные ранее виды тестирования и включенные в тестирование параметры.

Полный список результатов тестирования состоит из следующих элементов:

1. Функциональные тесты синхронного и асинхронного SQL адаптера
2. Функциональный тест веб-адаптера по негативному сценарию
3. Функциональные тесты синхронного и асинхронного SQL адаптера по негативному сценарию
4. Нагрузочное тестирование синхронного и асинхронного адаптера

Для создания программы для тестирования будет использоваться инструмент Microsoft Visual Studio 2019, разработка будет вестись на языке C#.

После компиляции проекта и старта программы для тестирования на локальном компьютере с правами администратора был запущен экземпляр сервисной шины предприятия.

Для исследования информационного обмена был запущен инструмент для контроля сетевого траффика и последовательно были выполнены все виды тестирования.

4.2.1 Функциональное тестирование

Функциональное тестирование было проведено без замечаний, результат тестирования приведен на рисунке 17.

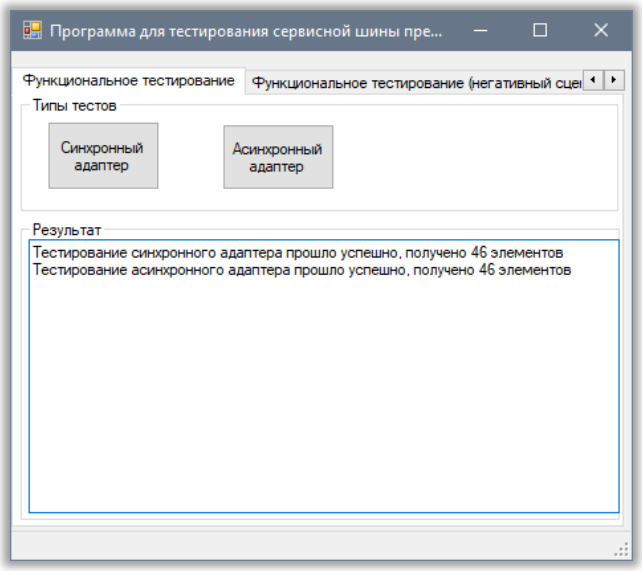

Рисунок 17 - Результат функционального тестирования

Для подтверждения исследования с помощью анализатора был изучен информационный обмен между системами.

На рисунке отображена корректная работа тестового приложения, он состоял из одного метода, клиентского запроса типа POST и ответа сервиса (рисунок 18), указан заголовок Expect с кодом 100, который указывает на ожидание ответа [75].

```
POST / HTTP/1.1
Content-Type: text/xml;charset=UTF-8
Host: localhost:4096
Expect: 100-continue
Connection: Close
Accept-Encoding: gzip, deflate
Content-Length: 488
<BusMessage>
    <MessageId>1c8dc7c3-833a-4eee-964f-5acd5ce847e1</MessageId>
    <MessageType>ce6afb94-e40d-4913-9833-8213779b973f</MessageType>
    <SenderId>715e6267-61a2-4615-9984-b178cc845a41</SenderId>
    <ReceiverId>7d34a3ad-27c0-4dc1-8ac3-a566d1f2141a</ReceiverId>
    <CreatedDate>2020-11-14T16:31:36.9785178+03:00</CreatedDate>
    <Payload>
        <SqlAdapterRequest>
            <Type>1</Type>
            <SqlText>SELECT * FROM dbo.GetPrice('2204')</SqlText>
        </SqlAdapterRequest>
    </Payload>
</BusMessage>
```

Рисунок 18 – Исходящее сетевое сообщение для синхронного SQL адаптера

В ответ тестовый клиент получил формализованный документ (рисунок 19), в котором указан корректный код ответа 200 [49].

В полезной нагрузке сообщения, которая содержится в поле Payload, находится информация, переданная от SQL адаптера, в данном случае прайс лист, содержащий позиции по макаронным изделиям.

```
HTTP/1.1 200 OK
Server: Microsoft-HTTPAPI/2.0
Date: Sat, 14 Nov 2020 13:31:37 GMT
Connection: close
Content-Length: 6753
<ServerResponseMessage>
    <ServerId>ESB v1.0.0.1</ServerId>
    <Status>1</Status>
    <Payload>
        <Root>
            <Item>
                <Barcode>2204012</Barcode>
                <Name>Мак.изделия вермишель в/с корот.25 кг</Name>
                <Price>546.47</Price>
            </Item>
            <Item>
                <Barcode>2204013</Barcode>
                <Name>Мак.изделия вермишель "Любительская" 16кг</Name>
                <Price>378.00</Price>
            </Item>
            <!.. 44 элемента удалено...!>
        </Root>
    </Payload>
</ServerResponseMessage>
```

Рисунок 19 – Ответ на исходящее сетевое сообщение синхронного SQL адаптера

Тест асинхронного адаптера был выполнен корректно. По результатам данных, полученных от сетевого анализатора, была получена структура запросов, отображенная на рисунках 20 и 21.

```
Первый запрос
POST / HTTP/1.1
Content-Type: text/xml;charset=UTF-8
Host: localhost:4096
Expect: 100-continue
Connection: Close
Accept-Encoding: gzip, deflate
Content-Length: 488

<BusMessage>
    <MessageId>7a41a738-40b1-4bc0-ae95-dfa77c41af2d</MessageId>
    <MessageType>a1718868-5cf0-417a-ac85-153a089661c5</MessageType>
    <SenderId>715e6267-61a2-4615-9984-b178cc845a41</SenderId>
    <ReceiverId>651c5612-28b4-450e-910c-6d01e56d0857</ReceiverId>
    <CreatedDate>2020-11-15T16:50:23.4804732+03:00</CreatedDate>
    <Payload>
        <SqlAdapterRequest>
            <Type>1</Type>
            <SqlText>SELECT * FROM dbo.GetPrice('2204')</SqlText>
        </SqlAdapterRequest>
    </Payload>
</BusMessage>

Первый ответ
HTTP/1.1 200 OK
Server: Microsoft-HTTPAPI/2.0
Date: Sun, 15 Nov 2020 13:50:23 GMT
Connection: close
Content-Length: 149

<ServerResponseMessage>
    <ServerId>ESB v1.0.0.1</ServerId>
    <Message>Message sent</Message>
</ServerResponseMessage>
```

