Архитектура информационных систем

Часть 2.
Платформенные архитектуры информационных систем.

Можно выделить три направления развития платформенных архитектур:

- Автономные.
- Централизованные.
- Распределённые.

Автономная архитектура.

Подразумевает наличие всех функциональных компонентов системы на одном физическом устройстве, например, компьютере и не должна иметь связей с внешней средой. Примером таких систем могут служить системные утилиты, текстовые редакторы и достаточно простые корпоративные программы.

В процессе построения корпоративной информационной системы, как правило, не должно формироваться не связанных узлов или модулей. Их появление может быть обусловлено определёнными требованиями к безопасности или надёжности.

Централизованная архитектура.

Подразумевает выполнение всех требуемых задач на специально отведённом узле, мощности которого достаточно, чтобы удовлетворить потребности всех пользователей.

Компоненты системы в данном случае распределяются между вычислительным узлом, который называется мейнфрейм (mainframe), и терминальной станцией, за которой работает пользователь. Терминал содержит компонент представления, а мейнфрейм – прикладной компонент и компонент управления ресурсами.

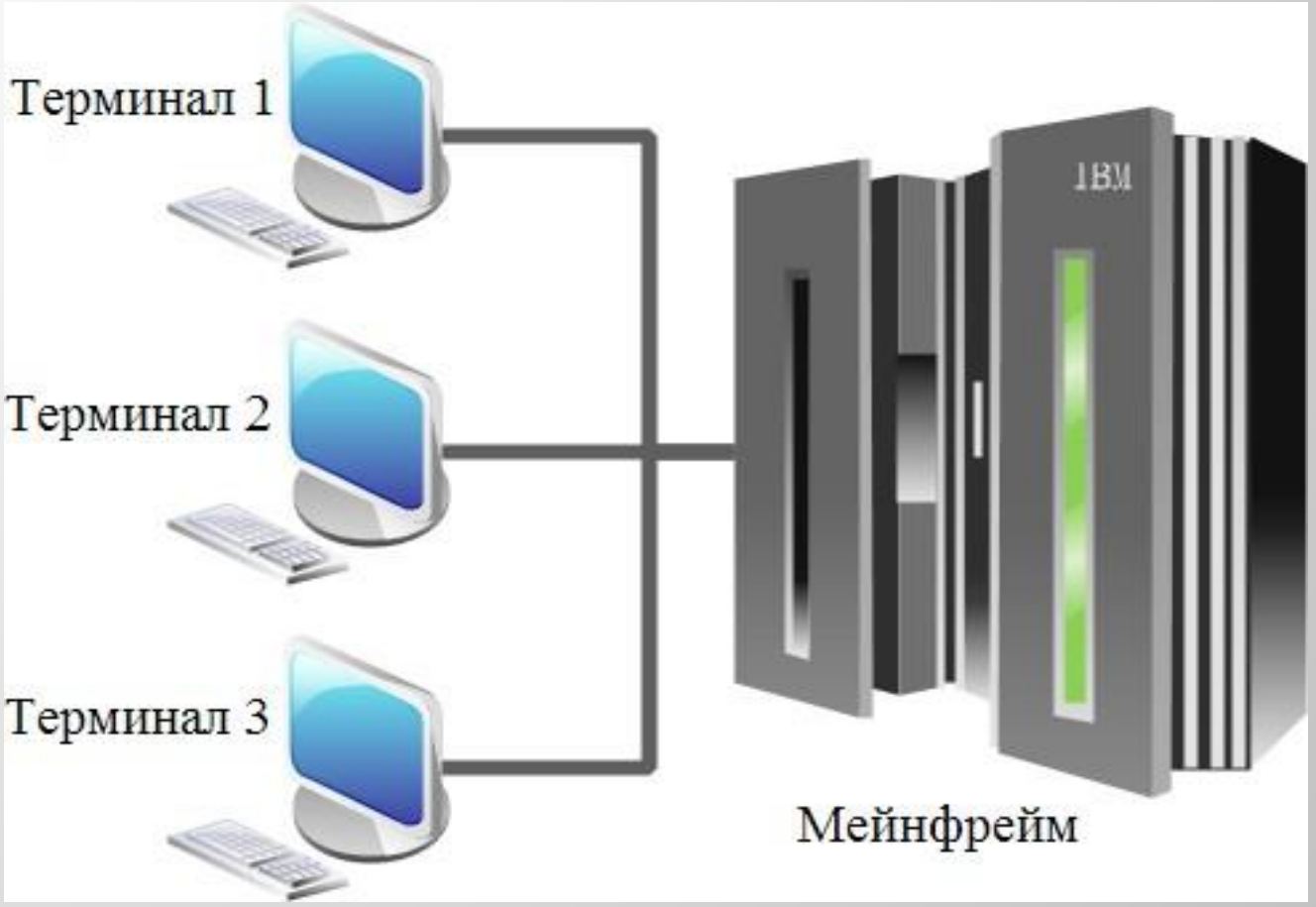


Рисунок 1 - Представление централизованной архитектуры

К достоинствам такой архитектуры можно отнести:

- отсутствие необходимости администрирования рабочих мест;
- лёгкость обслуживания и эксплуатации системы, поскольку все ресурсы сосредоточены в одном месте.

Недостатками подобной архитектуры являются:

- функционирование всей системы полностью зависит от главного узла (мейнфрейма);
- все ресурсы и программные средства являются коллективными и не могут быть изменены под нужды конкретных пользователей.

Чтобы избавиться от последнего недостатка, в современных информационных системах применяются технологии виртуализации, благодаря которым становится возможным выделить каждому пользователю необходимое количество ресурсов и установить требуемое программное обеспечение.

Распределенные архитектуры.

Их появление и развитие связано с интенсивным развитием технических и программных средств.

В данном типе архитектуры функциональные компоненты информационной системы распределяются по имеющимся узлам в зависимости от поставленных целей и задач.

Можно выделить шесть основных характеристик архитектуры распределённых систем:

- совместное использование ресурсов (как аппаратных, так и программных);
- открытость – возможность увеличения типов и количества ресурсов;
- параллельность – возможность выполнения нескольких процессов на различных узлах системы (при этом они могут взаимодействовать);
- масштабируемость – возможность добавлять новые свойства и методы;
- отказоустойчивость – способность системы поддерживать частичную функциональность за счёт возможности дублирования информации, аппаратной и программной составляющей.

К недостаткам распределённых систем следует отнести:

- структурная сложность;
- сложно обеспечить достаточный уровень безопасности;
- на управление системой требуется большое количество усилий;
- непредсказуемая реакция на изменения.

Все они связаны в первую очередь со сложной структурой, разноплановым оборудованием и сложной системой распределения прав доступа.

Существуют следующие виды распределённых архитектур:

- архитектура «файл-сервер»;
- архитектура «клиент-сервер» (двухзвенная, многозвенная);
- архитектура Web-приложений.
- Сервис-ориентированная архитектура.

Файл-серверная архитектура.

Подразумевает наличие выделенного сетевого ресурса для хранения данных. Такой ресурс называется «файловым сервером». При такой архитектуре все функциональные компоненты системы расположены на пользовательском компьютере, который называется «клиентом», а сами данные находятся на сервере.

Такая организация системы имеет следующие достоинства:

- многопользовательский режим работы с данными, хранящимися на сервере;
- централизованное управление правами доступа к общим данным;
- малая стоимость разработки;
- высокая скорость разработки.

Недостатки файл-серверной архитектуры:

- последовательный доступ к общим данным и отсутствие гарантии их целостности;
- производительность (зависит от производительности сети, клиента и сервера);

Классическое представление файл-серверной архитектуры представлено на рис. 2.

Рисунок 2 - Представление файл-серверной архитектуры

Архитектура «клиент-сервер» представляет собой сетевую инфраструктуру, в которой серверы являются поставщиками определённых сервисов (услуг), а клиентские компьютеры выступают их потребителями.

Классическое представление клиент-серверной архитектуры подразумевает наличие в сети сервера и нескольких подключённых к нему клиентов. В таких системах сервер, в основном, играет роль поставщика услуг по использованию базы данных.

