1 Оглавление

РЕФЕРАТ ---- 8
ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, СИМВОЛОВ И СПЕЦИАЛЬНЫХ ТЕРМИНОВ С ИХ ОПРЕДЕЛЕНИЕМ -9
ВВЕДЕНИЕ ---- 16
2 НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ ---- 18
2.1 Основные архитектуры распределенных систем. ---- 18
2.2 Способы взаимодействия в распределенных системах ---- 18
2.3 Основные механизмы в распределенных системах ---- 20
2.3.1 Передача файлов ---- 21
2.3.2 Общая база данных ---- 22
2.3.3 Удаленный вызов процедур. ---- 23
2.3.4 Асинхронный обмен сообщениями ---- 23
2.4 Описание предметной области. ---- 24
2.5 Построение различных архитектур ИС применительно к сфере банковской автоматизации. ---- 27
2.6 Описание задачи интеграции 2-х систем. ---- 32
2.7 Выводы по научно-исследовательскому этапу. ---- 32
3 КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ ---- 33
3.1 Цели разработки системы ---- 33
3.2 Условия. ---- 33
3.3 Компоненты системы интеграции. ---- 33
3.4 Физические компоненты развертывания системы «ЭПОС» ---- 35
3.5 Архитектура подсистемы взаимодействия с ПС «Рапида» ---- 38
3.6 Рефакторинг подсистемы обработки платежей ---- 40
3.7 Техническое задание на систему. ---- 40
3.7.1 Полное наименование работы и ее условное обозначение ---- 40
3.7.2 Основания для разработки ---- 41
3.7.3 Назначение разработки ---- 41
3.7.4 Цели и эффекты от интеграции ---- 41
3.7.5 Характеристика объектов автоматизации ---- 42
3.7.6 Общее описание автоматизируемых сервисов ---- 42
3.7.7 Требования к подсистеме ---- 43
3.7.8 Порядок контроля и приемки подсистемы ---- 44
3.7.9 Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу подсистемы в действие. ---- 45
4 ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ ---- 47

4.1 Типы запросов к платежной системе «Рапида» ----- 47
4.2 Шифрование запроса в ПС «Рапида». ----- 50
4.3 Реализация архитектуры подсистемы ----- 51
5 РАБОЧЕЕ ПРОЕКТИРОВАНИЕ ----- 53
5.1 Реализация запросов к ПС «Рапида» ----- 53
5.1.1 Реализация запроса на загрузку каталога услуг. ----- 54
5.1.2 Реализация запроса на проверку реквизитов платежа. ----- 54
5.1.3 Реализация запроса на исполнение платежа. ----- 55
5.1.4 Реализация запроса на отмену платежа. ----- 56
5.2 Реализация защищенного взаимодействия с ПС «Рапида» ----- 56
5.3 Реализация верхнего уровня ----- 57
5.3.1 Реализация логики загрузки каталога. ----- 57
5.3.2 Реализация логики обработки платежа. ----- 60
6 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ ----- 66
6.1 Сравнение альтернативных решений ----- 66
6.2 Основной критерий сравнения ----- 69
6.3 Тестирование подходов и сравнение производительности ----- 70
6.4 Выводы по исследовательскому этапу ----- 73
ЗАКЛЮЧЕНИЕ ----- 74
СПИСОК ЛИТЕРАТУРЫ ----- 75
ПРИЛОЖЕНИЕ 1. Листинг класса-клиента к платежной системе «Рапида». ----- 76
ПРИЛОЖЕНИЕ 2. Листинг класса веб-сервиса, осуществляющего загрузку каталога услуг ----- 93
ПРИЛОЖЕНИЕ 3. Листинг класса, осуществляющего обработку платежей ----- 96

РЕФЕРАТ

Отчет 121 с., 12 рис., 14 табл., 12 источников, 2 прил.

БАНКОВСКАЯ АВТОМАТИЗАЦИЯ, SOA, СИСТЕМЫ ИНТЕГРАЦИИ, СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА, РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ, ДИСТАНЦИОННОЕ БАНКОВСКОЕ ОБСЛУЖИВАНИЕ, СЕРВИС-ШИНА ПРЕДПРИЯТИЯ.

Объектом разработки модуль взаимодействия системы дистанционного банковского обслуживания ДБО «Частный клиент» и платежной системы «Рапида». Модуль разработан на базе интеграционной информационной системы «ЭПОС» с использованием концепции SOA и распределенных информационных систем

Цель работы – исследование архитектур распределенных систем применительно к области банковской автоматизации и разработка модуля для интеграции системы дистанционного банковского обслуживания и платежного сервиса.

При создании модуля проведены исследования, целью которых являлся анализ архитектур существующих информационных систем и оценка оптимальности выбранного решения применительно к области банковской автоматизации.

В результате работы произведен анализ существующих архитектур, выбор оптимальной и разработка модуля взаимодействия с платежной системой.

Результаты данной дипломной работы успешно используются при взаимодействии и дальнейшем развитии системы. В настоящее время разработанный модуль проходит бета-тестирование.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, СИМВОЛОВ И СПЕЦИАЛЬНЫХ ТЕРМИНОВ С ИХ ОПРЕДЕЛЕНИЕМ

ЧТЗ	-	Частное техническое задание
ЭД	-	Электронный документ
ЭПС	-	Электронная платежная система
ЭЦП	-	Электронная цифровая подпись
EBPP	-	Процедура электронного выставления и оплаты счетов (electronic bill presentment and payment)
Java EE	-	Набор спецификаций и соответствующей документации для языка Java, описывающей архитектуру серверной платформы для задач средних и крупных предприятий (java platform, enterprise edition)
JMS	-	Java - сервис сообщений
MOM	-	Программное обеспечение среднего слоя, ориентированное на обработку сообщений (message-oriented middleware)
PKI	-	Инфраструктура открытых ключей (public key infrastructure)
SSL	-	Протокол безопасных соединений (secure socket layer)
SOA	-	Архитектура, ориентированная на сервисы (service-oriented architecture)
TCP/IP	-	Стек протоколов, описанный в стандартах и рекомендациях Internet Engineering Task Force (transmission control protocol/internet protocol)
WSDL	-	Язык определения веб – сервисов (web service definition language)
Автоматизированная система		Система, состоящая из взаимосвязанной совокупности подразделений организации (или коллектива специалистов) и комплекса средств автоматизации деятельности, реализующая автоматизированные функции по отдельным видам деятельности-исследованию, управлению,

		испытаниям и др., или по их сочетаниям [2]
Архитектура, ориентированная на сервисы	-	Архитектура, в которой система строится из набора слабосвязанных гетерогенных компонентов (сервисов) (service-oriented architecture)
Банковская система	-	Все финансовые учреждения, которые, в частности, принимают депозиты, предоставляют кредиты и/или предлагают платежные услуги непосредственно пользователям в качестве одной из своих ключевых бизнес-функций. Включает также центральный банк (banking system) [3]
Владелец сертификата ключа подписи	-	Физическое лицо, на имя которого удостоверяющим центром выдан сертификат ключа подписи и которое владеет соответствующим закрытым ключом электронной цифровой подписи, позволяющим с помощью средств электронной цифровой подписи создавать свою электронную цифровую подпись в электронных документах (подписывать электронные документы) [4]
Доступ к информации	-	Возможность получения информации и ее использования [4]
Закрытый ключ электронной цифровой подписи	-	Уникальная последовательность символов, известная владельцу сертификата ключа подписи и предназначенная для создания в электронных документах электронной цифровой подписи с использованием средств электронной цифровой подписи [4]
Информационная система	-	Совокупность содержащейся в базах данных информации и обеспечивающих ее обработку информационных технологий и технических средств [1]
Конечный пользователь	-	Клиент финансового учреждения, которому финансовое учреждение предоставляет платежные инструменты и услуги с целью способствовать завершению его коммерческих или финансовых операций (end user) [2]
Конфиденциальность информации	-	Обязательное для выполнения лицом, получившим доступ к определенной информации, требование не передавать такую информацию третьим лицам без согласия ее обладателя [4]

Оператор платежной системы	-	Организация, определяющая правила платежной системы, а также выполняющая иные обязанности, предусмотренные Федеральным законом [4]
Открытый ключ электронной цифровой подписи	-	Уникальная последовательность символов, соответствующая закрытому ключу электронной цифровой подписи, доступная любому пользователю информационной системы и предназначенная для подтверждения с использованием средств электронной цифровой подписи подлинности электронной цифровой подписи в электронном документе
Платежная инфраструктура	-	Инфраструктура платежной системы, представляющая собой совокупность операторов услуг платежной инфраструктуры [4]. Совокупность сетевого оборудования, технологий и процедур для осуществления доступа и операций с платежными инструментами, а также для обработки, клиринга и расчета по соответствующим платежам (payment infrastructure) [4]
Платежная система	-	Совокупность организаций, взаимодействующих по правилам платежной системы в целях осуществления перевода денежных средств, включающая оператора платежной системы, участников платежной системы, из которых, как минимум, три организации являются операторами по переводу денежных средств и (или) участниками финансовых рынков (в случае переводов денежных средств в целях расчета по сделкам с ценными бумагами и (или) сделок, совершенных на организованном рынке), а также операторов услуг платежной инфраструктуры [4] Определенный ряд инструментов, банковских процедур и систем межбанковского перевода денежных средств (например, клиринговых и расчетных), который обеспечивает денежное обращение [4] Совокупность инструментов, банковских процедур и, как правило, систем межбанковского перевода денежных средств, которые обеспечивают денежное обращение [4]

Платежное распоряжение (инструкция)	-	Распоряжение или сообщение с требованием о переводе денежных средств (в форме денежного требования к какой-либо стороне) по распоряжению получателя денежных средств. Распоряжение может относиться либо к кредитовому, либо к дебетовому переводу. Называется также платежной инструкцией (payment order (instruction) [4]
Плательщик	-	Физическое лицо, осуществляющее внесение платежному агенту денежных средств в целях исполнения денежных обязательств физического лица перед поставщиком [4]
Платежный агент	-	Юридическое лицо или индивидуальный предприниматель, привлекаемые оператором по переводу денежных средств, являющимся кредитной организацией, для принятия от физического лица наличных денежных средств в целях их последующего перевода без открытия банковского счета оператором по переводу денежных средств, а также для осуществления иных операций, предусмотренных Федеральным законом [**Ошибка! Источник ссылки не найден.**] Юридическое лицо или индивидуальный предприниматель, осуществляющие деятельность по приему платежей физических лиц. Платежным агентом является оператор по приему платежей либо платежный субагент [**Ошибка! Источник ссылки не найден.**] Платежный агент - юридическое лицо или индивидуальный предприниматель, заключившие с оператором по приему платежей договор об осуществлении деятельности по приему платежей физических лиц
Платежный терминал	-	Устройство для приема платежным агентом от плательщика денежных средств, функционирующее в автоматическом режиме без участия уполномоченного лица платежного агента [**Ошибка! Источник ссылки не найден.**]
Подтверждение	-	Процесс, в соответствии с которым участник рынка уведомляет своих контрагентов или

		клиентов о деталях сделки и, как правило, предоставляет им время, чтобы подтвердить или оспорить сделку (confirmation) [4]
Подтверждение подлинности электронной цифровой подписи в электронном документе	-	Положительный результат проверки соответствующим сертифицированным средством электронной цифровой подписи с использованием сертификата ключа подписи принадлежности электронной цифровой подписи в электронном документе владельцу сертификата ключа подписи и отсутствия искажений в подписанном данной электронной цифровой подписью электронном документе [4]
Пользователь	-	Пользователями платежной системы являются организации, включающие в себя как институциональных участников инфраструктурных сетей, так и их клиентов на рынках конечных пользователей, которые приобретают и используют различные платежные услуги (user) [3]
Пользователь сертификата ключа подписи	-	Физическое лицо, использующее полученные в удостоверяющем центре сведения о сертификате ключа подписи для проверки принадлежности электронной цифровой подписи владельцу сертификата ключа подписи [3]
Поставщик	-	Юридическое лицо, за исключением кредитной организации, или индивидуальный предприниматель, получающие денежные средства плательщика за реализуемые товары (выполняемые работы, оказываемые услуги) в соответствии с настоящим Федеральным законом, а также юридическое лицо или индивидуальный предприниматель, которым вносится плата за жилое помещение и коммунальные услуги в соответствии с Жилищным кодексом Российской Федерации, а также органы государственной власти и органы местного самоуправления, бюджетные учреждения, находящиеся в их ведении, получающие денежные средства плательщика в рамках выполнения ими функций, установленных законодательством Российской Федерации [4]

Правила платежной системы	-	Условия участия в платежной системе, осуществления перевода денежных средств, оказания услуг платежной инфраструктуры и иные условия, определяемые оператором платежной системы в соответствии с Федеральным законом и принимаемые участниками платежной системы только путем присоединения к указанным условиям в целом [4]
Сертификат ключа подписи	-	Документ на бумажном носителе или электронный документ с электронной цифровой подписью уполномоченного лица удостоверяющего центра, которые включают в себя открытый ключ электронной цифровой подписи и которые выдаются удостоверяющим центром участнику информационной системы для подтверждения подлинности электронной цифровой подписи и идентификации владельца сертификата ключа подписи [4]
Сертификат средств электронной цифровой подписи	-	Документ на бумажном носителе, выданный в соответствии с правилами системы сертификации для подтверждения соответствия средств электронной цифровой подписи установленным требованиям [4]
Средства электронной цифровой подписи	-	Аппаратные и (или) программные средства, обеспечивающие реализацию хотя бы одной из следующих функций - создание электронной цифровой подписи в электронном документе с использованием закрытого ключа электронной цифровой подписи, подтверждение с использованием открытого ключа электронной цифровой подписи подлинности электронной цифровой подписи в электронном документе, создание закрытых и открытых ключей электронных цифровых подписей [4]
Транзакция	-	Операция, инициируемая держателем карточки, которая выражается в последовательности сообщений, вырабатываемых и передаваемых друг другу участниками системы для обслуживания держателей карточек (transaction)
Участники платежной системы	-	организации, присоединившиеся к правилам платежной системы, определенным в

		соответствии с Федеральным законом [3]
Электронная платёжная система	-	Система безналичных расчётов, заключения контрактов и перевода денег между продавцами и покупателями, банками и их клиентами с помощью средств электронной коммуникации с применением средств кодирования информации и её автоматической обработки
Электронный документ	-	Документ, в котором информация представлена в электронно-цифровой форме [4]

ВВЕДЕНИЕ

Сегодня платежи от физических лиц являются объектом особого внимания со стороны банковского розничного бизнеса, в особенности это касается платежей в адрес торгово-сервисных предприятий или поставщиков товаров и/или услуг.

Платежи совершаются частными лицами различными способами и по разнообразным каналам — наличными и безналичными средствами, через платежные системы и системы электронных денег, через банкоматы и операционные кассы банков, узлы почтовой связи и т. д. и т. п.

По различным оценкам, совокупный оборот крупнейших платежных систем, систем электронных денег и интернет-платежей в 2009 году превышал 1 триллион рублей, и в последующие годы он будет только расти.

Не секрет, что львиная доля этих платежей вносится наличными. Обычным сценарием оплаты услуг физическим лицом является следующий: дважды в месяц наличные деньги изымаются из банкомата зарплатного банка. Далее, оплата той или иной услуги производится наличными, либо через платежные терминалы (сотовые операторы, реже — интернет-провайдеры, совсем редко — другие платежи), либо через операционные отделения тех банков, в которых открыты расчетные счета получателей платежей. Парадокс заключается в том, что даже если товар приобретается в онлайн режиме через тот или иной интернет-магазин, то в подавляющем большинстве случаев оплата производится наличными курьеру! Описанная ситуация требует от участников расчетов огромных затрат и предоставляет серьезные неудобства плательщикам.

Проблема может быть решена банковским сообществом путем предоставления физическим лицам удобного и защищенного инструмента для проведения оплат необходимых товаров и/или услуг безналичным способом в онлайн режиме.

Такой инструмент существует на рынке достаточно давно — это системы дистанционного банковского обслуживания (ДБО). Использование сервисов ДБО на порядок удобнее и безопаснее, но и на этом пути, существуют сложности, которые в настоящее время не позволяют перенести значительную часть платежей в онлайн.

2 НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

2.1 Основные архитектуры распределенных систем.

Распределенная система - это набор независимых компьютеров, представляющихся их пользователям единой объединенной системой.

Основная задача распределенных систем программного обеспечения - облегчить их пользователям доступ к удаленным ресурсам, а также контролировать совместное использование этих ресурсов. Ресурсы могут быть виртуальными, но могут быть и традиционными - компьютерами, принтерами, устройствами хранения файлов, файлами и данными.

В качестве основного требования к распределенным системам предъявляется достижение их прозрачности, открытости и масштабируемости.

2.2 Способы взаимодействия в распределенных системах

Основной характеристикой способа взаимодействия подсистем распределенной системы является его синхронность или асинхронность. По отношению к исследуемым системам точнее говорить о блокирующем или неблокирующем взаимодействии. Блокирующим называется взаимодействие, если вовлеченные стороны прежде, чем переходить к следующим работам, должны дожидаться окончания взаимодействия. Следует понимать, что параллельная работа фрагментов систем не имеет никакого отношения к асинхронности и неблокирующему взаимодействию. Сервер, получив запрос от одного из своих клиентов, может, обрабатывая запрос, работать параллельно другим своим клиентам, синхронность же относится к работе запрашивающего обслуживания клиента и того процесса в сервере, который этот запрос обрабатывает.

Если работа клиента на время обработки запроса сервером приостанавливается - это синхронное взаимодействие. Если вместо блокировки клиент после выдачи запроса выполняет другие действия – это взаимодействие асинхронное:

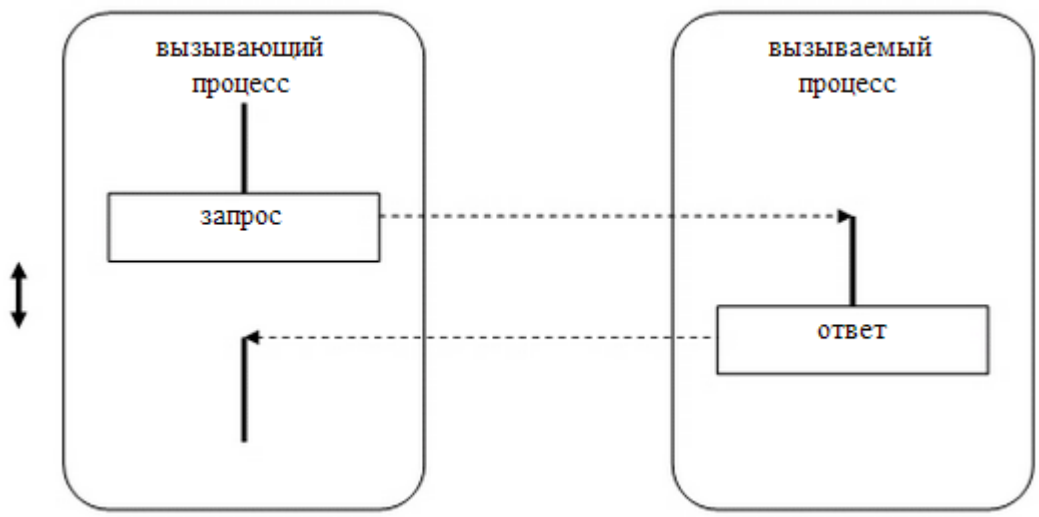


Рис.1. Синхронное взаимодействие.

Синхронное взаимодействие (Рис.1) обладает преимуществом простоты. Обрабатывая запрос, сервер может быть уверен, что состояние клиента во время этой обработки не изменится. В результате синхронное взаимодействие применяется в подавляющем большинстве систем промежуточного слоя. Эти преимущества являются одновременно и недостатками. Синхронное взаимодействие приводит к существенным потерям времени, а значит и производительности. Ожидающий процесс может, вообще, быть удален из оперативной памяти, что приведет к еще большим потерям времени.

Однако не всегда требуется работать синхронно. Один из примеров асинхронной связи - электронная почта. Асинхронные распределенные системы строятся аналогичным образом. Вместо вызова и ожидания ответа выполняется отправка сообщения (Рис. 1.4), а через некоторое время программа может проверить, прибыл ли ответ. Это позволяет программе

выполнять другие работы и делает ненужной какую-либо координацию между сторонами взаимодействия

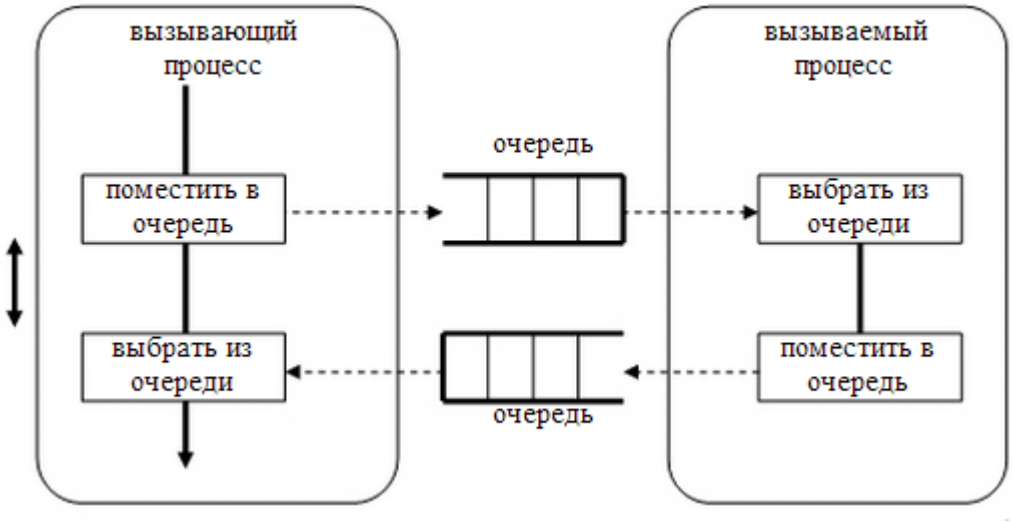


Рис.2. Асинхронное взаимодействие.