Рисунок 20 – Первая часть асинхронного запроса

```
Второй запрос
POST / HTTP/1.1
Content-Type: text/xml;charset=UTF-8
Host: localhost:4096
Expect: 100-continue
Connection: Close
Accept-Encoding: gzip, deflate
Content-Length: 490

<BusMessage>
    <MessageId>7de6c8f3-ad23-4881-9af6-0ce66899a35b</MessageId>
    <MessageType>15855713-fdaf-4c92-b3bb-950776bb5c10</MessageType>
    <SenderId>715e6267-61a2-4615-9984-b178cc845a41</SenderId>
    <ReceiverId>47464ec5-ed72-4f34-b3d6-b7cf0e3d031c</ReceiverId>
    <CreatedDate>2020-11-15T16:50:23.8277797+03:00</CreatedDate>
    <Payload>
        <ResponseAdapterRequest>
            <MessageId>7a41a738-40b1-4bc0-ae95-dfa77c41af2d</MessageId>
        </ResponseAdapterRequest>
    </Payload>
</BusMessage>


Второй ответ
HTTP/1.1 200 OK
Server: Microsoft-HTTPAPI/2.0
Date: Sun, 15 Nov 2020 13:50:23 GMT
Connection: close
Content-Length: 6753

<ServerResponseMessage>
    <ServerId>ESB v1.0.0.1</ServerId>
    <Status>1</Status>
    <Payload>
        <Root>
            <Item>
                <Barcode>2204012</Barcode>
                <Name>Мак.изделия вермишель в/с корот.25 кг</Name>
                <Price>546.47</Price>
            </Item>
            <Item>
                <Barcode>2204013</Barcode>
                <Name>Мак.изделия вермишель "Любительская" 16кг</Name>
                <Price>378.00</Price>
            </Item>
            <!.. 44 элемента уджалено...!>
        </Root>
    </Payload>
</ServerResponseMessage>
```

Рисунок 21 – Вторая часть асинхронного запроса

4.2.2 Функциональное тестирование с негативным сценарием

Функционально тестирование с негативным сценарием было проведено без замечаний, ответ приложения приведен на рисунке 22. Все адаптеры корректно отработали получение документа некорректного формата и передали соответствующий ответ клиенту.

Для производства подачи запросов к сервисной шине использовались специально переназначенные запросы, в которых искажены части запроса, элементы его структуры – XML тэги. Такие запросы не должны проходить корректно процедуру десериализации, поскольку не отвечают структуре объекта и типам его элементов и при попытке вызова программной процедуры вызывает исключение класса InvalidOperationException [74].

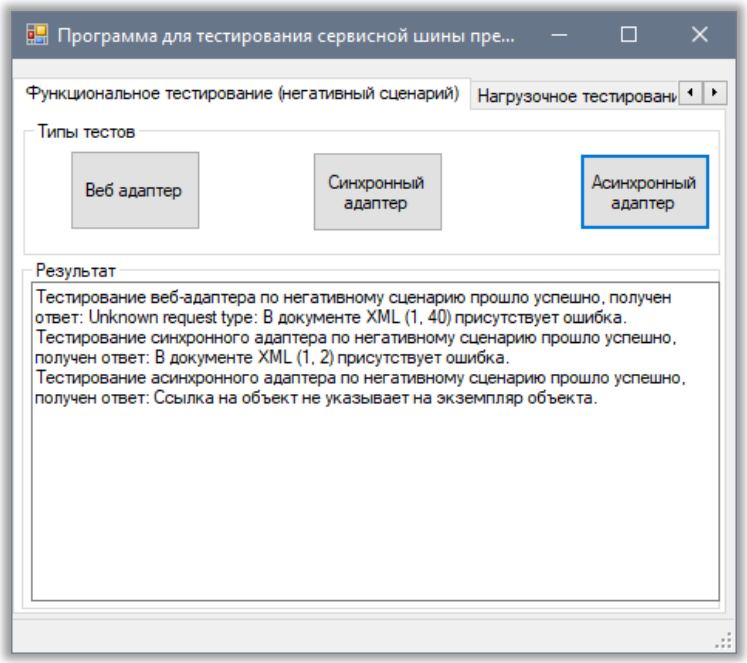

Рисунок 22 - Результат функционального тестирования по негативному сценарию

Информационный обмен, зафиксированный анализатором траффика, отображен на рисунке 23. При тесте веб-адаптера был использовано

некорректное название XML тэга – “WrongBusMessage”. В ответ пришло сообщение с кодом 400, которое обозначает некорректный запрос [76].

