УДК 0004.45:004.65

ИНТЕГРАЦИОННАЯ ШИНА ДЛЯ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ

А.В. Кучинский
Студент БГУИР.
Инженер-программист
в IBA Group

В.Н. Гутковский
Студент БГУИР
Инженер-программист
в IBA Group

И.И. Пилецкий
Доцент кафедры информатики БГУИР, кандидат физико-математических наук, доцент, старший научный сотрудник

Белорусский государственный университет информатики и радиоэлектроники,
Республика Беларусь
IBA-Group, Республика Беларусь
E-mail: alexkuchinskydev@gmail.com

А.В. Кучниский
Сертифицированный архитектор AWS, работает в IBA Group на должности Big Data Engineer. В область интересов входит: распределенная обработка данных, безопасность систем, работа с большими объемами данных, облачные вычисления.

В.Н. Гутковский
Работает full-stack разработчиком в IBA Group, по совместительству менеджер команды. Интересуется веб программированием, высоконагруженными системами, облачными вычислениями.

И.И. Пилецкий
В сфере IT более 47 лет. Участие в разработке нескольких десятков крупных проектов: главный конструктор проекта, главный архитектор программно-информационного обеспечения, руководитель проекта, начальник отдела, заведующий лабораторией (НИИ ЭВМ, Академия наук Беларуси, МБА, БГУИР). Автор десятков исследований, имеет более 95 публикаций.

Аннотация. Современные приложения редко работают изолированно. Приложение не может сделать что-либо значимое без взаимодействия с другими приложениями. Сервис-ориентированная архитектура интегрирует приложения для совместной работы и ускоряет их работу, разбивая приложение на части, которые могут быть объединены друг с другом. Взаимодействие между сервисами (отдельными приложениями) можно реализовать, используя интеграционную шину, которая упрощает вызов службы как для потребителя, так и для поставщика, управляя всеми сложными взаимодействиями между ними. Интеграционная шина не только упрощает вызов службы приложениями (или их частями), но и помогает им передавать данные и распространять уведомления о событиях. На текущий момент программные продукты в сегменте EBS (enterprise service bus) представлены такими крупными компаниями как IBM, Oracle и т.д. При этом отсутствуют полноценное open source решения для данного программного продукта. Наше решение предоставляет легковесную EBS, которая разработана и используется в рамках проекта СКАД ИИ [1] по обработке и анализу больших массивов данных.

Ключевые слова: сервисная шина предприятия, EBS, kafka обработка больших объемов данных, сервис ориентированная архитектура, open source.

Краткая характеристика предметной области. Сервисная шина предприятия реализует множество признанных шаблонов проектирования и спецификаций стандартов. Сервисная шина предприятия (enterprise service bus, ESB) – связующее программное обеспечение, обеспечивающее централизованный и унифицированный событийно-ориентированный обмен сообщениями между различными информационными системами на принципах сервис-ориентированной архитектуры [2, 3].

Сервисная шина реализует ряд важных связующих функций, обеспечивает интеграцию данных и приложений, обеспечивает взаимодействие всех компонент крупной территориально распределенной системы, как по данным, так и по управлению. Является брокером и обеспечивает гарантированную доставку сообщений (MQ) [4]. Служба - сервис, приложение, отдельный компонент для обработки сообщения. Потребитель – это компонент который передает сообщение службе и получает от нее ответ, направляя его для дальнейшей обработки.

Потребитель может вызвать службу синхронно либо асинхронно. С точки зрения потребителя различие заключается в следующем:

– Синхронный потребитель использует один поток для вызова службы; поток передает запрос, блокируется на время выполнения службы и ждет ответ;

– Асинхронный потребитель использует два потока для вызова службы; один - для передачи запроса, второй – для приема ответа.

Понятия асинхронный и синхронный часто путают с понятиями последовательный и параллельный. Последние понятия относятся к порядку выполнения различных задач, в то время как синхронный и асинхронный имеют дело со способом выполнения потоком одной задачи, такой как вызов одной службы. Хорошим способом понять различие между синхронным и асинхронным вызовом является рассмотрение последствий восстановления после сбоя:

– Синхронный. Если у потребителя возникает аварийная ситуация во время блокирования при работе службы, нельзя повторно подключиться к этой службе после перезапуска, поэтому ответ теряется. Потребитель должен повторить запрос и надеяться на отсутствие аварийной ситуации;

– Асинхронный. Если у потребителя возникает аварийная ситуация во время ожидания ответа на запрос, после перезапуска потребитель может продолжать ожидать ответ, то есть ответ не теряется.

