Лекция 2

Архитектура информационных систем

Вопросы

1. Базовые функции информационных систем

2. Традиционные архитектуры информационных систем

1. Базовые функции информационных систем

Архитектура информационной системы - концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы.

Компоненты информационной системы по выполняемым функциям можно разделить на три слоя: слой представления, слой бизнес-логики и слой доступа к данным Рисунок 1.

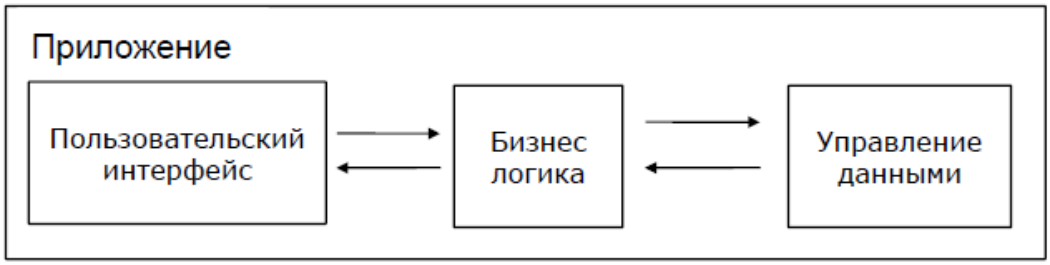

Рисунок 1. Компоненты информационной системы

Пользовательский интерфейс - все, что связано с взаимодействием с пользователем: нажатие кнопок, движение мыши, отрисовка изображения, вывод результатов поиска и т.д.

Бизнес логика - правила, алгоритмы реакции приложения на действия пользователя или на внутренние события, правила обработки данных.

Управление данными - хранение, выборка, модификация и удаление данных, связанных с решаемой приложением прикладной задачей

Традиционные архитектуры информационных систем

1. Файл-серверная архитектура

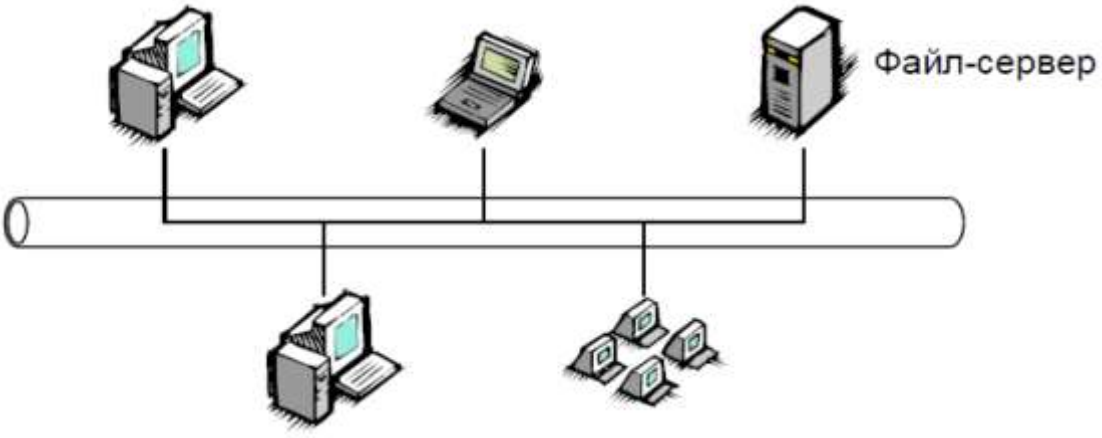

Рисунок 2. Файл-серверная архитектура

Файл-серверные приложения — приложения, схожие по своей структуре с локальными приложениями и использующие сетевой ресурс для хранения программы и данных. Функции сервера: хранения данных и кода программы. Функции клиента: обработка данных происходит исключительно на стороне клиента Рисунок 3.

Количество клиентов ограничено десятками.

Рисунок 3.Модельфайлового сервера

Плюсы:

Многопользовательский режим работы с данными;

Удобство централизованного управления доступом;

Низкая стоимость разработки;

Минусы:

Первым недостатком архитектуры с файловым сервером данные хранятся в одном месте, а обрабатываются в другом.

Это означает, что их нужно передавать по сети, что приводит к очень высоким нагрузкам на сеть и, вследствие этого, резкому снижению производительности приложения при увеличении числа одновременно работающих клиентов.