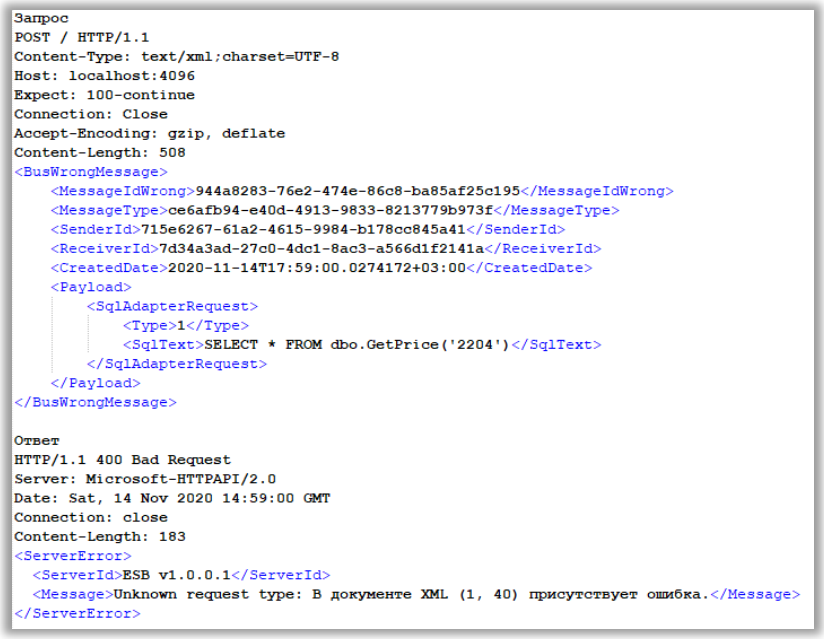

Рисунок 23 – Взаимодействие клиента и веб адаптера при функциональном тестировании с негативным сценарием

Информационный обмен функционального тестирования синхронного SQL адаптера по негативному сценарию отображен на рисунке 24. В этом запросе специально искажено имя тэга полезной нагрузки на “SqlAdapterWrongRequest”. В нем отображен корректный ответ сервера с кодом 200, однако в полезной нагрузке ответа отображено сообщение с информацией об ошибке, что подтверждает корректность проведенного тестирования.

```
Запрос
POST / HTTP/1.1
Content-Type: text/xml;charset=UTF-8
Host: localhost:4096
Expect: 100-continue
Connection: Close
Accept-Encoding: gzip, deflate
Content-Length: 498
<BusMessage>
    <MessageId>0ad094c7-7886-4ac3-9f56-7802ec5273e2</MessageId>
    <MessageType>ce6afb94-e40d-4913-9833-8213779b973f</MessageType>
    <SenderId>715e6267-61a2-4615-9984-b178cc845a41</SenderId>
    <ReceiverId>7d34a3ad-27c0-4dc1-8ac3-a566d1f2141a</ReceiverId>
    <CreatedDate>2020-11-14T18:03:15.4658669+03:00</CreatedDate>
    <Payload>
        <SqlAdapterWrongRequest>
            <Type>1</Type>
            <SqlText>SELECT * FROM dbo.GetPrice('2204')</SqlText>
        </SqlAdapterWrongRequest>
    </Payload>
</BusMessage>

Ответ
HTTP/1.1 200 OK
Server: Microsoft-HTTPAPI/2.0
Date: Sat, 14 Nov 2020 15:03:15 GMT
Connection: close
Content-Length: 263
<ServerResponseMessage>
    <ServerId>ESB v1.0.0.1</ServerId>
    <Status>1</Status>
    <Payload>
        <ErrorResponse>
            <Error>Ссылка на объект не указывает на экземпляр объекта.</Error>
        </ErrorResponse>
    </Payload>
</ServerResponseMessage>
```

Рисунок 24 – Взаимодействие клиента и синхронного SQL адаптера при тестировании с негативным сценарием

Данные функционального тестирования асинхронного SQL адаптера отображены на рисунках 25 и 26. Первая часть взаимодействия, на рисунке 25 отображает запрос клиента к сервисной шине и ее корректный ответ, поскольку сообщение сервисной шины не имеет ошибок, и они заложены ниже, на уровне сообщения адаптера, в объекте полезной нагрузки, имени тэга документа “SqlAdapterWrongRequest”.

```
Первый запрос
POST / HTTP/1.1
Content-Type: text/xml;charset=UTF-8
Host: localhost:4096
Expect: 100-continue
Connection: Close
Accept-Encoding: gzip, deflate
Content-Length: 497
<BusMessage>
    <MessageId>332ae410-363a-42d0-8cec-b308daeb470f</MessageId>
    <MessageType>a1718868-5cf0-417a-ac85-153a089661c5</MessageType>
    <SenderId>715e6267-61a2-4615-9984-b178cc845a41</SenderId>
    <ReceiverId>651c5612-28b4-450e-910c-6d01e56d0857</ReceiverId>
    <CreatedDate>2020-11-14T18:06:59.300755+03:00</CreatedDate>
    <Payload>
        <SqlAdapterWrongRequest>
            <Type>1</Type>
            <SqlText>SELECT * FROM dbo.GetPrice('2204')</SqlText>
        </SqlAdapterWrongRequest>
    </Payload>
</BusMessage>

Первый ответ
HTTP/1.1 200 OK
Server: Microsoft-HTTPAPI/2.0
Date: Sat, 14 Nov 2020 15:06:59 GMT
Connection: close
Content-Length: 149
<ServerResponseMessage>
    <ServerId>ESB v1.0.0.1</ServerId>
    <Message>Message sent</Message>
</ServerResponseMessage>
```

Рисунок 25 – Взаимодействие клиента и сервисной шины при первом запросе функционального тестирования асинхронного SQL адаптера с негативным сценарием

Во второй части взаимодействия, отображённом на рисунке 26, используется запрос ответа от сервисной шины с идентификатором ранее отправленного сообщения. В ответ пришло сообщение с корректным кодом, но содержащее в своей полезной нагрузке сообщение об ошибке. Что свидетельствует о корректной обработке этой исключительной ситуации адаптером.

