Санкт-Петербургский государственный университет

Кафедра технологии программирования

Матлаш Александр Николаевич

Магистерская диссертация

Проектирование, разработка и внедрение API для информационных систем и сервисов СПбГУ

Направление 02.04.02

Фундаментальная информатика и информационные технологии

Магистерская программа «Технологии баз данных»

Научный руководитель,

ст. преподаватель

Севрюков С.Ю.

Оглавление

Введение

В настоящее время довольно сложно представить себе работу организации, которая не задействовала бы информационные технологии для автоматизации и оптимизации различного рода бизнес-процессов.

Если говорить о Санкт-Петербургском государственном университете (СПбГУ) , рассматривая эволюцию развития правил обучения в период с 1991 года и по настоящее время, то можно сказать, что он, как организация, прошел достаточно сложный и неоднозначный путь.

На настоящее время в СПбГУ, только в рамках информационной системы “Обучение” насчитывается более десяти самостоятельных комплексных приложений, каждое из которых предназначено для выполнения некоторого обособленного набор задач. В свою очередь, упомянутые приложения сами могут состоять из множества различных условно независимых компонент.

Например, в рамках ИС “Обучение” на момент написания работы в эксплуатации находятся такие подсистемы как “Обучающиеся” (“Карточка студента” “Дисциплины”, “Учебные планы”, “Педагогические поручения”, ИС “Выпускник”, или же такие, более известные для абитуриентов и студентов СПбГУ как “Прием”, или “Электронное расписание”.

Стоит отметить, что несмотря, на первый взгляд, на достаточно разные задачи, зачастую, для реализации какого-либо бизнес-процесса, необходимо обеспечить взаимодействие некоторого набора подсистем друг с другом. Например, фиксация зачисления абитуриента в СПбГУ осуществляется в рамках подсистемы “Прием”, а основной учет студенческих данных (номер курса, группы, осуществление процессов перевода, ведение успеваемости и т.п.) в подсистеме “Обучающиеся”. Соответственно, каждый год, после окончания приемной кампании, возникает задача импорта данных о зачисленных студентах из одной подсистемы в другую.

Описанный случай достаточно тривиален, однако, в действительности, существуют и более сложные процессы, которые могут задействовать более двух систем, требовать реализации синхронизации данных в разных направлениях (из одной системы в другую и обратно) и при этом обеспечивать допустимый показатель задержки синхронизации. Реализацию такого рода механизмов взаимодействия между системами будем называть интеграцией, а бизнес-процесс, для осуществления которого требуется обеспечение взаимодействия - интеграционным сценарием.

Существует множество технических способов реализации механизмов взаимодействия между системами. Анализ такого рода способов, с подробным описанием достоинств и недостатков каждого из них, как правило, можно найти в литературе, посвященной вопросам интеграции корпоративных приложений. Каждый из авторов может отдавать предпочтение и уделять большее внимание какому-то одному из существующих способов интеграции, однако, все специалисты, как правило, сходятся в одном - любая реализация интеграционной логики всегда повышает сложность поддержки информационной системы.

Стоит отметить, что на момент написания данной работы, автор являлся сотрудником Управления-Службы Информационных технологий СПбГУ, в обязанности которого входила разработка и поддержка некоторого набора из озвученных ранее подсистем ИС СПбГУ. В рамках выполнения своих должностных обязанностей, автору пришлось столкнуться со всем многообразием существующих в организации систем и интеграционных решений. Можно говорить о том, что сделанные в ходе работы наблюдения и заметки относительно существующих технологических решений послужили началом написания данной диссертации.

Был обнаружен ряд недостатков, присущих существующим процессам и механизмам взаимодействия между системами. В свою очередь, можно

говорить о том, что эти недостатки вели к снижению эффективности поддержки, разработки, и, что самое главное, эксплуатации (то есть удобства для конечных пользователей) существующих приложений.

Анализу такого рода проблем, выявлению недостатков и выработке путей улучшения существующих систем и интеграционных процессов между ними посвящена данная работа.

Обзор литературы

Все многообразие исследованных источников можно разделить на определенные категории, каждая из которых имеет сопоставление с некоторым набором сформулированных ранее задач.

Так, в рамках данной работы, прежде всего, ставится задача оценки существующих интеграционных решений, направленных на обеспечение взаимодействия между компонентами ИС СПбГУ. Стоит отметить, что задача осуществления подобного анализа и классификации интеграционных решений не является уникальной. Так, освещению данных вопросов посвящены некоторые фундаментальные труды [1,2]. Авторы в подробной форме описывают существующие способы интеграции приложений, предоставляют оценку достоинств и недостатков, сопровождая анализ практическими примерами. Материал данных источников был взял за основу для анализа интеграционных решений ИС ВУЗа, представленного далее в этой работе.

Зачастую, результатом разработки интеграционного решения является сложная распределенная система, состоящая из некоторого набора взаимодействующих компонентов. Детали изложены в рамках практических разделов, посвященных анализу и модификации интеграционных решений между подсистемой “Обучающиеся” и библиотечным модулем, или подсистемой “Обучающиеся” и СЭДД “Дело”. Стоит отметить, что способы построения любой распределенной системы могут значительно отличаться. В зависимости от выбранного подхода, полученное решение будет обладать собственным набором достоинств и недостатков. В рамках диссертации были изучены источники, посвященные анализу упомянутых задач. [3,4]. Отдельно стоит упомянуть вопросы, связанные с определением последовательности

возникновения событий в рамках различных компонентов сложной распределенной системы, изучение которых осуществлялось отдельно [4,5].

При реализации сложного многокомпонентного интеграционного решения, особое внимание следует уделить изучению практик и инструментов, направленных на упрощение и ускорение процессов тестирования полученных результатов и их развертывания на различном окружении. Решению данных вопросов способствуют такие практики как Continuous Integration и Continuous Delivery, использование которых стало возможным благодаря изучению некоторого набора источников [6,7,8,9].

Также не менее важным является использование инструментов для оркестровки, мониторинга и сбора ошибок, с целью оперативного обнаружения и реагирования на возникающие проблемы в ходе работы сложного многокомпонентного решения [10,11].

Другие упомянутые в работе источники можно рассматривать как некоторый набор практических примеров реализации тех или иных интеграционных решений. Целью изучения данных работ являлось заимствование полученного авторами опыта, и ознакомления с используемым для решения задач интеграции инструментальным стеком технологий [12,13,14,15].

Цели и задачи

Конечной целью данной работы является минимизация ресурсов, затрачиваемых на поддержку, эксплуатацию и разработку сложной многокомпонентной информационной системы ВУЗа.

Достижение такого рода цели подразумевает выполнение следующего набора задач:

1. Осуществить сбор информации:
    - Собрать информацию о ключевых компонентах, используемых в рамках ИС СПбГУ.
    - Зафиксировать информацию об основных сценариях взаимодействия между ИС ВУЗа. Ознакомиться с их технической реализацией.
    - Выявить основные классы проблем, характерных для текущих способов интеграции.
    - Оценить последствия выявленных недостатков. Выделить наиболее критические сценарии. Рассмотреть вопрос о целесообразности их модификации с целью улучшения.
2. Сформировать основу для оценки существующих интеграционных решений путем изучения литературы, посвященной вопросам интеграции корпоративных приложений. Оценить возможность улучшения и/или переиспользования существующих интеграционных решений для снижения стоимости разработки, внедрения и поддержки компонентов ИС.
3. Изучить и апробировать средства категории devops, направленные на упрощение процесса тестирования и развертывания сложных многокомпонентных решений в рамках различного окружения.

4. Осуществить модификацию выделенных в рамках пункта №1-d интеграционных механизмов на основе проанализированной в рамках пункта №2 информации. В качестве критериев улучшения можно рассматривать:
    - Достижение экономии ресурсов на поддержку и эксплуатацию полученного решения в сравнении с исходным вариантом.
    - Возможность переиспользования полученных результатов для решения задач интеграции иных наборов компонентов ИС ВУЗа.

    В рамках данной задачи можно выделить следующие подзадачи:

    - Описать бизнес-процессы, на обеспечение выполнения которых направлено интеграционное решение.
    - Описать и оценить исходное интеграционное решение. Выделить ключевые преимущества и недостатки.
    - Предложить пути улучшения. Осуществить и описать процесс реализации.
5. Проанализировать полученные результаты.
6. Определить возможность распространения полученного решения на иные интеграционные сценарии. Оценить возможные трудозатраты на переиспользование.

Глава 1. Анализ и оценка качества интеграции компонентов ИС СПбГУ

В рамках данного раздела будет представлен анализ существующих интеграционных решений, используемых в ИС СПбГУ и обеспечивающих возможность осуществления бизнес-процессов, требующих вовлечения в процесс нескольких компонентов информационной системы.

Ввиду большого количества компонентов информационной системы и широкого многообразия интеграционных решений, обзор будет построен по принципу “от общего к частному”. Все многообразие интеграционных решений будет разделено на 4 основных класса, будет представлен обзор ключевых идей, заложенных в основе каждого из них, будет осуществлена оценка достоинств и недостатков того или иного подхода. Описание каждого из классов будет подкреплено примерами интеграционных решений, реализованных и используемых в рамках взаимодействия компонентов ИС организации на момент выполнения этой работы.

Стоит отметить, что основной целью данного раздела являются демонстрация многообразия и разнородности используемых интеграционных решений, выявление основных недостатков (или достоинств), присущих тем или иным решениям, ознакомление с возможными альтернативами, поиск и оценка способов улучшения (в случае необходимости) текущих решений с целью увеличения эффективности эксплуатации, поддержки и разработки существующих компонентов информационной системы.

Прежде всего, перед непосредственно анализом текущих решений, автором был осуществлен сбор информации, касающейся общего количества используемых компонентов информационной системы “Обучение”, функциональности, предоставляемой каждой из её компонентов, сценариев взаимодействия между ними и того, с помощью каких подходов и средств

данные сценарии реализованы. В ходе выполнения данной работы было задействовано множество источников, среди которых, в качестве примера, можно выделить такие как история и описание commit-ов в репозиторий, закрытые и открытые задачи в task-менеджере, информация из корпоративной базы знаний, встречи и переписка с сотрудниками УСИТ СПбГУ. С целью систематизации полученной информации, в качестве одного из результатов проделанной работы, была создана схема, иллюстративно изображающая основные компоненты ИС, сценарии их взаимодействия между собой, а также паттерны и средства, используемые для осуществления данного взаимодействия. Данная схема на момент написания работы включает в себя 11 компонентов подсистемы “Обучающиеся” и около 15 сценариев взаимодействия систем со схематичным указанием используемого в ходе интеграции стека технологий. Схема находится в режиме активного использования и дополнения. Ввиду объемности схемы автором было принято решение отказаться от ее прямого размещения в работе, но предоставить ссылку, по которой с ней можно ознакомиться [16].

Прежде чем перейти к дальнейшей части работы, уточним значение используемого определение интеграции. В ходе работы автором было найдено множество определений, каждое из которых обладало разной степенью точности и детализации. В контексте данной работы, в качестве общего определения интеграции следует понимать обеспечение взаимодействия между множеством программ, выполняющихся под управлением различных платформ и расположенных в различных местах предприятия [1]. Более того, исходя из собранной информации о текущих сценариях взаимодействия между приложениями, можно говорить о том, что наиболее востребованными типами интеграции являются т.н. интеграция данных между корпоративными приложениями (Enterprise Application Integration, EAI) и интеграция бизнес-процессов (Business Process

Improvement, BPI) [17].

Итак, опираясь на выше представленную схему, проанализируем существующие интеграционные решения путем их разбиения на ряд ключевых категорий. В некоторых источниках, выделяются, по меньшей мере, 4 способа интеграции приложений, а именно, это передача файлов между приложениями, использования общей базы данных, вызов удаленной процедуры и обмен сообщениями [1]. Рассмотрим каждый из них в отдельности.

1.1. Передача файла

Использование общих данных осуществляется путем передачи и/или совместного использования файла интегрируемыми системами.

В качестве примера такого рода способа интеграции можно привести механизм взаимодействия между подсистемой “Обучающиеся” и СЭД “Дело” в рамках бизнес-процесса создания и/или редактирования проекта приказа, или механизм импорта сведений о поступивших студентах из ИС “Прием” в подсистему “Обучающиеся”.

Однако, стоит отметить, что в [1] под экспортом и импортом автор подразумевает некоторые алгоритмы, реализованные в интегрируемых системах, выполнение которых не требует вмешательства разработчиков и происходит в автоматическом режиме.

В рамках взаимодействия приложений ИС СПбГУ зачастую экспорт и импорт производится вручную разработчиками соответствующих приложений путем формирования запросов к целевым БД.

1.2. Общая база данных

Взаимодействие осуществляется с помощью базы данных, в которой сохраняется общая информация.

Стоит отметить, что относительно ИС СПбГУ, данный интеграционных паттерн чаще всего применяется в рамках создания и поддержки систем, состоящих из нескольких модулей, предназначенных для разных участников одних и тех же или смежных бизнес-процессов, и разработка которых осуществлялась и осуществляется одной обособленной командой.

В качестве примера использования данного подхода можно привести подсистему “Обучающиеся”, включающую в себя web-приложение “Личный кабинет обучающегося” и настольный модуль “Карточка студента”, или же модули подсистемы “Библиотека” (ЭБИС) (Web-расписание, планировщик). Данные примеры относятся к случаям использования одной базы данных приложениями, в целом, входящими в общую “подсистему” и выполняющими единый набор задач. Однако, существуют примеры использования единой базы данных несколькими приложениями, выполняющими в целом совершенно разные задачи, разрабатываемыми и поддерживаемыми разными группами разработчиков. В качестве примера такого рода сценариев использования общей базы данных можно привести процесс формирования ведомостей в модуле “Карточка студента”, который, для выполнения данной задачи, запрашивает данные о дисциплинах напрямую из базы данных электронного расписания СПбГУ (“TimeTable”).

Стоит отметить, что факт поддержки и разработки упомянутых систем разными командами, в целом обостряет достаточно классическую для данного интеграционного подхода проблему отсутствия инкапсуляции при использовании общей базы данных. Иными словами, увеличивается вероятность возникновения ошибки в одном из приложений по причине изменения схемы, или непосредственно самих общих данных другим приложением. Таким образом, можно говорить о том, что общая база данных является примером подхода к интеграции, затрудняющим поддержку

работающих с этой базой данных приложений. Помимо взаимодействия настольного модуля “Карточка студента” и базы данных “TimeTable”, данная проблема особенно отчетливо проявляется при поддержке “Карточки студента” и “Личного кабинета обучающегося”

Помимо того что указанные приложения используют общую базу данных, общей для них является также единая объектно-ориентированная модель, являющаяся результатом объектно-ориентированного отображения реляционной схемы данных на объектно-ориентированный язык.

Говоря о плюсах данного подхода, с одной стороны, можно подчеркнуть тот факт, что общее хранилище и модель данных упрощают процедуру добавления нового функционала или внесения изменений, общих как для “Карточки студента”, так и для “Личного кабинета обучающегося”. К примеру, при реализации возможности выбора студентом темы ВКР из “Личного кабинета обучающегося”, необходимо реализовать механизм, который бы позволял сотрудникам учебных отделов просматривать выбранные студентами темы с использованием ПО “Карточка студента”. В данном случае оба приложения используют одни и те же данные и изначальное сохранение информации о выборе студентом темы в ином, отличном от БД “Карточки студента” хранилище, потребовало бы реализации дополнительной логики для интеграции источников данных.

