При использовании материалов статьи необходимо использовать данную ссылку:
Щекочихин О.В., Черкасова Н.В. Анализ архитектурных шаблонов на основе сервисов в задачах проектирования информационных систем // Информационно-экономические аспекты стандартизации и технического регулирования. 2019. № 5. (51). С. 30-36.

УДК 654.05

АНАЛИЗ АРХИТЕКТУРНЫХ ШАБЛОНОВ НА ОСНОВЕ СЕРВИСОВ В ЗАДАЧАХ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Щекочихин О.В., Черкасова Н.В.

Приводится описание основных архитектурных моделей для разработки информационных систем на основе сервисов, показаны их достоинства и недостатки, варианты развития технологий моделирования, рекомендации по их использованию и возможностям их комбинирования.

Ключевые слова: шаблон архитектуры, сервис-ориентированная архитектура, событийно-ориентированная архитектуры, микросервисная архитектура, информационная система.

С настройкой IoT-сред в наших личных, профессиональных и социальных целях для подключения интеллектуальных устройств, объем создаваемых, собираемых и обрабатываемых многоструктурных данных растет в геометрической прогрессии. Появление множества специфических и универсальных, встраиваемых и сетевых устройств приводит к глобальной цифровизации и проектированию больших данных. Это открывает новые возможности для бизнеса, а также провайдеров ИТ-услуг.

Учитывая тенденции и изменения, происходящие в ИТ-пространстве, предложены ряд научных разработок направленных на решение задач неограниченной масштабируемости, обеспечение целостности и доступности данных в условиях динамичного изменения функционала информационной системы.

В статье рассмотрены шаблоны архитектур ИС, которые являются основополагающими для создания и запуска любых программных приложений корпоративного и промышленного уровня. Чтобы приспособиться к эволюции информационных технологий и преодолеть сложности, возникающие из-за постоянных изменений в сфере бизнеса, рекомендуется рациональное использование разнообразных программных архитектурных моделей в качестве основы для развития ИС. Все чаще для проектирования и разработки сложных и интеллектуальных ИС тщательно выбираются различные модели для их последующего объединения и создания новых.

В настоящей статье рассмотрены существующие модели проектирования информационных систем на основе сервисов, проведен анализ их достоинств и недостатков [1-7].

При переход от монолитных ИС к интегрированным и распределенным получил распространение шаблон клиент-серверной архитектуры. Основой для взаимодействия между клиентами и серверами является метод запроса и ответа. Однако, передаваемые сообщения в этом случае являются синхронными. Для этого клиенты и серверы должны быть доступны в сети, чтобы инициировать и выполнить функции ИС. Когда запросы на обслуживание обрабатываются и выполняются серверными машинами, запрашивающие услуги клиенты должны ждать получения ответа от серверов и не могут выполнять какую-либо другую работу. То есть когда происходит событие (приходит запрос), приложения должны получать информацию о нём и немедленно приступать к его выполнению. Такая схема взаимодействия является ограничивающим фактором для совершенствования ИС. В случае, когда связь

Щекочихин Олег Владимирович, кандидат технических наук, доцент, инженер информационной безопасности ООО «ММТР технологии», г. Кострома
Черкасова Наталья Владимировна, аспирант, ФГБУН ВИНИТИ РАН, г. Москва

становится асинхронной нет необходимости, чтобы участвующие приложения были постоянно доступны онлайн.

Событийно-ориентированная архитектура (EDA, event-driven architecture), основывается на асинхронной коммуникационной модели, управляемой сообщениями, для распространения информации по всему корпоративной ИС, описывая деловые операции как серию событий.

Отправка вэб-форм и нажатие на некоторые гиперссылки генерируют события. Механизмы синхронизации базы данных, считывания RFID, сообщения электронной почты, служба коротких сообщений (SMS), обмен мгновенными сообщениями в мессенджерах и т.д. - это события, которые могут быть, как крупными, так и мелкими. Как правило, крупное событие состоит из нескольких мелких событий.

Каждое событие имеет:

- заголовок, который содержит элементы, описывающие подробности возникновения события: идентификатор спецификации, тип события, имя, создателя, отметка времени и т.д.

