**ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ**

**ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

**Факультет компьютерных наук**

**Кафедра «Информационные системы»**

*<Система интеграции Web-сайтов рекрутмента>*

071900 (230201) Информационные системы и технологии

*<Информационные системы и телекоммуникации>*

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_Тюкачев Н.А. к.ф.-м.н., доцент \_\_.\_\_.2013

Студент \_\_\_\_\_\_\_\_\_\_Петрушин И.А. \_\_.\_\_.2013

Руководитель \_\_\_\_\_\_\_\_\_\_Сапегин С.В. к.т.н., доцент \_\_.\_\_.2013

Воронеж2013

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc366594968)

[ВВЕДЕНИЕ 3](#_Toc366594969)

[1. ПОСТАНОВКА ЗАДАЧИ 5](#_Toc366594970)

[2. АНАЛИЗ ЗАДАЧИ 6](#_Toc366594971)

[2.1 Анализ предметной области 6](#_Toc366594972)

[2.2 ESB как подход к интеграции на уровне данных 7](#_Toc366594973)

[2.3 Очереди Сообщений 8](#_Toc366594974)

[2.4 Анализ требований 9](#_Toc366594975)

[2.4.1 Компонент “ Web Service Business-Logic ” 11](#_Toc366594976)

[2.4.1.1 Диаграмма вариантов использования 12](#_Toc366594977)

[2.4.1.2 Требования к созданию профиля резюме 13](#_Toc366594978)

[2.4.2 Компонент “Message Broker ” 14](#_Toc366594979)

[2.4.2.1 Архитектура и общие термины JMS 15](#_Toc366594980)

[2.4.2.2 Модель взаимодействия точка-точка 16](#_Toc366594981)

[2.4.2.3 Модель взаимодействия издание-подписка 17](#_Toc366594982)

[2.4.2.4 Требования к JMS провайдеру 18](#_Toc366594983)

[2.4.3 Rules Repository 19](#_Toc366594984)

[2.4.3.1 Диаграмма отношений сущностей 19](#_Toc366594985)

[3. АППАРАТНЫЕ И ПРОГРАММНЫЕ СРЕДСТВА 20](#_Toc366594986)

[3.1 Требования к составу и параметрам технических средств 21](#_Toc366594987)

[3.2 Требования к программным средствам, используемым системой 21](#_Toc366594988)

[4. РЕАЛИЗАЦИЯ 22](#_Toc366594990)

[4.1 Объектная модель профиля резюме 22](#_Toc366594991)

[4.2 Компонент “WS Business-logic” 29](#_Toc366594997)

[4.2.1 Диаграмма классов 29](#_Toc366594998)

[4.3 Компонент “Rules Repository” 30](#_Toc366594999)

[4.3.1 Диаграмма Базы Данных 30](#_Toc366595000)

[4.4 Компонент “Message Broker” 31](#_Toc366595002)

[4.4.1 Конфигурация очередей сообщений и маршрутизации 32](#_Toc366595003)

[4.4.2 Проверка сообщений 34](#_Toc366595004)

[4.4.3 Обработка исключительных ситуаций и ошибок 35](#_Toc366595005)

[4.5.1 Трансформация сообщений 37](#_Toc366595006)

[4.5.1.1 Cлой доступа к данным репозитирия правил транформации 38](#_Toc366595007)

[4.5.2 Передача преобразованных запросов к сторонним системам. 41](#_Toc366595008)

[4.5.2.1 Selenium WebDriver 41](#_Toc366595009)

[4.5.2.2 Диаграмма классов 42](#_Toc366595010)

[4.5.3 Диаграмма последовательностей 45](#_Toc366595011)

[ЗАКЛЮЧЕНИЕ 46](#_Toc366595012)

[СПИСОК ЛИТЕРАТУРЫ 47](#_Toc366595013)

[ПРИЛОЖЕНИЕ 1 48](#_Toc366595014)

[ПРИЛОЖЕНИЕ 2 53](#_Toc366595015)

[ПРИЛОЖЕНИЕ 3 55](#_Toc366595017)

[ПРИЛОЖЕНИЕ 4 60](#_Toc366595019)

ВВЕДЕНИЕ

Ни для кого не секрет, что наше общество и экономика перешагнули стадию постиндустриализма и стали первооткрывателями новой ступени развития – стадии Информационного общества. С каждым днём растёт роль информации, знаний и новых технологий в жизни каждого современного человека. Рамки глобального информационного пространства расширяются с каждым годом, что позволяет людям эффективно взаимодействовать друг с другом, получать доступ к мировым информационным ресурсам и удовлетворять свои потребности в услугах.

Бурное технологическое развитие открыло миру огромное множество различных видов аппаратных и программных решений, которые в синтезе позволяют не только увеличить скорость обработки информации и сделать человека сверх коммуникабельным, но и оптимизировать затраты, связанные с эксплуатацией данных технологий.

На этом фоне появляются проблемы интеграции уже существующих продуктов, реализованных с помощью концептуально разных технологических средств и не имеющих общего, понятного всем протокола взаимодействия.

Ответной реакцией на такие удручающие перспективы послужило создание новых методологий и принципов, которые в итоге легли в основу универсальных интеграционных механизмов и стандартов.

В реальной жизни цель интеграции сводится к оптимизации каких-либо бизнес-процессов и/или объединение нескольких информационных ресурсов в одно целое. Во втором случае, количество до сих пор нерешённых задач пугает своим масштабом.

На данный момент существует много различных интернет ресурсов, на которых соискатель работы может выложить своё резюме в целях получить предложение о трудоустройстве от какой-либо заинтересованной в нём компании. Как правило, эти же ресурсы предоставляют аналогичную возможность публикации вакансий HR) отделам компаний. Как правило, соискатели \ работодатели публикуют идентичные резюме \ вакансии, сразу на нескольких ресурсах, чтобы увеличить вероятность ответных предложений \ соглашений.

Как было отмечено ранее, на сегодняшний день существует большое количество рекрут-сайтов. Самые популярные из них:

* Русские: headhunter.ru , job.ru, superjob.ru, rabota.ru.
* Зарубежные: monster.com, dice.com.

Таким образом, для управления своим резюме\вакансией пользователю необходимо заходить на каждый сайт, где был опубликован профиль и проделывать одну и ту же работу, в целях обновления, по каким-либо причинам не актуальной на текущий момент, информации. Самые простые примеры таких ситуаций:

* Соискатель работы хочет добавить в своё резюме несколько новых навыков и изменить желаемую заработную плату;
* Работодателю необходимо изменить требования, предъявляемые к какой-либо должности.

Количество времени, требуемое для обновления информации, прямо пропорционально количеству ресурсов. Очевидно, что это далеко не оптимальный с точки зрения производительности и не удобный, из-за монотонности действий, способ. Но другого выбора у пользователей нет. Данные интернет ресурсы не предоставляют никаких программных интерфейсов для управления профилями и далеко не все поддерживают их импорт/экспорт.

В рамках данной курсовой работы будет реализована система синхронизации информации между рекрут - сайтами, услугами которых, пользуется пользователь.

1. ПОСТАНОВКА ЗАДАЧИ

**Требуется разработать систему, которая должна предоставлять клиенту:**

1. **Программный интерфейс для взаимодействия с системой.**
2. **Правила формирования запроса к системе, включая формат представления данных экземпляра профиля-резюме.**
3. **Возможность управлять пользовательским профилем (публикация/обновление резюме на указанных web-ресурсах).**
4. **Надежный механизм хранения и синхронизации информации между системой и web-ресурсами, поддерживающий асинхронную модель передачи данных.**
5. АНАЛИЗ ЗАДАЧИ
   1. Анализ предметной области

Прежде чем строить интеграционное решение, необходимо четко понять, какой подход будет наиболее эффективен для решения стоящей перед лицом задачи.

Существует три подхода к интеграции информационных систем:

1. Интеграция на уровне данных. Суть данного подхода заключается в следующем: приложения работают независимо друг от друга, каждое использует свой набор данных. В случае необходимости осуществляется обмен данными между приложениями. При этом, если обмен данными осуществляется путем вызова сервисов или отправки/получения сообщений.
2. Интеграция на уровне бизнес-процессов. Суть данного подхода заключается в следующем: приложения выставляют сервисы, являющиеся интерфейсами к бизнес - логике данных приложений. Взаимодействие между приложениями реализовано в рамках бизнес-процесса, на отдельных шагах которого осуществляется вызов того или иного сервиса. Реализуется данный подход с помощью сервисной шины предприятия (ESB), которая занимается виртуализацией сервисов, предоставляемых приложениями, и решений класса Business Process Management System (BPMS), как правило основанных на языках BPEL или BPMN, которые реализуют логику процесса.
3. Интеграция на уровне композитных приложений. Бизнес-логика отдельного приложения строится путем вызова сервисов, предоставляемых как данным приложением, так и другими системами. Таким образом, на одном шаге бизнес-процесса могут взаимодействовать несколько сервисов, в то время как при интеграции на уровне бизнес-процессов на одном шаге процесса вызывается один сервис. Реализация композитных приложений осуществляется с помощью использования технологий Java Business Integration (JBI, JSR 208) или Service Component Architecture (SCA).

Отсюда можно сделать вывод, что наиболее целесообразным медом интеграции, в рамках поставленной задачи, является подход, описанный в первом пункте.

* 1. ESB как подход к интеграции на уровне данных

Самым оптимальным вариантом интеграции на уровне данных является использование сервисной шины предприятия - Enterprise Service Bus (ESB).

Основной принцип сервисной шины — концентрация обмена сообщениями между различными системами через единую точку, в которой, при необходимости, обеспечивается транзакционный контроль, преобразование данных, сохранность сообщений. Все настройки обработки и передачи сообщений предполагаются также сконцентрированными в единой точке, и формируются в терминах служб, таким образом, при замене какой-либо информационной системы, подключённой к шине, нет необходимости в перенастройке остальных систем.

Важной особенностью сервисной шины является поддержка двух режимов передачи информации: синхронного и асинхронного.

В рамках поставленной задачи важно наличие именно второй характеристики, так её принцип обеспечивает надёжную передачу данных между приложениями, за счёт возможности использования сервисной шиной в качестве транспортного механизма технологии очередей сообщений (Message Queue). Таким образом, одно приложение посредством ESB может передать данные другому приложению без необходимости вызова процедуры получателя и определенно без ожидания результата. Отправитель не обязан знать, как найти получателя. Он может просто направить данные в ESB и быть уверенным, что они будут переданы.