При этом использование одной и той же сборки (в терминах .NET платформы), содержащей модель данных, избавляет разработчиков от необходимости внесения изменений, связанных с моделью данных и затрагивающих оба приложения, одновременно в несколько мест.

С другой стороны, при использовании общей базы данных следует принимать во внимание характерный для данного подхода недостаток, заключающийся в возможности резкого увеличения нагрузки на базу, которая, в таком случае, может стать “узким горлышком” для интегрируемых

систем с точки зрения производительности. Это особенно актуально для случая, если одно из приложений может использоваться широким кругом пользователей и характеризуется наличием моментов резкого возрастания нагрузки на приложение. В качестве примера можно привести “Личный кабинет обучающегося”, являющийся веб-приложением, работающим с базой данных как в режиме чтения (предоставление студентам информации), так и записи (прием студенческих заявок). Поскольку период приема тех или иных заявок ограничен, то в ходе эксплуатации “Личный кабинет обучающегося” наблюдается возрастание нагрузки (увеличение количества запросов) на базу данных в последние дни этого периода, что может привести в некоторых случаях к уменьшению производительности как приложения “Личный кабинет”, так и модуля “Карточка студента”, используемого сотрудниками различных отделов университета, или к появлению ошибок в каждом из них.

Еще одним сценарием интеграции, реализация которого подразумевает использование общей базы данных, может являться процесс формирования выпускных документов, выполнение которого осуществляется с использованием модуля “Выпускник”. Однако, для выполнения этой задачи в модуле должны быть доступны данные об успеваемости студентов, учет которых ведется в “Карточке студента”. В качестве способа предоставления данных об успеваемости было выбрано решение, заключающееся в создании представления (view) на уровне базы данных “Карточки студента” с выдачей соответствующих разрешений на вызов данного представления. Представленное решение работает в режиме “только на чтение”, на вход принимает в качестве параметра логин студента и выдает список имеющихся оценок. Вызов данного представления осуществляется из кода модуля “Выпускник” и позволяет по требованию импортировать (скопировать) данные об оценках конкретного студента из БД “Карточки студента” с возможностью внесения в эти данные последующих правок. Однако,

механизм обратной синхронизации не поддерживается, что в долгосрочной перспективе может привести к рассинхронизации используемых в двух системах данных, ввиду возможности их параллельной обработки.

1.3. Удаленный вызов процедуры

Выполнение определенных действий или передача данных осуществляется путем удаленного вызова процедур, предоставляемых другим приложением.

Данный вид интеграции является достаточно мощным подходом, позволяющим инкапсулировать логику работы с данными посредством программного кода, упростить логику работы целевой системы с внешними ресурсами путем выноса интеграционного кода в отдельный сервис, однако в то же время влечет за собой сильную связанность приложений.

Вообще говоря, оценка достоинств и недостатков данного подхода и подхода, заключающегося в обмене сообщениями между системами (будет рассмотрен далее) во многом основана на существующей теории из области построения распределенных систем. Так, для любой системы, попадающей в категорию распределенных становится актуальной CAP теорема [3], утверждающая, что в ходе разработки и эксплуатации таковых можно достичь одновременного выполнения лишь двух из трех свойств, в число которых входят доступность, устойчивость к разделению, и согласованность данных. Аналогичным образом, становятся актуальными вопросы обеспечения согласованности и определения порядка возникновения событий, происходящих в отдельных компонентах распределенной системы.

В случае использования RPC-подхода, системы, как правило обладают свойствами согласованности и доступности, однако, при этом, теряют устойчивость к разделению (или устойчивость к сбоям). Иными словами, в случае выхода из строя (по различным причинам) сервиса,

предоставляющего удаленные методы, функциональность в целевой системе, использующая данные методы, будет недоступна.

С реализацией интеграционного решения, основанного на использовании удаленного вызова процедур можно ознакомиться при рассмотрении взаимодействия между модулем “Карточка студента” и СЭДД “Дело” в ходе осуществления процесса синхронизации информации в рамках управления документооборотом. Детальному анализу и оценке качества текущей реализации взаимодействия между двумя упомянутыми системами посвящен отдельный раздел данной работы.

1.4. Обмен сообщениями

Под обменом сообщениями подразумевается стиль интеграции, при котором взаимодействие между приложениями осуществляется с помощью системы обмена сообщениями, которые используются для обмена данными и выполнения действий.

Среди основных характеристик данного метода, выделяются как правило:

- Возможность частого, асинхронного обмена данными
- Возможность уведомления получателя о совершенной отправке сообщения
- Возможность повторной отправки сообщения в случае сбоя
- Слабая связанность интегрируемых приложений

За счет данных характеристик достигается устойчивость выполняемого процесса к сбоям (в сравнении с RPC - подходом), а также обеспечение инкапсуляции данных для интегрируемых приложений.

Было бы не совсем корректно говорить о том, что данный подход противопоставляется использованию вызова удаленного метода. Скорее эти подходы являются ортогональными (а не противоположными) друг другу,

т.е. направлены на решение разных задач. В случае использования удаленных методов, системы обладают свойством согласованности, но теряют устойчивость к разделению. В то же время, при использовании паттерна обмена сообщениями, выполнение сценариев взаимодействия систем, основанные на таком подходе, как правило, устойчиво к выходу из строя компонентов решения (способно позднее продолжить работу без повторного выполнения) и обладает свойством асинхронности, однако, теряя при этом свойство согласованности в терминах CAP теоремы (ограничиваясь лишь согласованность в конечном счете).

В зависимости от той или иной задачи, каждый из этих методов может использоваться как по отдельности, , так и в совокупности [17] .

Если говорить о практическом использовании данного подхода в рамках ИС СПбГУ, то можно отметить, что данной подход является самым редкоиспользуемым. Реализацию механизма обмена сообщениями удалось наблюдать при исследовании сценария интеграции между “Карточкjq студента” и СЭД “Дело”. В данном случае, в качестве системы обмена сообщениями используются промежуточная база данных OfficeDocDB, предназначенная для размещения записей о проектах приказов, подлежащих согласованию и направляемых из “Карточки студента” в СЭДД “Дело”. Как уже было отмечено ранее, более подробный результат анализа данного сценария интеграции будет представлен в работе далее .

1.5. Другие способы

Несмотря на представленное условное разделение, в настоящий момент существуют в том числе и такие решения, которые явно не попадают ни в одну из рассмотренных категорий. К числу таких решений могут относиться следующие.

1.5.1. Использование ETL-средств

Используются, как правило, для осуществления пакетной синхронизации данных между двумя источниками и могут запускаться как по требованию, вручную, путем привлечения разработчиков, либо путем запуска раз в определенный интервал времени с помощью планировщика. Стоит отметить, что именно данный подход является наиболее распространенным (более 50%) при решении задач интеграции систем в рамках университета. Причинами такой распространенности заключается, прежде всего, в простоте использования средств для реализации ETL-сценария, отсутствие необходимости разработки и поддержки промежуточных средств интеграционных средств (как в случае с RPC или обмена сообщениями). К недостаткам же следует отнести возможные задержки при интеграции данных, трудности, возникающие при необходимости синхронизации не данных, а событий изменения этих данных (т.н. event-sourcing) [32] , сложность реализации комплексных интеграционных сценариев, отличных от подхода (извлечь - загрузить), характерные для данного подхода разрастание и пересечение интеграционных связей, в целом усложняющие архитектуру информационной системы (т.н. взаимодействие точка-точка).

При оценки эффективности использования ETL-инструментов в рамках задачи интеграции в качестве одного из критериев можно руководствоваться следующей блок-схемой (рис. 1):

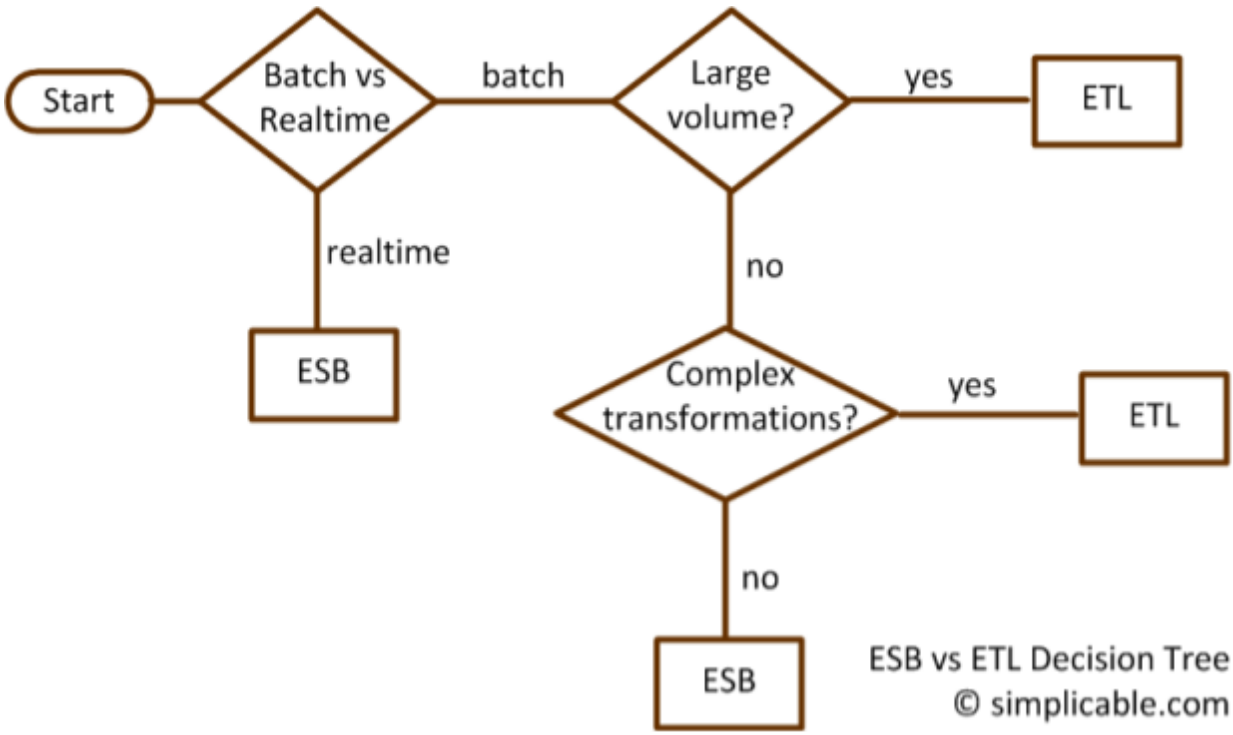


Рис. 1: Выбор интеграционного инструмента [33]

А именно, перед принятием решения, уточнить ответы на такие вопросы как:

- Какова частота изменения данных?
- Каков объем синхронизируемых данных?
- Насколько сложной является трансформация синхронизируемых данных?

Стоит внести уточнения в два пункта данной блок-схемы:

- Под ESB может пониматься не только классическое “коробочное” Enterprise Service Bus - решение, но и любое решение, удовлетворяющие принципам слабой связанности интегрируемых программных компонентов и решающая задачу синхронизации часто изменяемых данных между множеством систем (например, решения построенные на основе принципа обмена сообщениями).

- Стоит разделять сложность трансформации данных (упомянутую на блок-схеме) и сложность интеграционного бизнес-процесса (в качестве примера “сложного” бизнес-процесса можно привести интеграцию между СЭД “Дело” и “Карточкой студента”, рассмотренную в работе далее). В последнем случае, более предпочтительным может являться ESB-решение.

1.5.2. Пакетная загрузка данных через UI-интерфейс

Как правило, данная функциональность разрабатывается строго для выполнения конкретных интеграционных сценариев и ограничена возможностью работы со строго определенной схемой данных и источником строго определенного формата.

Может применяться в тех случаях, когда реализация любого иного интеграционного подхода является достаточно трудозатратной. Например, такая ситуация может возникать в случае необходимости внесения в систему объемного блока данных, создание и учет которых осуществлялся сотрудниками в системах, ранее эксплуатируемых в течение длительного периода времени и на этапе реализации которых не закладывались средства дальнейшего расширения в сторону интеграции с другими системами.

В то же время, как показывает практика, данные в такого рода системах-источниках могут носить противоречивый (некачественный) характер (например, устаревшие данные). В ряде случаев, в ходе синхронизации такого рода данных между системами, зачастую не представляется возможным осуществить качественную “фильтрацию” в автоматическом режиме, единственным возможном средством, гарантирующим загрузку в другую систему актуального и непротиворечивого набора данных, является ручная выверка данных

сотрудниками университета. Как показывает практика, достаточно часто процесс такого рода “выверки” осуществляется с использованием различного рода офисных приложений, например, табличного редактора Excel. В результате, после завершения процесса сверки данных единственным актуальным источником качественной информации является полученный в результате файл, на загрузку данных из которого и направлены реализуемые UI-механизмы.

Примером такого рода подхода к интеграции может служить процесс загрузки информации о зачисленных абитуриентах (студентах первого курса) из системы “Прием” в БД модуля “Карточка студента”

1.6. Альтернативная классификация интеграционных методов

Вышепредставленная классификация интеграционных паттернов основана по большей мере на специфике используемых технологических решений в ходе интеграции, будь-то использование файла, общей базы, вызова удаленного метода или ETL-сценария. В то же время, для более глубокого понимания всего существующего многообразия интеграционных процессов, можно рассмотреть альтернативные способы классификации. Например, в некоторых источниках [16] выделяются следующие классы интеграционных методов:

- По времени запуска
    - Реального времени - механизм синхронизации запускается сразу после возникновения изменений в системе (пример - отправка проекта приказа из подсистемы “Обучающиеся” в СЭД “Дело”)
    - Отложенная - синхронизация запускается после возникновения определенного события или по расписанию (под данную категорию подходит каждый из существующих ETL-сценариев)

- По способу анализа информации
    - По текущему состоянию - сравнение записей данных в источнике и приемнике, и на основе этой информации принятие решения о синхронизации тех или иных данных (ETL-сценарии, загрузка excel-файлов через UI)
    - Дельта-репликация - синхронизация данных, изменение которых (включая создание) произошло с момента последней операции синхронизации (отчасти, в рамках интеграции с СЭД “Дело”)
- По направлению интеграции:
    - Одностороннее - изменение данных только в одном приложении и доставка в другие без возможности модификации
    - Многостороннее - модификация данных происходит в нескольких интегрируемых системах, синхронизация осуществляется между каждой из систем на импорт и на экспорт.

1.7. Выводы

Подводя итог исследования существующих интеграционных средств, используемых в рамках ИС СПбГУ, можно выделить следующий набор выявленных недостатков:

1) Большинство интеграционных сценариев носит отложенный, асинхронный характер, но выполняется в ручном режиме.

2) Некоторые интеграционные сценарии, требующие выполнения синхронизации в режиме дельта-репликации, реализованы на основе анализа текущего состояния данных, ввиду сложности реализации механизмов учета совершенных изменений в системе. Отказ от данной логики упрощает реализацию интеграционного решения, но в

некоторых случаях может приводить к нарушению согласованности данных в виде коллизий или дублей.

3) Взаимодействие точка-точка (то есть, “каждый с каждым”) приводит к большому многообразию, пересечению и дублированию интеграционной логики. Существуют случаи, при которых в изменениях, возникающих в одной системе, заинтересованы несколько систем-потребителей, и для каждого из таких систем создается отдельный ETL-сценарий синхронизации данных.

4) Отсутствие средств для отслеживания потоков данных (dataflow), которыми обмениваются системы.