- тело события однозначно описывает, что произошло, то есть оно должно иметь всю необходимую и актуальную информацию, чтобы любая заинтересованная сторона могла использовать её для своевременного принятия необходимых мер. Если событие описано не полно, то заинтересованная сторона должна вернуться к исходной системе, чтобы проанализировать дополнительную информацию.

С помощью механизмов обработки событий, решений для промежуточного программного обеспечения, ориентированных на сообщения, таких как очереди сообщений и посредников для сбора и хранения данных и сообщений о событиях, можно собирать, анализировать и отправлять миллионы событий по множеству тем. Шаблон EDA построен на уведомлениях о событиях, чтобы облегчить немедленное распространение информации и выполнение бизнес-процессов. Поскольку создатели событий публикуют уведомления, получатели могут выбрать дальнейшее действие с ним в режиме реального времени. При этом информация может распространяться на все службы и приложения. Шаблон обеспечивает создание масштабируемых и высокоскоростных корпоративных приложений с возможностью аналитики в реальном времени. Архитектура состоит из специализированных компонентов обработки событий, которые анализируют события и обрабатывают их асинхронно.

В работе с EDA есть два основных направления:

ШАБЛОН EDA ПОСТРОЕН НА УВЕДОМЛЕНИЯХ О СОБЫТИЯХ, ЧТОБЫ ОБЛЕГЧИТЬ НЕМЕДЛЕННОЕ РАСПРОСТРАНЕНИЕ ИНФОРМАЦИИ И ВЫПОЛНЕНИЕ БИЗНЕС-ПРОЦЕССОВ. ПОСКОЛЬКУ СОЗДАТЕЛИ СОБЫТИЙ ПУБЛИКУЮТ УВЕДОМЛЕНИЯ, ПОЛУЧАТЕЛИ МОГУТ ВЫБРАТЬ ДАЛЬНЕЙШЕЕ ДЕЙСТВИЕ С НИМ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ. ПРИ ЭТОМ ИНФОРМАЦИЯ МОЖЕТ РАСПРОСТРАНЯТЬСЯ НА ВСЕ СЛУЖБЫ И ПРИЛОЖЕНИЯ.

1. Имеется одна очередь событий и посредник, который направляет каждое из событий соответствующим обработчикам событий. В этом случае события подаются в обработчики, проходящие через специальный канал, для фильтрации или предварительной обработки. Реализация очереди событий может быть в форме простой очереди сообщений или через интерфейс передачи сообщений.

2. Очередь событий отсутствует. Обработчики получают события, обрабатывают их и публикуют другое итоговое событие. То есть обработчики действуют, как посредники в цепочке событий. При этом некоторые обработчики занимаются только обработкой, но основная масса ещё и публикует результирующие новые события. Это сильно отличается от многоуровневой архитектуры, где все данные обычно проходят через все уровни.

Несмотря на достоинства данной архитектуры, в ней отсутствует атомарность транзакций, поскольку отсутствует последовательность выполнения событий. Процессоры событий реализуются с высокой степенью распределения, развязки и асинхронности, в следствие чего такие системы очень тяжело тестировать.

На сегодняшний день, информационные системы и их подсистемы всё чаще выражаются и представляются как сервисы и могут работать независимо от технологии реализации ядра ИС. Реализованы сервисы могут быть с использованием любых языков программирования, в том числе скриптовых языков. Сервисы взаимодействуют друг с другом посредством обмена сообщениями между

поставщиками (разработчиками услуг) и потребителями (клиентами). Каждый сервис состоит из двух частей: интерфейса и реализации. При помощи интерфейса запрашивается услуга, также интерфейсы дают необходимое разделение между сервисами. Необходимо использовать определённую схему структуры сообщений и обмена ими, особенно если сервис используется несколькими другими сервисами. Политика договора определяет масштабируемость, устойчивость, безопасность и другие критерии.

Сервис-ориентированная архитектура (SOA, service-oriented architecture) [2] отличается от архитектуры клиент-сервер тем, что сервисы являются универсально доступными и не сохраняют состояния активности в сети, в то время, как архитектура клиент-сервер требует тесной связи между участниками. Кроме того, данная модель позволяет предоставлять функциональность ИС в виде набора сервисов и создавать как персональные, так и профессиональные приложения, использующие программные сервисы.

