1. Определение распределенной информационной системы, его основные следствия. Участник распределенной системы.

РСОИ - система взаимодействующих независимых АИС (участников данной РСОИ).
Независимые АИС - системы, которые принадлежат и управляются разными юридическими лицам или организациям, преследуют собственные цели.

Следствия:

1. Участники РСОИ зачастую созданы для достижения **разных целей** (прибыль одной организации не влечёт прибыль её партнеров).
2. Каждая АИС имеет **собственную логику работы**, скрытую для других участников.
3. Обмен информацией между участниками осуществляется через **публичные каналы доступа и недоверенную инфраструктуру**.
4. Каждый участник имеет **собственные БД**, прямой доступ к которым закрыт.
5. Невозможно получить достоверное состояние другой системы, поскольку невозможно отдать независимой системе приказ типа «прекрати работу, зафиксируй своё состояние и сообщи его мне; я скажу, когда продолжить». Даже если один участник сообщит другому некоторую достоверную информацию о части своего состояния на некоторый момент времени, за время передачи этой информации состояние уже изменится.
6. АИС могут выдавать **упрощенную или искажённую информацию** другим участникам. Упрощённая: качественные величины, «этого товара много». Искажённая: вместо отказа сообщить об отсутствии ресурса («мест нет! а то вы больно часто снимаете бронь»).

2. Требования к АИС, участвующей в РСОИ. Предложения по её программной структуре.

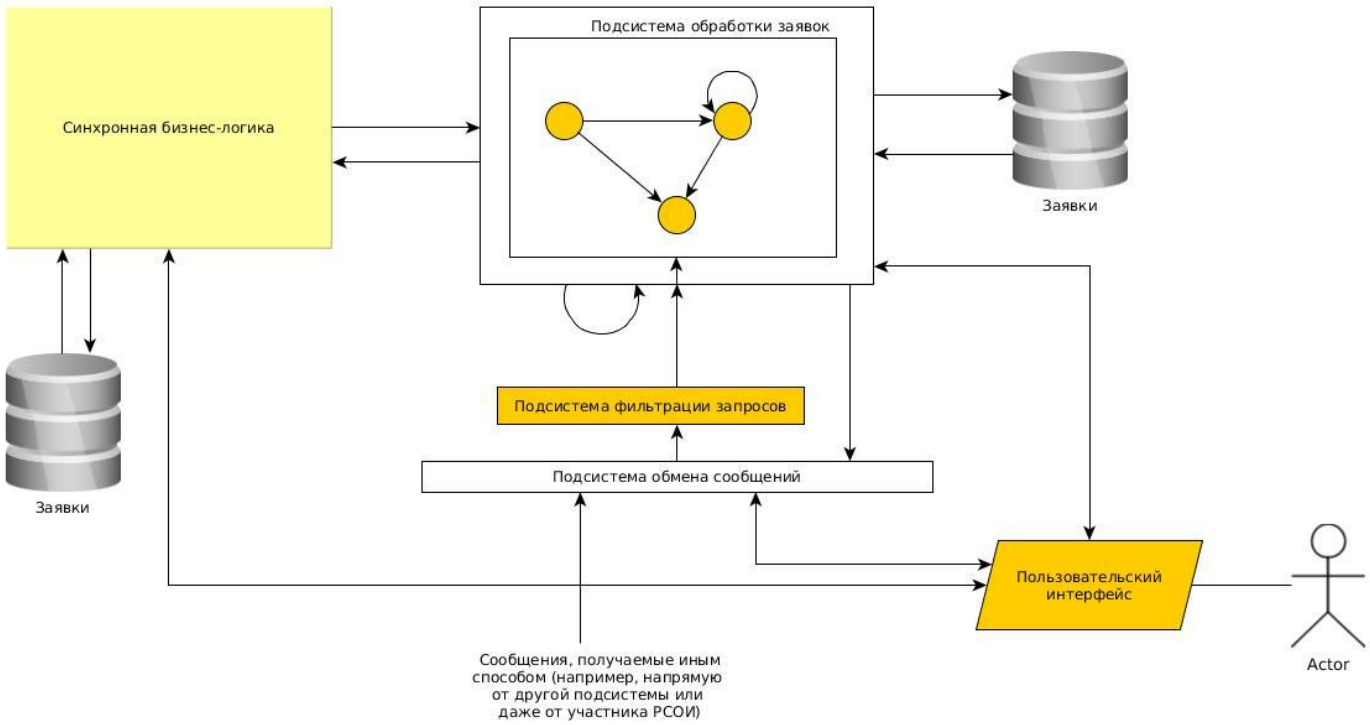
1. **Открытость:** сетевой протокол, включая форматы сообщений и последовательность обмена, должен быть специфицирован и описан. В качестве нижестоящего протокола должны применяться стандартные платформо-независимые протоколы.
2. **Надёжность**: участник РСОИ должен предполагать, что все остальные участники работают плохо:
    a. Корректность работы одного участника не зависит от корректности работы других участников или линий связи. Взаимодействие между двумя участниками РСОИ не должно зависеть от третьего участника.

b Работа участника РСОИ не должна приводить других участников в неожиданное состояние.

3. **Масштабируемость**: участник должен быть способен работать с несколькими одинаковыми системами и с разными системами близкого назначения. Добавление взаимодействия с новой системой должно вызывать минимальные изменения в ПО.
4. **Экономичность:** участник должен разумно тратить ресурсы ЭВМ, которые использует. Пример: глупо создавать по потоку ОС на каждую обрабатываемую заявку; плохо делать транзакции в БД, ожидающие получения сообщения от удалённой системы.
5. **Эффективность** (НЕкомпьютерных ресурсов): нельзя тратить ресурсы так, как будто они бесконечны
6. **Безопасность**: нет полного доверия к каналам связи и к партнёрам. Сокрытие полной информации о состоянии системы. Пароли пользователей.
7. **Синхронизация:** нет синхронизации в чистом виде (когда в некоторый момент времени известно, что две сущности имеют одинаковое представление о чём-либо).