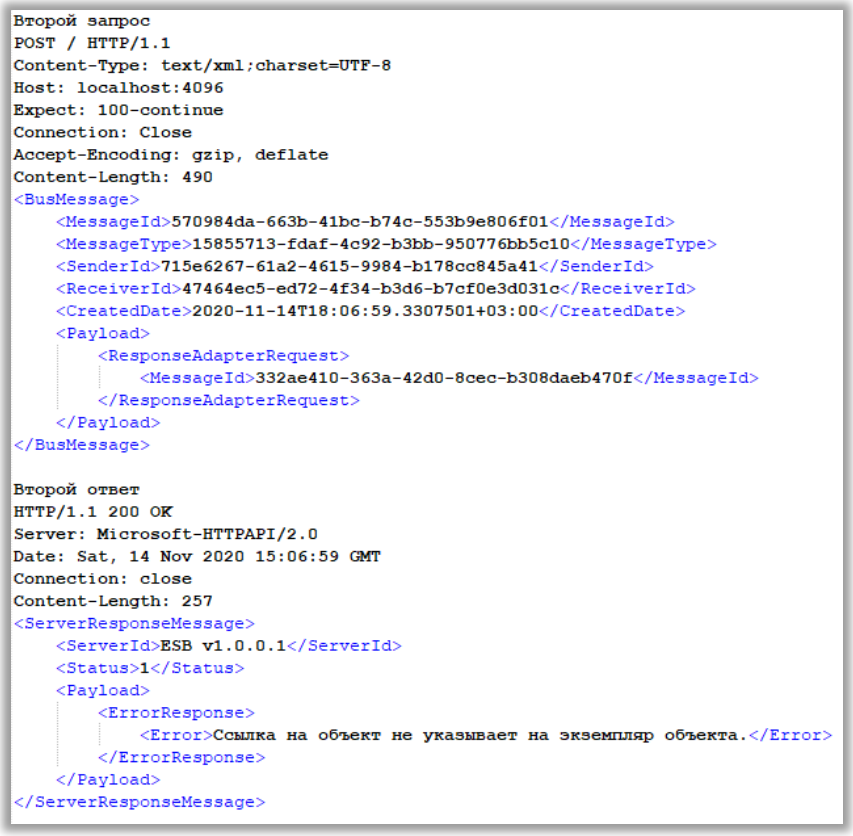

Рисунок 26 – Взаимодействие клиента и сервисной шины при втором запросе функционального тестирования асинхронного SQL адаптера с негативным сценарием

4.2.3 Нагрузочное тестирование

Нагрузочное тестирование проводилось сначала для синхронного SQL адаптера, а потом для асинхронного. После каждого теста, последовательно, увеличивалась нагрузка.

При проведении нагрузочного теста синхронного адаптера, зафиксирована корректная работа адаптера и отсутствие ошибок. Результаты теста с максимальной нагрузкой отображены на рисунке 27.

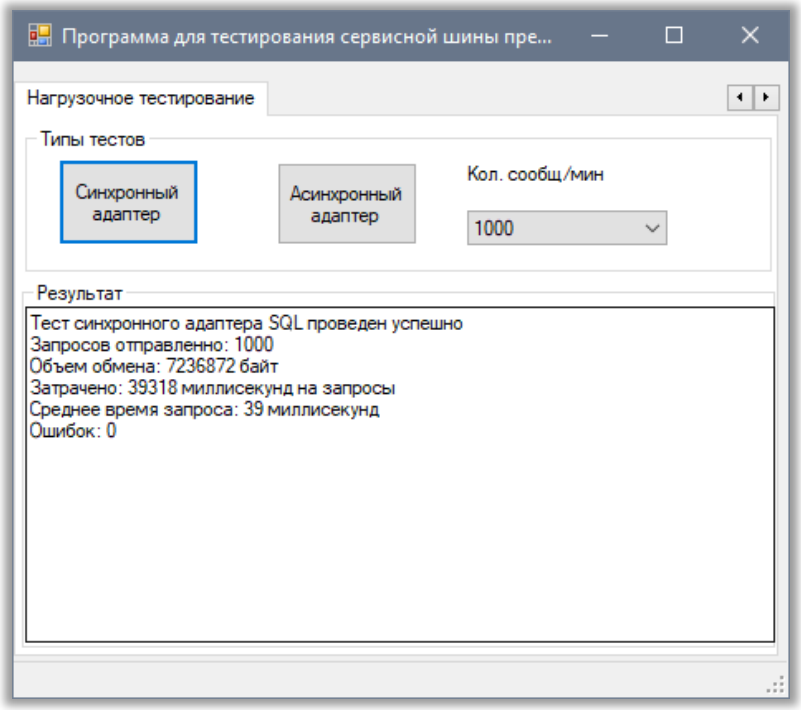

Рисунок 27 – Нагрузочное тестирование синхронного адаптера с нагрузкой 1000 сообщений в минуту

При проведении нагрузочного теста асинхронного адаптера, зафиксирована корректная работа адаптера и полное отсутствие ошибок. Результаты теста с максимальной нагрузкой отображены на рисунке 30.