СЕРВИСЫ МОГУТ ИНТЕГРИРОВАТЬ РАЗРОЗНЕННЫЕ И РАСПРЕДЕЛЕННЫЕ ПРИЛОЖЕНИЯ И ИСТОЧНИКИ ДАННЫХ. СЕРВИСНАЯ ШИНА (ENTERPRISE SERVICE BUS, ESB) – ЭТО ПРОМЕЖУТОЧНЫЙ КОМПОНЕНТ ИС, ОБЕСПЕЧИВАЮЩИЙ ИНТЕГРАЦИЮ НЕСКОЛЬКИХ РЕСУРСОВ НА ОСНОВЕ СЛУЖБ.

Сервисы могут интегрировать разрозненные и распределенные приложения и источники данных. Сервисная шина (Enterprise Service Bus, ESB) – это промежуточный компонент ИС, обеспечивающий интеграцию нескольких ресурсов на основе служб. ESB обеспечивает взаимосвязанность услуг, маршрутизацию, исправление, управление потоками данных и приложений. ESB предназначена для интеграции в любой сервисной среде, где сообщение является связующим звеном для взаимодействия между сервисами. Функция соответствия предписывает, какой поток сообщений должен выполняться при поступлении сообщения в ESB. В ESB отсутствует необходимость использования пользовательских разъемов, драйверов и адаптеров для интеграции процессов или приложений, источников данных и пользовательских интерфейсов.

Среди других важных функций рассматриваемой архитектуры можно отметить: маршрутизацию, перевод и преобразование формата сообщения. Выходной параметр, при маршрутизации сообщений от одного сервиса к другому, часто используется модулем потока сообщений, чтобы описать, какая служба будет вызываться для конкретного входящего сообщения. Существует много приложений и протоколов передачи сообщений. ESB может преобразовать протокол запроса в совместимый с поставщиком протокол. Что же касается преобразования формата сообщения, то когда запросчик отправляет сообщение в формате SOAP, ESB может вызвать провайдера с форматом сообщения EDIFACT. Технологией, лежащей в основе таких преобразований формата сообщений, может быть проверенное преобразование языка таблиц стилей XML (XSLT).

SOA - это динамический набор сервисов, которые взаимодействуют друг с другом. Связь может включать в себя либо простую передачу данных, либо две или более служб, координирующих какую-либо деятельность. Данная архитектура основана на традиционном механизме запрос-ответ. Потребитель услуг вызывает поставщика услуг через сеть и должен дождаться завершения операции на стороне провайдера. Ответы отправляются обратно потребителю синхронно. Такая архитектура предназначена для предоставления услуг и интеграции на основе сервисов монолитных и массовых приложений. ESB является наиболее оптимальным промежуточным программным обеспечением, позволяющим разрозненным и распределенным приложениям общаться друг с другом.

К недостатку данной архитектуры можно отнести тот факт, что его использование приводит к тесной взаимосвязи функций приложения из-за синхронной связи. Шаблон SOA может затруднить выполнение требований к корпоративным ИТ-системам следующего поколения. Шаблон может быть полезен, если требуется только синхронно отправлять запросы и получать ответы, но не для асинхронной обработки событий в реальном времени.

Следующим этапом развития архитектурных шаблонов стала *сервис-ориентированная архитектура, управляемая событиями (ED-SOA, event-driven service-oriented architecture),* которая сочетает в себе проверенные механизмы запроса-ответа SOA и события (публикации-подписки на событие) от EDA. Такой подход основан на модели асинхронного обмена сообщениями для передачи информации по всем видам

приложений корпоративного уровня ИС. Сервисы активируются событиями из разных источников, и полученные сообщения о событиях проходят через необходимые сервисы для выполнения определенной бизнес-операции. В этой модели устраняются все виды зависимостей. Архитектура ED-SOA подходит для работы с большими данными, для которых используются модели распределённых вычислений. Этот, управляемый событиями, шаблон позволяет системным архитекторам и дизайнерам обрабатывать, как сообщения о событиях, так и запросы на обслуживание, что обеспечивает более тесную связь между потребностями бизнеса и соответствующими ИТ-решениями.