Эта архитектурная модель называется двухзвенной или двухуровневой (two-tier architecture). Двухзвенная архитектура представлена на рис. 3.

Рисунок 3 - Двухзвенная (двухуровневая)клиент-серверная архитектура

Преимущества данной архитектуры:

- поддержка многопользовательской работы;
- гарантия целостности данных;
- наличие механизмов управление правами доступа к ресурсам сервера;
- возможность распределения функций между узлами сети.

Недостатки:

- выход из строя сервера может повлечь неработоспособность всей системы;
- высокая стоимость оборудования;
- требуется высокий уровень технического персонала.

Трёхуровневая архитектура

По сравнению с клиент-серверной или файл-серверной архитектурой трёхуровневая архитектура обеспечивает, как правило, большую масштабируемость (за счёт горизонтальной масштабируемости сервера приложений и мультиплексирования соединений, большую конфигурируемость (за счёт изолированности уровней друг от друга), более широкие возможности по обеспечению безопасности и отказоустойчивости. Кроме того, в сравнении с клиент-серверными приложениями, использующими прямые подключения к серверам баз данных, снижаются требования к скорости и стабильности каналов связи между клиентом и серверной частью. Реализация приложений, доступных из веб-браузера или из тонкого клиента, как правило, подразумевает развёртывание программного комплекса в трёхуровневой архитектуре. При этом обычно разработка приложений для трёхуровневых программных комплексов сложнее, чем для клиент-серверных приложений, также наличие дополнительного связующего программного обеспечения может налагать дополнительные издержки в администрировании таких комплексов.

Слой клиента

Самый верхний уровень приложения с интерфейсом пользователя. Главная функция интерфейса представление задач и результатов понятных пользователю.

Слой логики

Этот слой координирует программу, обрабатывает команды, выполняет логические решения и вычисления, выполняет расчеты. она также перемещается и обрабатывает данные между двумя окружающими слоями.

Слой данных

Здесь храниться информация и извлекается из базы данных и файловой системе. Информация отправлякться в логический слой для обработки, и в конечном счете возвращается пользователю.

Для доступа к тем или иным сетевам сервисам используются клиенты, возможности которых характеризуются понятием «толщины». Оно определяет конфигурацию оборудования и программное обеспечение, имеющиеся у клиента. Возможные граничные значения:

«Тонкий» клиент. Этот термин определяет клиента, вычислительных ресурсов которого достаточно лишь для запуска необходимого сетевого приложения через web-интерфейс. Пользовательский интерфейс такого приложения формируется средствами статического HTML (выполнение JavaScript не предусматривается), вся прикладная логика выполняется на сервере.

Для работы тонкого клиента достаточно лишь обеспечить возможность запуска web-браузера, в окне которого и осуществляются все действия. По этой причине web-браузер часто называют "универсальным клиентом".

«Толстый» клиент. Таковым является рабочая станция или персональный компьютер, работающие под управлением собственной дисковой операционной системы и имеющие необходимый набор программного обеспечения. К сетевым серверам «толстые» клиенты обращаются в основном за дополнительными услугами (например, доступ к web-серверу или корпоративной базе данных).

Так же под «толстым» клиентом подразумевается и клиентское сетевое приложение, запущенное под управлением локальной ОС. Такое приложение совмещает компонент представления данных (графический пользовательский интерфейс ОС) и прикладной компонент (вычислительные мощности клиентского компьютера).

При увеличении масштабов системы может потребоваться замена аппаратной части сервера и клиентских машин. Однако, при увеличении числа пользователей возникает необходимость синхронизации версий большого количества приложений. Для решения этой проблемы используют многозвенные архитектуры (три и более уровней).

Часть общих приложений переносится на специально выделенный сервер, тем самым снижаются требования к производительности клиентских машин. Клиенты с низкой вычислительной мощностью называют «тонкими клиентами», а с высокой производительностью – «толстыми клиентами».

При многозвенной архитектуре с выделенным сервером приложений существует возможность использования портативных устройств. Многозвенная архитектура показана на рис.4.