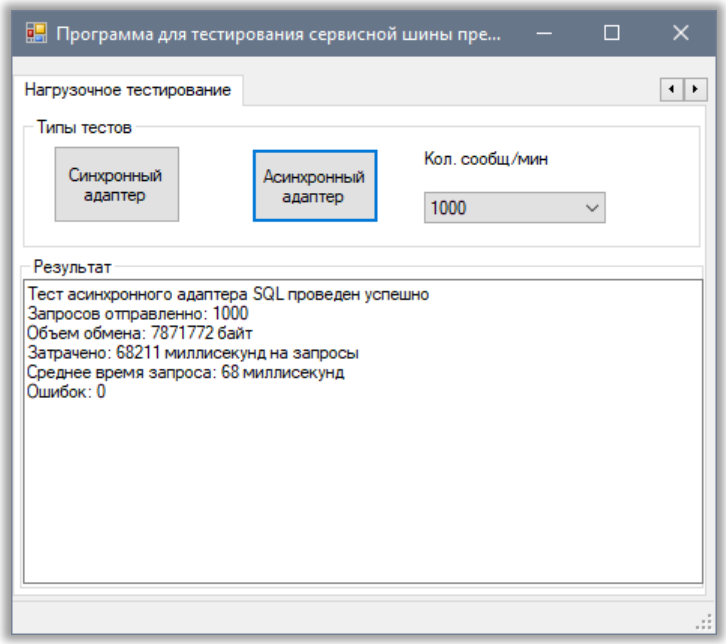

Рисунок 28 – Нагрузочное тестирование асинхронного адаптера с нагрузкой 1000 сообщений в минуту

4.3 Результаты исследования

В результате анализа полученных данных была получены информация, характеризующая качество работы сервисной шины.

Функциональное тестирование прошло без замечаний, все запросы корректно были обработаны и на них получены корректные ответы. Все три функциональных теста с негативным сценарием были обработаны сервисной шиной корректно, ни один из запросов не прошел проверки и на все запросы отправителю было возвращено сообщение, содержащее тест ошибки.

Нагрузочное тестирование показало, что среднее время синхронного запроса почти в два раза меньше, чем среднее время асинхронного. Это связанно с тем, что асинхронный запрос требует двойного обращения к сервисной шине, в то время как синхронному достаточного только одного.

Эта особенность не является признаком некорректной работы сервисной шины, который требует вмешательства в код или изменения программных алгоритмов. В логике асинхронных запросов не требуется оперативность, которая больше присуща синхронным запросам.

Работа с синхронного, как и асинхронного адаптера во всех режимах, кроме максимального, в 1000 сообщений, умещается в установленные временные рамки и отвечает требованиям к сервисной шине предприятия.

Средняя скорость выполнения запроса при использовании синхронного адаптера составляет 39мс, это позволяет выполнить в минуту 1538 запросов. Для асинхронного запроса, с его средней скоростью запроса в 65мс, эта цифра составляет 923 запроса в минуту.

Имеет место факт того, что при выполнении нагрузочного теста с максимальной нагрузкой были превышены временные рамки. Этот момент связан с запросами адаптеров к базе данных и потерями на сетевое взаимодействие. Тысяча запросов при минимальном возможной продолжительности запроса в 63мс будет длиться не менее 63 секунд. С учетом того, что требования по производительности были превышены

сервисной шиной предприятия для синхронного адаптера в 26 раз, а для асинхронного в 15 раз, такой момент не имеет практического значения и может трактоваться как несоответствие изначально превышенным ожиданиям от тестирования.

По результатам исследования трафика, при среднем объеме сообщения в 7500 байт, информационный обмен может составлять до 7МБ в минуту. Такой объем в 221 раз превышает требования к сервисной шине и подтверждает качественность интегрированного решения по этому требованию. После проведения тестирования полученными результатами был заполнен чек-лист (приложение А).

Выводы по четвертой главе

Таким образом, была произведена апробация результатов интеграции на предприятия разработанного программного продукта класса сервисных шин предприятия.

На первом этапе была изучена теоретическая основа для проведения тестирования. Были выбраны конкретные виды тестирования и подход к ним, описание результатов тестирования.

На втором этапе было проведено исследование путем последовательного запуска выбранных ранее тестов. Были получены и обработаны результаты.

На третьем этапе были подведены выводы исследования, был заполнен чек-лист, который стал результатом всей проведенной работы.

По результатам тестирования было подтверждено, что разработанный продукт соответствует предъявленным к нему требованиям и реальная скорость его работы во много раз превышает реальную нагрузку. Принято решение о запуске продукта в коммерческую эксплуатацию, продукт выведен из тестового в продуктивный режим.

Заключение

Все этапы интеграции связующего программного обеспечения – сервисной шины предприятия, с корпоративной информационной системой завершены. В процессе исследования на каждом этапе были получены результаты, которые становились основой для последующих этапов:

1. В теоретической части был сделан анализ явления связующего программного обеспечения, которое может обеспечить предприятие, на котором оно внедряется, возможностью упорядочивания архитектуры используемых программных средств и позволит создавать интеграционные решения. Было проведено исследование существующих решений и сделан вывод о необходимости разработки собственного решения.
2. В аналитической части было сделано исследование корпоративной информационной системы, в которую интегрировалось сервисная шина предприятия. Был сделан выбор анализ подходов к интеграции, проведен этап моделирования и сделан выбор инструментов для разработки.
3. В практической части была проведена разработка сервисной шины предприятия, ее ядра и адаптеров. Была разработана база данных, настроен брокер сообщений RabbitMQ. После этого была проведена отладка и пробный запуск, который подтвердил готовность разработанного продукта к приемочным испытаниям.
4. В финальной четвертой части этого исследования была проведена апробация разработанного продукта помощью приемочных испытаний - автоматизированного тестирования. Была проверена работа сервисной шины в обычном режиме с помощью функционального тестирования, в исключительных режимах с помощью функционального тестирования по негативному сценарию и нагрузочного тестирования. Все этапы тестирования прошли успешно, сервисная шина показала свою работоспособность и соответствие требуемым характеристикам и была запущена в эксплуатацию.