* 1. Очереди сообщений

Помимо возможности асинхронной передачи данных технология MQ обладает следующими важными свойствами:

* Слабое связывание — очереди сообщений создают неявные интерфейсы обмена данными, которые позволяют процессам быть независимыми друг от друга т.е вы просто определяете формат сообщений отправляемых от одного процесса другому.
* Избыточность — Очереди позволяют избежать случаев неэкономного использования ресурсов процесса(например памяти) в результате хранения необработанной (лишней информации) информации.
* Масштабируемость — очереди сообщений позволяют распределить процессы обработки информации. Таким образом, они позволяют легко наращивать скорость, с которой сообщения добавляются в очередь и обрабатываются.
* Эластичность и возможность выдерживать пиковые нагрузки — очереди сообщений могут выполнять роль своего рода буфера для накопления данных в случае пиковой нагрузки, смягчая тем самым нагрузку на систему обработки информации и не допуская ее отказа.
* Отказоустойчивость — очереди сообщений позволяют отделить процессы друг от друга, так что если процесс, который обрабатывает сообщения из очереди падает, то сообщения могут быть добавлены в очередь на обработку позднее, когда система восстановится.
* Гарантированная доставка — использование очереди сообщений гарантирует, что сообщение будет доставлено и обработано в любом случае (пока есть хотя бы один обработчик).
* Гарантированный порядок доставки — большая часть систем очередей сообщений способны обеспечить гарантии того, что данные будут обрабатываться в определённом порядке (чаще всего в том порядке в котором они поступили).
* Буферизация — очереди сообщений позволяет отправлять и получать сообщения при этом работая с максимальной эффективностью, предлагая буферный слой — процесс записи в очередь может происходить настолько быстро, насколько быстро это в состоянии выполнить очередь сообщений, а не обработчик сообщения.
* Понимание потоков данных — очереди сообщений позволяют выявлять узкие места в потоках данных приложения, легко можно определить какая из очередей забивается, какая простаивает и определить что необходимо делать — добавлять новых обработчиков сообщений или оптимизировать текущую архитектуру.
  1. Анализ требований

**Для реализации всей необходимой функциональности (пункт “Постановка задачи”) данная система должна состоять из четырёх основных компонентов (**Рисунок 2.5.1**):**

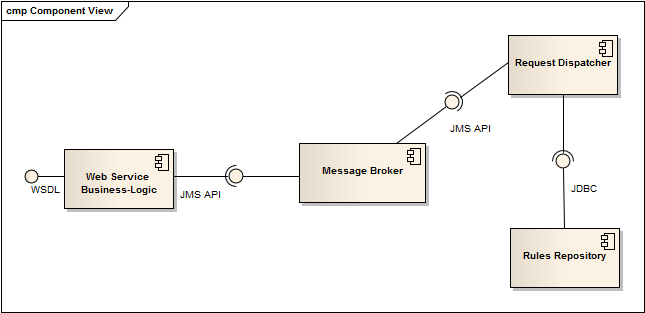


Рисунок 2.1 Диаграмма Компонентов

1. WS Business-Logic – компонент, предоставляющий клиенту универсальных программный интерфейс к бизнес-логике системы.
2. Message Broker – компонент системы, представляющий собой платформу для хранения, маршрутизации и трансформации запросов.
3. Request Dispatcher – компонент отвечающий за передачу обработанных системой запросов к сервисам назначения (web- сайтам рекрутмента).
4. Rules Repository – хранилище правил, в соответствии с которыми, система производит трансформацию запросов.

Для реализации данной системы целесообразно использовать стек Java - технологий, так как они позволяют строить кроссплатформенные и масштабируемые решения. Кроме того, на сегодняшний день существует довольно широкий спектр фрэймворков для быстрого и удобного построения программных решений, основанных на этой платформе.

* + 1. Компонент “ Web Service Business-Logic ”

В целях построения гибко расширяемой архитектуры системы, компоненты презентационного уровня и уровня бизнес-логики системы, должны быть максимально независимы друг от друга (Рисунок диаграмма компонентов). Данное требование, в рамках сервис-ориентированной архитектуры (SOA – Service Oriented Architecture), может быть реализовано на основе технологии веб-сервисов.

На сегодняшний день наибольшее распространение получили следующие протоколы реализации веб-сервисов:

* SOAP (Simple Object Access Protocol) ;
* REST (Representational State Transfer);
* XML-RPC (XML Remote Procedure Call).

В рамках поставленной задачи, целесообразно использовать первый протокол, так как SOAP более применим в сложных архитектурах, где взаимодействие с объектами выходит за рамки теории CRUD  (Create Read Update Delete), и требуется обеспечение надёжности и безопасности передачи данных.

SOAP включает в себя ещё два стандарта:

* **UDDI** ( Universal Description Discovery & Integration )описывает способ опубликования и обнаружения информации о Web-службах.
* WSDL (Web Services Description Language) — язык описания [веб-сервисов](http://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81" \o "Веб-сервис) и доступа к ним, основанный на языке [XML](http://ru.wikipedia.org/wiki/XML).

Таким образом, клиент должен иметь возможность обращаться к бизнес-функциям системы посредством WSDL файла, опубликованного сервисом.

* + - 1. Диаграмма вариантов использования

Как видно из диаграммы вариантов использования (Рисунок 2.4.1.1.1), сервис должен предоставлять клиенту следующие возможности:

* Получить список всех доступных значений для инициализации профиля-резюме, включая список всех доступных web-ресурсов, с которыми система поддерживает интеграцию.
* Выполнить запрос о публикации резюме, предварительно указав необходимую информацию об учётных записях web-ресурсов (логин/пароль) и данные о публикуемом профиле.

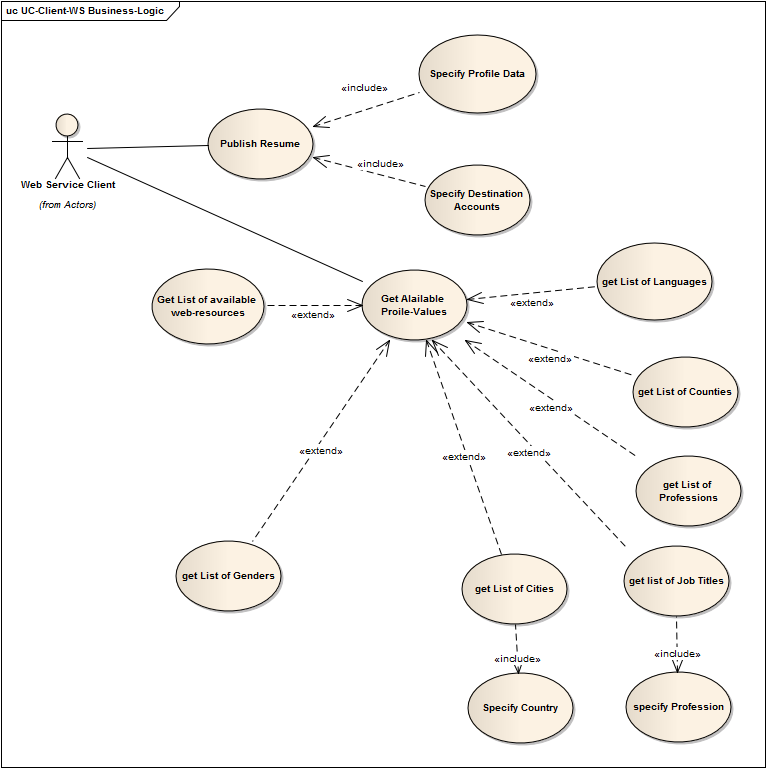


Рисунок 2.4.1.1.1 Диаграмма вариантов использования

* + - 1. Требования к созданию профиля резюме

На начальном этапе внедрения системы, необходимо предоставить пользователю возможность публикации/синхронизации своего резюме с двумя самыми популярными сайтами рекрутмента: hh.ru и master.com.

Что касается создания резюме, то тут пользовательский интерфейс должен предоставлять все необходимые возможности для внесения информации, которая впоследствии будет преобразована в соответствии с форматом её представления на других рекрут-ресурсах, а именно:

* Личная информация: Ф.И.О, дата рождения, гражданство, пол.
* Контактная информация: адрес электронной почты, местоположение (страна и город проживания), контактный телефон, дополнительные средства связи (ICQ, Skype,Facebook , Vkontakte,Twitter, веб-сайт), предпочитаемый вид связи(почта, мобильный)).
* Информация об образовании:
* основное образование: уровень (среднее, среднее специальное, неоконченное высшее, бакалавр, магистр, специалист, кандидат наук, доктор наук), название учебного заведения, факультет/специальность, даты обучения, местоположение (страна, город);
* повышение квалификации: название, организация, полученная квалификация, период;
* электронные сертификаты: название, год получения, ссылка.
* Информация об опыте работы: название компании, местоположение, контактный сайт, сфера деятельности, должность, период работы, обязанности и результаты.
* Ключевые навыки: описание, период, где применялись.
* Языковые навыки: название языка, уровень владения.
* Интересы: описание.
* Требования: желаемая позиция (желаемая з/п, должность), профессиональные области, профессии, тип желаемой должности (полная занятость, частичная занятость, проектная/временная работа, стажировка),должностной статус (полный рабочий день, неполный рабочий день, гибкий график, удалённая работа), возможность переезда (возможен, невозможен ,желателен), отношение к командировкам, дата готовности выйти на работу, желаемое время в пути до работы.
  + 1. Компонент “Message Broker ”

Взаимодействие сервиса, к которому имеет доступ клиент, с messaging-платформой системы должно осуществляться посредством **Java Message Service**  (JMS).

JMS - это спецификация J2EE технологии, определяющая набор интерфейсов к системам, ориентированных на работу через сообщения (message-oriented application programming). Таким образом, программа, написанная с использованием JMS, будет корректно работать с любой системой сообщений, поддерживающей эту спецификацию (или имеющую соответствующие интерфейсы).