Вторым важным недостатком такой архитектуры является децентрализованное решение проблем целостности и согласованности данных и одновременного доступа к данным.

Такое решение снижает надежность приложения.

2. Клиент-серверная архитектура

Архитектуры клиент-сервер позволило создавать надежные (в смысле целостности данных) многопользовательские ИС с централизованной базой данных, независимые от аппаратной (а часто и программной) части сервера БД и поддерживающие графический интерфейс пользователя на клиентских станциях, связанных локальной сетью. Причем издержки на разработку приложений существенно сокращались Рисунок 4.

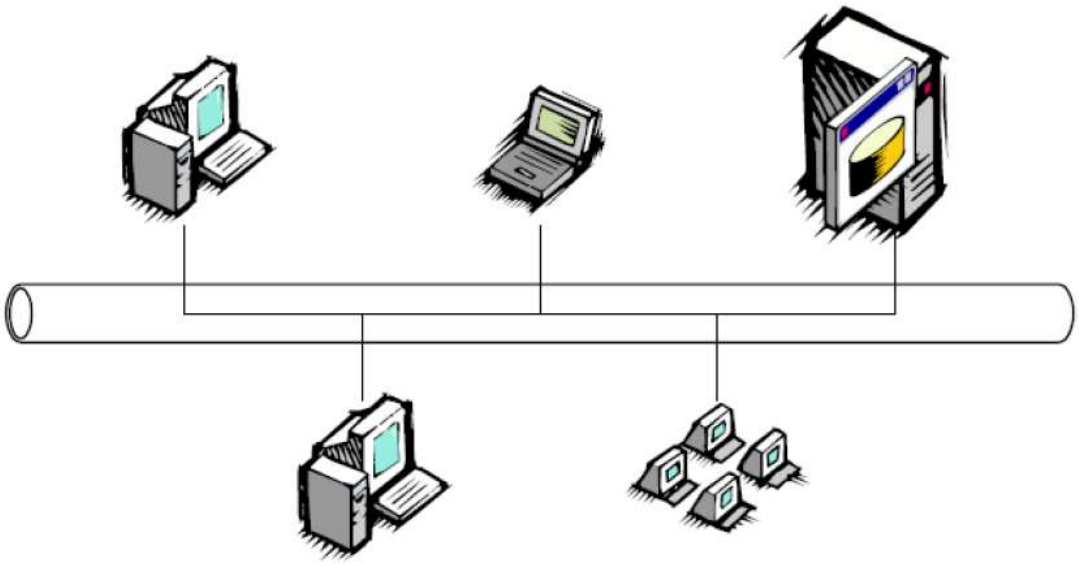

Рисунок 4. Клиент-серверная архитектура

Основные особенности:

Клиентская программа работает с данными через запросы к серверному ПО.

Базовые функции приложения разделены между клиентом и сервером Рисунок 5.