5) Неоднородность применяемых интеграционных стилей даже в рамках одного сценария взаимодействия между системами. В некоторых случаях это может приводить к нивелированию достоинств и накоплению недостатков каждого из применяемых интеграционных методов. (см. главу “Интеграция ИС “Обучающиеся и СЭД “Дело””)

6) Отсутствие централизованных средств оркестровки и мониторинга активности интеграционных решений. В качестве примера - многообразие ETL-пакетов, размещенных в нескольких репозиториях, запускаемых вручную по требованию, либо через механизм заданий, запущенных на разных экземплярах SQL Server на разных физических серверах. Либо же множество веб-сервисов, аналогично, запущенных на разных хостах, осуществляющих логирование в собственной локальной директории файловой системы. При такой ситуации становится затруднительным оперативное реагирование на проблемы, возникающие в ходе работы механизма интеграции .

7) Как продолжение предыдущего пункта, отсутствуют механизмы для быстрого сбора информации о возникающих ошибках в ходе интеграции. Отсутствуют механизмы оповещения сотрудников поддержки и разработчиков о возникновении ошибок (большинство ошибок обнаруживается лишь после обращения конечного пользователя с информацией о возникновении проблем при эксплуатации ПО).

С одной стороны, выявленные недостатки носят исключительно “технический” характер, однако, при более детальном рассмотрении последствий, можно говорить о том, что данные технологические недостатки в той или иной степени, но влекут за собой вполне измеряемые проблемы, находящие свое выражение в увеличении затрат на поддержку, разработку и эксплуатацию компонентов информационной системы университета.

Стиль точка-точка увеличивает сложность информационной системы в целом, приводит к дублированию интеграционной логики и, как следствие, требует дополнительных затрат на разработку и поддержку со стороны программистов [2].

Отсутствие понимания потоков данных может приводить к принятию не самых эффективных решении при проектировании и разработке новой интеграционной функциональности. Это может приводить к дублированию данных или их несогласованности, реализации одноразовых решений, без возможности последующего переиспользования [18].

Некорректное смешивание интеграционных стилей (например, RPC и обмена сообщениями) может приводить к накоплению недостатков, и в то же время к нивелированию достоинств, присущих каждому из интеграционных подходов. Как следствие, в могут наблюдаться проблемы согласованности

данных, производительности и отказоустойчивости (пример такого рода проблем будет представлен в работе далее).

Использование ETL-сценариев в тех случаях, когда необходима более быстрая синхронизация часто изменяемых данных между компонентами ИС [12]. В результате - затраты ресурсов разработчиков в случаях отсутствия настроенных механизмов автоматического запуска ETL-сценариев и большие интервалы времени, в течение которых данные находятся в несогласованном состоянии, что в некоторых случаях вынуждает пользователей осуществлять их ручную актуализацию.

Отсутствие средств оркестровки, мониторинга и оповещения о возникающих ошибках, лишает сотрудников техподдержки, системных администраторов и разработчиков возможности оперативного реагирования на возникающие в ходе работы системы ошибки, а это, как следствие, приводит к снижению эффективности эксплуатации системы конечными пользователями.

Стоит отметить, что решение упомянутых проблем является достаточно актуальным при построении большинства сложных гетерогенных информационных систем. Анализ источников, в которых рассматриваются вышеописанные проблемы, показал, что в той или иной степени, авторы в представленных работах в качестве основного, или по крайней мере одного из способов решения, рассматривают использование интеграционного паттерна, основанного на обмене сообщениями. Ввиду его распространенности, при последующем проектировании и реализации решений, направленных на нивелирование вышеописанных недостатков при рассмотрении конкретных практических примеров, автором в качестве основного интеграционного паттерна рассматривался именно обмен сообщениями.

Среди достоинств такого подхода, так или иначе упоминаемых в рассмотренных источниках, как правило выделяют следующие:

1) Уменьшение связности систем. Упрощение разработки и поддержки.
2) Упорядочение потоков данных.
3) Возможность избавления от сценариев интеграции типа точка-точка, реализованных с помощью ETL-инструментов.
4) Возможность использования очереди при реализации сценариев, требующих дельта-синхронизации данных (не обязательно с использованием отдельного брокера сообщений, но через формирование сообщений о возникновении событий).
5) Ввиду допустимости асинхронности, использование очередей возможно, но не обязательно. ETL-пакеты также подходят, но могут возникнуть проблемы из пунктов 1, 2

Помимо выбора наиболее эффективного интеграционного шаблона, был осуществлен поиск и анализ инструментов, используемых при интеграции приложений, направленных на решение вышеописанных проблем и предоставляющих инструменты для построения интеграционных решений, в том числе основанных на обмене сообщениями между системами. Обзор такого рода решений будет представлен в работе далее .

Глава 2. Анализ интеграционных инструментов

Все многообразие существующих решений можно условно поделить на некоторый набор классов. Как правило, основными критериями разделения служат набор предоставляемой функциональности и сложность освоения и эксплуатации.

В некоторых источниках выделяется по крайней мере 3 класса существующих решений, используемых при интеграции приложений, среди которых выделяют [12], [19]:

- Integration frameworks
- Enterprise Service Bus
- Integration Suite.

Стоит отметить, изучение на практике решений каждого из упомянутых категорий оказалось достаточно трудозатратным в виду высокого входного порога на эксплуатацию решений некоторых классов (в частности, решений категории ESB и Integration suite), и ограниченного количества времени на выполнение данной работы.

Однако, для ознакомления с возможностями и ограничениями решений каждой из категорий, был проведен сбор информации, посвященной оценке уже существующего опыта использования тех или иных решений. Ознакомиться с ключевыми особенностями решений тех или иных категорий, их сравнением и целесообразностью использования при решении конкретных задач, можно в представленных источниках. При этом в рамках работы стоит отметить, что для решения озвученных ранее проблем было выбрано решение, относящееся к классу интеграционных фреймворков. Данный выбор был обусловлен следующими особенностями, характерными для данной категории решений:

1) Предназначены для интеграции приложений на основе обмена сообщениями. Реализация большинства EAI-паттернов, как правило, уже входит в состав решения.
2) Более просты в освоении, настройке и эксплуатации.
3) Существуют решения на основе .NET платформы, что также упрощает процесс эксплуатации, ввиду распространенности данной платформы при разработке компонентов ИС СПбГУ.

Из функциональности, как правило, присутствующей в более сложных решениях категории ESB и Integration suite, но отсутствующей в интеграционных фреймворках, можно упомянуть [13]:

1) Предоставление готовых адаптеров. Например, для работы с файловыми источниками (excel), протоколами (ftp, http, smtp), базами данных предоставляются большинством ESB-решениями. В случае использования интеграционного фреймворка для работы с большинством источников и приемников необходимо писать собственный код.
2) Функциональность для маппинга схем данных между источником и приемником через визуальный интерфейс.
3) Функциональность для управления бизнес-процессами (сервисная оркестровка, отслеживание потоков данных и т.д.).
4) Предоставление общего для всех разрабатываемых компонентов с интеграционной логикой сервера приложений (application server).
5) Мониторинг производительности, оповещение об ошибках и т.п.

Однако, нельзя сказать, что упомянутых возможностей совершенно невозможно достичь используя интеграционные фреймворки. Например, в качестве различного рода адаптеров можно использовать готовые

библиотеки для того или иного языка программирования (например, компоненты .NET для работы с excel-таблицами, ftp-серверами и т.п).

Отслеживание потоков данных между системами может достигаться, например, через мониторинг процесса прохождения сообщений через те или иные каналы брокера сообщений [20].

Централизованные оркестровка и мониторинг активности компонентов интеграционного решения могут достигаться, например, с использованием активно развивающихся и используемых в настоящие время средств классов service discovery и контейнерной виртуализации [9].

Таким образом, можно говорить о том, что для достижения вышепредставленной функциональности использование сложных комплексных решений классов ESB и Integration Suite является лишь одним из возможных решений.

Стоит также отметить что, как уже было сказано ранее, ввиду ограниченного количества времени, большая часть работы посвящена непосредственно задачам проектирования и построения интеграционного решения, основанного на обмене сообщений, и меньшая - вопросам построения (настройки) средств мониторинга и оркестровки.

Исходя из вышеизложенного, дальнейший поиск инструмента для интеграции осуществлялся в рамках категории интеграционных фреймворков. При этом предпочтения отдавались решениям, которые предназначались для использования на основе .NET платформы.

Среди проанализированных решений были выделены два интеграционных фреймворка: MassTransit и NServiceBus. Немного заходя вперед, стоит сказать, что на начальной стадии проектирования и реализации интеграционного решения были опробованы оба фреймворка, но в итоге выбор был сделан в пользу решения MassTransit. Если говорить, о предоставляемой функциональности, то здесь рассматриваемые решения

оказались достаточно похожи. Если сравнивать объем и качество документации, предоставляемую поддержку и размер сообщества, то NServiceBus показал себя как более зрелый продукт. Однако, стоит отметить, что во время выполнения настоящей работы разработчики MassTransit полностью переработали, унифицировали и структурировали документацию, в результате чего достаточно сильно повысилось удобство использования продукта.

Основным же критерием при выборе инструмента для данной работы являлись лицензионные условия использования каждого из решений. MassTransit предоставляется под лицензией Apache 2.0, что позволяет использовать данный продукт как для персональных, так и для внутренних корпоративных решений на безвозмездной основе. Использование же решения NServiceBus для внутренних проектов возможно лишь на основе платной лицензии [21].

Глава 3. Использование техник непрерывной интеграции при решении задач интеграции приложений

Непрерывная интеграция (Continuous Integration, CI) - набор практик, используемых в ходе разработки ПО, при которых члены команды придерживаются принципа частого внесения изменений в исходный код, размещенный в командном репозитории. Каждое изменение сопровождается автоматизированным выполнением сборки исходного кода и выполнением тестов с целью быстрого обнаружения возникающих ошибок [6]. Как правило, при использовании инструментов, реализующих данный подход, выполняется следующая цепочка действий [7]:

1) В репозиторий вносятся изменения
2) Выполняется сборка последней версии исходного кода
3) В случае отсутствия ошибок, осуществляется выполнение unit-тестов
4) После прохождения unit-тестов приложение размещается в тестовом окружении с целью выполнения интеграционных и нагрузочных тестов, тестов приемки.
5) В случае отсутствия ошибок, собранный билд размещается в специальном месте (например, с целью последующей доставки на production-окружение).
6) После окончания процесса, происходит уведомление всех заинтересованных лиц об итоговом результате (посредством email, RSS и др.) и формирование подробного отчета с указанием номера сборки, информации о пройденных или не пройденных тестах.

В некоторых источниках шаги 4 и 5 обозначают отдельным термином “Непрерывная доставка” (“Continuous Delivery”, CD) - практика автоматизированного развертывания приложения в требуемом окружении

[8]. Среди видов окружения выделяются такие как development (разработка), integration (интеграция), staging (тестовая эксплуатация) и production (производственная эксплуатация)[9].

Эффект от использования данных средств особенно заметен при разработке, тестировании и развертывании сложных систем, включающих в себя некоторый набор обособленных компонентов. Например, особенно важно применение данных техник при проектировании и разработке систем, построенных на основе сервис-ориентированной архитектуры (SOA) или микросервисов [10].

Рассматриваемые в рамках данной работы механизмы интеграции приложений ИС СПбГУ могут включать в себя такой набор компонентов как непосредственно интегрируемые приложения, базы данных, брокер сообщений, api-сервисы, сервисы, содержащие саму интеграционную логику и т.д. Соответственно, для тестирования корректности реализации различных интеграционных сценариев может возникнуть потребность в быстром развертывании всего набора задействованных компонентов. Аналогичным образом, потребность в одновременном развертывании может потребоваться и при доставке новой версии на окружение для производственной эксплуатации.

В ходе работы автором был получен опыт использования инструментов continuous integration и deployment, предоставляемых компанией Microsoft в рамках платформы Visual Studio Team Services [22]. Инициирована практика использования данных средств, в ходе которых были настроены процессы автоматической сборки, тестирования и развертывания таких приложений, входящих в состав ИС СПбГУ, как настольный модуль “Обучающиеся”, веб-приложение “Личный кабинет студента” и другие.

Полученный опыт использования данных инструментов делает возможным эффективное применение практик CI и CD для упрощения

процедур развертывания и тестирования многокомпонентных интеграционных решений, процесс проектирования и разработки которых будет представлен в работе далее.

Глава 4. Интеграция между подсистемой “Обучающиеся” и ЭБИС

В рамках практической части данной работы был рассмотрен конкретный процесс интеграции двух программных компонентов ИС СПбГУ, целью выполнения которого является актуализации статуса читательской записи (читательского билета) в Электронной библиотечной информационной системы (ЭБИС) отраслевого отдела по направлению юриспруденция библиотеки им. Горького при изменении статуса студента в подсистеме “Обучающиеся”.

Информационная система “Обучающиеся” (“Карточка студента”), (техническое название - StudentCard) - настольный модуль (win-приложение) для просмотра и редактирования данных о студентах. Основной целевым классом данных, для учета и ведения которых была внедрена система подсистема “Обучающиеся”, являются персональные и студенческие данные, например, такие как ФИО, паспортные данные, направление обучение студента, статус обучения, номер студенческого билета, набор ведомостей,оценок и др. Помимо этого, данный модуль является одним из компонентов ИС СПбГУ, используемым для управления электронным документооборотом и делопроизводством.

подсистема“Обучающиеся” на данный момент является первоисточником данных о контингенте и успеваемости. Эксплуатация ЭБИС осуществляется с целью упрощения поддержки исполнения бизнес-процессов, задача которых состоит в обеспечении студентов ресурсами библиотечного фонда. При этом, в рамках исполнения данного бизнес-процесса относительно конкретного студента, при осуществлении ряда процедур (например, таких как создание или архивирование читательского билета, проверка разрешения на выдачу книг, или определение

срока возврата книг в библиотеку) библиотечная система должна обладать актуальной информацией о текущем статусе студента, первоисточником которой, как было отмечено, является подсистема “Обучающиеся”.

Поскольку реализация описанного бизнес-процесса не может быть осуществлена в рамках эксплуатации какой-либо одной из приведенных выше систем, возникает задача их интеграции.

4.1. Описание сценария взаимодействия между модулем “Карточка студента” и ЭБИС

Зависимость ЭБИС от подсистемы “Обучающиеся” определена на уровне данных. В частности, статус студента в подсистеме “Обучающиеся” связан со статусом читательского билета данного студента в ЭБИС . Все изменения студенческого статуса должны приводить к изменениям читательского билета.

Читательский билет может находиться в одном из двух состояний:

а) Активен. Читательский билет должен находиться в данном состоянии начиная от момента выхода приказа о зачислении студента в Университет на направление юриспруденция и заканчивая моментом сдачи всех книг в библиотеку после выхода приказа об отчисления (в независимости от основания).

б) Аннулирован. Читательский билет переводится в данный статус после выхода приказа об отчислении студента и погашении долгов данного студента перед библиотекой. Перевод читательского билета в данный статус осуществляется сотрудниками библиотеки после погашения задолженности по книжному фонду и не требует автоматизации.

Среди бизнес-процессов, требующих автоматизации изменения данных на уровне ЭБИС можно выделить следующие:

1) Появление нового студента в подсистеме “Обучающиеся”, относящегося к направлению юриспруденция. В этом случае в ЭБИС должна создаваться новая читательская запись (билет) для этого студента. При этом, данный процесс должен осуществляться и в том случае, если одна и та же персона поступила на новую ступень обучения. Например, для студента-бакалавра и для студента-магистра в системе должны храниться две разные читательские записи.

2) Перевод студента на следующий курс. В этом случае при работе с читательским билетом студента сотрудникам, использующим библиотечный модуль, должна быть доступна актуальная информация о курсе/потоке/группе данного студента. Также должна быть доступна информация о том, является ли данный студент “хвостовиком”, т.е. имеющим академическую задолженность за предыдущий учебный период.