* + - 1. Архитектура и общие термины JMS

Архитектура JMS выглядит следующим образом (Рисунок 2.4.2.1.1):

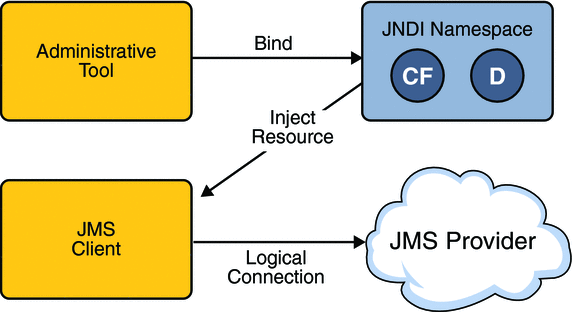


Рисунок 2.4.2.1.1 Архитектура JMS

* JMS client - Прикладные программы Java, использующие JMS;
* JMS-провайдером (JMS provider) - Система обработки сообщений, управляющая маршрутизацией и доставкой сообщений,
* Приложение JMS (JMS application) – это прикладная система, состоящая из нескольких JMS клиентов, и, как правило, одного JMS-провайдера. JMS-клиент, посылающий сообщение, называется поставщиком (producer). JMS-клиент, принимающий сообщение, называется потребителем (consumer). Один и тот же JMS клиент может быть одновременно и поставщиком и потребителем в разных актах взаимодействия;
* Сообщения (Messages) – это объекты, передающиеся и принимающиеся компонентами (клиентами JMS);
* Средства администрирования (Administrative tools) – средства управления ресурсами, использующимися клиентами.

JMS предоставляет два подхода к передаче сообщений:

* «издание-подписка» (publish an subscribe);
* «точка-точка» (point to point) .

Спецификация JMS называет эти два подхода зонами сообщений (Messaging Domains).

* + - 1. Модель взаимодействия точка-точка

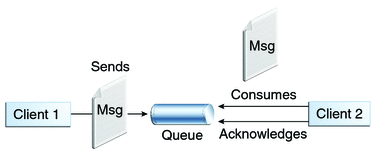
Модель передачи сообщений «точка-точка» предоставляет возможность клиентам JMS посылать и принимать сообщения (как синхронно, так и асинхронно) через виртуальные каналы, называемые очередями (queues). Модель передачи сообщений «точка-точка» основывается на методе опроса, при котором сообщения явно запрашиваются (считываются) клиентом из очереди. Несмотря на то, что чтение из очереди могут осуществлять несколько клиентов, каждое сообщение будет прочитано только единожды - провайдер JMS это гарантирует. 

Рисунок 2.4.2.2.1 Модель взаимодействия точка-точка

* + - 1. Модель взаимодействия издание-подписка

При использовании модели взаимодействия «издание-подписка» один клиент (поставщик) может посылать сообщения многим клиентам (потребителям) через виртуальный канал, называемый темой (topic). Потребители могут выбрать подписку (subscribe) на любую тему. Все сообщения, направляемые в тему, передаются всем потребителям данной темы. Каждый потребитель принимает копию каждого сообщения. Модель передачи сообщений издание-подписка, по существу, представляет собой модель, сервера, инициирующего соединение и «проталкивающего» информацию на клиента. В JMS эта концепция реализуется с помощью специальных «слушателей» (Listener), регистрируемых в системе. При возникновении нового события Listener, закрепленный за данной темой, возбуждается. Следует отметить, что при использовании модели «издание-подписка» клиенты JMS могут устанавливать долговременные подписки, позволяющие потребителям отсоединиться и позже снова подключиться и получать сообщения, поступившие во время отключения связи.



Рисунок 2.4.2.3.1 Модель взаимодействия издание-подписка

* + - 1. Требования к JMS провайдеру

Поскольку JMS является лишь оболочкой или интерфейсом, описывающим доступные для приложения методы, для работы приложения понадобится определенная реализация JMS, называемая провайдером JMS API.

В рамках данной курсовой работы целесообразно использовать провайдер, который относится к списку открытых решений:

* ActiveMQ (Apache)
* Fuse MQ (Red Hat)
* OpenJMS ( The OpenJMS Group)
* JBoss Messaging ( JBoss)
* JORAM (OW2)

Что касается функциональных и технических требований – важно чтобы провайдер предоставлял следующие возможности:

* Конфигурирование, администрирование очередей сообщений посредством консоли управления (желательно с графической оболочкой).
* Предоставление программного интерфейса к бизнес-функциям провайдера сторонним системам. Наличие данной характеристики позволит расширить стандартную функциональность посредством сторонних открытых решений.
* Предоставление специализированных инструментов для мониторинга характеристик производительности JMS-системы.
* Распределение нагрузки и отказоустойчивость.
* Поддержка системы сообщений и уведомлений об ошибках.
* Независимость от протокола связи.
  + 1. Rules Repository

Дынный компонент должен представлять собой СУБД платформу, для хранения и управления правил ,на основе которых система будет реализовывать интеграцию с веб-ресурсами.

Так как в рамках данной курсовой работы для реализации поставленной задачи необходимо пользоваться бесплатными решениями, то список подходящих технологий следующий:

* Apache Derby;
* DB2;
* HSQL DB;
* MySQL;
* Postgresql.
  + - 1. Диаграмма отношений сущностей

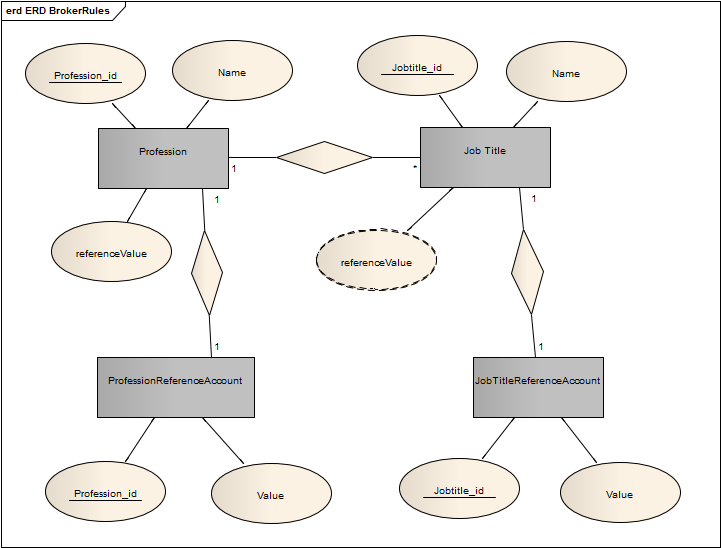


Рисунок 2.5.4.1.1 Диаграмма отношений сущностей

1. ****АППАРАТНЫЕ И ПРОГРАММНЫЕ СРЕДСТВА****

В качестве средств реализации были выбраны следующие программные продукты и технологии:

|  |  |
| --- | --- |
| **Технология** | **Описание** |
| **Java** | Язык программирования |
| **JUnit** | Фреймворк для тестирования |
| **Maven** | Инструмент для сборки проекта |
| **Git** | Система контроля версий |
| **Liquibase** | Технология версионирования для базы данных |
| **Apache Service Mix** | Интеграционная платформа |
| **Apache TomEE** | Сервер приложений |
| **Selenium WebDriver** | Фреймворк для симуляции действий пользователя в браузере. |
| **GitHub.com** | Хостинг – сервис для CVS Git |

Таблица 3.1 Средства реализации

* 1. Требования к составу и параметрам технических средств

|  |  |  |
| --- | --- | --- |
| **Требование** | **Минимальное значение** | **Рекомендуемое значение** |
| **Оперативная память** | 128 Мб | 1 Гб и более |
| **Свободное пространство на ЖД** | 500 Мб | 1 Гб и более |
| **Процессор** | 533 МГц | 1 Гб и более |

Таблица 3.1.1 Минимальные системные требования

* 1. Требования к программным средствам, используемым системой

Требования для разработки:

1. Операционная система с установленной JVM:

* Microsoft Windows;
* Linux;
* Mac OS.

1. Любая среда разработки под Java платформу.

|  |  |  |
| --- | --- | --- |
| **Требование** | **Минимальное значение** | **Рекомендуемое значение** |
| **Версия Java JDK , JRE** | 1.6. | 1.7.0 и выше |
| **Fuse IDE** | 2.4 | 3 и выше |
| **Apache Service Mix** | 4.5 | 4.5.2 и выше |
| **Apache TomEE** | 1.5 | 1.5.2 и выше |

Таблица 3.1.2 Минимальные программные требования

1. ****РЕАЛИЗАЦИЯ****
   1. Объектная модель профиля резюме

В соответствии с требованиями к формату представления запроса к системе была создана специальная XSD – схема (приложение 1).

Для работы с данным форматом в объектно-ориентированной форме, с помощью инструмента JAXB (Java XML Binding) были сгенерированы классы сущностей.

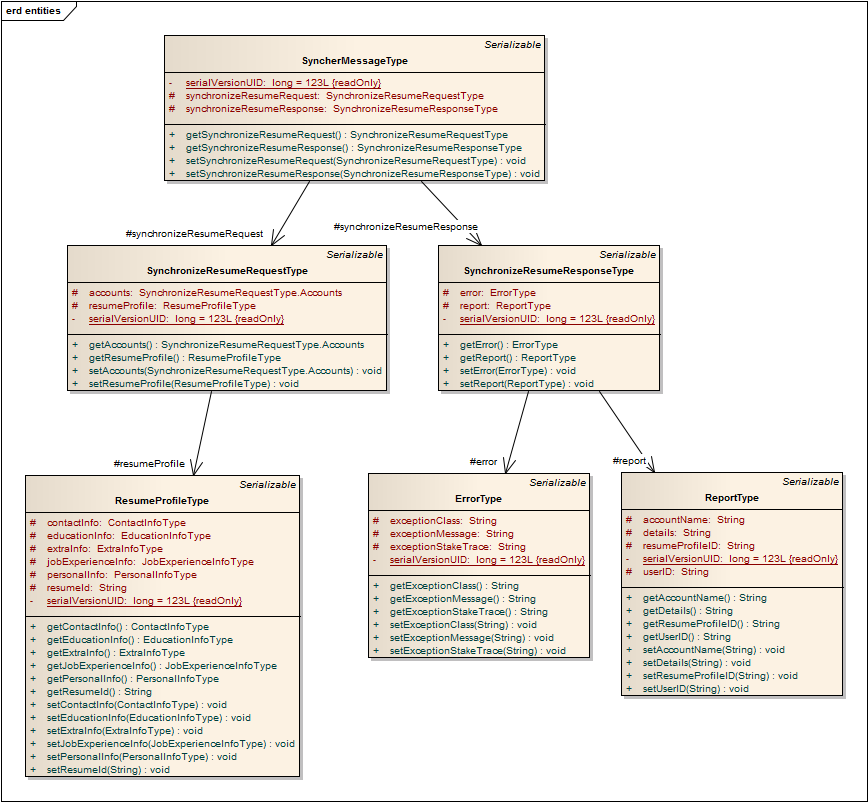


Рисунок 4.1.1 Диаграмма классов (часть 1)