Рисунок 4 - Многозвенная клиент-серверная архитектура

Распределение функциональных компонентов системы при использовании клиент-серверной архитектуры может производиться несколькими способами (рис.5).

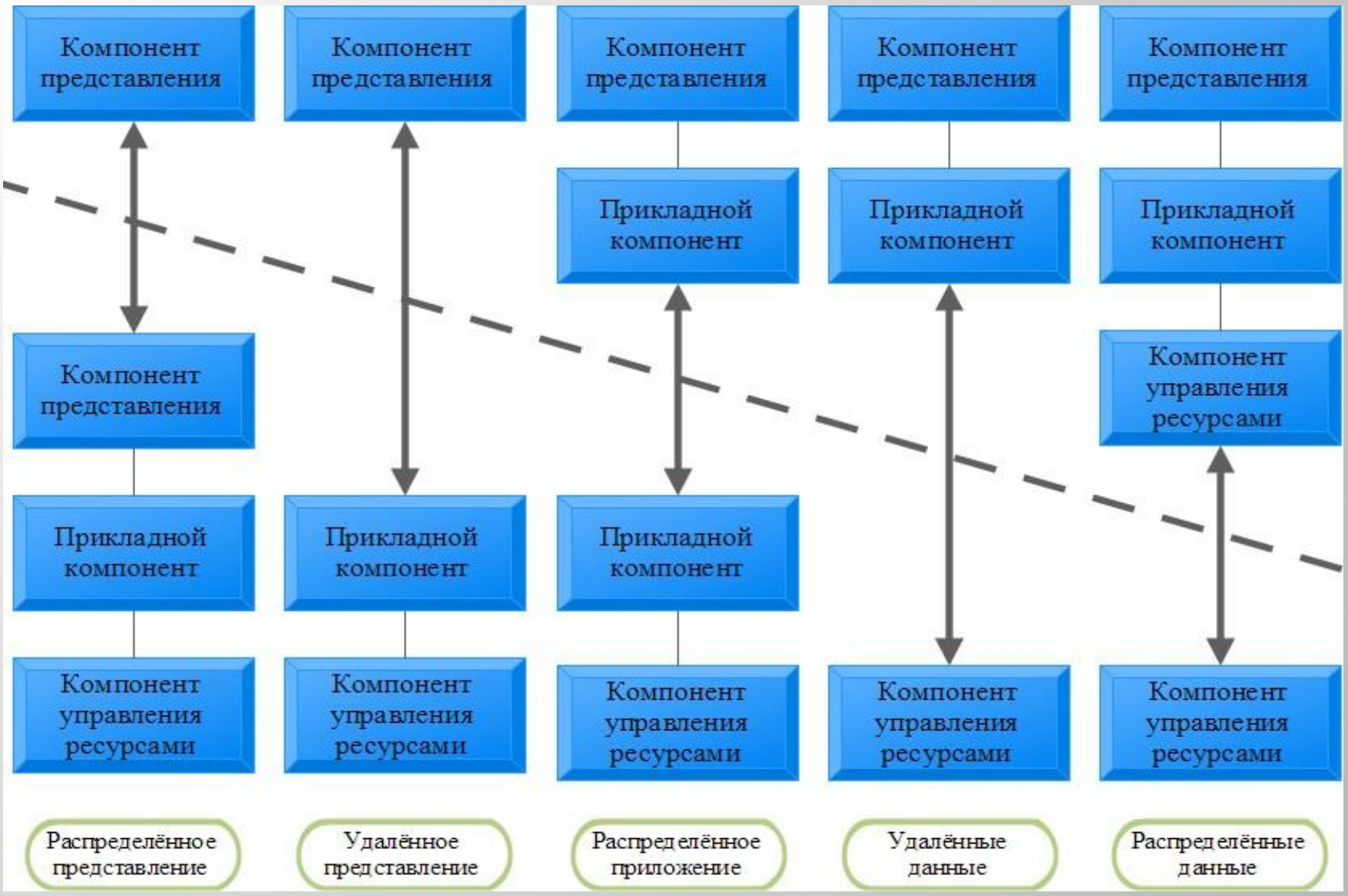


Рисунок 5 - Примеры распределения функциональных компонентов системы

Архитектура Web-приложений.