Таким образом была решена практическая задача и подтверждена гипотеза исследования.

Этап интеграции сервисной шины с корпоративной информационной системой предприятия стал первым шагом для перехода предприятия к сервисной и микро-сервисной архитектуре. Дальнейшими шагами исследования станут работы над созданием механизмов аутентификации и авторизации клиентов сервисной шины, предоставление и описание интерфейсов для внешних адаптеров, разработанных сторонними разработчиками в формате динамических библиотек. Изменится методология разработки нового программного обеспечения – все они будут взаимодействовать только через сервисную шину.

Дальнейшие теоретические исследования в этой области могут пойти в направлении развития единой архитектуры корпоративных информационных систем, основанных на принципах сервис-ориентированной архитектуры и связующего программного обеспечения. Довольно важным будет и вопрос стандартизации компонентов таких систем, для возможности разработки программного обеспечения для массового потребителя, которое сможет использовать для взаимодействия с другими компонентами сервисную шину предприятия.

Список используемой литературы

1. ГОСТ Р ИСО/МЭК ТО 12182-2002. ГОСТ Р ИСО/МЭК ТО 12182-2002. Информационная технология. Классификация программных средств. - М.: ИПК Издательство стандартов, 2002.
2. ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств:нац. стандарт Рос. Федерации: изд. офиц.: утв. введ.вдействие подготовлен ФГУП «НИИ Восход» на основе собственного аутентичного перевода на русский язык стандарта,. —Москва: Стандартинформ, 2011. —188 с.
3. Акрамовский Р.Н Автоматизация документооборота на предприятии с помощью брокера сообщений. / Р.Н. Акрамовский // Сборник материалов научно-практических конференций. Дни науки студентов владимирского государственного университета имени Александра Григорьевича и Николая Григорьевича Столетовых. Сборник материалов научно-практических конференций. 2019. – С. 1091-1097.
4. Албахарв Д. C# 6.0. Справочник. Полное описание языка, 6-е изд. / Албахарв Д., Албахари Б.: Пер. с англ. - М.: ООО «И.Д. Вильяме», 2016. - 1040 с.
5. Алонцева Е. Н. Структурное моделирование процессов и систем: учебное пособие по курсу «CASE и CALS технология» / Е. Н. Алонцева, А. Н. Анохин, С. П. Саакян. – Обнинск: ИАТЭ НИЯУ МИФИ, 2015. – 72 с.
6. Арапчор Т. А. Технологии баз данных в Delphi 10. / Т. А. Арапчор, С. М. Далаа // Научные труды тувинского государственного университета. Сборник материалов ежегодной научно-практической конференции преподавателей, сотрудников и аспирантов ТувГУ. - 2018. - С36-38.
7. Бен-Ган И. Microsoft SQL Server 2012. Основы T-SQL / И. Бен-Ган. – Москва: Эксмо, 2015. – 400 с.

8. Ваняшин С.В. Методы моделирования и оптимизации: учеб. пособие. / С.В. Ваняшин. Самара: ФГБОУ ВО ПГУТИ, 2017. 83 с.

9. Дворянкин, А. М. Основные методы тестирования программного обеспечения: учеб. пособие / А. М. Дворянкин, А. А. Ерофеев, А. В. Аникин; ВолгГТУ. – Волгоград, 2015. – 120 с.

10. Каролло Д. Как тестируют в Google / Д. Каролло — «Питер», 2012.

11. Ковалев Р. Использование концепций SOA в оптимизации транспортных систем предприятий / Р. Ковалев // Herald of the ural state university of railway transport. 2009, Номер: 1-2. Уральский государственный университет путей сообщения. 2009. – С. 29-33.

12. Котляров В.П. Основы тестирования программного обеспечения / В.П. Котляров. - М.: Национальный Открытый Университет «Интуит», 2016. – 349 с.

13. Кравцов А.А. Алгоритм автоматической сериализации со сжатием ё/ А.А. Кравцов, А.А. Тропченко // Международный научно-исследовательский журнал. -2016. №12(54), часть 3. - С. 111-114.

14. Куликов С. Тестирование программного обеспечения. Базовый курс. / С. Куликов. - М: EPAM Systems, 2017. - 295 с

15. Леоненков А. В. Самоучитель UML 2 / А. В. Леоненков — СПб.: БХВ-Петербург, 2007. — 576 с.

16. Леонов Д.Г. Сравнительный анализ способов организации хранения информационных объектов с динамической структурой. / Д.Г. Леонов // Информационные технологии и вычислительные системы. - 2015. №1. – С. 83-90.

17. Москвичева К.С., Долгачев М.В. Бумажная промышленность: Kafka против RabbitMQ. Сравнительное исследование двух отраслевых эталонных реализаций publish/subscribe / К.С. Москвичева, М.В. Долгачев // Форум молодёжной науки, Выпуск 1, № 4. 2020. - С. 3-17.

18. Нуриддинова М.Ш. Информационная система в среде Embarcadero Rad Studio // Международный научный журнал «Символ науки» №6/2016. 2016. С. 90-91.

19. Осетрова И.С. Разработка баз данных в MS SQL Server 2014 / И.С. Осетрова. - СПб: Университет ИТМО, 2016. – 114 с.

20. Попова, Ю. Б. Тестирование и отладка программного обеспечения: пособие / Ю. Б. Попова. – Минск: БНТУ, 2020. – 66 с.