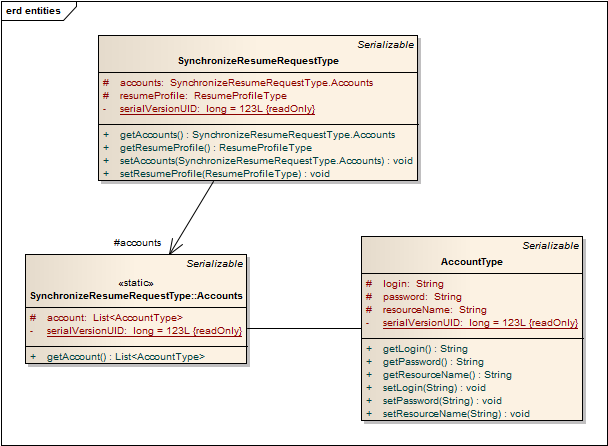


Рисунок 4.1.2 Диаграмма классов (часть 1, продолжение)

| **Класс** | **Описание/Назначение** |
| --- | --- |
| **SyncherMessageType** | Тип стандартного сообщения системы. Может включать в себя экземпляр запрос или ответа. |
| **SynchronizeResumeRequestType** | Тип запроса синхронизации. Содержит информацию об учётных записях которые необходимо синхронизировать и экземпляр профиля резюме. |

Таблица 4.1.1 Описание диаграммы классов

|  |  |
| --- | --- |
| Класс | Описание/Назначение |
| ResumeProfileType | Тип профиля резюме. |
| AccountType | Тип учётной записи. Содержит данные для авторизации в системе с которой необходимо осуществить синхронизацию. |
| SynchronizeResumeResponseType | Тип ответа на запрос о синхронизации. Может включать в себя экземпляр ошибки или отчёта. |
| ErrorType | Тип ошибки. Содержит информацию об имени класса возникшего исключения, сообщение об ошибке и прочие детали. |
| ReportType | Тип отчёта. Содержит следующую информацию: идентификатор успешно синхронизированного профиля резюме, идентификатор пользователя, совершившего запрос, информация об учётной записи сервиса с которым была совершена синхронизация. |

Таблица 4.1.2 Описание диаграммы классов (продолжение)

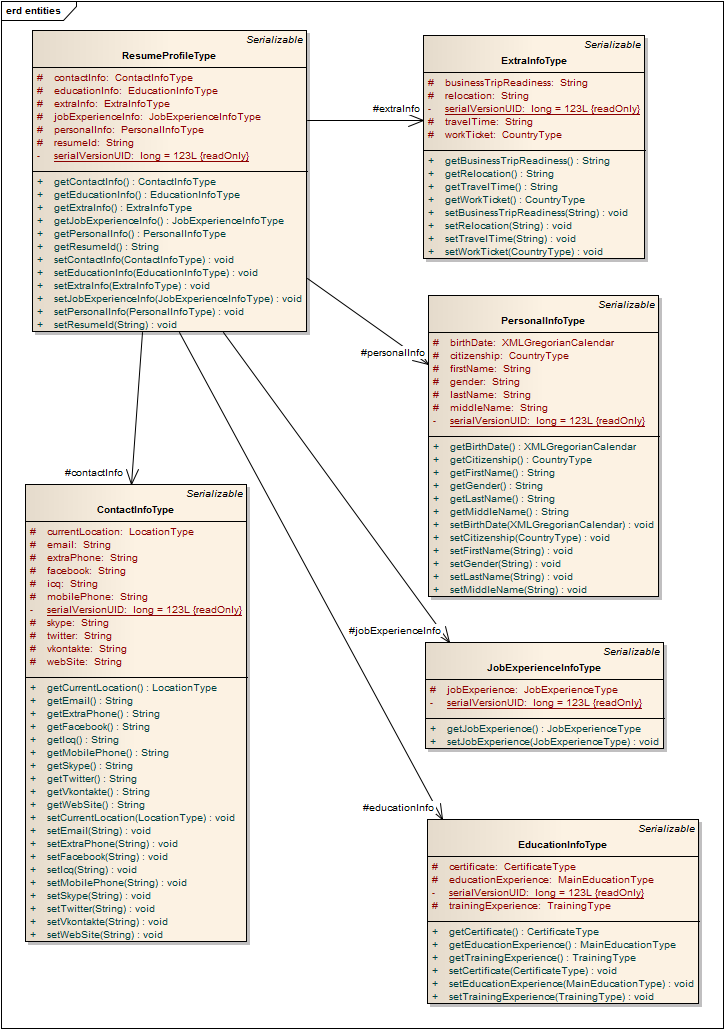


Рисунок 4.1.3 Диаграмма классов (часть 2)

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
| **ResumeProfileType** | Тип профиля резюме. |
| **ContactInfoType** | Тип контактной информации. |
| **PersonalInfoType** | Тип персональной информации. |
| **EducationInfoType** | Тип информации об образовании. |
| **JobExperienceInfoType** | Тип информации об опыте работы. |
| **ExtraInfoType** | Тип дополнительной информации. |

Таблица 4.1.3 Описание диаграммы классов (часть 2)

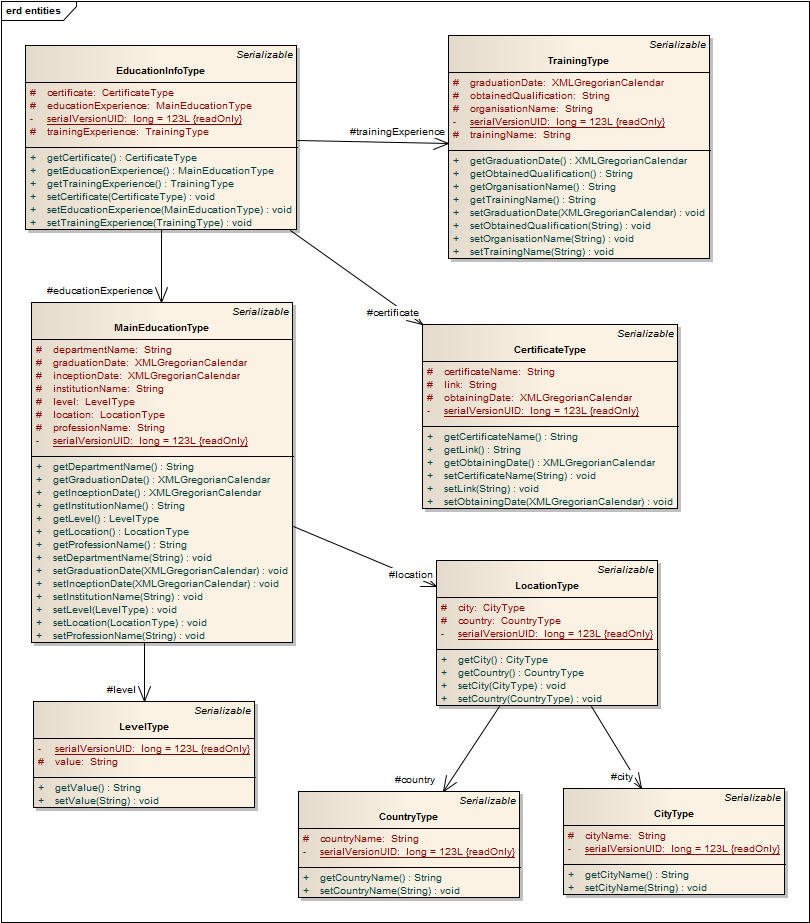


Рисунок 4.1.4 Диаграмма классов (часть 3)

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
| **EducationInfoType** | Тип информации об образовании. Может содержать в себе ссылки на экземпляры главного образования, тренингов, сертификаций. |
| **MainEducationType** | Тип главного образования. |
| **TrainingType** | Тип тренинга. |
| **CertificateType** | Тип сертификации. |
| **LevelType** | Тип уровня компетенции в той или иной области. |
| **LocationType** | Тип локации. Содержит в себе ссылки на экземпляры страны и города. |
| **CountryType** | Тип страны. |
| **CityType** | Тип города. |

Таблица 4.1.4 Описание диаграммы классов (часть 3)

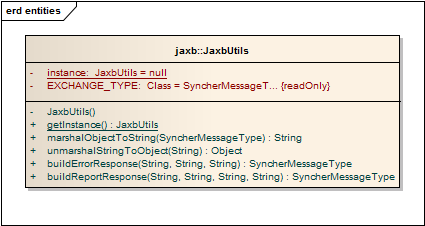


Рисунок 4.1.5 Диаграмма классов (часть 4)

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
| **JaxbUtils** | Класс - утилита. Позволяет преобразовать данные из машинного представления в высокоуровневое и наоборот. |

Таблица 4.1.5 Описание диаграммы классов (часть 4)

* 1. Компонент “WS Business-logic”

Для реализации данного веб-сервиса было решено использовать технологию Apache CXF.

CXF – реализация спецификации JAX-WS , что означает наличие возможности создания Web-сервис на основе существующего кода Java или же генерации код Java на основе WSDL-описания для использования или реализации сервиса. Очевидно, что такой подход делает разработку сервисов удобной и быстрой.

Важным преимуществом выбранной технологии перед другими аналогичными технологиями, является то , что Apache CXF хорошо подходит для решения задач, в которых необходимо SOAP-компонент встроить в уже существующую платформу.

* + 1. Диаграмма классов

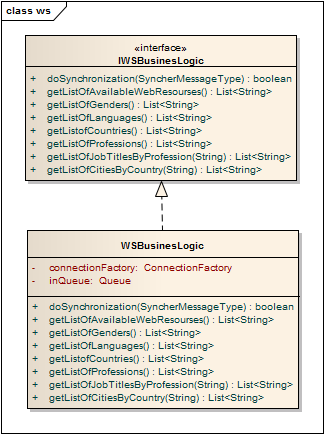


Рисунок 4.2.1.1 Диаграмма классов

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
| **IWSBusinesLogic** | Интерфейс для доступа к бизнес - логике системы |
| **WSBusinesLogic** | Класс-реализация контракта интерфейса «IWSBusinesLogic» |

Таблица 4.2.1.1 Описание классов

* 1. Компонент “Rules Repository”

Данный компонент был реализован в виде схемы базы данных.

В качестве платформы CУБД было решено использовать HSQLDB.