Для асинхронного взаимодействия сообщения должны запоминаться в некотором промежуточном месте, откуда они впоследствии могут извлекаться сервером. Эта промежуточная память открывает возможности для новой функциональности, которую больше не требуется делать частью отдельных компонентов. Такие системы очередей теперь превратились в брокеры сообщений, которые фильтруют их и управляют потоком сообщений. Тем самым появляется возможность менять способы фильтрации, преобразования и доставки сообщений без изменения самих компонентов, генерирующих и получающих эти сообщения.

2.3 Основные механизмы в распределенных системах

Согласно [2], ключевые различия ИС определяются тремя их архитектурными компонентами:

1. Схема или модель данных, которая лежит в их основе;
2. Технологический стек, на котором реализовано приложение (базовое программное обеспечение – среда времени исполнения, то есть СУБД, сервер приложений и т. д.);

3. Модель бизнес-процессов. Различие в моделях бизнес-процессов и в механизмах их реализации является главным препятствием для того, чтобы приложения стали частью единой области деятельности.

Кроме концептуальной стороны вопроса о сопряжении ИС следует также определить технические средства для решения данной задачи. На основании анализа работ [2–4] можно выделить несколько наиболее популярных на сегодня методов интеграции приложений:

- **Передача файлов.** Взаимодействие между приложениями осуществляется с помощью файлов, в которые помещаются общие данные.
- **Общая база данных.** Взаимодействие между приложениями осуществляется с помощью базы данных, в которой сохраняется общая информация.
- **Удаленный вызов процедуры.** Взаимодействие между приложениями осуществляется с помощью удаленного вызова процедур, использующихся для выполнения действий или обмена данными.
- **Обмен сообщениями.** Взаимодействие между приложениями осуществляется с помощью системы обмена сообщениями, которые используются для обмена данными и выполнения действий.

2.3.1 Передача файлов

Файлы— это универсальный механизм хранения данных, встроенный в любую операционную систему и поддерживающийся любым языком программирования. Таким образом, один из самых простых способов интеграции приложений может быть основан на использовании файлов.

Одним из наиболее важных решений является выбор общего формата файлов. До недавнего времени наиболее распространенным стандартным форматом файлов считался простой текстовый файл. Современные интеграционные решения основываются на использовании формата XML.

Приложение, которому требуется передать данные другому приложению, сохраняет их в файле. Ответственность за преобразование форматов файлов ложится на плечи разработчиков интеграционного решения. Частота создания файлов зависит от характера бизнес-логики компании.

Наиболее существенное преимущество файлов заключается в том, что разработчики интеграционного решения не нуждаются в дополнительных сведениях о внутренней реализации интегрируемых приложений. Основная задача интеграторов заключается в преобразовании форматов файлов (если это необходимо). Результатом подобного подхода является слабое связывание интегрируемых приложений. Единственными общедоступными интерфейсами приложений являются создаваемые этими приложениями файлы.

Один из наиболее существенных недостатков передачи файла заключается в возможности рассинхронизации интегрируемых систем вследствие низкой частоты обмена информацией. Таким образом, при определении частоты создания файлов следует всегда исходить из потребностей в наличии актуальной информации считывающих эти файлы приложений.

2.3.2 Общая база данных

Данный подход концептуально очень прост — несколько информационных систем или приложений используют одну базу данных. Главный его недостаток — связь между интегрированными приложениями настолько тесная, что иногда невозможно заметить границу между ними (обычно так интегрируются продукты одного производителя). Примером такого подхода могут служить большинство ERP-систем, где различные модули системы используют одну базу. Однако слишком тесная связь превращает конгломерат интегрированных приложений в монолит, в «суперсистему», отдельные части которой с трудом поддаются

самостоятельной модернизации и замене. С этим борются, используя механизмы серверов баз данных (представления данных, промежуточные таблицы и т.п.), но далеко не всегда эффективно.

2.3.3 Удаленный вызов процедур.

Выбор определенного метода и технологии из данного списка зависит от специфики связываемых ИС. Согласно [3, 4], наибольшая эффективность использования метода удалённого вызова процедур достигается в тех приложениях, в которых существует интерактивная связь между удалёнными компонентами с небольшим временем ответов и относительно малым количеством передаваемых данных.

Распределённые системы доступа к объектам и удалённого вызова процедур называют системами обмена сообщениями. Сообщение реализуется путём посылки пакетов информации получателю (приложению, процессу, потоку, сокету и т. д.). При этом передаваемой информацией могут являться сигналы, данные, а также удаленные вызовы процедур.

В настоящее время выделяют следующие наиболее распространенные технологии реализации метода удаленный вызов процедур:

1. Компонентная объектная модель Component Object Model (COM), разработанная корпорацией Microsoft;
2. Общая архитектура брокеров объектных запросов Common Object Request Broker Architecture (CORBA);
3. Веб-сервисы, в том числе, ESB и SOAP.

2.3.4 Асинхронный обмен сообщениями

Это, пожалуй, единственный из перечисленных подходов, который создавался специально для интеграции информационных систем. Идея концептуально проста и напоминает работу электронной почты. Когда приложению А необходимо вызвать какое-то действие в приложении Б,

оно формирует соответствующее сообщение с данными и инструкциями и отправляет его посредством системы доставки сообщений. Слово «асинхронный» означает, что приложение А не должно ждать, пока сообщение дойдет до Б, будет обработано, сформирован ответ и т.п. Сообщение гарантированно доставляется благодаря механизму очередей сообщений, которые снимают с взаимодействующих систем заботу о надежности сети передачи данных, работоспособности взаимодействующих систем в конкретные моменты времени и т.д.
Недостаток данного подхода — высокая цена. Система гарантированной доставки на основе очередей сообщений обычно сама по себе недешева; единственным известным мне исключением является Microsoft Message Queue (MSMQ), компонент серверных операционных систем семейства Windows. Правда, есть и свободно распространяемые бесплатные (например, ActiveMQ), которые, тем не менее, нужно развернуть, обучить специалистов, поддерживать, написать адаптеры между системой доставки и приложениями и т.д.

Выбор конкретной архитектуры зависит от специфики связываемых ИС.

2.4 Описание предметной области.

Дистанционное банковское обслуживание (ДБО) — общий термин для технологий предоставления банковских услуг на основании распоряжений, передаваемых клиентом удаленным образом (то есть без его визита в банк), чаще всего с использованием компьютерных и телефонных сетей. Для описания технологий ДБО используются различные в ряде случаев пересекающиеся по значению термины: Клиент-Банк, Банк-Клиент, Интернет-Банк, Система ДБО, Электронный банк, Интернет-Банкинг, on-line banking, remote banking, direct banking, home

banking, internet banking, PC banking, phone banking, mobile-banking, WAP-banking, SMS-banking, GSM-banking, TV-banking.

Необходимость выживания в условиях жесткой конкуренции в банковском секторе диктует банкам свои условия, особенно на фоне возросшей культуры потребления физических лиц. Клиентам уже мало просто иметь возможность получить тот или иной продукт – сейчас их интересует качество предоставляемой услуги и временные затраты на ее получение. Разумеется, необходимость при совершении каждой банковской операции выбирать время для личного посещения отделения банка, негативно оценивается клиентами. В данной ситуации конкурентное преимущество получают банки, предоставляющие своим розничным клиентам услугу по дистанционному банковскому обслуживанию (ДБО).
Важно отметить, что многие банки, предоставив, например, клиенту возможность получать sms-сообщения на мобильный телефон о проведенных операциях с его банковской картой, во всеуслышание заявляют о действующей в банке системе ДБО. Подобными действиями формируется абсолютно неверное представление у клиента о реальных возможностях системы ДБО, которые намного шире. По сути система дистанционного банковского обслуживания подразумевает предоставление клиенту практически полного спектра банковских услуг, включая и телефонный, и интернет-банкинг.
Сегодня платежи от физических лиц являются объектом особого внимания со стороны банковского розничного бизнеса, в особенности это касается платежей в адрес торгово-сервисных предприятий или поставщиков товаров и/или услуг.

Платежи совершаются частными лицами различными способами и по разнообразным каналам — наличными и безналичными средствами,

через платежные системы и системы электронных денег, через банкоматы и операционные кассы банков, узлы почтовой связи и т. д. и т. п.

По различным оценкам, совокупный оборот крупнейших платежных систем, систем электронных денег и интернет-платежей в 2009 году превышал 1 триллион рублей, и в последующие годы он будет только расти.

Не секрет, что львиная доля этих платежей вносится наличными. Обычным сценарием оплаты услуг физическим лицом является следующий: дважды в месяц наличные деньги изымаются из банкомата зарплатного банка. Далее, оплата той или иной услуги производится наличными, либо через платежные терминалы (сотовые операторы, реже — интернет-провайдеры, совсем редко — другие платежи), либо через операционные отделения тех банков, в которых открыты расчетные счета получателей платежей. Парадокс заключается в том, что даже если товар приобретается в онлайн режиме через тот или иной интернет-магазин, то в подавляющем большинстве случаев оплата производится наличными курьеру! Описанная ситуация требует от участников расчетов огромных затрат и предоставляет серьезные неудобства плательщикам.

Проблема может быть решена банковским сообществом путем предоставления физическим лицам удобного и защищенного инструмента для проведения оплат необходимых товаров и/или услуг безналичным способом в онлайн режиме.

Специфика банковского взаимодействия определяет то, что при интеграции систем дистанционного обслуживания с различными внешними системами критически важно обеспечить следующие условия:

1. Непрерывность и изолированность работы модулей систем (в случае неправильной работы одного из модулей не должны останавливатся остальные – то есть система должна быть **слабосвязанной**).

2. То, что интеграция идет с системой, через которую проходят сотни транзакций в минуту, требует того, чтобы обмен информацией между системами был как можно более частым.

3. То, что взаимодействие идет с платежным сервисом, требует от архитектуры и построения приложения обеспечения транзакционности обработки запроса (в случае проблем со связью, оборудованием и т.д. системы должны откатиться к прежнему состоянию).

2.5 Построение различных архитектур ИС применительно к сфере банковской автоматизации.

Для организации дистанционного банковского обслуживания, как правило, требуется обеспечить интеграцию системы ДБО с другими информационными системами банка. Архитектура решения и подход к реализации проекта в каждом случае могут быть различным. Как минимум, требуется интеграция системы ДБО с АБС банка, зачастую — с десятком распределенных информационных систем.

На первый взгляд, технология взаимодействия систем не влияет на функциональность систем, однако несоблюдение принципов слабосвязанности приводит к вполне конкретным проблемам:

- Вместо одного стандартного интерфейса для каждой системы разрабатывается и поддерживается несколько «связок», каждая из которых предназначена для интеграции с одной внешней системой. При этом один и тот же функционал (например, функционал обращения к внешней системе по специфичному протоколу),

приходится повторно реализовывать в каждой из интегрируемых систем.

- Из-за большого числа связок, а также ввиду узкой специализации каждой из них, не все они поддерживаются должным образом. В частности, поддержка связки в принципе невозможна, если в качестве интерфейса одной из систем используются внутренние объекты этой системы (процедуры, таблицы и т. п.).
- Изменения в одной системе (например, при смене версии) непредсказуемо влияют на работоспособность связанных систем. Для того, чтобы отследить эти изменения, необходимо интеграционное тестирование — т. е. одновременное тестирование всех систем с привлечением специалистов по каждой из них. Причем такое тестирование нужно производить при любых изменениях в любой из взаимодействующих систем.
- При внедрении новых систем/замене одних систем на другие/изменении технологии интеграции/смены версий систем — т. е. практически при любом изменении IT-инфраструктуры заказчик вынужден привлекать разработчиков прикладных систем, даже если изменений прикладной логики не требуется. Причем из-за недостатка документации требуется привлечение уникальных специалистов, обладающих уникальными знаниями по конкретным связкам. Это часто оказывается сдерживающим фактором при реализации планов заказчика. Более того, отсутствие открытого интерфейса позволяет разработчику ПО блокировать определенные инициативы заказчика (за счет отказа интегрироваться с системами конкурентов).
- При возникновении сбоя на стыке двух систем сложно определить причину и виновника сбоя. Форматы взаимодействия жестко не зафиксированы, ответственности за неизменность внутренних

объектов системы разработчик ПО не несет. Таким образом, ответственных за сбой связок при изменении прикладных систем нет.

Решение проблемы:

Для решения описанных выше проблем достаточно придерживаться простого принципа: у каждой прикладной системы должен существовать поддерживаемый внешний интерфейс, изолирующий прикладную логику от логики интеграционной.

Изоляция интеграционной логики от прикладной обеспечит следующие преимущества:

1. Формализация взаимного влияния систем. При изменениях прикладной системы легко отследить влияние этих изменений на другие системы:
   — если интерфейс не изменился, нет необходимости изменять другие, интегрированные с ней, системы;
   — если интерфейс изменился, то сделанное изменение легко отследить, описать и адаптировать остальные системы под это изменение.
2. Локализация ошибок. Четко определенный формат взаимодействия позволяет однозначно локализовать ошибки, возникающие в процессе эксплуатации, определить их причину (нарушение формата взаимодействия или внутренний сбой одной из систем) и ответственность за их исправление.
3. Снятие интеграционных задач с разработчиков прикладных систем. При наличии стандартных интерфейсов к прикладным системам нет необходимости привлекать разработчиков прикладных систем для решения интеграционных задач при внедрении и сопровождении. Кроме этого, появляется возможность выполнять

интеграционные и прикладные задачи одновременно и параллельно, что особенно актуально при внедрении системы.

4. Повторное использование интерфейсов. Наличие у прикладной системы стандартного интерфейса позволяет использовать один и тот же интерфейс разным внешним системам для различных задач без доработки интерфейса.
5. Использование специальных интеграционных инструментов. Для решения технологических интеграционных задач используются не средства прикладной системы, а специализированные инструменты, в частности, интеграционные шины.

За счет формализации взаимного влияния систем (т. е., по сути, наличия качественной документации на программный интерфейс, описывающей в том числе и его изменения):

- Существенно уменьшается риск сбоя систем в промышленной эксплуатации из-за изменений одной из систем. Для критичных к времени простоя систем это является наиболее значимым фактором.

За счет локализации ошибок:

- Сокращается время на обнаружение сбоя.
- Уменьшаются трудозатраты при локализации сбоя.
- Во многих случаях устраняется необходимость привлечения разработчиков ПО для обнаружения сбоя.
- Увеличивается прозрачность при определении ответственности за сбой программного комплекса.

За счет снятия интеграционных задач с разработчиков прикладных систем:

- Недоступность уникальных ресурсов у разработчика прикладной системы не является ограничителем для заказчика при реализации его планов: банк может внедрять и интегрировать прикладные системы самостоятельно или используя сторонних разработчиков.

- При внедрении прикладной системы разработчик в меньшей степени зависит от готовности и доступности других прикладных систем. Доработку каждой прикладной системы, ее тестирование и даже приемку можно выполнить, не дожидаясь, пока все системы, входящие в программный комплекс, будут доработаны и установлены в единой тестовой среде.
- При решении банка изменить способ интеграции (например, при вводе жестких правил по безопасности взаимодействия) банк может выполнить необходимые изменения самостоятельно — без привлечения разработчиков прикладных систем.
- Уменьшается время внедрения системы за счет того, что доработка всех интегрируемых систем и разработка интеграционного функционала может происходить одновременно и независимо.

За счет повторного использования интерфейсов:

- Если у сторонней системы возникает необходимость в обмене данными, который уже реализован в существующем интерфейсе, — нет необходимости в доработке прикладной системы, в которой этот интерфейс реализован.

За счет использования специальных интеграционных инструментов:

- Уменьшается трудоемкость разработки и сопровождения сложных интеграционных решений.
- С прикладной системы снимается нагрузка по решению интеграционных задач.
- Возможности интеграции прикладных систем не зависят от того, обладают ли они развитой интеграционной функциональностью (взаимодействие по специфическим протоколам, поддержка распределенных транзакций и длительных процессов и т. д.).

- Снимается необходимость повторной реализации сложных интеграционных механизмов в каждой из систем.

2.6 Описание задачи интеграции 2-х систем.

В настоящей работе требуется провести интеграцию 2-х систем: системы Дистанционного банковского обслуживания ДБО-ЧК и платежного сервиса «Рапида». Платежный сервис имеет специфический протокол взаимодействия, приведенный в приложении. При решении задачи интеграции требуется выбрать способ взаимодействия распределенных систем (сравнивая их по степени адекватности к поставленной задаче) и реализовать сервис обмена между платежными системами.

2.7 Выводы по научно-исследовательскому этапу.

После исследования существующих методов интеграции распределенных приложений и специфики сферы банковской автоматизации было принято решение использовать для интеграции банковских сервисов 2 технологии: удаленный вызов процедур (реализация через SOA-сервисы) и асинхронный обмен сообщениями (реализация через технологию JMS – java message service) для взаимодействия с платежной системой. Исследование показывает, что концепция SOA наиболее оптимально ложится на область банковской автоматизации, а специфика взаимодействия с платежными системами требует внедрения более совершенных технологий обмена сообщениями. Таким образом, итогом моего исследования стало то, что в архитектуре разрабатываемого модуля соединено 2 различных подхода к интеграции распределенных систем.

3 КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ

3.1 Цели разработки системы

Основнной целью разработки подсистемы является создание инструмента для интеграции скрупным поставщиком товаров и услуг. Целью системы также является обеспечение клиентов банков возможностью оплачивать услуги через свой «Личный кабинет» в банке и как следствие – повышения уровня сервиса, предоставляемого банком. Также как цель ставится повышение быстродействия и надежности разрабатываемого модуля по сравнению с существующими и как следствие повышение общей надежности системы.

3.2 Условия.

- Слабосвязанная архитектура исходных ИС.
- Решение в рамках интеграционной системы на базе концепции SOA (ИС «ЭПОС»).
- Стандартизированный протокол взаимодействия с системой дистанционного обслуживания.

3.3 Компоненты системы интеграции.

Разрабатываемая подсистема является частью сложной и распрелеленной системы интеграции, содержащей в себе множество взаимодействующих модулей. Основными компонентами системы «Эпос» являются:

1. ***База данных*** в которой хранится информация о поставщиках и кредитных организациях.
2. ***Сервер приложений***, включающий модули:

2.1. *Subsystems.ear* – включает в себя основные компоненты взаимодействующие с базой данных, классы ORM-преобразования, модули загрузки и компоненту интеграции с системой ДБО.

Данный архив состоит из нескольких компонент, отвечающих за различные задачи:

-*BillService.war* – веб-сервис, осуществляющий обработку платежей.

-*CatalogService*.war – веб-сервис, осуществляющий загрузку каталога услуг у поставщиков и формирующий каталоги для банков.

-*DataModel.jar* – компонент, осуществляющий ORM-преобразование таблиц БД в объекты для дальнейшей работы с ними как с сущностями. Как JPA провайдер используется EclipseLink, поставляемый вместе с сервером приложений WebLogic 10.3.3.

- *DocTransfer.war* – общий модуль, отвечающий за преобразование запросов.

- *QuittanceService.war* – веб-сервис, отвечающий за предоставление отчетов и реестров платежей для кредитных организаций и поставщиков услуг.

- *RegistrationService.war* – веб-сервис, отвечающий за регистрацию и автоматическое создание каталога для вновь подключенного банка.

2.2. Сервис-шина OSB.

Сервис-шина Oracle Service Bus – компонент служащий для интеграции различных SOA-приложений. OSB выполняет трансформацию запроса (xslt-преобразования при необходимости), маршрутизацию запроса по различным веб-сервисам, и является точкой входа для SOAP-запроса. Модуль, разработанный в данном дипломном проекте использует сервис-шину только для загрузки

каталога, что приводит к повышению производительности и уменьшению времени обработки запроса.

2.3. PaymentService.ear - в этот модуль входят шлюзы к основным платежным системам, таким как QIWI (ОСМП), А3, ПС ФСГ, Робокасса и др. В дипломном проекте в основном изменения затрагивают этот модуль – добавляется подсистема интеграции с ПС «Рапида».

В общем виде логическая схема системы представлена на рисунке:

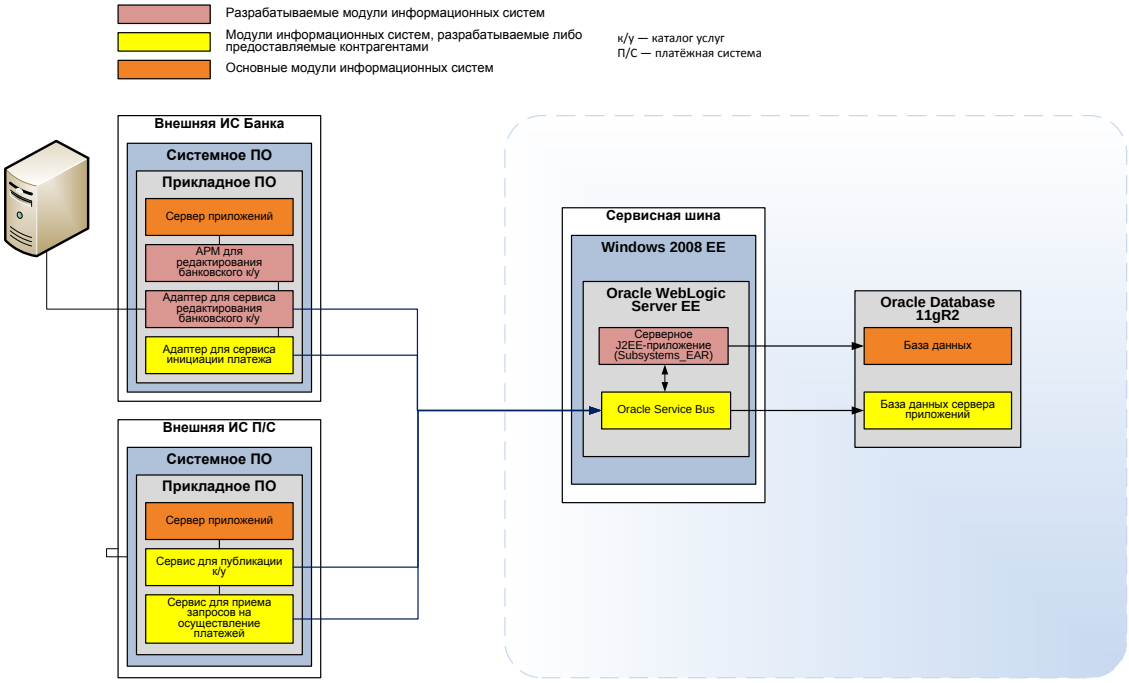


Рис.3. Логическая архитектура системы.

