электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.051

ПОСТРОЕНИЕ РАСПРЕДЕЛЕННОГО WEB-ПРИЛОЖЕНИЯ И СРАВНЕНИЕ 2-Х АРХИТЕКТУР

(С ПРИМЕНЕНИЕМ СЕРВИС-ШИНЫ ORACLESERVICEBUS И ТЕХНОЛОГИИ EJB 3.0)

Ядгарова Ю.В., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Компьютерные системы автоматизации производства»

Научный руководитель: Урусов А.В.,старший преподаватель
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
gss@bmstu.ru

Введение

Веб-приложения в настоящее время могут решать самые разные задачи. Архитектура SOA позволяет создавать на базе веб-сервисов сложные и распределенные схемы взаимодействия. В рассматриваемом примере существующее приложение служит интеграционным сервисом между ИС кредитной организации (банка) и различными платежными системами. Цель данной связки – позволить клиенту оплачивать счета и услуги в различных ПС из «Личного кабинета» банка, в котором он обслуживается. Специфика системы в том, что протокол ИС банка является неизменяемым, а по мере интеграции с различными платежными сервисами динамически добавляются модули-связки.

Общая архитектура системы

Архитектура системы – классическая, 3-х уровневая на базе инструментов OracleSOASuite.

Обращение в базу данных происходит каждый раз при проведении платежа, или синхронизации каталога услуг. Специфика работы с платежными системами предполагает в основном 2-4 типа запросов, включающие проверку платежа, обработку дополнительной информации, возвращаемой из ПС, проведение платежа и запрос статуса. Некоторые из этих запросов могут быть синхронными (то есть выполняться с ожиданием ответа от противоположной стороны), некоторые – асинхронными (без ожидания ответа). К примеру, периодический запрос статуса платежа является асинхронным, и при его отправке приложение не ждет ответа сразу, а переключается на обработку более приоритетных задач.

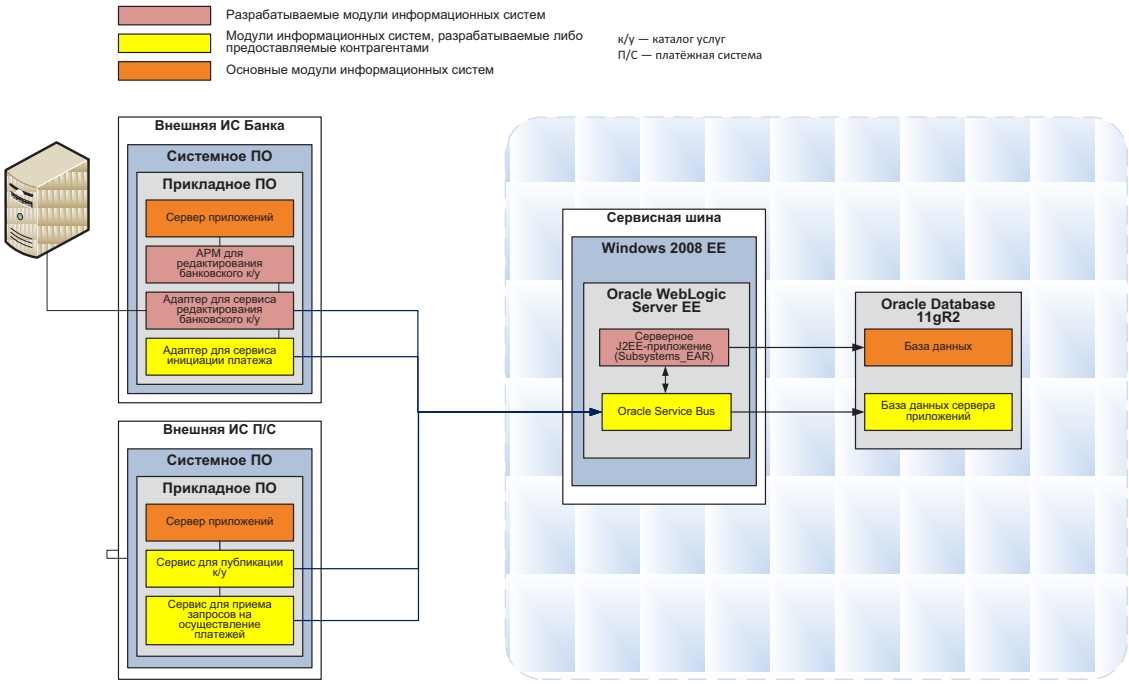

Рис. 1. Общая архитектура системы до изменений

Архитектура подобных приложений может строиться по-разному, в данном случае был выбран вариант с OracleServiceBus - из-за легкости развертывания, оперативности разработки и внесения изменений и масштабируемости[1].