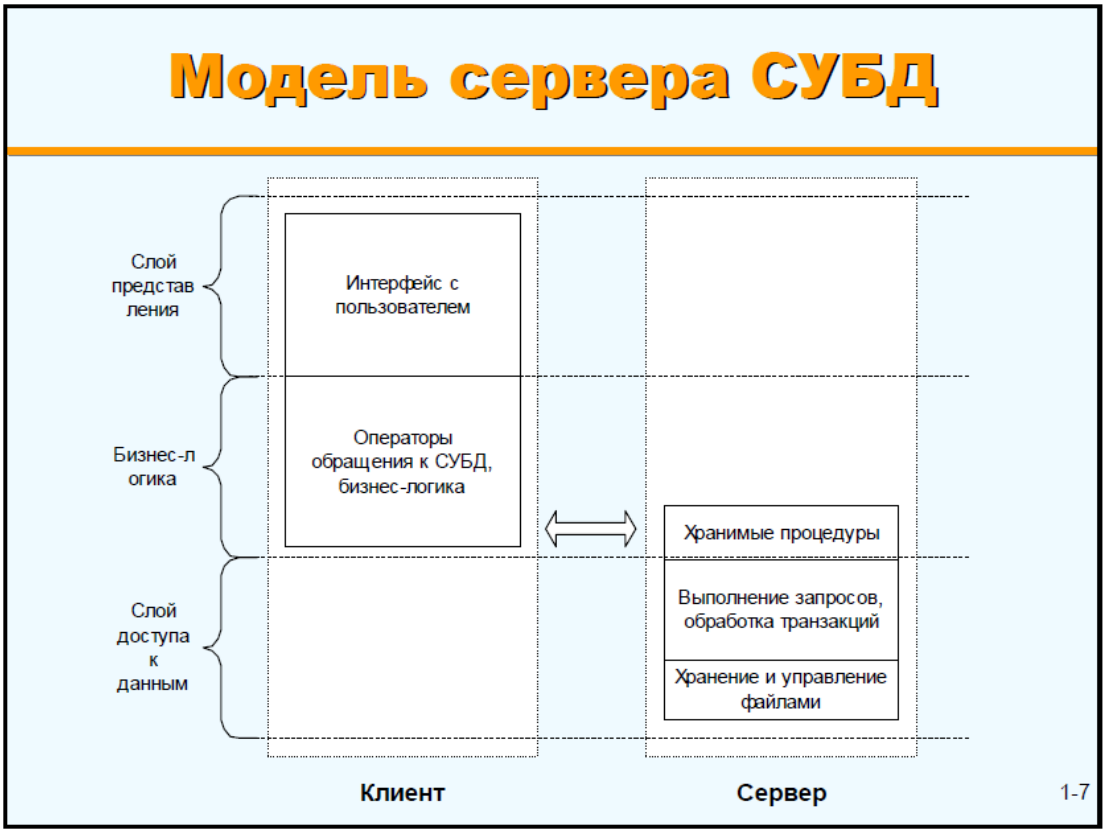

Рисунок 5 Модель сервера СУБД.

Плюсы:

Полная поддержка многопользовательской работы

Гарантия целостности данных

Минусы:

Бизнес логика приложений осталась в клиентском ПО. При любом изменении алгоритмов, надо обновлять пользовательское ПО на каждом клиенте.

Высокие требования к пропускной способности коммуникационных каналов с сервером, что препятствует использование клиентских станций иначе как в локальной сети.

Слабая защита данных от взлома, в особенности от недобросовестных пользователей системы.

Высокая сложность администрирования и настройки рабочих мест пользователей системы.

Необходимость использовать мощные ПК на клиентских местах.

Высокая сложность разработки системы из-за необходимости выполнять бизнес-логику и обеспечивать пользовательский интерфейс в одной программе.

Нетрудно заметить, что большинство недостатков классической или 2-х слойной архитектуры клиент-сервер проистекают от использования клиентской станции в качестве исполнителя бизнес-логики ИС.

Поэтому очевидным шагом дальнейшей эволюции архитектур ИС явилась идея "тонкого клиента", то есть разбиения алгоритмов обработки данных на части связанные с выполнением бизнес-функций и связанные с отображением информации в удобном для человека представлении.

При этом на клиентской машине оставляют лишь вторую часть, связанную с первичной проверкой и отображением информации, перенося всю реальную функциональность системы на серверную часть.

3. Переходная к трехслойной архитектуре (2.5 слоя)

Использование хранимых процедур и вычисление данных на стороне сервера сокращают трафик, увеличивают безопасность. Клиент все равно реализует часть бизнес-логики.

В качестве клиентских программ часто применяют стандартные www-броузеры.

Программы для серверной части пишут, в основном, на специализированных языках, пользуясь механизмом хранимых процедур сервера БД.

Таким образом, на уровне логической организации, ИС в архитектуре клиент-сервер с тонким клиентом расщепляется на три слоя –

слой данных,

слой бизнес-функций (хранимые процедуры)

слой представления.

2.5-слойная архитектура обычно не требует наличия высокоскоростных каналов связи между клиентской и серверной частями системы, так как по сети передаются уже готовые результаты вычислений - почти все вычисления производятся на серверной стороне.

Существенно улучшается также и защита информации - пользователям даются права на доступ к функциям системы, а не на доступ к ее данным.

Недостатки

ограниченную масштабируемость,

зависимость от программной платформы, ограниченное использование сетевых вычислительных ресурсов.