Предложения по структуре:

1. Подсистема **обмена сообщениями**, использующая нижестоящие протоколы.
2. Подсистема **обработки заявок**, ответственная за их жизненный цикл.
3. Подсистема **фильтрации сообщений**, находящаяся между подсистемами обмена и обработки заявок.



3. Составные части описания сетевого протокола. Протокол РСОИ. Сообщения, запросы и ответы, заявки. Идентификация сообщений.

Протокол – пятёрка:

1. Назначение, возможности (что использует и для чего нужен).
2. Формат сообщения – физический вид передаваемого сообщения (типы данных и т.п.).
3. Алфавит – множество (конечное!) типов сообщений. Может быть описан конечным автоматом с выходом.
4. Словарь сообщений – описание всех возможных типов и форматов сообщений; семантика сообщений.
5. Правила обмена сообщениями и алгоритм их обработки

Сообщение – единица обмена с точки зрения протокола.
Запрос – сообщение, предполагающее активность. Создаёт *заявку*.
Заявка – единица работы, выполняемая АИС.
Идентификация сообщений
Наличие у всех сообщений уникальных (в пределах системы-отправителя) идентификаторов сообщений позволяет отбросить дублирующиеся сообщения.

4. Задачи подсистемы обработки заявок. Обработка таймаутов. Обработка неожиданных и противоречивых сообщений.

Подсистема обработки заявок реализует часть протокола участника РСОИ, связанную с жизненным циклом заявки.
Сообщения ей передаются в некотором удобном внутреннем виде.
Разумна организация очереди сообщений и их содержание в постоянном хранилище (БД) с приоритетами. Аналогичная очередь может быть использована для передачи внутренних сообщений к подсистеме обмена сообщениями. Преобразованные во внутреннее представление сообщения передаются и принимаются через *подсистему обмена сообщениями*.
Источником внутренних сообщений, кроме *подсистемы обмена сообщениями* и *подсистемы обработки заявок*, может быть также основная бизнес-логика.

Подсистема обработки заявок:

- обрабатывает внутренние сообщения, извлекая из них заявки (семантический анализ);
- говорит подсистеме обмена сообщениями послать что-то куда-то;
- обращается к подсистеме принятия решений для определения дальнейших действий;
- обращается к подсистеме бизнес-логики для запроса и изменения состояния модели системы;

- обращается к системе статистик для ведения статистики по полученным от других систем сообщениям;
- переходит в другое состояние в соответствии с КА.

Обработка таймаутов

КА должен попадать в одно из конечных состояний за ***конечный*** промежуток времени при ***любой*** входной цепочке внешних событий. Таким образом, при каждой смене состояния должен запускаться (или продолжать работать) ***таймер***, который пришлёт сообщение о тайм-ауте, которое должно быть обработано автоматом.

Успешная отправка сообщения не означает, что сообщение принято к исполнению, поэтому нет оснований утверждать, что мы когда-нибудь получим ответ. Поэтому, в силу требования надежности, использование таймеров необходимо всегда.

Запрос получения состояния данной системы в худшем случае может охватывать и *получение состояния от удалённых систем*. Поскольку нет гарантий, что эти системы работают в настоящий момент, то для реализации такого запроса необходим полноценный КА с тайм-аутами, и реализовать этот запрос в пределах одной сессии TCP/IP в принципе невозможно. Для решения этой проблемы можно использовать и кеширование информации о состоянии удалённых систем, что позволит избежать чрезмерно частого опроса их состояний и «разведёт» процесс опроса состояния удалённых систем и обработку запросов к локальной системе.

Для удобства иногда можно считать, что ответ от транспортной подсистемы вида «не удалось доставить» и событие о завершении интервала ожидания ответа («тайм-аут») также являются сообщениями-ответами на отправляемое сообщение.

Обработка неожиданных и противоречивых сообщений

КА должен реагировать на любое изменение состояния, приходящее к нам от другой системы. Если состояние постоянно меняется – мы не может стабилизироваться.

Пессимистический подход: если получили accepted, то ожидаем, что оно может измениться, если получили rejected – значит оно зареджектилось и более измениться не может.

Буферизация ответов: об изменении кеша лучше всего сообщать, если в течение определенного времени («периода недоверия», или “время hold on-a” :)) еще одного скачка состояния не произошло.

Недостаток: снижение производительности и увеличение времени отклика.

Достоинство: простота реализации.

5. Заявки. Идентификация заявок. Получение идентификатора созданной заявки.

Заявка должна иметь **ID, позволяющие**:

1. Идентифицировать заявку внутри **локальной системы**.
2. Идентифицировать заявку в **удалённой системе** (при получении её состояния).

Способы получения ID:

1. **Один** идентификатор, сообщаемый системе-инициатору заявки до или после создания заявки. После - плохо, т.к. ответ может потеряться.
2. **Два** разных идентификатора:
    a. ID в системе-инициаторе.
    b. В системе, обрабатывающей заявку – пара:
        i. ID системы-инициатора;
        ii. ID заявки в системе-инициаторе.
3. **Плохо**: система, обрабатывающая заявку, полагается на ID, полученный в запросе.

Замечание: Наличие в пришедшем сообщение на создание заявки её ID, назначенного удалённой системой, позволяет отбросить дубликаты таких сообщений.

6. Жизненный цикл заявки. Формализация в виде КА: допущения, выделение входного алфавита и состояний. Ограничения независимой обработки заявок. Создание, изменение и отмена заявки.

Причины создания заявки:

1. Внешний запрос.
2. Внутренний запрос.

Основное состояние заявки принадлежит некоторому конечному множеству.

Минимальное количество состояний – 3:

1. Начальное состояние.
2. Конечные состояния:
    a. Успех.
    b. Провал.