Преимущества составного шаблона ED-SOA:

- Эффективная интеграция данных. В синхронной архитектуре, управляемой запросами, основное внимание уделяется повторному использованию удаленных функций и реализации ориентированной на процессы интеграции. Это означает, что интеграция данных, которая является важным аспектом интегрированных сред, изначально не поддерживается в средах SOA. Но в случае EDA внутренняя интеграция данных (сообщения о событиях) осуществляется, поскольку они являются базовой единицей связи и совместной работы.

- Своевременность и достоверность. События распространяются по всем участвующим приложениям для сбора, обработки, принятия решений и активации данных в режиме реального времени. Своевременный обмен данными (сообщениями о событиях) позволяет ИС иметь наиболее точное и свежее представление о состоянии в бизнесе.

- Улучшенная масштабируемость и устойчивость. Асинхронные системы имеют тенденцию быть более масштабируемыми по сравнению с синхронными. Отдельные процессы менее зависимы друг от друга, поэтому любые замены и усовершенствования могут быть легко выполнены в несвязанных системах.

Таким образом, выгодная комбинация шаблонов SOA и EDA позволяет создавать в реальном времени адаптивные и расширяемые информационные системы.

У SOA есть некоторые недостатки и ограничения. SOA в основном опирается на общую модель данных с несколькими иерархиями. Совместное использование баз данных в SOA имеет тенденцию создавать своего рода тесную связь данных между сервисами и другими компонентами системы, что приводит к появлению нежелательных ситуаций. Например, если несколько сервисов тесно связаны с внутренней базой данных и если в схеме базы данных произведены какие-либо изменения, то существует необходимость повторного тестирования сервисов, чтобы проверить работоспособность сервисов на измененной схеме.

Другая проблема заключается в том, что сервисы в SOA, как правило, имеют крупно-модульную структуру, и, следовательно, повторное использование является довольно сложным. Большинство компонентов приложений оснащены сервис-ориентированными интерфейсами для обеспечения возможности обнаружения, интеграции и взаимодействия. На уровне инфраструктуры проблем с зависимостями нет, потому что эти сервисные компоненты приложения могут запускаться буквально где угодно, и нет никаких ограничений для совместного размещения. SOA использует своего рода многоуровневую организационную архитектуру, которая содержит централизованное программное обеспечение для обмена сообщениями для вызова и координации сервисов. Но проблема заключается в том, что компоненты приложения должны знать соответствующие детали друг друга, чтобы инициировать и обеспечить требуемое сотрудничество, подтверждение и команду для закрытия.

СЕРВИСЫ В SOA, КАК ПРАВИЛО, ИМЕЮТ КРУПНО-МОДУЛЬНУЮ СТРУКТУРУ, И, СЛЕДОВАТЕЛЬНО, ПОВТОРНОЕ ИСПОЛЬЗОВАНИЕ ЯВЛЯЕТСЯ ДОВОЛЬНО СЛОЖНЫМ. БОЛЬШИНСТВО КОМПОНЕНТОВ ПРИЛОЖЕНИЙ ОСНАЩЕНЫ СЕРВИС-ОРИЕНТИРОВАННЫМИ ИНТЕРФЕЙСАМИ ДЛЯ ОБЕСПЕЧЕНИЯ ВОЗМОЖНОСТИ ОБНАРУЖЕНИЯ, ИНТЕГРАЦИИ И ВЗАИМОДЕЙСТВИЯ.

Например, группа разработчиков по руководством Дэвид Коррал-Плаза представила целостную, ориентированную на события сервис-ориентированную архитектуру (ED-SOA), которая имеет следующие важные свойства:

- Производители данных должны собирать данные из нескольких источников (базы данных, IoT-датчики, социальные сети и т. д.) и отправлять их сборщику данных.

- Сборщик данных выполняет необходимые преобразования, чтобы полученная информация могла использоваться на следующих этапах их решения. Это промежуточный уровень, который выполняет процесс согласования, поскольку в большинстве сценариев информация, скорее всего, будет получена в разных форматах и структурах.