21. Потапенко В.Я. Тестирование и отладка программного обеспечения. Методические указания для практических занятий. / В.Я. Потапенко. Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова. - Новочеркасск: ЮРГПУ (НПИ), 2016. – 40 с.

22. Ромашкин Д.О. Анализ сетевого трафика. Снифферы. / Д.О. Ромашкин, Е.В. Жилина // Новая наука: от идеи к результату. - 2017. - № 2. - С. 146-150.

23. Савин Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. — М.: Дело, 2007. — 312 с.

24. Соболев П.Ю. Использование языка программирования assembler в языках программирования высокого уровня / П.Ю. Соболев, Е.С. Филенко // Научные перспективы XXI века, материалы Международной (заочной) научно-практической конференции. 2018 С. 105-110.

25. Сорокин Использование JSON в современной разработке высокие технологии / А. А. Сорокин, Р. А. Коваленко, Е. А. Яковлева // Наука и образование: актуальные вопросы, достижения и инновации. Сборник статей VIII Всероссийской научно-практической конференции. 2020. – С. 43-45.

26. Старушенкова Е. Е. Язык SQL как средство создания баз данных / Е. Е. Старушенкова, Е. Н. Шиманова, К.Д. Радаев // E-SCIO Издательство: Информационная Мордовия (Саранск) // Номер: 3 (42). 2020. – С. 325-330.

27. Тепляков С. Паттерны программирования на платформе .NET. / Тепляков С. -СПб.: Питер, 2018. -320 с.

28. Троелсен Э. Язык программирования C# 7 и платформы .NET и .NET Core, 8-е изд. / Э. Троелсен, Ф. Джепикс.: Пер. с англ. — СПб.: ООО “Диалектика”, 2018 — 1328 с.

29. Шарп Д. Microsoft Visual C#. Подробное руководство. 8-е изд. / Д. Шарп. — СПб.: Питер, 2017. — 848 с.:

30. Agafonov E. Multithreading with C# Cookbook. Second Edition / E. Agafonov. Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. 2015. – 264 p.

31. Agafonov E. Mastering C# Concurrency / E. Agafonov, A. Koryavchenko. Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. 2015. -285 p.

32. Aley R. PHP Beyond the Web / R. Aley. Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. 2016. -214 p.

33. Ammann P. Introduction to software testing / P. Ammann, J. Offutt - Cambridge University Press. The Edinburgh Building, Cambridge CB2 8RU, UK. 2008. – 346 p.

34. Ayanoglu E. Mastering RabbitMQ. /Yusif A., Nahum D. Packt Publishing. 2015. – 164 p.

35. Cooling J. Modelling software with pictures / J. Cooling. Lindentree Associates. 2015. - 269 p.

36. Copeland L. A Practitioner's Guide to Software Test Design / L. Copeland - Artech House Publishers. Boston, London. Library of Congress and British CIP. Norwood, MA 02062. 2004. – 358 p.

37. Davidson L. Pro SQL Server Relational Database Design and Implementation / L. Davidson, J. Moss. Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. 2016. - 312 p.

38. Gaddis T. Starting out with Visual C#, Fourth Edition / T. Gaddis. Pearson Education, Inc. 2017. – 793 p.

39. Hogan R. A Practical Guide to Database Design Second Edition / R. Hogan. CRC Press Taylor & Francis Group. 6000 Broken Sound Parkway NW, Suite 300 Boca Raton. 2018. – 431 p.

40. Johansson L. The Optimal RabbitMQ Guide / L. Johansson, E. Vinka. 84codes AB. 2020. – 138 p.

41. Josuttis M. SOA in Practice, 2007.: O'Reilly Media, Inc. 1005 Gravenstein Highway North, Sebastopol, CA 95472. 2007. – 344p.

42. Kevin C. JBoss ESB Beginner's Guide / Kevin Conner, Tom Cunningham, Len DiMaggio, Magesh Kumar B, 2012: Packt Publishing, Livery Place 35 Livery Street Birmingham B3 2PB, UK. – 320 p.

43. Mendoza J. Developing Windows Services Succinctly / J. Mendosa. Syncfusion Inc. 2501 Aerial Center Parkway Suite 200 Morrisville, NC 27560 USA All rights reserved. 2015. – 78 p.

44. Nayyar A. Instant Approach to Software Testing. Principles, Applications, Techniques and Practices. / A. Nayyar - BPB Publications, India. 2019. – 137 p.

45. Rossel S. SQL Server for C# Developers Succinctly. / S. Rossel. Syncfusion, Inc. 2501 Aerial Center Parkway Suite 200 Morrisville, NC 27560 USA. 2016. – 96 p.

46. Software engineering, report on a conference sponsored by the NATO SCIENCE COMMITTEE Garmisch, Germany, 7th to 11th October 1968.

47. Strate J. Expert Performance Indexing in SQL Server 2019: Toward Faster Results and Lower Maintenance / J. Strate. Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. 2019. – 630p.

48. Weisfeld M. The Object-Oriented Thought Process Fifth Edition / M. Weisfeld. Pearson Education, Inc. 2019. - 198 p.

49. Веб-технологии для разработчиков, коды ответа HTTP. -2020. – URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Status (дата обращения 15.11.2020).

50. Галактика ESB. -2020. – URL: https://galaktika.ru/esb (дата обращения 15.11.2020).

51. Диаграмма классов (Wikipedia). -2020. – URL: https://ru.wikipedia.org/wiki/Диаграмма_классов (дата обращения 23.11.2020).

52. Класс HttpListener. -2020. – URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.net.httplistener (дата обращения 27.11.2020).