Все это снижает быстродействие системы, повышает трудоемкость создания с модификации ИС

4. Трехуровневая клиент-серверная архитектура

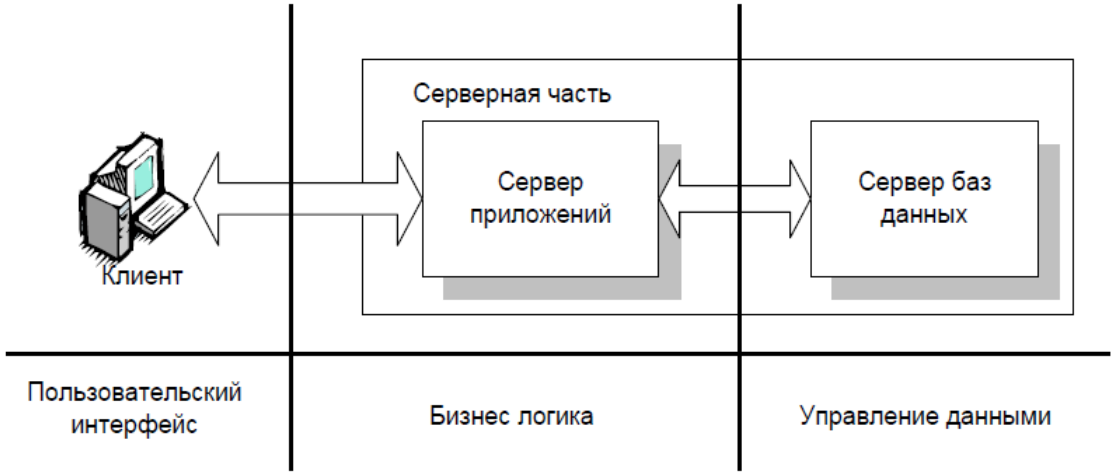

Рисунок 6 Трехуровневая клиент-серверная архитектура

Физическое разделение программ, отвечающих за хранение данных (СУБД) от программ обрабатывающих эти данные

Такое разделение программных компонент позволяет оптимизировать нагрузки как на сетевое, так и на вычислительное оборудование комплекса.

Рисунок 6 Модель сервера приложений

Компоненты трèхзвенной архитектуры, с точки зрения программного обеспечения реализуют определенные сервера БД, web-сервера и браузеры.

Плюсы:

1. Тонкий клиент.

2. Между клиентской программой и сервером приложения передается лишь минимально необходимый поток данных - аргументы вызываемых функций и возвращаемые от них значения. Это теоретический предел эффективности использования линий связи, даже работа с ANSI-терминалами (не говоря уже об использование протокола http) требует большей нагрузки на сеть.

3. Сервер приложения ИС может быть запущен в одном или нескольких

экземплярах на одном или нескольких компьютерах, что позволяет использовать вычислительные мощности организации столь эффективно и безопасно как этого пожелает администратор ИС.

4. Дешевый трафик между сервером приложений и СУБД. Трафик между сервером приложений и СУБД может быть большим, однако это всегда трафик локальной сети, а их пропускная способность достаточно велика и дешева. В крайнем случае, всегда можно запустить СП и СУБД на одной машине, что автоматически сведет сетевой трафик к нулю.

5. Снижение нагрузки на сервер данных по сравнению с 2.5-слойной схемой, а значит и повышение скорости работы системы в целом.

6. Дешевле наращивать функциональность и обновлять ПО.

Минусы:

1. Выше расходы на администрирование и обслуживание серверной части.