- Обработка данных должна обеспечивать модуль комплексной обработки событий, контекстной ясности и прогнозирования.

- Потребители данных, которыми могут быть базы данных, конечные пользователи или дополнительные конечные точки, прокладывают путь для совместной архитектуры. Такие потребители данных связываются с предыдущим модулем через специальный интерфейс.

Архитектура микросервисов (MSA, microservices architecture) [1], [4] позволяет легко и быстро достичь нефункциональных требований, таких как масштабируемость, доступность и надежность для любой ИС. MSA предназначена для создания мелко-модульных, слабосвязанных, горизонтально масштабируемых, независимо развёртываемых, взаимодействующих, общедоступных, доступных для сети, легко управляемых и компонуемых услуг, которые не только являются оптимизированной единицей построения ИС, но также позволяют реализовать быстрое развертывание и её запуск.

Разрабатываемые ИС должны соответствовать современным требованиям и быть адаптивными, динамичными, открытыми, легко модифицируемыми и расширяемыми. В результате новейшие облачные, мобильные и встроенные приложения корпоративного класса создаются с использованием мощного и новаторского шаблона MSA. Растущая экосистема инструментов, механизмов, платформ и других программных инфраструктурных решений ускоряет процесс создания модулей ИС, ориентированных на микросервисы.

Микросервисы построены на концепции, известной как ограниченный контекст, который приводит к самодостаточной связи между отдельным сервисом и его данными. Каждый микросервис получает собственный источник данных, который может представлять собой файловую систему, SQL, NoSQL, NewSQL, кэш в памяти и т.д. Существуют шлюзовые решения API для оптимизации сквозного управления жизненным циклом микросервисов. Микросервисы полагаются исключительно на межсервисное общение. Каждый микросервис вызывает другой микросервис, необходимый для выполнения его функции. Кроме того, вызываемые микросервисы могут вызывать другие сервисы по мере необходимости в цепочке услуг. Микросервисы используют некоординирующий уровень API поверх сервисов, составляющих приложение. Наиболее подходящим вариантом хранения для микросервисов являются контейнеры Docker.

Каждый микросервис разрабатывается, как отдельный и самодостаточный программный продукт. Реализация ИС часто требует составления нескольких вызовов для этих отдельных функций и распределенных конечных точек. Синхронные вызовы запрос-ответ требуются, когда запрашивающий ожидает немедленного ответа, шаблоны интеграции, основанные на событиях и асинхронном обмене сообщениями, обеспечивают максимальную масштабируемость и отказоустойчивость.

MSA подходит для обслуживания и обработки больших данных. Требуемая модульность и экономически эффективное масштабирование достигается путём развёртывания служб на многих обычных аппаратных серверах. Модульность микросервисов также облегчает независимое обновление и помогает избежать единых точек отказа.

Микросервисный подход дает разработчикам больше гибкости в выборе оптимального промежуточного программного обеспечения для обмена сообщениями в пределах ИС. У каждого варианта применения микросервисов будут собственные требования по использованию различных технологий обмена сообщениями, таких как Apache Kafka, RabbitMQ или управляемые событиями сетки данных NoSQL, такие как Apache Geode / Pivotal GemFire.

МИКРОСЕРВИСНЫЙ ПОДХОД ДАЕТ РАЗРАБОТЧИКАМ БОЛЬШЕ ГИБКОСТИ В ВЫБОРЕ ОПТИМАЛЬНОГО ПРОМЕЖУТОЧНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ОБМЕНА СООБЩЕНИЯМИ В ПРЕДЕЛАХ ИС. У КАЖДОГО ВАРИАНТА ПРИМЕНЕНИЯ МИКРОСЕРВИСОВ БУДУТ СОБСТВЕННЫЕ ТРЕБОВАНИЯ ПО ИСПОЛЬЗОВАНИЮ РАЗЛИЧНЫХ ТЕХНОЛОГИЙ ОБМЕНА СООБЩЕНИЯМИ, ТАКИХ КАК APACHE KAFKA, RABBITMQ ИЛИ УПРАВЛЯЕМЫЕ СОБЫТИЯМИ СЕТКИ ДАННЫХ NOSQL, ТАКИЕ КАК APACHE GEODE / PIVOTAL GEMFIRE.