Восстановление после сбоя – это не единственное различие между синхронным и асинхронным вызовами, но, если вы пробуете определить стиль конкретного используемого вызова, рассмотрение восстановления после сбоя поможет сделать это. В основе асинхронной ESB лежит известный шаблон Message Bus (Шина сообщений) [4].

Шина сообщений представляет собой набор каналов сообщений (известных также как очереди или темы), обычно настроенных как пары каналов запрос-ответ. Каждая пара представляет службу, которая может быть вызвана потребителем, использующим шину.

Этот потребитель передает сообщение в очередь запросов службы и затем (в асинхронном режиме) слушает очередь ответов для получения результата. Он знает, какой результат соответствует его конкретному запросу, поскольку результат имеет правильный корреляционный идентификатор.

Вызывающий службу потребитель на самом деле не знает о том, кто предоставляет службу. Провайдеры служб тоже подключены к шине сообщений и прослушивают ее на наличие сообщений запросов. Если имеется несколько провайдеров службы, они соревнуются друг с другом как потребители за получение конкретного запроса. Система сообщений, которую реализует шина сообщений, работает как диспетчер сообщений и распределяет сообщения запросов провайдерам служб, каким-либо образом оптимизируя это распределение в зависимости от баланса нагрузки, сетевых задержек и т.д. После получения запроса провайдером службы он выполняет службу и вкладывает результат в сообщение в обусловленную очередь ответов. То есть, провайдер и потребитель никогда прямо не знают о месторасположении друг друга; они знают только о шине сообщений и об адресе соответствующих каналов и могут взаимодействовать посредством общих каналов.

Такая шина сообщений является сущностью ESB, и здесь нет ничего нового. Интеграторы приложений использовали эту технологию более десятилетия, применяя такие продукты обработки очередей сообщений как WebSphere® MQ и TIBCO Enterprise Message Service [4, 5]. На самом деле часто говорят, что если вы используете продукты корпоративного обмена сообщениями, то у вас есть ESB.

Это не так, шина сообщений определенно является основой асинхронной ESB, но полная ESB – это нечто большее. ESB обладает способностями, которые никогда не имели шины сообщений:

– Способностями самоописания;

– Способностями обнаруживаемости,

Недостатком традиционной технологии шины сообщений является отсутствие способности к самоописанию. С точки зрения потребителя существует множество каналов для вызова множества служб. Но какой из этих каналов нужен для вызова потребителем желаемой службы? Потребитель не может просто предать запрос в любой канал запросов; он должен знать правильный канал, использующийся для вызова конкретной службы. В противном случае он может купить авиабилет вместо желаемой книги. Более того, даже после определения (каким-то образом) правильного канала (и канала для прослушивания на наличие ответа), потребитель должен знать формат данных, в котором нужно передать запрос (и в каком формате данных ожидать ответ).

Ниже приведены основные функции промышленных шин предприятия:

– Предоставление точек интеграции, интерфейсов взаимодействия;

– Оркестровка сервисов, маршрутизация (прием сообщений из сервисов отправка сообщений в сервисы);

– Преобразование сообщений;

– Доступ к данным из сторонних информационных систем с помощью готовых или специально разработанных адаптеров;

– Поддержка синхронного и асинхронного способа вызова служб;

– Использование защищенного транспорта, с гарантированной доставкой сообщений поддерживающего транзакционную модель;

– Обработка ошибок;

– Логирование;

– Разнообразные механизмы контроля и управления;

– Функции аудита;

– Функции самоописания;

– Функции самообнаружения.

Основные промышленные ESB (не open source) применяемы для разработки крупных корпоративных систем:

1. WebMethods (Software AG, семейство продуктов WebMethods, поглощённой одноимённой компанией);

2. ActiveMatrix Service Bus (Tibco);

3. Oracle Service Bus (Oracle, семейство Fusion Middleware);

4. WebSphere Message Broker (IBM, семейство WebSphere)

Достоинствами вышеперечисленных промышленных ESB является:

1) Организация размещения существующих систем осуществляется быстрее и дешевле;

2) Повышение гибкости;

3) ESB основывается на общепризнанных стандартах.

4) Наличие большого количества конфигурации для интеграции

При этом все они обладают следующими недостатками:

1) Сложность реализации;
2) Требование больших ресурсов;
3) Сложность установки;
4) Является платным программным обеспечением, т.е. не open source.

Перечисленные недостатки являются критическими для небольшого и среднего размера систем, где многие функции и конфигурации промышленных аналогов иногда просто не нужны.