3.4 Физические компоненты развертывания системы «ЭПОС»

Система является распределенной и состоит из нескольких физических серверов, расположенных в разных серверных стойках.

Связь с участниками взаимодействия может осуществляться по 2 основным путям:

1. Использование защищенного канала на сетевом уровне – IPSec-туннель. Реализуется настройкой маршрутизаторов на обеих концах взаимодействия.
2. Испольлзование шифрования пакета на прикладном уровне – SSL-соединение с использованием клиентского сертификата.

Так как связь с интегрируемыми системами осуществляется через сеть Интернет по протоколу TCP/IP, на границе подсети расположен маршрутизатор Cisco 2821 с белым IP-адресом, на который приходят запросы от участников взаимодействия. На этом этапе выполняется маршрутизация и фильтрация по статическим IP-адресам участников. Далее запрос попадает в демилитаризованную зону (ДМЗ), где на границе стоит еще один маршрутизатор, который анализирует пакет и в зависимости от порта перенаправляет запрос на требуемый сервис. Подробная схема взаимодействия указана ниже:

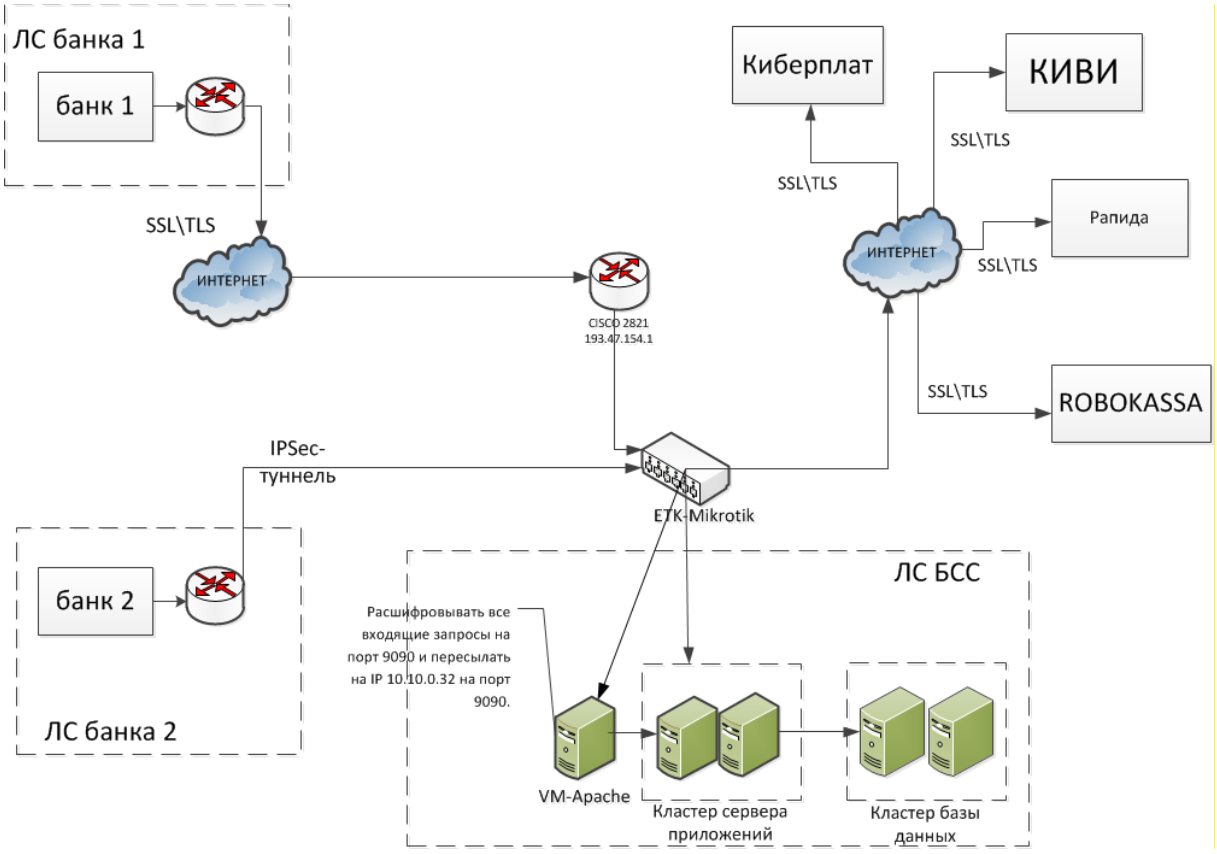


Рис. 4. Физическая архитектура системы

Далее имеются некоторые различия в обработке запроса, в зависимости от того, по какому пути пришел запрос. В случае IPSec-туннеля запрос сразу попадает на сервер приложений, где осуществляется его дальнейшая обработка. В случае соединения по SSL на границе сети, после маршрутизатора стоит сервер, который расшифровывает входящий пакет и отправляет его далее на балансировщик нагрузки.

Физическая архитектура системы на боевой среде состоит из кластера, на котором развернута сервис шина и сервер приложений (OSB 11.1.1.3 + WebLogic 10.3.3), с балансировщиком нагрузки на базе Apache. В качестве операционной системы используется Win2008 R2 x64. В зависимости от загруженности нод кластера запрос попадает либо на один сервер, либо на другой.

Сервер приложений работает с базой данных Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit развернутой на другом кластере.

Тестовая архитектура системы повторяет боевую за исключением того, что на тестовой среде прикладное ПО развернуто не на кластере, а на единичных машинах.

Тестовые модули системы дистанционного банковского обслуживания разных версий расположены на виртуальных машинах и доступны из любой машины подсети.

3.5 Архитектура подсистемы взаимодействия с ПС «Рапида»

После решения поставленной задачи в распоряжении клиента банковского сервиса должна появиться возможность оплачивать товары и услуги, предоставленные ПС «Рапида». Для этого необходимо добавить компоненты, отвечающие за запрос к платежной системе, выполняющие все необходимые преобразования над запросом, приходящим из системы ДБО чтобы трансформировать его в запрос по протоколу ПС «Рапида». Также в этом модуле необходимо осуществить шифрование и подпись запроса для осуществления безопасности платежей.

Поставленная задача решалась путем проектирования «снизу-вверх».

Выбор способа взаимодействия был обусловлен спецификой запросов: согласно протоколу, первый запрос, который отправляется в систему должен быть синхронным, то есть выполняться с ожиданием пользователя ответа от сервиса. Для этого как нельзя лучше подходит способ удаленного вызова процедур RPC

Сначала был написан низкоуровневый класс, содержащий в себе статические методы установления соединения с ПС, шифрования запроса и преобразования ответа в формат для дальнейшей обработки – RapidaClient.jar.

Далее над этим классом надстраивались еще несколько:

1. RapidaCatalogService.war – веб-сервис, отвечающий за загрузку каталога платежной системы. Данный сервис анализирует запрос на загрузку каталога, пришедший из системы ДБО, сохраняет запрос в БД, логирует его, а затем вызывает клиента к ПС. Далее данный сервис анализирует ответ ПС, выявляя ошибки и статусы, отличные от нормальной работы и подготавливает ответ для системы ДБО. В случае ошибок от платежной системы он выводит клиенту системы читаемое сообщение об ошибке.
2. PaymentService.war – веб-сервис, отвечающий за обработку платежей по протоколу ПС. Аналогично предыдущему, он принимает запрос на платеж от системы ДБО и выполняет полный цикл взаимодействия, от отправки запроса на проверку реквизитов платежа до запроса на подтверждение платежа.
3. PaymentBean.java – вспомогательная компонента, отвечающая за сохранение платежа в базе данных, валидирующая реквизиты и выполняющая асинхронное взаимодействие (для запросов проверки проведения платежа).

При взаимодействии с ПС по протоколу «Рапиды» запросы приходят в формате xml. Работа с этим форматом удобна с точки зрения объектно-ориентированного программирования тем, что запросы и ответы xml можно представить объектами и работать дальше с ними как с сущностями. Поэтому для обработки ответов из ПС «Рапида» было добавлено несколько вспомогательных классов, которые реализуют технологию JAXB – автоматическое преобразование (маршаллинг) xml-запросов в объекты. JAXB предполагает использование специальных аннотаций над полями класса для обеспечения соответствия тегов xml полям класса.

3.6 Рефакторинг подсистемы обработки платежей

Старая архитектура системы предполагала обработку запроса через сервис-шину. Это было сделано в связи с легкостью разработки и интеграции сервисов на OSB и малым временем от проектирования до внедрения. В настоящем проекте было принято решение отказаться от данного модуля и попробовать осуществить интеграцию с ПС «Рапида» используя технологию EJB 3.0. Данная технология является стандартом в области разработки распределенных систем, и большая сложность ее освоения и реализации компенсируется надежностью и быстротой сервисов. Данная технология поддерживает транзакционность на уровне сервера приложений, что позволяет разработчику не думать об управлении своими транзакциями. Важно и то, что решения на базе EJB 3.0 намного более легкие и занимают в памяти гораздо меньше места. Таким образом, веб-сервис, отвечающий за разработку будет также осуществлять маршрутизацию запроса.

3.7 Техническое задание на систему.

3.7.1 Полное наименование работы и ее условное обозначение

Полное наименование работы: Интеграции системы «ДБО BS-Client. ЧК» с сервисами платежной системы Rapida с использованием сервисов Автоматизированной системы поддержки выставления и оплаты счетов за оказанные услуги и поставленные товары «Электронное предоставление и оплата счетов (ЭПОС)».

Условное обозначение работы: «Интеграция ЭПОС с Рапида-ДБО.ЧК»

Ввод в действие осуществляется в соответствии с «ГОСТ 34.601-90. ГОСТ 24.601-86. Автоматизированные системы. Стадии создания». Порядок оформления и предъявления заказчику результатов работ по

«Интеграции ЭПОС с Рапида-ДБО.ЧК» определяется контрактом (условиями договора).

3.7.2Основания для разработки

1) Разработка ведется на основании следующих документов:

- Частное техническое задание на систему.

- Календарный план на выполнение работ.

2) Документы утверждены « 12 » декабря 2012 года.

3.7.3Назначение разработки

«Интеграция ЭПОС с Рапида-ДБО.ЧК» предназначена для предоставления сервисов системы Rapida клиентам банков - пользователям системы «ДБО BS-Client. ЧК».

3.7.4Цели и эффекты от интеграции

Целью «Интеграции ЭПОС с Рапида-ДБО.ЧК» является создание в рамках системы «ЭПОС» единого технического решения, предлагаемого существующим и потенциальным владельцам системы «ДБО BS-Client. ЧК» в качестве дополнительной опции.

Положительные эффекты от «Интеграции ЭПОС с Рапида-ДБО.ЧК» выражены в повышении конкурентоспособности системы «ЭПОС» за счет предложения банкам - пользователям системы «ЭПОС» дополнительных сервисов и, как следствие, возможности экономии средств, затрачиваемых банком на доработки собственных систем.

«Интеграция ЭПОС с Рапида-ДБО.ЧК» дает банкам следующие преимущества:

- Увеличение количества сервисов системы «ДБО BS-Client. ЧК», предлагаемых клиентам банков, и как следствие усиление привлекательности банка для клиентов;

- Получение дополнительного комиссионного дохода от проведения клиентами банка операций с электронными кошельками.
- Увеличение пассивов банка за счет депозита компании-уполномоченного агента Rapida, открываемого в Банке для обеспечения операций с электронными кошельками.
- Снижение расходов на KYC-мониторинг.

3.7.5Характеристика объектов автоматизации

В соответствии с настоящим ЧТЗ объектами автоматизации являются:

1) процесс предоставления следующих сервисов системами «ЭПОС» и «ДБО BS-Client. ЧК»:

- покупка услуг и товаров, номинированных в каталоге ПС «Рапида», с оплатой с клиентского счета в банке без использования электронного кошелька.

3.7.6Общее описание автоматизируемых сервисов

Взаимодействие осуществляется путем взаимного вызова веб – служб (WS), размещенных на обеих сторонах. Канал взаимодействия защищен SSL. Взаимодействие между системами основано на принципе «запрос-ответ». Система-инициатор обращается с запросом на выполнение определенной операции к WS, расположенному на отвечающей стороне. Запрос передается по протоколам HTTP (для «ДБО BS-Client. ЧК») и SOAP. Канал взаимодействия между ЭПОС, «ДБО BS-Client. ЧК» и ПС «Рапида» защищен средствами SSL.

Сторона-инициатор взаимодействия после отправки запроса ждет результат обработки. Для сервисов «Рапида» устанавливается предельное время ожидания возврата результата (тайм-аут).

По результатам обработки запроса WS формирует выходное SOAP-соединение в виде набора данных, описанного в выходных параметрах для

каждого поставщика услуг. Отсутствие ответа поставщика, как и ошибки, возникающие в процессе обработки запроса, являются исключительными ситуациями, обрабатываемыми вызывающей стороной.

Предполагается, что отсутствует техническая возможность формировать запросы к системе «ДБО BS-Client. ЧК». Отсюда следует, что передача управления системе «ДБО BS-Client. ЧК» от системы «ЭПОС» должно строиться на механизме опросов со стороны системы «ДБО BS-Client. ЧК».

Предлагается этот механизм строить на основе двух типов запросов, формируемых системой «ДБО BS-Client. ЧК»:

Периодический запрос к системе «ЭПОС» на поставку ID и назначений существующих, но ранее непоставленных внешних вызовов к системе «ДБО BS-Client. ЧК»

Запрос к системе «ЭПОС» на поставку внешнего вызова с заданным ID. Формируется на основании анализа данных, полученных по первому запросу, в соответствии с внутренней логикой системы «ДБО BS-Client. ЧК»

3.7.7Требования к подсистеме

3.7.7.1 Общие требования к интегрирующим сервисам

Поскольку интеграция с системой «Рапида» в существующей конфигурации невозможна без реализации реакции системы «ДБО BS-Client. ЧК» на запросы системы «Рапида», то с учетом невозможности на момент написания настоящего ЧТЗ внешних вызовов сервисов «ДБО BS-Client. ЧК», представляется целесообразным оснащение «ДБО BS-Client. ЧК» универсальным механизмом опроса системы ЭПОС на предмет наличия в ней сохраненных запросов к системе «ДБО BS-Client. ЧК» от внешних систем, включая Рапиду и сам ЭПОС.

3.7.7.2 Требования к интеграции

Система ЭПОС должна поддерживать порядок взаимодействия систем «ДБО BS-Client. ЧК» и Рапида.

Должны быть разработаны интерфейсы системы ЭПОС и преобразователи форматов, обслуживающие следующие запросы:

Таблица 1. Типы запросов к платежной системе

	HTTP запросы от «ДБО BS-Client. ЧК» к системе ЭПОС	SOAP запросы от системы ЭПОС к системе Рапида	SOAP запросы от системы Рапида к системе ЭПОС	Преобразователи форматов сообщений в системе ЭПОС
	Общие запросы			
1.	Запрос на поставку внешнего вызова с заданным ID			Рапида XML -> eBPP XML
2.	Запрос на поставку Рапида -инвойсов			eBPP XML
3.	Запрос на подтверждение \ отмену Рапида -инвойсов			eBPP XML
4.		Запрос на подтверждение \ отмену Рапида -инвойсов		eBPP XML -> Рапида XML
5.			Отмена Рапида -инвойса	Рапида XML -> eBPP XML
6.			Подтверждение возможности платежа	Рапида XML -> eBPP XML
7.	Запрос подтверждения возможности платежа			eBPP XML
8.	Запрос-подтверждение платежа			eBPP XML
9.		Запрос-подтверждение платежа		eBPP XML -> Рапида XML
10.				

3.7.8Порядок контроля и приемки подсистемы

К сдаточным единицам системы относятся:

- Комплекс средств автоматизации, установленный на объекте внедрения.
- Комплекс средств автоматизации оператора системы.

Сдача системы должна осуществляться на объектах внедрения согласно перечню объектов и план-графику внедрения, утвержденным заказчиком.

Приемка должна осуществляться в виде проведения испытаний.

К видам испытаний относятся:

- Опытная эксплуатация.
- Приемочные испытания.

Состав, объем и методы испытаний должны быть определены в документах «Программа и методика испытаний», разрабатываемых для составных частей системы на этапе рабочей документации в соответствии с действующими нормами.

Перечень участвующих предприятий, организаций, место и сроки проведения испытаний, порядок согласования и утверждения приемочной документации должны определяться в организационно-распорядительной документации (приказы).

Для проведения испытаний должны быть созданы приемочные комиссии:

- для приемки системы в опытную эксплуатации составных частей системы,
- для проведения приемочных испытаний составных частей системы,
- для проведения испытаний системы.

3.7.9 Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу подсистемы в действие.

Состав работ должен включать:

- работы по подготовке пилотных объектов (площадок) к вводу в опытную эксплуатацию опытных образцов ППО,
- работы по подготовке объектов внедрения к вводу в действие системы.

Должно быть проведено обучение пользователей в соответствии с утвержденными программами проведения обучения и с обязательным контролем знаний.

- На объекте внедрения должны быть проведены следующие мероприятия:
- поступающая в систему информация должна быть приведена к виду, пригодному для обработки на ЭВМ,
- должны быть созданы условия функционирования системы, при которых гарантируется соответствие создаваемой системы требованиям настоящего ТЗ,
- должна быть обеспечена необходимая комплектация пользователями и обслуживающим персоналом.

Завершение работ по подготовке объекта внедрения к вводу в действие системы оформляется актом, подписываемым заказчиком, исполнителем, исполнителями работ по внедрению составных частей системы и организацией - объектом внедрения.

4 ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

4.1 Типы запросов к платежной системе «Рапида»

В рамках данного проекта к платежной системе «Рапида» выполняется 4 типа запросов:

1. Запрос на загрузку каталога услуг.

Синхронный запрос. Выполняется по протоколу ПС «Рапида», формат запроса имеет вид: http://<URL_тестового_сервиса>/?function=getfee. В ответ ПС «Рапида» возвращает XML файл со справочником получателей и их параметров, необходимых для проведения запросов на платежи.

Данный запрос ввиду большого размера возвращаемого файла парсится средствами сервис-шины и трансформируется в формат, понятный ПС «ЭПОС». После преобразования на сервис-шине запрос на загрузку с необходимыми параметрами передается в модуль взаимодействия с ПС «Рапида» (RapidaCatalogService), где осуществляется вызов сервиса. В ответ получается XML-документ, содержащий получателей платежей и параметры, такие как номер телефона, аккаунта или квартиры. После этого полученный файл отправляется на сервис-шину и там выполняется его дальнейшее преобразование.

2. Запрос на проверку правильности реквизитов платежа.

Запрос является синхронным. Согласно протоколу платежной системы необходимо вызвать процедуру с 8 параметрами:

Таблица 2. Параметры запроса на проверку платежа

	Описание параметра	Параметр	Формат
1	Идентификатор запроса	Function	Константа = «check»
2	Уникальный номер платежа у Участника Системы. При запросе на платеж, этот код должен соответствовать коду в запросе на проверку. **Замечание!** Уникальность этого номера проверяется только для платежей, давность которых не превышает 30 суток.	PaymExtId	От 2 до 20 символов.
3	Код ТСП в ПС «Рапида»	PaymSubjTp	Цифровой код получателя в ПС «Рапида»
4	Сумма платежа/денежного перевода	Amount	Сумма в копейках (три и более цифр). Например: 053, 2000.
5	Коды и значения параметров платежа.	Params	Строка с параметрами платежа в специальном формате см. п. 1.3.
6	Тип платежного инструмента	TermType	Код, в соответствии с таблицей в Приложении "*Типы платежных инструментов*"
7	Идентификатор ППП	TermId	**От 2-х до 7-ми** символов. Допускается использовать **цифры** от 0 до 9 и **заглавные буквы латинского алфавита** (A-Z). Уникальный номер, соответствующий списку зарегистрированных терминалов
8	Сумма комиссии с Клиента (Плательщика)	FeeSum	Сумма в копейках комиссии с Плательщика. Если комиссия не берется – указывается 0.

Запрос выполняется из модуля PaymentService.war. Ответ системы возвращается сразу и выполняется маршаллинг его в объект. Далее с объектом ответа идет работа как с обычным компонентом: вызываются методы проверки полей, анализируется наличие ошибок / нестандартных статусов в объекте ответа. В зависимости от успешности / неуспешности проведения проверки модуль либо возвращает ошибку на этапе проверки, либо далее вызывает PaymentBean для сохранения платежа в базу данных и выполнения запроса на проверку платежа.

3. Запрос на исполнение плаетжа.

Запрос может быть асинхронным, и выполняться несколько раз.