Стоит отметить, что на данный момент для решения задачи актуализации таких данных студента как ФИО, дата рождения, номер телефона, e-mail, курс, поток, группа, форма обучения, ступень обучения используется ранее введенный в эксплуатацию web сервис, предоставляющий эти данные в ответ на запрос по протоколу http . Благодаря этому решение данной задачи не требует реализации какой-либо дополнительной логики по интеграции систем. Однако, существующая реализация механизма создания новых читательских записей для зачисленных студентов обладает рядом недостатков, анализу и исправлению которых и посвящена данная часть работы.

4.2. Анализ исходного решения

Существующий механизм интеграции основан на получении от разработчиков системы “Прием” excel-файла с данными о поступивших студентах и последующей загрузке полученных данных в два источника:

1) Базу данных “Обучающиеся”
2) В базу данных ЭБИС

Более детальная последовательность действий выглядит следующим образом:

1) Разработчик системы “Прием” формирует excel-файл, содержащий набор данных о поступивших студентах, строго определенного формата (формат заранее согласован и утвержден разработчиками двух систем).
2) С помощью вызова реализованного в клиентском настольном модуле подсистемы “Обучающиеся” механизма, ответственное лицо (на практике, как правило, это разработчик одной из двух систем), имеющее соответствующие права доступа, осуществляет массовую загрузку данных о студентах в базу данных системы. Данный механизм представляет собой автоматизированный алгоритм записи данных из excel-файла в базу данных, с осуществлением валидации (проверка структуры представленных в файле данных, проверка на дубли студенческих записей и т.п.). Использование данного механизма доступно через UI-интерфейс клиентского модуля, который позволяет указать путь до соответствующего excel-файла, и осуществить запуск алгоритма по нажатию на кнопку.
3) После загрузки данных в подсистему “Обучающиеся” разработчиком инициируется процедура создания читательского билета в библиотечной системе - вручную запускается ETL-сценарий, который

на вход принимает ранее предоставленный excel-файл и записывает имеющиеся данные в определенные таблицы базы данных библиотечного модуля.

В выполнении данного процесса можно выделить следующие недостатки:

а) Дублирование ручных действий.

Необходимость в переносе данных из одной системы в другую, как правило, возникает несколько раз в ходе приемной комиссии. Разработчику необходимо каждый раз запрашивать новую версию excel-файла, и инициировать загрузку данных в две системы.

б) Реализация интеграционной логики на основе анализа текущего состояния данных вместо более подходящей дельта-репликации.

В ходе анализа опыта эксплуатации данного интеграционного подхода, было выявлено, что каждая новая версия excel-файла содержит в себе набор данных как новых, так и ранее предоставляемых в предыдущих версиях excel-файлов. В результате этого, при реализации логики синхронизации необходимо учитывать возможность возникновения дублей или коллизий и предпринимать меры, направленные на недопущение возникновения такого рода ситуаций (например, путем реализации дополнительной логики проверки существования данной студ. записи в системе)

в) Наличие задержки при синхронизации данных.

Данный недостаток обусловлен тем, что процесс разработки рассматриваемых систем осуществлялся в разное время отличными по составу и независимыми друг от друга командами разработки. Разработчики, в случае потребности в данных от другой системы, вынуждены формировать запрос на получение данных, адресованный другой команде разработки (как правило все взаимодействие осуществляется посредством электронной

почты). Таким образом, ввиду отличного друг от друга набора задач и уровня загруженности, существующая задержка является логичным и объяснимым фактором, который необходимо принимать во внимание.

г) Вероятность возникновения ошибок вследствие ручного характера синхронизации.

Например, при осуществлении данного процесса необходимо держать в памяти и учитывать требуемую последовательность загрузки данных в системы (сначала в подсистеме “Обучающиеся”, после этого - в библиотечный модуль).

4.3. Реализация интеграционного решения на основе механизма обмена сообщениями

4.3.1. Описание алгоритма

Принятие решения об использовании подхода, основанного на обмене сообщениями было сформировано на основе совокупности недостатков, присущих текущему решению и возможным преимуществам упомянутого интеграционного паттерна. Процесс создания читательской записи на основе спроектированного решения представлен в приложении А. Опишем основные шаги интеграционного процесса:

1) В систему “Обучающиеся” с использованием ранее реализованного и упомянутого UI-интерфейса инициируется загрузка данных о поступивших студентах из excel-файла, сформированного разработчиком системы “Прием”. Для каждой из записей выполняется следующий набор действий.

2) Осуществляется проверка наличия данной записи в системе “Обучающиеся”. В случае ее отсутствия, она вносится в базу данных.

3) После формирования студенческой записи в базе данных, выполняется формирование и отправка сообщения в очередь, расположенную в отдельном брокере сообщений (в качестве брокера используется решение RabbitMQ). В сообщении передается уникальный идентификатор созданной студенческой записи.

4) Специальный интеграционный сервис извлекает сообщения из очереди

5) На основе идентификатора, полученного из сообщения, сервис извлекает необходимую информацию из базы данных, с помощью которой принимает решение о необходимости вызова метода создания читательской записи. В качестве критерия выступает значение направления обучения данного студента (это должно быть направление юриспруденция).

6) В случае удовлетворения записи всем необходимым условиям, вызывается метод API-сервиса, в котором реализовано логика создания читательской записи в базе данных модуля библиотеки.

7) API-сервис вносит необходимые изменения в базу данных модуля библиотеки.

4.3.2. Преимущества

На основе представленного алгоритма и его схемы, опишем основные преимущества данного решения:

- Аналогично исходному решению, использование очереди сообщений не усиливает связанность интегрируемых приложений.
- Данный подход избавляет от необходимости реализации избыточной интеграционной логики, заключающейся в проверке каждого синхронизируемого набора данных на предмет ранее осуществленной

синхронизации. В результате, становится возможным реализация интеграционной логики на основе дельта-репликации, что является более простым и естественным подходом в данном случае.

- Становится более простым определение существующих потоков данных. Благодаря использованию брокера сообщений, на основе построенной топологии очередей сообщений становится возможным определение того, из каких и в какие системы направляются те или иные данные, определять частоту возникновения и количество событий (сообщений), характер и темпы синхронизации.

- Автоматизация и уменьшение времени процесса синхронизации. Разрабатываемое решение в том числе направлено на избавление от необходимости ручного запуска ETL-сценариев синхронизации. Стоит отметить, что данного эффекта можно достичь и без использования очередей, например, путем автозапуска ssis-пакета с определенной частотой. Однако, поскольку это не было реализовано в исходном решение, то этот пункт стоит упомянуть.

- Использование механизма очередей предоставляет возможность обработки одного и того же события (в данном случае, создание студенческой записи в системе “Обучающиеся”) несколькими заинтересованными потребителями (обработчиками). Например, в случае рассматриваемого сценария интеграции обработчиком выступает сервис по созданию читательской записи в библиотечном модуле. Но также имеется возможность реализовать иные обработчики, например, для логирования данного события с использованием специализированных средств (например, ELK-стека) и дальнейшее использование полученной информации разработчиками, сотрудниками техподдержки и иными категориями пользователей.

Однако, несмотря на все перечисленные преимущества, реализации данного подхода присущи некоторые сложности и недостатки, рассмотрение которых будет осуществлено далее в работе.

4.3.3. Доставка сообщений в брокер

В ходе реализации рассматриваемого интеграционного сценария, прежде всего возникла задача доставки данных о возникших в системе-источник событий в брокер сообщений. Среди вопросов, ответы на которые было необходимо получить, выделялись следующие:

1) Какие данные размещать в формируемом к отправке сообщении?
2) Каким образом осуществлять отправку сообщения в брокер?

При решении задачи определения необходимого набора данных было рассмотрено два противоположных друг другу подхода.

- Первый заключается в формировании сообщения с минимально возможным набором полей, достаточным для того, чтобы сервис-обработчик смог определить, какой тип изменения (создание, удаление, изменение значения некоторого поля), над каким объектом и в какое время был осуществлен. Преимуществом является унификация формата сообщений и возможность избежать сильной связанности подсистемы “Обучающиеся” с другими приложениями через зависимость от формата сообщений. К числу недостатков следует отнести потребность в обеспечении доступа к данным подсистемы “Обучающиеся” для сторонних компонентов, задействованных в процессе интеграции и ограничение типов возможных событий значениями “удален”, “изменен”, “создан”.
- Второй способ - это размещение всей необходимой для интеграционного процесса информации внутри сообщения. Основной

недостаток данного подхода - необходимость модуля “Карточки студента” “знать” обо всех сервисах, участвующих в обработке сообщений, что может являться причиной частого внесения изменений в контракты и усложнением поддержки приложения.

К числу же преимуществ стоит отнести отсутствие необходимости обеспечения доступа к данным для сторонних компонентов и возможность оперировать событиями уровня бизнес-логики.

При решении данной задачи автором был выбран первый подход. Выбор был обусловлен количеством существующих и потенциальных интеграционных сценариев между компонентами, в которых “Карточка студента” является источником.

В качестве инструмента был выбран модуль аудита (о нем более подробно - далее в работе), входящий в комплект XAF-фреймворка, широко используемого при разработке большинства приложений, входящих в состав ИС СПбГУ. Модуль предоставляет удобные инструменты для отслеживания всех возникающих в системе событий изменения данных (включая, обновление, создание и удаление) с указанием информации о том, какой это объект, какого типа и времени его возникновения.

По умолчанию, данный модуль осуществляет запись такого рода лога в отдельные таблицы базы данных. Однако, разработчиками данного модуля была предусмотрена возможность расширения функциональности, в случае необходимости.

Автором на основе данного модуля была реализована собственная надстройка, позволяющая без перекомпиляции приложения, через UI-интерфейс выполнять конфигурацию модуля с возможностью выбора конкретных событий определенного типа, которые необходимо отслеживать. Помимо этого, была реализована логика, позволяющая отправлять такого

рода данные в брокер сообщений, помимо (или вместо) таблиц по-умолчанию, расположенных в БД.

Если в рамках предыдущего вопроса затрагивались темы формирования сообщений, то следующей задачей оказалась реализация логики отправки сообщения в брокер.

Для работы с брокером сообщений был использован интеграционный фреймворк, предоставляющий API, методы которого инкапсулируют всю низкоуровневую логику транспортного уровня, и предоставляют высокоуровневый интерфейс для таких операций как установка и закрытие соединения с брокером, или для отправки и получения сообщений.

Проблема же состояла не в том, как реализовать метод отправки сообщений в очередь, а в том, в какой момент выполнения программы его вызывать. Было рассмотрено и протестировано три подхода.

Первый, самый простой способ, это отправка сообщения в том же методе, в котором осуществляется изменение целевых данных. Например, предположим, что в системе существует метод ImportNewStudent, на вход принимающий значения строк из excel-таблицы, и в цикле осуществляющий создание новых студенческих записей, соответствующих входным данным.

В этом случае, сразу после создания студенческой записи в системе, можно было бы сформировать соответствующее сообщение и отправить его в брокер. Однако, данному подходу присущи два критических недостатка:

- Операции внесения изменений в реляционную базу данных и отправка сообщения в брокер не являются атомарными [25]. Как следствие, в случае ошибки при отправке сообщения в брокер, данное сообщение будет потеряно без возможности повторной отправки.
- Инъекция кода для отправки сообщений в очередь в методы бизнес-логики способствуют его распространению по всему приложению. Это может привести к сильной связанности между

брокером и приложением, к усложнению отладки, тестирования и поддержки.

Альтернативный подход обусловлен возможностями, предоставляемыми модулем аудита, входящим в комплект XAF-фреймворка. А именно, данный модуль предоставляет инструменты для реализации собственного метода-”перехватчика”, вызываемого каждый раз при изменении данных в рамках приложения [26]. В рамках этого метода возможна реализация собственной логики работы с данными, содержащими информацию о совершенных изменениях в системе, в том числе осуществлять на их основе формирование и отправку сообщений в брокер.

К достоинствам данного метода можно отнести возможность реализации логики доставки и формирования сообщений в единой точке приложения, без распространения по слою бизнес-логики.

К числу недостатков стоит отнести аналогичную предыдущему подходу проблему обеспечения атомарности выполняемых операций изменения данных в БД и доставки сообщения в очередь. Более того, более детальное изучение материалов и исходного кода модуля аудита показали, что данный инструмент не обеспечивает возможности атомарного изменения целевых данных и сохранения логов изменения даже в рамках одной БД.

Способы решения данной проблемы будут рассмотрены далее в работе.

4.3.4. Обеспечение атомарности операций изменения данных и доставки событий в очередь

В качестве одного из вариантов решения данной проблемы может рассматриваться следующий подход:

1) Сохранение сообщений о возникновении событий в общей с целевыми данными базе. Как уже упоминалось ранее, используемый модуль аудита не предоставляет даже этой возможности. Однако, в ходе

исследования было найдено альтернативное решение [27], заключающееся в создании и реализации собственной логики отслеживания происходящих событий и их сохранения в БД подсистемы “Обучающиеся” в рамках одной транзакции. Идея данной реализации была предложена сотрудниками поддержки DevExpress и основана на обработке событий контроллера XAF-фреймворка.

2) Реализация отдельного сервиса, осуществляющего проверку наличия в БД новых изменений с целью их последующей отправки в брокер сообщений.

Вышепредставленные способы основаны на подходе, заключающемся в отслеживании возникающих событий на уровне кода приложения, с целью их сохранения в брокере сообщений или базе данных.

Для более полной картины стоит отметить, что в качестве альтернативного способа определения “среза” данных, а именно получения информации о произошедших изменениях и, в том числе, о состоянии данных в прошлом, предполагается, что можно использовать некоторые средства уровня базы данных. В качестве примера такого рода решений можно привести технологию MS Server Service broker [23], или решение Change Data Capture на основе MS SQL Server [24]. Технология Change Data Capture согласно описанию позволяет получать ретроспективные срезы данных за различные промежутки времени, которые могут в дальнейшем использоваться для генерации на их основе сообщений-событий для отправки в брокер. MS Server Service broker - более ранняя технология, входящая в состав SQL Server начиная с 2005-ой версии. Данное решение представляет собой брокер сообщений, встроенный в ядро базы данных. Соответственно, при использовании данного брокера, согласно описанию, представляется возможным решение проблемы атомарности изменений,

путем размещения сообщений в Service broker с последующей пересылкой во внешний брокер сообщений (например, RabbitMQ).

Как можно заметить, существует как минимум несколько подходов, позволяющих обеспечить согласованность данных (или, как минимум, согласованность в конечном счете) при выполнении операций взаимодействия приложения с базой данных и брокером сообщений. Ввиду необходимости обеспечения согласованности и гарантии доставки сообщений в очередь хотя бы в конечном счете, в итоге, в качестве решения в первом приближении был выбран подход, заключающийся в инъектировании логики формирования и сохранения события в бизнес-логику приложения, т.е. непосредственно в метод, осуществляющий создание студенческих записей по предоставленным данным из excel-файла. Как уже было сказано ранее, данное решение не является оптимальным.

В качестве дальнейших шагов к улучшению рассматривается реализация собственного “трекера” на базе XAF-контроллера. Видится, что при реализации данного решения будет возможно переиспользование уже реализованной логики по гибкой настройке отслеживаемых событий и логики сохранения событий в структуры базы данных приложения. Однако, ввиду трудоемкости данного решения, так же как и потенциальных решений, основанных на использовании функциональность MS SQL Server, решение данного вопроса планируется к осуществлению вне данной работы.