5. Internet/Intranet – технологии

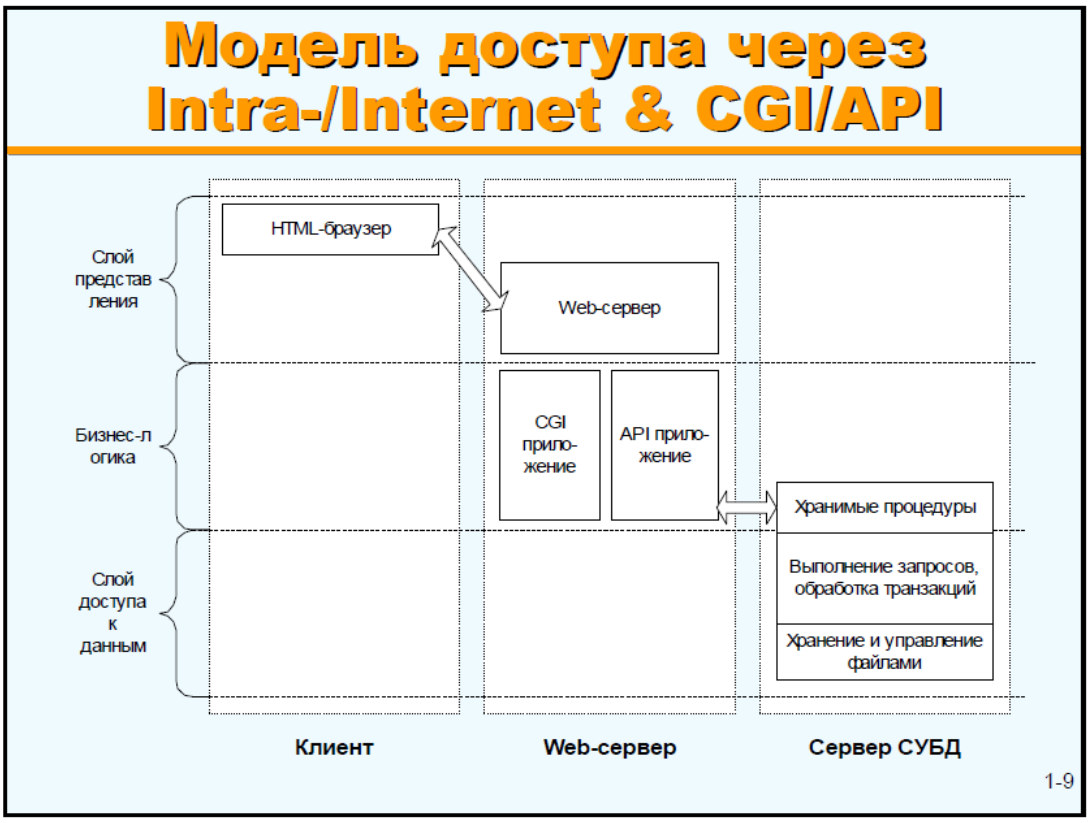

Рисунок 7 Модель Internet/Intranet

6. Архитектура на основе Internet/Intranet с мигрирующими программами

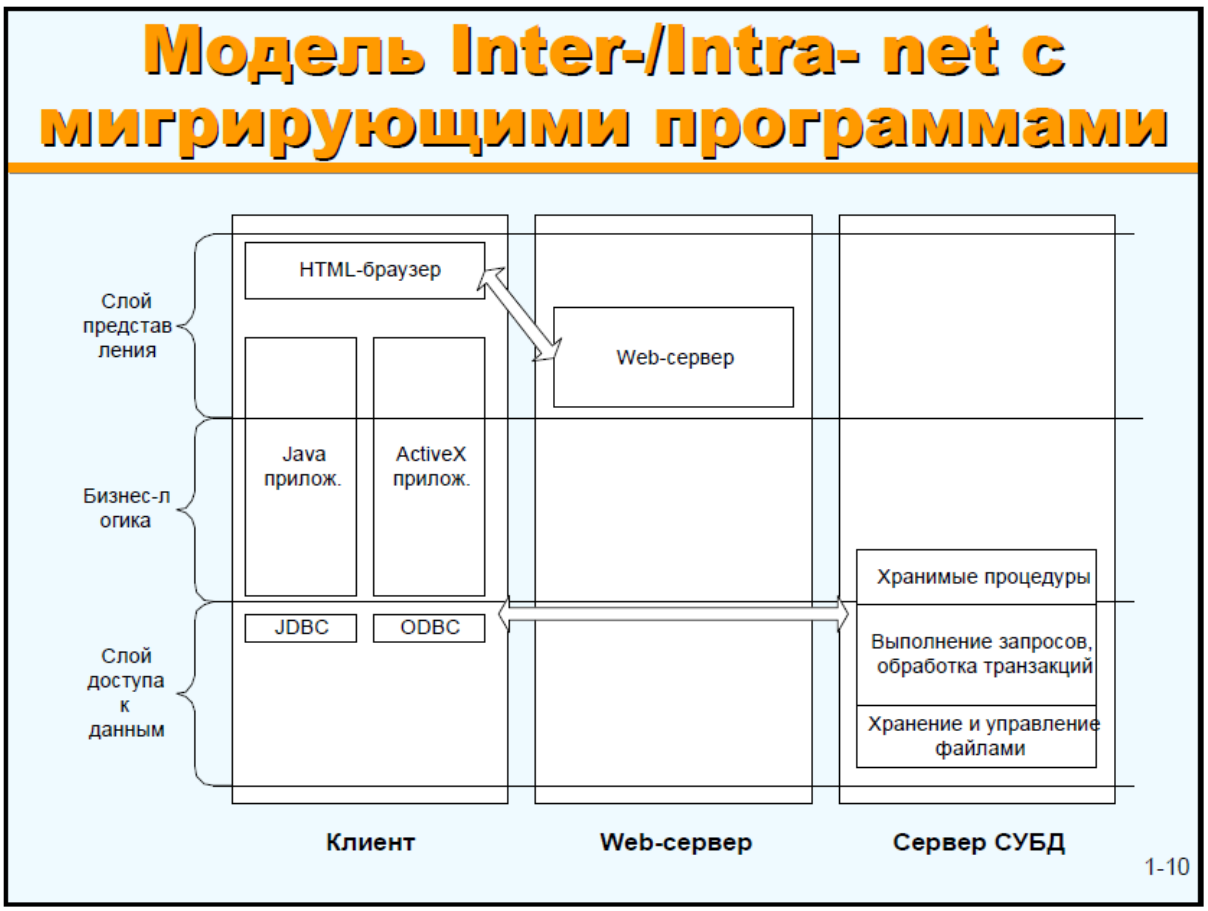

Рисунок 7 Модель Internet/Intranet с мигрирующими программами.

2. Традиционные архитектуры информационных систем

Распределенные информационные системы

Распределенная система — это набор независимых вычислительных машин, представляющийся их пользователям единой объединенной системой.

Рисунок 8 Распределенные информационные системы

Характеристики распределенных систем:

1. От пользователей скрыты различия между компьютерами и способы связи между ними.

2. Пользователи и приложения единообразно работают в распределенных системах, независимо от того, где и когда происходит их взаимодействие.

3.Распределенные системы должны также относительно легко поддаваться расширению, или масштабированию.

Особенности распределенных ИС

• Ссылки

• Задержки выполнения запросов

• Активация/деактивация

• Постоянное хранение

• Параллельное исполнение

• Отказы

• Безопасность

Ссылки

Ссылки на объекты в программных модулях на ОО языках

программирования (например, C++) являются указателями в памяти.

1. Ссылки на объекты в распределенных системах в противоположность являются более комплексными:

1.1.Содержат информацию о размещении

1.2.Информацию о безопасности

1.4.Ссылки на объектные типы

2. Ссылки на распределенные объекты значительно больше (40 байт для Orbix)

Задержки выполнения запросов

Локальные вызовы требуют порядка пары сотен наносекунд

Запрос к объекту требует от 0.1 до 10 миллисекунд