HSQLDB полностью написана на [Java](http://ru.wikipedia.org/wiki/Java). Может использоваться и как отдельный сервер с поддержкой сетевых соединений по [JDBC](http://ru.wikipedia.org/wiki/JDBC), и в виде библиотеки для использования непосредственно в коде программы.

Несмотря на то, что данный провайдер достаточно легковесный, он обладает всем необходимым функционалом для реализации поставленной задачи.

* + 1. Диаграмма Базы Данных

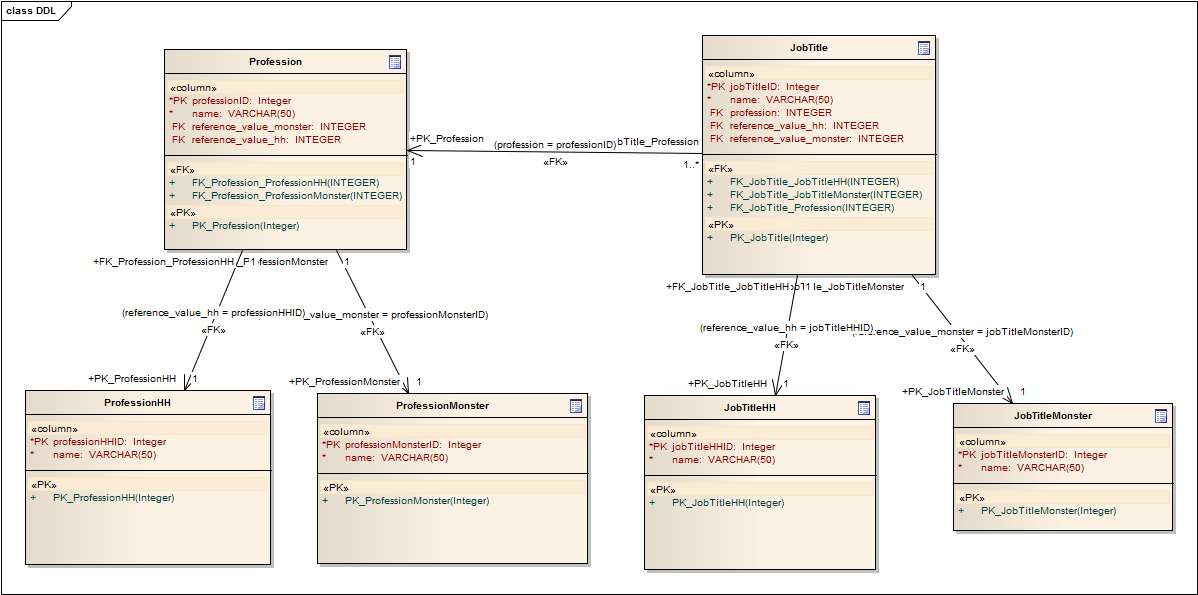
****

Рисунок 4.3.1.1 Диаграмма БД

|  |  |
| --- | --- |
| **Таблица** | **Описание/Назначение** |
| **Profession** | Хранение информации о профессии и ссылок на аналогичные значения в сторонних системах |
| **ProfessionHH** | Информация о профессии в сервисе HeadHunter.ru |
| **ProfessionMonster** | Информация о профессии в сервисе monster.com |
| **JobTitle** | Хранение информации о должности и ссылок на аналогичные значения в сторонних системах |
| **JobTitleHH** | Информация о должности в сервисе HeadHunter.ru |
| **JobTitleMonster** | Информация о должности в сервисе monster.com |

Таблица 4.3.1.1 Описание таблиц БД

* 1. Компонент “Message Broker”

Для реализации данного компонента была выбрана бесплатная, open-source интеграционная платформа Apache ServiceMix. Данная технология распространяется под лицензией apache Licence 2.0. Это означает, что данное решение может быть использовано как в академических проектах, так и в коммерческих (единственное требование – прикрепить к дистрибутиву разработанного продукта файл лицензии Apache).

Apache ServiceMix состоит из пяти компоентов :

* ApacheMQ – messaging платформа.
* Apache Camel – Фреймворк для конфигурации интеграции систем.
* Apache CXF – платформа для создания и развертывания веб-сервисов (SOAP, XML/HTTP, RESTful HTTP)
* Apache Karaf – легковесный OSGi – контейнер.
* Apache ODE - Фреймворк для обработки бизнес процессов в соответствии с [WS-BPEL](http://ode.apache.org/ws-bpel-20.html) стандартом.

Выбранная технология полностью отвечает требованиям, описанным в пункте (2.4.2.4)

* + 1. Конфигурация очередей сообщений и маршрутизации

Фреймворк apache Camel работает поверх JMS- провайдера ApacheMQ и позволяет настроить очереди сообщений и логику маршрутизации между ними с помощью скрипта в формате XML (Приложение 2). Описание логики маршрута представлено в диаграммах BPMN (Рисунок 4.4.1.1, 4.5.1.2).

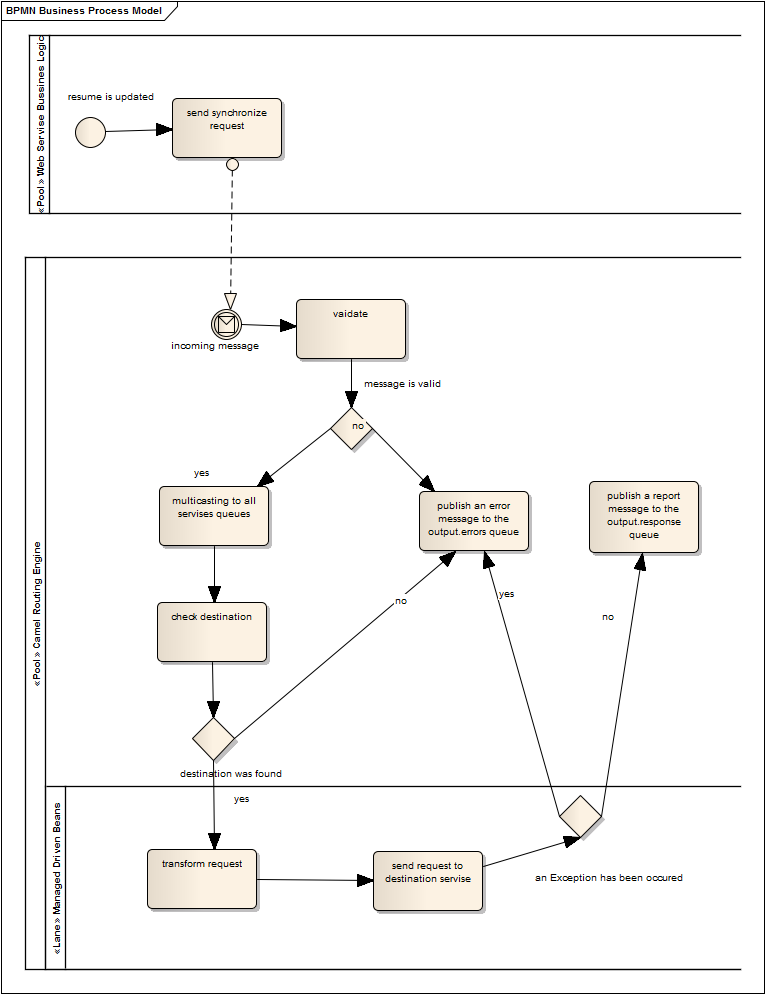


Рисунок 4.4.1.1 Диаграмма BPMN (часть 1)

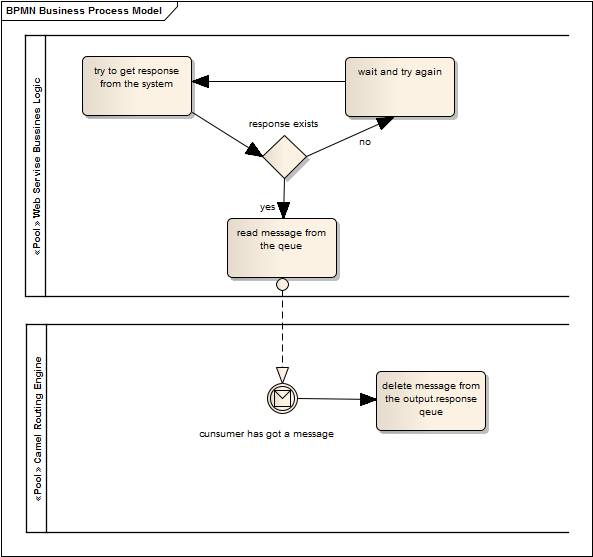


Рисунок 4.4.1.1 Диаграмма BPMN (часть 1)

* + 1. Проверка сообщений

Все сообщения принимаемые очередью input.request проверяются на соответствие с форматом, представленным схемой запроса (приложение 1).

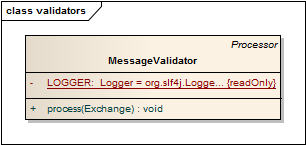


Рисунок 4.4.2.1 Диаграмма классов

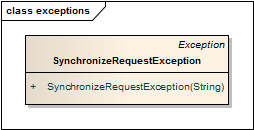
За проверку данного соответствия отвечает класс MessageValidator.

|  |  |
| --- | --- |
| **Таблица** | **Описание/Назначение** |
| **MessageValidator** | Отвечает за проверку формата входящих сообщений. |

Таблица 4.4.2.1 Описание диаграммы классов

* + 1. Обработка исключительных ситуаций и ошибок

Все сообщения принимаемые очередью input.request должны строго соответствовать формату схемы запроса (приложение 1). В противном случае система сгенерирует исключение SynchronizeRequestException , которое будет обработано специальным классом, для формирования отчёта об ошибке.



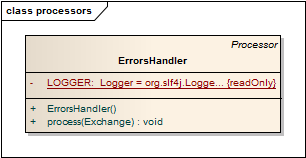


Рисунок 4.4.3.1 Диаграмма классов

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
| **SynchronizeRequestException** | Базовый класс – Exception. Используется для генерации исключений связанных с ошибками обработки запроса синхронизации. |
| **ErrorHandler** | Отвечает за обработку исключительных ситуаций, формирование отчётов о статусе запроса. |

Таблица 4.4.3.1 Описание диаграммы классов

* 1. **Компонент «Request Dispatcher»**

Данный компонент реализован в виде EJB-модуля. Для взаимодействия (прослушивания и отправки) с очередями сообщений используются Message Driven Beans. Данные компоненты реализуют интерфейс MessageListener и отвечают за запуск процессов дальнейшей обработки входящих сообщений. По окончанию обработки, отчёты или, в случае возникновения исключительных ситуаций, сообщения об ошибке отправляются в очередь «output.response».