Процедура вызывается с 8 параметрами:

Таблица 3. Параметры запроса на исполнение платежа

	Описание параметра	Параметр	Формат
1	Идентификатор запроса	Function	Константа = «payment»
2	Уникальный номер платежа (денежного перевода) у Участника Системы. При запросе на платеж, этот код должен соответствовать коду в запросе на проверку. **Замечание! Уникальность этого номера проверяется только для платежей, давность которых не превышает 30 суток.**	PaymExtId	От 2-х до 20 символов
3	Код ТСП в ПС «Рапида»	PaymSubjTp	Цифровой код получателя в ПС «Рапида»
4	Сумма платежа (денежного перевода)	Amount	Сумма в копейках (три и более цифр). Например: 053, 2000
5	Коды и значения параметров платежа.	Params	Строка с параметрами платежа в специальном формате (п.1.3).
6	Тип платежного инструмента	TermType	Цифровой код, в соответствии с таблицей
7	Идентификатор ППП (Уникальный номер, соответствующий списку зарегистрированных терминалов)	TermId	**От 2-х до 7-ми** символов. Допускается использовать **цифры** от 0 до 9 и **заглавные буквы латинского алфавита** (A-Z)
8	Сумма комиссии с Клиента (плательщика)	FeeSum	Сумма в копейках комиссии с Плательщика. Если комиссия не берется – указывается 0.
9	Дата/время фактического формирования операции на стороне Участника Системы с указанием часового пояса ППП* .	TermTime	Дата/время в формате ISO 8601 YYYYMMDDThhmmss±hhmm

Данный запрос служит и для исполнения, и для последующей проверки успешности проведения платежа. В зависимости от статуса ответа из ПС «Рапида» запрос либо прекращает выполнение, либо повторяется с периодичностью раз в 2 минуты. Это достигается путем применения технологии JMS. Данная технология предполагает наличие очередей

сообщений, в которые отправляется запрос. В случае недоступности получателя, или ошибочного статуса сервер приложений будет отправлять данное сообщение несколько раз с заданной периодичностью, пока сообщение не попадет в очередь ошибок (настраиваемая). Также все действия выполняются в транзакции. За выполнение указанной функции отвечает модуль PaymentBean.

4. Запрос на отмену платежа

Запрос является синхронным и включает 2 параметра:

Таблица 4. Параметры запроса на отмену платежа.

	Описание параметра	Параметр	Формат
1	Идентификатор запроса	Function	Константа = «cancelreq»
2	Идентификатор запроса у Участника Системы.	PaymExtId	От 2 до 20 символов.

4.2 Шифрование запроса в ПС «Рапида».

Взаимодействие с любой платежной системой должно осуществляться с применением средств криптографической защиты информации. В ПС «Рапида» для обеспечения конфиденциальности, целостности и доступности информации применяется соединение по протоколу https с авторизацией по клиентскому сертификату. Это означает, что клиент ПС «Рапида» должен сгененрировать ключ для шифрования и цифровой подписи запроса и получить на него сертификат, выпущенный данной ПС. Далее запросы к ПС «Рапида» подписываются открытым ключом платежной системы и шифруются закрытым ключом банка-клиента. Запос на сертификат выполняется средствами утилиты OpenSSL, также с ее помощью полученный сертификат преобразуется в формат PKCS12 (хранилище) для того чтобы использоваться в запросах.

В данной работе данный функционал реализован с помощью библиотеки javax.net.ssl и находится в модуле RapidaClient.java.

4.3 Реализация архитектуры подсистемы

На концептуальном этапе проектирования был сделан вывод о необходимости разделения низкоуровневых функций, отвечающих за связь с базой данных и веб-сервисов, выполняющего маршрутизацию запроса.

На этапе рабочего проектирования был сделан модуль отвечающий за связь системы с ПС «Рапида» - RapidaClient. Данный модуль получает на вход параметры соединений с ПС, а также параметры запросов и создает соединение с БД по протоколу https. Данный модуль должен вызываться всегда, когда выполняется взаимодействие с ПС «Рапида». В него также включены базовые операции, выполняемые с ответом из «Рапиды» - преобразование потока байт в текстовый объект для последующей обработки.

На верхнем уровне существует несколько классов, отвечающих за маршрутизацию запроса и работу с JMS-очередями и базой данных. Элемент PaymentService.war является точкой входа сервиса, и при получении запроса на платеж осуществляет вызов RapidaClient с нужными параметрами, проверку полей ответа и в зависимости от этого маршрутизацию запроса далее. В случае успешного ответа на запрос проверки реквизитов вызывается модуль PaymentBean, выпоняющий схожие функции, но работающий с очередями сообщений. PaymentBean через механизм ResourceInjection получает Factory для взаимодействия с базой данных, а также названия очередей на сервере приложений. Как вспомогательные классы были спроектированы интерфейсы и классы ответов к ПС: RapidaResponse, RapidaPaymentStatus, JaxbRapidaResponse и др. Для формирования из них объектов были применены аннотации, согласно технологии Jaxb. Пример класса с аннотацией:

```
@XmlRootElement(name = "Response")
@XmlAccessorType(XmlAccessType.FIELD)
public class JaxbRapidaResponse implements RapidaResponse {

    private String Result;
    private String PaymNumb = "-1";
    private String PaymExtId;
    private String Balance;
    private String ResCode = "-1";
    private String Description;
```

5 РАБОЧЕЕ ПРОЕКТИРОВАНИЕ

5.1 Реализация запросов к ПС «Рапида»

Вызов платежной системы «Рапида» выполняется из модуля DefaultRapidaClient и осуществляется функцией getRapidaResponse объекта Request:

```
public byte[] getRapidaResponse();
```

Данный метод вызывается каждый раз когда требуется обратиться к ПС. Метод формирует URL для соединения с параметрами запроса

```
if(method == HTTP_METHOD.GET){
                    urlStr += (urlStr.indexOf('?') == -1 ? "?":
"&") + paramsStr;
                }
                _url = new URL(null, urlStr, new
sun.net.www.protocol.https.Handler());
```

А также создает объект соединения, устанавливает метод запроса (GET), таймаут подключения и ожидания ответа:

```
                try {
                    conn =
(proxy==null?_url.openConnection():_url.openConnection(proxy));
                    if(sslsf != null  && conn instanceof
HttpsURLConnection)

((HttpsURLConnection)conn).setSSLSocketFactory(sslsf);
                } catch (Exception ex) {
                    throw new RuntimeException(ex);
                }

                conn.setReadTimeout(100000);
                conn.setConnectTimeout(100000);
```

Далее на основании данного метода осуществляются вызовы процедур ПС с параметрами согласно протоколу:

5.1.1Реализация запроса на загрузку каталога услуг.

Запрос на загрузку каталога услуг выполняется из модуля DefaultRapidaClient и осуществляется статической функцией getFee:

```
            public String getFee()
```

Метод загружает параметры вызова процедуры ПС «Рапида» вызова справочника, а затем вызывает вышеуказанный метод getRapidaResponse:

```
Map<String, String> reqParams = new LinkedHashMap<String, String>();
        reqParams.put("function", FUNCTION_GET_FEE);
        Request r = new Request(url, Request.HTTP_METHOD.GET, reqParams,
sslsf, proxy) {
            @Override
            protected byte[] responseProcessor(byte[] data) {
                return data;
            }
        };
        byte[] fee = r.getRapidaResponse();
```

Далее данный метод создает из потока байт ответа ПС «Рапида» объект типа Document для последующего парсинга:

```
        Document doc = null;
        try{
            Document doc1;
            DocumentBuilder builder;
            DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
```

Метод возвращает строку, соответствующую ответу Рапиды в кодировке ср1251.

5.1.2Реализация запроса на проверку реквизитов платежа.

Запрос на проверку параметра платежа производится методом

```
public JaxbRapidaResponse check(String paymExtId, int paymSubjTp,
List<String> params, int amount, int feeSum);
```

Метод принимает на вход параметры для проверки платежа, такие как уникальный номер транзакции, тип вызова, параметры, сумма платежа и

комиссия за платеж. Далее данный метод вызывает метод sendRapidaRequest() с указанными параметрами:

```
private JaxbRapidaResponse sendRapidaRequest(String functionName, String
paymExtId, int paymSubjTp, List<String> params, int amount, int feeSum);
```

Метод подставляет указанные параметры в запрос и вызывает getRapidaResponse:

```
                Map<String, String> reqParams = new LinkedHashMap<String,
String>();
                reqParams.put("function", functionName);
                reqParams.put("PaymExtId", paymExtId);
                reqParams.put("PaymSubjTp", String.valueOf(paymSubjTp));
                reqParams.put("Amount", moneyFormat.format(amount));
                String paramsStr = "";
```

Далее с полученным потоком байт выполняется трансформация, и создается возвращаемый JAXB-объект:

```
            InputSource is = new InputSource( new ByteArrayInputStream(
byteRapidaResponse ) );
            try {
                doc1 = builder.parse( is );
//                System.out.println("Message body: \n\t" + new
String(byteRapidaResponse, Charset.forName("cp1251")).replace("\n",
"\n\t"));
                rapidaResponseJaxb =
rapidaPaymentHelper.getRapidaResponseJaxb(doc1);
                return rapidaResponseJaxb;
```

Метод возвращает JAXB-объект для дальнейшей работы с ним.

5.1.3Реализация запроса на исполнение платежа.

Запрос на исполнение платежа проводится методом

```
public JaxbRapidaResponse payment(String paymExtId, int paymSubjTp,
List<String> params, int amount, int feeSum)
```

Метод принимает на вход параметры для проведения платежа (уникальный номер платежа, тип запроса к ПС «Рапида», параметры платежа, сумма и сумма комиссии). Далее он вызывает метод sendRapidaRequest с указанными параметрами.

Возвращаемый JAXB объект дальше используется для обработки запроса.

5.1.4Реализация запроса на отмену платежа.

Запрос на отмену платежа проводится методом

```
public JaxbRapidaResponse cancel(String paymExtId)
```

Метод принимает на вход уникальный номер платежа и вызывает метод sendRapidaRequest с указанным параметром. Далее, в зависимости от ответа платежной системы выполняется обработка ответа.

5.2 Реализация защищенного взаимодействия с ПС «Рапида»

Для защиты канала связи с платежной системой применяется взаимодействие по протоколу HTTPS с авторизацией по клиентскому сертификату. Для реализации взаимодействия были предприняти предварительные шаги:

а) С помощью утилиты OpenSSL был сгенерирован запрос на сертификат, отправляемый в ПС «Рапида». Запрос:

```
openssl req \
  -new -newkey rsa:1024 -nodes \
  -keyout mykey.pem -out myreq.pem
```

Далее указанный запрос отправляется в ПС «Рапида» для подписи. В ответ возвращается сертификат клиента. Ключ, соответствующий данному сертификату хранится у клиента и используется для подписи запросов к ПС.

б)Для использования в запросах ключ и сертификат необходимо преобразовать в формат хранилища сертификатов (keystore). Обычно используется 2 вида хранилища: PKCS (стандарт PKCS12) и JKS (java-хранилище сертификатов). Преобразование выполняется с помощью утилиты OpenSSL запросом вида:

```
# export mycert.pem as PKCS#12 file, mycert.pfx
openssl pkcs12 -export \
  -out mycert.pfx -in mycert.pem \
  -name "My Certificate"
```

Далее полученный файл формата .pkcs можно использовать при шифровании / подписи запросов, выполняемых по протоколу SSL. Файл содержит зашифрованный закрытый ключ клиента и сертификат.

Параметры указанного хранилища (путь к файлу, пароль закрытого ключа) хранятся в базе данных. Это сделано с целью облегчения замены сертификата при истечении срока его действия.

5.3 Реализация верхнего уровня

Для включения бизнес-логики и верхнеуровневой обработки приходящего запроса от системы дистанционного банковского обслуживания были спроектированы классы, отвечающие за логику обработки запроса и маршрутизирующие запрос в зависимости от полей.

5.3.1Реализация логики загрузки каталога.

Запрос на загрузку каталога в систему поступает по таймеру, каждый день в 00.00. Каталог услуг, ввиду большого размера и сложности xml-файла загружается на сервис-шине, вызывающей затем веб-сервис связи с ПС. При послуплении запроса на загрузку сначала на сервис-шине выполняется xslt-преобразование запроса:

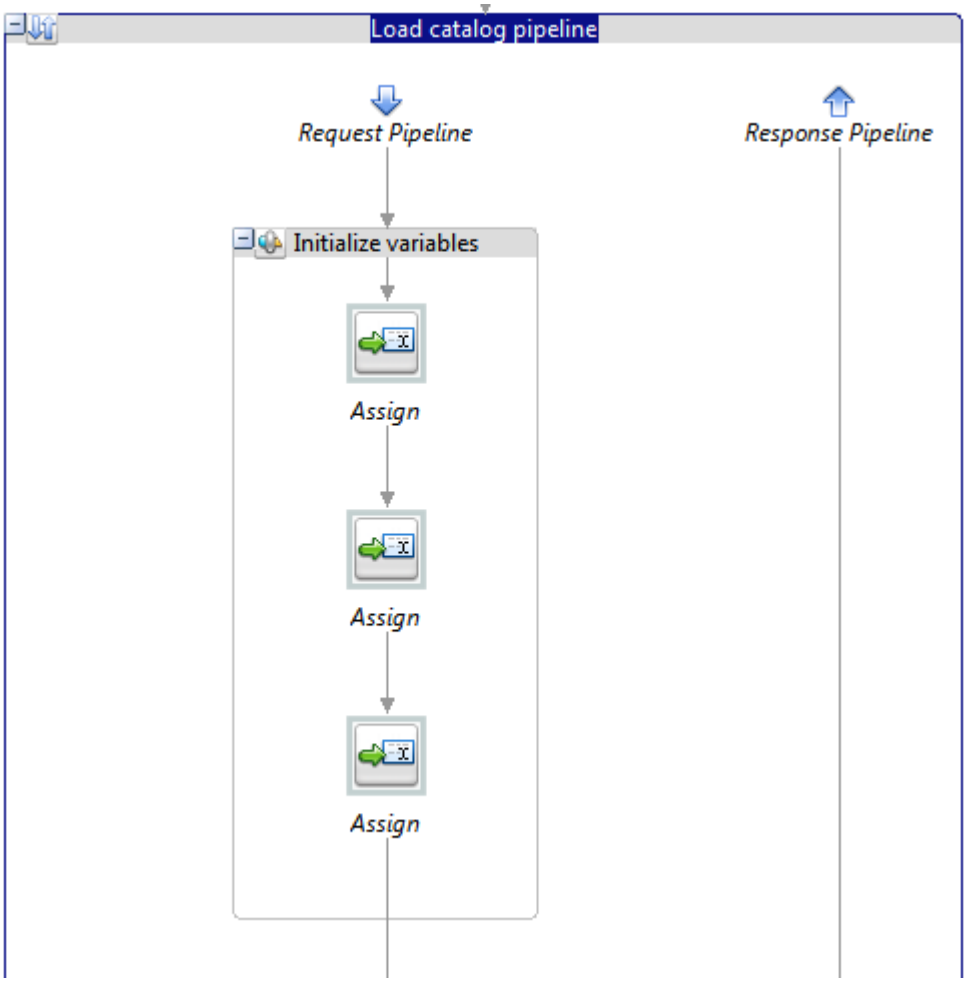


Рис. 5. Xslt-преобразование запроса на загрузку каталога на сервис-шине.

Затем из сервис-шины вызывается сервис RapidaCatalogRequest:

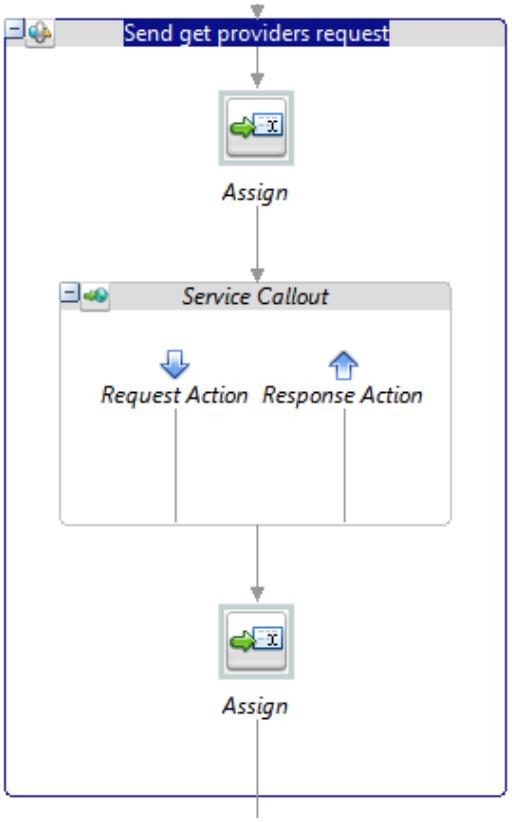


Рис.6. Запрос, вызывающий сервис загрузки каталога.

Сервис RapidaCatalogRequest в свою очередь находит в базе данных реквизиты данного поставщика, создает для него SSL-контекст и создает объект RapidaClient чтобы загрузить каталог:

```
final List bspConnections = BspConnection.findByBspGuidCppGuid(em, bspGuid,
cppGuid);
private String loadRapidaCatalog(BspConnection bspConnection){
        EntityManager em = emf.createEntityManager();

        RapidaClientObjectFactory rcof = new RapidaClientObjectFactory();

         SSLSocketFactory sslsf =
rcof.createRapidaSslSocketFactory(KEYSTORE_PATH, KEYSTORE_PASS,
TRUSTSTORE_PATH, TRUSTSTORE_PASS);

        RapidaClient rapidaClient =
rcof.createRapidaClient(bspConnection.getUri(),"003-10",
bspConnection.getSystemId(), sslsf, null);

        String rapidaResponse = rapidaClient.getFee(); //getRapidaCatalog,
dom parser

        return rapidaResponse;
```

Далее полученный каталог в формате ПС Рапида подвергается преобразованию для загрузки его в БД и загружается в БД сервисом LoadCatalog:

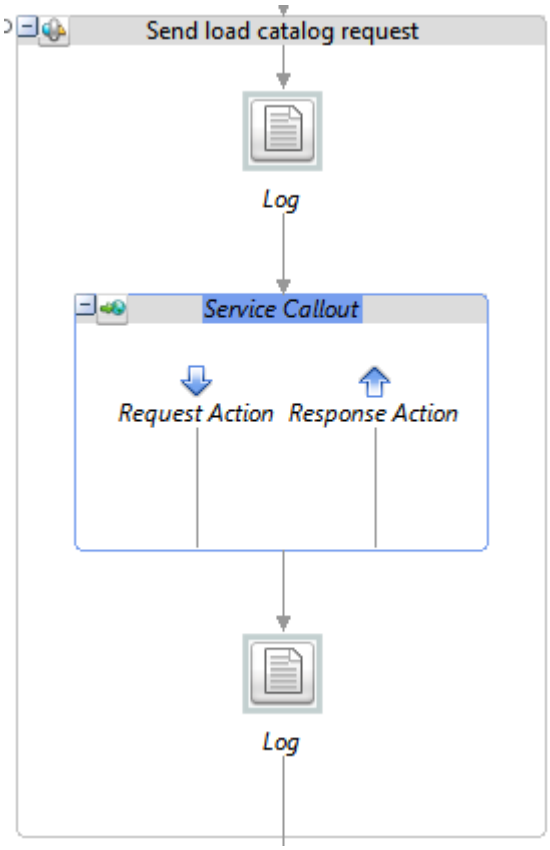


Рис. 7. Запрос сохранения в базу данных

5.3.2Реализация логики обработки платежа.

Прохождение платежа через сисему определяется веб-сервисом RapidaPaymentService, который является входной точкой для системы ДБО. Адрес WSDL-описания данного сервиса указывается в настройках системы дистанционного банковского обслуживания и исходящий запрос с меткой PAYMENT уходит на данный веб-сервис.

Обработка платежа в данном проекте сделана с использованием технологии EJB 3.0 в отличие от предыдущих подобных шлюзов, где маршрутизация и преобразование запроса выполнялось на сервис-шине. Это сделано с целью повышения быстродействия и надежности системы.

Веб-сервис является Stateless EJB компонентой. Это подразумевает создание данной компоненты при обращении к ней и не сохранение состояния при разрыве сессии. В качестве передаваемых в нее внешних компонентов используются jms-очереди на сервере приложений (которые будут обрабатывать дальнейшие запросы), а также connectionFactory – также ресурс, определенный на сервере Weblogic:

```
    @Resource(name = "jmsConnectionFactory")
    private ConnectionFactory jmsConnectionFactory;
    @Resource(name = "confirmPaymentInputQueue")
    private Destination confirmPaymentInputQueue;
```

Сначала веб-сервис проверяет пришедший к нему запрос по статусу платежа: в случае нулевого статуса (запрос только что создан), он отправляется на проверку методом checkPayment:

```
if(PaymentState.valueOf(requestPaymentStatus).equals(PaymentState.NEW)){
                        JaxbRapidaResponse rapidaResponse =
(JaxbRapidaResponse)checkPayment(pdInitial, bspConnection);

                        if ((RapidaPaymentStatus.valueOf(rapidaResponse,
"check").equals(RapidaPaymentStatus.CHECK_IN_PROGRESS)||(RapidaPaymentStatu
s.valueOf(rapidaResponse, "check").equals(RapidaPaymentStatus.CHECKED)))){

        pmsgrq.getPaymentDetail().setStatus((BigInteger.valueOf(1)));
                         PaymentMsgRs paymentMsgRs =
this.createPaymentMsgRs(pmsgrq.getMsgHdr(), pdInitial, null);
                         log.info("Incoming request has been processed
successfully");
                         return
this.createHandlePaymentResponse(paymentMsgRs, null);
                        }
```

Запрос checkPayment вызывает RapidaPaymentClient с параметрами проверки платежа и возвращает ответ Rapida, преобразованный к объекту. Далее, в зависимости от полей в ответе он либо отправляется дальше на проведение, либо выводится сообщение об ошибке:

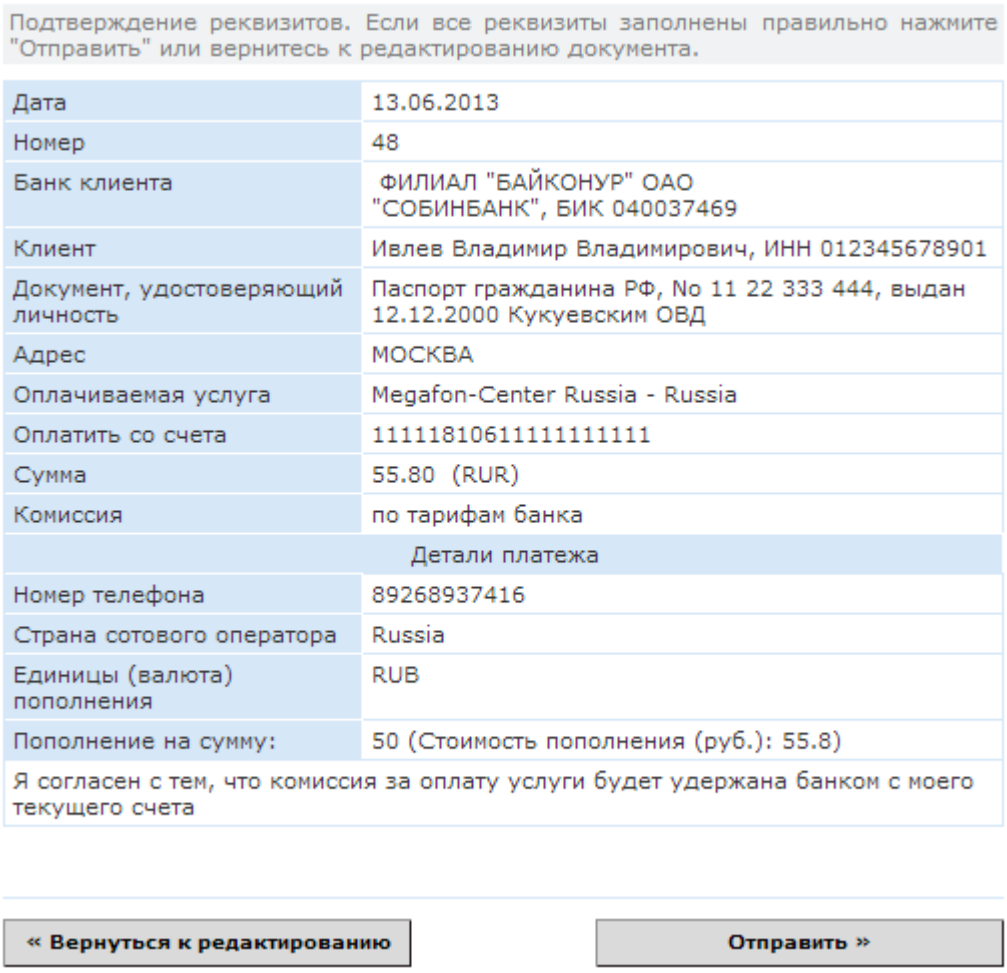
```
               else {

                         log.error("Failed to process the incoming check
request");
                         em = this.emf.createEntityManager();
                         String errorTextForUser =
rapidaResponse.getDescription();
                         em.close();

pmsgrq.getPaymentDetail().setStatus((BigInteger.valueOf(5)));
```

```
                        ErrorType error = this.createErrorType(null,
errorTextForUser);
                        PaymentMsgRs pmsgrs =
this.createPaymentMsgRs(pmsgrq.getMsgHdr(), pdInitial, error);
                                return
this.createHandlePaymentResponse(pmsgrs, null);
```

При успешной проверке платежа в возвращаемом ответе меняется статус платежа на 1 (платеж проверен), и далее возвращается в систему ДБО. Клиенту в это время показывается окно с введенными реквизитами и предложением подтвердить оплату:


Подтверждение реквизитов. Если все реквизиты заполнены правильно нажмите "Отправить" или вернитесь к редактированию документа.