Или архитектура Web-сервисов подразумевает предоставление некоторого сервиса, доступного в сети Internet, через специальное приложение.

Основой для предоставления таких услуг служат открытые стандарты и протоколы SOAP, UDDI и WSDL. SOAP(Simple ObjectAccess Protocol) определяет формат запросов к Web-сервисам. Данные между клиентом и сервисом передаются в SOAP-конвертах (envelops). WSDL (Web Service Description Language) служит для описания интерфейса предоставляемого сервиса

Перед развёртыванием web-приложения требуется составить его описание, указать адрес, список поддерживаемых протоколов, перечень допустимых операций, а так же форматы запросов и ответов.

UDDI (Universal Description, Discovery and Integration) представляет собой протокол поиска Web-сервисов в сети Internet. Поиск осуществляется по их описаниям, которые расположены в специальном реестре.

Архитектура таких сервисов схожа по концепции с многозвенной клиент-серверной, однако, сервера при- ложений и баз данных располагаются в сети Internet.

Можно выделить три технологии, которые возможно использовать для построения распределённой архитектуры Web-сервиса:

1. **EJB** (Enterprise JavaBeans).

2. **DCOM** (Distributed Component Object Model).

3. **CORBA** (The Common Object Request Broker Architecture).

EJB

Идеей для создания **EJB** было желание создать такую инфраструктуру, в которой было бы легко добавлять и удалять компоненты, при этом изменяя функциональность сервера.

EJB позволяет разработчикам составлять собственные приложения из заранее созданных модулей. При этом возможно их изменение, что делает процесс разработки гибким и гораздо более быстрым. Данная технология совместима с CORBA и Java API.

Взаимодействие между клиентов и сервером в данном случае представляется как взаимодействие EJB-объекта, генерируемого специальным генератором, и EJB-компонента, написанного разработчиком. При необходимости вызвать метод у EJB-компонента, находящегося на сервере, вызывается одноимённый метод EJB-объекта, расположенного на стороне клиента, который связывается с требуемой компонентой и

Достоинства EJB:

- простое и быстрое создание;
- Java-оптимизация;
- кроссплатформенность;
- встроенная безопасность.

Недостатки EJB:

- сложность интегрирования с приложениями;
- плохая масштабируемость;
- низкая производительность;
- отсутствие международной стандартизации.

DCOM

представляет собой распределённую программную архитектуру от компании Microsoft.

С её помощью программный компонент одного компьютера может передавать сообщения программному компоненту другого компьютера, причём соединение устанавливается автоматически.

Для надёжной работы требуется обеспечить защищённое соединение между связанными компонентами, а также создать систему перенаправления трафика.

Достоинства DCOM:

- независимость от языка;
- динамическое нахождение объектов;
- масштабируемость;
- открытый стандарт;

Недостатки DCOM:

- сложность реализации;
- зависимость от платформы;
- поиск через службу Active Directory;
- отсутствие именования сервисов через URL.

CORBA

Технология **CORBA** рассматривает все приложения в распределённой системе как набор объектов. Объекты могут одновременно выступать в роли клиента и сервера, вызывая методы других объектов и отвечая на их вызовы. Применение данной технологии позволяет строить системы, превосходящие по сложности и гибкости системы с архитектурой клиент-сервер (как двухуровневой, так и трёхуровневой).

Достоинства CORBA:

- независимость от платформы;
- независимость от языка;
- динамические вызовы;
- динамическое обнаружение объектов;
- масштабируемость;
- индустриальная поддержка.

Недостатки:

- отсутствие именования по URL;
- практически полное отсутствие реализации CORBA-сервисов;

Сервис-ориентированная архитектура

При грамотном подходе к построению архитектуры информационной системы может потребоваться использование сразу нескольких из рассмотренных технологий.

Каждая из них будет реализовывать определённый функционал, который может потребовать также нескольких типов платформенных архитектур.

В одной системе могут работать несколько файл-серверов, несколько серверов приложений и несколько серверов баз данных.

Таким образом можно распределять нагрузку в системе или группировать

Сервис-ориентированная архитектура (SOA)

Сервис-ориентированная архитектура (service-oriented architecture, SOA) придумана в конце 1980-х.