53. Методология тестирования программного обеспечения: сайт. - 2020. - URL: https://scienceforum.ru/2017/article/2017033412 (дата обращения 15.11.2020).

54. Нотация и семантика языка UML. - 2020. – URL: https://intuit.ru/studies/courses/32/32/lecture/1004 (дата обращения 23.11.2020).

55. Основные принципы .NET. Управляемая поточность. -2020. – URL: https://docs.microsoft.com/ru-ru/dotnet/standard/threading/threads-and-threading (дата обращения 27.11.2020).

56. Приложение SQL Server Profiler. - 2020. – URL: https://docs.microsoft.com/ru-ru/sql/tools/sql-server-profiler/sql-server-profiler (дата обращения 23.11.2020).

57. Проектирование информационных систем. Диаграммы вариантов использования. - 2007. – URL: https://www.sites.google.com/site/anisimovkhv/learning/pris/lecture/tema12/tema12_2 (дата обращения 23.11.2020).

58. Работа с базами данных, курс лекций ИНТУИТ. Лекция 4: Проектирование баз данных. -2020. – URL: https://intuit.ru/studies/courses/3439/681/lecture/14021 (дата обращения 27.11.2020).

59. Разработка приложений служб Windows. -2020. – URL: https://docs.microsoft.com/ru-ru/dotnet/framework/windows-services/introduction-to-windows-service-applications (дата обращения 27.11.2020).

60. Сервисная шина предприятия. -2020. – URL: https://ru.wikipedia.org/wiki/Сервисная_шина_предприятия (дата обращения 04.10.2020).

61. Система «Галактика ERP». Описание функциональности системы. - 2017. – URL: http://galaktika.ru/docs/ERP_about.pdf (дата обращения 23.11.2020).

62. Создание бизнес-процесса с помощью инструментов Rational и WebSphere. Лекция 9: Реализация. Создание корпоративной сервисной шины. -2020. – URL: https://intuit.ru/studies/courses/1152/258/lecture/6597 (дата обращения 15.11.2020).

63. Структура Guid. -2020. – URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.guid (дата обращения 27.11.2020).

64. Сценарии и решения использования шины Enterprise Service Bus в сервис-ориентированной архитектуре. -2020. – URL: https://www.ibm.com/developerworks/ru/library/ws-esbscen (дата обращения 04.10.2020).

65. Уровни моделирования (проектирования) БД. - 2016. – URL: https://infopedia.su/1x61fe.html (дата обращения 23.11.2020).

66. Функциональные возможности DATAREON ESB. -2020. – URL: https://www.datareon.ru/products/esb-servisnaya-shina-dannykh/integracionnaya-shina-dannyh-funktsionalnye-vozmozhnosti (дата обращения 04.10.2020).

67. A modern ESB for the digital era. -2020. – URL: https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb (дата обращения 04.10.2020).

68. Gartner Says Worldwide Application Infrastructure and Middleware Market Grew 12 Percent in 2017. -2020. – URL: https://www.gartner.com/en/newsroom/press-releases/2018-06-04-gartner-says-worldwide-application-infrastructure-and-middleware-market-grew-12-percent-in-2017 (дата обращения 04.10.2020).

69. Get Started with RabbitMQ on Docker. -2020. – URL: https://codeburst.io/get-started-with-rabbitmq-on-docker-4428d7f6e46b (дата обращения 20.12.2020).

70. IBM Integration Bus. -2020. – URL: https://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/bb43020_.htm (дата обращения 04.10.2020).

71. Introduction to Oracle Enterprise Service Bus. -2020. – URL: https://docs.oracle.com/cd/E11036_01/doc.1013/e10295/esb_intro.htm (дата обращения 04.10.2020).

72. Mediator ESB. -2020. – URL: https://dasystems.ru/mediator.html (дата обращения 04.10.2020).

73. Microsoft BizTalk и интеграционные решения. -2020. – URL: https://partner.microsoft.com/ru-ru/solutions/microsoft-biztalk-server (дата обращения 04.10.2020).

74. Microsoft .NET, XmlSerializer.Deserialize. -2020. – URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.xml.serialization.xmlserializer.deserialize (дата обращения 15.11.2020).

75. RFC7231, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. -2014. – URL: https://tools.ietf.org/html/rfc7231 (дата обращения 15.11.2020).

76. Yandex, справочник по кодам статуса HTTP. -2020. – URL: https://yandex.ru/support/webmaster/error-dictionary/http-codes.html (дата обращения 15.11.2020).

ПРИЛОЖЕНИЕ А. Таблица соответствия функциональности импортных решений

Функциональное тестирование	
	Результат теста
Синхронный SQL адаптер	Тестирование синхронного адаптера прошло успешно
Асинхронный SQL адаптер	Тестирование асинхронного адаптера прошло успешно

Функциональное тестирование с негативным результатом	
	Результат теста
Веб адаптер	Тестирование веб-адаптера по негативному сценарию прошло успешно
Синхронный SQL адаптер	Тестирование синхронного адаптера по негативному сценарию прошло успешно,
Асинхронный SQL адаптер	Тестирование асинхронного адаптера по негативному сценарию прошло успешно

Нагрузочное тестирование, синхронный SQL адаптер			
Сообщений	10	100	1000
Ошибок	0	0	0
Среднее время запроса (мс)	39	39	39
Объем трафика	72368	72693	7236872

Нагрузочное тестирование, асинхронный SQL адаптер			
Сообщений	10	100	1000
Ошибок	0	0	0
Среднее время запроса (мс)	64	63	68
Объем трафика	78719	787186	7871772