Дата	13.06.2013
Номер	48
Банк клиента	ФИЛИАЛ "БАЙКОНУР" ОАО "СОБИНБАНК", БИК 040037469
Клиент	Ивлев Владимир Владимирович, ИНН 012345678901
Документ, удостоверяющий личность	Паспорт гражданина РФ, No 11 22 333 444, выдан 12.12.2000 Кукуевским ОВД
Адрес	МОСКВА
Оплачиваемая услуга	Megafon-Center Russia - Russia
Оплатить со счета	11111810611111111111
Сумма	55.80 (RUR)
Комиссия	по тарифам банка
Детали платежа	
Номер телефона	89268937416
Страна сотового оператора	Russia
Единицы (валюта) пополнения	RUB
Пополнение на сумму:	50 (Стоимость пополнения (руб.): 55.8)
Я согласен с тем, что комиссия за оплату услуги будет удержана банком с моего текущего счета	

« Вернуться к редактированию Отправить »

Рис. 8. Подтверждение оплаты.

Далее при нажатии кнопки «Подтвердить» платеж отправляется опять в модуль обработки и теперь, при статусе 1 (проверен), отправляется на выполнение:

```
if(PaymentState.valueOf(requestPaymentStatus).equals(PaymentState.CHECKED))
{
```

```
                    userTransaction.begin();
                this.executePayment(parameters.getPaymentMsgRq());
                userTransaction.commit();
```

Данная операция выполняется в транзакции.

Метод executePayment выполняет сохранение запроса со статусом 1 в базу данных, перевод платежа из статуса 1 в 2 (в обработке), и затем отправляет в очередь сообщений `confirmPaymentInputQueue` на сервере приложений:

```
private void executePayment(PaymentMsgRq paymentMsgRq) throws Exception{

        EntityManager em = emf.createEntityManager();
        PaymentBean paymentBean = new PaymentBean();
        paymentBean.acceptPayment(paymentMsgRq, em);

paymentMsgRq.getPaymentDetail().setStatus(BigInteger.valueOf(PaymentState.P
ROCESSING.getCode()));
        paymentBean.changePaymentStatus(paymentMsgRq, null, em);
        JAXBContext jaxbContext =
JAXBContext.newInstance(com.bssys.xsd.ebpp.PaymentMsgRq.class);
        Marshaller marshaller = jaxbContext.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, false);
        marshaller.setProperty(Marshaller.JAXB_ENCODING, ENCODING);
        ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
        marshaller.marshal(paymentMsgRq, byteArrayOutputStream);

JmsUtils.sendMsgToDestination(byteArrayOutputStream.toString(ENCODING),
this.jmsConnectionFactory, this.confirmPaymentInputQueue);
        em.close();
```

Теперь платеж находится в асинхронной обработке. Из очереди сообщений периодически (раз в 120 секунд отправляется запрос в Рапиду на проведение платежа). В случае ответа от ПС «В обработке» платеж повторно отправляется, до тех пор пока не будет получен финальный статус. Статусы 3 и 5 являются финальными, и означают:

3 – платеж проведен;

5 – платеж отвергнут по какой-то причине:

```
                if
(RapidaPaymentStatus.CONFIRM_PAYMENT.equals(rapidaPaymentStatus)){
                    log.info("Payment has been executed, changing payment
status...");
                    PaymentBean paymentBean = new PaymentBean();

paymentMsgRq.getPaymentDetail().setStatus(BigInteger.valueOf(PaymentState.E
XECUTED.getCode()));
```

```
                    EntityManager entityManager =
this.emf.createEntityManager();
                    paymentBean.changePaymentStatus(paymentMsgRq, null,
entityManager);
                    entityManager.close();
                    log.info("Payment status has been changed");


                } else
if(RapidaPaymentStatus.PAYMENT_IN_PROGRESS.equals(rapidaPaymentStatus)){
                    log.info("Payment is still being executed by Rapida.
Forcing the associated transaction to rollback");
                    context.setRollbackOnly();
                } else if
(RapidaPaymentStatus.EXECUTION_ERROR.equals(rapidaPaymentStatus)){
                    log.warn("Payment has been rejected by OSMP, changing
payment status...");
                    EntityManager entityManager =
this.emf.createEntityManager();
                    PaymentBean paymentBean = new PaymentBean();
                    paymentMsgRq.getPaymentDetail().setStatus(
BigInteger.valueOf(PaymentState.NOT_ACCEPTED.getCode()) );
                    paymentBean.changePaymentStatus(paymentMsgRq, null,
entityManager);
                    entityManager.close();
                }
```

Указанная логика выполняется в модуле PaymentBean. Данный модуль является MessageDriven компонентом, который включается в работу при появлении сообщения в очереди Weblogic. Связь данного модуля и внешнего ресурса(очереди) выполняется через настройки, прописанные в weblogic-ejb-jar.xml – файле конфигурации приложения:

```
    <!-- Rapida adapter beans -->

    <weblogic-enterprise-bean>
    <ejb-name>RapidaPaymentService</ejb-name>
    <stateless-session-descriptor></stateless-session-descriptor>
    <resource-description>
        <res-ref-name>jmsConnectionFactory</res-ref-name>
        <jndi-name>bss.ebpp.XAConnectionFactory</jndi-name>
    </resource-description>
    <resource-env-description>
        <resource-env-ref-name>confirmPaymentInputQueue</resource-env-ref-
name>
        <jndi-name>bss.ebpp.rapida.ConfirmPaymentInputQueue</jndi-name>
    </resource-env-description>
    </weblogic-enterprise-bean>
    <weblogic-enterprise-bean>
        <ejb-name>RapidaPaymentConfirmationBean</ejb-name>
        <message-driven-descriptor>
            <connection-factory-jndi-
name>bss.ebpp.XAConnectionFactory</connection-factory-jndi-name>
            <destination-jndi-
name>bss.ebpp.rapida.ConfirmPaymentInputQueue</destination-jndi-name>
        </message-driven-descriptor>
```

```
        <transaction-descriptor>
            <trans-timeout-seconds>120</trans-timeout-seconds>
        </transaction-descriptor>

        <resource-description>
            <res-ref-name>jmsConnectionFactory</res-ref-name>
            <jndi-name>bss.ebpp.XAConnectionFactory</jndi-name>
        </resource-description>
    </weblogic-enterprise-bean>
```

Согласно протоколу ПС «Рапида» и договоренности с программистами, ответственными за их модули, проверка платежа выполняется раз в 120 секунд. Предельное количество запросов к платежной системе составляет 10. По истечении этого количества запросов платежу присваивается статус 5 (Отказан платежной системой), и сохраняется в базу данных. В базу данных также сохраняются все платежи с финальными статусами 3 и 5. Далее по ним выполняется составление реестров для поставщиков услуг и банков.

6 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

6.1 Сравнение альтернативных решений

Основной задачей данного дипломного проекта было создание надежного, отказоустойчивого модуля, с адекватным временем разработки и улучшенным быстродействием. Таким образом в начале проектирования данной системы были исследованы несколько альтернативных подходов к созданию данного модуля, и выбран оптимальный по критерию улучшения производительности и быстродействия.

Первое из альтернативных решений предполагало использовать сервис-шину OracleServiceBus для маршрутизации запроса, а обработку платежа выполнять вызовом клиента к сервису, но на низком уровне.

Корпоративная сервисная шина (EnterpriseServiceBus) или интеграционная шина является одним из ключевых элементов сервис-ориентированной архитектуры (SOA). Шина предоставляет инфраструктуру для объединения существующих приложений (независимо от их платформы) и позволяет организовать взаимодействие между системами на основе сервисов, включая их публикацию, объединение и использование. Это дает возможность получить большую отдачу от существующих систем (и инвестиций в них) и минимизировать стоимость интеграции новых приложений в существующую инфраструктуру. Обладая средствами подключения, как к стандартным системам, так и к приложениям собственной разработки, а также инструментами интеграции с другими источниками данных, корпоративная сервисная шина играет центральную роль в любой SOA-стратегии[1].

Данный подход реализован в ранее спроектированных модулях и успешно работает при небольшой загрузке и малом количестве платежей:

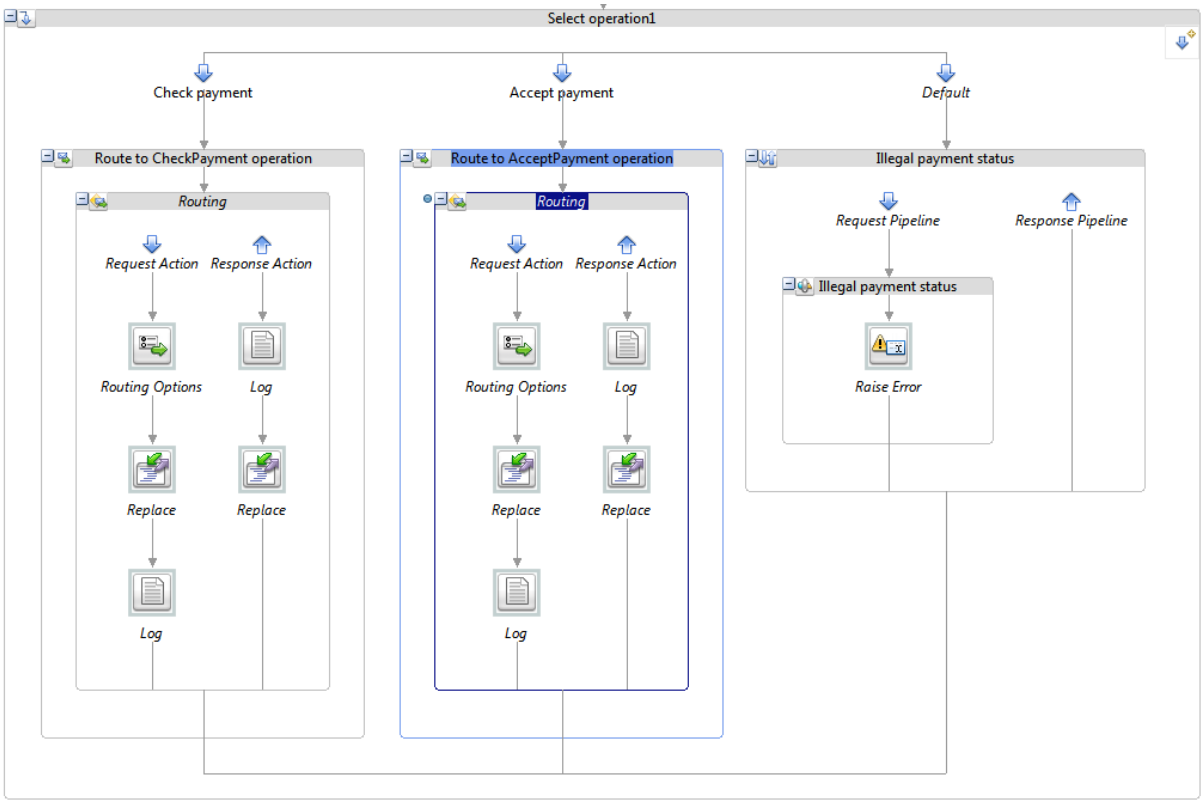


Рис.9. Прокси-сервис обработки платежа на сервис-шине.

К преимуществам разработки на основе сервис-шины можно отнести быстроту проектирования сервисов, богатый набор адаптеров к различным протоколам, полная поддержка SOAP- и REST- Web-сервисов, расширяемость решения и ускоренное внедрение веб-сервисов.

Указанный подход легок в разработке и относительно прост для понимания, а время разработки шлюза при таком подходе значительно сокращается. Подход с использованием сервис-шины был реализован на этапе стартапа проекта для ускорения разработки и пилотных платежей.

При данном подходе есть несколько неудачных моментов:

1. При сложной обработке запроса данный прокси-сервис также приобретает очень сложную структуру. Это отражается на последующем сопровождении кода и исправлении ошибок.
2. Транзакционность можно обеспечить лишь с момента отправки запроса к внешнему веб-сервису. До этого момента если произошло какое-то непредвиденное событие, то произойдет ошибка, но

транзакция не откатится. Это приводит к тому что требуется дополнительно осуществлять проверки, делающие код еще более сложным для сопровождения и рефакторинга.

3. Сервис-шина как любая надстройка повышает сложность системы и как следствие время обработки запроса, а также понижает отказоустойчивость. Время обработки – ключевой критерий при масштабируемости системы, поэтому при дальнейшей разработке необходимо было найти методы для повышения производительности.

Таким образом, архитектура системы до изменения выглядела так:

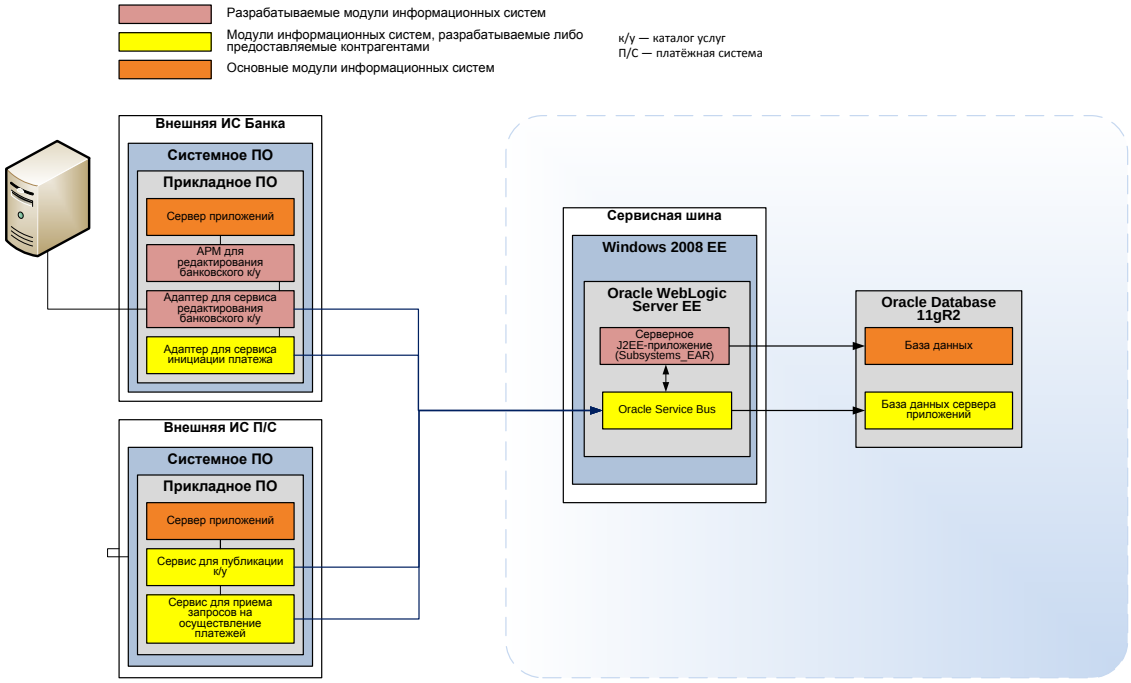


Рис. 10. Общая архитектура системы до изменений.

Второе альтернативное решение было использовать технологию EJB 3.0. Данная спецификация является стандартом разработки Java EE приложений, и успешно используется во многих распределенных веб-приложениях. Основными плюсами данной технологии являются:

1. Надежность и хорошая документированность стандарта.
2. Возможность работы приложения, построенного на стандарте EJB 3.0 не только на сервере Oracle Weblogic, но и на любом другом,

включая OpenSource решение TomCat. Это важное преимущество, позволяющее не иметь проблем в дальнейшем при замене сервера приложений и масштабируемости системы.

3. Возможность работы в транзакции, контролируемой самим сервером приложений. Все, что требуется от разработчика при таком подходе – обеспечить класс, отвечающий за обработку запроса необходимыми аннотациями, которые говорят серверу о необходимости выполнять данный запрос в транзакции:

```
4. @Stateless
5. @TransactionManagement(TransactionManagementType.BEAN)
6. @WebService
7. public class RapidaPaymentService implements PaymentServicePort
```

Также у данного подхода существует несколько недостатков:

1. Повышенное время разработки вследствие увеличения количества кода, требуемого для выполнения действия (в отличие от OSB, которая ориентирована на веб-разработку)
2. Необходимость конфигурирования и настройки файлов разворачивания приложения, синхронизация ресурсов с сервером приложений, что приводит к более сложной и долгой его настройке.

6.2 Основной критерий сравнения

Как два основных критерия по сравнению были выбраны:

1. Быстродействие системы.
2. Надежность системы.

Исходя из этих критериев, было принято решение строить новые модули системы на основании технологии EJB 3.0, с последующим сравнением производительности решения. Таким образом, предлагаемая структура системы с использованием нового подхода имеет вид:

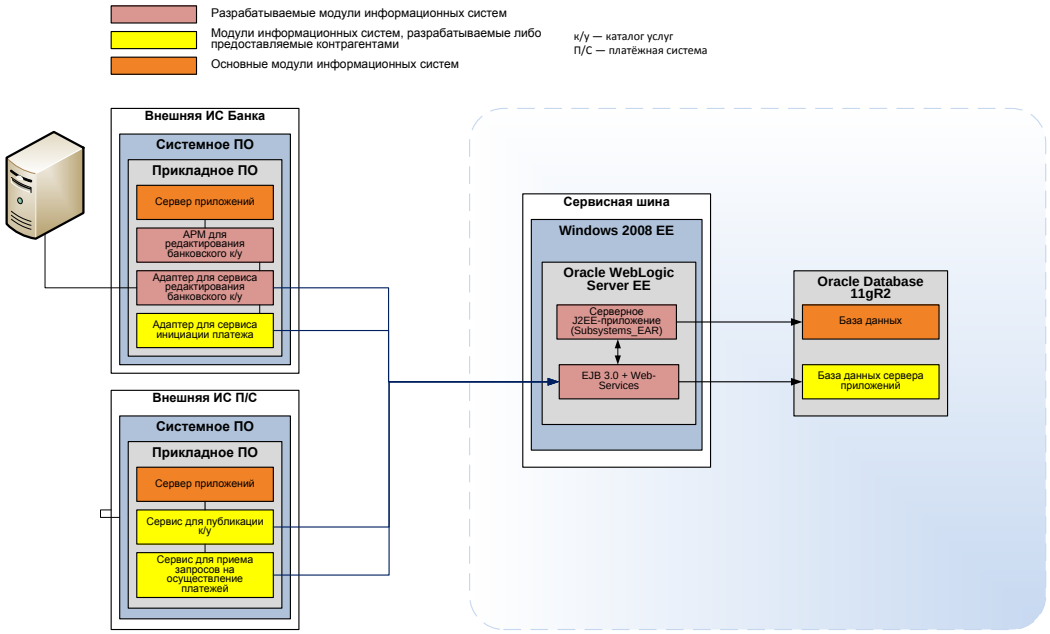


Рис. 11. Общая структура системы после перестроения

Проект, описанный здесь, выполнялся согласно последнему подходу. При этом было произведено сравнение 2-х архитектур по производительности и надежности.

6.3 Тестирование подходов и сравнение производительности

Производительность приложения до и после рефакторинга измерялась средствами Weblogic Diagnostic Framework. Указанный модуль входит в стандартную конфигурацию сервера приложений и предназначен для сбора и оценки информации и различных метрик для оценки производительности сервера и приложений. Модуль имеет большое количество расширений, в данной работе использовались инструменты для управления и мониторинга приложениями.