Она берёт своё начало в идеях, изложенных в CORBA, DCOM, DCE и других документах. О SOA написано много, есть несколько её реализаций. Но, по сути, SOA можно свести к нескольким идеям, причём архитектура не диктует способы их реализации:

- Сочетаемость приложений, ориентированных на пользователей.
- Многократное использование бизнес-сервисов.
- Независимость от набора технологий.
- Автономность (независимые эволюция, масштабируемость и развёртываемость)

SOA — это набор архитектурных принципов, не зависящих от технологий и продуктов, совсем как полиморфизм или инкапсуляция.

SOA — не набор технологий, а прежде всего **процессно-ориентированная архитектура** ИС. Можно определить SOA следующим образом: это архитектура приложений, построенная на основе **формализованных бизнес-процессов**, функции которых представлены в виде многократно используемых сервисов с прозрачными описаниями

Сервис-ориентированная архитектура (SOA)

Паттерны (образец, шаблон), относящиеся к SOA:

- Общая архитектура брокера объектных запросов (CORBA).
- Веб-сервисы.
- Очередь сообщений.
- Сервисная шина предприятия (ESB).
- Микросервисы.

Микосервисная архитектура ИС

В последние десятилетия SOA сильно эволюционировала. Благодаря неэффективности прежних решений и развитию технологий сегодня мы пришли к микросервисной архитектуре.

Эволюция шла по классическому пути: сложные проблемы разбивались на более мелкие, простые в решении.

Проблему сложности кода можно решать так же, как мы разбиваем монолитное приложение на отдельные доменные компоненты (разграниченные контексты). Но с разрастанием команд и кодовой базы увеличивается потребность в независимом развитии, масштабировании и развёртывании. SOA помогает добиться такой независимости, упрочняя границы контекстов.