4.3.5. Инкапсуляция логики создания читательских записей

Еще одна проблема, возникшая при реализации рассматриваемого интеграционного решения, заключалась в отсутствии у автора знаний о логике работы и структурах базы данных, используемой модулем библиотеки. Соответственно, корректная реализация логики по созданию читательской записи является крайне трудозатратной по причине сложности

решения ЭБИС. В текущем случае, единственным способом получения информации является коммуникация с разработчиком, поддерживающим данную систему. Однако, в таком случае, с целью экономии ресурсов, вместо изучения технических деталей реализации библиотечного модуля, более эффективным является делегирование реализации необходимой логики основному разработчику данного решения.

Для возможности параллельной работы над разными частями интеграционного решения, упрощения поставки и использования данной функциональности (инкапсулирования сложной внутренней логики), было принято решение о разработке данной функциональности в рамках метода отдельного API-сервиса.

На момент написания данной работы, упомянутый API-сервис находится в разработке. Предполагается, что после окончания работы, разработанный сервис удастся легко интегрировать с представленным автором решением.

4.4. Выводы

На примере достаточно простой задачи синхронизации данных в двух источниках были выявлены достоинства и недостатки как исходного решения, основанного на репликации данных посредством ETL-сценария, так и предложенного в качестве альтернативны решения, основанного на синхронизации данных посредством обмена сообщениями.

Отметим ключевые наблюдения, возникшие в ходе построении решения, основанного на очередях сообщений.

Из достоинств апробированного подхода можно выделить возможность мониторинга и гибкого управления потоками данных в ИС СПбГУ, благодаря использованию единого брокера. С помощью представленного решения обеспечивается слабая связанность интегрируемых решений,

имеется возможность гибкого добавления дополнительных сервисов-обработчиков одних и тех же событий, без внесения изменений в другие системы.

С другой стороны, преимущества слабой связанности приложений могут быть нивелированы в случае использовании большого количества различных типов сообщений с уникальными наборами используемых бизнес-данных. В этом случае, увеличивается риск необходимости внесения изменений, и как следствие, увеличение сложности поддержки системы-источника. Чтобы избежать такого рода последствий, можно придерживаться принципа минимизации данных в используемых контрактах и решить задачу реализации отдельного слоя доступа к данным, которым могли бы пользоваться все заинтересованные обработчики сообщений.

Проблема иной категории связана с обеспечением атомарности операций изменения целевых данных и отправке сообщений в брокер. Было найдено несколько способов решения данной проблемы, включая реализацию собственного механизма трекинга событий в системе с последующей записью в целевую БД, или же использование готовых решений, реализованных на уровне базы данных. Однако, в рамках данной работы выбор был сделан в пользу наиболее простого решения, заключающегося в изначальном внесении изменений в целевые данные и сохранение соответствующих данным изменениям событий в общей базе данных, за счет чего возможно обеспечение атомарности при выполнении данных действий. Данное решение обладает рядом достаточно серьезных недостатков, и в дальнейшем планируется к пересмотру, но уже вне данной работы.

Стоит также отметить, что реализация интеграционного решения, как правило, не обязательно должна основываться исключительно на одном из существующих интеграционных паттернов. Так, в рамках решения данной

задачи, помимо обмена сообщения, было продемонстрировано использование такого решения, как вызов удаленной процедуры. Данный подход позволил инкапсулировать сложную специфическую логику создания читательской записи путем создания и предоставления для интеграционного сервиса уже готового метода, без необходимости прямой работы с базой данных.

Одно из упомянутых ключевых достоинств использования обмена сообщениями - асинхронность - не удалось продемонстрировать в рамках решения данной задачи, в виду того, что исходное решение, основанное на репликации данных с помощью ETL-сценария, также удовлетворяло данному свойству. Однако, в некоторых ситуациях данное свойство может привести к существенному повышению производительности системы, упрощению поддержки, и увеличению эффективности эксплуатации. Достижение такого рода улучшений при внедрении механизма обмена сообщениями будет продемонстрировано далее в работе, в рамках рассмотрения и улучшения интеграционного механизма между подсистемой “Обучающиеся” и СЭД “Дело”.

Глава 5. Интеграция между подсистемой “Обучающиеся” и СЭДД “Дело”

В качестве более сложного процесса интеграции можно рассмотреть сценарий взаимодействия между подсистемой “Обучающиеся” и СЭД “Дело” в рамках процесса управления документооборотом и делопроизводством в Университете.

Описание подсистемы “Обучающиеся” (“Карточка студента”) было представлено ранее в работе.

Другим компонентом, также задействованной в данном процессе, является Система Электронного Документооборота и Делопроизводства - СЭДД “Дело” - стороннее приобретенное программное обеспечение, предоставляющее обширные возможности, в том числе и для совместной работы, подготовки, согласования и утверждения проектов документов (в частности, например, приказов).

Стоит отметить, что существующие бизнес-процессы, в рамках которых осуществляется работа с приказами, требуют взаимодействия между собой обеих систем. Если говорить в общем, то в рамках подсистемы “Обучающиеся” пользователями осуществляется создание и учет проектов приказов, а в СЭД “Дело” - процедуры согласования, утверждения или отзыва. Соответственно, для возможности выполнения перечисленных сценариев, между системами необходимо обеспечить взаимодействие, заключающееся в обмене актуальными данными.

С подробным описанием упомянутых систем можно ознакомиться в приложении **Б**, а с предпосылками возникновения исходного решения в приложении **В**.

В приложении **Г** представлено подробное описание текущей реализации интеграционного решения - информация о наборе используемых в интеграционного решении компонентов, техническое описание и роль каждого из них. В качестве примера взаимодействия был рассмотрен процесс отправки созданного в системе “Обучающиеся” проекта приказа на согласование в СЭДД “Дело”. С целью иллюстрации данного процесса, была создана схема взаимодействия компонентов интеграционного решения, с которой можно ознакомиться в приложении **Д.**

В рамках данной работы будет выделен набор проблем, выявленных в исходном решении, а также представлена архитектура нового решения, направленного на нивелирование найденных недостатков, также в работе описан процесс реализации и проведена оценка полученного результата.

5.1. Анализ проблем

Осуществленный анализ текущего решения показал, что существующий механизм обладает рядом недостатков, возникновение которых достаточно характерно для решений, разрабатываемых в рамках построение сложной многокомпонентной распределенной системы [4].

Автором, в ходе исследовательской работы, включающей в себя сбор обратной связи от пользователей, сотрудников технической поддержки и разработчиков, анализ открытых и закрытых bug-report-ов и более детальный анализ технического решения, был выявлен ряд проблем, последствиями которых являются дополнительные трудозатраты ресурсов пользователей в ходе эксплуатации системы и технических специалистов в ходе поддержки и доработки данной реализации.

Весь набор выявленных проблем можно разделить на 4 категории:

- Отсутствие отказоустойчивости при недоступности (выхода из строя) одного из сервисов, задействованных в процессе интеграции.
- Ошибки, связанные с некорректной синхронизацией событий, независимо возникающих в разных системах.
- Проблемы, связанные с отсутствием транзакционности при выполнении операции внесения изменений, затрагивающие несколько компонентов интеграционного решения.
- Трудности в осуществлении мониторинга и обнаружения ошибок, возникающих в ходе эксплуатации исходного механизма.

С более подробным описанием упомянутых проблем и тем, в чем они выражены в рамках исходного механизма интеграции и к каким негативным последствиям приводят, можно ознакомиться в приложении Е.

5.2. Решение проблем исходной реализации

Предложенное в рамках данной работы решение направлено, прежде всего, на нивелирование вышеописанных недостатков, с целью повышения эффективности эксплуатации, поддержки и разработки интегрируемых систем.

С иллюстративным представлением архитектуры полученного решения можно ознакомиться в приложении **3.** Далее в работе будут описаны ключевые особенности представленного решения в контексте разбора проблем, упомянутых ранее.

5.2.1. Достижение отказоустойчивости

Одной из ключевых проблем, присущей исходному решению, является отсутствие отказоустойчивости при взаимодействии таких компонентов как:

1) Модулем “Обучающиеся” и общим сетевым файловым ресурсом
2) Модулем “Обучающиеся” и удаленным web сервисом.

Если рассматривать данную проблему с формальной точки зрения, то отсутствием отказоустойчивости в рассматриваемом случае является неудовлетворение распределенной системы свойству устойчивости к разделению (partition tolerance), описанному ранее упомянутой CAP-теореме.

Таким образом, можно говорить о том, что само по себе отсутствие у системы устойчивости к разделению, в том числе и в данном случае, является не допущенной в ходе проектирования и реализации системы ошибкой, а лишь следствием альтернативного выбора доступных при построении системы вариантов.

В приложении Ж, посвященному более детальному обзору данной проблемы, упомянуто, что, на самом деле, разные наборы компонентов исходного интеграционного решения обладают разными свойствами в разрезе данной теоремы. Так, можно говорить о том, что механизм доставки данных из “Карточки студента” в промежуточную базу данных OfficeDocDB удовлетворяет свойствам согласованности и доступности, при этом в целом между “Карточкой студента” и СЭДД “Дело” данное свойство не выполняется. В итоге, механизм интеграции в целом не обладает ни устойчивостью к разделению, ни согласованностью (лишь согласованностью в конечном счете). Со стороны пользователей модуля “Карточка студента” это приводит к увеличению времени, затрачиваемого на выполнение процесса отправки на согласование проекта приказа, поскольку:

1) В ходе данного процесса модуль “Обучающиеся” осуществляет синхронный вызов удаленного метода, с целью записи в удаленную БД записей; осуществляет формирование pdf-документа печатной формы проекта приказа; осуществляет копирование необходимых файлов в

удаленную сетевую папку. Это приводит к увеличению задержки на выполнение данного метода.

2) В случае сбоя при выполнении, пользователю необходимо повторно инициировать процесс отправки.

Для решения данных проблем, в разработанном автором решении был использован ранее упоминаемый подход, основанный на обмене сообщениями. При описании преимуществ очередей сообщений, в качестве одного из свойств выделялась возможность достижения асинхронного взаимодействия между интегрируемыми системами. С целью достижения данного эффекта были осуществлены следующие действия:

1) Осуществлен рефакторинг логики изменения статусов проектов приказов в рамках приложения “Обучающиеся”.
2) Логика взаимодействия с удаленными ресурсами вынесена из приложения в отдельный интеграционный сервис
3) Логика формирования pdf - документа вынесена в отдельный интеграционный сервис.

Благодаря упомянутым действиям была ликвидирована сильная связанность подсистемы “Обучающиеся” с вспомогательными интеграционными компонентами. Выполнение всех ресурсоемких операций было делегировано отдельному интеграционному сервису (см. пункты 3 и 4 схемы в приложении). Для предоставления сервису информации о необходимости выполнения синхронизации тех или иных проектов приказов был использован подход, основанный на обмене сообщениями.

Стоит упомянуть, что в исходном решении существовала возможность сохранения и отслеживания всех изменений статусов каждого из проектов приказов. Информация об изменениях сохранялась в отдельной таблице БД, в которой хранятся и сами целевые данные. В ходе реализации новой версии

интеграционного решения данная функциональность была переиспользована и расширена.

Прежде всего, был осуществлен рефакторинг исходного механизма работы со статусами в рамках приложения “Обучающиеся”.

Исходная логика подразумевала плоские правила переходов путем использования многочисленных условных операторов, “зашитых” в коде приложения. Соответственно, любое изменение или добавление статуса или правила перехода требовали перекомпиляции и выпуска новой версии приложения. Данный механизм был переработан. Взятым за основу новой реализации инструментом стал модуль, входящий в состав XAF-фреймворка, под названием State Machine Module [28]. Благодаря использованию данного модуля стала возможным настройка статусов, правил переходов, правил валидации в режиме исполнения через пользовательский интерфейс, что значительно упростило поддержку и доработку исходного решения.

После осуществленной модификации механизма, к уже существующим статусам проектов приказов были добавлены некоторые вспомогательные, такие как: “Подлежит отправке на согласование”, или “Подлежит отзыву”, факт перехода к которым, аналогично остальным, также сохранялся в БД.

Стоит отметить, что, исходя из последнего факта, можно говорить о том, что, в отличие от ранее рассматриваемого сценария интеграции подсистемы “Обучающиеся” и модуля библиотеки, в данном случае не возникает проблемы, связанной с необходимостью одновременной фиксацией изменений в целевых данных и данных, отражающий факт изменения целевых данных.

В итоге, после выполнения описанной работы, на основе сохраняемых в БД фактах изменения статусов, стала возможной реализация логики по определению проектов приказов, подлежащих синхронизации на стороне интеграционного сервиса.

Для обеспечения доставки событий в порядке их возникновения в целевой системе, каждому событию на уровне БД сопоставляется автоинкрементируемый целочисленный идентификатор, а в базе данных OfficeDocDB сохраняется номер последнего актуализированного события. Тем самым, помимо задачи упорядочивания, решается проблема потенциального дублирования событий при их синхронизации.

5.2.2. Задача упорядочивания событий в распределенной системе

Еще одна задача, которую было необходимо решить, обусловлена найденным недостатком в исходной реализации. В приложении **Ж** описана проблема возможного нарушения последовательности обновления статусов, изменение которых осуществляется параллельно в системах “Обучающиеся” и “Дело”. Так же, в указанном приложении был рассмотрен конкретный пример процесса, потенциально приводящего распределенную систему в противоречивое, недопустимое состояние. Помимо описания, для демонстрации данной проблемы была создана sequence-диаграмма, с которой можно ознакомиться в приложении Б.

Стоит отметить, что задача восстановления хронологической последовательности возникновения событий, происходящих во множестве элементов системы с целью установления между событиями причинно-следственной связи, или возможности обнаружения нарушения таковой, является достаточно классической для любой сложной многокомпонентной системы.

Для решения обнаруженной проблемы в рамках нового решения, был использован подход, основанный на алгоритме скалярных часов, также известный как Часы Лэмпорта [5].

С практической точки зрения, для использования данного алгоритма была осуществлена модификация существующей структуры данных проекта

приказа в подсистеме “Обучающиеся” - в каждую запись проекта приказа было добавлен целочисленное поле, которое изначально, при создании нового документа равно нулю. При инициировании пользователем процесса отправки проекта приказа на согласование или процесса отзыва проекта приказа данное поле увеличивается на единицу. В СЭД “Дело” вместе с основными данными проекта документа передается указанный идентификатор. В дальнейшем, при попытке синхронизации данных из СЭД “Дело” в подсистеме “Обучающиеся” осуществляется сверка значений данного идентификатора. Случай совпадения значений трактуется как принадлежность синхронизируемого изменения текущей версии документа. В случае же несовпадения, можно рассматривать входящее событие как устаревшее, и недопустимое к применению (актуализации).

Тестирование

С целью валидации полученных в рамках практической части данной работы интеграционных решений, автором, было выбрано интеграционное тестирование, позволяющего выявить ошибки или убедиться в корректности полученных результатов.

Предварительный процесс тестирования, осуществленный автором, заключался в ручном развертывании всей необходимой инфраструктуры и выполнении целевых интеграционных сценариев.

Ключевым недостатком данного подхода оказалась высокая трудоемкость процесса тестирования ввиду сложности интеграционных решений, заключающейся в потребности развертывания (как начальной, так и после каждого обновления какого-либо из компонент) большого количества компонентов.

Возможность дальнейшего улучшения качества тестирования и упрощения процесса его выполнения видится в использовании ранее рассмотренных в работе инструментов категории CI и CD.