Тестовое окружение:

Для тестирования на одной из машин был развернут сервер приложений, в конфигурации, включающей Oracle Service Bus. На указанном сервере был создан единственный домен testDomain (см. рисунок 3) и развернуты 2 EAR-модуля (с применением сервис-шины, и

без нее) Каждый EAR-архив в свою состоял из нескольких модулей, но точкой входа являлось веб-приложение, доступное по определенному URL. С помощью JMeter (open-source средство автоматизированного тестирования приложений сервисно-ориентированной архитектуры) с 3 разных машин в сети был отправлен ряд запросов на оба приложения. Ниже, на рисунках, показаны результаты, полученные с помощью WLDF.

Результаты были сделаны на основании увеличения FreeJavaHeapSpace – количества свободной памяти при работе java-машины на сервере osb34 (рис. 3), уменьшении количества активных потоков, обрабатываемых сервером (рис. 4), а также увеличении числа обработанных запросов за единицу времени (рис. 5):

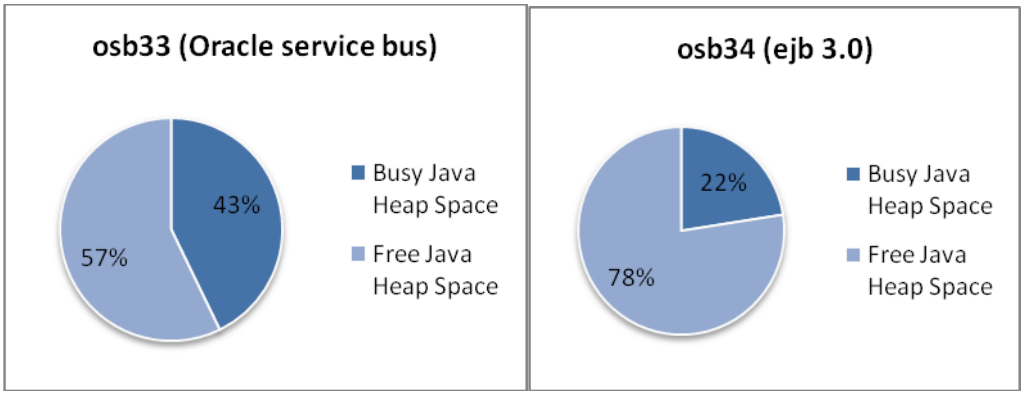


Рис. 12. Сравнение по размеру FreeJavaHeapSpace

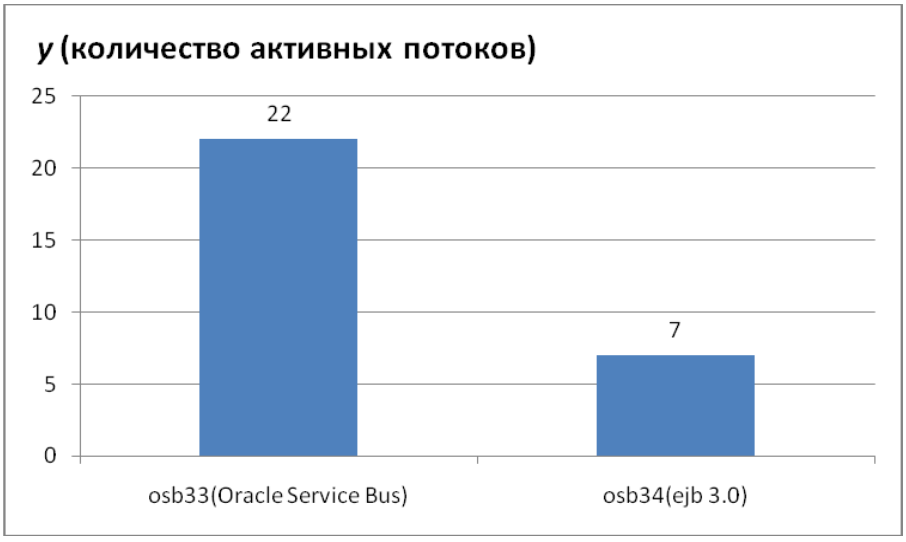


Рис. 13. Сравнение по количеству потоков

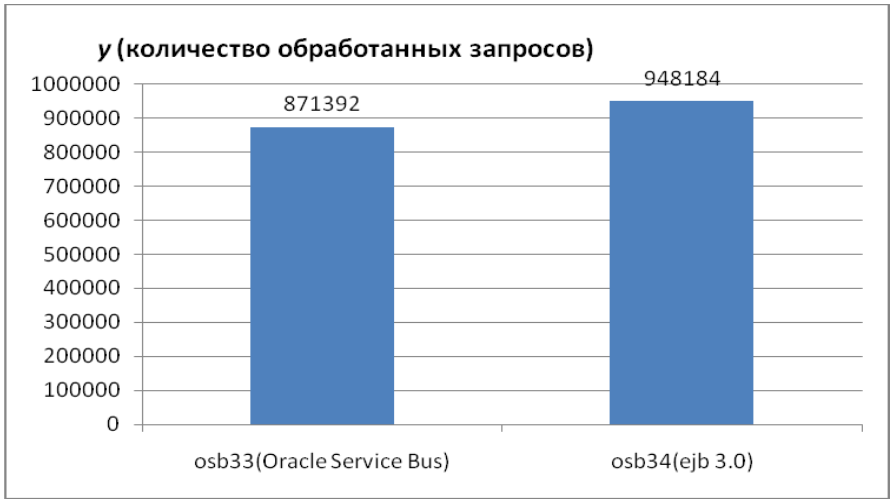


Рис. 14. Сравнение по количеству обработанных запросов

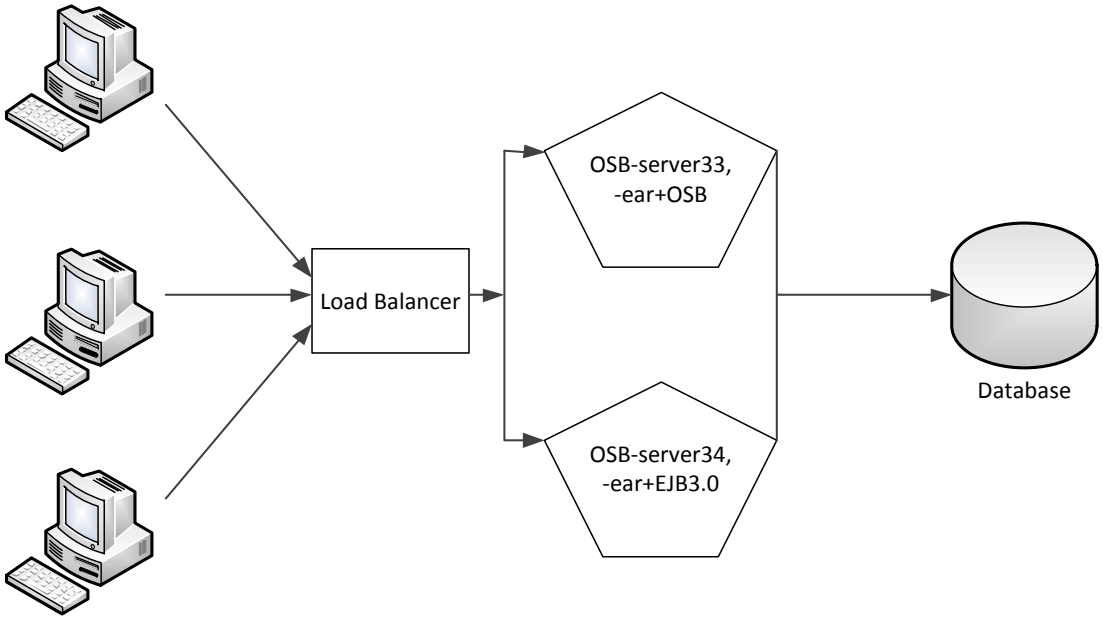


Рис. 15. Тестирование.

Результаты исследования показали, что применительно к данной задаче технология ejb 3.0 показывает лучшие результаты по сравнению с использованием сервис-шины.

Тестирование производительности показало снижение нагрузки на сервер и уменьшение времени обработки запроса на сервере 34, где был развернут модуль приложения с применением EJB 3.0. По масштабируемости и сопровождению кода данное решение проиграло сервис-шине, что объясняется тем, что изначально OSB более

ориентирована на интеграцию нескольких веб-сервисов. Для сравнения можно рассмотреть то, что разработка веб-сервиса на OSB в среднем требует не таких значительных усилий, как разработка веб-сервиса используя технологию ejb 3.0.

6.4 Выводы по исследовательскому этапу

Анализ показал, что применительно к данной задаче по нескольким критериям технология EJB 3.0 показывает лучшие результаты чем применение сервис-шины. Результаты работы были задействованы в рефакторинге существующей системы и в построении модуля, описанного в данном дипломном проекте.

ЗАКЛЮЧЕНИЕ

В рамках данного дипломного проекта были получены результаты:

Проведено предпроектное исследование и сравнение различных архитектур построения распределенных приложений и методов интеграции выбрано оптимальное решение применительно к области банковской автоматизации. На основе данного исследования был разработан модуль взаимодействия с платежной системой, соединяющий 2 подхода к интеграции приложений.

На этапе концептуального проектирования системы с помощью диаграммы компонентов нотации UML укрупненно показано внутреннее общее устройство системы.

На этапе технического проектирования разработана архитектура подсистемы, в соответствии с ней разработана диаграмма классов UML проектируемой подсистемы, а также диаграммы последовательности обработки запроса и диаграмма состояний подсистемы.

На этапе рабочего проектирования написан программный код для реализации спроектированных раннее решений и архитектуры компонентов модуля интеграции с платежной системой Рапида. Проведены отладка и тестирование новой системы, в ходе которых исправлялись найденные ошибки. Работоспособность системы была проверена на тестовой среде и в настоящее время введена в эксплуатацию.

В исследовательской части проводилось сравнение полученного решения с существующими по критериям отказоустойчивости и производительности, а также проведено тестирование и сбор данных на основании которых делался вывод о производительности системы.

Таким образом, поставленная цель дипломного проекта достигнута в полном объеме.

СПИСОК ЛИТЕРАТУРЫ

1. Граничин О.Н., Кияев В.И. Информационные технологии в управлении БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий — ИНТУИТ.ру, 2008

2. Грегор Хоп, Бобби Вульф Шаблоны интеграции корпоративных приложений Вильямс, 2006.: 627с.

3. Балабанова И. Т. Банки и банковская деятельность. - СПб.: Питер, 2008.

4. Положение от 03.10.2002 г. № 2-П «О безналичных расчетах в РФ».

5. Карпов Л.Е.Архитектура распределенных систем программного обеспечения. – Москва: МАКС Пресс , 2007.:130с.

6. Guido Schmutz, Edwin Biemond, Jan van Zoggel and MischaKöllikerOracle Service Bus 11g Development Cookbook –UK:Olton Birmingham,2012.-506с.

7. Rima Patel Sriganesh, Gerald Brose, Micah Silverman Mastering Enterprise JavaBeans 3.0 – USA:Wiley Publishing, Inc.,2006.-685с.

8. The Java EE 5 Tutorial [Электронныйресурс] / Eric Jendrock, Jennifer Ball, Debbie Carson, Ian Evans, Scott Fordin, Kim Haase, 2010. – Режимдоступа: http://docs.oracle.com/javaee/5/tutorial/doc/bncnt.html, свободный.