Из конечного состояния заявка не меняет своего состояния, находится в архиве.

Кроме основного состояния, заявка характеризуется **двумя группами параметров**:

- **Параметры запроса:** полученные извне параметры заявки.
- **Параметры обработки заявки**: добавленные в ходе обработки заявки.

Создание протокола, допускающего изменение параметров запроса – сложно. Обычно, извне только создание и отмена заявки. Для всего остального – создание новой заявки.

7. Реализация КА заявки. Хранение состояний заявки. Использование списка будущих событий. Обработка таймаутов после включения.

Формально: **множество состояний КА** – подмножество декартова произведения множества состояний заявки и всех множеств значений параметров обработки заявки. Тогда, например, каждая из N попыток повтора опроса M идентичных систем представлена отдельным состоянием. Это плохо. Поэтому используется сокращённый автомат из основных состояний заявки.

Состояния автомата соответствуют моментам времени, в которые он ожидает получения сообщений.

Функция перехода отображает состояние заявки, параметры заявки и сообщение (со всеми его параметрами) в новое состояние заявки и в новые параметры заявки (например, число неудавшихся попыток увеличилось на единицу).

Выход автомата:

1. Посылка сообщений другим системам.
2. Запуск таймеров.
3. Обращение к бизнес-логике (а также через неё к пользователю).

Два требования к автоматам в РС:

1. При получении **неожиданного сообщения**:
    a. Состояние не меняется.
    b. Добавляет запись в журнал.
    c. Автомат не ругается.
2. **Конечное состояние** должно быть достигнуто **за конечный промежуток времени**. Для страховки **всегда** применяются таймеры.

Внешние события в случае параллельного опроса M однородных систем, когда необходимо получить $R \in [1,M]$ ответов:

1. Получен ответ, число ответов меньше R.
2. Получено R ответов.
3. Тайм-аут.

Состояние заявки нужно хранить в persistent-хранилище (БД) (отказоустойчивость). Действия над заявкой нужно осуществлять транзакционно.

Для выполнения любого действия над заявкой, необходимо знать её текущее состояние.

После завершения действий новое состояние заявки записывается обратно в хранилище. Состояние заявки не может передаваться через ОП для выполнения действий вне одной транзакции.

8. Связь подсистем участника РСОИ на основе очередей внутренних сообщений. Однопоточная и многопоточная обработка сообщений.

Методы передачи сообщений от одной удаленной системы к другой:

1. **Непосредственный обмен сообщениями:** передача происходит напрямую, возможна только в том случае, если принимающая сторона готова принять сообщение в этот же момент времени.
2. **Использование очередей сообщений:** используется посредник – *менеджер очередей сообщений*. Компонента посылает сообщение в одну из очередей менеджера, после чего может продолжить свою работу. Получающая сторона извлечет сообщение из очереди менеджера и приступит к его обработке.

Существует несколько разработок в области промежуточного программного обеспечения, реализующего высокоуровневые сервисы для обмена сообщениями между программными компонентами (Microsoft Message Queuing, IBM MQSeries, Sun Java System Message Queue etc.). Они предоставляют следующие базовые примитивы:

1. добавить сообщение в очередь;
2. взять первое сообщение из очереди, с блокировкой до появления в очереди хотя бы одного сообщения;
3. проверить очередь на наличие сообщений;
4. установить обработчик, вызываемый при появлении сообщений в очереди.

Менеджер очереди сообщений в таких системах может находиться на компьютере, отличном от компьютеров, на которых располагаются участвующие в обмене компоненты. В этом случае сообщение первоначально помещается в исходящую очередь на компьютере с посылающей сообщение компонентой, а затем пересылается менеджеру требуемой. Для создания крупных систем обмена сообщениями может использоваться маршрутизация сообщений: сообщения не передаются напрямую менеджеру, поддерживающему очередь, а проходят через ряд промежуточных менеджеров очередей.

Способы обработки сообщений:

- **Однопоточный:** один рабочий поток по очереди обрабатывает сообщения.

Один рабочий поток выбирает сообщения из очереди и обрабатывает их.
Преимущества: эффективное использование ресурсов (нет простоев).
Недостатки: есть риск того, что скорость обработки заявки будет ниже скорости их поступления в очередь, что повлечет DoS (Denial of Service), переполнение очереди.

- **Многопоточный:** несколько рабочих потоков обрабатывают сообщения.

Зачастую реализуется как “один клиент (участник РСОИ) - один поток”. В чистом виде не используется.
Недостатки: неэффективен с точки зрения использования системных ресурсов (дескрипторов, процессорного времени, памяти и т.д.).

Решение проблемы: “несколько очередей - по рабочему потоку на очередь”.
Преимущества: недостатки модели “поток на клиент” устранены, повышена скорость обработки заявки по сравнению с однопоточной моделью.
Недостатки: проблема добавления сообщений в несколько очередей (по какому признаку и в какую очередь? а не получится ли так, что сообщение, пришедшее позже, будет обработано раньше? и т.д.).

9. Цели участников РСОИ. «Овербукинг» и его причины. Доверие в РСОИ. Проблема синхронизации в РСОИ.

Цели, стоящие перед фирмой, владеющей АИС, (могут отчасти противоречить друг другу):

- увеличение прибыли;
- создание хорошей репутации.

Овербукинг(сверхбронирование) — стратегия сбыта, при которой поставщик принимает на себя больше обязательств по поставке, чем может выполнить, в расчете на то, что не все из взятых обязательств действительно придется выполнять.

Обмен информацией между участниками распределённой системы почти наверняка использует публичные каналы доступа и недоверенную инфраструктуру. Таким образом, у каждого участника **нет** полного **доверия** не только к каналам связи, но и к партнёрам.