В качестве MQ очереди, можно использовать open source решения:

- Apache Kafka [6];
- RabbitMQ [7].

Данные решения очередей сообщений не обладают функциями самоописания, само обнаруживаемости, управления. Они могут использоваться как один из компонентов ESB, но не вместо него.

Решение по разработке на базе Apache Kafka ESB для СКАД ИИ [1]. Управляющий компонент системы комплексного анализа данных интернет источников является легковесным аналогом EBS - промышленных продуктов, в котором реализованы специфичные для конкретной системы функции.

Для решения имеющейся задачи была предложена реализация управляющего компонента (шины предприятия, ESB) на базе Apache Kafka и Spring Framework.

Компонент системы представляет собой единую интеграционную платформу, как по данным так и по компонентам (приложениям) системы. УК обеспечивает взаимодействие всех компонентов, поддерживает все возможные форматы данных и является брокером СКАД ИИ (см. рисунок 1 и рисунок 2).

Так, УК обрабатывает следующие форматы входных и выходных данных:

− Документы публикаций (pdf, docx, txt, doc).
− Сложные JSON объекты.
− HTTP запросы.

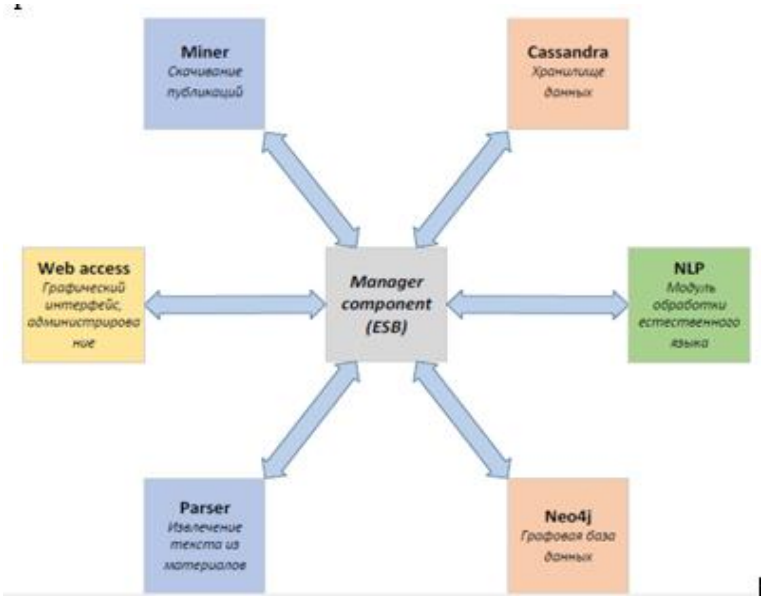

Рисунок 1. – Общая архитектура СКАД ИИ

Сама СКАД ИИ построена на принципах сервис-ориентированной архитектуры, где каждый компонент представляет собой сервис – отдельный и независимый модуль. Для реализации взаимодействия между компонентами используется разработанная сервисная шина ESB и RESTful, использующий HTTP протокол для передачи данных по сети. Внутри компонента обмен данными осуществляется издателями и подписчиками, которые, в свою

очередь, используют очереди сообщений.

Функции выполняемые ESB СКАД ИИ:

– Наличие обобщенного подхода (интерфейса, абстрактного класса) к обработке входящих и исходящих запросов/данных.

– Синхронная обработка событий с возможностью запуска нескольких потоков.

– Передача данных между компонентами должна осуществляться по HTTP протоколу.

– Передача данных внутри управляющего компонента должна осуществляться через очереди.

– В качестве менеджера очередей используются следующие продукты: Apache Kafka, Zookeeper.

– Компонент предоставляет возможность конфигурации рабочих очередей.

– Компонент предоставляет возможность конфигурации адресов сторонних сервисов.

– Реализован механизм обработки исключительных ситуаций

– Реализовано логирование.

– Функция получения информации об очередях.

– Функция получения информации о процессах и их статусов.

– Функция получения описания подключенных компонентов.

– Внедрение управляющего компонента не вызвало кардинального изменения уже существующих компонентов.

– Исключено прямое взаимодействие между компонентами, все операции осуществляются через обращения к интерфейсам управляющего модуля.

На рисунке 2, приведена логическая схема работы управляющего компонента СКАД ИИ. Для каждого модуля определены свои бизнес процессы.

На рисунке 3 приведена схема потоков данных и подписчики, издатели, очереди, а также брокеры, участвующие в передачи данных непосредственно внутри управляющего компонента.