ПРИЛОЖЕНИЕ 1. Листинг класса-клиента к платежной системе «Рапида».

```
8. package com.bssys.ebpp.rapida.client.impl;
9.
10.
11.  import java.io.BufferedInputStream;
12.  import java.io.ByteArrayInputStream;
13.  import java.io.File;
14.  import java.io.FileInputStream;
15.  import java.io.IOException;
16.  import java.io.InputStream;
17.  import java.io.OutputStreamWriter;
18.  import java.io.UnsupportedEncodingException;
19.  import java.net.HttpURLConnection;
20.  import java.net.MalformedURLException;
21.  import java.net.Proxy;
22.  import java.net.URL;
23.  import java.net.URLConnection;
24.  import java.net.URLEncoder;
25.  import java.nio.charset.Charset;
26.  import java.security.KeyStore;
27.  import java.security.SecureRandom;
28.  import java.text.DecimalFormat;
29.  import java.text.NumberFormat;
30.  import java.util.Iterator;
31.  import java.util.LinkedHashMap;
32.  import java.util.List;
33.  import java.util.Map;
34.  import java.util.prefs.Preferences;
35.
36.  import javax.net.ssl.HttpsURLConnection;
37.  import javax.net.ssl.KeyManager;
38.  import javax.net.ssl.KeyManagerFactory;
39.  import javax.net.ssl.SSLContext;
40.  import javax.net.ssl.SSLSocketFactory;
41.  import javax.net.ssl.TrustManager;
42.  import javax.net.ssl.TrustManagerFactory;
43.  import javax.xml.bind.JAXBContext;
44.  import javax.xml.bind.Unmarshaller;
45.  import javax.xml.parsers.DocumentBuilder;
46.  import javax.xml.parsers.DocumentBuilderFactory;
47.
48.  import org.apache.log4j.Logger;
49.  import org.w3c.dom.Document;
50.  import org.xml.sax.InputSource;
51.  import org.xml.sax.SAXException;
52.
53.  import com.bssys.ebpp.rapida.client.model.RapidaClient;
54.  import com.bssys.ebpp.rapida.client.model.RapidaResponse;
55.  import
   com.bssys.ebpp.rapida.client.transformation.JaxbRapidaResponse;
56.
57.  /**
58.   *
59.   * @author YadYV
60.   * Client to connect with Rapida
```

```
61.   *
62.   */
63.
64.  class DefaultRapidaClient implements RapidaClient {
65.
66.                 private static final String KEYSTORE_PATH =
   "C:/certificates/rapidaJKS.jks";
67.             private static final String TARGET_HTTPS_SERVER =
   "https://gate.rapida.ru/test";   // "https://gate.rapida.ru/test";
68.             private static final int    TARGET_HTTPS_PORT   = 443;
69.
70.
71.             private static final Logger log =
   Logger.getLogger(DefaultRapidaClient.class);
72.
73.             private final SSLSocketFactory sslsf;
74.             private final Proxy proxy;
75.
76.             private final String url;
77.             private final String termType;
78.             private final String termId;
79.
80.             private static final String RESULT_OK                =
   "OK";
81.
82.             private static final String RESULT_ERROR             =
   "Error";
83.
84.             private static final String FUNCTION_CHECK           =
   "check";
85.
86.             private static final String FUNCTION_PAYMENT         =
   "payment";
87.
88.             private static final String FUNCTION_CANCEL          =
   "cancelreq";
89.
90.             private static final String FUNCTION_GET_PAYM_SUBJ   =
   "getpaymsubj";
91.
92.             private static final String FUNCTION_GET_BALANCE     =
   "getbalance";
93.
94.             private static final String FUNCTION_GET_FEE         =
   "getfee";
95.
96.             private static final String FUNCTION_GET_STATUS      =
   "getstatus";
97.
98.             public class RapidaPaymentHelper {
99.
100.                public JaxbRapidaResponse
   getRapidaResponseJaxb(Document doc) throws Exception{
101.                    JaxbRapidaResponse rapidaResponse = null;
102.                    String transformationResult;
103.
104.                 try{
105.                     JAXBContext context =
   JAXBContext.newInstance(JaxbRapidaResponse.class);
```

```
106.                        Unmarshaller unm =
   context.createUnmarshaller();
107.                        rapidaResponse =
   (JaxbRapidaResponse)unm.unmarshal(doc);
108.                    } catch(Exception ex){
109.                        throw new Exception("Failed to parse Rapida
   response", ex);
110.                    }
111.                                return rapidaResponse;
112.                           }
113.            }
114.
115.
116.            /**
117.             * Method send check request to Rapida
118.             */
119.            public JaxbRapidaResponse check(String paymExtId, int
   paymSubjTp, List<String> params, int amount, int feeSum) throws
   Exception{
120.                return sendRapidaRequest(FUNCTION_CHECK, paymExtId,
   paymSubjTp, params, amount, feeSum);
121.            }
122.            /**
123.             * Methodm execute payment
124.             */
125.            public JaxbRapidaResponse payment(String paymExtId, int
   paymSubjTp, List<String> params, int amount, int feeSum) throws
   Exception{
126.                return sendRapidaRequest(FUNCTION_PAYMENT,
   paymExtId, paymSubjTp, params, amount, feeSum);
127.            }
128.
129.            /**
130.             * Send request to Rapida
131.             * @param paymExtId
132.             * @param paymSubjTp
133.             * @param params
134.             * @param amount
135.             * @param feeSum
136.             * @return JaxbRapidaResponse
137.             * @throws Exception
138.             */
139.            private JaxbRapidaResponse sendRapidaRequest(String
   functionName, String paymExtId, int paymSubjTp, List<String> params,
   int amount, int feeSum) throws Exception{
140.                NumberFormat moneyFormat = new DecimalFormat("000");
141.
142.                Map<String, String> reqParams = new
   LinkedHashMap<String, String>();
143.                reqParams.put("function", functionName);
144.                reqParams.put("PaymExtId", paymExtId);
145.                reqParams.put("PaymSubjTp",
   String.valueOf(paymSubjTp));
146.                reqParams.put("Amount", moneyFormat.format(amount));
147.                String paramsStr = "";
148.                Iterator<String> i = params.iterator();
149.                while(i.hasNext()){
150.                    paramsStr += i.next();
151.                    if(i.hasNext()) paramsStr += ";";
152.                }
```

```
153.                reqParams.put("Params", paramsStr);
154.                reqParams.put("TermType", termType);
155.                reqParams.put("TermId", termId);
156.                reqParams.put("FeeSum", moneyFormat.format(feeSum));
157.                Request r = new Request(url,
   Request.HTTP_METHOD.GET, reqParams, sslsf, proxy) {
158.                    @Override
159.                    protected byte[] responseProcessor(byte[] data)
   {
160.                        return data;
161.                    }
162.                };
163.
164.                byte[] byteRapidaResponse = r.getRapidaResponse();
165.
166.                // Log rapida response
167.
168.                String s = new String(byteRapidaResponse, "windows-
   1251");
169.
170.                log.info("response from Rapida: " + s);
171.
172.            Document doc1;
173.            DocumentBuilder builder;
174.            DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
175.            JaxbRapidaResponse rapidaResponseJaxb = null;
176.            RapidaPaymentHelper rapidaPaymentHelper = new
   RapidaPaymentHelper();
177.                try{
178.                 builder = factory.newDocumentBuilder();
179.            }catch(Exception ex){
180.                 throw new RuntimeException(ex);
181.            }
182.            InputSource is = new InputSource( new
   ByteArrayInputStream( byteRapidaResponse ) );
183.            try {
184.                doc1 = builder.parse( is );
185. //                System.out.println("Message body: \n\t" + new
   String(byteRapidaResponse, Charset.forName("cp1251")).replace("\n",
   "\n\t"));
186.                rapidaResponseJaxb =
   rapidaPaymentHelper.getRapidaResponseJaxb(doc1);
187.                return rapidaResponseJaxb;
188.            } catch (SAXException ex) {
189.                System.out.println(rapidaResponseJaxb + "Полученные
   данные не соответствуют спецификации XML.");
190. //                System.out.println("Message body: \n\t" + new
   String(rapidaResponse, Charset.forName("cp1251")).replace("\n",
   "\n\t"));
191.            } catch (IOException ex) {
192.
193.                } catch (Exception ex){
194.            throw new Exception("Failed to send request to Rapida",
   ex);
195.        }
196.            return rapidaResponseJaxb;
197.            }
198.
199.            /**
```

```
200.              * Method get Catalog from Rapida
201.              */
202.             public String getFee(){
203.         Map<String, String> reqParams = new LinkedHashMap<String,
   String>();
204.         reqParams.put("function", FUNCTION_GET_FEE);
205.         Request r = new Request(url, Request.HTTP_METHOD.GET,
   reqParams, sslsf, proxy) {
206.             @Override
207.             protected byte[] responseProcessor(byte[] data) {
208.                 return data;
209.             }
210.         };
211.         byte[] fee = r.getRapidaResponse();
212.         Document doc = null;
213.         try{
214.             Document doc1;
215.             DocumentBuilder builder;
216.             DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
217.                try{
218.                 builder = factory.newDocumentBuilder();
219.             }catch(Exception ex){
220.                 throw new RuntimeException(ex);
221.             }
222.
223.         InputSource is = new InputSource( new ByteArrayInputStream(
   fee ) );
224.         try {
225.             doc1 = builder.parse( is );
226.             System.out.println("Тело ответа: \n\t" + new
   String(fee, Charset.forName("cp1251")).replace("\n", "\n\t"));
227.             String rapidaCatalogResponse = new String(fee,
   Charset.forName("cp1251"));
228.             return rapidaCatalogResponse;
229.         } catch (SAXException ex) {
230.             System.out.println("Полученные данные не соответствуют
   спецификации XML.");
231.             System.out.println("Тело ответа: \n\t" + new
   String(fee, Charset.forName("cp1251")).replace("\n", "\n\t"));
232.         } catch (IOException ex) {
233.
234.         } catch (Exception ex){
235.         throw new Exception("Failed to send request to Rapida",
   ex);
236.     }
237.     }
238.     catch(Exception ex){
239.         ex.printStackTrace();
240.     }
241.     return null;
242.     }
243.
244.
245.
246.             // comment because not used
247.
248. //          private RapidaResponse cancel(String paymExtId){
249. //              Map<String, String> reqParams = new
   LinkedHashMap<String, String>();
```

```
250. //              reqParams.put("function", FUNCTION_CANCEL);
251. //              reqParams.put("PaymExtId", paymExtId);
252. //
253. //              Request r = new Request(url,
   Request.HTTP_METHOD.GET, reqParams, sslsf, proxy) {
254. //                  @Override
255. //                  protected byte[] responseProcessor(byte[] data)
   {
256. //                      return data;
257. //                  }
258. //              };
259. //              return r.getRapidaResponse();
260. //          }
261.
262. //          private byte[] getPaymSubj(PAYM_SUBJ_TYPE
   paym_subj_type, String param){
263. //              Map<String, String> reqParams = new
   LinkedHashMap<String, String>();
264. //              reqParams.put("function", FUNCTION_GET_PAYM_SUBJ);
265. //              reqParams.put("Code", paym_subj_type.toString());
266. //              reqParams.put("params", param);
267. //              Request r = new Request(url,
   Request.HTTP_METHOD.GET, reqParams, sslsf, proxy) {
268. //                  @Override
269. //                  protected byte[] responseProcessor(byte[] data)
   {
270. //                      return data;
271. //                  }
272. //              };
273. //              return r.getRapidaResponse();
274. //          }
275. //
276. //          private byte[] getBalance(String paymExtId){
277. //              Map<String, String> reqParams = new
   LinkedHashMap<String, String>();
278. //              reqParams.put("function", FUNCTION_GET_BALANCE);
279. //              reqParams.put("PaymExtId", paymExtId);
280. //              Request r = new Request(url,
   Request.HTTP_METHOD.GET, reqParams, sslsf, proxy) {
281. //                  @Override
282. //                  protected byte[] responseProcessor(byte[] data)
   {
283. //                      return data;
284. //                  }
285. //              };
286. //              return r.getRapidaResponse();
287. //          }
288. //
289. //
290. //          private byte[] getStatus(String paymExtId){
291. //              Map<String, String> reqParams = new
   LinkedHashMap<String, String>();
292. //              reqParams.put("function", FUNCTION_GET_STATUS);
293. //              reqParams.put("PaymExtId", paymExtId);
294. //              Request r = new Request(url,
   Request.HTTP_METHOD.GET, reqParams, sslsf, proxy) {
295. //                  @Override
296. //                  protected byte[] responseProcessor(byte[] data)
   {
297. //                      return data;
```

```
298. //                  }
299. //              };
300. //              return r.getRapidaResponse();
301. //          }
302.
303.             protected DefaultRapidaClient(String url, String
   termType, String termId, SSLSocketFactory sslsf, Proxy proxy) {
304.                 this.url = url;
305.                 this.termType = termType;
306.                 this.termId = termId;
307.                 this.sslsf = sslsf;
308.                 this.proxy = proxy;
309.             }
310.
311.             public static SSLSocketFactory getFactory(File
   pkcs12Keystore, String passwordKeystore, File jksTruststore, String
   passwordTruststore){
312.                 try{
313.
314.
315.                         KeyManagerFactory   keyManagerFactory   =
   KeyManagerFactory.getInstance("SunX509");
316.                     TrustManagerFactory trustManagerFactory =
   TrustManagerFactory.getInstance("SunX509");
317.
318.                     KeyStore keyStore   =
   KeyStore.getInstance("PKCS12");
319.                     KeyStore trustStore =
   KeyStore.getInstance("jks");
320.
321.                     InputStream keyInput  = new
   FileInputStream(pkcs12Keystore);
322.                     keyStore.load(keyInput,
   passwordKeystore.toCharArray());
323.                     keyInput.close();
324.
325.                     InputStream certInput = new
   FileInputStream(jksTruststore);
326.                     trustStore.load(certInput,
   passwordTruststore.toCharArray());
327.                     certInput.close();
328.
329.                     keyManagerFactory.init(keyStore,
   passwordKeystore.toCharArray());
330.                     trustManagerFactory.init(trustStore);
331.
332.                     SSLContext context =
   SSLContext.getInstance("SSL");
333.
334.                     KeyManager[]   kms =
   keyManagerFactory.getKeyManagers();
335.                     TrustManager[] tms =
   trustManagerFactory.getTrustManagers();
336.                     context.init(kms, tms, new SecureRandom());
337.
338.                     return context.getSocketFactory();
339.                 }catch(Exception ex){
340.                     throw new RuntimeException(ex);
341.                 }
342.             }
```

```
343.
344.            public static Document respAlalyzer(byte[] resp){
345.                Document doc;
346.                DocumentBuilder builder;
347.                DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
348.                try{
349.                    builder = factory.newDocumentBuilder();
350.                }catch(Exception ex){
351.                    throw new RuntimeException(ex);
352.                }
353.
354.                String result="P PµP·CѓP»CЊC,P°C,: ";
355.                InputSource is = new InputSource( new
   ByteArrayInputStream( resp ) );
356.                try {
357.                    doc = builder.parse( is );
358.                } catch (SAXException ex) {
359.                    System.out.println(result +
   "PµPsP»CѓC‡PµPSPSC‹Pµ PґP°PSPSC‹Pµ PSPµ CЃPsPsC,PIPµC,CЃC,PICѓCЊC,
   CЃPїPµC†PёC„PёPєP°C†PёPё XML.");
360.                    System.out.println("PўPµP»Ps PsC,PIPµC,P°: \n\t"
   + new String(resp, Charset.forName("cp1251")).replace("\n", "\n\t"));
361.                    return null;
362.                } catch (IOException ex) {
363.                    throw new RuntimeException(ex);
364.                }
365.
366.                return doc;
367.                           }
368.
369.            public static abstract class Request{
370.                public static enum HTTP_METHOD{
371.                    POST,
372.                    GET
373.                }
374.
375.                private String url;
376.                private final HTTP_METHOD method;
377.                private final Map<String, String> params;
378.                private final SSLSocketFactory sslsf;
379.                private final Proxy proxy;
380.
381.                public Request(String url, HTTP_METHOD method,
   Map<String, String> params, SSLSocketFactory sslsf, Proxy proxy) {
382.                    this.url = url;
383.                    this.method = method;
384.                    this.params = params;
385.                    this.sslsf = sslsf;
386.                    this.proxy = proxy;
387.                }
388.
389.                public byte[] getRapidaResponse() {
390.                    String paramsStr = "";
391.
392.                    for(Map.Entry<String, String> el :
   params.entrySet()){
393.                        if(paramsStr.length() > 0) paramsStr += "&";
394.
395.                        try {
```

```
396.                             paramsStr +=
   URLEncoder.encode(el.getKey(), "UTF-8") + "="
397.                                       +
   URLEncoder.encode(el.getValue(), "UTF-8");
398.                         } catch (UnsupportedEncodingException ex) {
399.                             throw new RuntimeException(ex);
400.                         }
401.                     }
402.                     URLConnection conn;
403.                     URL _url;
404.                     try{
405.                         String urlStr = url;
406.                         if(method == HTTP_METHOD.GET){
407.                             urlStr += (urlStr.indexOf('?') == -1 ?
   "?": "&") + paramsStr;
408.                         }
409.                         _url = new URL(null, urlStr, new
   sun.net.www.protocol.https.Handler());
410.
411.                             log.info("RapidaRequest = " + urlStr);
412.
413.                         //                      _url = new URL(null,
   urlStr, new sun.net.www.protocol.http.Handler());
414.                     }catch(MalformedURLException ex){
415.                         throw new RuntimeException(ex);
416.                     }
417.
418.                     try {
419.                         conn =
   (proxy==null?_url.openConnection():_url.openConnection(proxy));
420.                         if(sslsf != null  && conn instanceof
   HttpsURLConnection)
421.
   ((HttpsURLConnection)conn).setSSLSocketFactory(sslsf);
422.                     } catch (Exception ex) {
423.                         throw new RuntimeException(ex);
424.                     }
425.
426.                     conn.setReadTimeout(100000);
427.                     conn.setConnectTimeout(100000);
428.
429.                     if(this.method.equals(HTTP_METHOD.POST)){
430.                        conn.setDoOutput(true);
431.                        OutputStreamWriter wr;
432.                             try {
433.                                 wr = new
   OutputStreamWriter(conn.getOutputStream());
434.                             } catch (IOException ex) {
435.                                 throw new RuntimeException(ex);
436.                             }
437.                             try {
438.                                 wr.flush();
439.                                 wr.close();
440.                             } catch (IOException ex) {
441.                                 throw new RuntimeException(ex);
442.                             }
443.                     }
444.
445.                     int code;
446.                     try {
```

```
447.                         if(conn instanceof HttpsURLConnection) code
   = ((HttpsURLConnection)conn).getResponseCode();
448.                         else if(conn instanceof HttpURLConnection)
   code = ((HttpURLConnection)conn).getResponseCode();
449.                         else throw new RuntimeException("Unsupported
   protocol.");
450.                     } catch (IOException ex) {
451.                         throw new RuntimeException(ex);
452.                     }
453.
454.                     if(code != HttpURLConnection.HTTP_OK)
455.                         throw new RuntimeException("When you request
   an exception occurs. HTTP Error code: "+String.valueOf(code));
456.
457.                     BufferedInputStream rd;
458.                     try {
459.                         rd = new
   BufferedInputStream(conn.getInputStream());
460.                     } catch (IOException ex) {
461.                         throw new RuntimeException(ex);
462.                     }
463.                     byte[] result = new byte[0];
464.                     byte[] buff = new byte[512];
465.                     int len;
466.                     try {
467.                         while((len = rd.read(buff)) != -1){
468.                             result = margeByteArray(result,
   result.length, buff, len);
469.                         }
470.                         rd.close();
471.                     } catch (IOException ex) {
472.                         throw new RuntimeException(ex);
473.                     }
474.                     return responseProcessor(result);
475.                 }
476.
477.                 private byte[] margeByteArray(byte[] arr1, int len1,
   byte[] arr2, int len2){
478.                     byte[] result = new byte[len1+len2];
479.                     System.arraycopy(arr1, 0, result, 0, len1);
480.                     System.arraycopy(arr2, 0, result, len1, len2);
481.                     return result;
482.                 }
483.
484.                 protected abstract byte[] responseProcessor(byte[]
   data);
485.             }
486.
487. /*
488.  *
489.  *
490.  *
491.  *
492.     private static final String ENCODING = "windows-1251";
493.     private static final String KEYSTORE_TYPE = "JKS";
494.     private static final String KEYSTORE_PATH =
   "C:/certificates/rapidaJKS.jks";
495.     private static final String TARGET_HTTPS_SERVER =
   "https://gate.rapida.ru/test";   // "https://gate.rapida.ru/test/";
496.     private static final int    TARGET_HTTPS_PORT   = 443;
```

```
497.
498.
499.     private static final Logger log =
   Logger.getLogger(DefaultRapidaClient.class);
500.
501.     public static SSLSocketFactory getFactory(File pkcs12, String
   password){
502.     try{
503.         KeyManagerFactory keyManagerFactory =
   KeyManagerFactory.getInstance("SunX509");
504.         KeyStore keyStore = KeyStore.getInstance("PKCS12");
505.
506.         InputStream keyInput = new FileInputStream(pkcs12);
507.         keyStore.load(keyInput, password.toCharArray());
508.         keyInput.close();
509.
510.         keyManagerFactory.init(keyStore, password.toCharArray());
511.
512.         SSLContext context = SSLContext.getInstance("SSL");
513.         TrustManager[] trustAllCerts = new TrustManager[] { new
   X509TrustManager() {
514.             public void checkClientTrusted(X509Certificate[] chain,
   String authType) throws CertificateException {}
515.
516.             public void checkServerTrusted(X509Certificate[] chain,
   String authType) throws CertificateException {}
517.
518.             public X509Certificate[] getAcceptedIssuers() { return
   new X509Certificate[] {}; }
519.
520.             public void checkClientTrusted(X509Certificate[] xcs,
   String string, String string1, String string2) throws
   CertificateException {}
521.
522.             public void checkServerTrusted(X509Certificate[] xcs,
   String string, String string1, String string2) throws
   CertificateException {}
523.         }};
524.
525.         KeyManager[] kms = keyManagerFactory.getKeyManagers();
526.         context.init(kms, trustAllCerts, new SecureRandom());
527.
528.         return context.getSocketFactory();
529.
530.     } catch(Exception ex){
531.         throw new RuntimeException(ex);
532.     }
533.     }
534. //    private SSLContext getSSLContext(String jksFilePath, String
   jksPwd) throws Exception{
535. //        KeyStore trustStore = KeyStore.getInstance("JKS");
536. //        FileInputStream fileInputStream = new FileInputStream(new
   File(jksFilePath));
537. //        char[] pass = {'1','1','1','1','1','1'};
538. //        trustStore.load(fileInputStream, pass);
539. //        fileInputStream.close();
540. //
541. //        // Initializing the keystore
542. //        KeyManagerFactory kmf =
   KeyManagerFactory.getInstance("SunX509");
```

```
543. //        kmf.init(trustStore, pass);
544. //
545. //        // Initializing the trustore
546. //        TrustManagerFactory tmf =
   TrustManagerFactory.getInstance("SunX509");
547. //        tmf.init(trustStore);
548. //
549. //        // Here comes the context
550. //        javax.net.ssl.SSLContext sslContext =
   javax.net.ssl.SSLContext.getInstance("SSL");
551. //        X509TrustManager tm = new X509TrustManager() {
552. //              public void checkClientTrusted(X509Certificate[]
   xcs, String string) throws CertificateException { }
553. //
554. //              public void checkServerTrusted(X509Certificate[]
   xcs, String string) throws CertificateException { }
555. //
556. //              public X509Certificate[] getAcceptedIssuers() {
557. //                  return null;
558. //              }
559. //          };
560. //        sslContext.init(kmf.getKeyManagers(), new
   TrustManager[]{tm}, null);
561. //
562. //        return sslContext;
563. //       }
564.
565.
566.
567.     public RapidaResponce getRapidaResponse(RapidaRequestInfo
   paymentInfo, RapidaClientConfig configParameters) throws Exception{
568.
569.        if(!this.verifyConfigParameters(configParameters)){
570.             throw new Exception("Required configuration parameters
   are missing");
571.         }
572.
573.         String paymentParametersString = "";
574.
575.         for(RapidaPaymentParameter paymentParameter :
   paymentInfo.getPaymentParameters() )
576.         {
577.                if (paymentParametersString.length() == 0){
578.                       paymentParametersString =
   paymentParameter.getName() + "+" + paymentParameter.getValue() + ';';
579.                }
580.                else{
581.                       paymentParametersString =
   paymentParametersString + paymentParameter.getName() + "+" +
   paymentParameter.getValue() + ';';
582.                }
583.         }
584.
585.         System.out.println(paymentParametersString);
586.
587.         String rapidaProtocolResponse;
588.
589.         try{
```

```
590.                rapidaProtocolResponse =
   this.getRequest(paymentInfo, configParameters,
   paymentParametersString);
591.         } catch(Exception ex){
592.             throw new Exception("Failed to send request to Rapida",
   ex);
593.         }
594.
595.         Document transformationResult;
596.
597.         JaxbResponseWrapper jaxbResponseWrapper = null;
598.         try{
599.                transformationResult =
   this.parseRapidaResponce(rapidaProtocolResponse);
600.             JAXBContext context =
   JAXBContext.newInstance(JaxbResponseWrapper.class,
   JaxbResponce.class, DefaultRapidaClientConfig.class);
601.             Unmarshaller unmarshaller =
   context.createUnmarshaller();
602.             jaxbResponseWrapper = (JaxbResponseWrapper)
   unmarshaller.unmarshal(transformationResult);
603.         } catch(Exception ex){
604.             throw new Exception("Failed to parse Rapida response",
   ex);
605.         }
606.
607.         if( jaxbResponseWrapper.getResponseResultCode() == 0 &&
   jaxbResponseWrapper.getActionResultCode() == 0 &&
   paymentInfo.getPaymExtId() ==
   String.valueOf(jaxbResponseWrapper.getJaxbRapidaResponce().getPaymExt
   Id())){
608.             return
   DefaultRapidaResponce.create(jaxbResponseWrapper.getJaxbRapidaResponc
   e());
609.         } else {
610.             throw new Exception("Erroneous response from Rapida");
611.         }
612.
613.     }
614.
615.     public boolean isPaymentCheckOrConfurm(RapidaResponce
   rapidaResponce)
616.     {
617.         boolean result = false;
618.         if (rapidaResponce.getPaymNumb() == "-1"){
619.                return true;
620.         }
621.         else{
622.         return result;
623.         }
624.     }
625.
626.     public RapidaRequestInfo getPaymentStatus(RapidaRequestInfo
   paymentInfo, RapidaClientConfig configParameters) throws Exception{
627.         if(!this.verifyConfigParameters(configParameters)){
628.             throw new Exception("Required configuration parameters
   are missing");
629.         }
630.
631.         return null;
```

```
632.     }
633.
634.     private boolean verifyConfigParameters(RapidaClientConfig
   configParameters){
635.         if(configParameters == null)
636.             return false;
637.
   if(StringUtils.isBlank(configParameters.getKeyStoreFilePath()))
638.             return false;
639. //
   if(StringUtils.isBlank(configParameters.getKeyStorePassPhrase()))
640. //            return false;
641.         if(StringUtils.isBlank(configParameters.getRequestUri()))
642.             return false;
643.         if(StringUtils.isBlank(configParameters.getTerminal()))
644.             return false;
645.         if(StringUtils.isBlank(configParameters.getTerminalType()))
646.                return false;
647.         return true;
648.     }
649.
650.     private String getRequest(RapidaRequestInfo requestContent,
   RapidaClientConfig clientConfig, String paramString) throws
   Exception{
651.
652.         String paramsStr = "";
653.         for(Map.Entry<String, String> el : params.entrySet()){
654.             if(paramsStr.length() > 0) paramsStr += "&";
655.
656.             try {
657.                 paramsStr += URLEncoder.encode(el.getKey(), "UTF-
   8") + "="
658.                              + URLEncoder.encode(el.getValue(), "UTF-
   8");
659.             } catch (UnsupportedEncodingException ex) {
660.                 throw new RuntimeException(ex);
661.             }
662.         }
663.
664.         URLConnection conn;
665.         URL _url;
666.         try{
667.             String urlStr = clientConfig.getRequestUri();
668.                 urlStr += (urlStr.indexOf('?') == -1 ? "?": "&") +
   paramsStr;
669.             }
670.             _url = new URL(urlStr);
671.         }catch(MalformedURLException ex){
672.             throw new RuntimeException(ex);
673.         }
674.
675.
676.
677.
678.
679.         SchemeRegistry schemeRegistry = new SchemeRegistry();
680.         schemeRegistry.register(httpsScheme);
681.
682.         ClientConnectionManager cm = new
   SingleClientConnManager(schemeRegistry);
```

```
683.  //     HttpClient httpClient = new DefaultHttpClient(cm);
684.
685.         HttpClient httpClient = new DefaultHttpClient(cm);
686.
687. //      httpGet.getParams().setParameter("function",
   requestContent.getFunction());
688. //      httpGet.getParams().setParameter("PaymExtId",
   requestContent.getPaymExtId());
689. //      httpGet.getParams().setParameter("PaymSubjTp",
   requestContent.getPaymSubjType());
690. //      httpGet.getParams().setParameter("Amount",
   requestContent.getAmount());
691. //      httpGet.getParams().setParameter("Params",
   requestContent.getParamString());
692. //      httpGet.getParams().setParameter("TermType",
   requestContent.getTermType());
693. //      httpGet.getParams().setParameter("TermID",
   requestContent.getTermId());
694. //      httpGet.getParams().setParameter("FeeSum",
   requestContent.getFeeSum());
695.         //Header[] hdr = httpGet.getAllHeaders();
696.
697.         List<NameValuePair> params1 = new
   LinkedList<NameValuePair>();
698.
699.         params.add(new BasicNameValuePair("function",
   requestContent.getFunction()));
700.         params.add(new BasicNameValuePair("PaymExtId",
   requestContent.getPaymExtId()));
701.         params.add(new BasicNameValuePair("PaymSubjTp",
   String.valueOf(requestContent.getPaymSubjType())));
702.         params.add(new BasicNameValuePair("Amount",
   String.valueOf(requestContent.getAmount())));
703.         params.add(new BasicNameValuePair("Params",
   requestContent.getParamString()));
704.         params.add(new BasicNameValuePair("TermType",
   requestContent.getTermType()));
705.         params.add(new BasicNameValuePair("TermID",
   requestContent.getTermId()));
706.         params.add(new BasicNameValuePair("FeeSum",
   String.valueOf(requestContent.getFeeSum())));
707.
708.
709.         String param1String = URLEncodedUtils.format(params,
   "windows-1251");
710.         String url = TARGET_HTTPS_SERVER + '?' + param1String;
711.         HttpGet httpGet = new HttpGet(url);
712.
713. //      URI uri = URIUtils.createURI("https", TARGET_HTTPS_SERVER,
   443, "/test/", URLEncodedUtils.format(params, "windows-1251"), null);
714. //      URLEncodedUtils.format(params, "windows-1251");
715.
716.
717.         for(Header hdr : httpGet.getAllHeaders()){
718.                 String strHeader = hdr.toString();
719.                 System.out.println(strHeader);
720.         }
721.         String param1 = httpGet.getParams().toString();
722.         System.out.println(param1);
723.         String rqstLine = httpGet.getRequestLine().toString();
```

```
724.        System.out.println(rqstLine);
725.
726.
727.        HttpResponse httpResponse = httpClient.execute(httpGet);
728.                int httpStatusCode =
   httpResponse.getStatusLine().getStatusCode();
729.                if(httpStatusCode != HttpStatus.SC_OK){
730.                        throw new Exception(String.format("Http
   response status code: %s", httpStatusCode));
731.                }
732.                HttpEntity entity = httpResponse.getEntity();
733.
734.                if (entity == null)
735.                        throw new Exception("...");
736.
737.                StringBuilder sb = new StringBuilder();
738.                InputStream inputStream = null;
739.                try {
740.                        inputStream = entity.getContent();
741.                    InputStreamReader inputStreamReader = new
   InputStreamReader(inputStream);
742.                    BufferedReader bufferedReader = new
   BufferedReader(inputStreamReader);
743.                    String line;
744.                    while ((line = bufferedReader.readLine()) !=
   null) {
745.                        sb.append(line);
746.                    }
747.                } finally {
748.                        if(inputStream != null){
749.                                try{
750.                                        inputStream.close();
751.                         } catch(IOException ignore){}
752.                        }
753.                }
754.
755.                String responseBody = sb.toString();
756.
757.        if(responseBody.isEmpty() == true ){
758.            throw new Exception(String.format("cannot parse empty
   object"));
759.         }
760.         return responseBody;
761.     }
762.
763.     public Document parseRapidaResponce(String
   rapidaProtocolResponce) throws Exception
764.     {
765.        DocumentBuilderFactory dbf =
   DocumentBuilderFactory.newInstance();
766.        Document doc = null;
767.        try{
768.                DocumentBuilder db = dbf.newDocumentBuilder();
769.                InputSource is = new InputSource();
770.                is.setCharacterStream(new
   StringReader(rapidaProtocolResponce));
771.
772.                doc   = db.parse(is);
773.
774.                return doc;
```

```
775.                 }
776.         catch (Exception e){
777.                 e.printStackTrace();
778.         }
779.         finally{
780.                 return doc;
781.         }
782.     }
783. */
784.
785.
786. }
```

