1 апреля 2019 г.

18.30-20.00

Ауд. А7

Проектирование систем распределенной и параллельной обработки данных

Лекция № 4

Тема: Оркестровка при работе микросервисов в системах обработки данных корпоративного уровня

Корпоративные информационные системы (КИС) – важный инструмент управления бизнесом, важнейшее средство производства современного предприятия в различных сферах деятельности. КИС включают в себя инфраструктуру (сети, серверы, рабочие станции, сетевое программное обеспечение (СПО), приложения) и информационные сервисы. Принцип построения КИС заключается в сегментировании сети по территориально-производственной принадлежности. Современные архитектуры КИС работают и управляются с помощью программного обеспечения (ПО) различного назначения.

Отказы прикладного ПО в КИС вызывают неудобства, замедление или временное прекращение работы на время перезагрузки ОС, но не приводят к столь существенным последствиям и потерям, которые могут быть в случае выхода из строя СПО.

Интеграционный подход в работе КИС подразумевает внедрение специализированной интеграционной платформы – программного комплекса, консолидирующего логику решения определенных интеграционных задач в рамках всей КИС предприятия.

Комплексное интеграционное решение на основе технологий глубокого интеграционного подхода представляет собой интеграционную платформу, состоящую в общем случае из всех интеграционных компонентов –

корпоративных информационных систем, СПО, операционных систем, систем управления бизнес-процессами, сотрудников и процессов

Все современные интеграционные платформы включают в себя СПО на уровнях:

– системы очередей сообщений (брокера сообщений);

– приложений внутри предприятия (Enterprise Application Integration (EAI));

– представления и преобразования данных;

– сервисной шины предприятия (Enterprise Service Bus (ESB));

– поддержки Web Services (в рамках SOA);

– автоматизации и управления бизнес-процессами (Business Process Automation (BPA), Business Process Management (BPM), развертывания и запуска BPEL на сервере приложений);

– взаимодействия с партнерами по бизнесу (Business To Business (B2B));

ESB (enterprise service bus, т.е. «сервисная шина предприятия») и SOA service-oriented architecture, т.е. «сервис-ориентированная архитектура» , проектирование компонентов и bounded contexts («ограниченный контекст» — паттерн из Domain-Driven Design).

Оркестровка — автоматическое размещение, координация и управление сложными компьютерными системами и службами.

Оркестровка описывает то, как сервисы должны взаимодействовать между собой, используя для этого обмен сообщениями, включая бизнес-логику и последовательность действий. Оркестровка подчинена какому-то одному из участников бизнес-процесса. В сервис-ориентированной архитектуре оркестровка сервисов реализуется согласно стандарту Business Process Execution Language (WS-BPEL).

BPEL расширяет модель взаимодействия веб-служб и включает в эту модель поддержку транзакций.

В общем виде конфигурация BPEL-проекта выглядит следующим образом:

– BPEL-визуальный редактор;

– Сервер управления бизнес-процессами.

Основные файлы BPEL-проекта:

– .bpel — логический синтез и координация веб-служб

– .wsdl — описание интерфейсов для обмена сообщениями

– .xsd — описание структур данных проекта.

Термины оркестровка и хореография описывают два аспекта разработки бизнес-процессов на основе объединения Web-сервисов. На рис. 1 в общем виде показана взаимосвязь этих аспектов, которые в какой-то мере дополняют друг друга.