А именно, в качестве следующего этапа развития процесса тестирования рассматривается:

1) Автоматизация выполнения интеграционных тестов
2) Автоматизация процесса развертывания разрабатываемого решения в рамках различного вида окружений (test, staging, production).

Стоит отметить, что еще одним, финальным видом проверки корректности работы разработанного решения, на данный момент не осуществленным, но планируемым к выполнению в ближайшее время, является внедрение разработанных решений в тестовую эксплуатацию, в рамках которой разработанная функциональность будет доступна малому, ограниченному набору пользователей. После чего, можно будет принимать

решение о переводе полученных результатов в промышленную эксплуатацию.

Дальнейшее развитие

Ранее в данной работе было упомянуто, что в рамках функционирования сложной многокомпонентной информационной системы университета очень часто возникает потребность в быстром обнаружении возникающих в системе сбоев и неполадок с целью их оперативного исправления. Помимо мониторинга и диагностики непосредственно целевых систем Университета, данные вопросы являются не менее актуальными и относительно эксплуатации различных интеграционных решений.

Так, для большинства рассмотренных решений достаточно типична ситуация, при которых логи “разбросаны” по множеству различных серверов, а процедуры развертывания, обновления, запуска и остановки компонентов осуществляются вручную. Что, в свою очередь ведет к повышению затрат ресурсов на поддержку данных систем со стороны разработчиков, и снижения качества эксплуатации для конечных пользователей.

В рамках данной работы было проведено исследование возможных способов решения такого рода проблем, однако, практическая часть, направленная на апробацию тех или иных решений с последующим введением в эксплуатацию, сама по себе является обособленной сложной задачей и вынесена за рамки диссертации.

Если говорить о конкретных примерах, то, потенциально, решению задач мониторинга и оркестровки сервисов может способствовать, например, изучение и внедрение сложных программных продуктов из категории ESB или Integration Suite. В качестве альтернативы, могут рассматриваться в настоящее время активно развивающиеся и широко распространенные при построении решений, основанных на микросервисной архитектуре,

инструменты построения кластеров из легковесных контейнеров, вроде Docker Swarm [29] или Kubernetes [30].

Для решения задачи создания централизованного хранилища логов может быть использован, например, достаточно популярный стек инструментов ELK [31], предназначенных для сбора логов, хранения и визуализации.

При этом, используемый в рамках данной работы подход основанный на интеграции систем через брокер сообщений, может быть использован в качестве решения для асинхронной доставки из систем-источников в хранилище [15]. Стоит отметить, что данное решение может быть использовано для сбора не только чисто “технической” информации, но и для сбора “бизнес-логов”, т.е. для фиксации действий пользователей в различных системах и предоставления инструментов для их анализа и просмотра, например, сотрудникам техподдержки.

Заключение

В рамках данной работы был осуществлен сбор информации о ключевых программных компонентах, входящих в состав ИС СПбГУ и способах их интеграции. Была собрана информация о технической реализации данных решений, выявлены основные классы проблем, и представлена оценка степени их влияния на эффективность поддержки, разработки и эксплуатации компонентов ИС.

С целью оценки возможностей улучшения существующих механизмов взаимодействия между компонентами, была исследована литература, посвященная вопросам интеграции корпоративных приложений.

Были изучены подходы и инструменты, направленные на упрощение тестирования и развертывания сложных многокомпонентных систем.

На основе изученной информации и инструментов, была осуществлена модификация двух существующих интеграционных решений с целью минимизации ресурсов на поддержку и эксплуатацию в сравнении с исходным вариантом. Была осуществлена оценка возможности переиспользования полученных результатов в ходе решения других интеграционных задач.

Результаты осуществленного тестирования полученных решений показали уменьшение ошибок в ходе эксплуатации и увеличение производительности отдельных компонентов ИС, что может говорить о достижении цели работы в их контексте.

Список литературы

1. Хоп Г., Вульф Б. Шаблоны интеграции корпоративных приложений //М.: Вильямс. – 2007. 672 с.
2. Bussler C. B2B integration: Concepts and architecture. – Springer Science & Business Media, 2013. 418 с.
3. Gilbert S., Lynch N. Perspectives on the CAP Theorem // Computer. – 2012. – Т. 45. – №. 2. – С. 30-36.
4. Таненбаум Э. и др. Распределенные системы. Принципы и парадигмы. – СПб.: Питер, 2003. 880 с.
5. Косяков М. С. Введение в распределенные вычисления // Санкт-Петербург: НИУ ИТМО. – 2014. 155 с.
6. Continuous Integration //URL: https://martinfowler.com/articles/continuousIntegration.html (дата обращения: 24.05.2017).
7. Stolberg S. Enabling agile testing through continuous integration //Agile Conference, 2009. AGILE'09. – IEEE, 2009. – С. 369-374.
8. Chen L. Continuous delivery: Huge benefits, but challenges too //IEEE Software. – 2015. – Т. 32. – №. 2. – С. 50-54.
9. Development Sandboxes: An Agile 'Best Practice' //URL: http://www.agiledata.org/essays/sandboxes.html (дата обращения: 24.05.2017).

10. Stubbs J., Moreira W., Dooley R. Distributed systems of microservices using docker and serfnode //Science Gateways (IWSG), 2015 7th International Workshop on. – IEEE, 2015. – С. 34-39.

11. Thönes J. Microservices //IEEE Software. – 2015. – Т. 32. – №. 1. – С. 116-116

12. Serrano N., Hernantes J., Gallardo G. Service-oriented architecture and legacy systems //IEEE software. – 2014. – Т. 31. – №. 5. – С. 15-19.

13. Zhu Y., An L., Liu S. Data updating and query in real-time data warehouse system //Computer science and software engineering, 2008 international conference on. – IEEE, 2008. – Т. 5. – С. 1295-1297.

14. de Hullu C. Evaluating .NET-Based Enterprise Service Bus Solutions. University of Twente. Department of Computer Science. – 2012. https://www.utwente.nl/ewi/trese/graduation_projects/2012/RT-003.pdf

15. Nguyen V. N., Tran V. C. An Efficient Log Management System //VNU Journal of Computer Science and Communication Engineering. – 2016. – Т. 32. – №. 2. – С. 43-48.

16. Взаимодействие компонентов ИС СПбГУ //URL:https://drive.google.com/file/d/0B3oR8zRXOvKUUHBMTmpjTHNiM1E/ (дата обращения: 24.05.2017).

17. Семенов С. П., Серегин В. Д., Татаринцев П. Б. Обеспечение слабой связанности интегрируемых информационных систем посредством асинхронного обмена сообщениями через сервисную шину //Вестник Югорского государственного университета. – 2011. – №. 3 (22).

18. Alonso G. et al. Exotica/FMQM: A persistent message-based architecture for distributed workflow management //Information Systems Development for Decentralized Organizations. – Springer US, 1995. – С. 1-18.

19. Choosing the Right ESB for Your Integration Needs //URL: https://www.infoq.com/articles/ESB-Integration (дата обращения: 24.05.2017).

20. RabbitMQ Management Plugin //URL:https://www.rabbitmq.com/management.html (дата обращения: 24.05.2017).

21. NServiceBus license //URL: https://particular.net/licensing (дата обращения: 24.05.2017).

22. Continuous integration and deployment /URL:https://www.visualstudio.com/en-us/docs/build/overview (дата обращения: 24.05.2017).

23. SQL Server Service Broker //URL:https://msdn.microsoft.com/ru-ru/library/bb522893.aspx (дата обращения: 24.05.2017).

24. About Change Data Capture (SQL Server) //URL:https://docs.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server (дата обращения: 24.05.2017).

25. Joona-Pekka Kokko. Distributed document migration between ERP systems by means of messaging and event sourcing // Lappeenranta University of Technology. 2013. URL:http://www.doria.fi/handle/10024/94341 (дата обращения: 24.05.2017).

26. Customize the Audit Trail System
//URL:https://documentation.devexpress.com/#eXpressAppFramework/CustomDocument112783 (дата обращения: 24.05.2017).

27. XAF Audit Trail
URL:https://www.devexpress.com/Support/Center/Question/Details/T320989 (дата обращения: 24.05.2017).

28. State Machine Module Overview
//URL:https://documentation.devexpress.com/#eXpressAppFramework/CustomDocument113713 (дата обращения: 24.05.2017).

29. Swarm mode overview
//URL:https://docs.docker.com/engine/swarm/ (дата обращения: 24.05.2017).

30. Kubernetes
//URL: https://kubernetes.io/ (дата обращения: 24.05.2017).

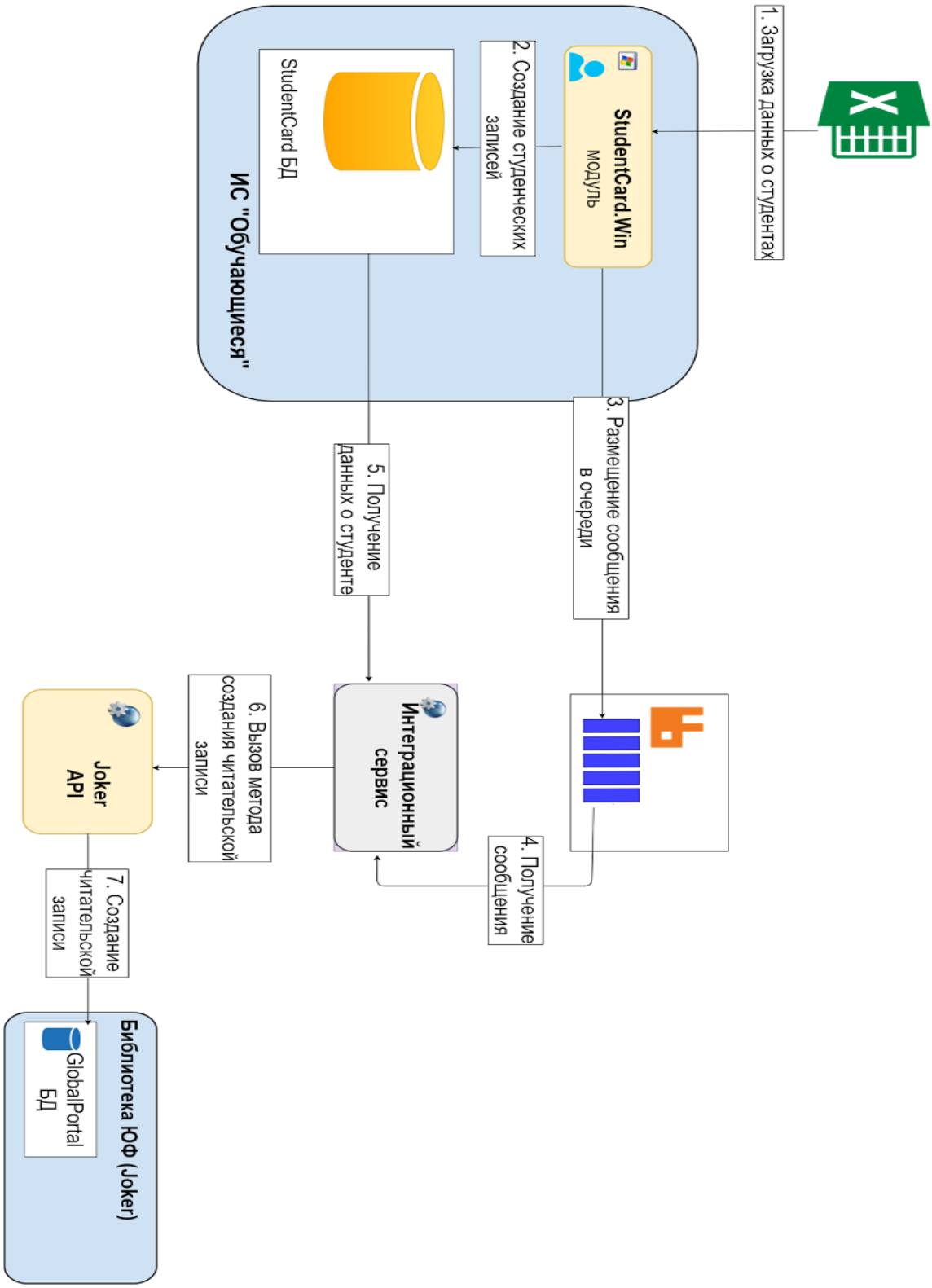
31. The Open Source Elastic Stack
//URL: https://www.elastic.co/products (дата обращения: 24.05.2017).

32. Betts D. et al. Exploring CQRS and Event Sourcing: A journey into high scalability, availability, and maintainability with Windows Azure. – 2013.

33. JBoss Fuse or ETL?
//URL:http://wei-meilin.blogspot.ru/2015/04/jboss-fuse-jboss-fuse-or-etl.html (дата обращения: 24.05.2017).

Приложение А

Процесс взаимодействия модуля “Карточка студента” (подсистема “Обучающиеся”) и ЭБИС.



Приложение Б

Обзор модуля “Карточка студента” и СЭДД “Дело”.

Подсистема “Обучающиеся”

Модуль “Карточка студента”, (или подсистема “Обучающиеся”) (техническое название - StudentCard) - настольное (win) приложение (win-приложение) для просмотра и редактирования данных о студентах. Разрабатывается и поддерживается в рамках университетской инициативы по разработке и внедрению информационной системы “Обучение”, включающей в себя, помимо указанного модуля, еще как минимум 7 систем, среди которых есть такие, которые известны и эксплуатируются лишь в рамках внутренних процессов университета, например, ИС “Дисциплины”, ИС “Учебные планы”, ИС “Педагогические поручения”, ИС “Выпуск”, и такие, которые очень хорошо известны и которые активно используются, в том числе, непосредственно абитуриентами и студентами университета, например ИС “Прием” и ИС “Электронное расписание”.

Возвращаясь к описанию непосредственно “Карточки студента” стоит упомянуть, учет и ведение какого рода данных и какой целевой группой сотрудников университета осуществляется с использованием рассматриваемого приложения. Стоит отметить, что данное приложение разрабатывается и поддерживается как глобальная общеуниверситетская система, введенная в эксплуатацию взамен ранее активно и очень долгое время используемым обособленным индивидуальным для каждого из университетских направлений (факультетов) экземпляров системы учета студенческих данных “Студент 2000”.

Изначально “Карточка студента” разрабатывалась как замена “Студент 2000”, где основной целевой группой пользователей являлись сотрудники

учебных отделов Университета. Основной целевым классом данных, для учета и ведения которых была внедрена “Карточка студента”, являются персональные и студенческие данные. К их числу можно отнести такие общие персональные данные как, например, ФИО, паспортные данные и сведения, которые характеризуют конкретную персону как конечного участника образовательного процесса, т.е. идентифицируют конкретную персону как студента. То есть, это такие условно-статические данные как направление обучения, учебный план, профиль, номер студенческого билета, так и данные, определяющие прогресс в ходе обучения в динамике, например, информация о группе, текущем статусе студента (студент, выпускник, отчисленный и т.п.), набор ведомостей и оценок, полученных студентом по ходу обучения, выплачивается или не выплачивается стипендия и т.п.

Стоит отметить, что выше приведенный список далеко не исчерпывающий - спроектированная архитектура описываемого модуля предоставляет достаточно гибкий механизм по добавлению или модификации уже существующих структур данных - так, в достаточно короткие сроки, помимо непосредственно сотрудников учебных отделов, к эксплуатации данной системы присоединились и сотрудники других подразделений, например, сотрудники управления по работе с молодежью (УРМ), факультета военного обучения и др, которые осуществляют сбор и ведение данных, например, по студенческим заявкам на получение материальной помощи, или по заявкам для поступления на факультет военного обучения.