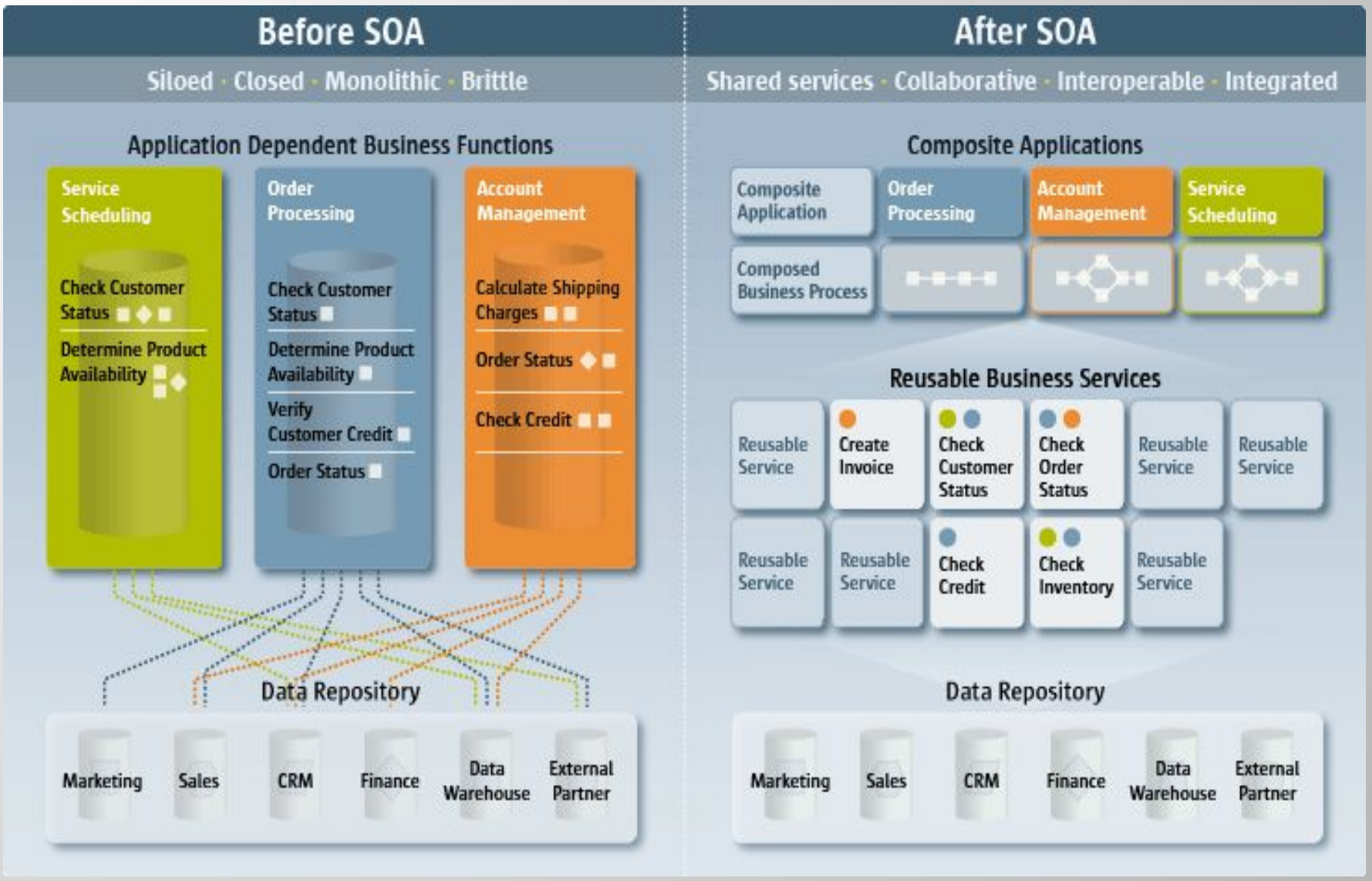


Рисунок 6 – Диаграмма До SOA и После SOA

Диаграмма, показанная на рисунке 6, дает объяснение с помощью визуализации бизнес-модели, показывая разницу между архитектурой, ориентированной на обслуживание, и монолитной архитектурой.

Сервис-ориентированная архитектура (SOA) сильно развилась в последние несколько лет и широко используется в мире веб-разработки. SOA имеет тенденцию разбивать крупные монолитные приложения на более мелкие службы, что приводит к снижению сложности процесса разработки. Однако эта архитектурная модель не удовлетворяла потребности всех разработчиков в сообществе разработчиков программного обеспечения. Для того, чтобы воспользоваться преимуществами архитектуры, ориентированной на обслуживание, и избежать их недостатков, вводится микросервисная архитектура. Одним из приложений для использования шаблона микрослужб является создание приложения для Интернета вещей. Интернет вещей (IoT) считается перспективным подходом к интеграции физического мира в компьютерный, виртуальный мир, который повысит эффективность электронного использования объектов физического мира.

Интернет вещей

Под Интернетом вещей будем понимают единую сеть, соединяющую окружающие нас объекты реального мира и виртуальные объекты.
IOT – концепция пространства, в котором все из аналогового и цифрового миров может быть совмещено – это переопределит наши отношения с объектами, а также свойства и суть самих объектов.

По одному из определений, с точки зрения IoT, «*вещь*» – *любой реальный или виртуальный объект, который существует и перемещается в пространстве и времени и может быть однозначно определен*.

Т.е. Интернет вещей – это не просто множество различных приборов и датчиков, объединенных между собой проводными и беспроводными каналами связи и подключенных к сети Интернет, а это более тесная интеграция реального и виртуального миров, в котором общение производится между людьми и устройствами.

Предполагается, что в будущем «вещи» станут активными участниками бизнеса, информационных и социальных процессов, где они смогут взаимодействовать и общаться между собой, обмениваясь информацией об окружающей среде,

По мнению Роба Ван Краненбурга интернет вещей представляет из себя «четырех-слойный пирог».