Синхронизация в классическом виде, когда в какой-то момент времени достоверно известно, что две сущности имеют одинаковое представление о чём-либо, **невозможна**.
Пытаться синхронизировать хотя бы часть представления о мире между участниками РСОИ нужно для работы системы, хотя в общем случае состояние удалённой системы нельзя зафиксировать на некоторый промежуток времени.
Каждый участник пытается согласовывать часть своих представлений о мире с другими системами, обычно ведя постоянный повторяющийся обмен информации с ними.

Хорошо, чтобы протокол обмена сообщениями был синхронным.
Синхронный – состояние одной стороны не обгоняет состояние другой больше, чем на шаг.
Синхронность (в узком смысле – локальная синхронность) – когда отправляем сообщение в пределах одной заявки, должны получить один ответ от этой системы. То есть, пока не послали следующую заявку, должны получить только один ответ.

10. Взаимодействие подсистемы обработки заявок с основной бизнес-логикой. Связь состояний заявки и локальных транзакций. Многопоточная и однопоточная обработка заявок.

Замечания по бизнес-логике

С точки зрения данного в курсе определения, полноценные глобальные распределённые ACID-транзакции (использующие менеджер транзакций и произвольные ресурсы) в нашей РСОИ невозможны: система не может согласится на участие в транзакции, которая займёт неизвестный промежуток времени, зависящий от работы других систем.

Таким образом, подобные транзакции, охватывающие несколько ресурсов, могут охватывать только ресурсы одной отдельно взятой системы. Более того, любые ACID-транзакции начинаются и заканчивается во время обработки одного сообщения и, вероятно, даже при единственном вызове бизнес-логики. Решение, когда в одном состоянии автомата протокола транзакция начинается, а в другом — завершается, является, вероятно, полностью ошибочным.

11. Подсистема обмена сообщениями и её задачи. Буферизация сообщений.

Можно выделить следующие задачи подсистемы обмена сообщениями:

1. преобразование информации между внутренним представлением и видом, используемым в протоколе обмена информацией между двумя системами;
2. обеспечение безопасности в части шифрования сообщений и обеспечения их целостности и идентификации систем, включая подпись и проверку подписи сообщения.
3. обмен сообщениями между системами и обнаружением ошибок обмена;
4. реализация надежного асинхронного обмена (обычно это требуется, но возлагается на низ), реализация надежного синхронного обмена (но обычно это не требуется);
5. буферизация при обмене информацией.

Под буферизацией здесь понимается группировка нескольких сообщений протокола обмена сообщениями, передаваемых к одной и той же удалённой системе, в одно сообщение нижестоящего протокола. Для использования буферизации удалённая система должна быть способна принять несколько сообщений протокола РСОИ в одном сообщении нижестоящего протокола, будь то запрос HTTP или письмо, передаваемое по SMTP. Кроме того, иногда мы моем передать несколько нижестоящих сообщений в рамках одного сеанса нижестоящего протокола, что тоже даёт эффект (см. SMTP).

На практике буферизация реализуется следующим образом: обнаружив в очереди сообщение, которое необходимо отправить, подсистема обмена сообщениям не бросается сразу выполнять это указание а ждёт некоторое время (зависящее от системы и характера нагрузки, порядка секунд), и если в течении этого времени появляются другие сообщения для этой же удалённой системы, то они отсылаются вместе. Разумно считать время буферизации связанным со временем отклика на уровне IP или даже временем обмена с удалённой системой (если успешные пинги ходят за 200мс, то эта величина и является возможным временем буферизации).
Буферизация позволяет снизить накладные расходы и повысить производительность системы, особенно при частом периодическом опросе состояния удалённой системы («как там наши двадцать заявок?»). Неверное использование буферизации (слишком большое время задержки при малом числе сообщений для одной удалённой системы) может привезти к падению производительности.
Буферизация может работать и в обратном направлении: вместо того, чтобы сразу отправлять сообщение (во внутренней форме) в очередь обработки подсистемой обработки заявок, можно использовать буферизацию в качестве возможного метода борьбы с противоречивыми сообщениями (недостатком этого подхода является до, что для этого при буферизации нужно понимать семантику сообщений).

12. Запросы получения и изменения состояния удалённой системы, их особенности.

На практике при обработке запросов получения собственного состояния работа подсистемы протокола сводится к одной функции и не имеет состояний. Забегая вперёд, отметим, что такой запрос можно реализовать, например, на основе запроса-ответа HTTP в пределах одного сеанса TCP/IP (используя XML-RPC или HTTP/REST и т. д.).
Однако, запрос получения состояния в худшем случае может охватывать и получение состояния от удалённых систем. Поскольку нет гарантий, что эти системы работают в настоящий момент, то для реализации такого запроса необходим полноценный конечный автомат с тайм-аутами, и реализовать этот запрос в пределах одной сессии TCP/IP в принципе невозможно. Для решения этой проблемы можно использовать и кеширование информации о состоянии удалённых систем, что позволит избежать чрезмерно частого опроса их состояний и «разведёт» процесс опроса состояния удалённых систем и обработку запросов к локальной системе.

13. Кеширование состояния удалённой системы.

Кеширование в РС заключается в хранении полученного состояния удалённой системы для дальнейшего использования. Получение состояние при этом часто разумно организовать заблаговременно, получая состояние системы с некоторым интервалом и выполняя каждый раз несколько попыток в случае неудачи.

О когерентности такого кеша в РС речь не идёт. Между получением состояния удалённой системы и попыток его изменить (т. е. захватить какие-то ресурсы) состояние удалённой системы всегда может поменяться произвольным образом, поскольку по данному определению РС в ней нет нет ни критических секций, ни блокировок. Поэтому отсутствие у кеша состояния удалённой системы свойства когерентности, видимо, не является нашей самой большой проблемой.