Общий шаблон архитектуры в MSA основан на использовании дополнительного потока событий, например, Kafka или MapR Streams. MapR Streams помогает группировать события в логические наборы, называемые темами. Темы разделены для параллельной обработки по типу очереди. События доставляются в порядке их получения. В отличие от очереди, события сохраняются, даже после доставки они остаются в разделе, доступном для других клиентов. Старые сообщения автоматически удаляются в зависимости от настройки времени жизни потока. Сообщения не удаляются из тем при чтении, и темы могут иметь несколько разных клиентов. Это позволяет обрабатывать одни и те же сообщения разными клиентами для разных целей. Конвейерная обработка также возможна, когда клиент обновляет событие и публикует его в другой теме.

ДЛЯ ПРЕДОСТАВЛЕНИЯ ИС ВОЗМОЖНОСТИ РАБОТАТЬ ДАЖЕ ПОД БОЛЬШОЙ НАГРУЗКОЙ ПОЛЬЗОВАТЕЛЕЙ, РАЗРАБОТАНА *ПРОСТРАНСТВЕННАЯ АРХИТЕКТУРА (SBA, SPACE-BASED ARCHITECTURE)*. ЭТО ДОСТИГАЕТСЯ ПУТЕМ РАЗДЕЛЕНИЯ ОБРАБОТКИ И ХРАНЕНИЯ МЕЖДУ НЕСКОЛЬКИМИ СЕРВЕРАМИ. ДАННЫЕ РАСПРЕДЕЛЕНЫ ПО МНОГИМ УЗЛАМ. МОДЕЛЬ SBA ШИРОКО ИСПОЛЬЗУЕТСЯ ДЛЯ РЕШЕНИЯ ПРОБЛЕМ МАСШТАБИРУЕМОСТИ И СОГЛАСОВАННОСТИ.

Для предоставления ИС возможности работать даже под большой нагрузкой пользователей, разработана *пространственная архитектура (SBA, Space-based architecture)*. Это достигается путем разделения обработки и хранения между несколькими серверами. Данные распределены по многим узлам. Модель SBA широко используется для решения проблем масштабируемости и согласованности.

Высокая масштабируемость достигается за счет устранения ограничения централизованной базы данных и использования вместо этого реплицированных сеток данных в памяти. Данные приложений хранятся в памяти и реплицируются среди всех активных процессоров. Процессорные блоки могут динамически включаться и выключаться по мере увеличения и уменьшения пользовательской нагрузки, тем самым обеспечивая переменную масштабируемость. Поскольку централизованная база данных отсутствует, узкое место в базе данных устраняется, обеспечивая не ограниченную масштабируемость ИС. Большинство ИС, построенные на шаблоне SBA, внешне представляются стандартными веб-сайтами, которые получают запрос от браузера и выполняют какие-то действия.

Рекомендуется применять комбинацию нескольких архитектурных шаблонов, чтобы ускорить реализацию ИС и услуг следующего поколения. На пример, объединение SOA и EDA, дает возможность создать основу для реализации динамических и адаптивных ИС, также и другие архитектурные шаблоны могут быть синхронизированы, чтобы произвести составные модели для выполнения определённых функций.

Сервис-ориентированные и управляемые событиями архитектуры должны быть интегрированы с технологиями нового поколения, такими как IoT, чтобы быть актуальными в условиях глобальной цифровизации. Связанные вещи, датчики, исполнительные механизмы, роботы, дроны, маяки, машины, оборудование, инструменты, и другие устройства должны предоставлять программным службам возможность учитывать контекст. Существуют платформы анализа данных IoT, растущий массив распределенных источников событий, механизмы потоковой аналитики, множество соединителей, драйверов и адаптеров, информационные панели визуализации знаний и другие поддерживающие инфраструктуры, продукты, нацеленные на создание контекстно-зависимых приложений по отраслевым вертикалям. С возрастающим значением IoT, архитектуры ИС и данных настроены на создание типовых шаблонов модулей ИС различного назначения. iea