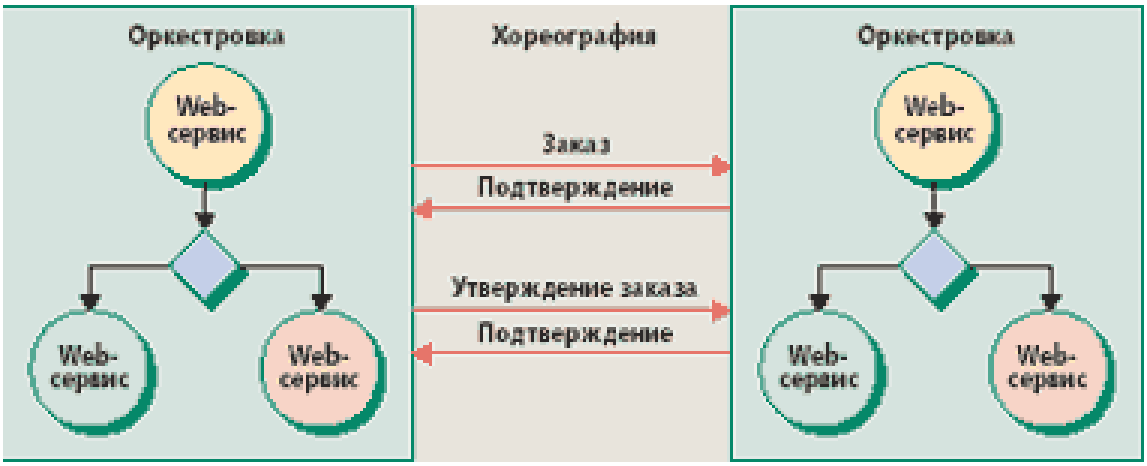

Рис. 1. Оркестровка относится к исполняемому процессу, а хореография позволяет отслеживать последовательности сообщений его участников

Оркестровка относится к исполняемому бизнес-процессу, который может взаимодействовать с внешними и внутренними Web-сервисами. Взаимодействия на основе обмена сообщениями включают в себя бизнес-логику и порядок выполнения задач. Они могут выходить за границы приложений и предприятий, определяя многошаговую транзакционную бизнес-модель.

Системы обнаружения сервисов автоматизируют процесс, позволяя получить ответ на вопрос, где работает нужный сервис, и изменить настройки в случае появления нового или отказа. Обычно под этим подразумевают набор сетевых протоколов (Service discovery protocols), обеспечивающих нужную функцию, хотя в современных реализациях это уже часть архитектуры, позволяющей обнаруживать связанные компоненты. На сегодня существует несколько решений, реализующих хранение информации об инфраструктуре, — как относительно сложных, использующих key/value-хранилище и гарантирующих доступность (ZooKeeper, Doozer, etcd), так и простых (SmartStack, Eureka, NSQ, Serf). Но, предоставляя информацию, они не слишком удобны в использовании и сложны в настройках.

В связи с активным применением сервисно-ориентированного подхода к организации архитектур корпоративных информационных систем возрастает значение методов и средств управления качеством Service Oriented Architecture (SOA), учитывающих их особенности. Так, в SOA нужная функциональность собирается (композиция) из сетевых сервисов, является слабосвязанной и общедоступной.

Сервис-ориентированная архитектура (SOA, англ. service-oriented architecture) — модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных (англ. loose coupling) заменяемых компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам.

Программные комплексы, разработанные в соответствии с сервис-ориентированной архитектурой, обычно реализуются как набор веб-служб, взаимодействующих по протоколу SOAP, но существуют и другие реализации (например, на базе jini, CORBA, на основе REST).

Интерфейсы компонентов в сервис-ориентированной архитектуре инкапсулируют детали реализации (операционную систему, платформу, язык программирования) от остальных компонентов, таким образом обеспечивая

комбинирование и многократное использование компонентов для построения сложных распределённых программных комплексов, обеспечивая независимость от используемых платформ и инструментов разработки, способствуя масштабируемости и управляемости создаваемых систем.

Стандарты оркестровки и хореографии должны удовлетворять ряду технических требований, обеспечивающих разработку бизнес-процессов с привлечением Web-сервисов. Эти требования касаются как языка, служащего для описания последовательности действий в бизнес-процессе, так и инфраструктуры для его выполнения.

Во-первых, для обеспечения надежности и универсальности, необходимых современным вычислительным средам, важна возможность асинхронного обращения к сервису. Повысить производительность процесса позволяет возможность одновременного обращения к нескольким сервисам. Для реализации асинхронных Web-сервисов требуется механизм корреляции запросов, в качестве которого архитекторы программных систем обычно используют идентификаторы корреляции.

Во-вторых, необходимо, чтобы архитектура бизнес-процесса обеспечивала управление исключительными ситуациями и целостностью транзакций. Кроме обработки ошибок и тайм-аутов оркестрованные Web-сервисы должны гарантировать доступность ресурсов при выполнении длительных распределенных транзакций. Традиционные транзакции обычно не вполне пригодны для реализации длительных распределенных бизнес-операций, поскольку не позволяют достаточно долго блокировать необходимые ресурсы. При использовании компенсирующих транзакций каждый метод включает в себя операцию отмены, которую координатор транзакций может вызвать в случае необходимости (под компенсирующей транзакцией понимают возможность отката операции при ее отмене процессом или потребителем).