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].

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.

Перестроение приложения

На этапе начала проекта такое решение было оптимальным из-за того, что широкое количество различных протоколов взаимодействия между ПС и короткие заявленные сроки разработки требовали быстрого и правильного процесса разработки. С применением сервис-шины уменьшается стоимость разработки, сложность отдельно взятого модуля. Также определенное значение имеют особенности архитектуры решения: OracleServiceBus не требует постоянного подключения к БД, имеет мощные средства мониторинга, встроенные в sbconsole, кэширование, простую трансформацию на основе XQuery и XSLT.OracleServiceBus идеально подходит для реализации синхронного взаимодействия между потребителем и провайдером сервиса.

С увеличением количества подключенных к системе клиентов стало очевидным, что данное решение не является оптимальным из-за плохой поддержки асинхронного взаимодействия и разнородности интегрируемых систем. В данной ситуации было принято решение отойти от существующей архитектуры и попробовать методологию «точка-точка». Интеграция одной стабильной системы с различными протоколами других систем приводила к тому, что необходимо было писать практически дублирующие друг друга схемы обработки, разветвленные и сложные пути взаимодействия, множественные вызовы внешних сервисов. Для масштабируемости было принято решение отказаться от промежуточного звена в виде сервис-шины и реализовать сервисы, используя технологию EJB3.0. Выбор стоял между архитектурой, построенной на Spring + Hibernate и EJB3.0. Выбор EJB3.0 обусловлен главным образом тем, что данная технология является спецификацией, со встроенной поддержкой распределенных транзакций. К тому же ejb3.0 обладает гораздо большей поддержкой модульного тестирования.