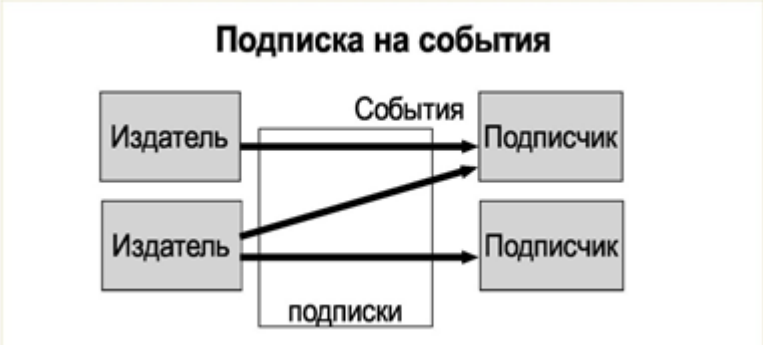
Закешированное состояние удалённой системы может использоваться пессимистически, как оценка сверху («час назад у ней не было нужного ресурса») или оптимистически: за оценку сверху берётся последнее количество ресурсов, увеличенное, к примеру, на 10 процентов. Следует также помнить, что удалённая система может приближённо сообщать о наличии ресурса («нет», «ждём», «есть») и имеют свою систему принятия решений, и только при обработке заявки станет точно известно, удастся ли нам его получить.

Основная проблема при кешировании – во ряде случаев состояние системы может быть слишком велико для того, чтобы быть переданным за разумное время. Причиной этому может быть слишком большое описание состояние, неэффективный формат передачи, медленные каналы связи, регулярная быстрая смена этого состояния (и высокие требования к «разумности» времени). В этом случае возможно, в теории, использовать репликацию, однако использование репликации (на уровне прикладного протокола) в РС находится под определённым вопросом и её, вероятно, разумно применять только к справочникам (например, ко всему каталогу продукции без указания наличия оной)

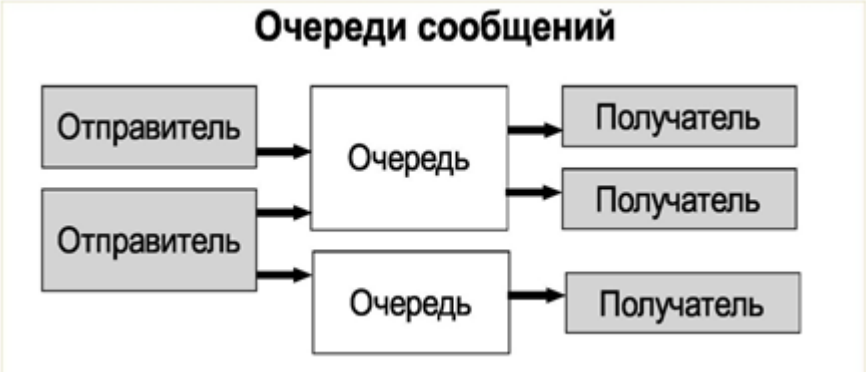
14. Метафоры информационного обмена: доставка сообщения, рассылка, запрос-ответ. Область применения в РСОИ. Принцип определения активного участника в РСОИ.

1. Клиент и сервер выполняются одновременно;
2. Клиент знает местонахождение сервера и его внешний интерфейс;
3. Клиент и сервер непосредственно взаимодействуют с использованием транспортного протокола (TCP или HTTP);
4. Ответ предполагается в ближайшем будущем (в течении одного соединения TCP/IP);

5. Обычно используется для получения информации от сервера;
6. Возможен вариант без ответа (односторонний запрос);



1. Взаимодействие слабо связанных подписчиков и издателей;
2. Используется вспомогательная сущность (система подписки);
3. События доставляются без хранения в очереди;
4. Издатели и подписчики должны выполняться одновременно;
5. Подходит для информирования удаленных систем (подписчиков) о своем состоянии;



1. Каждое сообщение приходит только к одному получателю;
2. Ожидать сообщения в очереди могут сколько угодно получателей;
3. При отсутствии получателей, сообщение хранится в очереди;
4. Может быть организована надежная доставка;
5. Ответы так же передаются через очереди;
6. Подходит, в частности, для передачи запросов об изменении состояния удаленных систем;

15. Спецификации корректных сообщений протокола РСОИ. Недостатки строго парсера сообщений.

В кратце - описать формат сообщения РСОИ; сказать, почему невозможно полностью определить корректные сообщения (или можно? суть в том, что кобинаций параметров очень много);

что такое “строгий парсер” - хоть убейте, не вспомню. Очень похоже на то, что у нас его вообще не было.

Строгий парсинг -- задание верхней границ информативности сообщений. Нижняя граница -- минимум информации(необходимые поля), верхняя -- отсутствие излишних.
Проблемы строгого парсинга в создании сильной связности систем. Т.е., например, мы планируем добавить поле *C* и говорим другим, что скоро оно станет обязательным. При нестрогом парсинге клиенты могут его добавить и не париться, при строгом же переключение должно быть синхронным.
Строгий парсинг бесполезен и не нужен, перекладывание с больной головы на здоровую.

16. Синхронные протоколы обмена информацией. Область применения в РСОИ, ограничения. Модель удалённого вызова и модель REST.

После отбрасывания всех платформо-зависимых RPC остались следующие кандидаты на роль синхронного (т.е. запроса-ответа протокола РСОИ, укладывающегося в один сеанс нижестоящего протокола) взаимодействия:
— прямое использование HTTP для передачи текстовых документов;
— использование HTTP с дополнительными соглашениями о методах HTTP (например, HTTP/REST);
— протоколы, основанные на создании поверх HTTP аналога удалённого вызова (SOAP, XML-RPC, JSON-RPC).