Возвращаясь к тому факту, что все же рассматриваемая система изначально разрабатывалась именно для учета студенческих данных, стоит упомянуть об одном основополагающем ограничении, которое является ключевым моментом всего дальнейшего повествования. Речь идет об

ограничении при эксплуатации данной системы, возникающем в силу специфики бизнес-процессов университета. А именно, внесение подавляющего числа изменений (в том числе создание и удаление) в данные, относящиеся к контингенту (студентам), должно сопровождаться процессом создания, согласования и утверждения соответствующего данным изменениям конкретного нормативного акта (как правило, приказа).

На ранней стадии эксплуатации возможности “Карточки студента” не предоставляли средств для какой-либо минимальной работы с приказами, отсутствовали средства, позволяющие хотя бы вносить информацию о существующих в университете приказах и “подвязывать” к этим приказам события об изменении тех или иных данных в системе. Только к лету 2016 года был разработан механизм, позволяющий создавать и осуществлять учет нормативно-правовых документов в системе, однако, и он не удовлетворял всех потребностей в реализации такого бизнес-процесса, как работа с приказами. Отсутствовали ключевые средства, позволяющие осуществлять процесс согласования, утверждения или отзыва приказов.

Однако, на протяжении данного промежутка времени и по настоящий момент в университете уже была внедрена и активно используется другая информационная система, удовлетворяющая всем требованиям по работе с нормативными документами - система электронного документооборота “Дело” (далее - СЭД “Дело””) описание которой будет представлено далее в работе.

Система электронного документооборота “Дело”

Система электронного документооборота СЭДД “Дело” - программное решение, включающие набор инструментов для управления

документооборотом и делопроизводством, позволяющее эффективно решать следующие задачи:

- регистрация входящей и исходящей корреспонденции
- совместная работа и подготовка проектов документов
- построение сложных маршрутов движения под различные бизнес-процессы
- организация ссылок между документами, проектами и поручениями
- единая рабочая среда для удаленных подразделений и филиалов и др.

На настоящий момент СЭДД “Дело” является основным программным решением, используемым в университете для управления документооборотом, в частности для работы с таким видом организационно-распорядительных документов, как приказы.

Приложение В

Предпосылки реализации интеграции между модулем “Карточка студента” и СЭДД “Дело”

Для того чтобы разобраться в предпосылках возникновения потребности в реализации интеграционного решения между двумя системами, стоит более детально описать особенности осуществления документооборота в СПбГУ в разрезе управления приказами. Как было упомянуто в приложении А, в качестве основной системы для создания, выпуска и учета приказов в течение длительного времени и по настоящий момент используется стороннее программное обеспечение СЭД “Дело”.

В общем случае в рамках процесса работы с приказами можно выделить несколько стадий:

а) Создание проекта приказа

б) Согласование проекта приказа

в) Утверждение проекта приказа

г) Выпуск приказа

Непосредственно сам проект приказа, как правило, состоит из нескольких элементов:

а) Печатная форма проекта документа

б) Документы-основания

В течение длительного времени конечному пользователю для создания нового проекта приказа было необходимо найти определенный файл с шаблоном печатной формы создаваемого проекта приказа, осуществить его ручное заполнение, самостоятельно внести в печатную форму информацию о соответствующих данному проекту приказа документах-основаниях и

загрузить в систему СЭД “Дело”. Ситуация осложнялась отсутствием удобных инструментов для централизованного доступа, создания и редактирования шаблонов печатных форм и документов-оснований. Более того, в некоторых случаях, после утверждения приказов определенных типов возникала потребность во внесении соответствующих изменений в целевые данные, расположенные в других системах. Например, после выхода приказа о зачислении студентов, или переводе на другой курс, сотрудники были вынуждены вручную вносить изменения в соответствующие информационные системы по учету этих данных. Таким образом, осуществлялась двойная работа в виде составления проекта приказа с указанием необходимых изменений (напр., о переводе конкретных студентов на другой курс), а после утверждения и выпуска приказа - внесение этих же изменений в целевые системы.

Стоит отметить, что ситуация частично улучшилась после введения в эксплуатацию модуля “Карточка студента”, который предоставляет:

1) Удобные инструменты для централизованного учета данных контингента
2) Инструменты для работы с различного рода документами-основаниями (например, создание, редактирование и хранение ведомостей, справок, хранение копий представленных студентом документов и т.п.)
3) Инструменты для работы с шаблонами и печатными формами проектов приказов, а также средства для создания и учета данных и метаданных, касающихся самих проектов приказов (включая функциональность по автоматизированному созданию печатных форм на основе определенного шаблона и автозаполнение целевыми данными).
4) Функциональность по автоматизированному изменению целевых данных (как правило, данных контингента) при утверждении соответствующего проекта приказа в рамках системы “Обучающиеся”.

Таким образом, была решена часть озвученных ранее проблем, связанных с созданием ,поиском и заполнением соответствующего конкретному проекту приказа шаблона печатной формы, или же с внесением, редактированием в системе и использованием различного рода документов-оснований. Однако, предоставляемая системой “Обучающиеся” функциональность все же не позволяла решить ряд проблем, и вынуждала пользователей к выполнению некоторого повторяющегося набора действий в двух системах.

Например:

1) Сохранилась потребность в ручном создании проекта приказа в СЭД “Дело”, ручной загрузке печатной формы и документов оснований (хотя процесс их создания и учета все же был упрощен).
2) Для внесения изменений в целевые данные, пользователям необходимо либо внести эти изменения вручную, либо создать в системе корректный проект приказа (данные которого должны совпадать с утвержденным в СЭД “Дело” приказом) и утвердить его, после чего внесение изменений произойдет в автоматическом режиме.

Стоит отметить, что помимо увеличения трудозатрат сотрудников университета, вызванных необходимостью эксплуатации двух систем и дублированием действий, связанных с внесением данных, в ходе такой эксплуатации возникает еще одна проблема, связанная с потенциальной несогласованностью данных в двух системах. Как правило, это касается “второстепенной” относительно работы с приказами системы “Обучающиеся”, где могут находиться некачественные (неполные и/или некорректные) данные относительно приказов и их статуса.

Именно с целью нивелирования вышеописанных проблем, в ходе

совместной работы разработчиков “Карточки студента” и СЭДД “Дело” было разработано решение, направленное на выполнение задачи интеграции двух систем для обеспечения согласованности данных и уменьшение трудозатрат пользователей на их внесение и актуализацию.

Приложение Г

Анализ исходного механизма взаимодействия между модулем “Карточка студента” и СЭДД “Дело”.

Как было упомянуто в приложении В, одной из основных проблем, на решение которой направлено интеграционное решение, является необходимость для пользователей работать и поддерживать похожие наборы данных одновременно в двух системах.

Исходное интеграционное решение предоставляет функциональность по автоматической синхронизации данных, связанных с проектами приказов между “Карточкой студента” и “Дело”. Если говорить более конкретно, то пользователь, используя “Карточку студента имеет возможность создать проект приказа и отправить его на регистрацию, с целью последующего согласования и утверждения в СЭД “Дело”. При этом интеграционное решение берет на себя задачу по созданию проекта приказа в СЭД “Дело” на основе данных из “Карточки студента”, включая перенос из системы в систему печатной формы проекта приказа и файлов соответствующих документов-оснований.

Также, помимо создания новых проектов приказов, интеграционное решение обеспечивает процесс взаимодействия между системами в рамках таких процессов как утверждение проекта приказа, или его отзыв с возможностью внесения изменений и повторной отправкой на согласование. Поддержка указанных процессов обеспечивается путем возможности “общения” двух систем в обе стороны. Иными словами, не только “Карточки студента” имеет возможность информировать СЭД “Дело” о создании нового проекта приказа или необходимости отзыва, но и СЭД “Дело” “передает” информацию об изменении статуса проекта приказа (например, факт

регистрации проекта приказа в системе, согласования, утверждения или необходимости внесения исправлений).

Остановимся и представим обзор текущей реализации интеграционного решения для взаимодействия между двумями системами. В приложении Д представлена схема архитектуры исходной реализации.

Опишем основные компоненты решения, изображенные на схеме:

- Подсистема“Обучающиеся” (“Карточка студента”) и СЭД “Дело” - целевые информационные системы, описание которых представлено ранее в работе.

- WCF Service - удаленный веб-сервис, построенный с использованием платформы Windows Communication Foundation и предоставляющий набор методов на основе Simple Object Access Protocol (SOAP). С помощью предоставляемых сервисом методов система “Обучающиеся” передает информацию о возникновении определенных событий в этой системе, в частности, создание нового проекта приказа, подлежащего согласованию, или отзыв ранее созданного проекта приказа.

- OfficeDocDB - промежуточная БД, в которую помещаются данные о возникновении событий в системе “Обучающиеся”.

- Файловое хранилище “Документы-основания и шаблоны приказов” - директория общего доступа, в которую помещаются такие файлы как печатные формы проектов приказов и документы-основания, созданные и хранящиеся в подсистеме “Обучающиеся” для последующего размещения и использования в СЭД “Дело”.

- “Очередь сообщений” - логическая сущность, выполняющая задачу хранения набора сообщений, отражающих события изменения состояний проектов приказов в СЭД “Дело” для последующей

актуализации на основе этих сообщений данных в системе “Обучающиеся”.

- DeloRestService - веб-сервис, построенный на основе технологии ASP.NET Web Api и предоставляющий методы для актуализации состояния (статусов) проектов приказов в системе “Обучающиеся” на основе сообщений, расположенных в компоненте “Очередь сообщений”.
- Интеграционные сервисы - набор программных компонентов, предназначенных для асинхронной доставки данных о возникших в системах изменениях в другие системы, в качестве входных источников использующих данные из промежуточных хранилищ (базы OfficeDocDB и очереди сообщений).

Для лучшего понимания процесса взаимодействия описанных систем рассмотрим пример выполнения такого сценария как создание проекта приказа в системе “Обучающиеся” с последующей отправкой на согласование в систему “Дело”. С иллюстрацией описываемого процесса можно ознакомиться в **приложении Д**. Опишем основные шаги данного процесса:

1) Пользователь в системе “Обучающиеся” с использованием определенного интерфейса формирует и сохраняет новый проект приказа, формируя набор связанных с данным проектом приказа документов-оснований и заполняя все необходимые данные.
2) Пользователь через интерфейс системы направляет проект приказа на согласование в систему “Дело”. В этот момент выполняется следующий набор действий:

a) На основе заполненных пользователем данных о проекте приказа формируется pdf - документ печатной формы проекта приказа.

b) Данный pdf-документ вместе с файлами документов-оснований размещается в специальной папке общего доступа, расположенной на определенном удаленном сетевом диске (**шаг №1** схемы)

c) Вызывается удаленный метод WCF сервиса, в который в качестве параметров передается некоторый набор метаданных, сигнализирующий об отправке проекта приказа на согласование. В набор этих данных помимо бизнес-информации (списки согласующих лиц, информация о подписанте и т.п.) включается путь до отправленных ранее в сетевую папку необходимых файлов (**шаг №2**)

d) Вызванный метод сервиса осуществляет запись переданной информации в определенный набор структур базы данных OfficeDocDB

e) Статус проекта приказа в системе “Обучающиеся” устанавливается (процессом системы) в значение “Отправлен на согласование” (либо пользователю выводится сообщение об ошибке в случае сбоя на одном из этапов)

f) Пользователю возвращается управление для работы с системой

3) Отдельный программный компонент (интеграционный сервис) асинхронно (относительно работы подсистемы “Обучающиеся” и СЭД “Дело”) осуществляет извлечение из базы OfficeDocDB внесенной ранее информации и на основе полученных данных создает в системе

СЭД “Дело” новый проект приказа (внутреннее название - рабочая карточка проекта документа - далее РКПД) (**шаг №3**)

4) После регистрации проекта документа системой “Дело” формируется сообщение о возникновении события изменении статуса проекта приказа (в частности, создание новой РКПД тоже является событием), а именно, событие о переводе РКПД в статус “Зарегистрирован”.
   Данное событие помещается в очередь сообщений (**шаг №4**).

5) Отдельный интеграционный сервис с некоторой периодичностью (в исходном решении период составляет 10 минут) обновляет статус проекта приказа в системе “Обучающиеся” на основе сообщений из очереди путем вызова соответствующего метода сервиса DeloRestService (**шаг №5**).

Благодаря выполнению вышеописанного алгоритма пользователь, создавший проект приказа в системе “Обучающиеся” освобождается от необходимости ручной загрузки проекта приказа в СЭД “Дело”. К тому же пользователь имеет возможность отслеживать текущее состояние проекта документа из системы “Обучающиеся” не прибегая к необходимости обращения к другой системе. Более того, создание в системе “Обучающиеся” проекта приказа в совокупности с автоматизированным механизмом синхронизации позволяют на основе утвержденных и выпущенных приказов автоматически изменять целевые бизнес-данные. Таким образом, на первый взгляд, спроектированный механизм успешно выполняет свои задачи, избавляя пользователей от необходимости дублирования действий по управлению приказами в двух системах и решая проблему потенциальной рассинхронизации релевантных друг другу данных.

Однако результаты дальнейшей проведенной автором

исследовательской работы, включающей в себя сбор обратной связи от пользователей, сотрудников технической поддержки и разработчиков, анализ открытых и закрытых bug-report-ов и более детальный анализ технического решения сигнализировали о наличии ряда проблем в текущем решении, последствиями которых являются дополнительные трудозатраты ресурсов пользователей в ходе эксплуатации системы и разработчиков вместе с сотрудниками техподдержки в ходе поддержки и доработки данной реализации.