http://sntbul.bmstu.ru/doc/635498.html

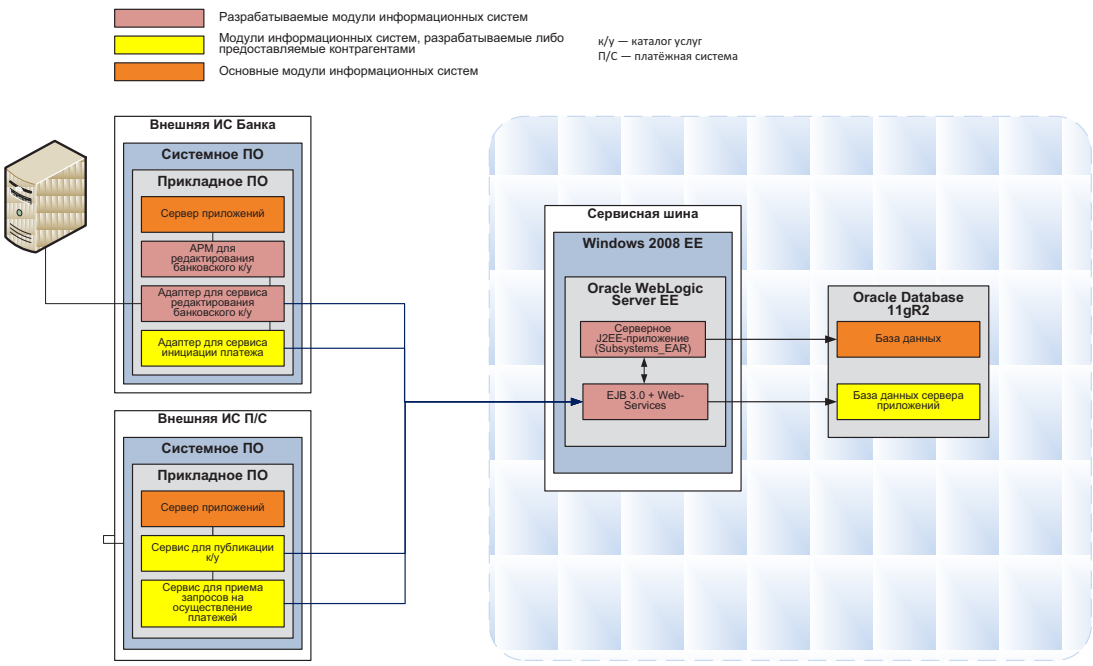

Рис. 2. Общая структура системы после перестроения

Технология EJB 3.0 позволяет не отходить от концепции SOA-архитектуры, применяя механизм веб-сервисов.

EJB 3.0 – стандарт в области разработки распределенных объектно-ориентированных приложений. Компоненты, разрабатываемые с помощью технологии EJB 3.0 – Sessionbeans и Message-drivenbeans. Sessionbeans инкапсулируют бизнес-процесс, message-drivenbeans позволяют создать коммуникации между приложениями используя механизм очередей. Компоненты EJB 3.0 являются переносимыми между серверами, а также платформенно-независимыми[2].

Session bean должен иметь локальный или удаленный интерфейс, для вызова компоненты.Message-driven bean работает с JMS сообщениями. Клиент, который может быть как standalone Java-приложением, так и другим компонентом JavaEE, посылает сообщение в очередь, которая управляется EJB-контейнером.

Кроме того, с использованием EJB 3.0 стало возможным обеспечить транзакционность обработки данных в приложении. На практике это означает то, что если происходит ошибка в действии, которое совершалось в транзакции, то происходит откат изменений к началу действия. Для интеграции платежных сервисов это является критичным, так как невыполнение этого условия влечет за то, что возможно снятие средств со счета клиента и не получение их на счете поставщика сервиса.

Тестирование

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.

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 SOAPUi (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):

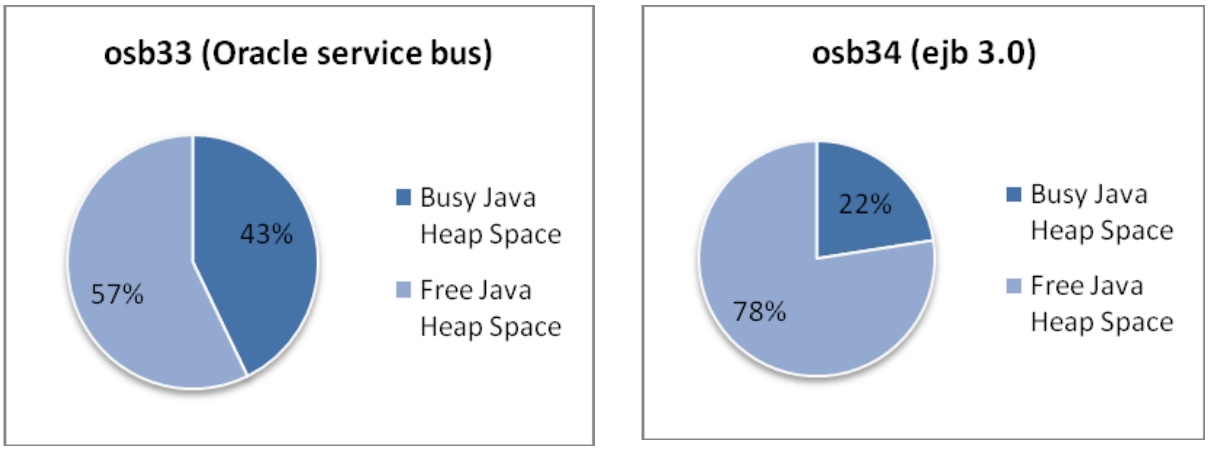

Рис. 3. Сравнение по размеру FreeJavaHeapSpace

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 OSBболее ориентирована на интеграцию нескольких веб-сервисов. Для сравнения можно

http://sntbul.bmstu.ru/doc/635498.html

рассмотреть то, что разработка веб-сервиса на OSB в среднем требует не таких значительных усилий, как разработка веб-сервиса используя технологию EJB 3.0.