Представители всех этих видов рассмотрены в моих предыдущих опусах, где можно с ними и ознакомиться. Здесь же отметим следующие особенности. Концепция HTTP/ REST, возможно, и интересна как простой интерфейс для получение иерархических данных, но использование методов HTTP для указания операций с ресурсами (удаление, редактирование) с точки зрения РСОИ скорее неправильно. Вместо прямого редактирования в РС следует явно посылать сообщение о необходимости редактирования ресурса или заявки.
Концепция прозрачного удалённого вызова достаточно спорна с точки зрения РС. В частности, сообщение о невозможности установить соединение — типичный результат удалённого вызова. Кроме того, рекомендованный для РС интерфейс удалённого вызова — единственный удалённый вызов для каждой категории партнёров, принимающий и получающий документ на языке размётки.
Спецификация SOAP включает формат спецификации внешнего интерфейса (на языке WSDL), которая может использоваться для автоматической генерации кода вызова на стороне клиента. Хотя этот метод и позволяет легко создать простые удалённые вызовы веб-сервиса даже неквалифицированному программисту, его премущества для РСОИ с передаваемыми в

виде сложных текстовых документов сообщений относительно невелики, поскольку XSD схема этих документов в WSDL не включается (если я не ошибаюсь; проверьте).

17. Асинхронные протоколы обмена информацией. Область применения в РСОИ, ограничения. Почтовая система и её развёртывание в РСОИ.

Механизм передачи сообщений означает, прежде всего, саму инфраструктуру, поддерживающую надёжную передачу сообщений по маршруту, и, во-вторых, протоколы общения с сервером отправки сообщений и сервером получения сообщений. Инфраструктура эта может начинаться на localhost виде локального SMTP-сервера (ведь до localhost мы сможем достучаться почти всегда).

Поскольку все системы, реализующие стандарт Java Message System, использовать не рекомендуется как платформено-зависимые, то из живых систем выбор в настоящий момент ограничен двумя возможностями:

— почтовой системой с протоколами SMTP и POP3;

— системой обмена сообщениями jabber с протоколом XMPP.

Хотя обмен системы были рассчитаны скорее для посылки сообщений от человека к человеку, ничто не мешает их использовать в качестве надёжного транспорта в РС. Недостатком использования POP3 является необходимость регулярно опрашивать POP3-сервер, но при необходимости это можно исправить, например использовав в конце маршрута передачи сообщения свой собственный модифицированный SMTP-сервер, непосредственно связанный с транспортной подсистемой.

Для отправки сообщения в локальной сети (или даже на локальной машине) требуется развернуть SMTP-smarthost, который и возьмёт на себя функции повтора попыток отправки на следующий SMTP-сервер (находящийся, возможно, на посреднике или на удалённой системе).

При использовании системы передачи сообщений внешний интерфейс системы с технологической точки зрения определяется её адресом и самим форматом сообщения на языке разметки.

Готовый механизм надёжной отправки рекомендуется использовать при посылке сообщений, изменяющих состояние удалённой системы, чтобы не реализовывать вручную повторы запросов по XML-RPC или подобному протоколу.

18. Обеспечение безопасного информационного обмена. Безопасность на уровне сетевого протокола, нижестоящего протокола, протокола РСОИ.

В роли нижестоящего протокола для базового сетевого взаимодействия в РС в теории может использоваться и напрямую TCP или даже UDP. Однако обычно практично

использовать работающий поверх TCP протокол, который позволяет передать произвольный текст с указанием его кодировки. Таких протоколов у нас уже есть несколько. К данным протоколам относятся следующие:
— протокол HTTP и основанные на нём протоколы высокоуровневого удалённого вызова (SOAP, XML-RPC);
— протоколы почтовой системы SMTP, POP3 и IMAP (последний нам явно не нужен);
— протокол XMPP, используемый в Jabber.

В силу установленных требований к РС следующие протоколы не могут рассматривается как претенденты на нижестоящий протокол РС:
— что все платформенно-зависимые протоколы высокоуровневого удалённого вызова (Java RMI, Remoting.NET, PyRo и др.);
— ОС-зависимые протоколы (MSMQ, «низкоуровневый» классический RPC);
— кросс-платформенные, но слишком сложные протоколы, имеющие единственную референсную реализацию, такие как Zeroc Ice, так же сложно рекомендовать к использованию в качестве транспорта РС.

19. Шифрование и его использование в РСОИ. Безопасный нижестоящий протокол на основе SSL. Шифрование сообщений протокола РСОИ.

Может показаться, что необходимость в шифровании и электронной подписи сообщений зависит от того, используется ли безопасный транспортный протокол (например, HTTPS или HTTP поверх VPN) или небезопасный (например, HTTP поверх публичной сети). Безопасным транспортом может быть сам нижестоящий протокол или один из протоколов под ним.
Однако с точки выявленных требований к РС можно сформулировать следующее положение: каждая система должна, в идеале, использовать ассиметричное шифрование и электронную подпись своих сообщений, поскольку этим достигается максимальная протяжённость безопасного пути для сообщения, вместе с безопасным нижестоящим протоколом.
Таким образом, безопасность на уровне сообщений не исключает требование использования безопасного транспортного протокола. Например, при использовании SMTP или HTTP мы можем использовать авторизацию по паролю (хотя курс склоняется к идентификации по сертификату открытого ключа, но конспект пока старый), что требует использования SSL для безопасной передачи пароля. Возможно, что мы не хотим показывать третьим лицам и адрес электронной почты получателя и другую служебную информацию.

Таким образом, в РС лучше всего использовать как безопасный протокол, защищённый от прослушки и проблемы «шпиона посредине» (читай — HTTPS или SMTP+SSL), так и шифрование на уровне сообщений, которые затем будут передаваться по безопасному нижестоящему протоколу.
Отметим, что наш нижестоящий протокол может как совпадать с HTTPS (HTTPS/REST), так и работать поверх него (XML-RPC, SOAP).

20. Идентификация участников РСОИ. Использование паролей и открытых ключей. Использование цифровой подписи.