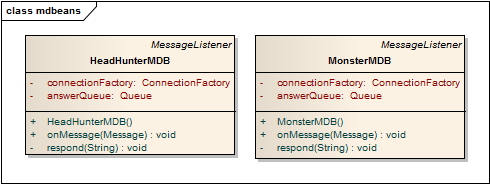


Рисунок 4.5.1 Диаграмма классов

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
| **HeadHunterMDB** | Отвечает за прослушивание очереди «hh». И отправки результатов обработки в очередь «output.response» |
| **MonsterMDB** | Отвечает за прослушивание очереди «monster». |

Таблица 4.5.1 Описание диаграммы классов

* + 1. Трансформация сообщений

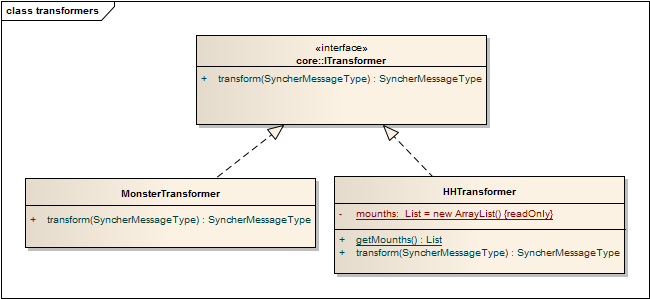


Рисунок 4.5.1.1 Диаграмма классов

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
| **ITransformer** | Базовый интерфейс |
| **HHTransformer** | Реализация логики трансформирования запроса для сервиса HeadHunter.ru |
| **MonsterTransformer** | Реализация логики трансформирования запроса для сервиса monster.com |

Таблица 4.5.1.1 Описание диаграммы классов

* + - 1. Cлой доступа к данным репозитирия правил транформации

На следующей диаграмме изображена архитекрура слоя DAO (Data access objects) (Рисунок 4.5.1.1.1):

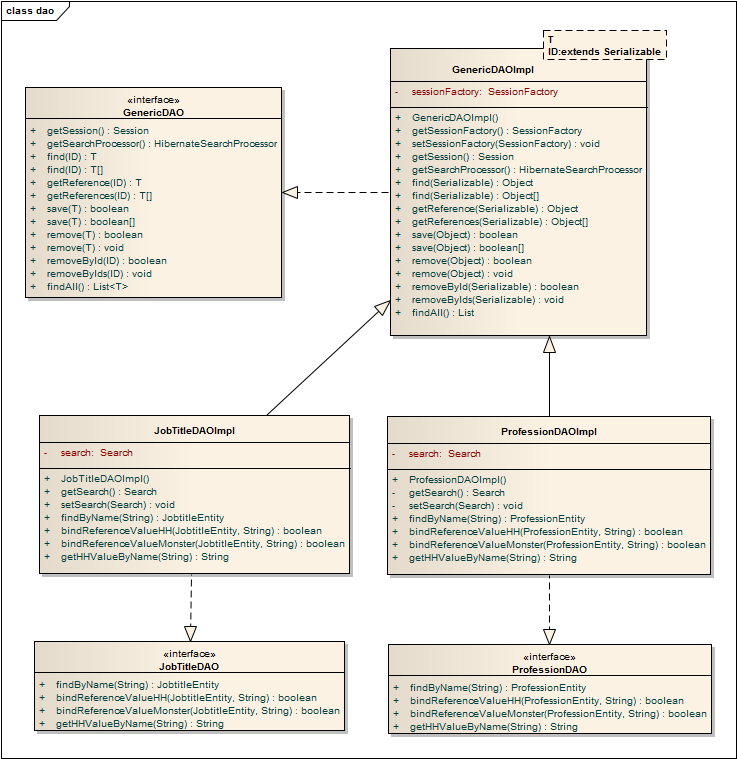


Рисунок 4.5.1.1.1 Диаграмма классов (часть 1)

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
| **GenericDAO** | Базовый интерфейс объекта доступа к данным (DAO – Data Access Object) |
| **GenericDAOImpl** | Реализация функциональности общей для всех DAO |
| **JobTitleDAO** | Базовый интерфейс |
| **JobTitleDAOImpl** | Реализация функций по работе с названиями должностей |
| **ProfessionDAO** | Базовый интерфейс |
| **ProfessionDAOImpl** | Реализация функций по работе с названиями профессий |

Таблица 4.5.1.1.1 Описание диаграммы классов (часть 1)

На следующей диаграмме представлены классы - сущности являющиеся объектным представлением таблиц схемы базы данных Rules Repository (пункт 4.3)

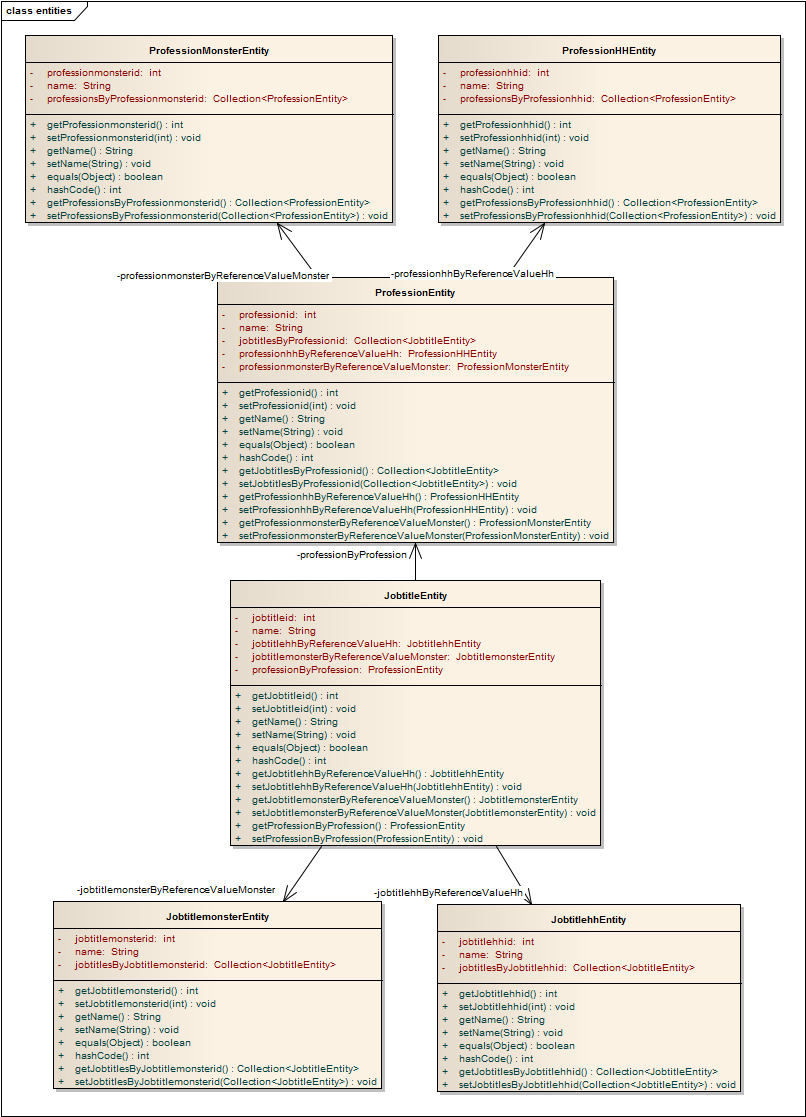


Рисунок 4.5.1.1.2 Диаграмма классов (часть 2)

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
| **ProfessionEntity** | Объектная модель таблицы Profession |
| **ProfessionHHEntity** | Объектная модель таблицы ProfessionHH |
| **ProfessionMonsterEntity** | Объектная модель таблицы ProfessionMonster |
| **JobtitleEntity** | Объектная модель таблицы JobTitle |
| **JobtitleHHEntity** | Объектная модель таблицы JobTitleHH |
| **JobtitleMonsterEntity** | Объектная модель таблицы JobTitleMonster |

Таблица 4.5.1.1.2 Описание диаграммы классов (часть 2)

* + 1. Передача преобразованных запросов к сторонним системам

Так как сторонние системы представляют из себя web-сайты, не предоставляющие никакого API для взаимодействия с ними, а именно для создания и обновления профиля резюме, строить интеграцию с ними возможно только через HTTP/HTTPS протоколы.

Исходя из этого ограничения, было решено использовать технологию Selenium WebDriver.

* + - 1. Selenium WebDriver

Selenium WebDriver представляет собой:

* спецификацию программного интерфейса для управления браузером;
* референсные реализации этого интерфейса для нескольких браузеров;
* набор клиентских библиотек для этого интерфейса на нескольких языках программирования (в частности для Java).

Кроме того, данный Фреймворк позволяет осуществить запись сценариев поведения пользователя, что делает разработку и поддержку логики интеграции намного удобнее и быстрее.