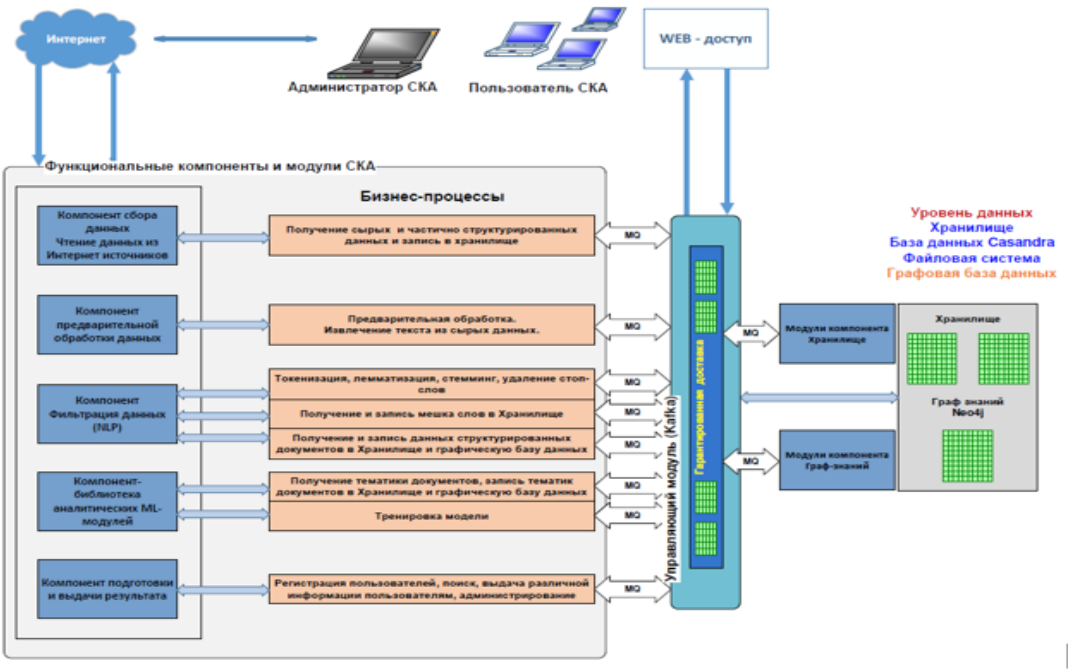

Рисунок 2. –Логическая схема работы управляющего компонента СКАД ИИ

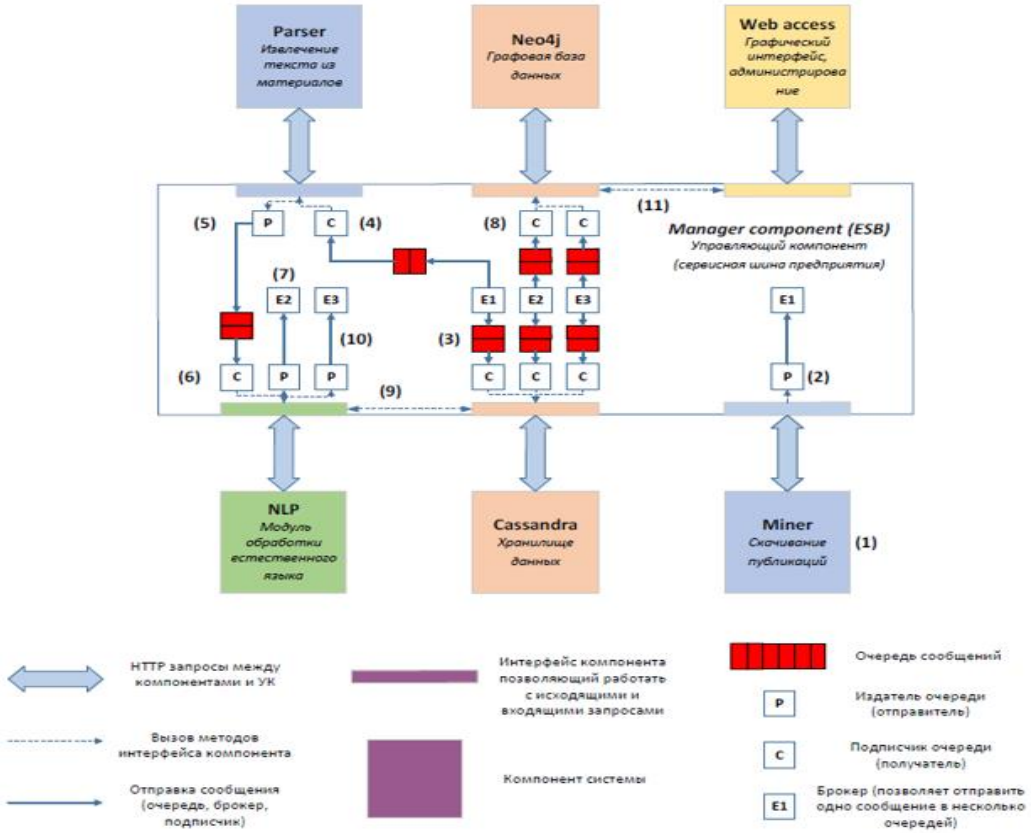

Рисунок 3. – Схема потоков данных

Основной результат:

Разработан управляющий компонент СКАД ИИ, который обеспечивает: автономную постоянную работу всех компонент системы,гарантированную доставку данных, взаимодействие всех компонент,надежное функционирование системы в целом (остановка работы одного из компонент не приводит к остановке работы всей системы в целом), мониторинг работы системы как по управлению, так и по данным, гибкую архитектуру системы, любой компонент может быть модифицирован независимо от другого, может быть добавлен новый компонент с другой функциональностью.

Данный подход может быть применен при разработке других сложных систем.

Список литературы

[1.] Батура М.П., Пилецкий И.И., Прытков В.А., Волорова Н.А., Козуб В.Н. Система комплексного анализа данных интернет источников // BIG DATA and Advanced Analytics = BIG DATA и анализ высокого уровня : сб. материалов V Междунар. науч.-практ. конф. (Республика Беларусь, Минск, 13–14 марта 2019 года). В 2 ч. Ч. 2 / редкол. : В. А. Богуш [и др.]. – Минск : БГУИР, 2019. – 379 с. ISBN 978-985-543-484-0 (ч. 2)..p/172-187

[2.] https://ru.wikipedia.org/wiki/ [2] Сервисная шина предприятия. [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/ Дата доступа: 24.02.2020.

[3.] https://www.ibm.com/developerworks/ru/library/ws-whyesb/index.html[3] Зачем разработчикам нужна Enterprise Service Bus? [Электронный ресурс] / Режим доступа: https://www.ibm.com/developerworks/ru/library/ws-whyesb/index.html Дата доступа: 28.02.2020

[4.] https://www.ibm.com/products/mq/deployment[4] Deploy IBM MQ everywhere your data is. [Электронный ресурс] / Режим доступа: https://www.ibm.com/products/mq/deployment Дата доступа: 28.02.2020.

[5.] https://www.tibco.com/products/tibco-enterprise-message-service[5] TIBCO Enterprise Message Service [Электронный ресурс] / Режим доступа: / Режим доступа: https://www.tibco.com/products/tibco-enterprise-message-service Дата доступа: 28.02.2020.

[6.] https://kafka.apache.org/[6] Apache Kafka. [Электронный ресурс] / Режим доступа: / Режим доступа: https://kafka.apache.org/ Дата доступа: 28.02.2020.

https://www.rabbitmq.com/#getstarted[7] RabbitMQ is the most widely deployed open source message broker. [Электронный ресурс] / Режим доступа: https://www.rabbitmq.com/#getstarted Дата доступа: 28.02.2020.
Key words: enterprice service bus, EBS, big data processing, service oriented architecture, open source

ENTERPRICE SERVICE BUS FOR BIG DATA PROCESSING

A.V. KUCHYNSKI
Student of Belarusian State University of Informatics and Radioelectronics. Software engineer IBA-Group.

U.N. HUTKOUSKI
Student of Belarusian State University of Informatics and Radioelectronics. Software engineer IBA-Group.

I.I. PILETSKI
PhD
Associate Professor of Informatics Department of the BSUIR

Belarusian State University of Informatics and Radioelectronics, Republic of Belarus
IBA-Group, Republic of Belarus
E-mail: alexkuchinskydev@gmail.com

Abstract. Nowadays modern applications are not working in isolated environment. Programs are communicating with each other with goal to perform requested operation. Service oriented architecture integrates applications and speeds up processing time decoupling this applications into separate parts making them loosely coupled. Communication between these services (separate applications) can be implemented using enterprise service bus (ESB).Bus simplifies communication between services both for producer and consumer, managing messages circulation between them. Enterprise service bus not only makes process of calling services easier, but also helps transferring data between system parts, delivery events, guarantee fault tolerance. Currently EBS software is represented by leading companies such as IBM, Oracle, etc. At the same time there is no open source projects devoted to this component. Our solution provides custom EBS which help us orchestrate services in local project which is devoted to massive data processing.