Для безопасного обмена информацией необходимо:

1. защищать информацию от изменения третьими лицами
2. защищать информацию от чтения третьими лицами

Эти функции могут выполнятся на уровне используемого транспортного протокола (примеры: HTTPS, SMTPZSSL, VPN), в этом случае говорят о transport layer security
Эти же функции так же могу выполняться и на протоколе обмена в распределенной системе (пример: различные расширения SOAP), в этом случае говорят о message security
В теории безопаснее реализовывать второй подход, поскольку сообщение при этом чуть дольше остается защищенным.
Однако первый подход позволяет использовать стандартные протоколы, поэтому он предпочтительнее с академической точки зрения.
Безопасность на транспортном уровне так же называют point-to-point security, в отличие от end-to-end security.
Для обеспечения надежной передачи информации используется цифровая подпись и шифрование.
При этом в ходе передачи с использованием Secure Socket Layer (протоколы SSL и TLS) используется временный ключ и симметричное шифрование (RC4, DES и др.), а для пересылки этого ключа используется шифрование с открытым ключом (RSA, DSA и др.).
Алгоритм применяет цифровые сертификаты открытых ключей (X.509) и хеш-функции (SHA, MD5).
SSL является стандартом, используемым для реализации безопасных транспортов (HTTPS, SMTPZSSL, POP3ZSSL, OpenVPN)
SSL может также реализовывать идентификацию клиента и сервера на основе проверки их сертификатов (используется, например, в OpenVPN). Однако это не должно подменять механизм идентификации клиентов распределенной системы.

21. Распределённые транзакции. Двухфазная фиксация. Использование в РСОИ.

Транзакция – последовательность операций с какими-либо данными, которая либо успешно выполняется полностью, либо не выполняется вообще. В случае невозможности успешно выполнить все действия происходит возврат к первоначальным значениям всех измененных в течение транзакции данных (откат транзакции). Транзакция должна обладать следующими качествами.

· Атомарность. Транзакция выполняется по принципу "все или ничего".

·Согласованность. После успешного завершения или отката транзакции все данные находятся в согласованном состоянии, их логическая целостность не нарушена.

·Изоляция. Для объектов вне транзакции не видны промежуточные состояния, которые могут принимать изменяемые в транзакции данные. С точки зрения "внешних" объектов, до успешного завершения транзакции они должны иметь то же состояние, в котором находились до ее начала.

·Постоянство. В случае успешности транзакции сделанные изменения должны иметь постоянный характер (т.е. сохранены в энергонезависимой памяти).

Транзакции являются основой приложений, работающих с базами данных, однако в распределенной системе может быть недостаточно использования только транзакций систем управления базами данных. Например, в распределенной системе в транзакции может участвовать несколько распределенных компонент, работающих с несколькими независимыми базами данных.

Распределенной называется транзакция, охватывающая операции нескольких взаимодействующих компонент распределенной системы. Каждая из этих компонент может работать с какими-либо СУБД или иными службами, например, использовать очереди сообщений, или даже работать с файлами. При откате транзакции все эти операции должны быть отменены. Для этого необходимо выполнение двух условий:

1. промежуточная среда должна поддерживать управление распределенными между несколькими компонентами транзакциями;
2. компоненты распределенной системы не должны работать с какими-либо службами или ресурсами, которые не могут участвовать в транзакции.

Распределенные транзакции являются важнейшим элементом поддержания целостности данных в распределенной системе. Поэтому для более широкого их применения промежуточная среда может содержать механизмы, которые при необходимости (и определенных затратах времени на написание кода) позволят использовать в распределенных транзакциях внешние службы, не поддерживающие транзакции. Такой

механизм называется компенсирующим менеджером ресурса. Компенсация в данном случае означает возврат ресурса к первоначальному состоянию при откате транзакции.

22. Подсистема фильтрации сообщений. Обнаружение ошибочных сообщений и нежелательных заявок.

Подсистема фильтрации реализует часть политики фирмы по обработке заказов. В качестве своей главной своей задачи она пытается определить, можно ли взяться за исполнение заказа, учитывая:

1. известные сведения о заказчике;
2. известное состояние самой системы и предположительное состояние систем-партнёров.

В нашей работе мы будет использовать простейшую подсистему «принятия решений» в виде логики предикатов первого порядка, поскольку в этом курсе для нас важно понимать её наличие.

Для описания правил принятия решения должен использоваться (в идеале) специализированный язык, достаточно понятный специалисту по бизнес-процессу. В его качестве на практических занятиях мы будем использовать пролог, а описывать правила мы будет в форме дизъюнктов Хорна.

Логично требовать, что принятие решений системой фильтрации должно быть достаточно быстрым, хотя и не обязательно почти моментальным: принятие решений в основном используется при обработке запросов на изменения состояния, которые обычно носят асинхронный характер. При обработке идемпотентных (и обычно синхронных) запросов на получение состояния логика принятия решения об обслуживании сообщения также может использоваться («а не слишком ли многое хочет знать о нас другая система?»).

Подсистема фильтрации решение принимает на основе:

1. параметров заявки («кажется, здесь лишний ноль»);
2. состояния модели (например, базы данных с таблицами остатков товаров, поставками и платежами данного клиента);
3. дополнительной статистике об истории «отношений» с обслуживаемой системой;
4. представлений о состоянии смежников: можно ли получить недостающие ресурсы у третьих фирм;
5. самих правил и значений констант в программке принятия решений (в нашем случае это будет простейшее сравнение пороговыми значениями).

Результатом работы подсистемы принятия решений может быть одно следующего списка:

1. немедленно начать выполнять заявку;

2. отклонить заявку;
3. запросить подтверждение человека о выполнении заявки;
4. подождать с выполнением некоторое время, по истечении которого «подумать» ещё раз;
5. не выполнять, но ответить что приняли к исполнению;
6. понизить приоритет заявки или сообщения;
7. и так далее.