* + - 1. Диаграмма классов

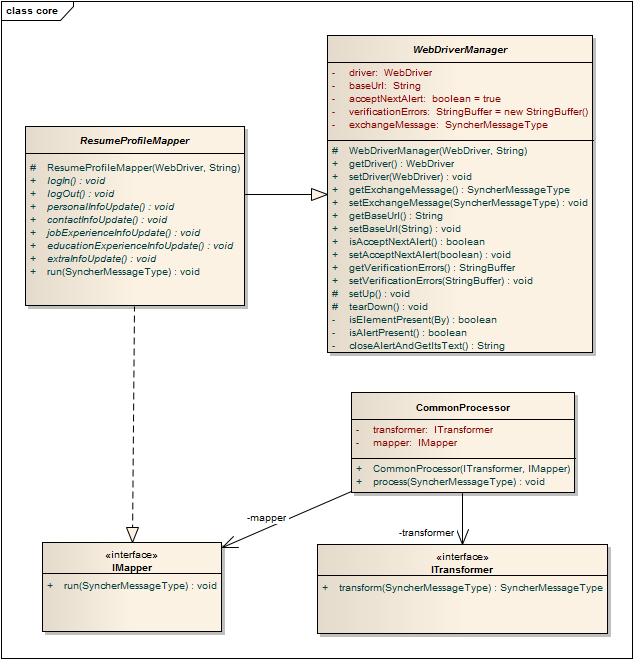
****

Рисунок 4.5.2.2.1 Диаграмма классов (часть 1)

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
| **IMapper** | Интерфейс определяющий сигнатуру метода запуска процесса передачи запросов к сторонним системам |
| **ResumeProfileMapper** | Абстрактный класс, предоставляющий базовую реализацию методов по обработке профиля резюме |
| **WebDriverManager** | Абстрактный класс, реализующий инициализацию Фреймворка Selenium WebDriver |
| **CommonProcessor** | Используется объектами-слушателями сообщений для запуска процесса обработки входящего запроса синхронизации. При инициализации экземпляра данного класса в конструктор необходимо передать ссылки на экземпляры классов - реализаций трансформации профиля резюме и его логики маппинга в стороннюю систему. |

Таблица 4.5.2.2.1 Описание диаграммы классов (часть 1)

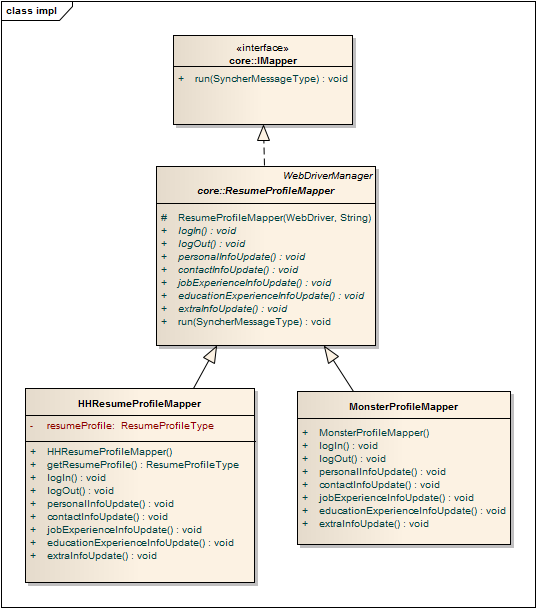
****

Рисунок 4.6.2.2.2 Диаграмма классов (часть 2)

|  |  |
| --- | --- |
| **Класс** | **Описание/Назначение** |
| **HHResumeMapper** | Реализация маппинга для сервиса HeadHunter.ru |
| **MonsterProfileMapper** | Реализация маппинга для сервиса monster.com |

Таблица 4.5.2.2.2 Описание диаграммы классов (часть 2)

* + 1. Диаграмма последовательностей

Следующая диаграмма иллюстрирует последовательность вызова методов обработки запроса, на примере реализации для сервиса HeadHunter.ru (Рисунок 4.6.3.1).

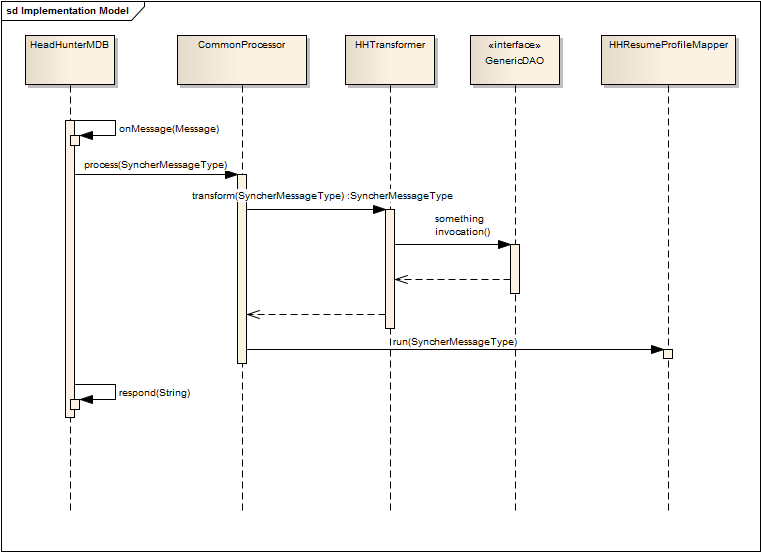
****

Рисунок 4.6.3.1 Диаграмма последовательностей

# ****ЗАКЛЮЧЕНИЕ****

**В результате проделанной работы была разработана система с гибко расширяемой архитектурой и предоставляющая клиенту:**

1. **Программный интерфейс для взаимодействия с системой.**
2. **Правила формирования запроса к системе, включая формат представления дынных экземпляра профиля-резюме.**
3. **Возможность управлять пользовательским профилем (публикация/обновление резюме на указанных веб-ресурсах).**
4. **Надежный механизм хранения и синхронизации информации между системой и веб-ресурсами, поддерживающий асинхронную модель передачи данных.**

Дальнейшие перспективы проекта:

* Расширение интеграции с job.ru, superjob.ru, rabota.ru, dice.com.
* Реализация web-front end – клиента.
* Расширение общей функциональности системы.

# ****СПИСОК ЛИТЕРАТУРЫ****

1. Henryk Konsek.Instant Apache ServiceMix How-to, 2013.– 66 c.
2. Bruce Snyder, Dejan Bosanac, Rob Davies. ActiveMQ in Action, 2011. - 375 c.
3. Claus Ibsen , Jonathan Anstey. Camel in Action, 2011. - 552 c.
4. Naveen Balani, Rajeev Hathi. Apache CXF Web Service Development, 2009. - 336 c.
5. Debu Panda, Reza Rahman , Ryan Cuprak , Michael Remijan. EJB 3 in Action, 2013. - 800 c.
6. Apache Camel. Official documentation. - (http://camel.apache.org/documentation.html)
7. Apache ServiceMix. Official documentation. –

(http://servicemix.apache.org/documentation.html)

1. Apache TomEE. Official documentation – (http://tomee.apache.org/documentation.html)
2. Selenium. Official documentation. – (http://docs.seleniumhq.org/docs/)
3. Apache Maven. Official documentation. – (http://maven.apache.org/guides/)
4. Liquibase. Official documentation. – (http://www.liquibase.org/documentation/index.html)
5. JMS. Java Oracle. Official documentation. – (http://docs.oracle.com/javaee/1.3/jms/tutorial/)

# ****ПРИЛОЖЕНИЕ 1. Request-schema.xsd****

<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="com/ipetrushin/syncher/request/jaxb/entities/"

xmlns="com/ipetrushin/syncher/request/jaxb/entities/">

<xs:element id="root" name="syncherMessage" type="syncherMessageType"/>

<xs:complexType name="syncherMessageType">

<xs:choice minOccurs="1" maxOccurs="1">

<xs:element name="synchronizeResumeRequest" type="synchronizeResumeRequestType"/>

<xs:element name="synchronizeResumeResponse" type="synchronizeResumeResponseType"/>

</xs:choice>

</xs:complexType>

<xs:complexType name="synchronizeResumeRequestType">

<xs:sequence>

<xs:element name="accounts">

<xs:complexType>

<xs:sequence>

<xs:element name="account" type="accountType" minOccurs="1" maxOccurs="5"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="resumeProfile" type="resumeProfileType" minOccurs="1" maxOccurs="1"/>