В-третьих, оркестровка Web-сервисов должна быть динамичной, гибкой и адаптивной, чтобы отвечать изменяющимся потребностям бизнеса.

Достижению гибкости способствует четкое разделение логики процесса и используемых Web-сервисов, обычно обеспечиваемое механизмом оркестровки. Он обрабатывает поток работ бизнес-процесса, вызывая соответствующие Web-сервисы и определяя, какие шаги следует выполнить. Этот подход позволяет организации заменять сервисы в потоке работ.

Микросервисная архитектура — вариант сервис-ориентированной архитектуры программного обеспечения, ориентированный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — ***микросервисов***, получивший распространение в середине 2010-х годов в связи с развитием практик гибкой разработки и DevOps.

Если в традиционных вариантах сервис-ориентированной архитектуры модули могут быть сами по себе достаточно сложными программными системами, а взаимодействие между ними зачастую полагается на стандартизованные тяжеловесные протоколы (такие, как SOAP, XML-RPC), в микросервисной архитектуре системы выстраиваются из компонентов, выполняющих относительно элементарные функции, и взаимодействующие с использованием экономичных сетевых коммуникационных протоколов (в стиле REST с использованием, например, JSON, Protocol Buffers, Thrift). За счёт повышения гранулярности модулей архитектура нацелена на уменьшение степени зацепления и увеличение связности, что позволяет проще добавлять и изменять функции в системе в любое время.

Свойства, характерные для микросервисной архитектуры[1]:

– модули можно легко заменить в любое время: акцент на простоту, независимость развёртывания и обновления каждого из микросервисов;

– модули организованы вокруг функций: микросервис по возможности выполняет только одну достаточно элементарную функцию;

– модули могут быть реализованы с использованием различных языков программирования, фреймворков, связующего программного обеспечения, выполняться в различных средах контейнеризации, виртуализации, под

управлением различных операционных систем на различных аппаратных платформах: приоритет отдаётся в пользу наибольшей эффективности для каждой конкретной функции, нежели стандартизации средств разработки и исполнения;

– архитектура симметричная, а не иерархическая: зависимости между микросервисами одноранговые.

Философия микросервисов фактически копирует философию Unix, согласно которой каждая программа должна «делать что-то одно, и делать это хорошо» и взаимодействовать с другими программами простыми средствами: микросервисы минимальны и предназначаются для единственной функции. Основные изменения в связи с этим налагаются на организационную культуру, которая должна включать автоматизацию разработки и тестирования, а также культуру проектирования, от которой требуется предусматривать обход прежних ошибок, исключение по возможности унаследованного кода (микросервисы часто заменяют целиком, поскольку их функции элементарны).

Наиболее популярная среда для выполнения микросервисов — системы управления контейнеризованными приложениями (такие как Kubernetes и её надстройки OpenShift и CloudFoundry[en], Docker Swarm, Apache Mesos[en]), в этом случае каждый из микросервисов как правило изолируется в отдельный контейнер или небольшую группу контейнеров, доступную по сети другим микросервисам и внешним потребителям, и управляется средой оркестрации, обеспечивающей отказоустойчивость и балансировку нагрузки. Типовой практикой является включение в контур среды выполнения системы непрерывной интеграции, обеспечивающее автоматизацию обновления и развёртывания микросервисов.

В последнее время получили развитие альтернативные подходы к созданию веб-сервисов, основанные на архитектурном стиле REST («RESTful-веб-сервисов»). Разработка методов создания грид-сервисов на основе этого архитектурного стиля позволяет упростить интерфейсы грид-

сервисов, тем самым расширяя возможности их прямого использования из прикладных программ.

Подробнее на http://www.dissercat.com/content/printsipy-postroeniya-grida-s-ispolzovaniem-restful-veb-servisov#ixzz5jmqWEvvB

Пример возможной реализации гипотетической платформы для работы с видео: сначала в монолитном представлении (один большой блок), а затем — в микросервисном.