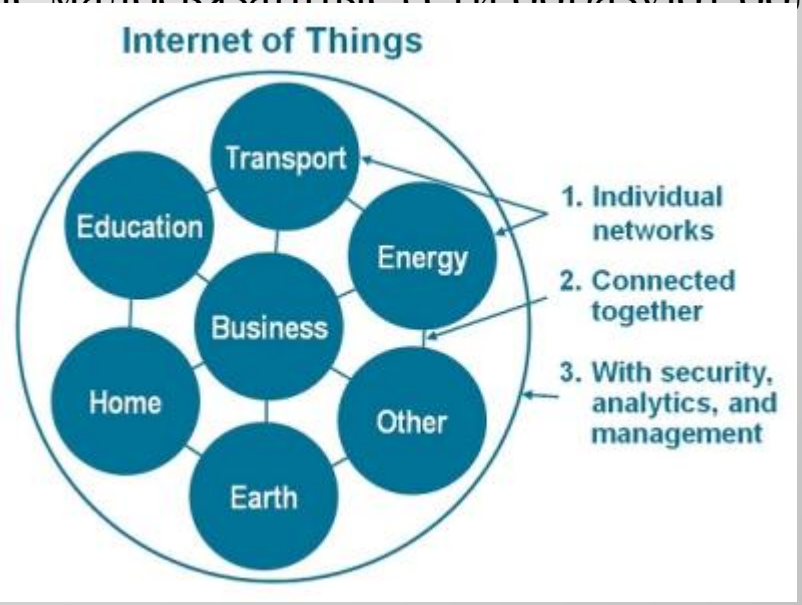
1 уровень связан с идентификацией каждого объекта.

2 уровень предоставляет с сервисом по обслуживанию потребностей потребителя (можно рассматривать как сеть собственных «вещей», частный пример – «умный дом»).

3 уровень связан с урбанизацией городской жизни. Т.е. это концепция «умного города», где вся информация, которая касается жителей этого города, стягивается в конкретный жилой квартал, в Ваш дом и соседние дома.

4 уровень – сенсорная планета.

Иными словами Интернет вещей можно рассматривать как сеть сетей, в которой небольшие малосвязанные сети образуют более крупные.



Для общения и взаимодействия приборов необходим единый язык. Компания Cisco провела тщательный технический анализ, показавший, что IP вполне может быть адаптирован к требованиям сетей нового типа. В таком случае «Интернет вещей» получит те же преимущества: совместимость, масштабируемость и, самое главное, единый общий язык, — которые в свое время превратили сложный массив частных и общедоступных сетей в единую глобальную коммуникационную систему, известную как Интернет.

IP – это всего лишь средство связи - «голосовые связки и уши» устройств.

Данную концепцию связывают, как правило, с развитием двух технологий. Это радиочастотная идентификация (RFID) и беспроводные сенсорные сети (БСС).

Беспроводные сенсорные сети

Беспроводная сенсорная сеть — это распределенная, самоорганизующаяся сеть множества датчиков (сенсоров) и исполнительных устройств, объединенных между собой посредством радиоканала. Причем область покрытия подобной сети может составлять от нескольких метров до нескольких километров за счет способности ретрансляции сообщений от одного элемента к другому.

Применяется данная технология для решения многих практических задач связанных с мониторингом, управлением, логистикой и пр.

RFID

RFID (Radio Frequency Identification, радиочастотная идентификация) — метод автоматической идентификации объектов, в котором посредством радиосигналов считываются или записываются данные, хранящиеся в так называемых транспондерах, или RFID-метках.

Данная технология хорошо подходит для отслеживания движения некоторых объектов и получения небольшого объема информации от них. Так, например, если бы все продукты были оснащены RFID-метками, а холодильник RFID-ридером, то он легко мог бы отслеживать срок годности продуктов, а мы могли бы, например, уходя с работы удаленно заглянуть в холодильник и определить, что надо закупить еще.

Проблемы и недостатки

Самой главной проблемой на сегодняшний день является отсутствие стандартов в данной области, что затрудняет возможность интеграции предлагаемых на рынке решений и во многом сдерживает появление новых.

Так же для полноценного функционирования такой сети необходима автономность всех «вещей», т.е. датчики должны научиться получать энергию из окружающей среды, а не работать от батареек, как это происходит сейчас.

Наличие огромной сети, контролирующей весь окружающий мир, глобальная открытость данных и прочие особенности могут иметь и негативные последствия.