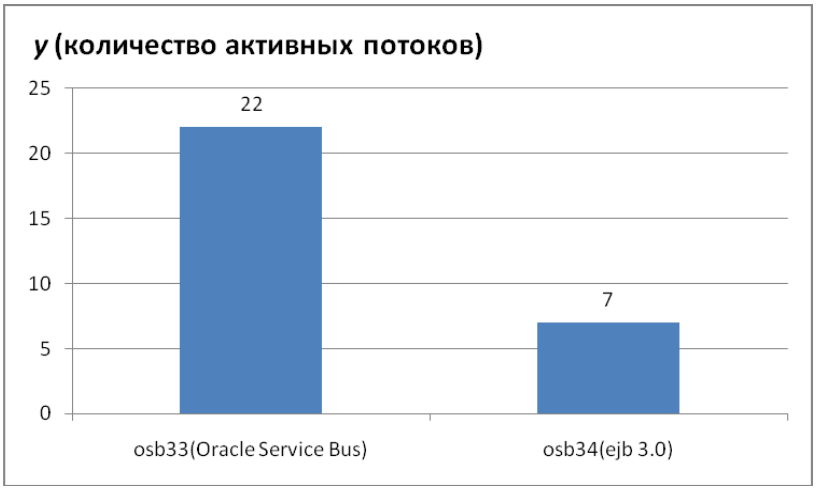

Рис. 4. Сравнение по количеству потоков

Рис. 5. Сравнение по количеству обработанных запросов

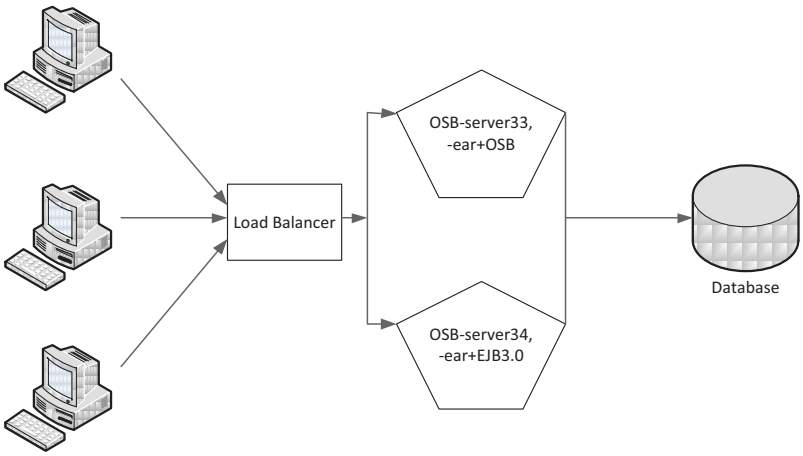

Рис. 6 Тестирование

Результаты

В настоящей статье был проведен сравнительный анализ двух широко применяющихся в разработке SOA – архитектуры подходов на примере системы интеграции с платежным сервисом. Рассмотрено построение системы на базе интеграции сервисов с применением сервис-шины, а также с помощью EJB 3.0.

Анализ показал, что применительно к данной задаче по нескольким критериям технология EJB 3.0 показывает лучшие результаты чем применение сервис-шины.В качестве критериев использовалось количество активных потоков на сервере приложений, а также количество обработанных запросов. Результаты работы были задействованы в рефакторинге существующей системы и в настоящее время проходят стадию тестирования.

Список литературы

1. Guido Schmutz, Edwin Biemond, Jan van Zoggel and MischaKöllikerOracle Service Bus 11g Development Cookbook –UK:Olton Birmingham,2012.-506c.
2. Rima Patel Sriganesh, Gerald Brose, Micah Silverman Mastering Enterprise JavaBeans 3.0 – USA:Wiley Publishing, Inc.,2006.-685c.
3. The Java EE 5 Tutorial [Электронныйресурс] / Eric Jendrock, Jennifer Ball, Debbie Carson, Ian Evans, Scott Fordin, Kim Haase, 2010. – Режим доступа: http://docs.oracle.com/javaee/5/tutorial/doc/bncnt.html (дата обращения: 13.03.2003).