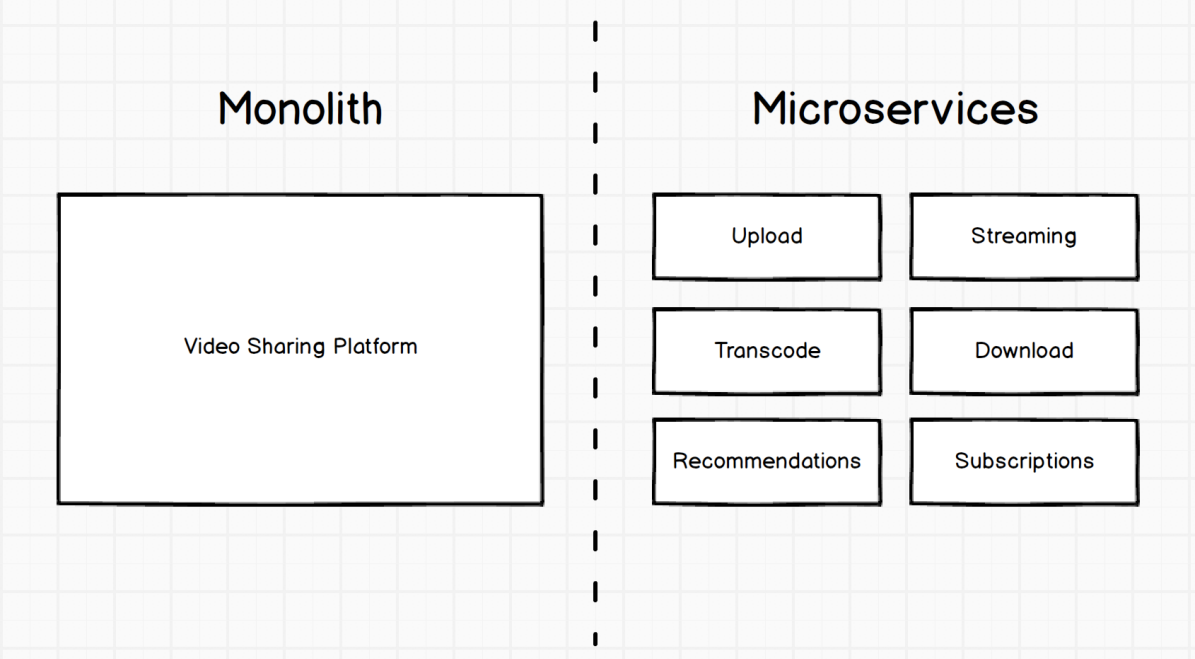

Рис.2 . Сопоставление подходов традиционного и микросервисного

Разница между этими двумя системами заключается в том, что первая — единый большой блок, т.е. монолит. Вторая — набор из маленьких специфичных сервисов. У каждого сервиса своя конкретная роль.

Когда схема представлена на таком уровне детализации, легко увидеть её привлекательность. Тут целый набор из потенциальных плюсов:

Независимая разработка. Маленькие независимые компоненты могут создаваться маленькими независимыми командами. Группа может работать над изменениями в сервисе *Upload*, не затрагивая сервис *Transcode* и даже не

зная о нём. Объём времени, необходимого для изучения компонента, значительно снижается, и разрабатывать новые функции становится проще.

Независимое развёртывание. Каждый отдельный компонент можно деплоить независимо. Это позволяет выпускать новые возможности (фичи) быстро и с меньшими рисками.

Независимая масштабируемость. Каждый компонент можно масштабировать независимо от другого. Во время повышенного пользовательского спроса, когда выходят новые передачи, компонент *Download* можно отмасштабировать для увеличившейся нагрузки без необходимости в масштабировании каждого компонента, что делает масштабирование гибким и снижает расходы.
Возможность повторного использования. Компоненты реализуют свою маленькую конкретную функцию. Это означает, что их проще адаптировать для использования в других системах, сервисах или продуктах. Компонент *Transcode* может быть использован другими подразделениями бизнеса или даже превращён в новый бизнес, предлагающий услуги транскодирования другой аудитории.