23. Тестирование подсистемы обработки заявок.

Для тестирования протокола АИС её системы-смежники заменяются заглушками, реализующих полностью случайное, предположительное (случайное) или некоторое детерминированное (например, наихудшее) поведение. Тестирование реализации протокола РС «в лоб» может быть достаточно бессмысленно в силу наличия часовых и более таймаутов, поэтому тестируется «масштабированный» по величине тайм-аутов вариант.

Кроме различных видов тестирования, для исследования протокола может применяться имитационное и (в ряде случаев) аналитическое моделирование, как на основе создания отдельно модели (например, в simpy), так и на основе использования самого кода протокола или его подмножества.

Тестирование и имитационное моделирование протокола, к сожалению, не может гарантировать обнаружения всех возможных проблем. Поскольку количество состояний самого протокола в каждой из ИС конечно, а от счётного множества состояний основной БД системы можно отчасти абстрагироваться, то обычно можно создать модель, позволяющую полностью перебрать все состояния РС для обработки одной первоначальной заявки, и произвести её верификацию путём рассмотрения всего графа состояний (см. SPIN/Promela). У этого подхода есть два недостатка, небольшой: в реальности множество состояний может превосходить наши вычислительные возможности, и крупный: если верификация модели не выявила проблем, это не значит, что их нет у самой реализации (обратное проверяется).

24. Внешний интерфейс участника РСОИ. Отделение интерфейса от логики работы. Связь концепции MVC и структуры участника РСОИ.

В теории нам всё равно, как и на чём реализован интерфейс пользователя в РСОИ, лишь бы он не был монолитно связан с протоколом обмена сообщениями или с бизнес-логикой, а взаимодействовал с последней через чётко определенный интерфейс (например, на основе веб-сервисов).

Моделью в нашем случае выступает не компонент доступа к базе данных (в примерах к MVC-каркасам в этой роли выступает ORM), а интерфейс доступа к бизнес-логике (которая, вместе со своею БД, и есть модель). Бизнес-логика, в свою очередь, может обращаться к логике работы протокола.

С точки зрения данного в курсе определения, полноценные глобальные распределённые ACID-транзакции (использующие менеджер транзакций и произвольные ресурсы) в нашей РСОИ невозможны: система не может согласится на участие в транзакции, которая займёт неизвестный промежуток времени, зависящий от работы других систем.

Таким образом, подобные транзакции, охватывающие несколько ресурсов, могут охватывать только ресурсы одной отдельно взятой системы. Более того, любые ACID-транзакции начинаются и заканчивается во время обработки одного сообщения и, вероятно, даже при единственном вызове бизнес-логики. Решение, когда в одном состоянии автомата протокола транзакция начинается, а в другом — завершается, является, вероятно, полностью ошибочным.

25. Выбор лидера и его применение в РСОИ. Приоритеты участников. Основной и резервный лидеры, проверка живости. Метод полного опроса.

Многие распределенные алгоритмы требуют, чтобы один из процессов выполнял функции лидера, инициатора или некоторую другую специальную роль. Выбор такого специального процесса будем называть выбором координатора. При этом очень часто бывает не важно, какой именно процесс будет выбран. Можно считать, что обычно выбирается процесс с самым большим уникальным номером. Могут применяться разные алгоритмы, имеющие одну цель - если процедура выборов началась, то она должна закончиться согласием всех процессов относительно нового координатора.

26. Выбор лидера на основе алгоритма задиры. Достоинства и недостатки по сравнению с полным опросом.

Алгоритм задиры:

- нужны выборы! посылаются сообщения только процессам с большим идентификатором, чем у рассылающего. По истечению таймера:
  a. ответов нет: сообщает всем, что данная система - лидер.
  b. ответы есть: получив ответ ждём подтверждения, ждём сообщения о том, кто - лидер. Если приславший не подтвердил, то проводятся перевыборы.
- ответ (подтверждение) на выборы.
- обозначение лидера - посылается системой, если она - новый лидер (нет систем с большим ид).

27. Анализ предметной области. Использование нотаций IDEF0, ER, прецедентов.

IDEF0 — Function Modeling — методология функционального моделирования и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временна́я последовательность (WorkFlow).

Стандарт IDEF0 представляет организацию как набор модулей, здесь существует правило — наиболее важная функция находится в верхнем левом углу, кроме того есть правило стороны : — стрелка входа приходит всегда в левую кромку активности, — стрелка управления — в верхнюю кромку, — стрелка механизма — нижняя кромка, — стрелка выхода — правая кромка.

Описание выглядит как «чёрный ящик» с входами, выходами, управлением и механизмом, который постепенно детализируется до необходимого уровня. Также для того чтобы быть правильно понятым, существуют словари описания активностей и стрелок. В этих словарях можно дать описания того, какой смысл вы вкладываете в данную активность либо стрелку.

Также отображаются все сигналы управления, которые на DFD (Диаграмме Потоков Данных) не отображались. Данная модель используется при организации бизнес-проектов и проектов, основанных на моделировании всех процессов: как административных, так и организационных.

28. Проектирование РСОИ. Использование нотаций КА, коммуникаций, последовательностей, ER.

Модель сущность-связь (ER-модель) (англ. entity-relationship model, ERM) — модель данных, позволяющая описывать концептуальные схемы предметной области.

ER-модель используется при высокоуровневом (концептуальном) проектировании баз данных. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями.

Во время проектирования баз данных происходит преобразование ER-модели в конкретную схему базы данных на основе выбранной модели данных (реляционной, объектной, сетевой или др.).

ER-модель представляет собой формальную конструкцию, которая сама по себе не предписывает никаких графических средств её визуализации. В качестве стандартной графической нотации, с помощью которой можно визуализировать ER-модель, была предложена диаграмма сущность-связь (ER-диаграмма) (англ. entity-relationship diagram,

ERD).