ПРИЛОЖЕНИЕ 2. Листинг класса веб-сервиса, осуществляющего загрузку каталога услуг

```
787. package com.bssys.ebpp.catalog.rapida;
788.
789. import java.util.List;
790.
791. import javax.ejb.Stateless;
792. import javax.ejb.TransactionManagement;
793. import javax.ejb.TransactionManagementType;
794. import javax.jws.WebMethod;
795. import javax.jws.WebParam;
796. import javax.jws.WebResult;
797. import javax.jws.WebService;
798. import javax.net.ssl.SSLSocketFactory;
799. import javax.persistence.EntityManager;
800. import javax.persistence.EntityManagerFactory;
801. import javax.persistence.PersistenceUnit;
802.
803. import com.bssys.ebpp.error.ErrorCodes;
804. import com.bssys.xsd.ebpp.ErrorType;
805. import org.apache.log4j.Logger;
806.
807. import com.bssys.ebpp.model.BspConnection;
808. import
   com.bssys.ebpp.payment.adapter.rapida.impl.RapidaPaymentService;
809. import com.bssys.ebpp.rapida.client.impl.RapidaClientObjectFactory;
810. import com.bssys.ebpp.rapida.client.model.RapidaClient;
811.
812. /**
813.  * Session Bean implementation class RapidaCatalogRequest
814.  */
815. @Stateless
816. @TransactionManagement(TransactionManagementType.BEAN)
817. @WebService
818. public class RapidaCatalogRequest{
819.
820.     @PersistenceUnit(unitName = "EBPP_DataModel")
821.     private EntityManagerFactory emf;
822.
823.     private static final Logger log =
   Logger.getLogger(RapidaPaymentService.class);
824.
825.     private static final String ENCODING = "windows-1251";
826.
827.     private final String KEYSTORE_PATH = "C:\\Certificate.p12";
828.     private final String KEYSTORE_PASS = "111111";
829.     private final String TRUSTSTORE_PATH =
   "C:\\rapidaTruststore.jks";
830.     private final String TRUSTSTORE_PASS = "111111";
831.
832.     public RapidaCatalogRequest() {}
833.
834.     @WebMethod(operationName = "LoadCatalog")
```

```
835.     @WebResult(name = "rapidaCatalogResponse", targetNamespace =
   "http://www.bssys.com/ebpp/RapidaCatalogService/", partName =
   "parameters")
836.     public String loadCatalogRequest(@WebParam(name = "host")
   String host,
837.
   @WebParam(name = "systemType") String st,
838.
   @WebParam(name = "systemId") String sysId,
839.
   @WebParam(name = "bspGuid") String bspGuid,
840.
   @WebParam(name = "cppGuid") String cppGuid) {
841.
842.
843.         EntityManager em = emf.createEntityManager();
844.         String rapidaCatalogResponse = null;
845.
846.         final List bspConnections =
   BspConnection.findByBspGuidCppGuid(em, bspGuid, cppGuid);
847.          if(bspConnections.size() < 1){
848.              ErrorType error =
   this.createErrorType(ErrorCodes.EBPP0010.value(),
   com.bssys.ebpp.error.ErrorMsg.getErrorDesc(ErrorCodes.EBPP0010.value(
   )));
849.              return error.toString();
850.          }
851.          BspConnection bspConnection =
   (BspConnection)bspConnections.get(0);
852.
853.          String cat = this.loadRapidaCatalog(bspConnection);
854.
855.         return cat;
856.     }
857.
858.     private String loadRapidaCatalog(BspConnection bspConnection){
859.         EntityManager em = emf.createEntityManager();
860.
861.         RapidaClientObjectFactory rcof = new
   RapidaClientObjectFactory();
862.
863.          SSLSocketFactory sslsf =
   rcof.createRapidaSslSocketFactory(KEYSTORE_PATH, KEYSTORE_PASS,
   TRUSTSTORE_PATH, TRUSTSTORE_PASS);
864.
865.          RapidaClient rapidaClient =
   rcof.createRapidaClient(bspConnection.getUri(),"003-10",
   bspConnection.getSystemId(), sslsf, null);
866.
867.          String rapidaResponse = rapidaClient.getFee();
   //getRapidaCatalog, dom parser
868.
869.          return rapidaResponse;
870.     }
871.
872.     private ErrorType createErrorType(String errorCode, String
   errorDesc){
873.         ErrorType errorType = new ErrorType();
874.         errorType.setCode(errorCode);
875.         errorType.setErrorMessage(errorDesc);
```

```
876.         return errorType;
877.     }
878. }
```

ПРИЛОЖЕНИЕ 3. Листинг класса, осуществляющего обработку платежей

```
package com.bssys.ebpp.payment.adapter.rapida.impl;

import java.io.ByteArrayOutputStream;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;

import javax.annotation.Resource;
import javax.ejb.Stateless;
import javax.ejb.TransactionManagement;
import javax.ejb.TransactionManagementType;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.net.ssl.SSLSocketFactory;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.PersistenceUnit;
import javax.persistence.Query;
import javax.transaction.UserTransaction;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;

import com.bssys.ebpp.paymentservice.*;
import org.apache.commons.lang3.StringUtils;
import org.apache.log4j.Logger;

import com.bssys.ebpp.common.ParamValueType;
import com.bssys.ebpp.common.ParameterType;
import com.bssys.ebpp.common.ParametersType;
import com.bssys.ebpp.common.SimpleParameterType;
import com.bssys.ebpp.error.ErrorCodes;
import com.bssys.ebpp.error.ErrorMsg;
import com.bssys.ebpp.model.BsProvider;
import com.bssys.ebpp.model.BspConnection;
import com.bssys.ebpp.model.Payment;
import com.bssys.ebpp.model.Service;
import com.bssys.ebpp.payment.adapter.rapida.model.PaymentState;
import com.bssys.ebpp.payment.core.PaymentBean;
import com.bssys.ebpp.payment.core.payment.ErrorUtils;
import com.bssys.ebpp.paymentinfo.SettlementDocIdentificationType;
import com.bssys.ebpp.rapida.client.impl.RapidaClientObjectFactory;
import com.bssys.ebpp.rapida.client.model.RapidaClient;
import com.bssys.ebpp.rapida.client.model.RapidaPaymentStatus;
import com.bssys.ebpp.rapida.client.model.RapidaResponse;
import com.bssys.ebpp.rapida.client.transformation.JaxbRapidaResponse;
import com.bssys.ebpp.utils.CurrencyUtils;
import com.bssys.ebpp.utils.JmsUtils;
import com.bssys.xsd.ebpp.ErrorType;
```

```
import com.bssys.xsd.ebpp.MsgHdrType;
import com.bssys.xsd.ebpp.PaymentDetail;
import com.bssys.xsd.ebpp.PaymentMsgRq;
import com.bssys.xsd.ebpp.PaymentMsgRs;


@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
@WebService
public class RapidaPaymentService implements PaymentServicePort{

    @Resource
    private UserTransaction userTransaction;
//    @PersistenceContext(unitName = "EBPP_DataModel")
//    private EntityManagerFactory emf;

    @PersistenceUnit(unitName = "EBPP_DataModel")
    private EntityManagerFactory emf;

    @Resource(name = "jmsConnectionFactory")
    private ConnectionFactory jmsConnectionFactory;
    @Resource(name = "confirmPaymentInputQueue")
    private Destination confirmPaymentInputQueue;

    private static final String QUERY_NEXT_PAYMENT_ID = "select
RAPIDA_PMNT_ID_SEQ.nextval from dual";

    private static final Logger log =
Logger.getLogger(RapidaPaymentService.class);

    private static final String ENCODING = "windows-1251";

    private final String KEYSTORE_PATH = "C:\\Certificate.p12";
    private final String KEYSTORE_PASS = "111111";
    private final String TRUSTSTORE_PATH = "C:\\rapidaTruststore.jks";
    private final String TRUSTSTORE_PASS = "111111";

    public RapidaPaymentService() {}


    @Override
    @WebMethod(operationName = "HandlePayment1", action =
"http://www.bssys.com/ebpp/PaymentService/HandlePayment1")
    @WebResult(name = "HandlePaymentResponse", targetNamespace =
"http://www.bssys.com/ebpp/PaymentService/", partName = "parameters")
    public HandlePaymentResponse handlePayment(@WebParam(name =
"HandlePaymentRequest", targetNamespace =
"http://www.bssys.com/ebpp/PaymentService/", partName = "parameters")
HandlePaymentRequest parameters) {

        try{

                PaymentMsgRq pmsgrq = parameters.getPaymentMsgRq();
            PaymentDetail pdInitial =
parameters.getPaymentMsgRq().getPaymentDetail();
            MsgHdrType requestMsgHeader =
parameters.getPaymentMsgRq().getMsgHdr();

            String requestSenderId =
parameters.getPaymentMsgRq().getMsgHdr().getSender().getSenderId();
```

```
                EntityManager em = emf.createEntityManager();
                SettlementDocIdentificationType pid =
pdInitial.getPaymentIdentification();

        pdInitial.getPaymentIdentification().setCPPDocId(this.getNextPaymen
tId(em));
            BsProvider bsp = Service.findBspBySrvCodeAndCppEbppId(em,
pdInitial.getServiceCode(), requestSenderId);
            final List bspConnections = BspConnection.find(em,
bsp.getGuid(), requestSenderId);
            if(bspConnections.size() < 1){
                ErrorType error =
this.createErrorType(ErrorCodes.EBPP0010.value(),
ErrorMsg.getErrorDesc(ErrorCodes.EBPP0010.value()));
                return this.createHandlePaymentResponse(null, error);
            }
            BspConnection bspConnection = (BspConnection)
bspConnections.get(0);
            log.info("bspConnection inicialize succesfully");

            String bik =
pdInitial.getPaymentIdentification().getDrawer().getBIK();
            String settlementDocGUID =
pdInitial.getPaymentIdentification().getSettlementDocGUID();
            if(StringUtils.isEmpty(bik) ||
StringUtils.isEmpty(settlementDocGUID)){
                ErrorType error =
this.createErrorType(ErrorCodes.EBPP0007.value(),
ErrorMsg.getErrorDesc(ErrorCodes.EBPP0007.value()));
                return this.createHandlePaymentResponse(null, error);
            }

//            RapidaCatalogRequest rcr = new RapidaCatalogRequest();
//            String rapidaPaymentResponse =
rcr.loadCatalogRequest(bspConnection);
//
//            System.out.println(rapidaPaymentResponse);
            Payment existingPayment = Payment.find(em, bik,
settlementDocGUID);

            //Get existing payment state
            PaymentState existingPaymentState;
            if( existingPayment == null ){
                existingPaymentState = PaymentState.NEW;
            } else{
                existingPaymentState = PaymentState.valueOf(
existingPayment.getStatus() );
            }
            em.close();
            //Get payment status declared in the incoming request
            int requestPaymentStatus = pdInitial.getStatus().intValue();


if(PaymentState.valueOf(requestPaymentStatus).equals(PaymentState.NEW)){
                        JaxbRapidaResponse rapidaResponse =
(JaxbRapidaResponse)checkPayment(pdInitial, bspConnection);
```

```
                        if ((RapidaPaymentStatus.valueOf(rapidaResponse,
"check").equals(RapidaPaymentStatus.CHECK_IN_PROGRESS)||(RapidaPaymentStatu
s.valueOf(rapidaResponse, "check").equals(RapidaPaymentStatus.CHECKED)))){

        pmsgrq.getPaymentDetail().setStatus((BigInteger.valueOf(1)));
                         PaymentMsgRs paymentMsgRs =
this.createPaymentMsgRs(pmsgrq.getMsgHdr(), pdInitial, null);
                         log.info("Incoming request has been processed
successfully");
                         return
this.createHandlePaymentResponse(paymentMsgRs, null);
                        }
                        else {

                         log.error("Failed to process the incoming check
request");
                         em = this.emf.createEntityManager();
                         String errorTextForUser =
rapidaResponse.getDescription();
                         em.close();

pmsgrq.getPaymentDetail().setStatus((BigInteger.valueOf(5)));
                         ErrorType error = this.createErrorType(null,
errorTextForUser);
                         PaymentMsgRs pmsgrs =
this.createPaymentMsgRs(pmsgrq.getMsgHdr(), pdInitial, error);
                                 return
this.createHandlePaymentResponse(pmsgrs, null);
                        }
                }
                else
if(PaymentState.valueOf(requestPaymentStatus).equals(PaymentState.CHECKED))
{
                        userTransaction.begin();
                    this.executePayment(parameters.getPaymentMsgRq());
                    userTransaction.commit();
                    log.info("Incoming request has been processed
successfully");
                    PaymentMsgRs paymentMsgRs =
this.createPaymentMsgRs(pmsgrq.getMsgHdr(), pdInitial, null);
                    return this.createHandlePaymentResponse(paymentMsgRs,
null);

        //||(RapidaPaymentStatus.CHECK_IN_PROGRESS.equals(RapidaPaymentStat
us.valueOf(rapidaResponse, "check")))){
                //
        pdInitial.setStatus(BigInteger.valueOf(PaymentState.CHECKED.getCode
()));
                    }
                 else {

                    ErrorType error =
this.createErrorType(ErrorCodes.EBPP0010.value(),
ErrorMsg.getErrorDesc(ErrorCodes.EBPP0007.value()));
                    return this.createHandlePaymentResponse(null, error);
                }

        } catch (Exception ex){
```

```
            ErrorType error =
this.createErrorType(ErrorCodes.EBPP0010.value(),
ErrorMsg.getErrorDesc(ErrorCodes.EBPP0010.value()));
            return this.createHandlePaymentResponse(null, error);
        }
    }

    @Override
    @WebMethod(operationName = "CheckPayment", action =
"http://www.bssys.com/ebpp/PaymentService/CheckPayment")
    @WebResult(name = "CheckPaymentResponse", targetNamespace =
"http://www.bssys.com/ebpp/PaymentService/", partName = "parameters")
    public CheckPaymentResponse checkPayment(@WebParam(name =
"CheckPaymentRequest", targetNamespace =
"http://www.bssys.com/ebpp/PaymentService/", partName = "parameters")
CheckPaymentRequest parameters) {
        throw new UnsupportedOperationException("Requested operation is not
supported");
    }

    @Override
    @WebMethod(operationName = "AcceptPayment", action =
"http://www.bssys.com/ebpp/PaymentService/AcceptPayment")
    @WebResult(name = "AcceptPaymentResponse", targetNamespace =
"http://www.bssys.com/ebpp/PaymentService/", partName = "parameters")
    public AcceptPaymentResponse acceptPayment(@WebParam(name =
"AcceptPaymentRequest", targetNamespace =
"http://www.bssys.com/ebpp/PaymentService/", partName = "parameters")
AcceptPaymentRequest parameters) {
        throw new UnsupportedOperationException("Requested operation is not
supported");
    }

    private String makePaymExtId(){
        EntityManager em = emf.createEntityManager();
        String id = String.valueOf(this.getNextPaymentId(em));
        return id;
    }


    /**
    * Method load Rapida catalog.
    *
    * @return response
    * */
    private RapidaResponse checkPayment(PaymentDetail paymentDetail,
BspConnection bspConnection) throws Exception{
        boolean result = false;

        List<String> params;
        params = new ArrayList<String>();
        String paymExtId = makePaymExtId();

        EntityManager em = emf.createEntityManager();

        BigDecimal paymentAmount =
CurrencyUtils.toMajorUnits(paymentDetail.getAmount().getValue(),
paymentDetail.getAmount().getExponent().intValue());
```

```
        Service service = Service.findBySrvCode(em,
paymentDetail.getServiceCode());

        ParametersType pmntParamsInitial =
paymentDetail.getPaymentParameters();

        List<String> listRapidaParameter = new ArrayList();

        for(ParameterType parameter :
pmntParamsInitial.getSimpleParameterOrComplexParameter()){
            if(parameter instanceof SimpleParameterType){
                SimpleParameterType simpleParameter = (SimpleParameterType)
parameter;
                String parameterName = simpleParameter.getName();
                List<ParamValueType> parameterValues =
simpleParameter.getValue();
                String parameterValue = parameterValues.get(0).getData();
                listRapidaParameter.add(parameterName + " " +
parameterValue);
            } else{
                throw new IllegalArgumentException("Unsupported parameter
type");
            }
        }

        RapidaClientObjectFactory rcof = new RapidaClientObjectFactory();
        SSLSocketFactory sslsf =
rcof.createRapidaSslSocketFactory(KEYSTORE_PATH, KEYSTORE_PASS,
TRUSTSTORE_PATH, TRUSTSTORE_PASS);

        RapidaClient rapidaClient =
rcof.createRapidaClient(bspConnection.getUri(),"003-10",
bspConnection.getTerminalId(), sslsf, null);
        RapidaResponse rapidaResponse = rapidaClient.check(paymExtId,
Integer.parseInt(service.getBspSrvCode()), listRapidaParameter,
(int)paymentDetail.getAmount().getValue(), 0);
//Integer.parseInt(service.getBspSrvCode()) - instead of 101
        RapidaPaymentStatus rapidaPaymentStatus =
RapidaPaymentStatus.valueOf(rapidaResponse, "check");

        if(RapidaPaymentStatus.CHECKED.equals(rapidaPaymentStatus)){
               result = true;

        }
        else
if(RapidaPaymentStatus.CHECK_IN_PROGRESS.equals(rapidaPaymentStatus)){
               result = true;
        }
        else
if(RapidaPaymentStatus.EXECUTION_ERROR.equals(rapidaPaymentStatus)){

               result = false;
        } else{
            //TODO: handle
            throw new Exception("Rapida error");
        }
        return rapidaResponse;
    }
```

```
    private void persistPaymentAfterCheck (RapidaResponse
responsePaymentInfo, HandlePaymentRequest parameters) throws Exception{


        PaymentMsgRq  paymentMsgRq  = new PaymentMsgRq();
        MsgHdrType    msgHdr        =
parameters.getPaymentMsgRq().getMsgHdr();
        PaymentDetail paymentDetail =
parameters.getPaymentMsgRq().getPaymentDetail();

        RapidaPaymentStatus rapidaPaymentStatus =
RapidaPaymentStatus.valueOf(responsePaymentInfo, "check");
        BigInteger status =
BigInteger.valueOf(rapidaPaymentStatus.ordinal());

        paymentDetail.setStatus(status);
        paymentMsgRq.setMsgHdr(msgHdr);

        EntityManager em1     = emf.createEntityManager();

        PaymentBean   payment = new PaymentBean();
        try{
                payment.acceptPayment(paymentMsgRq, em1);
        }
        catch (Exception ex) {
                        throw new Exception("Failed to process incoming
accept payment request" , ex);
                }
        em1.close();
    }
    private void executePayment(PaymentMsgRq paymentMsgRq) throws
Exception{

        EntityManager em = emf.createEntityManager();
         PaymentBean paymentBean = new PaymentBean();
         paymentBean.acceptPayment(paymentMsgRq, em);

paymentMsgRq.getPaymentDetail().setStatus(BigInteger.valueOf(PaymentState.P
ROCESSING.getCode()));
         paymentBean.changePaymentStatus(paymentMsgRq, null, em);
         JAXBContext jaxbContext =
JAXBContext.newInstance(com.bssys.xsd.ebpp.PaymentMsgRq.class);
         Marshaller marshaller = jaxbContext.createMarshaller();
         marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, false);
         marshaller.setProperty(Marshaller.JAXB_ENCODING, ENCODING);
         ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
         marshaller.marshal(paymentMsgRq, byteArrayOutputStream);

JmsUtils.sendMsgToDestination(byteArrayOutputStream.toString(ENCODING),
this.jmsConnectionFactory, this.confirmPaymentInputQueue);
         em.close();
    }

    private HandlePaymentResponse createHandlePaymentResponse(PaymentMsgRs
paymentMsgRs, ErrorType error){
         HandlePaymentResponse handlePaymentResponse = new
HandlePaymentResponse();
         if(paymentMsgRs != null){
             handlePaymentResponse.setPaymentMsgRs(paymentMsgRs);
```

```
        } else {
            handlePaymentResponse.setError(error);
        }
        return handlePaymentResponse;
    }

    private String getNextPaymentId(EntityManager em){
        Query query = em.createNativeQuery(QUERY_NEXT_PAYMENT_ID);
        String paymentId1 = String.valueOf(query.getSingleResult());
        String paymentId2 = "RPD";
        String paymentId = paymentId2+paymentId1;
        return paymentId;
    }
    private ErrorType createErrorType(String errorCode, String errorDesc){
        ErrorType errorType = new ErrorType();

        errorType.setCode("-1");

        errorType.setStatus(BigInteger.valueOf(1));
        errorType.setErrorMessage(errorDesc);
        return errorType;
    }
    private PaymentMsgRs createPaymentMsgRs(MsgHdrType msgHeader,
PaymentDetail paymentDetail, ErrorType error){
        PaymentMsgRs paymentMsgRs = new PaymentMsgRs();
        paymentMsgRs.setMsgHdr(msgHeader);
        if(paymentDetail != null)
            paymentMsgRs.setPaymentDetail(paymentDetail);
        if(error != null)
            paymentMsgRs.setError(error);
        return paymentMsgRs;
    }
  }
```