</xs:sequence>

</xs:complexType>

<xs:complexType name="synchronizeResumeResponseType">

<xs:choice minOccurs="1" maxOccurs="1">

<xs:element name="error" type="errorType"/>

<xs:element name="report" type="reportType"/>

</xs:choice>

</xs:complexType>

<xs:complexType name="errorType">

<xs:all>

<xs:element name="exceptionClass" type="xs:string"/>

<xs:element name="exceptionMessage" type="xs:string"/>

<xs:element name="exceptionStakeTrace" type="xs:string"/>

</xs:all>

</xs:complexType>

<xs:complexType name="reportType">

<xs:all>

<xs:element name="resumeProfileID" type="xs:string"/>

<xs:element name="accountName" type="xs:string"/>

<xs:element name="userID" type="xs:string"/>

<xs:element name="details" type="xs:string"/>

</xs:all>

</xs:complexType>

<xs:complexType name="accountType">

<xs:attributeGroup ref="accountAttributes"/>

</xs:complexType>

<xs:attributeGroup name="accountAttributes">

<xs:attribute name="resourceName" type="xs:string" use="required"/>

<xs:attribute name="login" type="xs:string" use="required"/>

<xs:attribute name="password" type="xs:string" use="required"/>

</xs:attributeGroup>

<xs:complexType name="resumeProfileType">

<xs:all>

<xs:element name="resumeId" type="xs:string"/>

<xs:element name="personalInfo" type="personalInfoType"/>

<xs:element name="contactInfo" type="contactInfoType"/>

<xs:element name="educationInfo" type="educationInfoType"/>

<xs:element name="jobExperienceInfo" type="jobExperienceInfoType"/>

<xs:element name="extraInfo" type="extraInfoType"/>

</xs:all>

</xs:complexType>

<xs:complexType name="personalInfoType">

<xs:all>

<xs:element name="firstName" type="xs:string"/>

<xs:element name="lastName" type="xs:string"/>

<xs:element name="middleName" type="xs:string"/>

<xs:element name="birthDate" type="xs:date"/>

<xs:element name="citizenship" type="countryType"/>

<xs:element name="gender" type="xs:string"/>

</xs:all>

</xs:complexType>

<xs:complexType name="contactInfoType">

<xs:all>

<xs:element name="email" type="xs:string"/>

<xs:element name="currentLocation" type="locationType"/>

<xs:element name="mobilePhone" type="xs:string"/>

<xs:element name="extraPhone" type="xs:string"/>

<xs:element name="webSite" type="xs:string"/>

<xs:element name="facebook" type="xs:string"/>

<xs:element name="vkontakte" type="xs:string"/>

<xs:element name="skype" type="xs:string"/>

<xs:element name="icq" type="xs:string"/>

<xs:element name="twitter" type="xs:string"/>

</xs:all>

</xs:complexType>

<xs:complexType name="educationInfoType">

<xs:sequence>

<xs:element name="educationExperience" type="mainEducationType" minOccurs="0"/>

<xs:element name="trainingExperience" type="trainingType" minOccurs="0"/>

<xs:element name="certificate" type="certificateType" minOccurs="0"/>

</xs:sequence>

</xs:complexType>

<xs:complexType name="mainEducationType">

<xs:all>

<xs:element name="level" type="levelType"/>

<xs:element name="institutionName" type="xs:string"/>

<xs:element name="departmentName" type="xs:string"/>

<xs:element name="professionName" type="xs:string"/>

<xs:element name="inceptionDate" type="xs:date"/>

<xs:element name="graduationDate" type="xs:date"/>

<xs:element name="location" type="locationType"/>

</xs:all>

</xs:complexType>

<xs:complexType name="trainingType">

<xs:all>

<xs:element name="trainingName" type="xs:string"/>

<xs:element name="organisationName" type="xs:string"/>

<xs:element name="obtainedQualification" type="xs:string"/>

<xs:element name="graduationDate" type="xs:date"/>

</xs:all>

</xs:complexType>

<xs:complexType name="certificateType">

<xs:all>

<xs:element name="certificateName" type="xs:string"/>

<xs:element name="obtainingDate" type="xs:date"/>

<xs:element name="link" type="xs:string"/>

</xs:all>

</xs:complexType>

<xs:complexType name="countryType">

<xs:all>

<xs:element name="countryName" type="xs:string"/>

</xs:all>

</xs:complexType>

<xs:complexType name="cityType">

<xs:all>

<xs:element name="cityName" type="xs:string"/>

</xs:all>

</xs:complexType>

<xs:complexType name="locationType">

<xs:all>

<xs:element name="country" type="countryType"/>

<xs:element name="city" type="cityType"/>

</xs:all>

</xs:complexType>

<xs:complexType name="levelType">

<xs:attribute name="value" type="xs:string"/>

</xs:complexType>

<xs:complexType name="jobExperienceInfoType">

<xs:sequence>

<xs:element name="jobExperience" type="jobExperienceType" minOccurs="0"/>

</xs:sequence>

</xs:complexType>

<xs:complexType name="jobExperienceType">

<xs:all>

<xs:element name="companyName" type="xs:string"/>

<xs:element name="fromDate" type="xs:date"/>

<xs:element name="toDate" type="xs:date"/>

<xs:element name="industryArea" type="xs:string"/>

<xs:element name="position" type="xs:string"/>

<xs:element name="companyLocation" type="locationType"/>

<xs:element name="responsibilities" type="xs:string"/>

<xs:element name="link" type="xs:string"/>

</xs:all>

</xs:complexType>

<xs:complexType name="extraInfoType">

<xs:all>

<xs:element name="relocation" type="xs:string"/>

<xs:element name="businessTripReadiness" type="xs:string"/>

<xs:element name="workTicket" type="countryType"/>

<xs:element name="travelTime" type="xs:string"/>

</xs:all>

</xs:complexType>

</xs:schema>

# ****ПРИЛОЖЕНИЕ 2.****

# ****Пример запроса синхронизации профиля резюме****

<?xml version="1.0" encoding="Unicode" standalone="yes"?>

<ns2:syncherMessageType xmlns:ns2="com/ipetrushin/syncher/request/jaxb/entities/">

<synchronizeResumeRequest>

<accounts>

<account resourceName="hh.ru" login="userLogin" password="userPassword"/>

<account resourceName="monster.russia.ru" login="userLogin " password=" userPassword "/>

</accounts>

<resumeProfile>

<resumeId></resumeId>

<personalInfo>

<firstName>Иван</firstName>

<lastName>Петрушин</lastName>

<middleName>Александрович</middleName>

<birthDate>12.09.1991</birthDate>

<citizenship>

<countryName> Россия </countryName>

</citizenship>

<gender>мужской</gender>

</personalInfo>

<contactInfo>

<email>string</email>

<currentLocation>

<country>

<countryName>Россия</countryName>

</country>

<city>

<cityName>Воронеж</cityName>

</city>

</currentLocation>

<mobilePhone>89808000000</mobilePhone>

<extraPhone>89808000001</extraPhone>

<webSite></webSite>

<facebook> </facebook>

<vkontakte> </vkontakte>

<skype> </skype>

<icq> </icq>

<twitter> </twitter>

</contactInfo>

</resumeProfile>

</synchronizeResumeRequest>

</ns2:syncherMessageType>

# ****ПРИЛОЖЕНИЕ 3.****

# ****Сценарий интеграции Camel.****

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.0.xsd

http://camel.apache.org/schema/spring

http://camel.apache.org/schema/spring/camel-spring.xsd">

<!--ENVIRONMENT SETTINGS -->

<bean id="jmsConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">

<property name="brokerURL">

<!-- value>tcp://localhost:61616</value -->

<value>vm://localhost</value>

</property>

</bean>

<bean id="pooledJmsConnectionFactory" class="org.apache.activemq.pool.PooledConnectionFactory"

destroy-method="stop">

<property name="connectionFactory" ref="jmsConnectionFactory"/>

</bean>

<bean id="jmsConfig"

class="org.apache.camel.component.jms.JmsConfiguration">

<property name="connectionFactory" ref="pooledJmsConnectionFactory"/>

<property name="concurrentConsumers" value="10"/>

</bean>

<bean id="activemq"

class="org.apache.activemq.camel.component.ActiveMQComponent">

<property name="configuration" ref="jmsConfig"/>

</bean>

<!--INTEGRATION -->

<bean id="MessageValidator" class="com.ipetrushin.syncher.integration.camel.beans.validators.MessageValidator"/>

<bean id="ErrorsHandler" class="com.ipetrushin.syncher.integration.camel.beans.processors.ErrorsHandler"/>

<!— Unhandled EXEPTIONS -->

<bean id="myDeadLetterErrorHandler" class="org.apache.camel.builder.DeadLetterChannelBuilder">

<property name="deadLetterUri" value="activemq:deadLetters"/>

<property name="redeliveryPolicy">

<bean class="org.apache.camel.processor.RedeliveryPolicy">

<property name="maximumRedeliveries" value="1"/>

<property name="useExponentialBackOff" value="true"/>

</bean>

</property>

</bean>

<!— Routes -->

<camelContext xmlns="http://camel.apache.org/schema/spring" errorHandlerRef="myDeadLetterErrorHandler">

<onException>

<!— Handling of EXEPTIONS --> <exception>com.ipetrushin.syncher.request.exceptions.SynchronizeRequestException</exception>

<exception>java.lang.Exception</exception>

<redeliveryPolicy maximumRedeliveries="1"/>

<!-- mark this as handled -->

<handled>

<constant>true</constant>

</handled>

<!-- let our order service handle this exception-->

<process ref="ErrorsHandler"/>

<to uri="activemq:output.response"/>

</onException>

<route id="integrationRoute">

<from uri="activemq:input.request"/>

<process ref="MessageValidator"/>

<choice>

<when>

<xpath>//\*[local-name() = 'synchronizeResumeRequest']</xpath>

<multicast>

<choice>

<when>

<xpath>//\*[local-name() = 'account'][@resourceName = 'monster.com']</xpath>

<to uri="activemq:monster"/>

</when>

</choice>

<choice>

<when>

<xpath>//\*[local-name() = 'account'][@resourceName = 'hh.ru']</xpath>

<to uri="activemq:hh"/>

<!-- <to uri="file:syncher/jms/output/requests/hh/"/> -->

</when>

</choice>

</multicast>

</when>

<when>

<xpath>//ns2:requestDeleteResume</xpath>

<log message="request-delete-resume"/>

</when>

</choice>

</route>

<route id="dlcRoute">

<from uri="direct:deadletter"/>

<log message="File ${file:name} was moved to the dead letter channel"/>

<to uri="file:syncher/errors/deadletter"/>

</route>

<route id="outputRoute">

<from uri="activemq:output.response"/>

<choice>

<when>

<xpath>//\*[local-name() = 'synchronizeResumeResponse']</xpath>

<choice>

<when>

<xpath>//\*[local-name() = 'error']</xpath>

<to uri="activemq:output.response.errors"/>

</when>

<when>

<xpath>//\*[local-name() = 'report']</xpath>

<to uri="activemq:output.response.reports"/>

</when>

</choice>

</when>

</choice>

</route>

</camelContext>

</beans>

# ****ПРИЛОЖЕНИЕ 4.****

# Пример слушателя сообщений для сервиса HeadHunter.ru****.****

package com.ipetrushin.syncher.ejb.dispatcher.jmsprovider.mdbeans;

import com.ipetrushin.syncher.ejb.dispatcher.core.CommonProcessor;

import com.ipetrushin.syncher.ejb.dispatcher.mappers.impl.HHResumeProfileMapper;

import com.ipetrushin.syncher.ejb.dispatcher.transformers.HHTransformer;

import com.ipetrushin.syncher.request.jaxb.JaxbUtils;

import com.ipetrushin.syncher.request.jaxb.entities.SyncherMessageType;

import javax.annotation.Resource;

import javax.ejb.ActivationConfigProperty;

import javax.ejb.MessageDriven;

import javax.jms.\*;

@MessageDriven(

name = "HeadHunterMDB",

activationConfig = {

@ActivationConfigProperty(propertyName = "destinationType",

propertyValue = "javax.jms.Queue"),

@ActivationConfigProperty(propertyName = "destination",

propertyValue = "hh") // Ext. JNDI Name

}

)

public class HeadHunterMDB implements MessageListener {

@Resource

private ConnectionFactory connectionFactory;

@Resource(name = "output.response")

private Queue answerQueue;

public HeadHunterMDB() {

}

@Override

public void onMessage(Message message) {

System.out.println("\*\*\* received message: " + message);

HHResumeProfileMapper profileMapper = new HHResumeProfileMapper();

SyncherMessageType response = null;

String responseMessage = null;

try {

TextMessage textMessage = (TextMessage) message;

SyncherMessageType exchangeMessage = (SyncherMessageType) JaxbUtils.getInstance().unmarshalStringToObject(textMessage.getText());

CommonProcessor processor = new CommonProcessor(new HHTransformer(), new HHResumeProfileMapper());

processor.process(exchangeMessage);

response= JaxbUtils.getInstance().buildReportResponse();

} catch (Exception e) {

response= JaxbUtils.getInstance().buildErrorResponse(e.getClass().toString(),

e.getMessage(),

e.getStackTrace().toString());

}finally {

try{

responseMessage = JaxbUtils.getInstance().marshalObjectToString(response);

respond(responseMessage);

}catch (Exception ex){

ex.printStackTrace();

}

}

}

private void respond(String text) throws JMSException {

Connection connection = null;

Session session = null;

try {

connection = connectionFactory.createConnection();

connection.start();

// Create a Session

session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

// Create a MessageProducer from the Session to the Topic or Queue

MessageProducer producer = session.createProducer(answerQueue);

producer.setDeliveryMode(DeliveryMode.PERSISTENT);

// Create a message

TextMessage message = session.createTextMessage(text);

// Tell the producer to send the message

producer.send(message);

} finally {

// Clean up

if (session != null) session.close();

if (connection != null) connection.close();

}

}

}