Для их выполнения используются специальные грид-сервисы, обеспечивающие «оркестровку», то есть последовательный или одновременный запуск отдельных шагов композитных заданий в соответствии с заданной логикой и отслеживание процесса их выполнения. Такие грид-сервисы выполняют центральную роль во многих существующих грид-проектах. Разработка грид-сервиса, обеспечивающего запуск задач, и имеющего простой и надежный программный интерфейс, построенный на основе новых методов, базирующихся на архитектурном стиле REST является важной прикладной задачей и хорошей проверкой практической ценности и эффективности предложенных методов.

Решение задач проектного управления также осуществляется за счет применения специализированных программных инструментов. Однако существующие системы проектного управления ограничены субъективным фактором в процессе контроля выполнения проектных работ. Данное ограничение может быть преодолено за счет согласованного применения инструментов управления проектами с технологиями процессно-ориентированного управления проектным производством на базе процессно-ориентированных информационных систем (ПОИС).

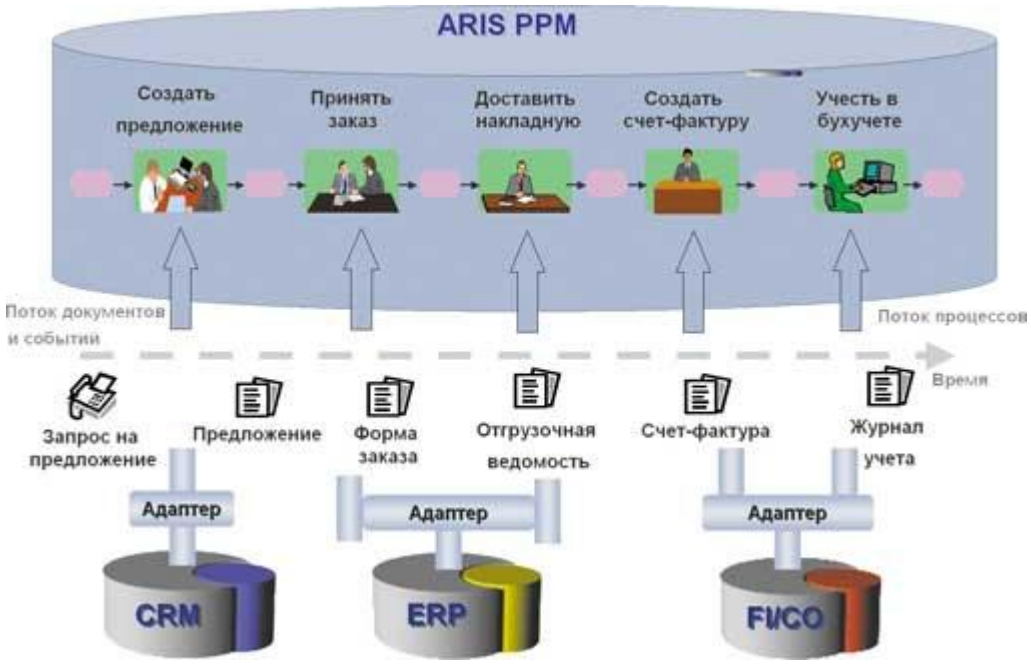

Рис. 3. Технология работы процессно-ориентированной аналитической системы.

Компоновка информации из разрозненных информационных систем в единое аналитическое хранилище

Результаты подобного анализа в различных вариантах могут быть представлены разным уровням управления компанией, начиная от высшего менеджмента и заканчивая участниками бизнес-процессов

В настоящее время существует множество информационных систем организационного управления, используемых в небольших организациях и подразделениях. Несмотря на разнообразие средств реализации и областей применения, эти системы имеют одинаковую архитектуру и множество особенностей, что позволяет выделить их в отдельный класс мультиаспектных информационных систем (МАИС). По мере появления новых функциональных задач и изменений требований к существующим задачам появляется необходимость модернизации и наращивания информационных систем. В данное время модернизация МАИС требует привлечения группы разработчиков: проектировщиков, специалистов по базам данных и программистов. Это приводит к большим трудозатратам на модернизацию и согласование отдельных компонентов системы. Одной из особенностей эксплуатации МАИС является постоянное внесение изменений в ее компоненты. Для повышения эффективности функционирования МАИС необходимо сократить трудозатраты на модернизацию, а также сократить долю участия в ней специалистов-разработчиков.