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ:

1. Cesar de la Torre. NET Microservices: Architecture for Containerized .NET Applications / Cesar de la Torre, Bill Wagner, Mike Rousos. - Redmond, Washington: Microsoft Corporation, 2019 – 324 с.
2. SOA patterns [Электронный ресурс] URL: http://www.soapatterns.org/ ;
3. David Chou. Using Events in Highly Distributed Architectures / The Architecture Journal, статья от 14.01.2009, [Электронный ресурс] URL: https://docs.microsoft.com/en-us/previous-versions/dd129913 ;
4. Microservice architecture // [Электронный ресурс] URL: http://microservices.io/index.html;

5. Anuradha Wickramarachchi. Event Driven Architecture Pattern / Towards data science, статья от 15.09.2017, [Электронный ресурс] URL: https://towardsdatascience.com/event-driven-architecture-pattern-b54fc50276cd;
6. Щекочихин О.В. Объектно-процессная модель данных для сервис-ориентированной архитектуры интегрированных информационных систем // Научно-технический вестник информационных технологий, механики и оптики. 2018. Т. 18. № 2. С. 307–312. doi: 10.17586/2226-1494-2018-18-2-307-312
7. Щекочихин О.В. Сервис-ориентированная архитектура как обеспечение расширения функций управления корпоративной информационной системы // Тенденции развития науки и образования. Сборник научных трудов, по материалам XXIX международной научно-практической конференции 31 августа 2017 г. Часть 3 Изд. НИЦ «Л-Журнал», 2017. С 15-18

ANALYSIS OF ARCHITECTURAL TEMPLATES BASED ON SERVICES IN TASKS OF DESIGNING INFORMATION SYSTEMS

Shchekochikhin Oleg V., Candidate of Technical Sciences, Associate Professor, Information Security Engineer of MMTR Technologies LLC, Kostroma
Cherkasova Natal'ya V., graduate student, FSBI VINITI RAS, Moscow

The basic architectural models for developing information systems based on services are described, their advantages and disadvantages, development options for modeling technologies, recommendations for their use and their combination possibilities are shown.
Keywords: architecture pattern, service-oriented architecture, event-oriented architecture, microservice architecture, information system.

REFERENCES:

1. Cesar de la Torre. NET Microservices: Architecture for Containerized .NET Applications. Cesar de la Torre, Bill Wagner, Mike Rousos. - Redmond, Washington: Microsoft Corporation, 2019 – 324 c.
2. SOA patterns. Available at: URL: http://www.soapatterns.org/;
3. David Chou. Using Events in Highly Distributed Architectures. The Architecture Journal. Available at: URL: https://docs.microsoft.com/en-us/previous-versions/dd129913;
4. Microservice architecture. Available at: URL: http://microservices.io/index.html ;
5. Anuradha Wickramarachchi. Event Driven Architecture Pattern. Towards data science. Available at: URL: https://towardsdatascience.com/event-driven-architecture-pattern-b54fc50276cd;
6. Shchekochikhin O.V. Ob'yektno-protsessnaya model' dannykh dlya servis-oriyentirovannoy arkhitektury integrirovannykh informatsionnykh sistem [*Object-process data model for service-oriented architecture of integrated information systems*]. Nauchno-tekhnicheskiy vestnik informatsionnykh tekhnologiy, mekhaniki i optiki. 2018. T. 18. № 2. pp. 307–312. doi: 10.17586/2226-1494-2018-18-2-307-312
7. Shchekochikhin O.V. Servis-oriyentirovannaya arkhitektura kak obespecheniye rasshireniya funktsiy upravleniya korporativnoy informatsionnoy sistemy [*Service-oriented architecture as providing expansion of management functions of the corporate information system*]. Tendentsii razvitiya nauki i obrazovaniya. Sbornik nauchnykh trudov, po materialam XXIX mezhdunarodnoy nauchno-prakticheskoy konferentsii [*Trends in the development of science and education. Collection of scientific papers, based on the materials of the XXIX international scientific and practical conference*]. 31.08.2017. Chast' 3. Izd. NITS «L-Zhurnal», 2017. pp. 15-18