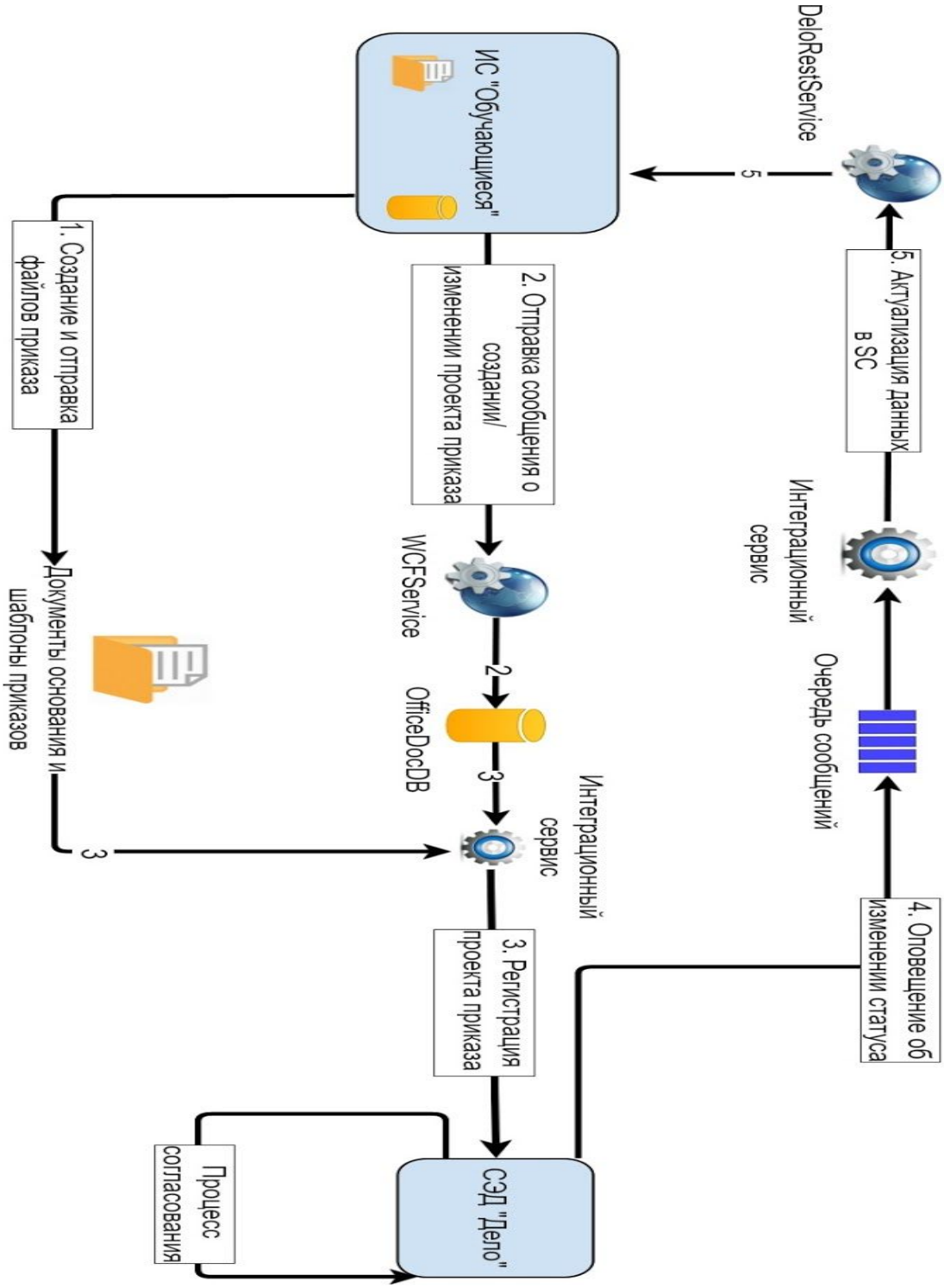
Весь набор возникающих проблем можно условно разделить на 4 категории:

- Отсутствие отказоустойчивости при недоступности (выхода из строя) одного из сервисов, задействованных в процессе интеграции (Ошибки типа “Отказано в доступе по пути” …. , недоступность сервиса во время обновления,)
- Проблемы, связанные с синхронизацией событий (статусов), возникающих в разных системах (а точнее с их рассинхронизацией)
- Проблемы, связанные с отсутствием транзакционности при выполнении операции, последовательно задействующей несколько компонентов интеграционного решения

С более детальным обзором упомянутых проблем можно ознакомиться в приложении Е.

Приложение Д

Иллюстрация процесса взаимодействия подсистемы “Обучающиеся” (“Карточка студента”) и СЭД “Дело” в рамках исходной реализации интеграционного решения.



Приложение Е

Анализ недостатков исходного механизма взаимодействия между подсистемой “Обучающиеся” (“Карточка студента”) и СЭДД “Дело”

Отсутствие отказоустойчивости

С неформальной точки зрения, под отсутствием отказоустойчивости можно подразумевать недоступность для пользователя функциональности по синхронизации приказов в случае выхода из строя одного из компонентов интеграционного решения, например сетевой папки для хранения документов, wcf-сервиса или промежуточной базы данных. Под “недоступностью”, опять же, с практической, неформальной точки зрения, можно понимать ситуацию, при которой запрос пользователя, адресованный системе, не может быть обработан в настоящий момент времени и должен быть в дальнейшем инициирован повторно.

Рассмотрим практические примеры, встречаемые на практике и приводящие к рассматриваемой проблеме:

1) Обновление WCF - сервиса
2) Недоступность удаленной сетевой папки, ввиду ошибок при настройке политик безопасности на уровне приложения

Если рассматривать данную проблему с формальной точки зрения, то отсутствием отказоустойчивости в рассматриваемом случае является неудовлетворение распределенной системы свойству устойчивости к разделению (partition tolerance), описанному в широко известной теореме Брюера (CAP теореме) [3]. Сама теорема говорит о том, что при построении любой распределенной системы можно обеспечить выполнение лишь любых

двух из трех свойств: доступность (availability), согласованность (availability), и устойчивость к разделению (partition tolerance).

Таким образом, можно говорить о том, что само по себе отсутствие у системы устойчивости к разделению, в том числе и в данном случае, является не допущенной в ходе проектирования и реализации систем, а лишь следствием альтернативного выбора доступных при построении системы вариантов.

Однако, при дальнейшем анализе распределенной системы на предмет удовлетворения тем или иным свойствам CAP-теоремы, было выявлено, что, различные части интеграционного решения реализованы по разному.

Так, в рамках всей архитектуры можно выделить два набора компонентов, реализация каждого из которых поддерживает два различных набора свойств.

Например, взаимодействие компонентов подсистемы “Обучающиеся”, WcfService, OfficeDocDB и сетевой папки удовлетворяет свойствам:

1) доступности, за счет конечного числа компонентов с предсказуемым временем отклика каждого из них (как успешного, так и с сообщением об ошибке)
2) согласованности, за счет синхронного вызова методов Wcf-сервиса из подсистемы “Обучающиеся”

С другой стороны, если рассмотреть обратный процесс, процесс доставки данных из СЭД “Дело” в “Обучающиеся”, то взаимодействие участвующих в этом процессе компонентов удовлетворяем уже иным свойствам, а именно устойчивости к разделению и доступности. Данное наблюдение следует из анализа архитектуры существующего решения. Действительно, актуализация как входящих, направляемых в “Дело” данных,

так и исходящих (в “Обучающиеся”) осуществляется в асинхронном режиме с помощью отдельных сервисов, запускаемых раз в определенный интервал времени. Таким образом, работоспособность в целом и время выполнения таких операций как согласование, утверждение проекта приказа и др. операции, возникающие в целевой системе “Дело” не зависят от состояния и доступности иных компонентов интеграционного решения, таких как промежуточная БД OfficeDocDB, подсистема “Обучающиеся” и др, за счет чего выполняются два вышеописанных свойства из CAP-теоремы. Однако, данная реализация не удовлетворяет свойству согласованности, поскольку операция изменения данных в системе “Дело” и процедура синхронизации этих данных в подсистеме “Обучающиеся” в совокупности не являются атомарными.

Исходя из вышесказанного, можно сделать вывод о том, что выполнение свойства согласованности данных между подсистемой “Обучающиеся” и промежуточной базой OfficeDocDB взаимен устойчивости к разделению не имеет смысла, поскольку, несмотря на это, в разрезе выполнения всей процедуры интеграции, свойство согласованности данных между подсистемой “Обучающиеся” и “Дело” не выполняется.

В данном случае, более оптимальным может являться вариант приведения всей распределенной системы к виду, при котором в рамках всей архитектуры будут выполнятся 2 фиксированных свойства. То есть, стоит выбор либо между устойчивостью к разделению и доступностью, либо доступностью и согласованностью.

Транзакционность

Проблема иной категории, обнаруженная при анализе текущего решения и подтвержденная набором существующих bug-report-ов, связана с

достаточно классической задачей атомарной записи набора изменений в нескольких независимых системах, то есть в рамках одной распределенной транзакции.

Данная проблема была обнаружена при взаимодействии подсистемы “Обучающиеся” с Wcf-сервисом, через который осуществляется внесение изменений в базу OfficeDocDb в ходе процесса отправки проекта приказа на согласование или отзыва.

Ранее в параграфе “Анализ текущего интеграционного решения” было представлено описание набора необходимых шагов, выполняемых при отправке пользователем проекта приказа на согласование. Так, при инициировании пользователем запроса об отправке документа, в рамках одного метода системой “Обучающиеся” осуществляются создание и копирование файлов в сетевую папку, вызов метода удаленного WCF-сервиса (для внесения изменений в базу OfficeDocDB), и, перед возвратом управления программой пользователю (в пункте *е*) смена статуса проекта приказа в БД системы “Обучающиеся”. Совокупность операций обращения к внешним ресурсам не является атомарной, тем самым допускается вероятность нарушения согласованности данных (например, отправка события в промежуточную базу завершается успешно, а при обновлении статуса в системе возникает ошибка).

Определение порядка изменения статусов между системами

В ходе эксплуатации текущего решения рассинхронизация данных может возникать не только в случае сбоя при инициировании запроса пользователем со стороны подсистемы “Обучающиеся”.

Для распределенной системы достаточно классической является задача восстановления хронологической последовательности возникновения событий, происходящих во множестве элементов системы с целью установления между событиями причинно-следственной связи, или возможности обнаружения нарушения таковой.

В рассматриваемой системе такого рода элементами, генерирующими события, являются системы “Обучающиеся” и “Дело”, а непосредственно событиями, последовательность возникновения которых необходимо определять, являются любые изменения состояния проекта приказа в рамках каждой из двух интегрируемых систем.

Анализ реализации текущего интеграционного решения показал, что на данный момент в используемом решении отсутствует механизм, позволяющий точно определять порядок возникновения событий. Для того, чтобы понять возможные последствия, рассмотрим следующий пример.

В приложении В представлена sequence-диаграмма, демонстрирующая процесс взаимодействия между основными компонентами интеграционного решения в ходе отправки проекта приказа на согласование. Описание основных акторов, изображенных на диаграмме, было представлено ранее в работе. Стоит отметить, что на диаграмме и в ходе дальнейшего описания умышленно не упоминается wcf-сервис, поскольку его роль сводится к получению синхронных запросов от подсистемы “Обучающиеся” и вставке необходимых данных в базу OfficeDocDB. Таким образом, для упрощения можно считать, что подсистему “Обучающиеся” обращается к базе напрямую.

Итак, опишем более детально проиллюстрированный на диаграмме

процесс, чтобы понять, возникновение каких проблем следует из его выполнения.

1. Проект приказа направляется пользователем на согласование. В подсистеме “Обучающиеся” происходит смена статуса: "Черновик->Отправлен на согласование".
2. В СЭД Дело создается новая РКПД. После этого, осуществляется смена статуса в “Обучающиеся”: "Отправлен на согласование” -> "Зарегистрирован в Дело".
3. В СЭД “Дело” осуществляется процесс согласования. Стоит отметить, процесс не имеет четких временных границ и его длительность может варьироваться от нескольких минут до нескольких дней.
4. Пользователь через систему “Обучающиеся” отзывает проект приказа с согласования. Происходит смена статуса "Зарегистрирован в Дело" -> "На доработке". Сообщение о необходимости отзыва проекта приказа размещается в промежуточной базе данных.
5. Пользователь вносит правки в проект приказа и отправляет на повторное согласование. Смена статуса "Черновик”->”Отправлен на согласование".
6. В СЭД “Дело”, до получения информации о необходимости отзыва предыдущей версии проекта приказа и согласования новой на основе сообщений из промежуточной базы, осуществляется согласование первой версии проекта приказа (уже отозванной на стороне подсистемы “Обучающиеся”).
7. Информация о событии согласования уже неактуальной версии проекта приказа направляется в подсистему “Обучающиеся”, и как следствие,

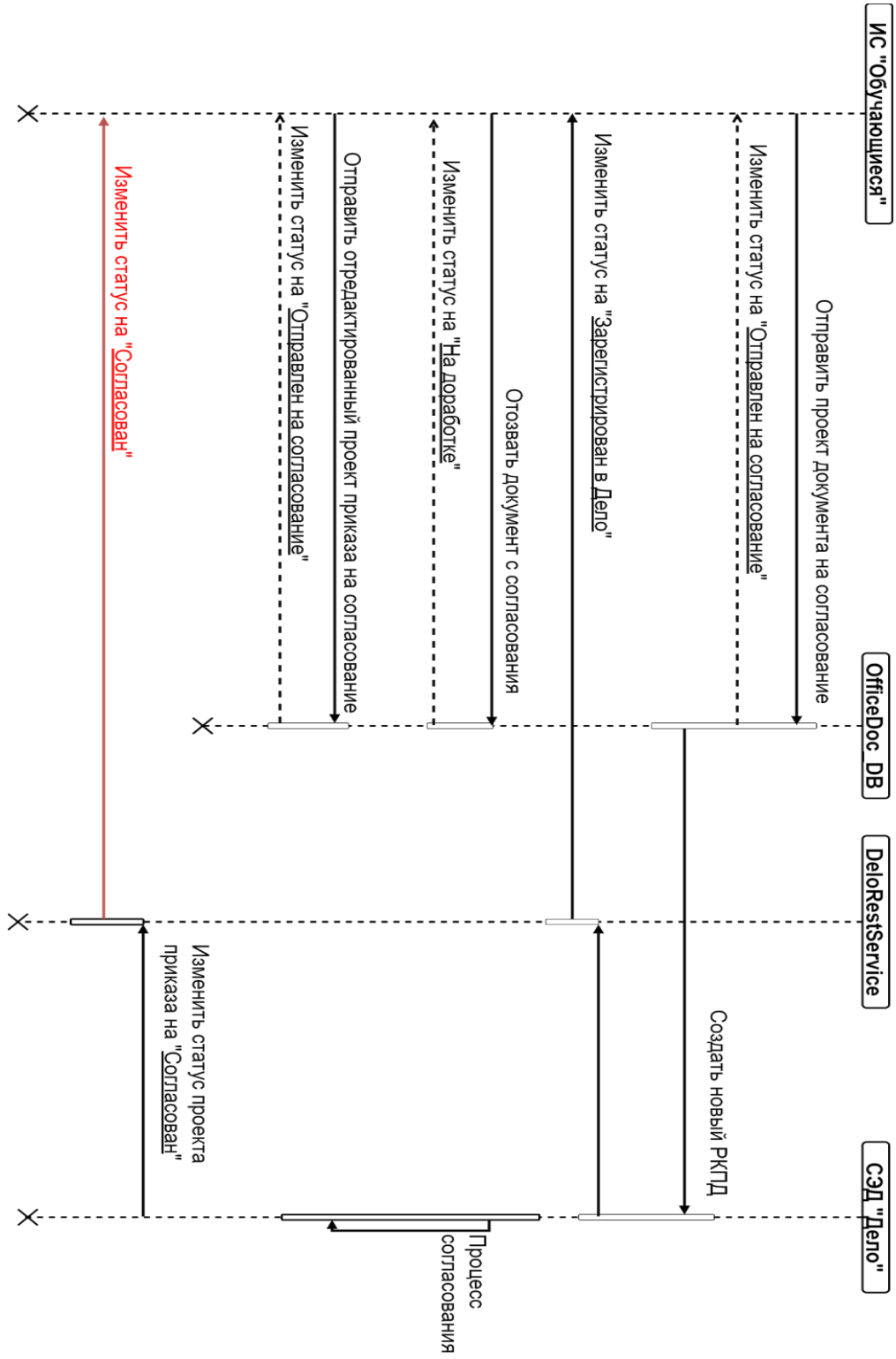
происходит смена статуса “Отправлен на согласование” -> “Согласован”.

Смена статуса, отмеченная в пункте №7 является заведомо недопустимой и приводит данные в некорректное состояние, поскольку статусом “Согласован” отмечается тот проект приказа, который, на самом деле может быть еще только доставлен в СЭД “Дело”. Такого рода конфликты не предупреждаются и не фиксируются системой, и единственным способом обнаружения и разрешения несогласованности является ручная сверка данных каждого проекта приказа пользователем в двух компонентах - подсистеме “Обучающиеся” и СЭД “Дело”.

Стоит отметить, что одним из вариантов разрешения такого рода коллизий может является достаточно классическое решение, заключающееся в использовании логических часов для определения порядка возникновения событий в нескольких системах.

Приложение Ж

Демонстрация проблемы рассогласования состояния проекта документа



Приложение 3

Архитектура разработанного интеграционного решения для обеспечения взаимодействия подсистемы “Обучающиеся” (“Карточка студента”) и СЭД “Дело”