Интерфейсы в распределенной системе должны быть спроектированы так, чтобы снизить время выполнения запросов:

1. Снизить частоту обращения;

2. Укрупнить выполняемые функции.

Активация/Деактивация

Объекты в ОО языках находятся в виртуальной памяти от создания до уничтожения

В распределенных системах

1. Больше объектов

2. Объекты могут не использоваться на протяжении долгого времени

Реализации распределенных объектов

1. Переносятся в память при активации

2. Удаляются из памяти при деактивации

Постоянное хранение

Объекты могут иметь или не иметь состояние.

Объекты имеющие состояние должны сохранять его на постоянный носитель между:

1. Деактивацией объекта

2. Активацией объекта

Может быть достигнуто:

1. Записью в файловую систему

2. Отражением на реляционные БД

3. С помощью объектных БД

Параллельное исполнение

В нераспределенных системах исполнение в основном последовательное, иногда конкурентное в разных нитях процессов. 14

Распределенные компоненты выполняются параллельно, что приводит к необходимости согласования выполнения.

Отказы

Запросы в распределенных системах имеют большую вероятность отказов

Клиенты обязаны проверять факт выполнения запросов сервером

Безопасность

Безопасность в ОО приложениях может выполняться на основе контроля сеансов.

При работе распределенных систем возникают вопросы безопасности:

1. Кто запрашивает выполнение операции?

2. Как мы можем удостовериться, что субъект является именно тем за кого он себя выдает?

3. Как мы примем решение предоставлять или нет субъекту право на выполнение сервиса?

4. Как мы можем неопровержимо доказать, что сервис был предоставлен?